НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія ”

на тему: Сервіс для створення та керування відкладеними платежами за покупки в інтернеті

Сервіс для створення та керування відкладеними платежами за покупки в інтернеті

Виконав : студент __4__ курсу, групи ІВ-81
(шифр групи)

_____Базова Лідія Геннадіївна_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____доц., к.т.н., доц. Русанова О.В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ст. вик. Сімоненко А. В. ___ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“ Комп’ютерні системи та мережі”

спеціальності 123 “ Комп’ютерна інженерія ”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2022 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Базової Лідії Геннадіївни

1. Тема проєкту *Сервіс для створення та керування відкладеними платежами за покупки в інтернеті*
керівник проєкту_____доц., к.т.н., доц. Русанова О.В. А. В. ___
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від

2. Термін здачі студентом закінченого проєкту *11 червня 2022 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз відомих сервісів, що працюють на ринку та постановка задачі.

Розділ 2. Огляд інструментів для створення сервісу.

Розділ 3. Опис структури сервіса та його компонентів.

Розділ 4. Демонстрація роботи сервісу та інструкція з його використання.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) схема функціонування сервісу (принципова схема), структурна схема, функціональна схема серверної частини, текст програмного коду мобільного додатку, текст програмного коду серверної частини, текст програмного коду сервісної веб-частини.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко А. В.		

7. Дата видачі завдання «30» серпня 2022 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2020-15.03.2022*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2022-25.03.2022*	
4.	*Розробка структур окремих підсистем*	*25.03.2022-5.04.2022*	
5.	*Програмна реалізація системи*	*5.04.2022-15.04.2022*	
6.	*Оформлення пояснювальної записки*	*15.04.2022-20.05.2022*	
7.	*Захист програмного продукту*	*25.04.2022*	
8.	*Передзахист*	*23.05.2022*	
9.	*Захист*	*21.06.2022*	

Студент-дипломник _____ Лідія БАЗОВА
(підпис)

Керівник проєкту _____ Ольга РУСАНОВА
(підпис)

АНОТАЦІЯ

В ході виконання данної роботи буде розглянуто основні принципи роботи сервісів для створення та керування відкладеними платежами за покупки в інтернеті. Буде проаналізовано їх слабкі та сильні сторони і на основі цього аналізу буде розроблено подібний сервіс, який найкращим чином підходить для функціонування в наших умовах. Сервіс буде мати веб та мобільну частину. В вебчастині буде функціонал для клієнтів та людей, що представляють інтернет-магазини. В мобільній версії буде представлено зручний функціонал для клієнтів. Данний сервіс буде зручним для використання для людей з різних країнах та з різними інтернет-магазинами.

Для виконання данного сервісу буде використана мова TypeScript, Dart, фреймворк Flutter для мобільного додатка та бізі данних PostgreSQL.

Ключові слова: Flutter, Dart, TypeScript, мобільний та веб сервіс, фінансові технології.

ANNOTATION

In this project the basic principles of services for creating and managing deferred payments for online purchases will be considered. Their strengths and weaknesses will be analyzed and based on this analysis, a similar service will be developed that is best suited to operate with our requirements. The service will have a web and a mobile part. The web section will have functionality for customers and people representing online stores. The mobile version will feature user-friendly functionality. This service will be convenient for people from different countries and with different online stores.

This service will use TypeScript, Dart, Flutter framework for mobile application and PostgreSQL database.

Keywords: Flutter, Dart, TypeScript, mobile and web-service, financial technologies.

Справки	Форма	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Сервіс для створення та керування	4		
			відкладеними платежами за покупки			
			в інтернеті			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Сервіс для створення та керування	74		
			відкладеними платежами за покупки			
			в інтернеті			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Сервіс для створення та керування	1		
			відкладеними платежами за покупки			
			в інтернеті			
			Схема функціонування сервісу			
	А4	***ІАЛЦ.4672008.005 Д2***	Сервіс для створення та керування	1		
			відкладеними платежами за покупки			
			в інтернеті			
			Структурна схема			
	А4	***ІАЛЦ.467200.006 Д3***	Сервіс для створення та керування	1		
			відкладеними платежами за покупки			
			в інтернеті			
			Функціональна схема серверної			
			частини			
	А4	***ІАЛЦ.467200.007 Д4***	Сервіс для створення та керування	46		
			відкладеними платежами за покупки			
			в інтернеті			
			Текст програмного коду			
			мобільного додатку			
	А4	***ІАЛЦ.467200.008 Д5***	Сервіс для створення та керування	37		
			відкладеними платежами за покупки			
			в інтернеті			
			Текст програмного коду			
			серверної частини			
	А4	***ІАЛЦ.467200.009 Д6***	Сервіс для створення та керування	37		
			відкладеними платежами за покупки			
			в інтернеті			
			Текст програмного коду			
			серверної частини			

					ІАЛЦ.467200.002 ТЗ
		№ докум.	Підпис	Дата	
Розробив	Базова Л. Г.			**Сервіс для створення та керування відкладеними платежами за покупки в інтернеті Технічне завдання**	Літ. / Аркуш 1 / Аркушів 4
Перевірив	Русанова О. В.				
					НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-81
Н. Контр.	Сімоненко В. П.				
Затвердив					

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Сервіс для створення та керування відкладеними платежами за покупки в інтернеті*»

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Це технічне завдання розроблено для планування розробки сервісу для оплати покупок частинами товарів в різних магазинах. Такі сервіси входять в категорію сервісів типу Buy Now Pay Later.

Користувачами цього сервісу є люди, які хочуть мати легкі та зручні умови для швидкої позики на оплату товарів інтернет-магазинів. Також проєктом будуть кристуватись магазини-мерчанти, що будуть надавати можливість оплати частинами товарів в їх магазині для заволікання більшої кількості клієнтів.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка сервісу для відкладених платежів, що буде надавати необхідний функціонал для зручного керування платежами за покупки в інтернеті та планування їх здійснення, а також керуванням майбутніх покупок з використанням сервісу.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проєкту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблений сервіс має виконувати такі вимоги:

- Простий і інтуїтивно-зрозумілий інтерфейс сервісу.
- Можливість реєстрації та авторизації користувача в системі.
- Доступ до частини сервісу для представників магазинів з браузеру
- Доступ до клієнтської частини сервісу з браузеру та з мобільного додатку.
- Реєстрація окремо для представників інтернет-магазинів (через браузер) та для клієнтів (через браузер та на мобільній платформі)
- Функції для мерчантів надання клієнтам можливості відкладеної оплати товарів магазину та планування цього процесу
- Функції для мерчантів перегляду інформації щодо покупок в їх магазинах
- Функції для користувача здійснення відкладеної оплати за товар з магазину-мерчанту за допомогою сервісу та її планування
- Надсилання нотифікацій користувачу щодо оплати товарів
- Надавати можливість вносити дані щодо профілю та акаунту для користувача
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- Комп’ютер з доступом до інтернету та встановленим браузер.
- Телефон на операційній системі Android 4.1 та вище, або iOS 9.0 та вище.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2021-15.12.2022
Вивчення та аналіз завдання	15.12.2021-15.03.2022
Розробка архітектури та загальної структури системи	15.03.2022-25.03.2022
Розробка структур окремих частин системи	25.03.2022-5.04.2022
Програмна реалізація системи	5.04.2022-15.04.2022
Виправлення помилок	24.05.2022-11.06.2022
Оформлення пояснювальної записки	11.06.2022

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Сервіс для створення та керування відкладеними платежами за покупки в інтернеті*»

Київ – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

BNPL	(Buy Now Pay Later) Тип короткострокового фінансування, що дозволяє споживачам робити покупки та оплачувати їх пізніше
IDE	(Integrated development environment) програмне забезпечення, яке надає комплексні засоби для розробки програмного забезпечення.
СУБД	Система керування базами даних

ВСТУП

Концепція «Купуй зараз плати пізніше» технології швидко стають глобальним явищем, очолюваним інноваторами та вченими. З розвитком інтернету, ці технології розповсюджуються і в інтернет-середовищі. Такі технології належать до категорії фінансових. Фінансові технології — це загальний термін для інноваційних фінансових послуг з підтримкою технологій та бізнес-моделей, які супроводжують ці послуги[2].

Створення та запровадження на нашій території інтернет-сервісу, що є представником фінансових технологій, є досить актуальною темою, зважаючи на те, що наразі подібних популярних функціонуючих сервесів для нашої території не існує. Через те, що все більша кількість людей надають перевагу для покупки товарів та бюрократичних справ інтернет-сервісам, очевидно, що подібні сервіси будуть набувати більшої популярності. Тому сервіс є потенційно перспективним.

Також процес створення подібних сервісів надасть навичок, що є дуже актуальними для світу, в якому інтернет та інтернет-технології набувають популярності.

Актуальності проєкту додає те, що крупномаштабна війна є загрозою фінансового кризису в усьому світі та через це у людей можуть виникнути фінансові труднощі в найближчий час. Тому для них може бути необхідним використання подібного сервісу. Звичайно, у такій ситуації треба відповідально ставитись до цього задля уникнення ще гірших наслідків. Це надає актуальності і для іншої категорії клієнтів – інтернет-магазинів, що будуть мерчантами та партнерами сервісу. Для них сервіс є можливістю залучити більшу кількість покупців.

РОЗДІЛ 1.
АНАЛІЗ ВІДОМИХ СЕРВІСІВ, ЩО ПРАЦЮЮТЬ НА РИНКУ

1.1 Опис сервісів типу Buy Now Pay Later. Ідея та історія виникнення подібних сервісів

Функціонал, що описується як But Now Pay Later (скорочено BNPL) характеризує можливість оплати куплених товарів або послуг через деякий час після моменти купівлі[1]. Кількість та розмір платежів варується для різних товарів та в залежності від бажання та обставин сторін, що утворюють угоду.

Зазвичай розділяють як мінімум дві сторони в цих відносинах: мерчанта, що надає товар та погоджується отримати повну плату за нього через деякий час, та покупця, що отримує товар та має обов'язок сплати за нього за визначений проміжок часу[2]. Іноді в таких угодах існує ще третя сторона, що може або сама виплатити всю суму, а потім брати гроші з клієнта, або третя сторона бере на себе відповідальність проконтролювати виплату запрошеної суми. Для цих функцій вона забезпечує зручне спілкування з клієнтом. У цьому є її сервіс як члена угоди. Доречним для неї буде забезпечення зручної складання угоди (як для мерчанта, так і для клієнта) та забезпечення швидкої та максимально легкої для клієнта подальшої оплати. Наприклад, легка прив'язка рахунку та перечислення коштів на нього. Або система сповіщень, що допоможуть не пропускати строки виплат. У роботі буде зроблено аналіз саме цих характеристик для виявлення найзручніших умов для успішного сервісу типу BNPL.

Угоди BNPL легко поплутати з угодами на оплату товару в кредит. Головна відмінність між ними полягає в тому, що оплата товару в кредит передбачає виплату процентів третій стороні. По суті, кредит є позикою коштів у третьої сторони. У зв'язку з цим клієнт (за данними з різних ресурсів) повинен виплатити в середньому від 3 до 10 процентів комісії третій стороні[2]. Якщо брати до уваги ці факти, можемо зробити висновок, що система BNPL є більш вигідною для користувачів.

З розвитком комунікаційних можливостей та, зокрема, інтернет-зв'язку, спосіб розрахунку BNPL став можливим онлайн. Наразі досить багато інтернет-магазинів надають можливість оплати частинами або в кредит товарів, що представлені у цьому магазині. Але, зазвичай, такі можливості доступні за умови попередньої реєстрації в якомусь конкретному банку та наявності рахунку в цьому банку. Не важко здогадатись, чому інтернет-магазини зобов'язують подібну реєстрацію від покупців. Таким чином інтернет-магазин упевнюється в тому, що покупець не вирішить раптом забути про своє позичення коштів. У банку є його адреса, а також паспортні данні. Отже, у разі шахрайства з боку покупця, магазин (або банк) мають усі ресурси для подання до суду заяви про шахрайство, а також постраждала сторона зможе відсудити свої кошти.

Описаний алгоритм діє вже досить давно. Потенційним користувачем цієї моделі є будь-яка людина, що бажає отримати в магазині товар або послугу, але за якихось причин не хоче оплачувати її в повному розмірі в день купівлі. Отже, описана схема користування є досить актуальною та загальновживаною в реаліях сучасного суспільства. Але ніщо не буває ідеальним. Навіть цю систему можна зробити більш досконалою та зручною. До її мінусів відноситься те, що для оплати частинами магазини мають договір з одним, або максимально двома, банками. Таким чином, для користування BNPL-функцією, покупець повинен мати відкритий незаблокований рахунок в одному з цих банків. На відкриття рахунку знадобиться невизначена кількість часу. Адже згідно з умовами більшості банків, для відкриття рахунку людині потрібно прийти у відділення банку та зареєструватися там. Потім очікувати процесу завершення реєстрації та, видачі картки. Через описані труднощі набувають популярності онлайн-сервіси.

Зокрема, до таких сервісів відноситься австралійська компанія Afterpay, сервіс Klarna, що зосереджені на договорах покупців з інтернет-магазинами. Цифрова версія дозволяє завантажувати додаток та оплачувати товари через нього. Таким чином, такий додаток є способом оплати (замість кредитної картки чи онлайн-банкінгу)[1]. З точки зору зручності та гнучкості це дуже добре. Адже

з появою таких сервісів у покупця більше немає проблем з тим, щоб мати картку конкретного банку для конкретної купівлі. І у мерчантів є переваги, адже вони розміщені на сервісі, з яким співпрацюють. Тобто, сервіс здійснює додаткове рекламу для мерчантів.

Але і в такої системи є недоліки. Зокрема, це те, що замовник зазвичай повинен сплатити відсотки за запізнення платежу[1]. Тому для сервірів важливо зосередити увагу на зручності використання та системи повідомлень, що дозволить користувачеві використовувати додаток найкращим чином. Розглянемо більш детально роботу сервісів AfterPay та Klarna.

1.2 Опис та особливості сервісу AfterPay

Afterpay – це компанія, що надає можливість покупцям, що можуть приймати участь у роздрібнених товарообмінах, отримати короткострокову фінансову допомогу [3]. Це доступно без перевірки кредитоспроможності та без комісій, коли вони платять вчасно. Послуга доступна у тисячах роздрібних продавців у всьому світі [5] та покупців з Великобританії, Австралії, Канади, Нової Зеландії та США

Коли користувач робить покупки за допомогою Afterpay, сума, яку він повинен сплатити, поділяється на чотири платежі. Він робить початковий авансовий внесок за покупку, після чого у нього є шеститижневий період, щоб здійснити решту платежів [6]. Afterpay не вказує мінімального необхідного платежу для купівлі. Однак на нього можуть поширюватися вимоги щодо покупки, встановлені мерчантом, у якого він робить замовлення [7]. Відгуки користувачів сервісу свідчать про те, що ця опція є досить зручною. Однак, її можна вдосконалити, надавши клієнтам обирати суму внеску та час для оплати з декількох представлених варіантів.

Щоб мати можливість користуватися сервісом онлайн, покупець повинен вибирати Afterpay як спосіб оплати під час оформлення замовлення. (рис. 1.1) Ця функція доступна у магазинах, що є зареєстрованими мерчантами AfterPay. Таких, поки що, не дуже багато.

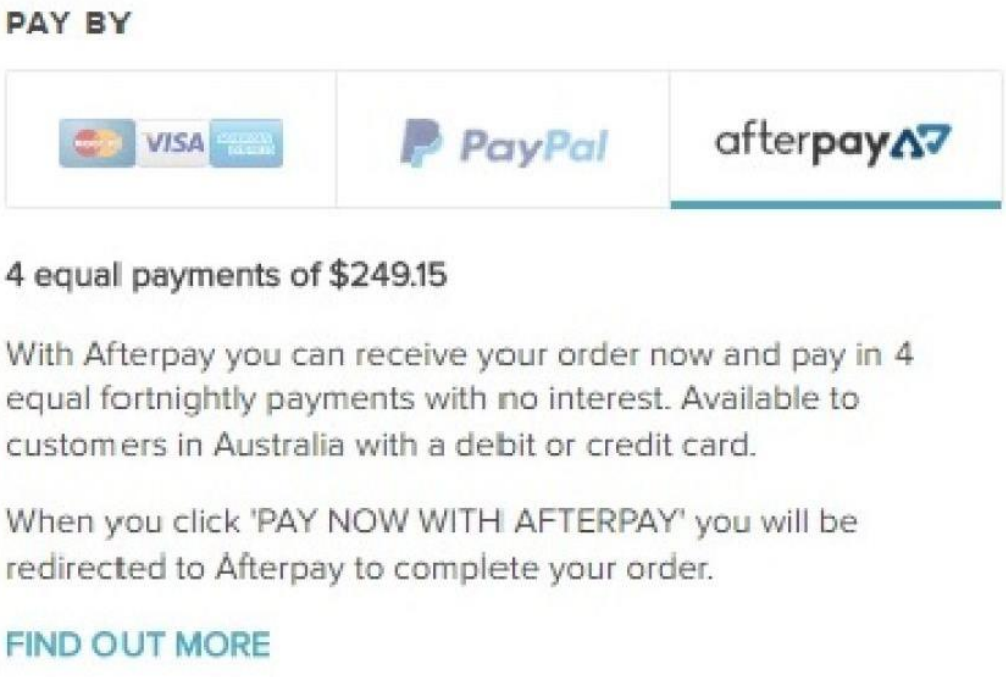


Рисунок – 1.1 Інтерфейс вибору Afterpay як спосіб оплати за товар в інтернет-магазині

Також є можливість швидко створити обліковий запис Afterpay, щоб здійснити перший платіж. Це реалізовано за допомоги двох-факторної автентифікації за допомогою email адреси та номеру телефону. Користувач вводить адресу своєї електронної пошти. Потім йому приходить лист з посиланням на підтвердження реєстрації або входу в акаунт. Перейшовши за цим посиланням, він перенаправляється одразу до застосунку Afterpay. Якщо він переходить за посиланням в браузері, він заходить на сайт Afterpay при переході за посиланням.

Щоб використовувати Afterpay у звичайних магазинах (які не є зареєстрованими мерчантами в сервісі), користувачу необхідно після завантаження програми Afterpay та створення облікового запису, відкрити вкладку «Картка» у програмі та пройти процес створення віртуальної картки. (рис. 1.2) Це дозволяє створити цифрову картку післяплати, яку можна додати в Apple Pay або Google Pay. [5] Цей процес не займає багато часу та не потребує паспортних даних користувача.

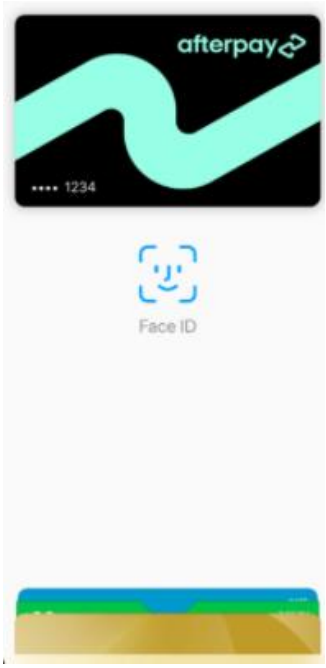


Рисунок – 1.2 Віртуальна картка від сервісу Afterpay

Після здійснення покупки, користувачу доступна інформація про те, коли і скільки слід сплатити наступного разу.

Afterpay не стягує відсотки за покупку, але стягує комісію у разі прострочення дати платежу. У різних странах діє різне обмеження для компанії сервісу на таке стягування. Натомість для користувачів немає ніяких штрафів або комісій за передоплату та дострокового погашення позики [8].

Щодо обмежень на суму позики, сервіс використовує гнучкі ліміти витрат, пристосовані до кожного клієнта та кожної покупки. Є декілька факторів, що враховуються під час встановлення та змінення ліміту позики [9]. Зокрема:

- Загальний час реєстрації на сервісі та час конкретного використання послугами сервісу
- Середня сума початкового платежу, що зазвичай сплачується та регулярність її сплати
- Середня вартість товарів, що купуються користувачем
- Кількість замовлень, на яких вже надано позики

На відміну від інших сервісів BNPL, Afterpay не виконує перевірки кредитоспроможності, під час подачі заявки на фінансування. Тож ніякого кредитного рейтингу не враховується. Це є плюсом програмного продукту. Але,

з іншого боку, це може бути небезпечно для користувачів бо менше контролює населення в загальному плані. Адже існують люди, яким краще не давати гроші в силу їх фінансової нестабільності і вони цього не усвідомлюють.

Afterpay має лаконічний та стильний дизайн як у веб-додатку, так і в мобільній програмі (рис. 1.3). Дизайн – це те, на що звертають увагу більшість користувачів. Тому, це може бути одна з причин, чому сервіс є популярним.

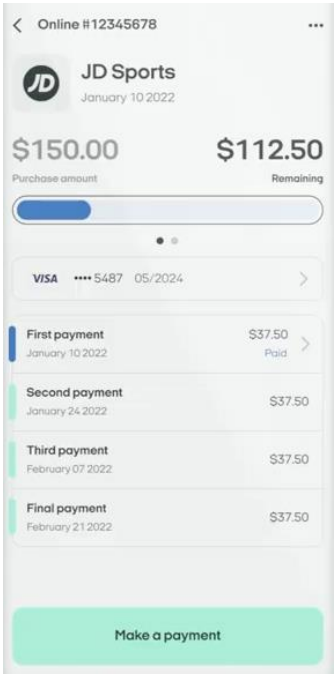


Рисунок – 1.3 Дизайн сторінки транзакцій в мобільній версії Afterpay

Мерчанти можуть зареєструватися на сервісі як мерчант і тодц сервіс буде для них додатковим рекламним майданчиком. Таким чином, мерчанти рекламуватимуть свої колекції та товари на домені сервісу (рис. 1.4). А користувачі матимуть змогу відкривати для себе нові інтернет-магазини.

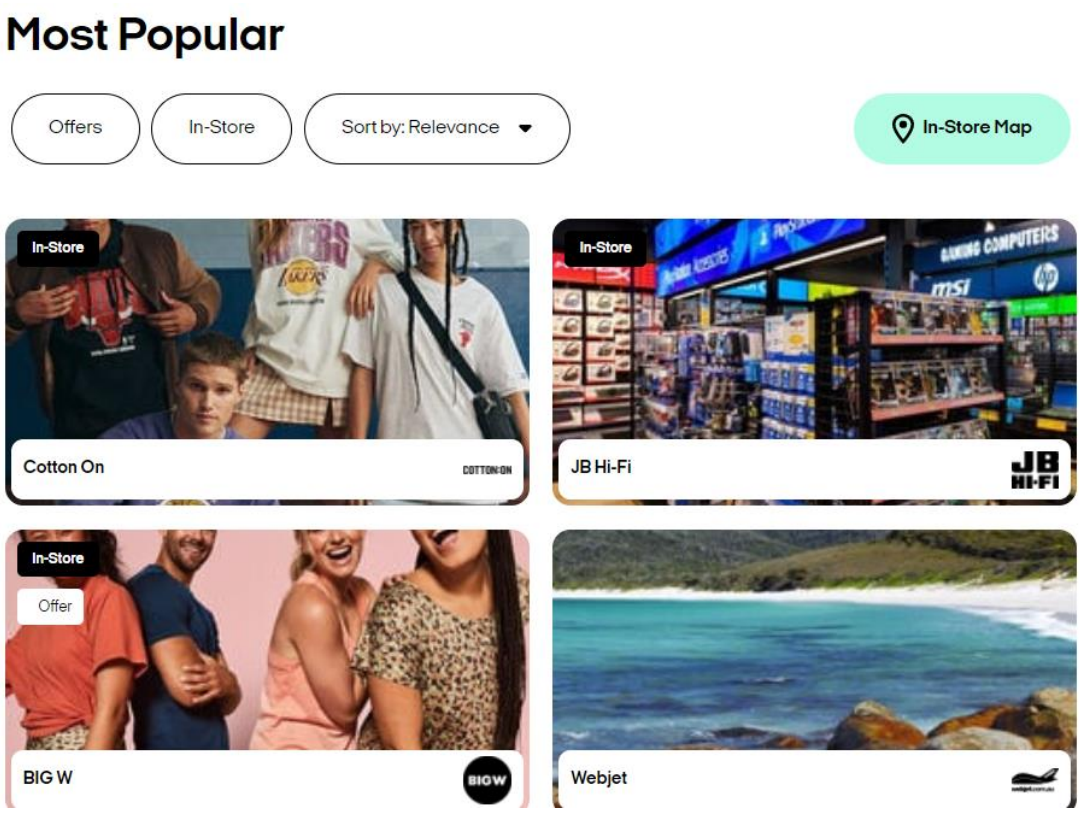


Рисунок 1.4 – Реклама мерчантів на веб-версії сервісу Afterpay

А для користувачів мобільної частини сервісу окрім рекламної сторінки з товарами з магазинів, мерчантів буде показано на карті регіону користувача. (рис. 1.5) По цій карті користувач може швидше зрозуміти, скільки триватиме доставка товару з магазину. Зазвичай, покупці, що купують товари за допомогою подібних сервісів прагнуть отримати своє замовлення якомога швидше. Тому це є досить цікава ідея, яку можна реалізувати і для дипломного програмного продукту. Через те, що сервіс працює без перевірки даних користувача та орієнтований на максимально швидке користування, серед BNPL сервісів цей було визнано найкращим для студентів [5].

З недоліків цього сервісу можна виділити те, що сервіс може відмінити замовлення клієнта [11].

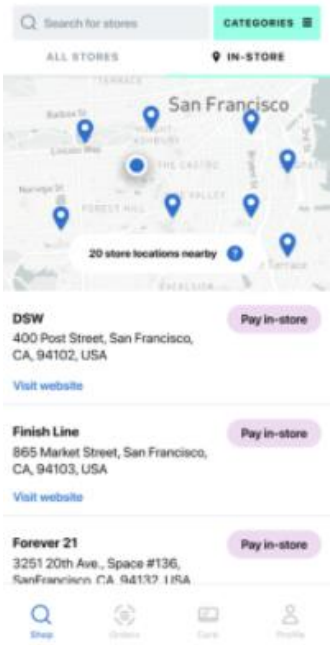


Рисунок 1.5 – Карта з магазинами-мерчантами в мобільній версії додатка Afterpay

Також суттєвим недоліком є те, що сервіс не є доступним для користування на нашій території. Сервіс існує з 2014 року, і про запровадження його в наші країни не було ніяких новин. Тому, є сенс створити йому альтернативу для нашого ринку.

1.3 Опис та особливості сервісу Klarna

Klarna — це ще одна BNPL компанія, яка дозволяє онлайн-покупцям купувати у великих роздрібних продавців без передоплати. Споживачі можуть оплачувати свої покупки чотирма відсотковими платіжками кожні два тижні або сплатити всю суму протягом 30 днів. Вони також можуть фінансувати свою покупку протягом строку від 6 до 36 місяців [10].

Компанія є досить великою. Klarna була заснована у 2005 році. Вона надає платіжні рішення для 205 000 мерчантів у 17 країнах та має 85 мільйонів клієнтів [12]. Партнерами Klarna є такі великі бренди, як ASOS, H&M, Dolce & Gabbana, Michael Kors, Ticketmaster, Sephora, Toms, Timberland, Lenovo, Ambercrombie & Fitch, Dyson, Sonos, Expedia, Air France та Bose [13]. Тому для при купівлі у досить великої кількості мерчантів доступна кнопка «оплатити за допомогою

сервісу Klarna” (рис. 1.6).Категорії роздрібної торгівлі включають такі категорії, як автомобілі, краса, мода та електроніка[14].

Рисунок 1.6 – Способи оплати товару в магазині-мерчанті Klarna

На цьому сервісі не потрібна реєстрація. Споживач тільки повинен надати інформацію про свою кредитну або дебетову картку, щоб Klarna здійснила м’яку перевірку кредитоспроможності.

Маркетологи компанії Klarna звертаються до представників інтернет-магазинів, що мають з тим, що покупці не здійснюють покупки після додавання товару в кошик. Це свідчить про те, що вони не готові віддати кошти на купівлю цих товарів. А ще покупці часто кидають свої кошики, тому, що не хочуть мати клопоту зі створенням облікового запису, або процес оплати є занадто складним. Тому співпраця з сервісом BNPL буде вигідною для таких інтернет-магазинів. Відсоток відмови від кошика в середньому становить близько 70% [15]. Klarna допомагає зменшити цей відсоток на третину. Це є досить гарною концепцією для розповсюдження сервісу, що також можна запровадити з готовим програмним продуктом.

Сервіс бере на себе фінансовий ризик, надаючи можливість покупцям зробити замовлення без оплати початкового внеску. Тобто Klarna повністю оплачує товар, а потім будує графік платежів для покупця. Фінансування для цієї операції надається банком WebBank. Комісія за прострочення оплати товару становить від 7 до 35,10 доларів США [16].

Зважаючи на те, що Klarna не стягує відсотки чи комісію з клієнтів за стандартні варіанти оплати, постає наступне питання. Як ця компанія заробляє гроші? Згідно з інформацією на сайті компанії, сервіс стя гує комісію за транзакцію з мерчантів. Комісія може бути від 3,29% до 5,99%, аде не менша 0.30

доларів США [17]. Klarna вважає, що роздрібні торговці готові платити ці збори, оскільки її послуги допомагають збільшити продажі. Згідно з оцінками, здатність споживачів платити в розстрочку підвищує середню вірогідність замовлення на 45% [18].

Власні дослідження Klarna показали, що коли споживачі мають можливість платити в розстрочку або з від сервісу типу BNPL, споживачі витрачають більше, ніж без фінансування [18]. Це викликає занепокоєння соціальних організацій та установ, що слідкують за виплатою податків. Адже молоді покупці можуть взяти на себе забагато боргів. Настільки багато, що не зможуть з ними впоратися.

У відповідь на ці занепокоєння Klarna повідомила новосній компанії The Guardians, про виконання фінансових запобіжних заходів, щоб запобігти надмірним витратам. Вона прикладає зусилля для того, щоб клієнти не могли робити необмежену кількість покупок. В сервісі були встановлені пороги для того, щоб клієнти здійснили платежі за вже зроблені покупки, перш ніж вони зможуть здійснити наступні транзакції [19]. Це є досить гарною та суспільно-корисною практикою, на яку теж потрібно звернути увагу в процесі подальшого розвитку програмного продукту у.

Функціонал Klarna на веб сайті та на телефоні надає користувачу майже однакові можливості. Користувач може переглянути чибагато інформації (рис. 1.7) Інформація розділена на наступних сторінках:

- Сторінка з інформацією про банк та баланс на своїх картках, збережених в сервісі.
- Сторінка з товарами від мерчантів. На цій сторінці товари можна зберегти для швидкого доступу або перейти за прямими посиланнями в інтернет-магазин, що пропонує ці товари.

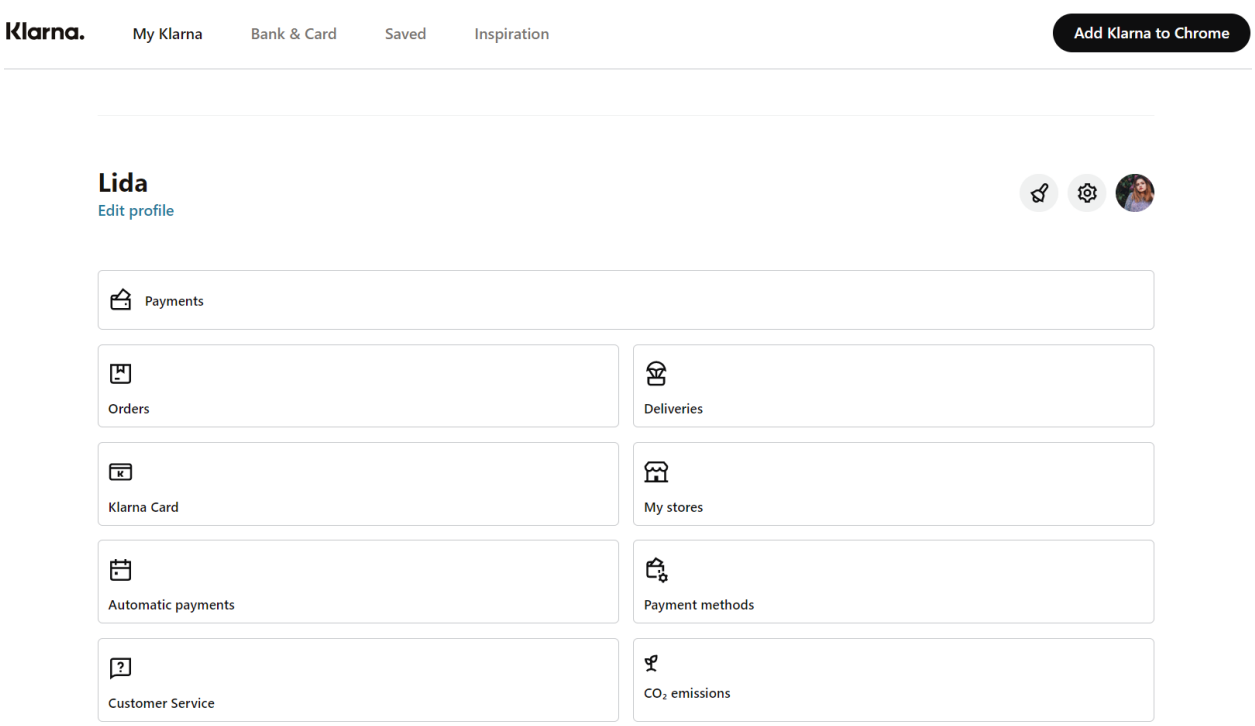
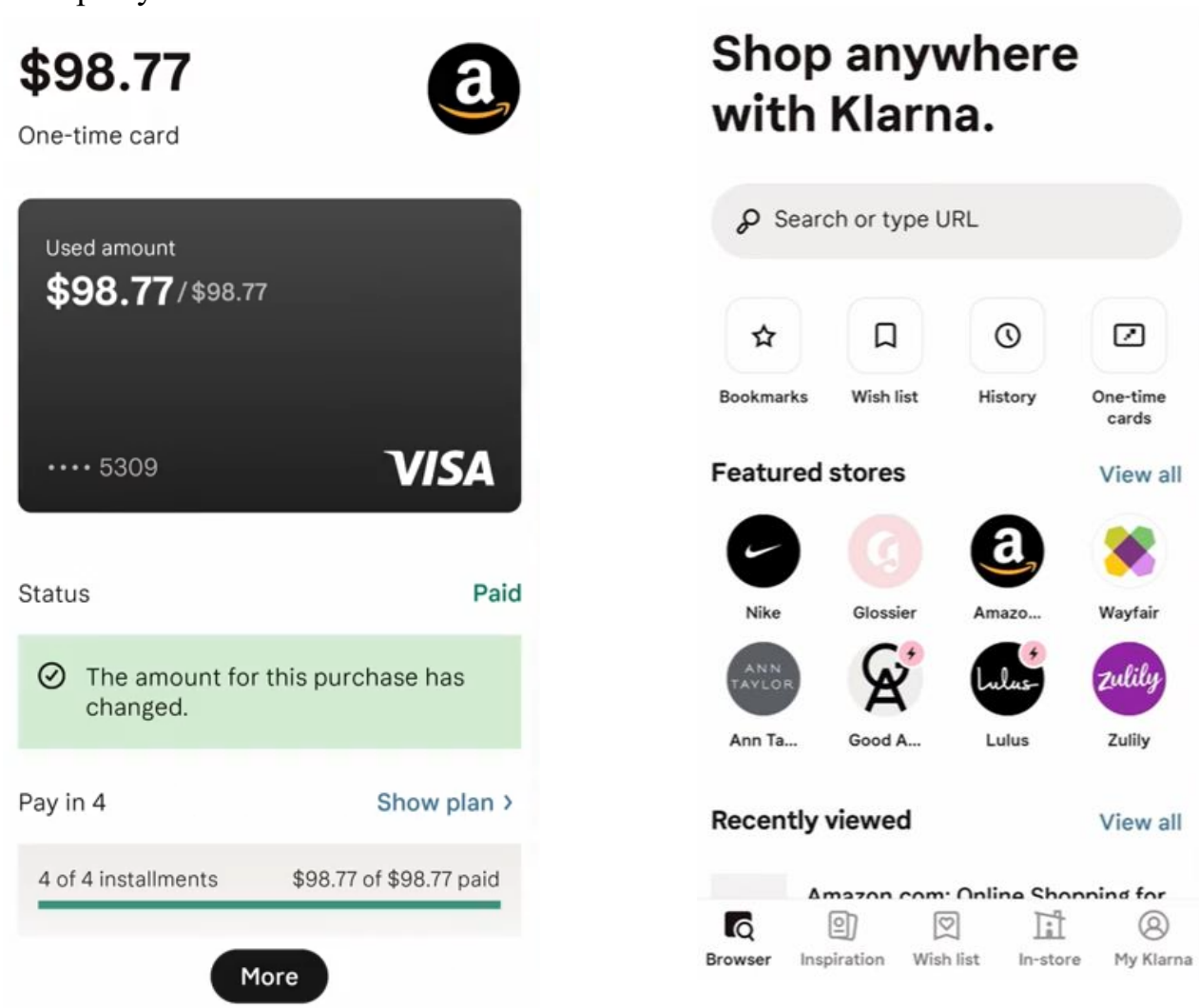


Рисунок 1.7 –Головне меню вервісу Klarna

- Сторінка з товарами, які користувач зберіг для швидкого доступу.
- Сторінка з інформацією про транзакції користувача.
- Інформація про наступні платежі, що заплановані для користувача.
- Інформація про доставку замовлених товарів
- Сторінка, на якій пропонується створити картку
- Сторінка з магазинами, в яких вже було здійснено замовлення з використанням сервісу.
- Сторінка, де користувач може керувати своїми регулярними платежами, які використовують Klarna як спосіб оплати. Під цю категорію підходить Netflix, Disney plus та подібні ресурси, що вимагають підписки.
- Сторінка, що показує способи оплати, які користувач приєднав до сервісу. Тут можна додавати банківські рахунки та банківські картки.
- Сторінка, на якій представлен сервіс Klarna. Там користувач може знайти відповіді на типові питання та зв’язатись з представниками сервісу для особистих запитань.

- Інформація про те, скільки CO2 викидів зробили покупки користувача, що він зробив за допомогою сервісу. Це досить цікава особливість, яка додає бали довіри до сервісу.

Як видно зі скріншоту, веб-версія Klarna, як і Afterpay, має досить лаконічний дизайн. Як видно на скріншоті (рис. 1.8), це стосується і мобільної версії сервісу.



Рисунки 1.8 – Дизайн мобільної версії сервісу Klarna

Серед BNPL сервісів цей було визнано найкращим для великих замовлень.

1.4 Опис та особливості сервісу Affirm

Affirm – ще одна компанія, що надає BNPL сервіс. Функціонал цього сервісу схожий до вже описаних сервісів Klarna та Afterpay. У нього теж є веб та мобільна версія для користувачів і він теж співпрацює з мерчантами та запрошує клієнтів до магазинів-партнерів.

Сервіс компанії діє лише у трьох країнах. США, Австралії та Канаді. Але в цих країнах він користується великою популярністю, отже, є сенс аналізу цього сервісу для запровадження прогресивних ідей, що будуть виявлені у висновках до аналізу.

До його особливостей також належить те, що цей сервіс не має загально встановленого мінімального чи максимального кредитного ліміту як такого. Натомість у сервісі існує верхня межа покупок у 17 500 доларів США [23]. Таким чином, клієнт може зробити декілька покупок від різних магазинів на суму, що не перевищує встановленого для нього максимального кредитного ліміту. Індивідуальний кредитний ліміт залежить від декількох факторів [24]:

- кредитна історія клієнта
- історія платежів за допомогою Affirm
- Скільки часу у клієнта є обліковий запис у Affirm та наскільки він активно його використовує
- Процентна ставка, запропонована продавцем, до якого клієнт подає заявку

Affirm також зазначає, при визначенні кредитного ліміту враховуються поточні економічні умови, що існують в країні користувача. Таким чином компанія оберігає себе та своїх клієнтів від впливу можливої економічної кризи.

На відміну від описаних сервісів, цей не стягує з клієнтів плату за запізнення платежу. Однак, в угоді, яку приймає користувач дрібним шрифтом вказано, що компанія не дає гарантії фінансування з нульовою процентною ставкою. Таким чином відсоткова ставка позики може досягати значення 10 та іноді навіть 30. Але особливість сервісу в тому, що він знімає фіксовані з моменту підписання угоди проценти з клієнта та відсотки ніколи не сумуються [26]. До того ж, для деяких купівель є обов’язковим початковий платіж [25]. Це свідчить про непрозорість системи та загальні труднощі для користувача.

Як і описані раніше сервіси, Affirm має мінімалістичний та лаконічний інтерфейс як для мобільного додатка (рис 1.9, 1.10), так і для веб-версії. Це

дозволяє користувачам швидко знайти те, що їм потрібно та покращує враження від використання сервісу.

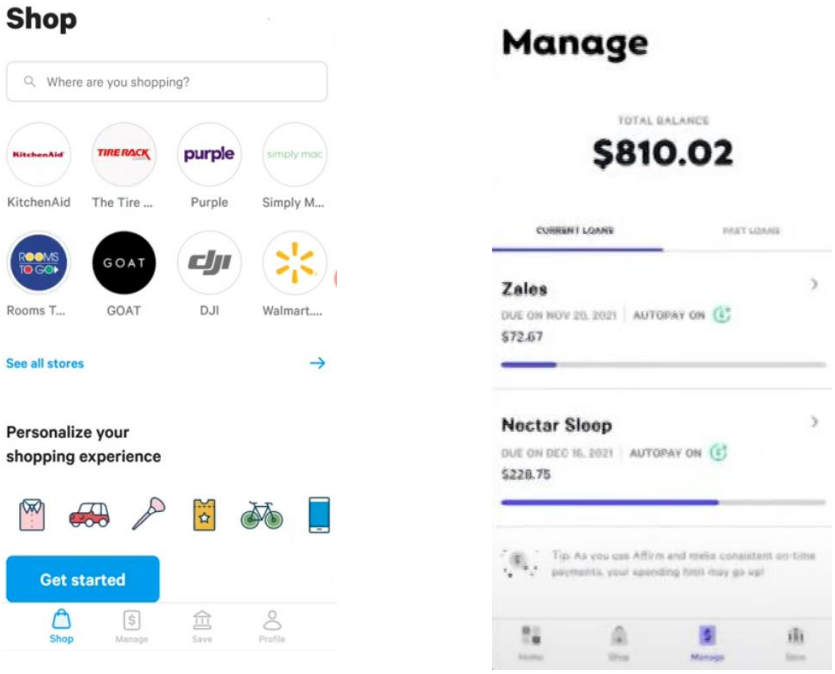


Рисунок 1.9 Дизайн мобільної версії сервісу Affirm

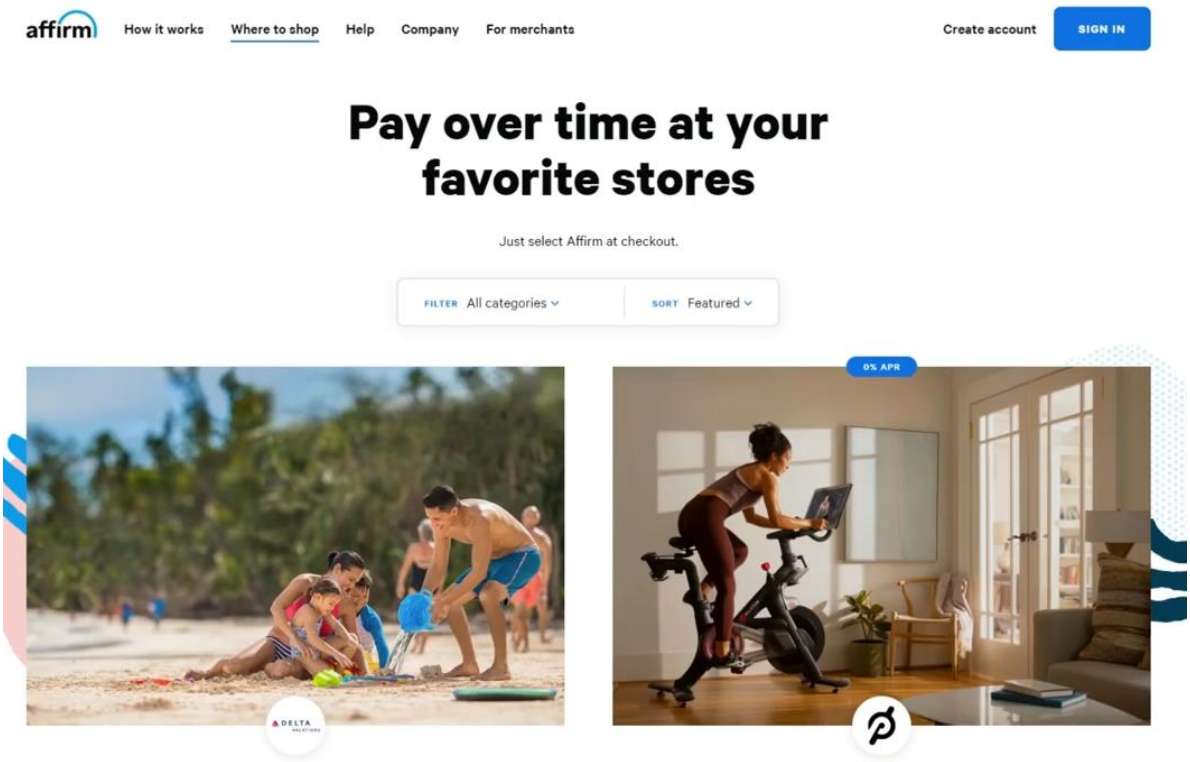


Рисунок 1.10 Дизайн веб-версії сервісу Affirm

Скористуватися сервісом можна під час:

- Оформлення замовлення на сайтах магазинів-мерчантів сервісу;
- Користування веб-сайтом affirm.com;

- Користування мобільним додатком Affirm.

Для користування сервісом також доступна можливість відкриття одноразової (для однієї покупки від незареєстрованого продавця) дебетової картки, а також кредитної картки. Їх можна додати в Apple Pay та Google Pay, де вона буде працювати як звичайна картка Visa. Компанія заробляє на комісії, що відбувається під час транзакції за допомогою цих карток [21].

У випадку відмови користувачем оплачувати товар, компанія повідомляє про це у відповідні органи, що є унікальними для кожної країни.

1.5 Узагальнення особливостей та аналіз

З технічної точки зору, усі розглянуті сервіси мають веб сайт та мобільний додаток для користувача та окремий веб сервіс для співпраці з мерчантами. Через те, що абсолютна більшість сервісів мають такі компоненти базового функціоналу, очевидно, що така модель є найбільш вдалою для сервісу, що знаходиться в розробці. Також виявлено, що всі розглянуті сервіси мають лаконічний та легкий для використання інтерфейс. Для сервісу, що пов'язаний з грошовими операціями та функціонал якого не є зрозумілим на перший погляд, це є особливо важливою характеристикою. Тому це, несумнівно, є обов'язковим пунктом для успішного створення сервісу.

Також було розглянуто основні відмінності щодо ведення бізнесу та фінансових операцій в роботі сервісів. В порівняльній таблиці 1.1 ця інформація представлена узагальнено та систематизовано для швидкого аналізу.

По наведених даних можна побачити, що різні успішні компанії обирають різні фінансові стратегії. Отже, використання будь-якої з них може бути гарним напрямком для сервісу. Тому є сенс надати клієнту або мерчанту можливість самостійно обрати особливості укладення угоди, використовуючи наведені дані як еталоні рамки, за які не можна виходити.

Жоден з наведених сервісів не працює в нашому регіоні, тому, розробка подібного сервісу для нашого регіону може стати досить прибутковою, бо не буде мати аналогів для місцевого ринку.

Таблиця 1.1 – Порівняння бізнес-стратегії популярних сервісів типу BNPL

	Afterpay	Klarna	Affirm
Розмір початкового внеску	25% від ціни за товар	25% від ціни за товар	від 0
Умови виплати за товар	25% від ціни за товар кожні 2 тижня	Гнучке, до 36 місяців	Залежить від типу товару та налаштувань
Відсотки для клієнтів за користування	Немає	Немає	0% або від 10% до 30%
Перевірення кредито-спроможності	немає	м’яка перевірка	Залежить від типу кредиту
Стягування за запізнення виплати	10 доларів за першу неділю прострочення; потім – ще по 7 доларів	7 доларів за кожну неділю	немає
Додаткові фінансові продукти	немає	Оплата протягом 30 днів без відсотків	

Однак, з цим виникають і труднощі, зокрема, процес запровадження відповідальності за несплату позичених коштів.

1.6 Постановка задачі на дипломний проєкт

Описаний програмний продукт буде вирізнітись від усіх інших тим, що він буде доступним на території України для користувачів та для мерчантів з

України та інших країн, в яких українські користувачі найчастіше роблять замовення. Інші відрізні риси сервісу проєкту можно переглянути в таблиці 1.2.

Таблиця 1.2 –Аналогічні сервіси та їх недоліки в порівнянні з продуктом проєкту

Недоліки	Afterpay	Klarna	Affirm
Наявність додаткових відсотків до сплати при запізненні платежу	наявний	наявний	немає
Додаткова плата при покупці товару за користуванням сервісу	немає	немає	наявний
Сервіс може відмінити замовлення клієнта	наявний	немає	немає
Сервіс перевіряє історію банківських рахунків та може відмовити користувачу в функціях в залежності від результатів перевірки	наявний	немає	немає
Сервіс не є доступним для користувачів з нашої країни	наявний	наявний	наявний

Задачею дипломного у є розробка програмного продукту. Програмний продукт матиме три основні функціонуючі компоненти, що будуть розміщені на двох платформах. Продукт матиме мобільний додаток для функціонування засобів для роботи з клієнтом та аналогічний функціонал на веб-просторі. Також в веб-просторі буде функціонал для реєстрації та роботи мерчанта.

На роботу сервісу вплинуть дві проблеми, пов’язані з фінансовими можливостями:

- для сервісу не буде достатнього початкового фінансового капіталу;
- через ситуацію, що склалась в країні важко знайти банк, який міг би фінансувати сервіс.

Зважаючи на це, сервіс буде таким, що мерчант сам надаватиме через сервіс можливість користувачам оплатити їх товари пізніше після купівлі. А сервіс в свою чергу буде надавати користувачам зручний інтерфейс для цієї операції та гарантію мерчанту про оплату товару (або можливість контролю

клієнта) в майбутньому. Для можливості контролю клієнта треба провести подальші аналізи законодавства.

Мерчант матиме наступні можливості:

- реєстрація та авторизація магазину на сайті;
- обрати суму для одного клієнту, оплати якої він готовий очікувати;
- обрати, які внески повинен зробити покупець та на скільки частин буде роздрібнена сума за товар;
- обрати товари та посилання, за якими буде проводитися реклама мерчанта на сайті та в мобільному додатку сервісу;
- налаштовувати можливості для клієнта щодо можливих варіантів порядку оплати товарів: розміру початкового внеску, розміру платежів та їх періодичність. Межі для цих характеристик будуть представлені в рамках аналізу бізнес-моделі, проведеному в попередньому розділі.
- змінювати інформацію профіля та додати інформацію щодо платіжного рахунку, на який користувач, що зробив замовлення буду перераховувати кошти.

На веб-платформі також буде доступ для клієнтів сервісу. Зокрема, для клієнтів буде представлений наступний функціонал:

- реєстрація та авторизація у ролі клієнта на сайті;
- перегляд інформації про мерчантів та їх товари, які вони поставили у рекламі на сайті сервісу;
- можливість додавати паспортну інформацію про себе та змінювати данні свого профілю;
- перегляд товарів, що він вже отримав за допомогою сервісу;
- перегляд своїх здійснених та майбутніх транзакцій;
- перегляд статистики щодо своїх останніх покупок;
- інформація щодо свого рахунку, можливостей його поповнення та списання кошт з нього.

Для клієнту в мобільній версії сервісу буде доступний той самий функціонал, що біло описано для веб платформи з маленькими корективами. В мобільній версії будуть приходити сповіщення щодо транзакцій за його рахунком, що повинні здійснитися у найближчій час. А також, не буде інформації про статистику щодо його останніх покупок. Таким чином, клієнт і скачає мобільний додаток для швидкого та зручного перегляду усієї необхідної інформації і для повідомлень, якщо вони стануть йому у нагоді. І, в той самий час, у нього буде мотивація зайти на веб-сервіс для перегляду додаткової статистики по його профілю.

Обирати сервіс як спосіб оплати товару користувач зможе тільки в магазині зареєстрованого мерчанта. Тому періодичність та розмір транзакцій покупець зможе обрати тільки через веб-версію додатка у розширені для інтернет-магазину, який він обрав серед мерчантів. Такі опції будуть представлені серед тих, що обрав мерчант для цього товару.

В подальшому для сервісу можливо залучення банку та запровадження віртуальної картки для користувачів сервісу. При такому сценарію користувач зможе оплачувати покупки в будь-яких магазинах. Для дипломного у такого функціоналу не буде представлено.

Щодо дизайну додатків, він буде мінімалістичним та неперевантаженим деталями. Адже згідно з результатами аналізу, саме лаконічний інтерфейс є доречним для даного виду додатків.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було виявлено, що платіжні рішення за допомогою сервісів типу BNPL дозволяють магазинам підвищити швидкість реалізації своїх товарів, тим самим збільшуючи їх дохід. Тому багато інтернет-магазинів готово співпрацювати з такими сервісами. Але в Україні наразі не діють подібні сервіси. Тому було розглянуто зарубіжні сервіси типу BNPL, їх роботу та бізнес-стратегії.

Виявлено спільні характеристики роботи сервісів такі, як платформи, функціонал та особливості застосування. Проведено аналіз подібних сервесів, що вже представлено на ринку. Виявлено їх стратегії розвитку та особливості кожного з них. Проаналізовано бізнес стратегію цих сервісів. На основі проведеного аналізу обрано бізнес-стратегію, а також платформи функціонування та характеристики продукту розробки.

Зроблено постановку задачі на дипломний проєкт. Проаналізовано недоліки програмних продуктів, що вже існують на ринку. Ці недоліки будуть вирішені у сервісі проєкту. Описано функціонал сервісу окремо для кожної платформи та для різних типів користувачів: мерчантів та покупців. Також визначено, що лаконічний та неперевантажений деталями інтерфейс є важливою характеристикою даного виду сервісу. Тому прийняте рішення Розробки сервісу саме з таким інтерфейсом.

РОЗДІЛ 2.
ОГЛЯД ІНСТРУМЕНТІВ ДЛЯ СТВОРЕННЯ СЕРВІСУ

2.1 Огляд та аналіз мов програмування для розробки веб-компонентів сервісу

Найважливіша ідея веб-сервісу полягає в тому, що про кожний програмний продукт та кожну компанію обов'язково буде інформація на веб-мережі. Тому кожна компанія повинна спланувати та запровадити свою стратегію розвитку на веб-мережі [27].

З веб-частиною зручніше за все працювати абсолютній більшості клієнтів майбутнього сервісу. Адже він доступний на всіх платформах не залежно від операційної системи. Це також зручно, з боку того, що мерчанти та клієнти не повинні будуть встановлювати додатки на свої пристрої. Для проєкту буде розроблено веб-проєкт з двома частинами: backend та frontend. Розглянемо мови програмування, за допомогою яких може бути створена веб-версія сервісу.

2.1.1 Огляд засобів для Frontend частини сервісу

Словом Frontend називають ту частину веб-програми, яку користувач може бачити. Сторону клієнта. Тож у цьому пункті визначимо засоби для написання візуальної частини веб-сайту.

2.1.1.1 Аналіз мов JavaSript та TypeScript

JavaScript наразі є найпопулярнішою для створення сайтів. Вона має широку популярність серед розробників по всьому світу. Зокрема, вона є популярною тому, що на ній можна писати одночасно backend та frontend для сайту, а також мобільні додатки (про це буде розказано більш детально в наступному підпункті). Також ця мова набула високої популярності через кількість бібліотек та фрейм ворків, які для ней створені.

Майже так само популярною є мова TypeScript. Головна відмінність цієї мови від мови JavaSript полягає в тому, що TypeScript є строго типізованою мовою програмування. Водночас, всі фрейм ворки та бібліотеки JavaScript можна використовувати з TypeScript. Справа в тому, що TypeScript написаний на основі

JavaScript. Тому їх розглядають як майже однакові опції для написання програмного продукту.

Використання мови TypeScript допомагає запобігти винекненню помилок, пов’язаних з типами даних, занчень, що повертають функції та при використанні інтерфейсів класів. В мові TypeScript існують наступні типи даних [31]:

- Boolean
- Number
- String
- Array
- Tuple
- Enum
- Unknown
- Any
- Void
- Null and Undefined
- Never
- Object

Для frontend частини JavaScript потребує застосування додаткових мов програмування: html та css та фрейм ворку, на якому проєкт буде запускатись. Тут існує кілька альтернативКритерій закінчення алгоритму полягає в заданні певної кількості ітерацій (поколінь). Але цей критерій потрібно використовувати обережно, тому що генетичний алгоритм є імовірнісним, і різні запуски алгоритму можуть сходитися з різною швидкістю. Тому в якості порога номера ітерації потрібно задавати досить велике число.

2.1.1.2 Аналіз бібліотеки React

React побудований на реактивній парадигмі та функціональному програмуванні. Це є перевагами цього варіанту. З цим фрейм ворком сервіс буде розділено на компоненти (рис. 2.1), в яких буде зосереджена і логіка під’єднання до сервісу, і розмітка сторінки за допомогою згаданої мови html і її дизайн за допомогою css.

До недоліків фрейм ворку належить те, що через його гнучкість можна заплутатись в імпортуваннях та використанні функцій в різних місцях. Через цю особливість треба створити власну архітектуру та процес роботи. Це допоможе не заплутатись у створених компонентах [28].

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState(state => ({
      seconds: state.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }
```

Рисунок 2.1 – Приклад компоненту фрейм ворку react на мові JavaScript

2.1.1.3 Аналіз бібліотеки Vue

Згідно з бібліотекою Vue, розмітка файлу буде написана разом з функціями та логікою в одному файлі, що має розширення vue. Його переваги в тому, що компоненти з цим фрейм ворком виходять невеликим та їх можна використовувати як частини сторінки або іншого компонента багато разів [30].

2.1.1.4 Аналіз бібліотеки Angular

З фрейм ворком Angular, точніше, AngularJS (AngularJS є сучасною версією фрейм ворка Angular), проєкт буде відрізнятись від React тим, що його буде побудовано на модулях, компонентах та сервісах. Кожен компонент буде мати клас Template, який визначає логіку, і ще два файли, що визначатимуть розмітку сторінки та дизайн (рис. 2.2).

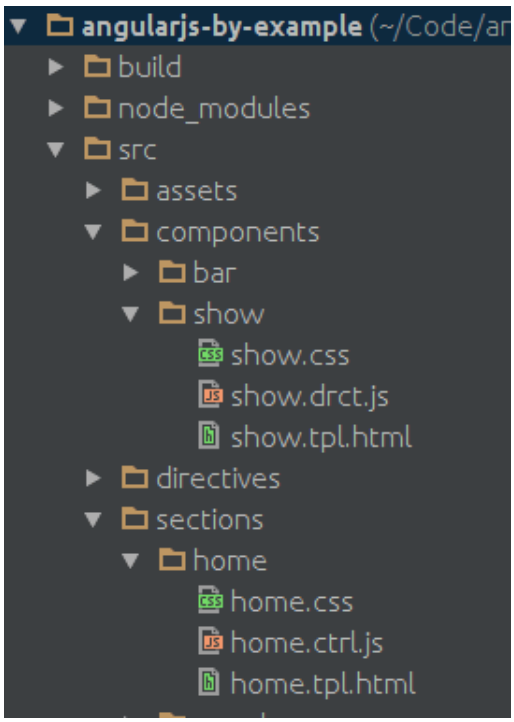


Рисунок 2.2 – Структура проєкту з використанням фрейм ворку Angular

Файл з бізнес-логікою буде написано на мові JavaScript; файл з розміткою – на мові html, файл зі стилями – мовою css. Така компонентна структура не дає заплутатись в модулях під час розробки. [29].

2.1.2 Огляд засобів для Backtend частини сервісу

Словом backend називають частину коду, що відповідає за функціонал сайту: те, як він буде під'єднаний до бази даних, як буде зберігати дані в ній та яку логіку буде виконувати на натискання на кнопки та запити від користувачів. Це сторона сервера.

Зважаючи на те, що для frontend частини вже обрано мови JavaScript або TypeScript, було б логічно писати і backend частину за допомогою однієї з них. Для backend частини зазвичай використовують разом із Node.js.

Node.js створений з нуля з метою обробки асинхронного вводу-виводу. Його поява стала актуальною оскільки він побудований на JavaScript, а JavaScript побудований як цикл подій. Як і подію onclick для кнопки на стороні клієнта, JavaScript обробляє так само події на стороні сервера. Node.js є таким незамінним тому, що хоча інші середовища мають функцію асинхроного запросу вводу-виводу, вони мають її з використанням сторонніх бібліотек або не створені з нуля

для тієї ж мети, як Node.js. Отже, вони повільніші. До таких аналогів відноситься: Event Machine – створена для Ruby, Twisted –створеня для Python, ліцензована за ліцензією MIT з відкритим кодом, і бібліотека мережевих рамок для Apache під назвою Apache MINA, яка також називається «Networking Socket Library» [32].

2.2 Огляд та аналіз мов програмування для розробки компонентів сервісу, що доступні з мобільного пристрою

У минулому розділі було вирішено, що для сервісу окрім веб-частини буде зроблено мобільний додаток. Він має показувати сповіщення, надавати можливості авторизації та збереження особистих даних, а також відображати необхідну інформацію для користувача.

Для створення такого додатка могли б підійти декілька мов програмування та середовищ для розробки мобільних додатків. Розглянемо детальніше найпопулярніші з них.

2.2.1 Особоливості програмування на Kotlin та Java

Розробка для мобільних платформ за допомогою мов Kotlin та Java орієнтована на платформи на базі операційної системи андроїд. Для розробки таких додатків використовують як мінімум дві моаи програмування: Kotlin або Java (на вибір користувача) та мову розмітки об’єктів на сторінці xml.

Щодо вибору між java або kotlin, серед них є невелика різниця, бо мова kotlin з’явилась відносно нещодавно на базі мови java. По суті, вона додає автоматично деякі анотації до функцій та класів та з переходом на неї, трохи змінюється синтаксис для написання коду.

Мова xml запроваджена в проєкти для мобільних платформ для того, щоб представити порядок розміщення елементів на екрані телефону або планшету та для розмітки інших елементів. Для вирішенння проблеми незручності, в таких середовищах розробки, як android studio, наявний графічний інтерфейс для планування розміщення елементів на екрані пристрою (рис 2.3). Ця опція є доволі зручною для розробників-початківців.

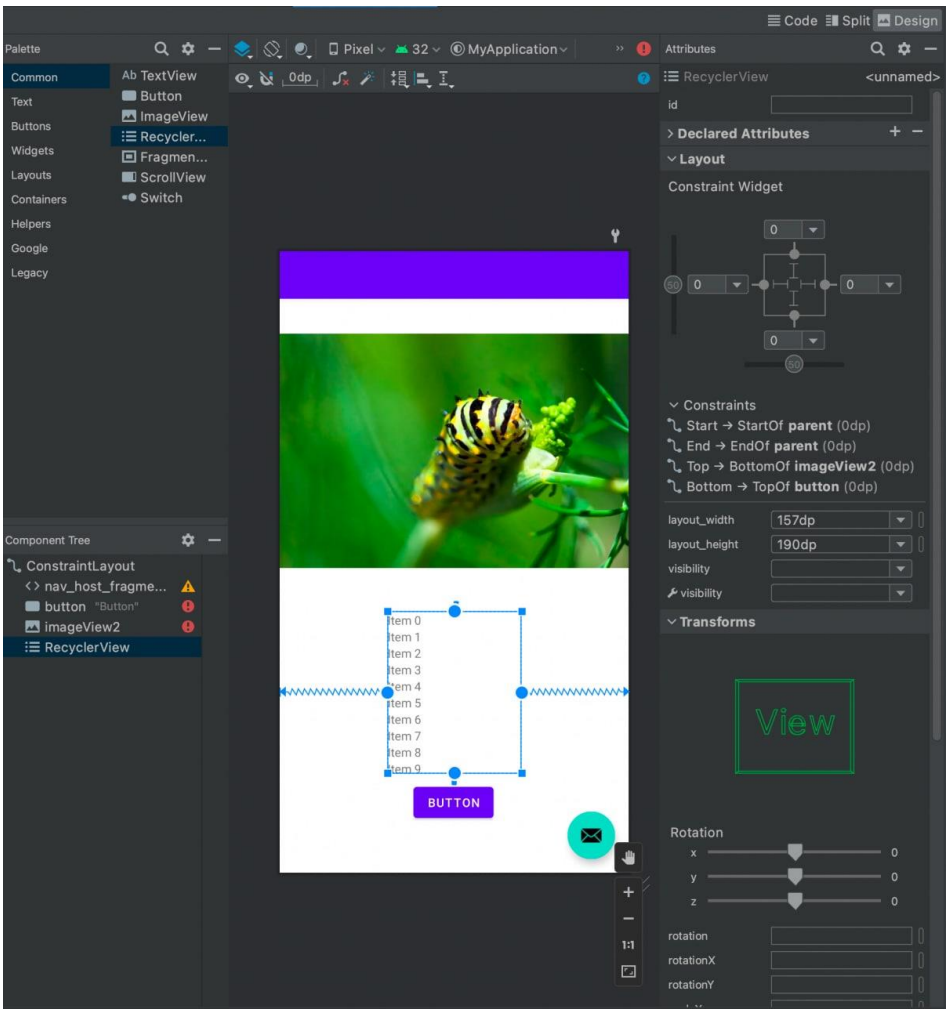


Рисунок 2.3 – Графічний інтерфейс для розміщення елементів на екрані

2.2.2 Особоливості програмування на Swift

Swift - це мова програмування, що використовується для розробки під продукти фірми Apple: ноутбуки на операційній системі MacOS, Apple watch, та мобільні телефони та планшети на операційній системі ios та iPadOS. З цього переліку нас цікавить можливість розробки під мобільні телефони та планшети.

Swift є досить новою мовою програмування. Тому вона заснована на результатах дослідження вже існуючих мов програмування та їх досвідом. Водночас, вона має несхожий на більшість інших мов синтаксис. Вона мультипарадигмова. Управління пам’яттю здійснюється автоматично за допомогою жорсткого, детермінованого підрахунку посилань, що зводить використання пам’яті до мінімуму без накладних витрат на збирання сміття.

Ця мова програмування є типізованою. Тому вона запобігає помилкам та проблемам, що можуть виникнутии при відсутності типізації. Також ця мова вловлює більшість помилок початківців під час компіляції.

Swift можна використовувати для написання серверної частини додатка. Наразі небагато мов програмування забезпечують цією можливістю. Але ця опція не завжди є доречною. Зважаючи на те, що сервіс матиме як мобільну частину, так і веб, не має сенсу переписувати серверну частину окремо на мові Swift.

2.2.3 Особоливості програмування на TypeScript та JavaScript на базі бібліотеки React Native

TypeScript та JavaScript можуть бути використані не лише для веб частини додатку, але і для мобільної. Для того, щоб існувала ця можливість, створено фреймворк React Native. Він розроблений для транслювання JavaScript або TypeScript коду на нативний код платформ. Таким чином, розробка могла б відбуватись повністю на одній мові програмування. Адже додатки написані за допомогою React Native можна встановити і на телефони з операційною системою android, і на телефони з операційною системою ios.

2.2.4 Особоливості програмування на Dart на базі Flutter

Flutter - це комплект розробки програмного забезпечення (software development kit) для створення додатків для мобільних, веб платформ, а також консолі та персонального комп’ютера з єдиної кодової бази. Тут потрібно зазначити, що під мобільними додатками мається на увазі додатки для як платформи Android, так і платформи IOS. Flutter робить процес розробки дуже легким, адже додатки можна скомпілювати (запустити після редагування вихідного коду) за долі секунд.

Однак історія Flutter розпочалась не так давно. Він був представлений на саміті розробників Dart у 2015 році. Перша версія Flutter мала назву “Sky” і працювала на операційній системі Android.

Після того, як Google випустив альфа-версію цієї технології в 2017 році, інтерес до нього швидко зростав. Ще до офіційного запуску в 2018 році, Flutter

швидко завоював неймовірну популярність. Це підтверджує кількість зірок на GitHub (Flutter має відкритий вихідний код на цій платформі): 141 тисяча. (Для порівняння, на проєкті з вихідним кодом React Native зараз 103 тисячі зірок.).

За допомогою Flutter вже було створено багато додатків. Серед найпопулярніших:

- китайський гігант електронної комерції Alibaba;
- головний додаток, що відповідає за рекламу від компаній Google, Google ads;
- офіційний додаток для відомих бродвейських мюзиклів — Hamilton.
- додаток на основі штучного інтелекту на основі когнітивно-поведінкової терапії та медитації — Reflectly.
- додаток, що пропонує шаблони та варіанти редагування історій та фотографій Instagram – Postmuse.

В якості мови програмування Flutter використовує Dart. Це є чи не найголовнішою причиною, для розробників використовувати цю технологію. Адже Dart — це об’єктно-орієнтована мова, розроблена компанією Google, яку легко вивчити, особливо якщо у розробника є досвід роботи з такими мовами програмування, як Java, C# або JavaScript..

Щоб допомогти розробникам, наприклад, виправляти помилки та додавати функції максимально швидко, Flutter пропонує інструмент під назвою Hot Reload. Це дозволяє відразу побачити зміни, внесені в код, без перезапуску програми. У разі розробки Native додаток слід часто перебудовувати, що може зайняти значну кількість часу. Hot Reload прискорює процес розробки, цим самим покращуючи життя розробників.

Flutter надає багатий каталог вбудованих компонентів інтерфейсу, які можна легко налаштувати. Існує два набори віджетів: віджети дизайну material (для платформи Android) та віджети cupertino (у стилі IOS), щоб зробити додаток ергономічним і звичним на вигляд та для використання на обох платформах.

Крім того, коли Flutter працює на пристроях зі старими версіями Android та IOS, він працює так само добре, як і на пристроях з сучасними операційними системами.

Додатки, створені за допомогою Flutter, мають гарантовану багаторічну підтримку від Google, оскільки компанія використовує саму технологію. Ви можете бути впевнені, що Google буде продовжувати виправляти помилки, випускати нові версії та робити внесок у технологію, наскільки це можливо. Розробка додатків Flutter бере участь у багатьох проєктах Google, таких як Google Fuchsia, що є доказом того, що Flutter буде існувати довгий час.

Dart компілюється у двійковий код, тому швидкість операцій порівнянна з Objective-C, Swift, Java та Kotlin, а також немає необхідності у додаткових застосунках між додатком і платформою. Іншими словами, Dart поєднує в собі багато хороших рішень, які допомагають швидше вирішувати складні завдання, що чудово пасує до Flutter.

Розробники Flutter заявляють про постійну швидкість 60 кадрів в секунду, що є швидкістю плавного і чіткого зображення сучасних екранів. Ця можливість є особливо актуальною для ігрової індустрії. Адже ігри повинні працювати гладко і швидко. Інакше користувачі будуть дуже розчаровані і відмовляться від додатка. Розробка додатків Flutter — чудовий вибір для створення високопродуктивного ігрового додатка, а також для інноваційних функцій штучного інтелекту та нейронних мереж.

Сервіси на вимогу (on demand) є одним з наймодніших ринків для створення додатків. Особливо актуальними вони є під час карантину, коли люди не мають змоги, або обмежені у можливості виходу на вулицю та купувати продукти харчування, ліки, одяг тощо. Саме тут знайшли свій потенціал програми на вимогу. Розробка додатків Flutter забезпечує оригінальну продуктивність, винятковий дизайн та чудовий користувальницький інтерфейс, що робить його першою технологією на ринку для цієї та ще багатьох категорій додатків.

Для розробки системи в даній роботі було обрано мову програмування C#, чому сприяло багато факторів, які будуть послідовно надаватися в наступних пунктах.

2.3 Вибір сервісу для роботи з базою даних

Для сервісу обов'язково потрібна база даних, за допомогою якою він буде функціонувати. Ця база повинна справлятися з задачею розсилання нотифікацій та аутентифікації.

2.3.1 Firebase

Firebase — це база даних типу NoSQL, що розміщена на серверах Google. Вона підтримує функціонування як з мобільними пристроями, так і з веб-додатками. Для її користування достатньо реєстрації на сервисі firebase, реєстрації там проєкту, підключення його до локального проєкту. Після цього, можна використовувати функції firebase, головни з яких:

- авторизація
- зберігання даних
- відсиланя повідомлень

Значною перевагою firebase є те, що дані в ньому синхрронізуються. Отже, якщо з одного користувача надходить повідомлення, воно буде одразу передано між усіма його пристроями та до інших користувачів, якщо це важливо [33]. Ця особливість є дуже доречною для нашої програми, адже користувач зможе мати дані без затримок як на веб-клієнті, так і на мобільному, або навіть декількох мобільних.

До її недоліків належить обмежений термін безкоштовного користування та платна підписка.

2.3.2 PostgreSQL

PostgreSQL — це розширена реляційна система управління базами даних (СУБД) корпоративного класу з відкритим кодом, яка підтримує запити як SQL (реляційні), так і JSON (нереляційні). Це високостабільна система керування базами даних, що підтримується більш ніж 20-річним розвитком спільноти, що

сприяло її високому рівню стійкості, цілісності та коректності. PostgreSQL використовується як основне сховище даних або сховище даних для багатьох веб-, мобільних, геопросторових та аналітичних додатків. Остання основна версія — PostgreSQL 12.

PostgreSQL має багату історію підтримки розширених типів даних і підтримує рівень оптимізації продуктивності, звичайний для його комерційних аналогів баз даних, таких як Oracle і SQL Server. AWS підтримує PostgreSQL через повністю керовану службу баз даних із Amazon Relational Database Service (RDS). Amazon Aurora з сумісністю PostgreSQL також створена за допомогою PostgreSQL.

До переваг PostgreAQL належить:

- Багатий функціонал та розширення
- Надійність і відповідність стандартам
- Відкритий код (можна змінювати функції та методи)
- Безкоштовна підписка та доступ до БД

2.3.3 MySQL

Як і PostgreSQL, MySQL є системою керування базами даних з відкритим кодом та є чудовим інструментом для аналізу та комерських проєктів. Ці дві системи є досить схожими

Для сервісів, що надають можливість користувачам використовувати хмарне сховище та гнучкі функції аналізу, бази даних MySQL є дуже актуальними через їх швидкість, що робить обмін великим об’ємом даних більш зручним[35].

До переваг MySQL відноситься:

- підтримука роботи з багатьма операційними системами: Linux, Windows, Solaris та Unix та доступність для багатьох мов програмування: C, Java, php, Python, C++, Perl та інші.
- Безперервне з’єднання, що забезпечує цілісність та захист даних. Підтримка протоколів TCP та UDP.

- Складні алгоритми шифрування, що забезпечують надійний захист даних
- Відкритий код та бюджетні ціни за користування

2.4 Загальні висновки щодо аналізу та вибір технологій та мов програмування

Для розробки сервісу прийнято рішення використовувати Type Script для веб-частини сервісу. React для frontend частини та Node для backend-у.

Не зважаючи на можливість писати проєкт на одній мові програмування з React Native для мобільної частини, прийнято рішення використовувати Flutter та мову Dart для мобільної частини сервісу. Він має дуже багато переваг в плані зручності, швидкості та легкості написання коду в порівнянні з іншими мовами.

Для керування базою даних обрано СУБД PostgreSQL адже це є найбільш доступним рішенням та найкраще підходить для поставлених задачей щодо зберігання та керування даними. Для проєкту не плануються великі обсяги зберігання даних та швидкий доступ клвєнтів для них, як і велика логіка щодо аналізування цих даних. Одже, MySQL є недоречним для проєкту.

2.5 Вибір інструментів для розробки

В якості інструментів для розробки буде використано систему контролю версій Git та її веб-репозиторій GitHub. Як система контролю версій, git забезпечує:

- Централізований контроль версій, що зусереджений на синхронізації, та доступ до резервної копії файлів.
- Розподілений контроль версій проєкту, що зосереджений на спільному доступі до змін (комітів). Кожна зміна має унікальний ідентифікатор.
- Розподілені системи не мають визначеної структури. Ви можете легко мати стиль SVN, централізовану систему з git.

В якості IDE (середовища розробки) буде використано продукт компанії JetBrains: Android Studio для розробки мобільного додатка та його функціонал android емулятора для його тестування додатка. Для тестування програми на ios-емуляторі, буде використано емулятор від IDE xCode компанії Apple.

Для розробки веб-додатка обрано продукт компанії Microsoft - Visual Studio Code. Адже він підтримує багато плагінів для комфортної роботи та має багато розширень.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі проведено оглад інструментів розробки. Зокрема:

- Мови програмування JavaScript/TypeScript для клієнтської частини веб-додатка (frontend) та аналіз бібліотек для цієї частини.
- Мови програмування для серверної частини додатка (backend) та бібліотеки NodeJS
- Детальний аналіз мов програмування та бібліотек для мобільної частини сервісу та обрано технологію, за допомогою якою буде розроблено мобільну частину.
- Детальний аналіз СУБД, що можуть бути використані для сервісу та вибрано одну, яка найбільш підходить до задач сервісу.

В якості головної мови програмування було оборано TypeScript з бібліотекою React та NodeJS для web-частини та Dart з бібліотекою Flutter для мобільного додатку. Як систему керування базами даних було обрано PostgreSQL за його доступність, пасування для різних платформ та відповідність вимогам до сервісу.

Також обрано інструменти для програмування, зокрема, git для контролю версій, github в якості хмарного сховища, Android Studio для розробки мобільного додатку та Visual Studio Code для розробки веб-частини сервісу.

РОЗДІЛ 3.
ОПИС СТРУКТУРИ СЕРВІСУ ТА ЙОГО КОМПОНЕНТІВ

Відповідно до описаної інформації щодо способу реалізації та компонентів програмного продукту, в цьому розділі буде описано фазу розробки програмного продукту. Також будуть описані основні частини розробленого продукту, функціональні блоки програми та схема їх взаємодії.

Структурна схема роботи сервісу представлена в додатку 2.

3.1 Опис об’єкта розробки та його особливості

Як було зазначено в розділі 1, матиме три основні функціонуючі компоненти: серверну частину, клієнтську частину та мобільну частину. У серверній частині буде здійснюватися контроль над базою даних та бекенд для арі-запитів. У клієнтській та мобільній частині буде доступ для мерчантів та їх клієнтів.

Мерчант матиме наступні можливості:

- реєстрація та авторизація магазину на сайті;
- обрати суму для одного клієнту, оплати якої він готовий очікувати;
- обрати, які внески повинен зробити покупець та на скільки частин буде роздрібнена сума за товар;
- обрати товари та посилання, за якими буде проводитися реклама мерчанта на сайті та в мобільному додатку сервісу;
- налаштовувати можливості для клієнта щодо можливих варіантів порядку оплати товарів: розміру початкового внеску, розміру платежів та їх періодичність. Межі для цих характеристик будуть представлені в рамках аналізу бізнес-моделі, проведеному в попередньому розділі.
- змінювати інформацію профіля та додати інформацію щодо платіжного рахунку, на який користувач, що зробив замовлення буду перераховувати кошти.

На веб-платформі також буде доступ для клієнтів сервісу. Зокрема, для клієнтів буде представлений наступний функціонал:

- реєстрація та авторизація у ролі клієнта на сайті;
- перегляд інформації про мерчантів та їх товари, які вони поставили у рекламі на сайті сервісу;
- можливість додавати паспортну інформацію про себе та змінювати данні свого профілю;
- перегляд товарів, що він вже отримав за допомогою сервісу;
- перегляд своїх здійснених та майбутніх транзакцій;
- перегляд статистики щодо своїх останніх покупок;
- інформація щодо свого рахунку, можливостей його поповнення та списання кошт з нього.

Для клієнту в мобільній версії сервісу буде доступний той самий функціонал, що біло описано для веб платформи з маленькими корективами. В мобільній версії будуть приходити сповіщення щодо транзакцій за його рахунком, що повинні здійснитися у найближчій час. А також, не буде інформації про статистику щодо його останніх покупок. Таким чином, клієнт і скачає мобільний додаток для швидкого та зручного перегляду усієї необхідної інформації і для повідомлень, якщо вони стануть йому у нагоді. І, в той самий час, у нього буде мотивація зайти на веб-сервіс для перегляду додаткової статистики по його профілю.

3.2 Характеристика компонентів мобільного додатка

Усі класи, що відображаються в інтерфейсі, є віджетами на Flutter: картинки, кнопки, надписи, функціональні блоки та області. Тому, кастомні елементи інтерфейсу мають наслідувати віджети StatelessWidget, StatefulWidget або похідні від них. Всі ці класи входять до бібліотеки Flutter. Для розробки під мобільні пристрої будуть використані бібліотеки material.dart та cupertino.dart для андроїд та ios платформ відповідно. Як вже було вказано у розділі 2, для

керування даними у віджетах буде використано блоки бібліотеки flutter_bloc та bloc.

Компіляція коду dart починається з функції main(). В і вона спочатку ініціалізує віджетита api, що будуть використані. Після цього запускає головний віджет додатка, що має назву MyApp. Він призводить до дії віджет App, який додає у дерево віджетів BlocProvider, забезпечує вкладені віджети інформацією про стан авторизації (рис. 3.1). Одразу виконується метод цього блоку, що ініціалізує стан авторизації.

```
— BlocProvider(
      create: (context) =>
          AuthenticationBloc()..add(AuthenticationInitialize())),
],
```

Рисунок 3.1 — BlocProvider, додає інформацію про стан авторизації в дерево віджетів

Далі в дереві віджетів будується stateful клас, який, в свою чергу, генерує MaterialApp віджет. В цей віджет запроваджує навігацію по сторінкам додатку, локалізація мови для додатка (рис. 3.2). Для навігації по сторінкам додатку, використовується клас Routes, що містить константні строкові значення, що відповідають посиланням на сторінки, що будуть доступні для користувача.

Для локалізації інтерфейсу (використання різних мов) використовується бібліотека flutter_localizations. Вона генерує функції-getters для доступа до локалізованих строк, що вказані в окремому файлі з розширенням arb (рис. 3.3). Цей підхід дозволяє суттєво спростити роботу при запровадженні додатка в інші країни.

В залежності від інформації про зміни стану автентифікації, генерується поточна сторінка інтерфейсу (рис. 3.4). Якщо користувач автентифікований, йому буде показана головна сторінка додатка — сторінка з інформацією про мерчантів. Якщо не автентифікований, йому буде показана сторінка логіну (що містить кнопки для перенаправлення на сторінку реєстрації так скидання

```
return MaterialApp(
  navigatorKey: _navigatorKey,
  localizationsDelegates: [
    S.delegate,
    GlobalMaterialLocalizations.delegate,
    GlobalWidgetsLocalizations.delegate,
    GlobalCupertinoLocalizations.delegate,
  ],
  supportedLocales: [
    const Locale('en', ''),
  ],
  routes: {
    Routes.splashScreen: (context) => SplashScreen(),
    Routes.loginPage: (context) => LoginPage(),
    Routes.registerPage: (context) => RegisterPage(),
    Routes.restorePasswordPage: (context) => RestorePasswordPage(),
    Routes.forgotPasswordPage: (context) => ForgotPasswordPage(),
    Routes.merchantsPage: (context) => MerchantsPage(),
    Routes.dashboardPage: (context) => DashboardPage(),
    Routes.orderTransactions: (context) => OrderTransactionsPage(),
    Routes.editProfilePage: (context) => EditProfilePage(),
    Routes.editPasswordPage: (context) => EditPasswordPage(),
    Routes.ordersPage: (context) => OrdersPage(),
  },
  builder: (context, child) {
```

Рисунок 3.2 — Визначення локалізації інтерфейсу та навігації для сторінок в дереві віджетів

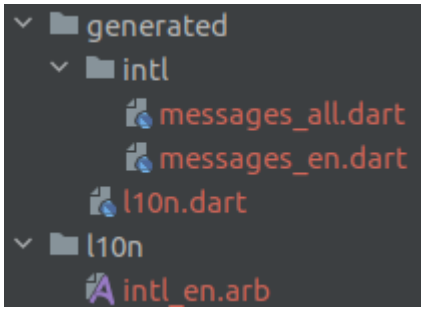


Рисунок 3.3 — Структура файлів, що використовуються для локалізації

паролю. У період, коли інформація про авторизацію завантажуєтсья, користувачу буде представлений екран завантаження.

Зі сторінки логіну користувач може перейти на сторінку скидання паролю та на сторінку реєстрації. Це сторінки авторизації. В кожній з них присутня форма з текстовими полями для вводу даних, кнопка відправлення запиту, що відповідає призначенню сторінки, та кнопки-надписи перенаправлення на інші доступні сторінки авторизації. Зі сторінок авторизації, на данному етапі для

```
BlocListener<AuthenticationBloc, AuthenticationState>(
  listener: (context, state) {
    if (state is AuthenticationAuthenticated) {
      _navigator?.pushNamedAndRemoveUntil(
          Routes.merchantsPage, (route) => false);
    } else if (state is AuthenticationUnauthenticated) {
      _navigator?.pushNamedAndRemoveUntil(
          Routes.loginPage, (route) => false);
    } else {
      _navigator?.pushNamedAndRemoveUntil(
          Routes.splashScreen, (route) => false);
    }
  },
), // BlocListener
],
```

Рисунок 3.4 — BlocListener, що налаштований на стан автентифікації та в залежності від нього обирає сторінку інтерфейсу

користувача не доступно перенаправлення на сторінку вводу нового паролю, адже ця сторінка повинна бути доступною тільки по переході за посиланням для скидання паролю.

На сторінках авторизації кожне поле має валідацію для захисту на ввід некоректних даних. Для сторінки реєстрації використано бібліотеку intl_phone_number_input для коректної валідації номеру користувача. Валідуються поля (видно повідомлення про не валідність поля) після натискання користувачем на кнопку запиту цієї сторінки. Для керування даними та здійсненн запитів також використовується бібліотека з блоками. У разі помилки, що може виникнути під час запиту, на екрані показується на декілька секунд SnackBar з повідомленням про цю помилку.

Сторінка логіну окрім стандартної кнопки дії (з функціоналом відправки запиту входу в обліковий запис), має кнопку входу в акаунт за допомогою відбитку пальця (або іншого методу, що підтримує телефон користувача. Користувач може її увімкнути в налаштуваннях акаунту. Коли він вмикає цю функцію, додаток ще раз просить у користувача ввести пароль від свого облікового запису. Після цього credentials користувача зберігаються в пам’ять

додатку під ключем-відбитком пальця. Таким чином, наступного разу користувач зможе зайти в свій обліковий запис використовуючи сканер відбитку пальця.

Що стосується внутрішніх сторінок інтерфейсу, перша сторінка, яку бачить аутентифікований користувач — це сторінка з мерчантами. В ній представлено мерчантів емблемою та підписом. При натисканні на підпис або емблему користувача перенаправляє на сайт мерчанта. Також на цій сторінці реалізовано пошуковий функціонал (по назві мерчанта) та пошук по категоріям. Для функціоналу пошуку використовується query-запити. Також в нижній частині сторінки представлені найпопулярніші мерчанти.

Для навігації по внутрішнім сторінкам інтерфейсу, використовується side drawer. Перехід між сторінками, що доступні за drawer виконується за допомогою вже впровадженою у дерево віджетів навігації (рис. 3.4). Це є найбільш зручний спосіб навігації адже він дозволяє переходити з будь-якої сторінки, що додано в список, в будь-яку інші з цього списку.

Через дравер можна перейти на сторінки зміни даних користувача, dashboard, сторінки заказів та виконати вихід з облікового запису.

```
Navigator.pushNamed(context, Routes.dashboardPage);
```

Рисунок 3.4 — Використання методів для навігації по сторінкам інтерфейсу

На сторінці зміни профілю представлено віджети для зміни інформації облікового запису. Для збереження змін, в верхній частині екрану є кнопка “save”. Якщо користувач змінив дані на сторінці (будь-які крім зміни окрім зміни фото профілю), тоді йому потрібно натиснути цю кнопку для не натиснув кнопку збереження, при натисканні кнопки “назад”, буде показано діалог, що інформує про незбережені зміни і питає, чи хоче користувач здійснити вихід без збереження, чи зберегти зміни перед виходом.

З цієї сторінки доступен перехід на дочірні сторінки зміни даних облікового запису. Це – сторінки:

- зміни адресу, де є поля для поштового коду, міста, області (або штату) та вулиці.
- Зміни сімейного статусу, де є поля для кількості дітей та bullet перемикач для статусу (в браці, або ні)
- Сторінка з перемикачом налаштувань входу: чи може користувач здійснювати логін в обліковий запис за допомогою сканеру відбитку пальців, або інших біометричних даних - користувач обирає самостійно.
- Сторінка для зміни паролю від облікового запису. На ній міститься форма. Після введення поточного паролю та двох разів нового паролю і та натискання на кнопку дії (зберігання нового паролю), пароль користувача змінеться.

Якщо користувач хоче змінити фото профілю, йому буде показано діалог для визначення джерела фотографії. Фотографію можна обрати з галереї, або з нового снімку камери. Коли обрана нова фотографія профілю, не потрібно натискати на кнопку збереження змін.

На сторінці dashboard міститься інформація щодо балансу користувача та транзакцій. З акаунту користувача за допомогою блок-структури дістається інформація про баланс користувача. На основі історії транзакцій, що представлена під інформацією про баланси, також показано, скільки коштів користувач уже сплатив за свої покупки та скільки загалом коштів йому треба сплатити на заплановані (ті, які вже прийняті та знаходяться в процесі оплати) товари. Для майбутнього функціоналу створення та перегляду балансу на віртуальній картці, створено 3D анімацію перегортання для кожного блоку-картці балансу.

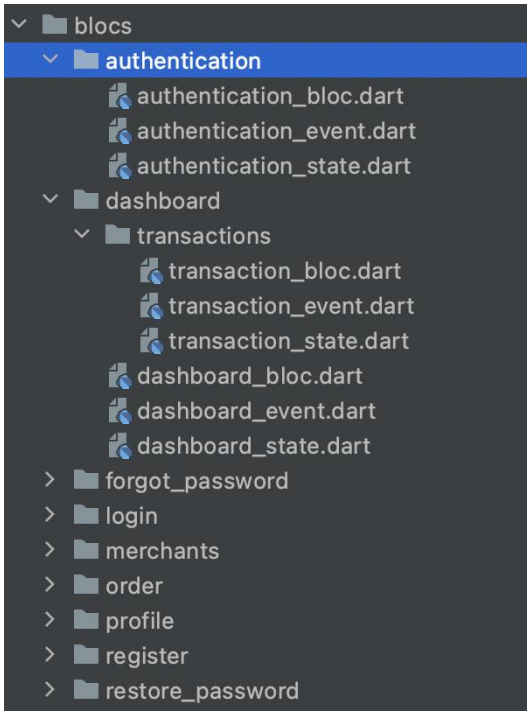
Список з транзакціями підгружається по мірі того, як користувач його гортає. Спочатку користувачу доступно 10 транзакцій для перегляду. Коли користувач передивився дев’яносто відсотків списку, відбувається запит на дзавантаження ще 5 транзакцій. Це зроблено для плавної роботи та запобігання великого завантаження оперативної пам’яті пристрою.

На сторінці заказів показано інформація про закази, які користувач вже здійсним за допомогою сервісу. При натисканні на заказ, користувача перенаправляє на сторінку з деталями замовлення. На ній показано транзакції по конкретно обраному замовленні. Для зручності, оплачені транзакції відрізняються від неоплачених кольором.

Усіма даними на сторінках керують блоки. Вони відповідають за заміну стану сторінки, посилання запитів на сервер та надання даних для сторінки, за яку цей блок відповідає. Для деяких сторінок створено декілька блоків для комфортної роботи з даними (рис. 3.5). До блоку нележать як мінімум три класи: сам блок (bloc), його стан (state) та подія (event) (рис. 3.5).

Рисунок 3.5 - Папки, що місятять блок структуру та структура файлів всередені них.

Класи інтерфейсу викликають класи з файлу event (різні івенти) щоб змінити state. State івдповідає за надання даних для інтерфейсу. Уся логіка зміни даних та здійснення API запитів розміщена в файлі bloc. Для цього там



перевизначена функція, яка відслідковує зміни, які інтерфейс робить за допомоги посилання подій. В залежності від події, робляться відповідні зміни.

Для визначення адреси домену бекенду, на який будуть надходити запити, створено файл lib/config/host_config.dart. У ньому розміщена логіка для надсилання запитів. Наразі запити на порт локалхоста на машині, де запущено бекенд, та коли буде готовий сервер, його можна вказати в константу _domain, і тоді всі запити програма буде робити саме на нього.

Повний код веб-додатку та структурну схему клвсів блоків та класів сервисів для них можна переглянути в 4 додатку до роботи.

3.3 Характеристика компонентів web-додатка

Веб-частина додатка складається з декількох частин: бекєнду, фронтеду та функціоналу для мерчантів. По суті, це три окремі проєкти. Що написані на TypeScript.

У бекенді в кремій директорії лежать файли, що відповідають за API запити. Вони потребують схеми, які будуть видавати на запит з клієнту. Ці схеми розміщені в директорії shared, що відповідає за ресурси, які потрібні для обох частин, фронтенду та бекенду сервісу. Файли, що написані в папці shared, створені для обох веб-частин сервісу.

Функціонал, що відповідає за API запити залежить від репозиторіїв, що лежать в директорії data/repository. Функції в репозиторіях дістають дані з бази даних за допомогою сервісів. Більш детально цей алгоритм описано в додатках. В додатку 5 представлено код файлів, що відноситься до серверної частини.

Сервісна частина проєкту містить інтерфейс, що написано на мові typescript та функціонал, що відсилає API запити на сервер.

Як і для мобільного додатку, в сервесній частині присутні валідатори на текстові поля, відобрежаня усієї необхідної інформації, що буде більш детально описано у наступному розділі. Більш детально про реалізацію сервісної частини веб-у можна дізнатись, переглянувши додаток 6.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було розглянуто файли, що відносяться до проєктів сервісу. Це показало загальну картину роботи сервісу та внутрішню структуру проєкту.

Було розглянуто розробку мобільного додатку, що відповідає за інформування клієнтів та нотифікації. Окрім нотифікацій, мобільний додаток включає в себе функціонал авторизації, зміни даних облікового запису, інформування користувача про мерчантів сервісу, інформування про транзакції та замовлення користувача.

Також описано структуру веб-частини сервісу та схему, по якій вона працює. Побудовано функціональну, принципову та структурну схеми. Написано програмний код, що відповідає вимогам, висунутим до сервісу та побудованим схемам. Програмний код та схеми можна переглянути в додатках до роботи.

РОЗДІЛ 4.
ІНСТРУКЦІЯ З ВИКОРИСТАННЯ РОЗРОБЛЕНОГО СЕРВІСУ

Базуючись на розробленій системі пошуку екстремуму багатомірної функції метою даного розділу є показати результати її роботи. Також потрібно провести порівняльну характеристику розроблених алгоритмів, дослідити ефективність роботи кожного з них на прикладі задачі глобальної оптимізації.

Схема функціонування сервісу представлена в додатку 1.

4.1 Інструкція для користування мобільним додатком

Мобільний додаток доступний тільки для клієнтів сервісу. Коли мобільний додаток відкривається вперше, користувачу доступна сторінка логіну (рис. 4.1). Після вводу логіну та паролю. Користувач заходить в сиистему натискаючи на кнопку входу. При некоректних даних, вхід в систему не здійснюється. Користувачу показано повідомлення про те, яке з полів містить некоректні дані. Після їх зміни, повідомлення зникає.

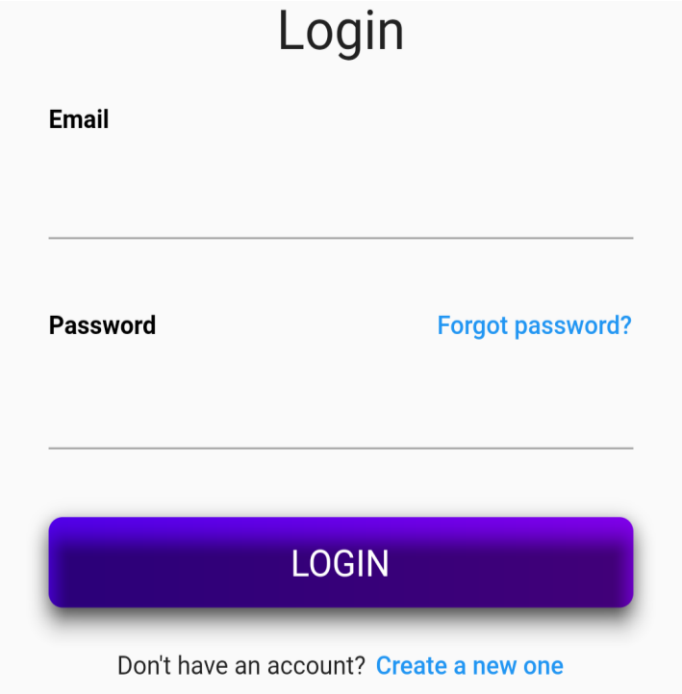
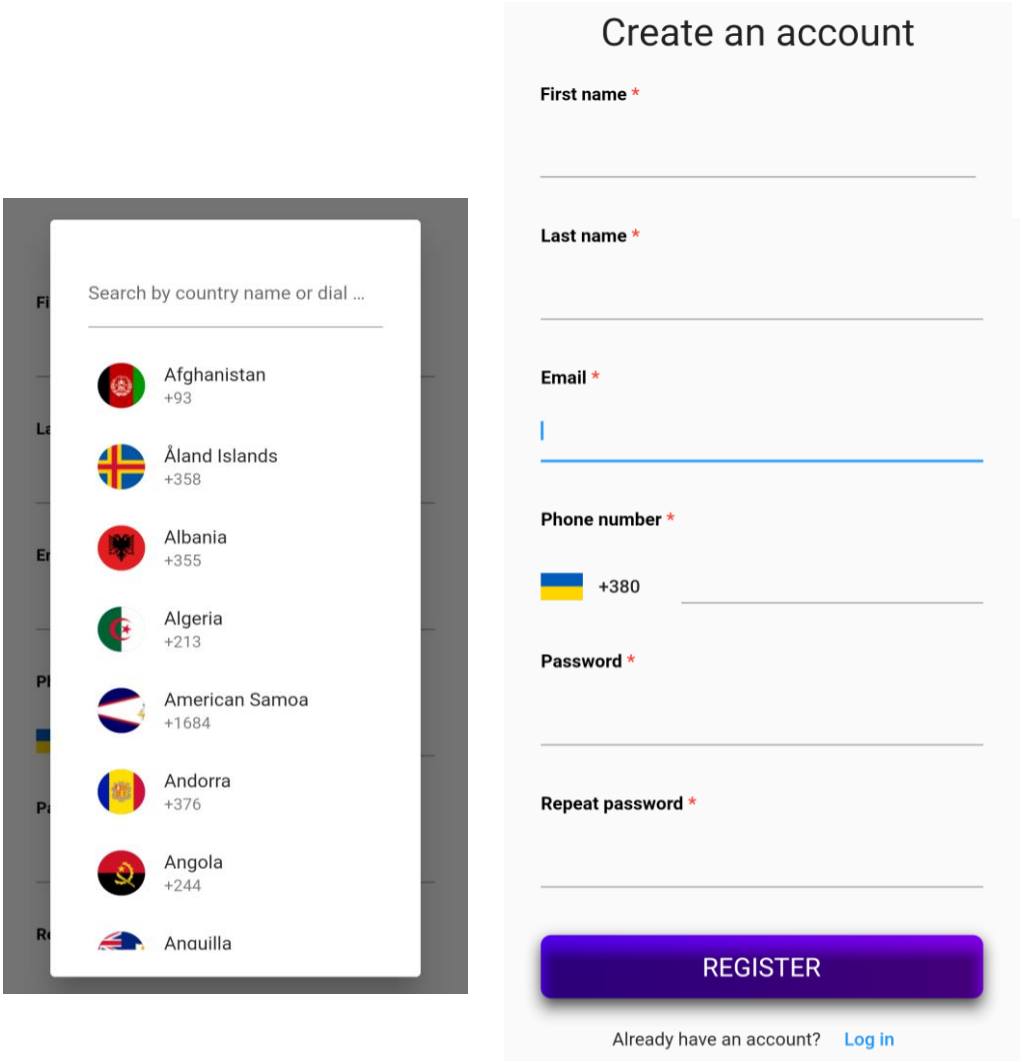


Рисунок 4.1 - Функціонал та іінтерфейс сторінки логіну

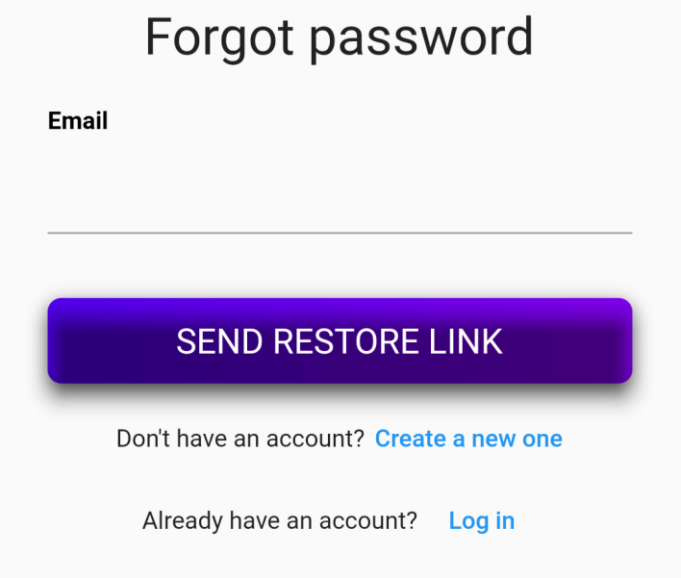
З сторінки логіну користувач може перейти на сторінку реєстрації. Там потрібно заповнити усі поля для вводу коректними даними та натиснути кнопку реєстрації для реєстрації (рис. 4.2). Для вводу номеру телефону, користувачу потрібно натиснути на флаг та обрати країну, в якій зареєстровано його номер. Після цього ввести номер без коду країни. Пароль користувача повинен містити великі та малі букви, символи та цифри. В іншому випадку, система не дасть зареєструватись в сервісі. Також потрібно ввести валідний електроний адрес, номер телефону та ім’я. В разі введення невалідних даних, висвічується повідомлення про невалідність.

Рисунок 4.2 – Функціонал та інтерфейс сторінки реєстрації



Для скидання паролю, користувачу потрібно перейти на сторінку скидання паролю. На сторінці скидання паролю користувач повинен ввести свію

електронну адресу (рис. 4.3), після чого йому прийде лист з посиланням на зміну адреси. Якщо користувач відкриває це посилання на мобільному пристрої з встановленим сервісом, додаток сервісу автоматично відкриється та користувачу буде відкинув сторінка, в якій він може вести свій новий пароль двічі. Після натискання кнопки збереження паролю, для облікового запису буде поновлено

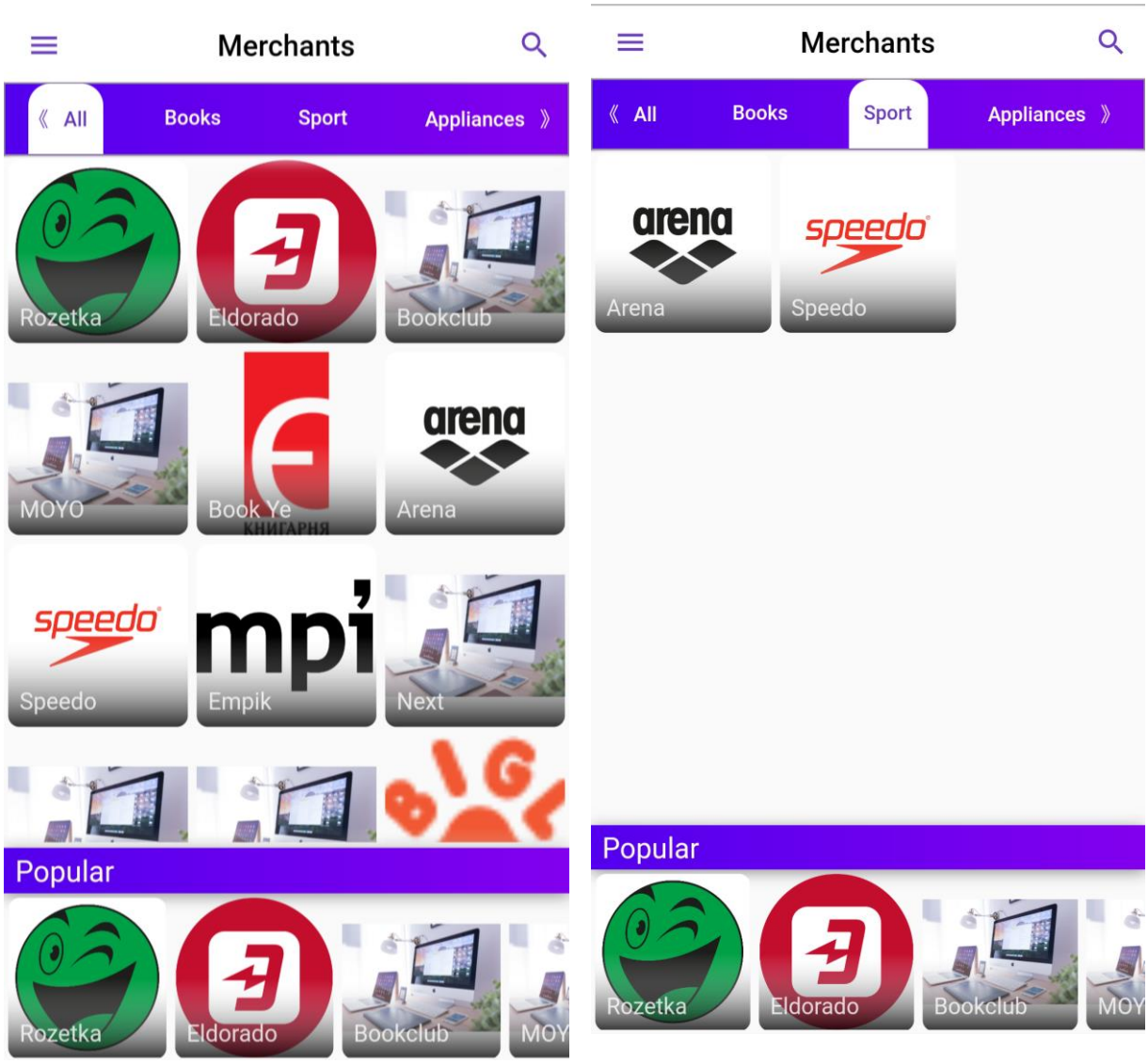


пароль та авторизація в сервісі відбудеться автоматично.

Рисунок 4.3 - Функціонал та інтерфейс сторінки скидання паролю

Після входу в обліковий запис, користувачу буде представлена сторінка з мерчантами сервісу (рис. 4.4). За натисканням на мерчантами, відкриється сайт мерчантами к браузері. На цій сторінці можна виконати пошук мерчантів та сортувати їх про категоріям. Список з мерчантами підгружається при здійсненні скролу. Для виконання пошуку потрібно натиснути на ікону з лупою і верхній частині екрана, ввести в тестове але назву або частину назви мерчантами, якого потрібно знайти та ще раз натиснути на іконку з лупою. Для перегляду мерчантів по категоріям, достатньо натиснути на категорію у верхній частині екрану (рис.

4.4). Список з категоріями можно прогорнути у правий бік для перегляду більшої



кількості категорій.

Рисунок 4.4 - Функціонал та інтерфейс сторінки з мерчантами

Якщо натиснути на іконку бокового меню в лівому верхньому куті, можно побачити меню та перейти на інші екрани (рис. 4.5). Для переходу на сторінку зміни інформації профілю, потрібно натиснути на фото профілю (або іконку для фото), надпис профіль, або іконку з олівцем біля цього надпису.

На сторінці зміни даних профілю можна змінити наступні дані (рис. 4.6):

- Ім’я та фамілію користувача

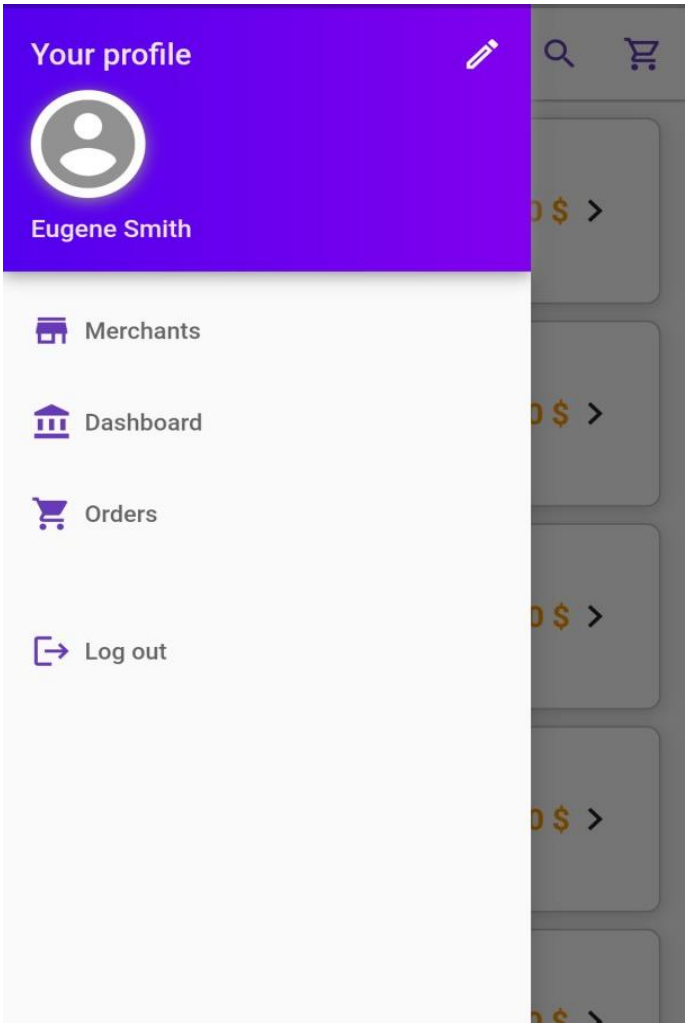


Рисунок 4.5 - Інтерфейс бокового меню

- Фото користувача (при натисканні на олівець біля фотографії). При цьому додаток запропонує джерело, з якого буде обрано фото (галерея або фотокамера) (рис. 4.7).
- Електрону адресу
- Номер телефону
- Дати народженн. Для коректної зміни дати народження, висвічується спеціальний інтерфейс для вибору дати.

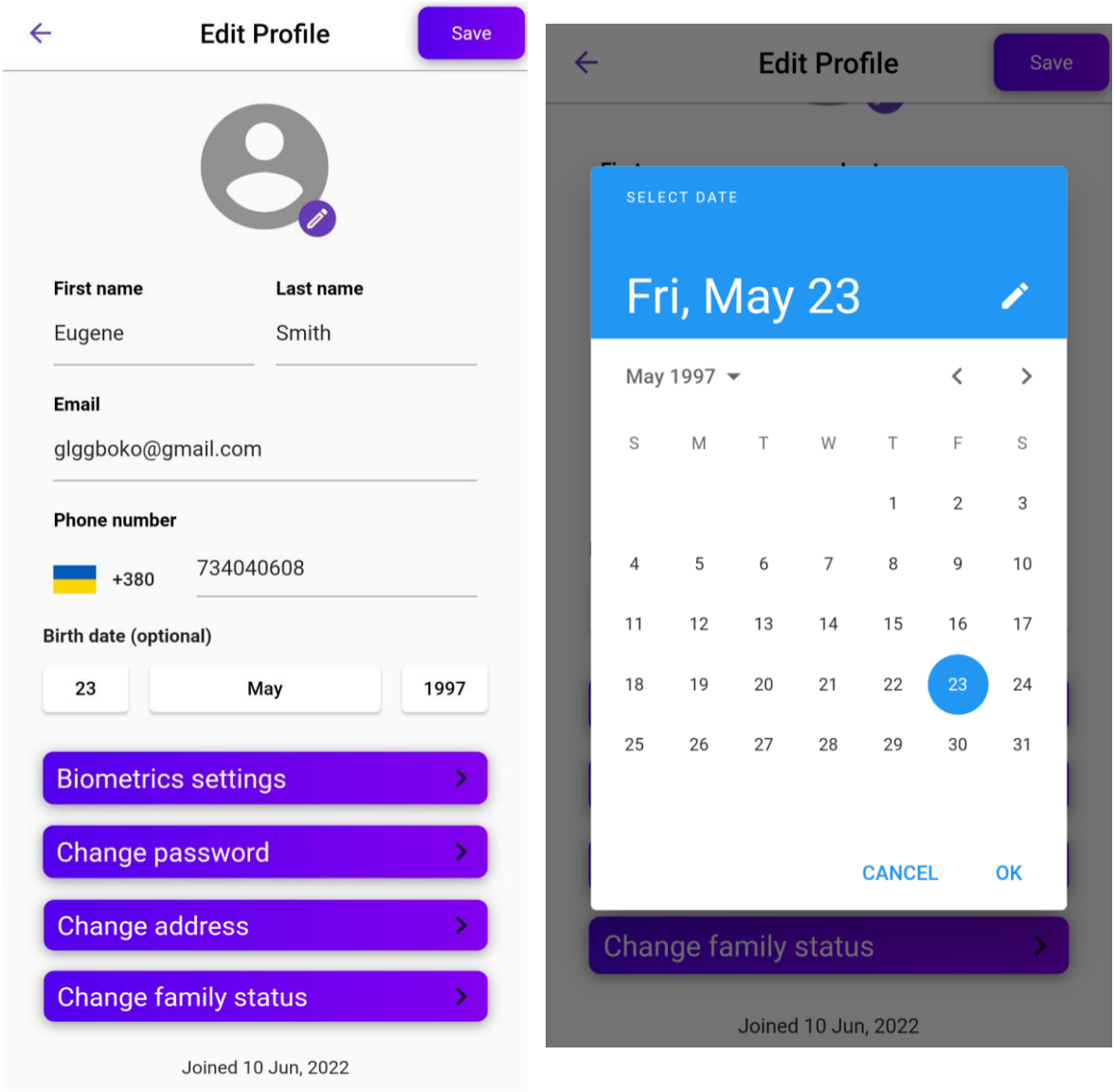


Рисунок 4.6 –Інтерфейс сторінки редагування профілю користувача

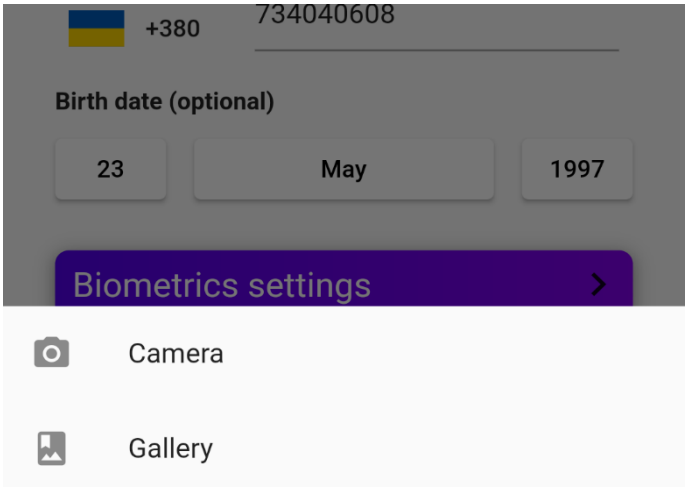


Рисунок 4.7 – Функціонал зміни фото профілю користувача

Коли користувач вводить нові дані та не зберігаючи їх, намагається вийти на попередню сторінку, він побачить діалог (рис. 4.8). За допомогою нього користувач може вибрати, чи хоче він зберегти поточні зміни.

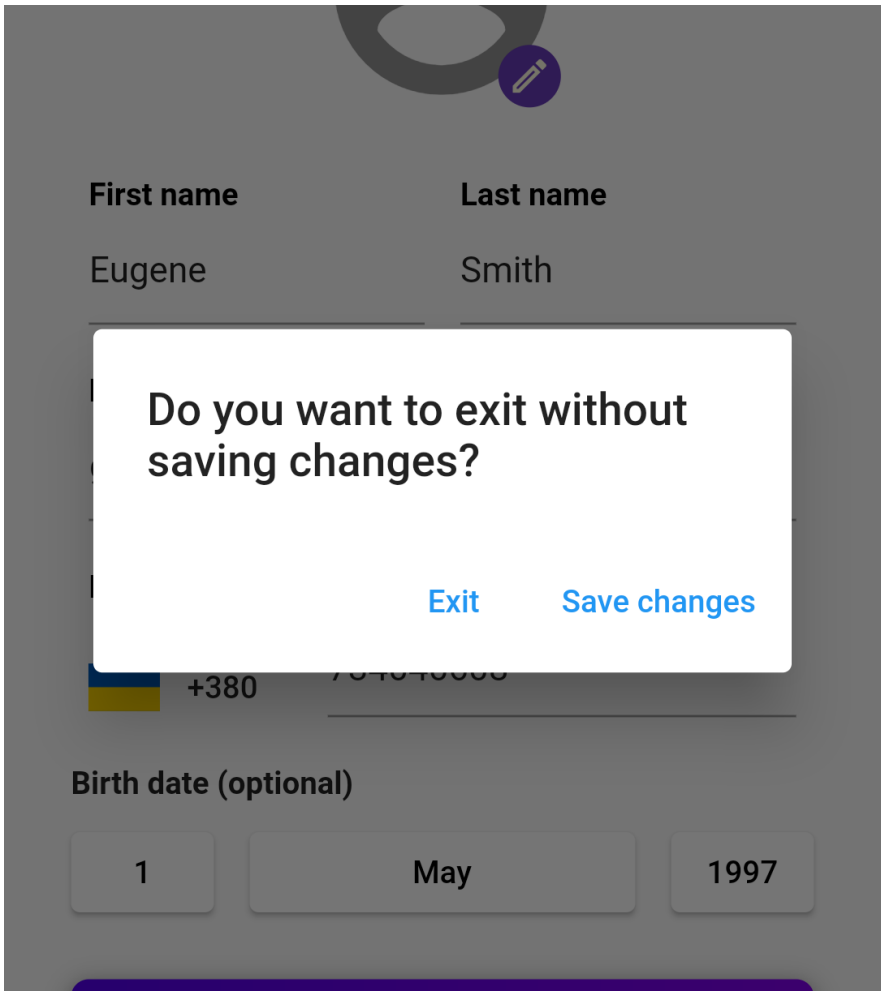


Рисунок 4.8 – Функціонування та інтерфейс діалогу про незбережені зміни

Також сторінка містить перехід на:

- Функціонал зміни паролю (рис. 4.9). На ній потрібно ввести поточний пароль та два рази ввести новий пароль. Потім натиснути кнопку збереження.

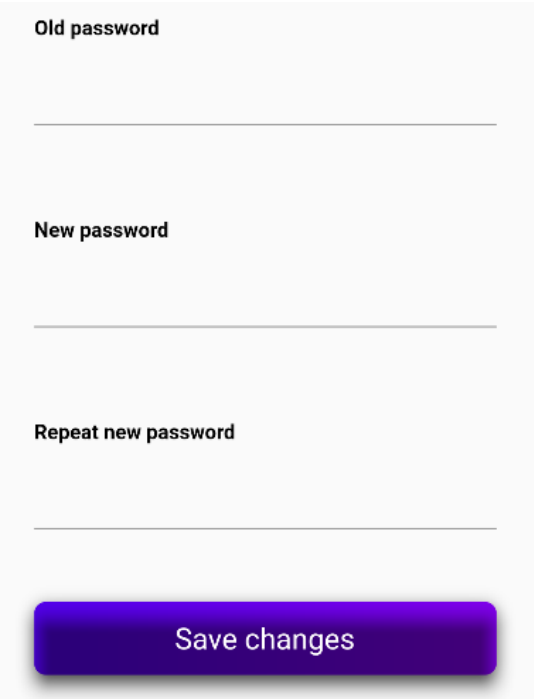
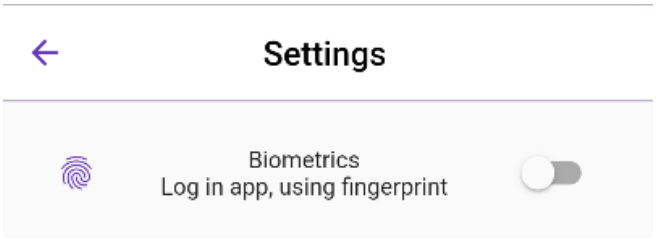


Рисунок 4.9 – Інтерфейс для зміни паролю користувача

- Функціонал зміни налаштувань щодо біометричного входу в акаунт (рис. 4.10). Коли перемикач увімкнено, користувач може увійти в систему наступного разу за допомогою біометричних даних.

Рисунок 4.10 - Інтерфейс зміни налаштувань використання біометричних даних



- Функціонал для зміни сімейного статусу (рис. 4.11). Тут можна змінити кількість дітей та сімейний статус
- Функціонал для зміни адреси (рис. 4.12). На рисунку видно, які саме поля можна змінити: поштовий код, область (або штат), місто, вулицю.

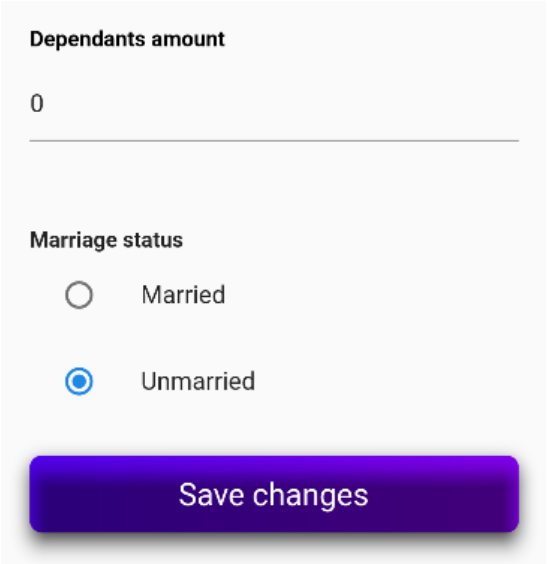


Рисунок 4.11 - Інтерфейс зміни полів щодо сімейного статутсу

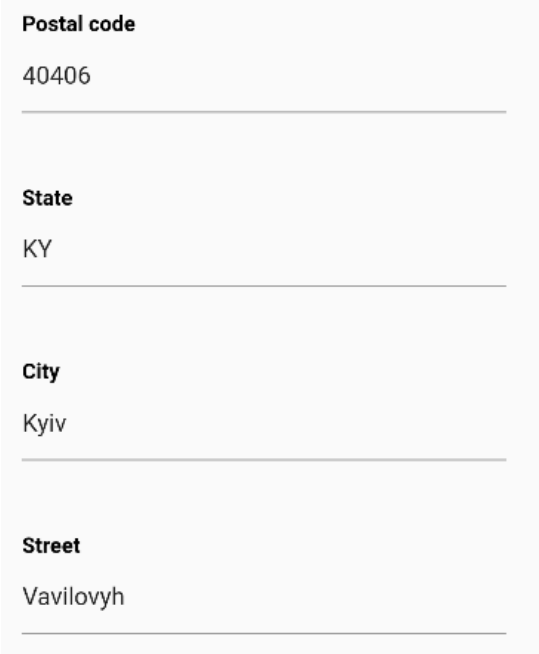


Рисунок 4.12 – Поля та функціонал для зміни адреси

Данні про адресу, ім’я та номер телефону передаються інтернет-магазину для автоматичного заповнення полів під час оформлення замовлення. Дані про дату народження та сімейний стан також передаються мерчанту для схвалення заявки на замовлення. Адже не кожну виплату можно довірити будь-якій людині.

Також на сторінці зміни даних профілю внизу написано, коли профіль було створено.

На сторінці замовлень доступні для перегляду усі замовлення, які біло зроблено користувачем за допомогою сервісу (рис. 4.13). Для пошуку

замовлення по назві, потрібно скористатись лупою (як і на сторінці з мерчантами). Для відображення замовлень у кошику потрібно натиснути на іконку кошику у правому верхньому углу.

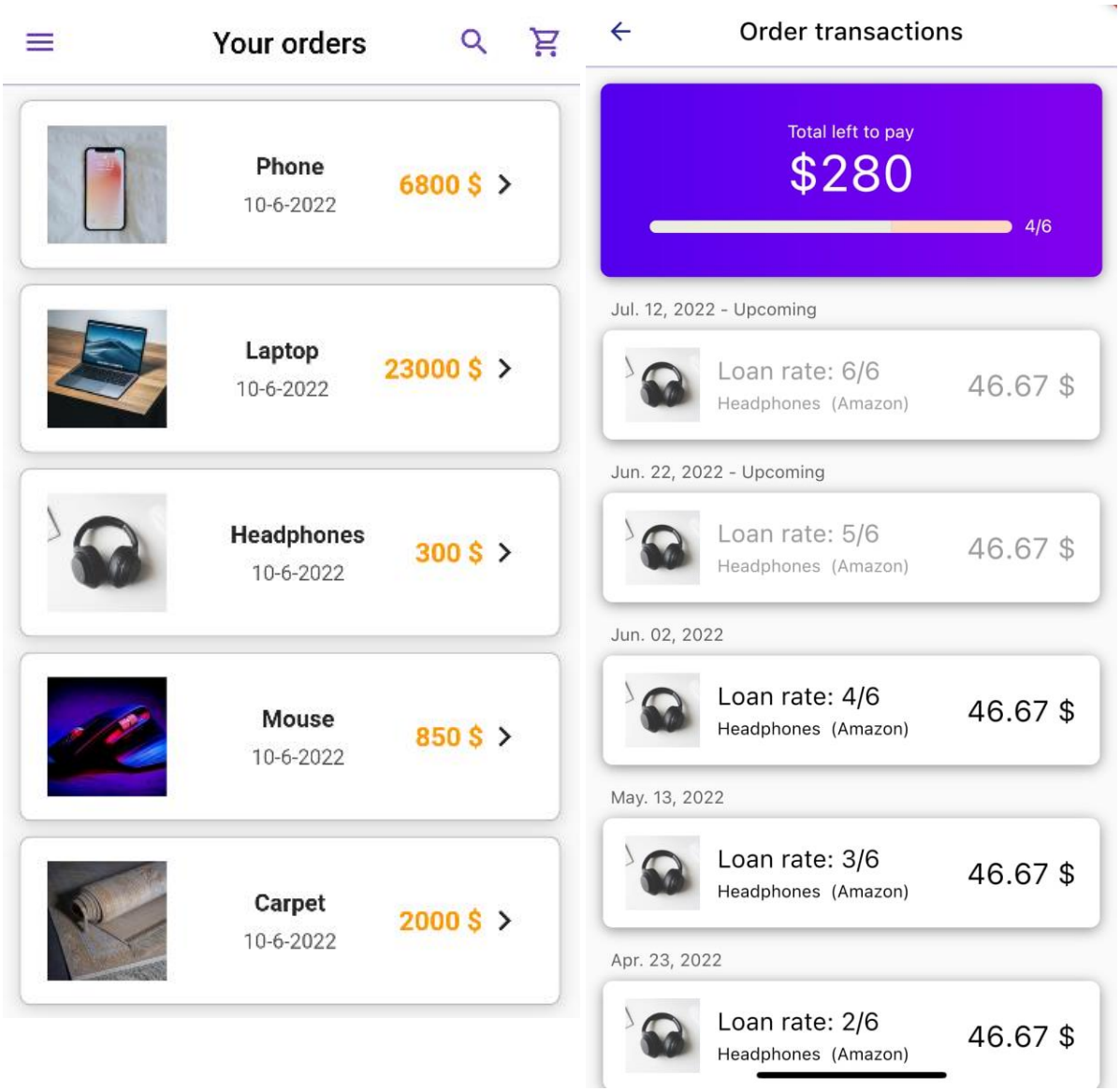


Рисунок 4.13 – Інтерфейс сторінки з замовленнями та сторінки з детальною інформацією про замовлення

Для перегляду деталей щодо замовлення та транзакцій щодо нього, потрібно натиснути на замовлення. Користувачу висвітиться сторінка з:

- Сумою замовлення
- Інформацією про те, скільки платежів залишилось для погашення цього заказу. Інформація представлена лінією під сумою заказу. Більш світлий

колір представляє погашений процент суми. Справа від цієї строки написано скільки платежів зі скольки погашено.

- Список транзакцій за цим замовленням (рис. 4.13). Над ними висвічується дата, до якого числа місяця та року ця транзакція повина відбутися. Транзакції, що ще не пройшли, підсвічено сірим кольором.

На сторінці Dashboard показано поточний рахунок користувача, суму, яка вже була використана на проведення транзакцій та суму, на яку транзакції вже заплановані (рис. 4.14). Під цими сумами можна побвачити список транзакцій, що вже пройшли з рахунком користувача. Ці транзакції завантажуються по мірі гортання сторінки. Транзакції груповані по датам.

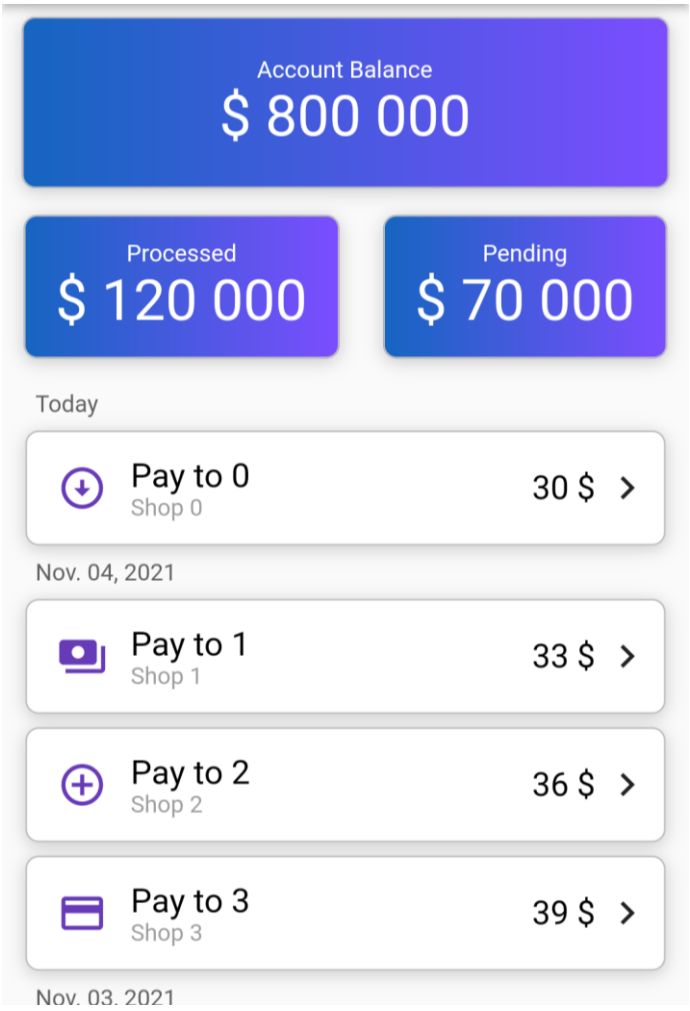


Рисунок 4.14 – Інтерфейс сторінки з усіма транзакціями користувача

Мобільний додаток також включає функціонал надсилання повідомлень. Повідомлення надходять про заплановані транзакції у день транзакції. Також,

якщо на рахунку у користувача не вистачає коштів на оплату транзакції, йому прийде повідомлення за тиждень до часу транзакції.

4.2 Інструкція для користування веб додатком для користувачів

Як і в мобільному додатку, на веб платформі користувач спочатку попадає на екрани авторизації (рис. 4.15 – 4.17). Вони мають аналогічний до сторінок на мбільному додатку функціонал.

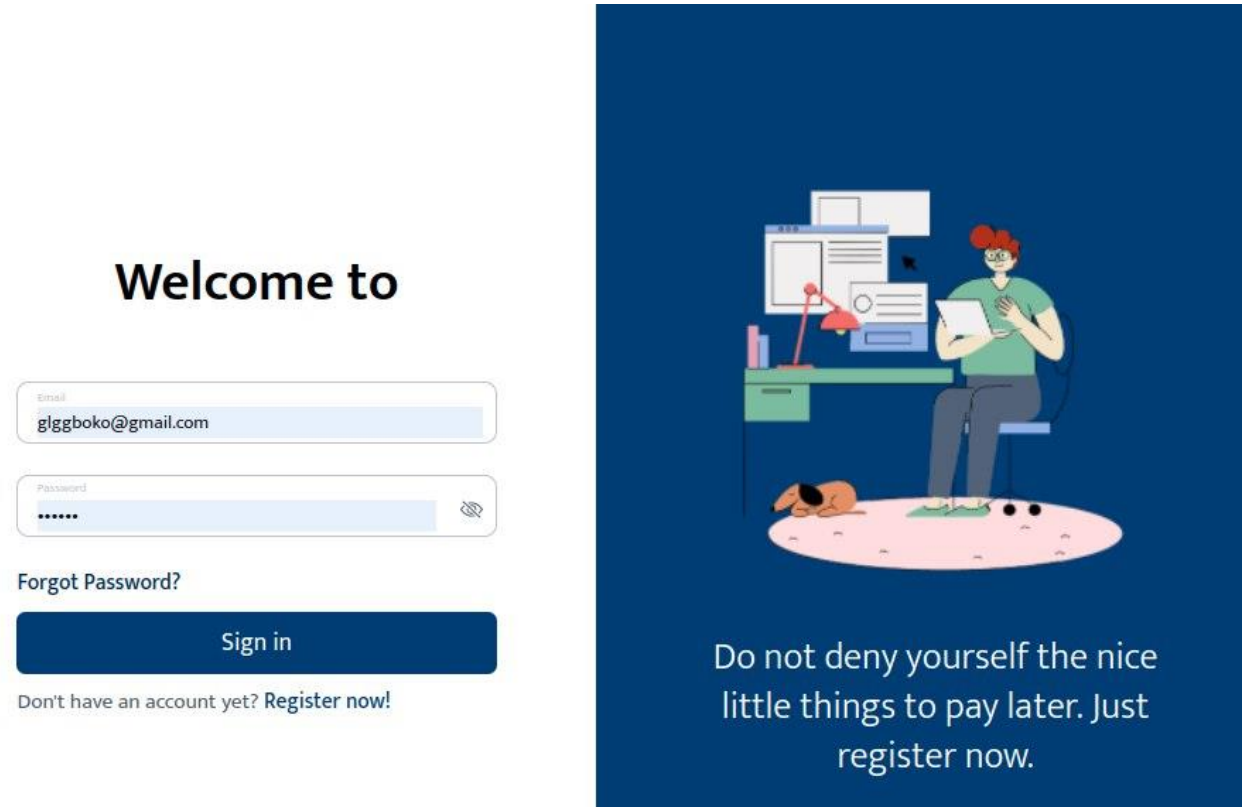


Рисунок 4.15 Інтерфейс сторінки логіну на веб сервісі

Після входу в обліковий запис, користувачу доступний внутрішній функціонал сервісу, а саме:

- Сторінка для редагування даних профіля (рис. 4.18). Вибір фото профіля користувача відбувається з внутрішніх файлів пристрою та нове фото доступно для редагування одразу після вибору (рис. 4.19). Це є досить зручно за статистикою роботи з фотографіями.

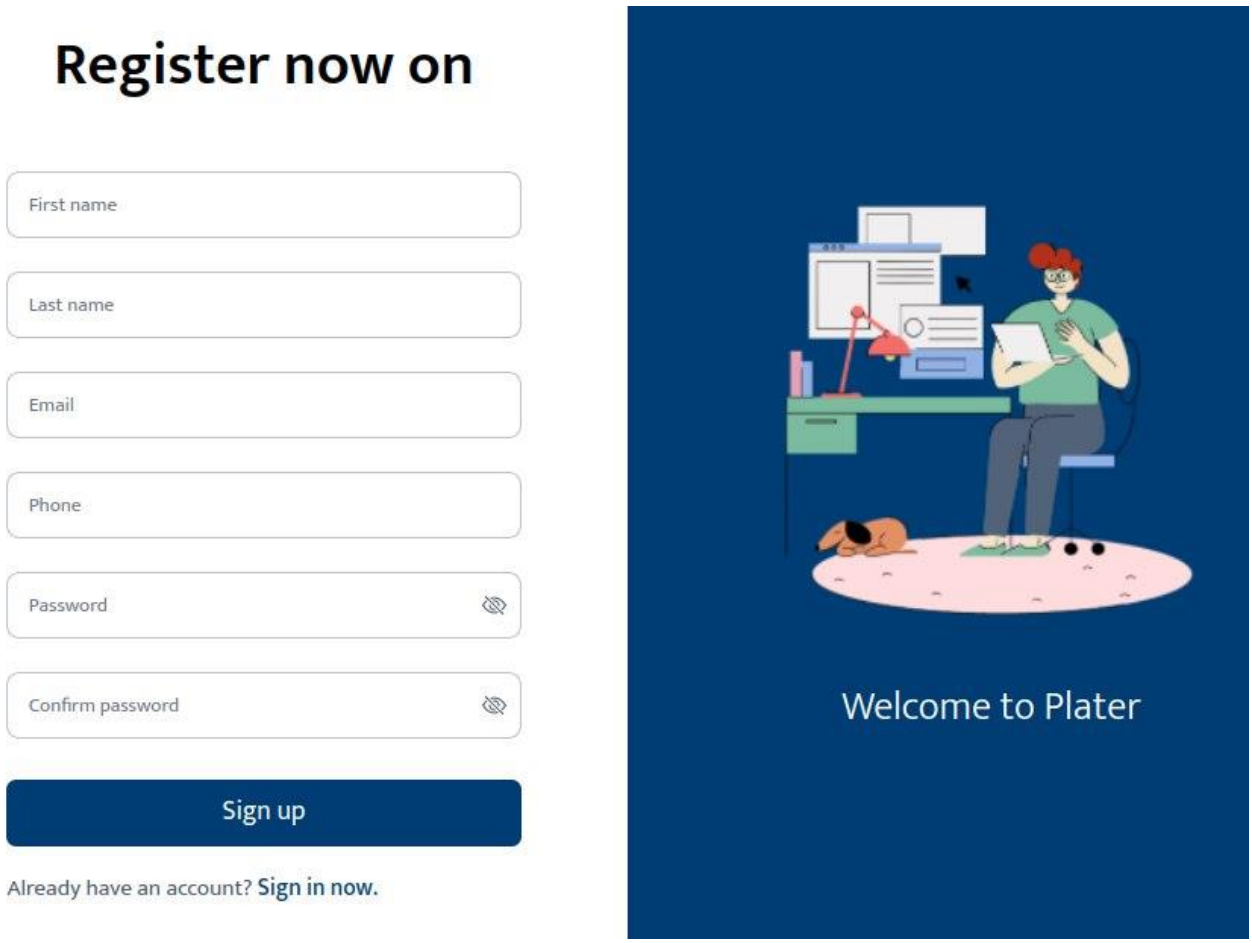


Рисунок 4.16 Інтерфейс сторінки реєстрації на веб сервісі

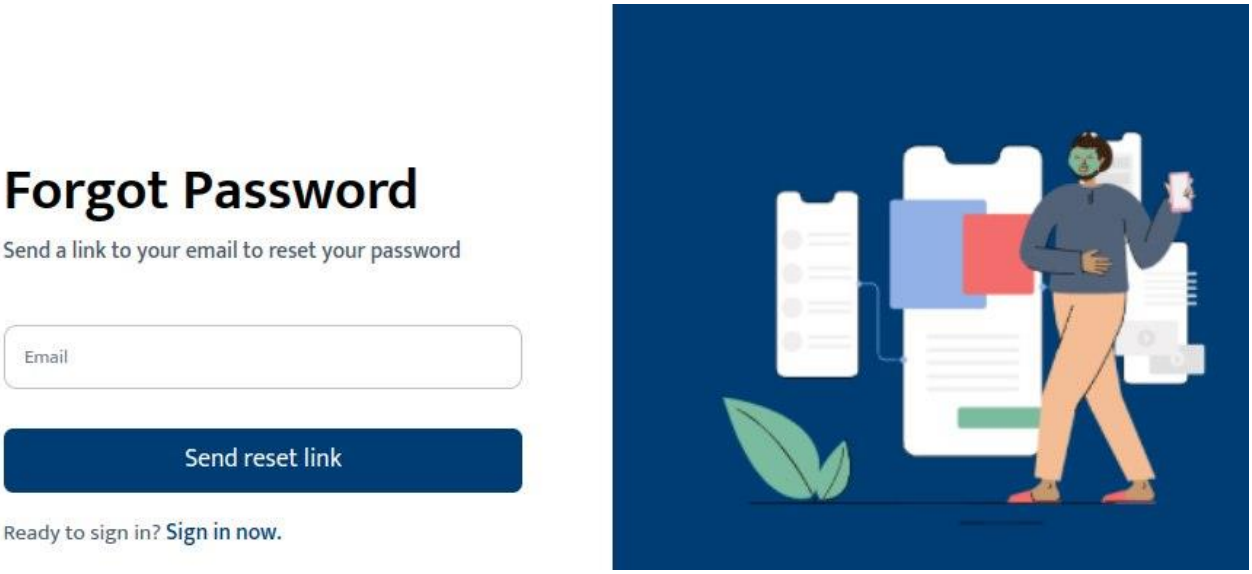


Рисунок 4.17 Інтерфейс сторінки відновлення паролю на веб сервісі

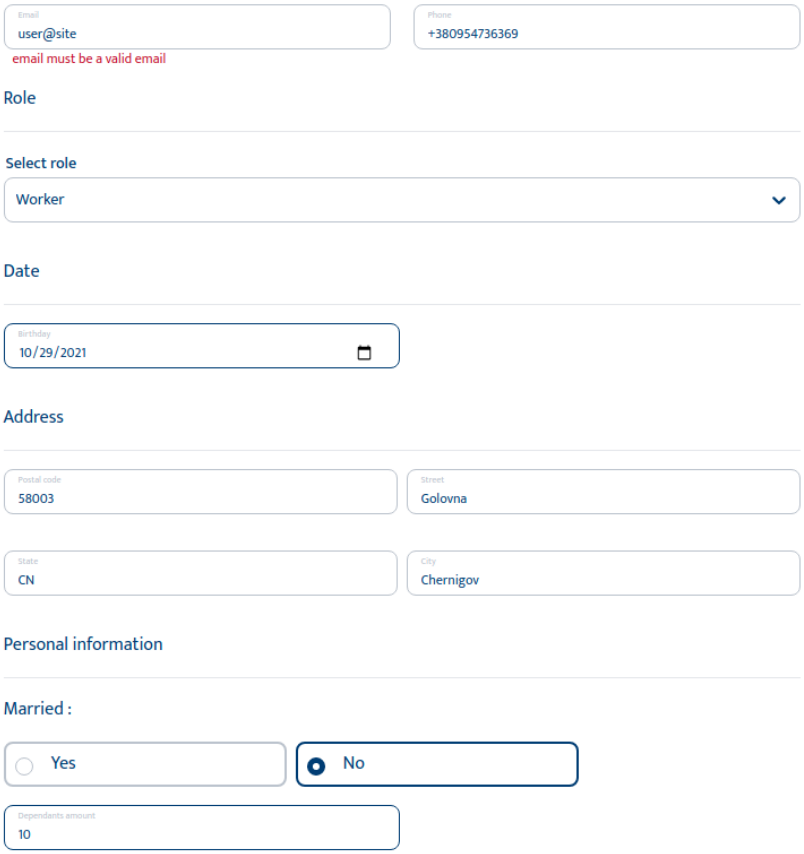


Рисунок 4.18 Сторінка зміни даних профіля та її функціонал

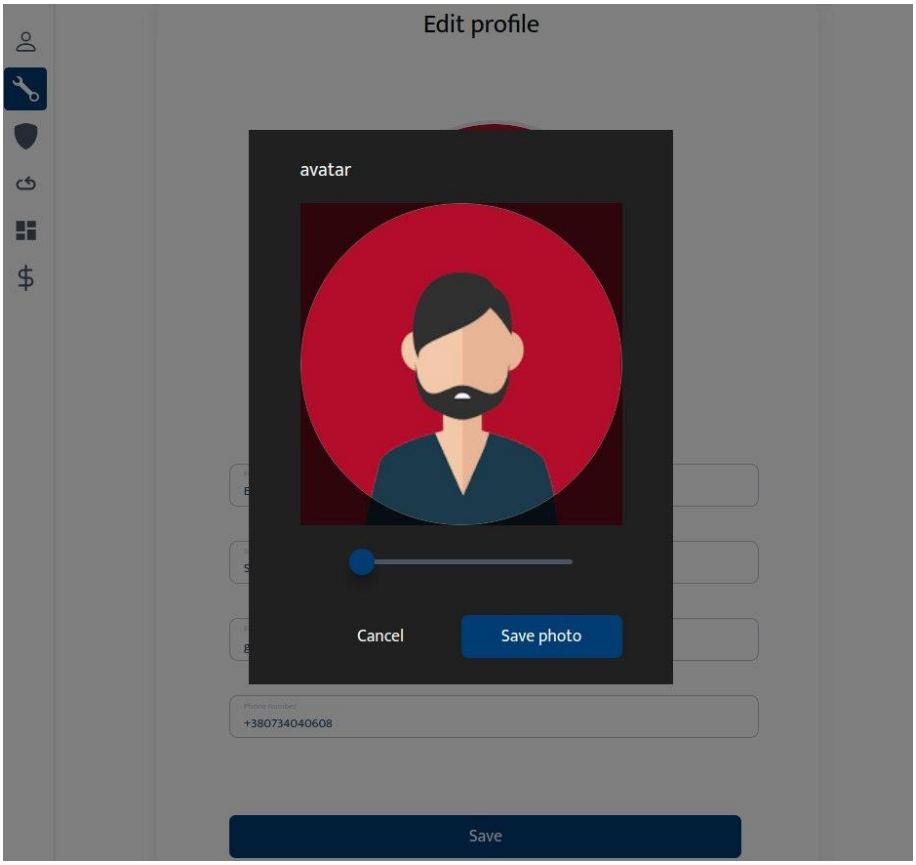


Рисунок 4.19 – Функціонал зміни фото профіля користувача у веб додатку

- Сторінка зміни паролю (рис. 4.20). На ній, як і в мобільній версії, треба ввести поточний пароль та два рази новий пароль та натиснути кнопку збереження. На пароль, як і усюди, діє валідація задля усунення встановлення занадто простих паролів. Тому, якщо буде вказано занадто простий пароль, сторінка про це повідомить і пароль користувача не буде змінено.

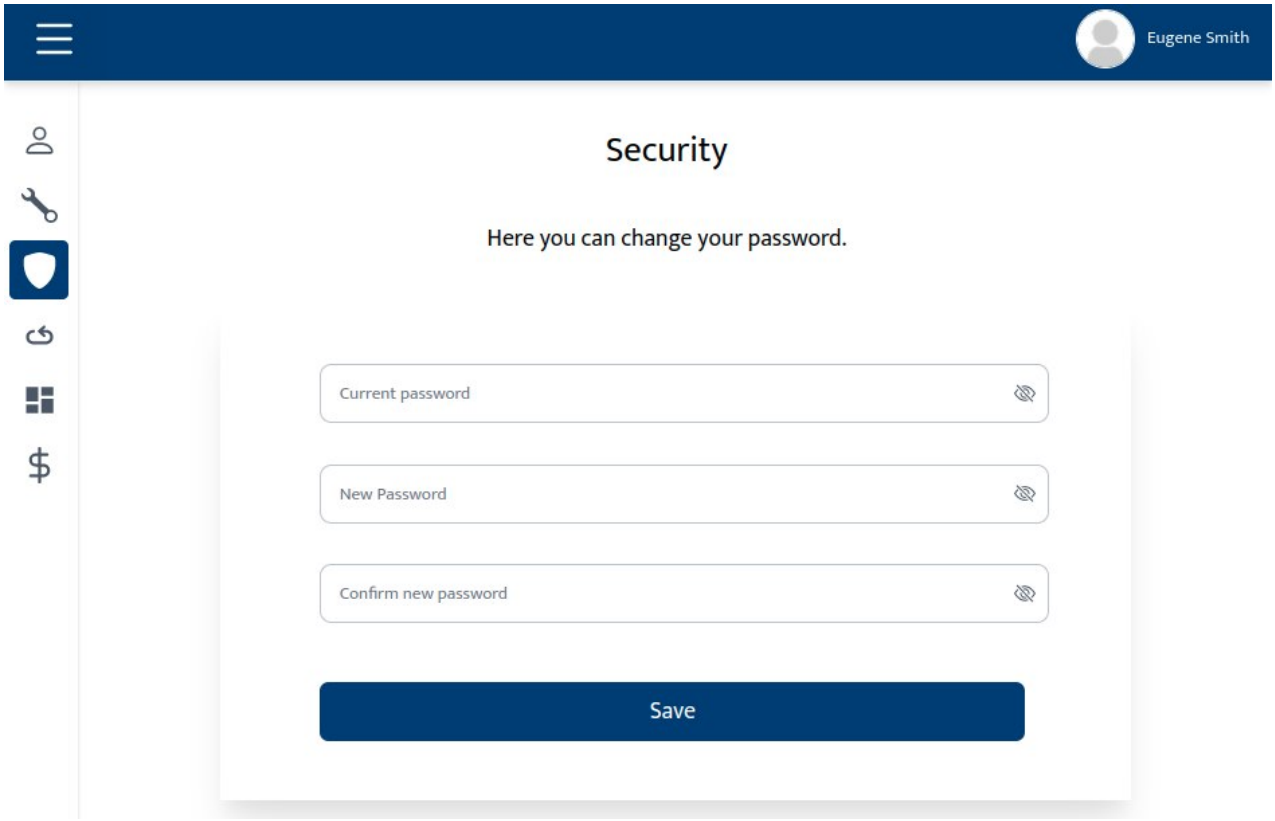


Рисунок 4.20 - Інтерфейс сторінки для зміни паролю облікового запису на веб сервісі

- Сторінка для перегляду транзакцій користувача та інформації щодо його рахунків (рис. 4.21). Тут можна переглянути все те, що й у мобільному додатку на відповідній сторінці. Поточний баланс користувача, запланована на витрати сума, а також сума вже витрачених коштів показана у верхній частині сторінки. У нижній показано список транзакцій. Статус транзакцій тут позначено у відповідній колонці. А також, на цій сторінці можна додати гроші на рвхунок та заплатити за транзакцію завчасно.

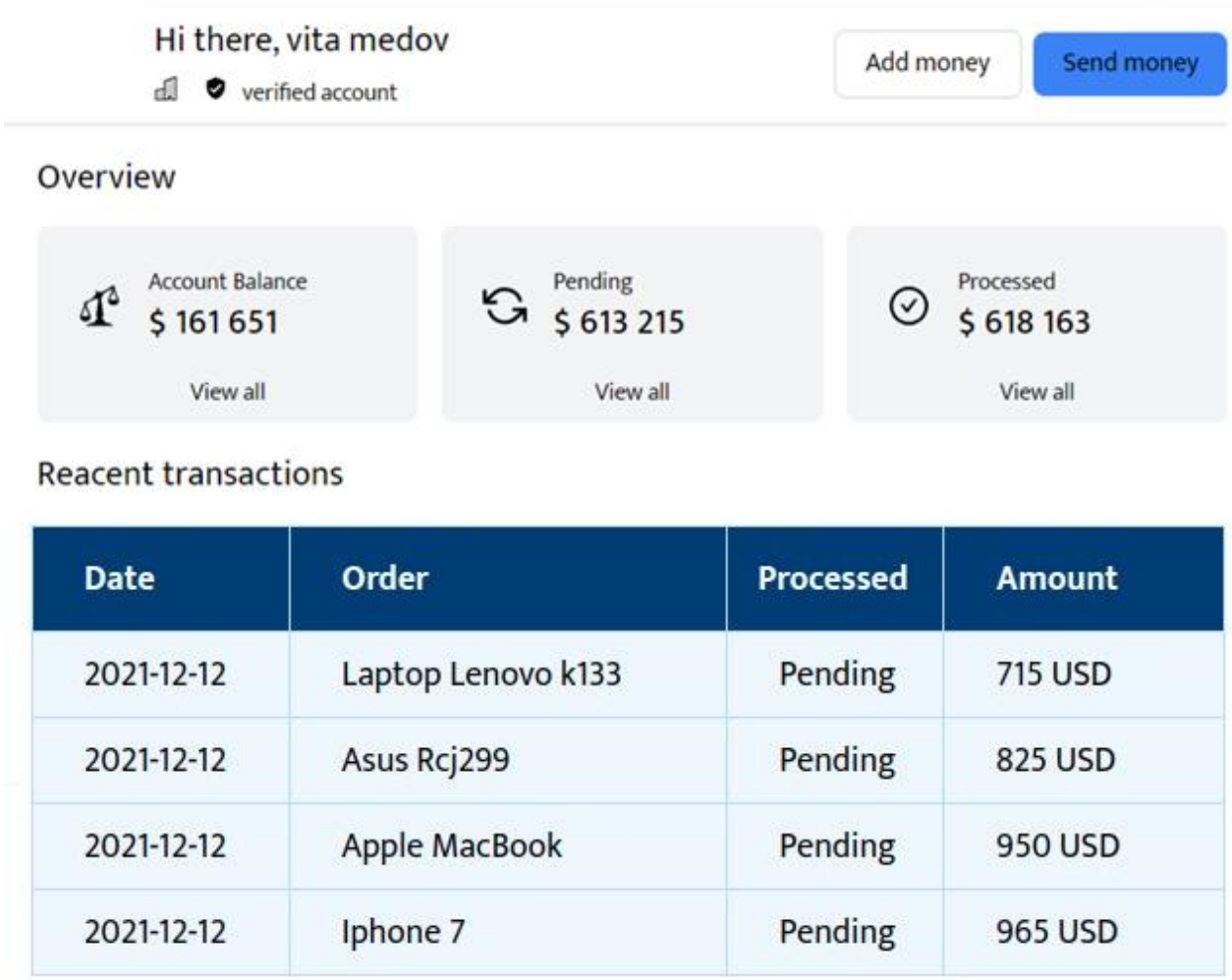


Рисунок 4.21 – Інтерфейс сторінки з транзакціями та інформацією про рахунки на веб-платформі для користувача

- Сторінка з мерчантами, в яких можна зробити замовлення за допомогою сервісу. При натисканні на фото або назву, відбувається перехід на домен мерчанта. Ц майбутньому, з мерчантами буде обговорюватись, які саме фото будуть їх презентувати та на яку саме сторінку буде відбуватись перенапрвлення. Можливо додавання розділів «знижки» або «популярне» на цю сторінку.

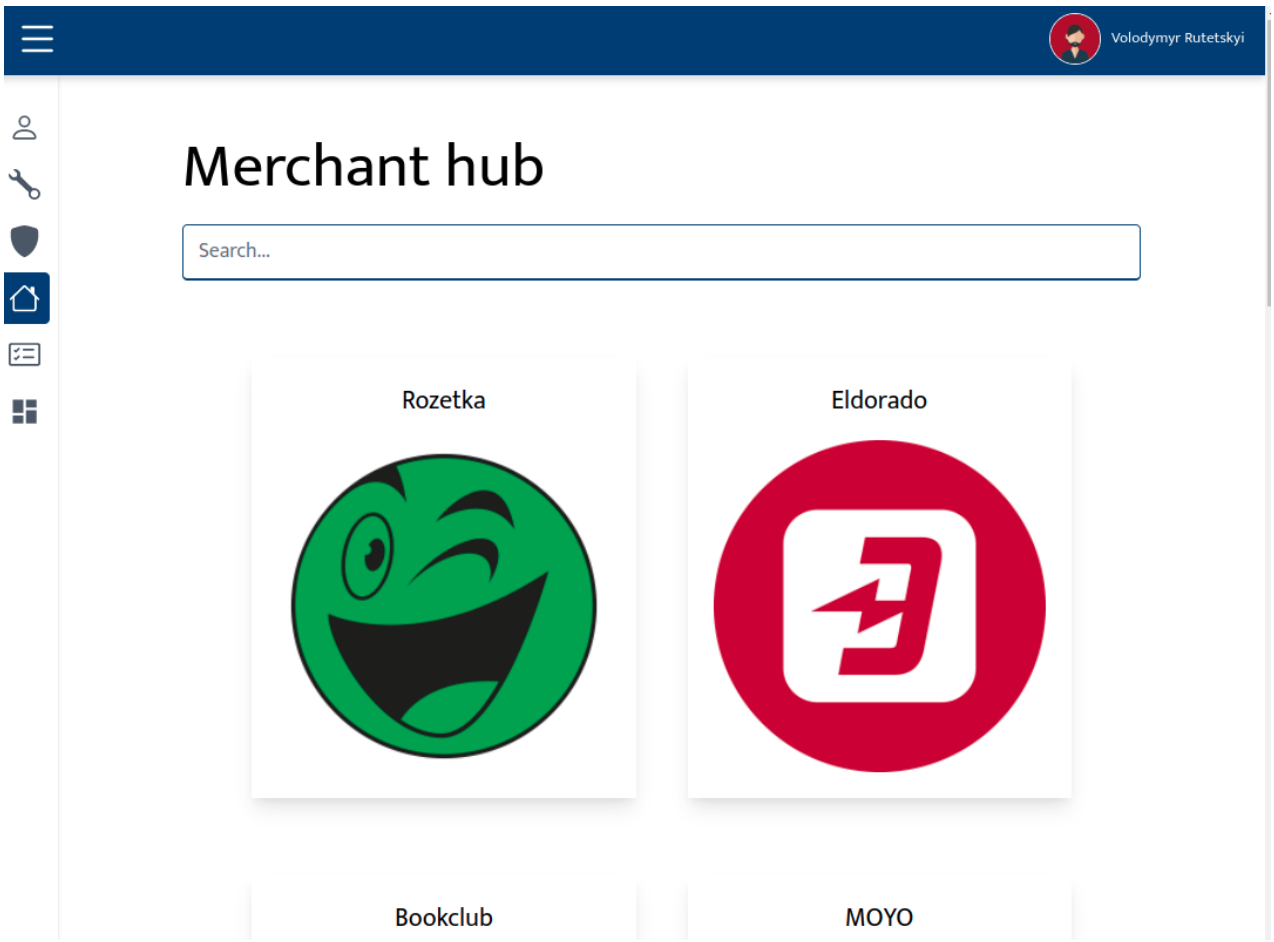


Рисунок 4.22 – Інтерфейс сторінки з мерчантами в веб додатку для клієнта

Для користувачів створено ще один елемент функціоналу з інтерфейсом. Його планується для розміщення на сайті мерчанта. Це – кнопка «оплатити за допомогою цього сервісу (рис. 4.23). Після натискання на неї користувача перенаправляє на сайти сервісу та йому пропонується вибір плану оплати товару. Ці плани будуть обговорені в особистому порядку з кожним мерчантом.

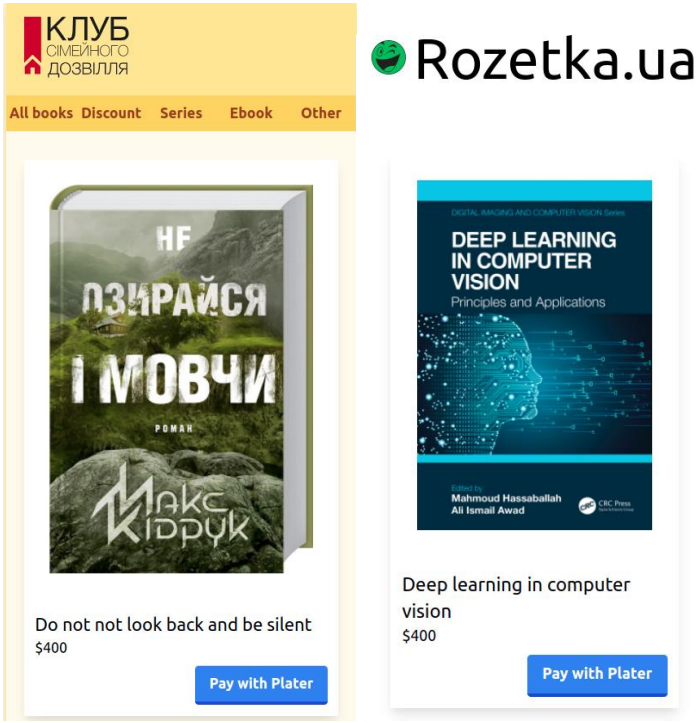


Рисунок 4.23 – Приклади інтерфейсу для магазинів мерчанту, що містять спеціальну кнопку для користування сервісом

4.3 Інструкція для користування веб додатком для мерчантів

Для мерчанту, що вирішив зареэструватися на сервісі спочатку доступні такі самі сторінки авторизації, як і для звичайного користувача (рис. 4.15 – 4.17). Після реєстрації він повинен зв’язатись з представниками сервісу для переведення його акаунту в статус мерчанта.

Після переведення стутусу акаунту мерчанту, йому доступні сторінки зміни профілю, сторінка безпеки та ще декілька інших:

- Сторінка зі статистикою мерчанта (рис. 4.24). Тут відображена інформація щодо заказів товарів мерчанту, що проведені за допомогою сервісу: загальна сума, що витратили користувачі на закази в магазині мерчанту, середня сума замовлення, що робились користувачами сервісу, скільки загалом користувачі сервісу зробили заказів в магазині мерчанту та відсоток вже оплачаних покупок. Також на цій сторінці представлена статистика залежністю

суми за визначений період часу. Періоди можна змінювати в випадаючому меню над графіком.

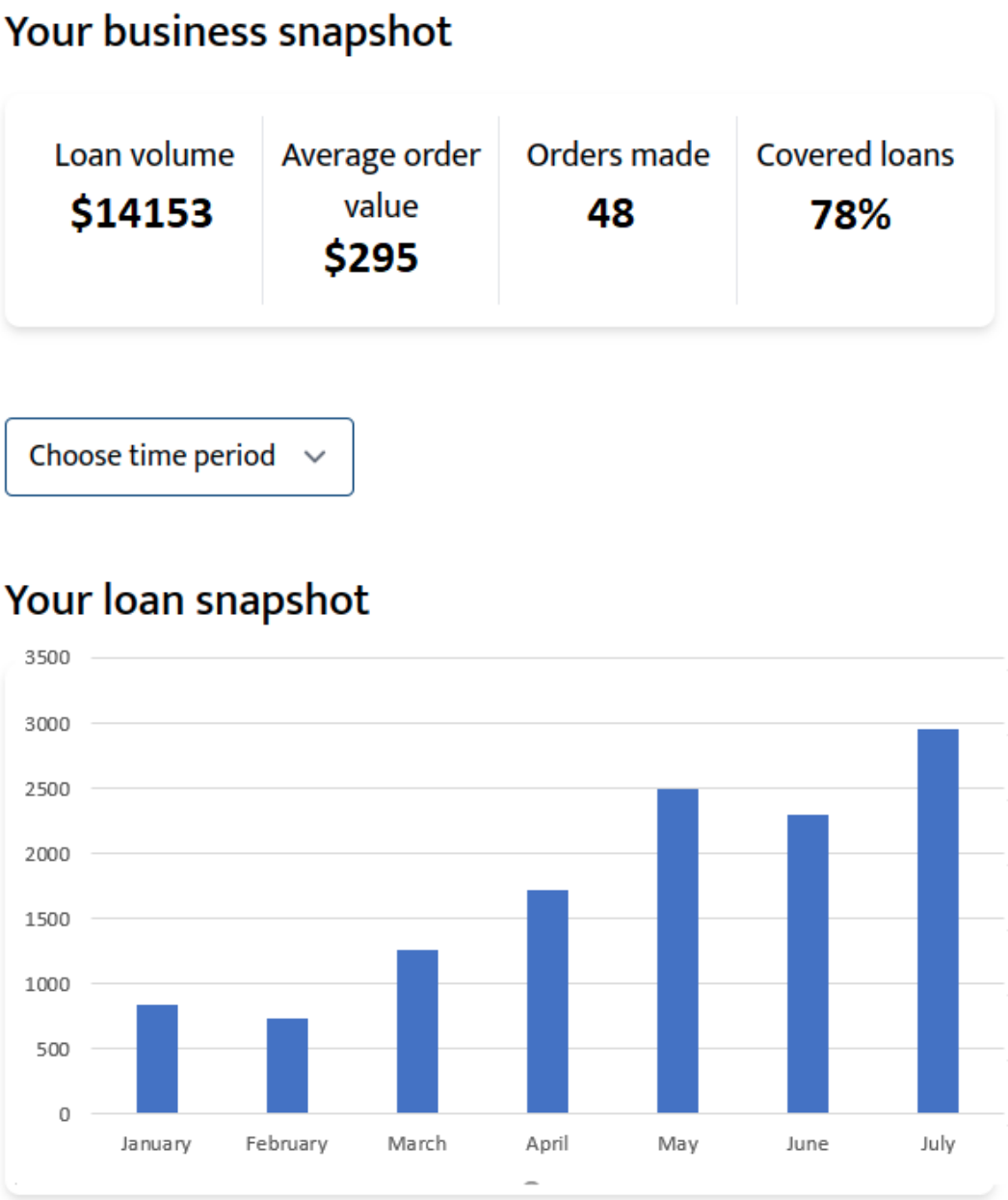


Рисунок 4.24 – Інтерфейс сторінки зі статистикою для мерчанта

- Сторінка з транзакціями, що були проведені з мерчантом (рис. 2.25). На ній показано кожну транзакцію, що була проведена між будь-яким клієнтом та мерчантом. В колонці статусу показано, пройшла транзакція вже, чи це є запланована транзакція.

Рисунок 4.25 – Інтерфейс сторінки для мерчанту з транзакціями

Для мерчанту зв’язок з представниками сервісу потрібен також і для налаштування сервісу (планування реклами та налаштування адресів для посилань на сайт) та інтерфейсу інтернет-магазину для співпраці. Також, таким чином буде здіснюватись обговорення реклами мерчанта на сайті сервісу та плани оплати товарів сервісу. Тож цей зв’язок є необхідним та обов’язковим.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі було наведено інструкцію з еористуання сервісом. Сервіс передбачає три сценарії використання:

- Використання клієнтом мобільного додтаку
- Використання клаєнтом веб-додатку
- Використання мерчантом веб додатку

Наведено скріншоти, що свідчить про функціонал сервісу та його компонентів та ілюструють елементи інтерфейсу сервісу. Детально розказано про те, для чого існують ці компоненти, яку інформацію вони відображають та як користуватися функціоналом, що представлено на них. Описано сценарій використання сервісу.

Описано схему, за допомогою якої користувач сервісу може набути прав марчанта та способи здійснення оплати товарів через червіс

Під час проведення тестування, виявлено помилки, та слабкі місця програми. Ці неддоліки було усунуто.

ВИСНОВКИ

Проаналізовано існуючі севіси, що дають можливість користувачам оплатити товари через деякий час після здійснення покупки. Проведено аналіз таких сервесів, вивлено їх переваги та недоліки, їх бізнес-стратегії та стратегії набуття клієнтів. На основі цього аналізу обрано бізнес-стратегію, а також платформи функціонування та характеристики продукту розробки. До програмного продукту висунуто умоги та описано його функціонал.

Обрано розробити проєкт на двох платформах: на мобільній та на веб. На веб платформі буде функціонувати як сервісна, так і серверна частина. Мобільна платформа теж буде виступати в якості сервісної частини.

Для розробки сервісу порівняно технології, за допомогою яких можливо розробити сервіс, що відповідає висунутим умогам. Аналіз дав змогу визначитися технологіями, які дозволяють оптимально реалізувати потрібний функціонал. Розроблений продукт написан на мові dart та typescript. В якості системи керування базами даних було обрано PostgreSQL.

Розроблено сервіс, що відповідає умогам, поставленим до нього. Описано процес розробки сервісу, його структурні елементи та як вони взаємодіють.

Показано користувацькі інтерфейсси. Для користування сервісу написано інструкцію. Інструкція містить скріншоти та опис процесу використання сервісу як для мерчанта, так і для звичайного клієнта. Наведені скріншоти, свідчить про функціонал та ілюструють елементи інтерфейсу сервісу.

Реалізовано усі зазначені в меті роботи функції для користувачів та мерчантів. При розробці застосовано патерни програмування, що забезпечать більш швидку роботу компонентів та кращий досвід користування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hanlon A. Digital Marketing: Strategic Planning & Integration / Annmarie Hanlon., 2021. – 480 с. – (SAGE).
2. Mention A. Research-Technology Management / Anne-Laure Mention. // The Future of Fintech. – 2019. – №62. – С. 59–63.
3. Fisher C. Developments in the Buy Now, Pay Later Market [Електронний ресурс] / C. Fisher, C. Holland, T. West // Reserve bank of Australia. – 2021. – Режим доступу до ресурсу: https://www.rba.gov.au/publications/bulletin/2021/mar/developments-in-the-buy-now-pay-later-market.html.
4. Xing Y. Australian Online BNPL Services Research: Building Gain Value Model of Individual Credit Background [Електронний ресурс] / Y. Xing, H. Chen, X. Zhuang // 2019 International Conference on Information Technology and Computer Communications. – 2019. – Режим доступу до ресурсу: https://dl.acm.org/doi/abs/10.1145/3355402.3355410.
5. Lake R. What You Need to Know About Afterpay [Електронний ресурс] / Rebecca Lake // Investopedia. – 2022. – Режим доступу до ресурсу: https://www.investopedia.com/how-afterpay-works-5184682.
6. How does Afterpay work? [Електронний ресурс] // Afterpay Limited. – 2021. – Режим доступу до ресурсу: https://help.afterpay.com/hc/en-us/articles/217425866-How-does-Afterpay-work-.
7. How much can I spend with Afterpay? [Електронний ресурс] // Afterpay Limited. – 2022. – Режим доступу до ресурсу: https://help.afterpay.com/hc/en-us/articles/218320803-How-much-can-I-spend-with-Afterpay-.
8. Is there a cost to using Afterpay? [Електронний ресурс] // Afterpay Limited. – 2022. – Режим доступу до ресурсу: https://help.afterpay.com/hc/en-us/articles/218320423-Is-there-a-cost-to-using-Afterpay-.

9. Do the spending limits encourage me to spend more than I can afford? [Електронний ресурс] // Afterpay Limited. – 2021. – Режим доступу до ресурсу: https://help.afterpay.com/hc/en-au/articles/900004386866-Do-the-spending-limits-encourage-me-to-spend-more-than-I-can-afford-.
10. What is Klarna [Електронний ресурс] // Klarna Holding AB. – 2021. – Режим доступу до ресурсу: https://www.klarna.com/us/what-is-klarna.
11. Huffman L. Afterpay Review [Електронний ресурс] / Lee Huffman // Investopedia. – 2022. – Режим доступу до ресурсу: https://www.investopedia.com/afterpay-review-5189707.
12. Klarna announces $460M equity raise to further support massive US growth [Електронний ресурс] // Klarna Holding AB. – 2019. – Режим доступу до ресурсу: https://www.klarna.com/international/press/klarna-announces-460m-equity-raise-to-further-support-massive-us-growth/.
13. Annual report 2020, 2020. – 97 с. – (Klarna Holding & AB) – Режим доступу до ресурсу: https://www.klarna.com/assets/2021/03/30041745/Annual-Report-Klarna-Holding-AB-2020-210329.pdf.
14. Our locations [Електронний ресурс] // Klarna Holding AB. – 2021. – Режим доступу до ресурсу: https://www.klarna.com/careers/our-locations/columbus/.
15. Cart Abandonment Stats [Електронний ресурс] // Baymard Institute. – 2021. – Режим доступу до ресурсу: https://baymard.com/lists/cart-abandonment-rate.
16. WebBank Klarna Credit Account Agreement [Електронний ресурс] // Klarna Holding AB. – 2022. – Режим доступу до ресурсу: https://cdn.klarna.com/1.0/shared/content/legal/terms/0/en_us/account-terms.
17. Boost your business with Klarna [Електронний ресурс] // Klarna Holding AB. – 2022. – Режим доступу до ресурсу: https://www.klarna.com/us/business/.
18. Bell A. How Klarna Lets You Pay Later With No Interest [Електронний ресурс] / Amy Bell // Investopedia. – 2022. – Режим доступу до ресурсу:

https://www.investopedia.com/articles/personal-finance/121316/how-klarna-lets-you-pay-later-no-interest.asp.

19. Lunn E. Klarna: 'buy now, pay later' system that is seducing millennials [Електронний ресурс] / Emma Lunn // The Guardians. - 2018. – Режим доступу до ресурсу: https://www.theguardian.com/money/2018/nov/17/klarna-buy-now-pay-later-system-that-is-seducing-millennials.
20. Alcazar J. The Rise of Buy Now, Pay Later: Bank and Payment Network Perspectives and Regulatory Considerations / J. Alcazar, T. Bradford., 2021. – 6 с. – (kcFED). – (Payments System Research Briefing).
21. KAGAN J. Financial Technology – Fintech / J. KAGAN, E. ESTEVEZ., 2020. – 12 с. – (Investopedia). – (FintechDefinition) – Режим доступу до ресурсу: https://www.cpaireland.ie/CPAIreland/media/Education-Training/Syllabus%20Articles/Financial-Technology-%E2%80%93-FintechDefinition.pdf.
22. Huffman L. Affirm Review [Електронний ресурс] / Lee Huffman // Investopedia. – 2022. – Режим доступу до ресурсу: https://www.investopedia.com/affirm-review-5189381.
23. How it works? [Електронний ресурс] // Affirm Holdings. – 2022. – Режим доступу до ресурсу: https://www.affirm.com/how-it-works.
24. Approved and declined loans [Електронний ресурс] // Affirm Holdings. – 2022. – Режим доступу до ресурсу: https://helpcenter.affirm.com/s/article/loan-approvals.
25. Order issues [Електронний ресурс] // Affirm Holdings. – 2022. – Режим доступу до ресурсу: https://helpcenter.affirm.com/s/article/where-is-my-order.
26. Goel S. How does Affirm make money | Business Model [Електронний ресурс] / Shikhar Goel // The Strategy Story. – 2022. – Режим доступу до ресурсу: https://thestrategystory.com/2022/02/26/how-does-affirm-make-money-business-model/.

27. Java Web Services Architecture / M.Stevens, J. McGovern, S. Tyagi, M. Sunil., 2003. – 832 с.
28. eact A JavaScript library for building user interfaces [Електронний ресурс] // ReactJS organization. – 2022. – Режим доступу до ресурсу: https://reactjs.org/.
29. The modern web developer's platform [Електронний ресурс] // Google LLC. – 2022. – Режим доступу до ресурсу: https://angular.io/.
30. The Progressive JavaScript Framework [Електронний ресурс] // VueJS organization. – 2022. – Режим доступу до ресурсу: https://vuejs.org/.
31. Basic Types [Електронний ресурс] // TypeScript organization. – 2022. – Режим доступу до ресурсу: https://www.typescriptlang.org/docs/handbook/basic-types.html.
32. Hezbullah S. Node.jsChallenges in Implementation / S. Hezbullah, R. S. Tariq. // Global Journals Inc.. – 2017. – №17. – С. 2–12.
33. Cloud Functions for Firebase [Електронний ресурс] // Google LLC. – 2022. – Режим доступу до ресурсу: https://firebase.google.com/docs/functions#key_capabilities.
34. Manda D. PostgreSQL vs. MySQL: Which Is Best? [Електронний ресурс] / David Manda // Database Journal. – 2022. – Режим доступу до ресурсу: https://www.databasejournal.com/mysql/postgresql-vs-mysql-which-is-best/.
35. Reactive programming - Streams - BLoC [Електронний ресурс] // Didier Boelens. – 2018. – Режим доступу до ресурсу: Manda D. PostgreSQL vs. MySQL: Which Is Best? [Електронний ресурс] / David Manda // Database Journal. – 2022. – Режим доступу до ресурсу: https://www.databasejournal.com/mysql/postgresql-vs-mysql-which-is-best/.

ДОДАТОК 1

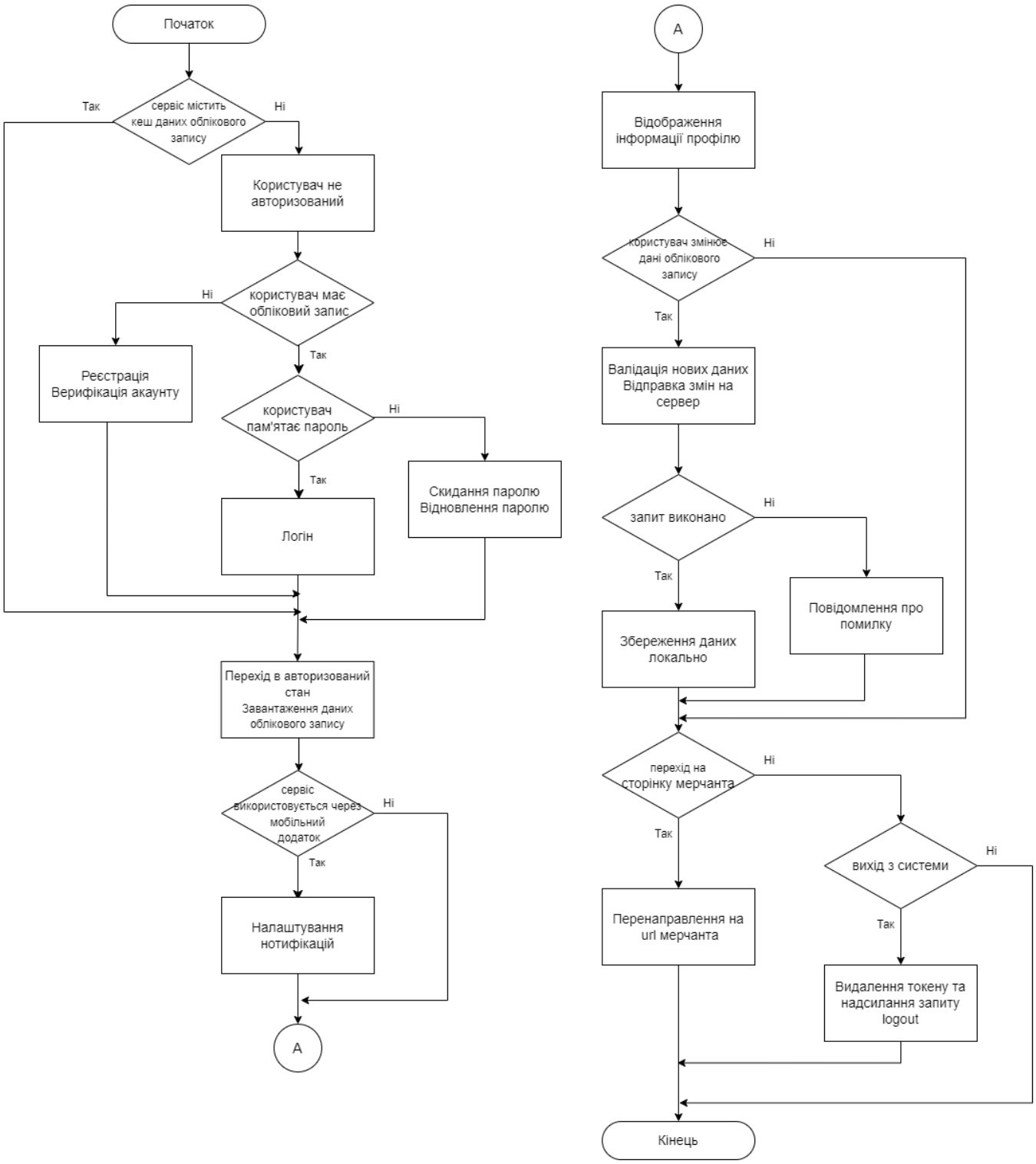
Сервіс для створення та керування відкладеними платежами за покупки в інтернеті

Схема функціонування сервісу (принципова схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробив		Базова Л. Г.			**Сервіс для створення та керування відкладеними платежами за покупки в інтернеті Схема бази даних**	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Русанова О. В.				
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-81
Н. Контр.		Сімоненко В. П.				
Затвердив						

ДОДАТОК 2

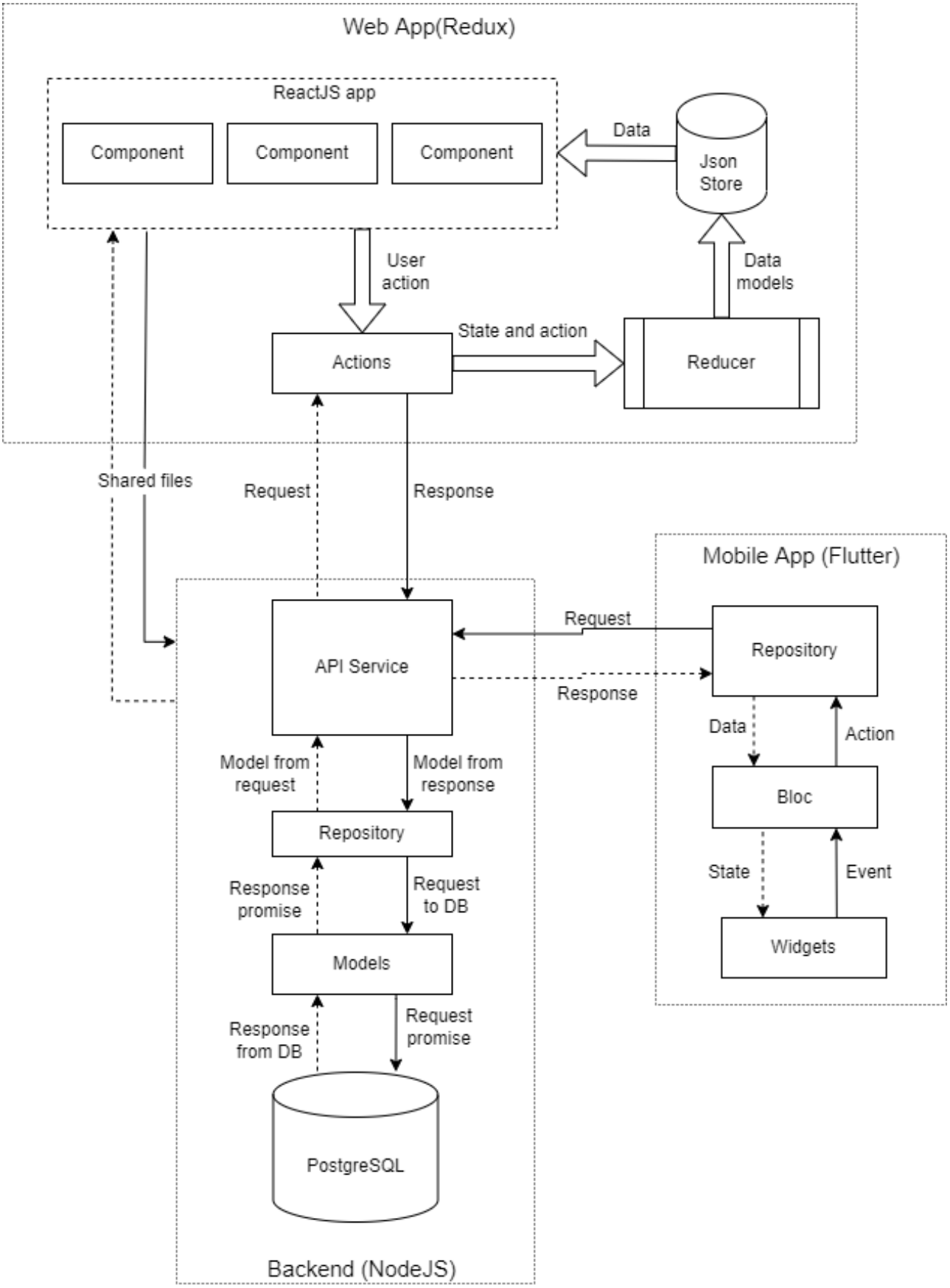
Сервіс для створення та керування відкладеними платежами за покупки в інтернеті

Структурна схема

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Базова Л. Г.			**Сервіс для створення та керування відкладеними платежами за покупки в інтернеті. Функціональна схема мобільного додатку**	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Русанова О. В.					
Н. Контр.		Сімоненко В. П.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-81	
Затвердив							

ДОДАТОК 3

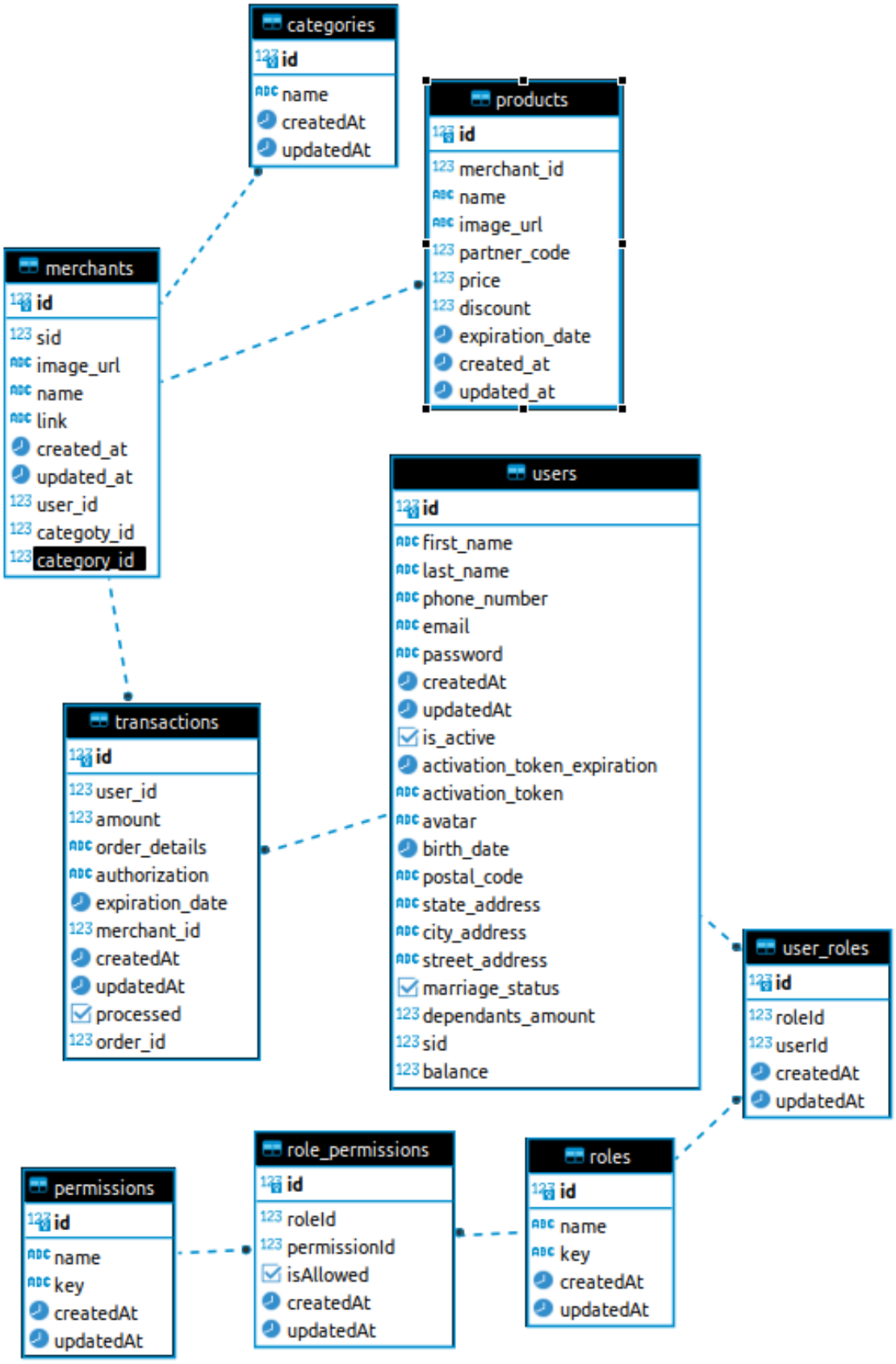
Сервіс для створення та керування відкладеними платежами за покупки в інтернеті

Функціональна схема серверної частини

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2021 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Базова Л. Г.			**Сервіс для створення та керування відкладеними платежами за покупки в інтернеті. Функціональна схема серверної частини**	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Русанова О. В.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІВ-81
Н. Контр.		Сімоненко В. П.				
Затвердив						

ДОДАТОК 4

Сервіс для створення та керування відкладеними платежами за покупки в інтернеті

Текст програмного коду мобільного додатку

ІАЛЦ.467200.007 Д4

Аркушів 46

Київ 2022 р

```
class OrderTransactionsBloc
    extends Bloc<OrderTransactionsEvent,
OrderTransactionsState> {
  final UserDataRepository _userDataRepository;

  OrderTransactionsBloc({UserDataRepository?
userDataRepository})
      : _userDataRepository = userDataRepository ??
UserDataRepository(),
        super(const OrderTransactionsState());

  @override
  Stream<OrderTransactionsState> mapEventToState(
    OrderTransactionsEvent event,
  ) async* {
    if (event is RefreshOrderTransactions) {
      yield* _mapRefreshOrderTransactionsToState();
    } else if (event is RefreshButtonPressed) {
      yield* _mapRefreshButtonPressedToState();
    }
  }

  Stream<OrderTransactionsState>
_mapRefreshOrderTransactionsToState() async* {
    try {
      final fetchedOrderTransactions =
          await
_userDataRepository.fetchOrderTransactions();
      yield
state.setOrderTransactions(fetchedOrderTransactions);
    } catch (_) {
      yield
state.setOrderTransactionsLoadingFailure();
    }
  }

  Stream<OrderTransactionsState>
_mapRefreshButtonPressedToState() async* {
    yield state.setRefreshingOrderTransactions();
    add(RefreshOrderTransactions());
  }
}

enum OrderTransactionsStatus { initial, success,
failure }

class OrderTransactionsState extends Equatable {
  const OrderTransactionsState({
    this.status = OrderTransactionsStatus.initial,
    this.orderTransactions,
  });

  final List<OrderTransactions>? orderTransactions;
  final OrderTransactionsStatus status;

  OrderTransactionsState
setOrderTransactionsLoadingFailure() {
    return _copyWith(status:
OrderTransactionsStatus.failure);
  }

  OrderTransactionsState setOrderTransactions(
      List<OrderTransactions> orderTransactions) {
    return _copyWith(
        orderTransactions: orderTransactions,
        status: OrderTransactionsStatus.success);
  }

  OrderTransactionsState
setRefreshingOrderTransactions() {
    return _copyWith(status:
OrderTransactionsStatus.initial);
  }

  OrderTransactionsState _copyWith(
      {List<OrderTransactions>? orderTransactions,
      required OrderTransactionsStatus status}) {
    return OrderTransactionsState(
        orderTransactions: orderTransactions ??
this.orderTransactions,
        status: status);

  @override
  List<Object?> get props => [orderTransactions,
status];
}

abstract class OrderTransactionsEvent extends
Equatable {
  const OrderTransactionsEvent();

  @override
  List<Object?> get props => [];
}

class RefreshOrderTransactions extends
OrderTransactionsEvent {}

class RefreshButtonPressed extends
OrderTransactionsEvent {}

class RestorePasswordBloc
    extends Bloc<RestorePasswordEvent,
RestorePasswordState> {
  RestorePasswordBloc(
      {required this.authenticationBloc,
AuthenticationRepository? authRepository})
      : this._authRepository = authRepository ??
AuthenticationRepository(),
        super(RestorePasswordState());

  final AuthenticationRepository _authRepository;
  AuthenticationBloc authenticationBloc;
  @override
  Stream<RestorePasswordState> mapEventToState(
    RestorePasswordEvent event,
  ) async* {
    if (event is ConfirmNewPasswordPressed) {
      yield* _mapConfirmNewPasswordPressedToState(
          event.password, event.confirmPassword);
    } else if (event is RestorePasswordShowMessage)
{
      yield*
_mapRestoreShowMessageToState(event.message);
    }
  }
  Stream<RestorePasswordState>
_mapConfirmNewPasswordPressedToState(
      String password, String confirmPassword)
async* {
    yield state.loading();
```

```
    try {
      final user = await
_authRepository.getUserByRestoredPassword(
          password, confirmPassword);

authenticationBloc.add(AuthenticationSetActiveUser
(user));
      add(RestorePasswordShowMessage(message:
'Password is restored'));
    } catch (exception) {
      add(RestorePasswordShowMessage(message:
exception.toString()));
    }
  }

  Stream<RestorePasswordState>
_mapRestoreShowMessageToState(
      String message) async* {
    yield state.showMessage(message);
  }
}

class RestorePasswordState extends Equatable {
  RestorePasswordState({
    this.errorMessage,
    this.isLoading = false,
  });

  final bool isLoading;
  final String? errorMessage;

  RestorePasswordState initial() {
    return RestorePasswordState(isLoading: false);
  }

  RestorePasswordState loading() {
    return RestorePasswordState(isLoading: true);
  }

  RestorePasswordState showMessage(String message)
{
    return copyWith(errorMessage: message,
isLoading: false);
  }

  RestorePasswordState copyWith({
    bool? isLoading,
    String? errorMessage,
  }) {
    return RestorePasswordState(
      isLoading: isLoading ?? this.isLoading,
      errorMessage: errorMessage ?? null,
    );
  }

  @override
  List<Object?> get props => [
        isLoading,
        errorMessage,
      ];
}

abstract class RestorePasswordEvent extends
Equatable {
  const RestorePasswordEvent();
}

class ConfirmNewPasswordPressed extends
RestorePasswordEvent {
  final String password;
  final String confirmPassword;

  ConfirmNewPasswordPressed(
      {required this.password, required
this.confirmPassword});

  @override
  List<Object?> get props => [password];
}

class RestorePasswordShowMessage extends
RestorePasswordEvent {
  final String message;

  RestorePasswordShowMessage({required
this.message});

  @override
  List<Object?> get props => [message];
}



class ForgotPasswordBloc
    extends Bloc<ForgotPasswordEvent,
ForgotPasswordState> {
  ForgotPasswordBloc({AuthenticationRepository?
authRepository})
      : this._authRepository = authRepository ??
AuthenticationRepository(),
        super(ForgotPasswordState());

  final AuthenticationRepository _authRepository;

  @override
  Stream<ForgotPasswordState> mapEventToState(
    ForgotPasswordEvent event,
  ) async* {
    if (event is SendRestoreLinkPressed) {
      yield*
_mapSendRestoreLinkPressedToState(event.email);
    } else if (event is ForgotPasswordShowMessage)
{
      yield*
_mapForgotPasswordShowMessageToState(event.message
);
    }
  }

  Stream<ForgotPasswordState>
_mapSendRestoreLinkPressedToState(
      String email) async* {
    yield state.loading();

    try {
      final response = await
_authRepository.forgotPasswordRequest(email);
      ForgotPasswordShowMessage(message: "Check
your email");
    } catch (exception) {
      ForgotPasswordShowMessage(message: "Check
your email");
    }
  }

  Stream<ForgotPasswordState>
_mapForgotPasswordShowMessageToState(
      String message) async* {
    yield state.showMessage(message);
  }
}

abstract class ForgotPasswordEvent extends
Equatable {
  const ForgotPasswordEvent();
}

class SendRestoreLinkPressed extends
ForgotPasswordEvent {
  final String email;

  SendRestoreLinkPressed({required this.email});

  @override
```

```
  List<Object?> get props => [email];
}

class ForgotPasswordShowMessage extends
ForgotPasswordEvent {
  final String message;

  ForgotPasswordShowMessage({required
this.message});

  @override
  List<Object?> get props => [message];
}

class ForgotPasswordState extends Equatable {
  ForgotPasswordState({
    this.errorMessage,
    this.isLoading = false,
  });

  final bool isLoading;
  final String? errorMessage;

  ForgotPasswordState initial() {
    return ForgotPasswordState(isLoading: false);
  }

  ForgotPasswordState loading() {
    return ForgotPasswordState(isLoading: true);
  }

  ForgotPasswordState showMessage(String message)
{
    return copyWith(errorMessage: message,
isLoading: false);
  }

  ForgotPasswordState copyWith({
    bool? isLoading,
    String? errorMessage,
  }) {
    return ForgotPasswordState(
      isLoading: isLoading ?? this.isLoading,
      errorMessage: errorMessage ?? null,
    );
  }

  @override
  List<Object?> get props => [
        isLoading,
        errorMessage,
      ];
}



const _merchantLimit = 15;
const _popularMerchantLimit = 15;
const throttleDuration = Duration(milliseconds:
100);

EventTransformer<E> throttleDroppable<E>(Duration
duration) {
  return (events, mapper) {
    return
droppable<E>().call(events.throttle(duration),
mapper);
  };
}

class MerchantBloc extends Bloc<MerchantEvent,
MerchantState> {
  final UserDataRepository _userDataRepository;

  MerchantBloc({UserDataRepository?
userDataRepository})
      : _userDataRepository = userDataRepository
?? UserDataRepository(),
        super(MerchantState(
            currentCategory: Categories.All,
            merchants:
Map.fromIterable(Categories.values,
                key: (e) => e, value: (_) =>
<Merchant>[]),
            hasReachedMax:
Map.fromIterable(Categories.values,
                key: (e) => e, value: (_) =>
false)));

  @override
  Stream<MerchantState>
mapEventToState(MerchantEvent event) async* {
    if (event is MerchantInitialFetch) {
      yield* _mapMerchantInitialFetchToState();
    } else if (event is MerchantFetchMore) {
      yield*
_mapMerchantFetchMoreToState(fetchPopular:
event.scrollingPopular);
    } else if (event is MerchantRefreshPressed) {
      yield* _mapMerchantRefreshPressedToState();
    } else if (event is MerchantTabSwitch) {
      yield*
_mapMerchantTabSwitchToState(event.currentCategory
);
    } else if (event is StartSearching) {
      yield state.copyWith(status:
MerchantStatus.searching);
    } else if (event is SearchQueryChanged) {
      yield*
_mapSearchQueryChangedToState(event.query);
    }
  }

  Stream<MerchantState>
_mapMerchantInitialFetchToState() async* {
    yield state.copyWith(status:
MerchantStatus.loading);
    try {
      final categorisedMerchants = await
_userDataRepository.fetchMerchants(
          limit: _merchantLimit, category:
state.currentCategory);
      final popularMerchants = await
_userDataRepository.fetchMerchants(
          limit: _popularMerchantLimit,
getPopular: true);

      state.merchants[state.currentCategory] =
categorisedMerchants;
      yield state.copyWith(
          status: MerchantStatus.success,
popularMerchants: popularMerchants);
    } catch (exception) {
      yield state.copyWith(status:
MerchantStatus.failure);
    }
  }

  Stream<MerchantState>
_mapMerchantFetchMoreToState(
      {required bool fetchPopular}) async* {
    yield state.copyWith(status:
MerchantStatus.loading);
    try {
      if (fetchPopular) {
        final popularMerchants = await
_userDataRepository.fetchMerchants(
            getPopular: true,
            limit: _popularMerchantLimit,
            startIndex:
state.popularMerchants.length);
        if (popularMerchants.isEmpty) {
```

```
          yield
state.copyWith(popularHasReachedMax: true);
        } else {

state.popularMerchants.addAll(popularMerchants);
          yield state.copyWith(status:
MerchantStatus.success);
        }
      } else {
        final categorisedMerchants = await
_userDataRepository.fetchMerchants(
            limit: _merchantLimit,
            category: state.currentCategory,
            startIndex:
state.merchants[state.currentCategory]!.length);
        if (categorisedMerchants.isEmpty) {

state.hasReachedMax[state.currentCategory] = true;
          yield state.copyWith(status:
MerchantStatus.success);
        } else {

state.merchants[state.currentCategory]!.addAll(cat
egorisedMerchants);
          yield state.copyWith(status:
MerchantStatus.success);
        }
      }
    } catch (exception) {
      yield state.copyWith(status:
MerchantStatus.failure);
    }
  }

  Stream<MerchantState>
_mapMerchantRefreshPressedToState() async* {
    yield MerchantState(
        merchants:
Map.fromIterable(Categories.values,
            key: (e) => e, value: (_) =>
<Merchant>[]),
        hasReachedMax:
Map.fromIterable(Categories.values,
            key: (e) => e, value: (_) => false),
        currentCategory: state.currentCategory);
    add(MerchantInitialFetch());
  }

  Stream<MerchantState>
_mapMerchantTabSwitchToState(
      Categories category) async* {
    yield state.copyWith(currentCategory:
category);
    if (state.merchants[category]!.isEmpty) {
      add(MerchantFetchMore());
    }
  }

  Stream<MerchantState>
_mapSearchQueryChangedToState(String query) async*
{
    if (query.isNotEmpty) {
      yield state.copyWith(status:
MerchantStatus.searchLoading);
      try {
        final searchMerchants = await
_userDataRepository.fetchMerchants(
            limit: _merchantLimit, searchQuery:
query);

        yield state.copyWith(
            status:
MerchantStatus.searchCompleted,
            searchMerchants: searchMerchants,
            searchQuery: query);
      } catch (exception) {
        yield state.copyWith(
            status: MerchantStatus.failure,
searchQuery: query);
      }
    } else {
      yield state.copyWith(status:
MerchantStatus.success, searchQuery: '');
    }
  }
}

enum MerchantStatus {
  loading,
  searchLoading,
  success,
  failure,
  searching,
  searchCompleted,
}

class MerchantState extends Equatable {
  const MerchantState({
    this.status = MerchantStatus.loading,
    required this.currentCategory,
    required this.merchants,
    required this.hasReachedMax,
    this.popularMerchants = const [],
    this.searchMerchants = const [],
    this.popularHasReachedMax = false,
    this.searchQuery = '',
  });

  final MerchantStatus status;
  final Categories currentCategory;

  final Map<Categories, List<Merchant>> merchants;
  final Map<Categories, bool> hasReachedMax;

  final List<Merchant> popularMerchants;
  final bool popularHasReachedMax;

  final List<Merchant> searchMerchants;
  final String searchQuery;

  MerchantState copyWith({
    MerchantStatus? status,
    Categories? currentCategory,
    Map<Categories, List<Merchant>>? merchants,
    Map<Categories, bool>? hasReachedMax,
    List<Merchant>? popularMerchants,
    bool? popularHasReachedMax,
    List<Merchant>? searchMerchants,
    String? searchQuery,
  }) {
    return MerchantState(
      status: status ?? this.status,
      currentCategory: currentCategory ??
this.currentCategory,
      merchants: merchants ?? this.merchants,
      hasReachedMax: hasReachedMax ??
this.hasReachedMax,
      popularMerchants: popularMerchants ??
this.popularMerchants,
      popularHasReachedMax: popularHasReachedMax
?? this.popularHasReachedMax,
      searchMerchants: searchMerchants ??
this.searchMerchants,
      searchQuery: searchQuery ??
this.searchQuery,
    );
  }

  @override
  List<Object> get props => [
        status,
        currentCategory,
        merchants,
        hasReachedMax,
```

```
        popularMerchants,
        popularHasReachedMax
      ];
}

abstract class MerchantEvent extends Equatable {
  @override
  List<Object> get props => [];
}

class MerchantInitialFetch extends MerchantEvent
{}

class MerchantFetchMore extends MerchantEvent {
  final bool scrollingPopular;

  MerchantFetchMore({this.scrollingPopular =
false});

  @override
  List<Object> get props => [scrollingPopular];
}

class MerchantRefreshPressed extends MerchantEvent
{}

class MerchantTabSwitch extends MerchantEvent {
  final Categories currentCategory;

  MerchantTabSwitch({required
this.currentCategory});
}

class SearchQueryChanged extends MerchantEvent{
  final String query;

  SearchQueryChanged(this.query);
}

class StartSearching extends MerchantEvent{}
```

```
class DashboardBloc extends Bloc<DashboardEvent,
DashboardState> {
  final UserDataRepository _userDataRepository;
  final TransactionBloc transactionBloc;

  DashboardBloc(
      {required this.transactionBloc,
UserDataRepository? userDataRepository})
      : _userDataRepository = userDataRepository
?? UserDataRepository(),
        super(const DashboardState());

  @override
  Stream<DashboardState>
mapEventToState(DashboardEvent event) async* {
    if (event is RefreshDashboard) {
      yield* _mapRefreshDashboardToState();
    } else if (event is RefreshButtonPressed) {
      yield* _mapRefreshButtonPressedToState();
    }
  }

  Stream<DashboardState>
_mapRefreshDashboardToState() async* {
    try {
      final fetchedDashboard = await
_userDataRepository.fetchDashboard();
      yield state.setDashboard(fetchedDashboard);
    } catch (_) {
      yield state.setDashboardLoadingFailure();
    }
  }

  Stream<DashboardState>
_mapRefreshButtonPressedToState() async* {
    yield state.setRefreshingDashboard();
    transactionBloc.add(RefreshIfFailed());
    add(RefreshDashboard());
  }
}
```

```
const _transactionLimit = 15;
const throttleDuration = Duration(milliseconds:
100);

EventTransformer<E> throttleDroppable<E>(Duration
duration) {
  return (events, mapper) {
    return
droppable<E>().call(events.throttle(duration),
mapper);
  };
}

class TransactionBloc extends
Bloc<TransactionEvent, TransactionState> {
  final UserDataRepository _userDataRepository;

  TransactionBloc({UserDataRepository?
userDataRepository})
      : _userDataRepository = userDataRepository
?? UserDataRepository(),
        super(const TransactionState());

  @override
  Stream<TransactionState> mapEventToState(
    TransactionEvent event,
  ) async* {
    if (event is TransactionRefreshPressed) {
      yield* _mapRefreshButtonPressedToState();
    } else if (event is RefreshIfFailed) {
      yield* _mapRefreshIfFailedToState();
    } else if (event is TransactionFetched) {
      yield* _mapTransactionFetched();
    }
  }

  Stream<TransactionState>
_mapTransactionFetched() async* {
    try {
      if (state.status ==
TransactionStatus.initial) {
        final transactions = await
_userDataRepository.fetchTransactions(
            transactionLimit: _transactionLimit);
        yield state.setTransaction(transactions);
      }
      final transactions = await
_userDataRepository.fetchTransactions(
          transactionLimit: _transactionLimit,
          startIndex: state.transactions.length);
      if (transactions.isEmpty) {
        yield state.setReachedMax(true);
      } else {
        yield state

.setTransaction(List.of(state.transactions)..addAl
l(transactions));
      }
    } catch (_) {
      yield state.setTransactionsLoadingFailure();
    }
  }

  Stream<TransactionState>
_mapRefreshButtonPressedToState() async* {
    yield state.setRefreshingTransactions();
    add(TransactionFetched());
```

```
  }

  Stream<TransactionState>
_mapRefreshIfFailedToState() async* {
    if (state.status == TransactionStatus.failure)
{
      yield state.setRefreshingTransactions();
      add(TransactionFetched());
    }
  }
}

abstract class TransactionEvent extends Equatable
{
  @override
  List<Object> get props => [];
}

class TransactionFetched extends TransactionEvent
{}

class TransactionRefreshPressed extends
TransactionEvent {}

class RefreshIfFailed extends TransactionEvent {}

enum TransactionStatus { initial, success, failure
}

class TransactionState extends Equatable {
  const TransactionState({
    this.status = TransactionStatus.initial,
    this.transactions = const <Transaction>[],
    this.hasReachedMax = false,
  });

  final TransactionStatus status;
  final List<Transaction> transactions;
  final bool hasReachedMax;

  TransactionState
setTransaction(List<Transaction>? transactions) {
    return _copyWith(
        status: TransactionStatus.success,
transactions: transactions);
  }

  TransactionState setReachedMax(bool
hasReachedMax) {
    return _copyWith(hasReachedMax:
hasReachedMax);
  }

  TransactionState setTransactionsLoadingFailure()
{
    return _copyWith(status:
TransactionStatus.failure);
  }

  TransactionState setRefreshingTransactions() {
    return _copyWith(status:
TransactionStatus.initial);
  }

  TransactionState _copyWith({
    TransactionStatus? status,
    List<Transaction>? transactions,
    bool? hasReachedMax,
  }) {
    return TransactionState(
      status: status ?? this.status,
      transactions: transactions ??
this.transactions,
      hasReachedMax: hasReachedMax ??
this.hasReachedMax,
    );
  }

  @override
  List<Object> get props => [status, transactions,
hasReachedMax];
}

enum DashboardStatus { initial, success, failure }

class DashboardState extends Equatable {
  const DashboardState({
    this.status = DashboardStatus.initial,
    this.dashboard,
  });

  final Dashboard? dashboard;
  final DashboardStatus status;

  DashboardState setDashboardLoadingFailure() {
    return _copyWith(status:
DashboardStatus.failure);
  }

  DashboardState setDashboard(Dashboard dashboard)
{
    return _copyWith(dashboard: dashboard, status:
DashboardStatus.success);
  }

  DashboardState setRefreshingDashboard() {
    return _copyWith(status:
DashboardStatus.initial);
  }

  DashboardState _copyWith(
      {Dashboard? dashboard, required
DashboardStatus status}) {
    return DashboardState(
        dashboard: dashboard ?? this.dashboard,
status: status);
  }

  @override
  List<Object?> get props => [dashboard, status];
}

abstract class DashboardEvent extends Equatable {
  const DashboardEvent();

  @override
  List<Object?> get props => [];
}

class RefreshDashboard extends DashboardEvent {}

class RefreshButtonPressed extends DashboardEvent
{}

class RegisterState extends Equatable {
  RegisterState({
    this.errorMessage,
    this.isLoading = false,
  });

  final bool isLoading;
  final String? errorMessage;

  RegisterState initial() {
    return RegisterState(isLoading: false);
  }

  RegisterState loading() {
    return RegisterState(isLoading: true);
  }

  RegisterState showMessage(String message) {
    return copyWith(errorMessage: message,
isLoading: false);
```

```
  }

  RegisterState copyWith({
    bool? isLoading,
    String? errorMessage,
  }) {
    return RegisterState(
      isLoading: isLoading ?? this.isLoading,
      errorMessage: errorMessage ?? null,
    );
  }

  @override
  List<Object?> get props => [
        isLoading,
        errorMessage,
      ];
}

abstract class RegisterEvent extends Equatable {
  const RegisterEvent();
}

class RegisterPressed extends RegisterEvent {
  final String firstName;
  final String lastName;
  final String email;
  final String phoneNumber;
  final String password;
  final String confirmPassword;

  RegisterPressed(
      {required this.firstName,
      required this.lastName,
      required this.email,
      required this.phoneNumber,
      required this.password,
      required this.confirmPassword});

  @override
  List<Object?> get props =>
      [firstName, lastName, email, phoneNumber,
password, confirmPassword];
}

class RegisterShowMessage extends RegisterEvent {
  final String message;

  RegisterShowMessage({required this.message});

  @override
  List<Object?> get props => [message];
}



class RegisterBloc extends Bloc<RegisterEvent,
RegisterState> {
  RegisterBloc({AuthenticationRepository?
authRepository})
      : this._authRepository = authRepository ??
AuthenticationRepository(),
        super(RegisterState());

  final AuthenticationRepository _authRepository;

  @override
  Stream<RegisterState> mapEventToState(
    RegisterEvent event,
  ) async* {
    if (event is RegisterPressed) {
      yield* _mapRegisterPressedToState(
          event.firstName,
          event.lastName,
          event.email,
          event.phoneNumber,
          event.password,
          event.confirmPassword);
    } else if (event is RegisterShowMessage) {
      yield*
_mapRegisterShowMessageToState(event.message);
    }
  }

  Stream<RegisterState>
_mapRegisterPressedToState(
      String firstName,
      String lastName,
      String email,
      String phoneNumber,
      String password,
      String confirmPassword) async* {
    yield state.loading();

    try {
      final response = await
_authRepository.registerUser(
          firstName, lastName, email, phoneNumber,
password, confirmPassword);

      add(RegisterShowMessage(message: "Check your
email"));
    } catch (exception) {
      add(RegisterShowMessage(message: "Check your
email"));
    }
  }

  Stream<RegisterState>
_mapRegisterShowMessageToState(String message)
async* {
    yield state.showMessage(message);
  }
}

enum EditProfileStatus {
  INITIAL,
  LOADING,
  PASSWORD_LOADING,
  ADDRESS_LOADING,
  FAMILY_STATUS_LOADING,
  EDITED,
  CHANGES_FAILED,
}

class EditProfileState extends Equatable {
  final EditProfileStatus editProfileStatus;
  final User user;
  final bool imageSourceActionSheetIsVisible;

  String get avatarPat => user.avatar;
  String get firstName => user.firstName;
  String get lastName => user.lastName;
  String get phoneNumber => user.phoneNumber;
  String get email => user.email;
  String? get birthDate => user.birthDate;
  String get avatarUrl => user.avatar;
  String get postalCode => user.postalCode;
  String get stateAddress => user.stateAddress;
  String get cityAddress => user.cityAddress;
  String get streetAddress => user.streetAddress;
  bool get marriageStatus => user.marriageStatus;
  int get dependantsAmount =>
user.dependantsAmount;
  String get joinedDate => user.createdAt;

  EditProfileState(
      {required this.user,
      this.editProfileStatus =
EditProfileStatus.INITIAL,
      this.imageSourceActionSheetIsVisible =
false});

  EditProfileState copyWith(
```

```
          {User? user,
          EditProfileStatus? editProfileStatus,
          bool? imageSourceActionSheetIsVisible})
=>
      EditProfileState(
          user: user ?? this.user,
          editProfileStatus: editProfileStatus ??
this.editProfileStatus,
          imageSourceActionSheetIsVisible:
imageSourceActionSheetIsVisible ??

this.imageSourceActionSheetIsVisible);

  @override
  List<Object?> get props =>
      [editProfileStatus, user,
imageSourceActionSheetIsVisible];
}



class EditProfileBloc extends
Bloc<EditProfileEvent, EditProfileState> {
  final _picker = ImagePicker();
  final EditProfileRepository
_editProfileRepository;

  EditProfileBloc(
      {required User user, EditProfileRepository?
editProfileRepository})
      : _editProfileRepository =
            editProfileRepository ??
EditProfileRepository(),
        super(EditProfileState(
            editProfileStatus:
EditProfileStatus.INITIAL, user: user));

  @override
  Stream<EditProfileState> mapEventToState(
    EditProfileEvent event,
  ) async* {
    if (event is SaveProfileChangesPressed) {
      yield* _mapSaveProfileChangesPressedToState(
          firstName: event.firstName,
          lastName: event.lastName,
          email: event.email,
          phoneNumber: event.phoneNumber,
          birthDate: event.birthDate);
    } else if (event is SavePasswordChanges) {
      yield* _mapSavePasswordChangesToState(
          oldPassword: event.oldPassword,
          newPassword: event.newPassword,
          confirmPassword: event.confirmPassword);
    } else if (event is SaveAddressChanges) {
      yield* _mapSaveAddressChangesToState(
          postalCode: event.postalCode,
          stateAddress: event.streetAddress,
          cityAddress: event.cityAddress,
          streetAddress: event.streetAddress);
    } else if (event is SaveFamilyStatusChanges) {
      yield* _mapSaveFamilyStatusChangesToState(
          marriageStatus: event.marriageStatus,
          dependantsAmount:
event.dependantsAmount);
    } else if (event is ChangeAvatarRequest) {
      yield
state.copyWith(imageSourceActionSheetIsVisible:
true);
    } else if (event is OpenImagePicker) {
      yield*
_mapOpenImagePickerToState(imageSource:
event.imageSource);
    }
  }

  Stream<EditProfileState>
_mapSaveProfileChangesPressedToState({
    required String firstName,
    required String lastName,
    required String email,
    required String phoneNumber,
    required String birthDate,
  }) async* {
    yield state.copyWith(editProfileStatus:
EditProfileStatus.LOADING);

    User editedUser = state.user;
    editedUser.firstName = firstName;
    editedUser.lastName = lastName;
    editedUser.email = email;
    editedUser.phoneNumber = phoneNumber;
    editedUser.birthDate = birthDate;

    yield* _editUserData(editedUser);
  }

  Stream<EditProfileState>
_mapSavePasswordChangesToState({
    required String oldPassword,
    required String newPassword,
    required String confirmPassword,
  }) async* {
    yield state.copyWith(editProfileStatus:
EditProfileStatus.PASSWORD_LOADING);

    try {
      await _editProfileRepository.changePassword(
          oldPassword: oldPassword,
          newPassword: newPassword,
          confirmPassword: confirmPassword);
      yield state.copyWith(
          editProfileStatus:
EditProfileStatus.EDITED,
          imageSourceActionSheetIsVisible: false);
    } catch (_) {
      yield state.copyWith(
          editProfileStatus:
EditProfileStatus.CHANGES_FAILED,
          imageSourceActionSheetIsVisible: false);
    }
  }

  Stream<EditProfileState>
_mapSaveAddressChangesToState(
      {required String postalCode,
      required String stateAddress,
      required String cityAddress,
      required String streetAddress}) async* {
    yield state.copyWith(editProfileStatus:
EditProfileStatus.ADDRESS_LOADING);

    User editedUser = state.user;

    editedUser.postalCode = postalCode;
    editedUser.stateAddress = stateAddress;
    editedUser.cityAddress = cityAddress;
    editedUser.streetAddress = streetAddress;

    yield* _editUserData(editedUser);
  }

  Stream<EditProfileState>
_mapSaveFamilyStatusChangesToState(
      {required int dependantsAmount, required
bool marriageStatus}) async* {
    yield state.copyWith(
        editProfileStatus:
EditProfileStatus.FAMILY_STATUS_LOADING);

    User editedUser = state.user;

    editedUser.dependantsAmount =
dependantsAmount;
    editedUser.marriageStatus = marriageStatus;
```

```
    yield* _editUserData(editedUser);
  }

  Stream<EditProfileState> _editUserData(User
editedUser) async* {
    try {
      await
_editProfileRepository.editProfile(user:
editedUser);
      yield state.copyWith(
          user: editedUser,
          editProfileStatus:
EditProfileStatus.EDITED,
          imageSourceActionSheetIsVisible: false);
    } catch (_) {
      yield state.copyWith(
          editProfileStatus:
EditProfileStatus.CHANGES_FAILED,
          imageSourceActionSheetIsVisible: false);
    }
  }

  Stream<EditProfileState>
_mapOpenImagePickerToState(
      {required ImageSource imageSource}) async* {
    yield state.copyWith(
        imageSourceActionSheetIsVisible: false,
        editProfileStatus:
EditProfileStatus.LOADING);

    final XFile? pickedImage = await
_picker.pickImage(source: imageSource);

    try {
      final imageUploadResponse =
          await
_editProfileRepository.uploadAvatar(pickedImage!.p
ath);
      User newUser = state.user;
      newUser.avatar = imageUploadResponse;
      yield* _editUserData(newUser);
    } catch (_) {
      yield state.copyWith(
          editProfileStatus:
EditProfileStatus.CHANGES_FAILED,
          imageSourceActionSheetIsVisible: false);
    }
  }
}

abstract class EditProfileEvent extends Equatable
{
  const EditProfileEvent();

  @override
  List<Object?> get props => [];
}

class SaveProfileChangesPressed extends
EditProfileEvent {
  final String firstName;
  final String lastName;
  final String email;
  final String phoneNumber;
  final String birthDate;

  SaveProfileChangesPressed(
      {required this.firstName,
      required this.lastName,
      required this.email,
      required this.phoneNumber,
      required this.birthDate});
}

class SavePasswordChanges extends EditProfileEvent
{
  final String oldPassword;
  final String newPassword;
  final String confirmPassword;

  SavePasswordChanges(
      {required this.oldPassword,
      required this.newPassword,
      required this.confirmPassword});
}

class SaveAddressChanges extends EditProfileEvent
{
  final String postalCode;
  final String stateAddress;
  final String cityAddress;
  final String streetAddress;

  SaveAddressChanges(
      {required this.postalCode,
      required this.stateAddress,
      required this.cityAddress,
      required this.streetAddress});
}

class SaveFamilyStatusChanges extends
EditProfileEvent {
  final bool marriageStatus;
  final int dependantsAmount;

  SaveFamilyStatusChanges({
    required this.dependantsAmount,
    required this.marriageStatus,
  });
}

class ChangeAvatarRequest extends EditProfileEvent
{}

class OpenImagePicker extends EditProfileEvent {
  final ImageSource imageSource;

  OpenImagePicker({required this.imageSource});
}

abstract class AuthenticationEvent extends
Equatable {
  const AuthenticationEvent();

  @override
  List<Object?> get props => [];
}

class AuthenticationInitialize extends
AuthenticationEvent {}

class AuthenticationSetActiveUser extends
AuthenticationEvent {
  AuthenticationSetActiveUser(this.user);

  final User user;

  @override
  List<Object?> get props => [user];
}

class AuthenticationSetInitializing extends
AuthenticationEvent {}

class AuthenticationSetUnauthorized extends
AuthenticationEvent {}

class AuthenticationLogOut extends
AuthenticationEvent {}
```

```
class AuthenticationBloc
    extends Bloc<AuthenticationEvent,
AuthenticationState> {
  AuthenticationBloc({AuthenticationRepository?
authRepository})
      : this._authRepository = authRepository ??
AuthenticationRepository(),
        super(AuthenticationInitial());

  final AuthenticationRepository _authRepository;

  @override
  Stream<AuthenticationState> mapEventToState(
    AuthenticationEvent event,
  ) async* {
    if (event is AuthenticationLogOut) {
      yield* _mapAuthenticationLogOutToState();
    } else if (event is AuthenticationInitialize)
{
      yield*
_mapAuthenticationInitializeToState();
    } else if (event is
AuthenticationSetActiveUser) {
      yield AuthenticationAuthenticated(user:
event.user);
    } else if (event is
AuthenticationSetUnauthorized) {
      yield AuthenticationUnauthenticated();
    } else if (event is
AuthenticationSetInitializing) {
      yield AuthenticationLoading();
    }
  }

  Stream<AuthenticationState>
_mapAuthenticationInitializeToState() async* {
    add(AuthenticationSetInitializing());
    try {
      User user = await
_authRepository.getAuthorisedUser();
      add(AuthenticationSetActiveUser(user));
    } catch (exception) {
      _authRepository.logout();
      add(AuthenticationSetUnauthorized());
    }
  }

  Stream<AuthenticationState>
_mapAuthenticationLogOutToState() async* {
    _authRepository.logout();
    yield AuthenticationUnauthenticated();
  }
}

enum AuthenticationStatus {
  AUTHENTICATED,
  UNAUTHENTICATED,
  UNKNOWN,
  INITIALIZING
}

abstract class AuthenticationState extends
Equatable {
  final authenticationStatus =
AuthenticationStatus.UNKNOWN;
  @override
  List<Object?> get props => [];
}

class AuthenticationInitial extends
AuthenticationState {}

class AuthenticationLoading extends
AuthenticationState {
  final authenticationStatus =
AuthenticationStatus.INITIALIZING;
}

class AuthenticationAuthenticated extends
AuthenticationState {
  final User user;
  final authenticationStatus =
AuthenticationStatus.AUTHENTICATED;

  AuthenticationAuthenticated({required
this.user});

  @override
  List<Object?> get props => [user];
}

class AuthenticationUnauthenticated extends
AuthenticationState {
  final authenticationStatus =
AuthenticationStatus.UNAUTHENTICATED;
}



class LoginBloc extends Bloc<LoginEvent,
LoginState> {
  LoginBloc(
      {AuthenticationRepository? authRepository,
      this.successMessage = 'Sign in successful',
      required this.authenticationBloc})
      : this._authRepository = authRepository ??
AuthenticationRepository(),
        super(LoginState());

  AuthenticationBloc authenticationBloc;
  final AuthenticationRepository _authRepository;
  final String successMessage;

  @override
  Stream<LoginState> mapEventToState(
    LoginEvent event,
  ) async* {
    if (event is LoginPressed) {
      yield* _mapLoginPressedToState(event.email,
event.password);
    }
  }

  Stream<LoginState> _mapLoginPressedToState(
      String email, String password) async* {
    yield state.loading();

    try {
      final User user = await
_authRepository.getUserByLogin(email, password);

authenticationBloc.add(AuthenticationSetActiveUser
(user));
      yield state.showMessage(successMessage);
    } catch (exception) {
      yield
state.showMessage(exception.toString());
    }
  }
}

abstract class LoginEvent extends Equatable {
  const LoginEvent();
}

class LoginPressed extends LoginEvent {

  final String email;
  final String password;

  LoginPressed({required this.email, required
this.password});

  @override
```

```
  List<Object?> get props => [email, password];
}

class LoginState extends Equatable {
  LoginState({
    this.message,
    this.isLoading = false,
  });

  final bool isLoading;
  final String? message;

  LoginState loading() {
    return _copyWith(isLoading: true);
  }

  LoginState showMessage(String message) {
    return _copyWith(errorMessage: message,
isLoading: false);
  }

  LoginState _copyWith({
    bool? isLoading,
    String? errorMessage,
  }) {
    return LoginState(
      isLoading: isLoading ?? this.isLoading,
      message: errorMessage ?? null,
    );
  }

  @override
  List<Object?> get props => [
        isLoading,
        message,
      ];
}


class HostConfig {
  static const String? _domain = null;
  static const String? _ip = null;
  static const String _port = "3011";
  static String? _generatedHost;
  static String host = "192.168.2.39:";

  static Future<String> getHost() async {
    if (_domain != null) {
      return _domain!;
    } else if (kIsWeb) {
      return host + _port;
    }
    _generatedHost = _generatedHost ?? await
_platformBasedHost();
    return _generatedHost as String;
  }

  static Future<String> _platformBasedHost() async
{
    final DeviceInfoPlugin deviceInfo =
DeviceInfoPlugin();

    if (Platform.isAndroid &&
        !(await
deviceInfo.androidInfo).isPhysicalDevice) {
      return host + _port;
    } else if (Platform.isIOS && !(await
deviceInfo.iosInfo).isPhysicalDevice) {
      return host + _port;
      // return "localhost:$_port";
    } else {
      return "$_ip:$_port";
    }
  }
}
class Order {
  String name;
  String image_url;
  double price;
  int numberOfPayments;
  DateTime created_at;

  Order({
    required this.name,
    required this.image_url,
    required this.numberOfPayments,
    required this.price,
    required this.created_at,
  });

  factory Order.fromJson(Map<String, dynamic>
json) {
    print(json);
    return Order(
      name: json['name'] as String,
      image_url: json['image_url'] as String,
      price: json['price'] as double,
      price: json['numberOfPayments'] as int,
      created_at:
DateTime.parse(json['created_at']),
    );
  }
}
enum StatusType { Pending, Upcoming, Waiting }
class OrderTransactions {
  int numberOfPayments;
  int currentPayment;
  double totalPrice;
  String brandName;
  String brandLogoUrl;
  double payments;
  DateTime paymentDates;
  bool isPaid;

  OrderTransactions({
    required this.numberOfPayments,
    required this.currentPayment,
    required this.totalPrice,
    required this.brandName,
    required this.brandLogoUrl,
    required this.payments,
    required this.paymentDates,
    required this.isPaid,
  });

  static DateTime tempDateCounter =
DateTime.now().add(Duration(days: 50));
  factory OrderTransactions.defaultData() {
    tempDateCounter =
tempDateCounter.subtract(Duration(days: 20));
    return OrderTransactions(
      numberOfPayments: 8,
      currentPayment: 3,
      totalPrice: 2328.0,
      brandName: "Amazon",
      brandLogoUrl:

'https://upload.wikimedia.org/wikipedia/commons/d/
de/Amazon_icon.png',
      payments: 291.0,
      paymentDates: tempDateCounter,
      isPaid:
DateTime.now().isAfter(tempDateCounter),
    );
  }
}


class User {
  get username => '$firstName $lastName';

  int id;
  String firstName;
  String lastName;
```

```
  String phoneNumber;
  String email;
  String password;
  String? birthDate;
  String avatar;
  bool isActive;
  String activationTokenExpiration;
  String? activationToken;
  String postalCode;
  String stateAddress;
  String cityAddress;
  String streetAddress;
  bool marriageStatus;
  int dependantsAmount;
  String createdAt;
  String updatedAt;
  List<Role> roles;

  User({
    required this.id,
    required this.firstName,
    required this.lastName,
    required this.phoneNumber,
    required this.email,
    required this.password,
    required this.birthDate,
    required this.avatar,
    required this.isActive,
    required this.activationTokenExpiration,
    required this.activationToken,
    required this.postalCode,
    required this.stateAddress,
    required this.cityAddress,
    required this.streetAddress,
    required this.marriageStatus,
    required this.dependantsAmount,
    required this.createdAt,
    required this.updatedAt,
    required this.roles,
  });

  factory User.fromJson(String body) {
    Map<String, dynamic> json = jsonDecode(body);

    List r = json['roles'];
    List<Role> roles = [];
    for (int i = 0; i < r.length; i++) {
      roles.add(Role.fromJson(r[i]));
    }

    return User(
      id: json['id'] as int,
      firstName: json['firstName'] as String,
      lastName: json['lastName'] as String,
      phoneNumber: json['phoneNumber'] as String,
      email: json['email'] as String,
      password: json['password'] as String,
      birthDate: json['birthDate'],
      avatar: json['avatar'] as String,
      isActive: json['isActive'] as bool,
      activationTokenExpiration:
json['activationTokenExpiration'] as String,
      activationToken: json['activationToken'],
      postalCode: json['postalCode'] as String,
      stateAddress: json['stateAddress'] as
String,
      cityAddress: json['cityAddress'] as String,
      streetAddress: json['streetAddress'] as
String,
      marriageStatus: json['marriageStatus'] as
bool,
      dependantsAmount: json['dependantsAmount']
as int,
      createdAt: json['createdAt'] as String,
      updatedAt: json['updatedAt'] as String,
      roles: roles,
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'id': id.toString(),
      'firstName': firstName,
      'lastName': lastName,
      'phoneNumber': phoneNumber,
      'email': email,
      'password': password,
      'birthDate': birthDate,
      'avatar': avatar,
      'isActive': isActive,
      'postalCode': postalCode,
      'stateAddress': stateAddress,
      'cityAddress': cityAddress,
      'streetAddress': streetAddress,
      'marriageStatus': marriageStatus,
      'dependantsAmount': dependantsAmount,
      'roles': jsonEncode(roles)
    };
  }
}
class Role {
  String key;
  String name;
  int id;

  Role({
    required this.key,
    required this.name,
    required this.id,
  });

  factory Role.fromJson(Map<String, dynamic> json)
{
    return Role(
        key: json['key'] as String,
        name: json['name'] as String,
        id: json['id'] as int,
        );
  }

  Map<String, dynamic> toJson() => {
    'id': id,
    'name': name,
    'key': key,
    'permissions': [],
  };
}

class Dashboard {

  String accountBalance;
  String pending;
  String processed;

  Dashboard({
    required this.accountBalance,
    required this.pending,
    required this.processed,
  });

  factory Dashboard.fromJson(String body) {
    Map<String, dynamic> json = jsonDecode(body);

    return Dashboard(
      accountBalance: json['accountBalance'] as
String,
      pending: json['pending'] as String,
      processed: json['processed'] as String,
    );
  }

  factory Dashboard.defaultData() {
    return Dashboard(
      accountBalance: '800 000',
```

```
      pending: '70 000',
      processed: '120 000',
    );
  }
}


class Transaction {
  String name;
  String brand;
  String amount;
  String progress;
  DateTime date;
  IconData icon;

  Transaction({
    required this.name,
    required this.brand,
    required this.amount,
    required this.progress,
    required this.date,
    required this.icon,
  });

  static const Map<String, IconData> _icons = {
    "type0": Icons.payments,
    "type1": Icons.add_circle_outline,
    "type2": Icons.payment,
    "type3": Icons.arrow_circle_down,
  };

  factory Transaction.fromJson(Map<String,
dynamic> json) {
    Map<int, String> merchants = {
      1: "Rozetka",
      2: "Eldorado",
      3: "Bookclub",
      4: "MOYO",
      5: "Book Ye",
      6: "Arena",
      7: 'Empik'
    };
    tempDateCounter =
Transaction.tempDateCounter.subtract(Duration(hour
s: 9));
    final _random = new Random();
    IconData icon =
_icons.values.elementAt(_random.nextInt(_icons.len
gth));
    return Transaction(
      name: json['order_details'] as String,
      amount: json['amount'].toString(),
      date: tempDateCounter,
      brand: merchants[_random.nextInt(7)]!,
      progress: json['processed'] ? "Paid" :
"Upcoming",
      icon: icon,
    );
  }

  static DateTime tempDateCounter =
DateTime.now();
  factory Transaction.defaultTransaction(int id) {
    tempDateCounter =
Transaction.tempDateCounter.subtract(Duration(hour
s: 9));
    final _random = new Random();
    IconData icon =
_icons.values.elementAt(_random.nextInt(_icons.len
gth));
    return Transaction(
      name: "Pay to $id",
      brand: "Shop $id",
      amount: ((id + 10) * 3).toString(),
      progress: "progress$id",
      icon: icon,
      date: tempDateCounter,
    );
  }
}
export 'user/user.dart';
export 'user/role.dart';

export 'merchant/categories.dart';
export 'merchant/merchant.dart';

export 'dashboard/dashboard.dart';
export 'dashboard/transaction.dart';

export 'order/order_transactions.dart';
export 'order/order.dart';
export 'order/status_type.dart';

class Merchant {
  int id;
  int sid;
  String imageUrl;
  String name;
  String link;
  // Categories category;
  Merchant({
    required this.id,
    required this.sid,
    required this.imageUrl,
    required this.name,
    required this.link,
    // required this.category,
  });

  factory Merchant.fromJson(Map<String, dynamic>
json) {
    return Merchant(
      id: json['id'] as int,
      sid: json['sid'] as int,
      imageUrl: json['image_url'] as String,
      name: json['name'] as String,
      link: json['link'] as String,
      // category: json['category'] as Categories,
    );
  }
}
enum Categories{
  All,
  Books,
  Sport,
  Appliances,

class SplashScreen extends StatelessWidget {
  const SplashScreen({Key? key}) : super(key:
key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(body: Center(child:
CircularProgressIndicator()));
  }
}


class MerchantAppBar extends StatefulWidget with
PreferredSizeWidget {
  @override
  State<MerchantAppBar> createState() =>
_MerchantAppBarState();

  @override
  Size get preferredSize =>
Size.fromHeight(kToolbarHeight) * 1.8;
}

class _MerchantAppBarState extends
State<MerchantAppBar> {
  final TextEditingController
_searchQueryController = TextEditingController();
```

```
  bool _isSearching = false;
  String searchQuery = '';

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<MerchantBloc,
MerchantState>(builder: (context, state) {
      _isSearching = state.status ==
MerchantStatus.searching;
      var searchedFound = state.status ==
MerchantStatus.searching ||
          state.status ==
MerchantStatus.searchCompleted ||
          state.status ==
MerchantStatus.searchLoading;
      return designAppBar(
        title: searchedFound ? searchQuery :
S.current.merchantsTitle,
        titleWidget: _isSearching ?
_buildSearchField(context, state) : null,
        bottom: searchedFound ? null :
MerchantsTabBar(),
        leading: _isSearching ? const BackButton()
: null,
        actions: _buildActions(context,
searchedFound),
      );
    });
  }

  Widget _buildSearchField(BuildContext context,
MerchantState state) {
    return TextField(
      controller: _searchQueryController,
      autofocus: true,
      decoration: InputDecoration(
        hintText: S.current.searchMerchantsLabel,
        border: InputBorder.none,
      ),
      onChanged: (query) => searchQuery = query,
    );
  }

  List<Widget> _buildActions(BuildContext context,
bool showClear) {
    return <Widget>[
      IconButton(
        icon: const Icon(Icons.search),
        onPressed: _isSearching
            ? () => context
                .read<MerchantBloc>()

.add(SearchQueryChanged(searchQuery))
            : () =>
context.read<MerchantBloc>().add(StartSearching())
,
      ),
      showClear
          ? IconButton(
              icon: const Icon(Icons.clear),
              onPressed: () {
                _isSearching = false;

context.read<MerchantBloc>().add(SearchQueryChange
d(''));
              },
            )
          : Container(),
    ];
  }
}


class HorizontalListWidget extends StatelessWidget
{
  final List<Merchant> merchants;
  final bool hasReachedMax;

  HorizontalListWidget({
    required this.merchants,
    required this.hasReachedMax,
  });

  @override
  Widget build(BuildContext context) {
    return merchants.length > 0
        ? Column(
            crossAxisAlignment:
CrossAxisAlignment.stretch,
            children: [
              Container(
                margin: EdgeInsets.only(top: 4),
                decoration: designBoxDecorator(
                    hasCircularBorders: false,
hasGradient: true),
                child: Container(
                  margin:
EdgeInsets.symmetric(horizontal: 8, vertical: 4),
                  child: Text(
                    S.current.popularLabel,
                    style: TextStyle(
                      color:
AppColors.lightContrastColor,
                      fontSize: 20,
                    ),
                  ),
                ),
              ),
              Container(
                height: 115,
                child: ListView.builder(
                  itemCount: merchants.length,
                  scrollDirection:
Axis.horizontal,
                  itemBuilder: (context, index) {
                    return MerchantWidget(
                        merchant:
merchants[index], givenSize: 110);
                  },
                ),
              ),
            ],
          )
        : Container();
  }
}

class MerchantsPage extends StatefulWidget {
  const MerchantsPage({Key? key}) : super(key:
key);

  @override
  _MerchantsPageState createState() =>
_MerchantsPageState();
}

class _MerchantsPageState extends
State<MerchantsPage> {
  final List<Categories> categories =
Categories.values;

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: Categories.values.length,
      child: Directionality(
        textDirection: TextDirection.ltr,
        child: MultiBlocProvider(
          providers: [
            BlocProvider<MerchantBloc>(
                create: (context) =>

MerchantBloc()..add(MerchantInitialFetch())),
          ],
```

```
          child: _merchantsContent(),
        ),
      ),
    );
  }

  Widget _merchantsContent() {
    return Scaffold(
        drawer: Drawer(
          key: const Key('all-merchants-page-
drawer'),
          child: MenuWidget(),
        ),
        appBar: MerchantAppBar(),
        body: BlocBuilder<MerchantBloc,
MerchantState>(
          builder: (context, state) {
            switch (state.status) {
              case MerchantStatus.failure:
                return RefreshButton(
                  label: S.current.loadingFailed,
                  onPressed: () => context
                      .read<MerchantBloc>()

.add(MerchantRefreshPressed()),
                );
              case MerchantStatus.success:
                return Column(children: [
                  Expanded(
                      child: _tabbedMerchants(
                          state.merchants,
state.hasReachedMax)),
                  HorizontalListWidget(
                    merchants:
state.popularMerchants,
                    hasReachedMax:
state.popularHasReachedMax,
                  )
                ]);
              case MerchantStatus.searchCompleted:
                return
_currMerchants(state.searchMerchants, true);
              case MerchantStatus.searching:
                return Center(
                  child:
Text(S.current.merchantsHintText,
                      textAlign: TextAlign.center,
textScaleFactor: 1.5),
                );
              default:
                return const Center(child:
CircularProgressIndicator());
            }
          },
        ));
  }

  Widget _tabbedMerchants(
      Map<Categories, List<Merchant>> merchants,
hasReachedMax) {
    return TabBarView(
        children: merchants.entries
            .map((e) => _currMerchants(e.value,
hasReachedMax[e.key]))
            .toList());
  }

  Widget _currMerchants(List<Merchant> merchants,
bool hasReachedMax) {
    return merchants.isEmpty
        ? Center(
            child:
Text(S.current.noMerchantsResult,
                textAlign: TextAlign.center,
textScaleFactor: 1.5))
        : GridView.builder(
            gridDelegate: const
SliverGridDelegateWithFixedCrossAxisCount(
              crossAxisCount: 3,
            ),
            itemCount: merchants.length,
            itemBuilder: (BuildContext context,
int index) {
              return MerchantWidget(merchant:
merchants[index]);
            });
  }
}

class MerchantsTabBar extends StatelessWidget with
PreferredSizeWidget {
  final List<Categories> categories =
Categories.values;

  @override
  Size get preferredSize =>
Size.fromHeight(kToolbarHeight) * 1.8;

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: designBoxDecorator(
          hasGradient: true, hasCircularBorders:
false, hasShadow: false),
      child: TabBar(
        labelColor: AppColors.darkAccentColor,
        unselectedLabelColor:
AppColors.lightContrastColor,
        indicatorSize: TabBarIndicatorSize.label,
        indicator: BoxDecoration(
            borderRadius: BorderRadius.only(
                topLeft: Radius.circular(15),
topRight: Radius.circular(15)),
            color: AppColors.backgroundColor),
        isScrollable: true,
        onTap: (index) {
          context
              .read<MerchantBloc>()

.add(MerchantTabSwitch(currentCategory:
categories[index]));
        },
        tabs: categories.map((e) {
          var tabTitle =

e.toString().replaceFirst('Categories.', '   ') +
'   ';
          if (e.index == 0) {
            tabTitle = '\u{300A}' + tabTitle;
          } else if (e.index == categories.length
- 1) {
            tabTitle = tabTitle + '\u{300B}';
          }
          return Tab(child: Text(tabTitle));
        }).toList(),
      ),
    );
  }
}

class DashboardPage extends StatefulWidget {
  const DashboardPage({Key? key}) : super(key:
key);

  @override
  _DashboardPageState createState() =>
_DashboardPageState();
}

class _DashboardPageState extends
State<DashboardPage> {
  @override
  Widget build(BuildContext context) {
```

```
    return Scaffold(
      drawer: Drawer(
        child: MenuWidget(),
      ),
      appBar: designAppBar(title:
S.current.dashboardTitle),
      body: dashboardPageBody(),
    );
  }

  Widget dashboardPageBody() {
    return MultiBlocProvider(
      providers: [
        BlocProvider(
            create: (context) =>
TransactionBloc()..add(TransactionFetched())),
        BlocProvider(
            create: (context) =>
                DashboardBloc(transactionBloc:
context.read<TransactionBloc>())
                  ..add(RefreshDashboard())),
      ],
      child: Column(
        children: [
          dashboardData(),
          Expanded(child: TransactionsList()),
        ],
      ),
    );
  }

  Widget dashboardData() {
    return Container(
      height: 242,
      width: double.infinity,
      child: BlocBuilder<DashboardBloc,
DashboardState>(
        builder: (context, state) {
          switch (state.status) {
            case DashboardStatus.failure:
              return RefreshButton(
                label: S.current.loadingFailed,
                onPressed: () => setState(() {

context.read<DashboardBloc>()..add(RefreshButtonPr
essed());
                }),
              );
            case DashboardStatus.success:
              return balances(
                  accountBalance:
state.dashboard!.accountBalance,
                  processed:
state.dashboard!.processed,
                  pending:
state.dashboard!.pending);
            default:
              return const Center(child:
CircularProgressIndicator());
          }
        },
      ),
    );
  }

  Widget balances(
      {required String accountBalance,
      required String processed,
      required String pending}) {
    return Container(
      child: IntrinsicHeight(
        child: Column(children: [
          NumberTitledBox(
              number: accountBalance,
              label:
S.current.accountBalanceLabel,
              scale: 0.4),
          Row(mainAxisAlignment:
MainAxisAlignment.spaceAround, children: [
            Expanded(
                child: NumberTitledBox(
                    number: processed, label:
S.current.processedLabel)),
            Expanded(
                child: NumberTitledBox(
                    number: pending, label:
S.current.pendingLabel)),
          ]),
        ]),
      ),
    );
  }
}

class TransactionsList extends StatefulWidget {
  @override
  _TransactionsListState createState() =>
_TransactionsListState();
}

class _TransactionsListState extends
State<TransactionsList> {
  final _scrollController = ScrollController();

  @override
  void initState() {
    super.initState();
    _scrollController.addListener(_onScroll);
  }

  @override
  Widget build(BuildContext context) {
    return BlocBuilder<TransactionBloc,
TransactionState>(
      builder: (context, state) {
        switch (state.status) {
          case TransactionStatus.failure:
            return
context.read<DashboardBloc>().state.status ==
                    DashboardStatus.failure
                ? Container()
                : RefreshButton(
                    label:
S.current.loadingFailed,
                    onPressed: () {
                      context
                          .read<TransactionBloc>()

.add(TransactionRefreshPressed());
                    },
                  );
          case TransactionStatus.success:
            if (state.transactions.isEmpty) {
              return Center(child:
Text(S.current.noTransactions));
            }
            return
groupedListView(state.transactions,
state.hasReachedMax);
          default:
            return const Center(child:
CircularProgressIndicator());
        }
      },
    );
  }

  Widget groupedListView(List<Transaction>
transactions, bool hasReachedMax) {
    return GroupedListView(
        sort: false,
        elements: transactions,
        separator: SizedBox(height: 8),
        groupBy: (Transaction element) =>
```

```
            DateTime(element.date.year,
element.date.month, element.date.day),
        groupSeparatorBuilder: (DateTime date) =>
DateListHeader(
              date,
              todayHeader: S.current.todayHeader,
            ),
        controller: _scrollController,
        indexedItemBuilder:
            (BuildContext context, Transaction
element, int index) {
          return ((index + 1) ==
transactions.length && !hasReachedMax)
              ? Column(children: [
                  transactionListItem(element),
                  BottomLoader(),
                ])
              : transactionListItem(element);
        });
  }

  Widget transactionListItem(Transaction
transaction) {
    return TransactionListItem(
        icon: transaction.icon,
        title: transaction.name,
        subtitle: transaction.brand,
        price: transaction.amount);
  }

  @override
  void dispose() {
    _scrollController
      ..removeListener(_onScroll)
      ..dispose();
    super.dispose();
  }

  void _onScroll() {
    if (_isBottom)
context.read<TransactionBloc>().add(TransactionFet
ched());
  }

  bool get _isBottom {
    if (!_scrollController.hasClients) return
false;
    final maxScroll =
_scrollController.position.maxScrollExtent;
    final currentScroll =
_scrollController.offset;
    return currentScroll >= (maxScroll * 0.9);
  }
}

class EditProfilePage extends StatefulWidget {
  const EditProfilePage({Key? key}) : super(key:
key);

  @override
  _EditProfilePageState createState() =>
_EditProfilePageState();
}

class _EditProfilePageState extends
State<EditProfilePage> {
  @override
  Widget build(BuildContext context) {
    return BlocBuilder<AuthenticationBloc,
AuthenticationState>(
        builder: (context, state) {
      return state is AuthenticationAuthenticated
          ? BlocProvider(
              create: (context) =>
EditProfileBloc(user: state.user),
              child: _editProfileContents(),
            )
          : Container(child: Text("Something went
wrong\nNo user found"));
    });
  }

  Widget _editProfileContents() {
    return BlocListener<EditProfileBloc,
EditProfileState>(
      listener: (context, state) {
        if (state.editProfileStatus ==
EditProfileStatus.CHANGES_FAILED) {
          _safeShowSnackBar(context,
S.current.editErrorMsg);
        } else if (state.editProfileStatus ==
EditProfileStatus.EDITED) {
          _safeShowSnackBar(context,
S.current.editSuccessMsg);
        } else if
(state.imageSourceActionSheetIsVisible) {
          _showImageSourceDialog(context);
        } else if (state.editProfileStatus ==
EditProfileStatus.LOADING) {
          _safeShowSnackBar(context,
S.current.waitLabel);
        }
      },
      child: EditProfileContent(),
    );
  }

  void _showImageSourceDialog(BuildContext
context) {
    showImageSourceActionSheet(
        selectImageSource: (imageSource) {
          context
              .read<EditProfileBloc>()
              .add(OpenImagePicker(imageSource:
imageSource));
        },
        galleryLabel: S.current.galleryLabel,
        cameraLabel: S.current.cameraLabel,
        context: context);
  }

  void _safeShowSnackBar(BuildContext context,
String message) {
    ScaffoldMessenger.of(context)
      ..hideCurrentSnackBar()
      ..showSnackBar(
        SnackBar(content: Text(message)),
      );
  }
}

class EditAddressPage extends StatefulWidget {
  EditAddressPage(
      {Key? key,
      required this.initPostalCode,
      required this.initStateAddress,
      required this.initCityAddress,
      required this.initStreetAddress})
      : super(key: key);

  final String initPostalCode;
  final String initStateAddress;
  final String initCityAddress;
  final String initStreetAddress;

  @override
  State<EditAddressPage> createState() =>
_EditAddressPageState();
}

class _EditAddressPageState extends
State<EditAddressPage> {
  late String postalCode;
  late String stateAddress;
```

```
  late String cityAddress;
  late String streetAddress;

  @override
  Widget build(BuildContext context) {
    postalCode = widget.initPostalCode;
    stateAddress = widget.initStateAddress;
    cityAddress = widget.initCityAddress;
    streetAddress = widget.initStreetAddress;
    return SafeArea(
        child: Scaffold(
      appBar: designAppBar(
        title: S.current.changeAddressLabel,
        leading: IconButton(
          icon: Icon(Icons.arrow_back, key:
Key('back-arrow-button')),
          onPressed: () => Navigator.pop(context),
        ),
      ),
      body: _editAddressBody(),
    ));
  }

  Widget _editAddressBody() {
    return SingleChildScrollView(
      child: Container(
        padding: EdgeInsets.symmetric(horizontal:
30, vertical: 30),
        child: Column(
          children: [
            _postalCodeField(),
            SizedBox(height: 26),
            _stateAddressField(),
            SizedBox(height: 26),
            _cityAddressField(),
            SizedBox(height: 26),
            _streetAddressField(),
            SizedBox(height: 12),
            DesignButton(
                onTap: _saveChanges,
                child: Text(
                  S.current.saveLabel,
                  style: radencyButtonTextStyle(),
                )),
          ],
        ),
      ),
    );
  }

  Widget _postalCodeField() {
    return CustomTextFormField(
      initialValue: postalCode,
      title:
fieldTitle(S.current.postalCodeField),
      onChangedCallback: (value) => postalCode =
value,
      titleSpace: 2.0,
    );
  }

  Widget _stateAddressField() {
    return CustomTextFormField(
      initialValue: stateAddress,
      title: fieldTitle(S.current.stateField),
      onChangedCallback: (value) => stateAddress =
value,
      titleSpace: 2.0,
    );
  }

  Widget _cityAddressField() {
    return CustomTextFormField(
      initialValue: cityAddress,
      title: fieldTitle(S.current.cityField),
      onChangedCallback: (value) => cityAddress =
value,
      titleSpace: 2.0,
    );
  }

  Widget _streetAddressField() {
    return CustomTextFormField(
      initialValue: streetAddress,
      title: fieldTitle(S.current.streetField),
      textInputType: TextInputType.streetAddress,
      onChangedCallback: (value) => streetAddress
= value,
      titleSpace: 2.0,
    );
  }

  void _saveChanges() {
    if (_dataIsChanged()) {
      Navigator.pop(
          context,
          SaveAddressChanges(
            stateAddress: stateAddress,
            streetAddress: streetAddress,
            cityAddress: cityAddress,
            postalCode: postalCode,
          ));
    } else {
      Navigator.pop(context);
    }
  }

  bool _dataIsChanged() {
    return widget.initStreetAddress !=
streetAddress ||
        widget.initCityAddress != cityAddress ||
        widget.initStateAddress != stateAddress ||
        widget.initPostalCode != postalCode;
  }
}

class EditPasswordPage extends StatefulWidget {
  const EditPasswordPage({Key? key}) : super(key:
key);

  @override
  State<EditPasswordPage> createState() =>
_EditPasswordPageState();
}

class _EditPasswordPageState extends
State<EditPasswordPage> {
  late String oldPassword;
  late String newPassword;
  late String confirmPassword;

  final GlobalKey<FormState> _oldPasswordFormKey =
      GlobalKey<FormState>(debugLabel: 'old
password key');
  final GlobalKey<FormState> _newPasswordFormKey =
      GlobalKey<FormState>(debugLabel: 'new
password key');
  final GlobalKey<FormState>
_passwordRepeatFormKey =
      GlobalKey<FormState>(debugLabel: 'password
repeat key');

  AutovalidateMode autovalidateMode =
AutovalidateMode.disabled;

  @override
  Widget build(BuildContext context) {
    return SafeArea(
        child: Scaffold(
      appBar: designAppBar(
        title: S.current.changePasswordLabel,
        leading: IconButton(
          icon: Icon(Icons.arrow_back, key:
Key('back-arrow-button')),
```

```
          onPressed: () => Navigator.pop(context),
        ),
      ),
      body: _editPasswordBody(),
    ));
  }

  Widget _editPasswordBody() {
    return SingleChildScrollView(
      child: Container(
        padding: EdgeInsets.symmetric(horizontal:
30, vertical: 30),
        child: Column(
          children: [
            _oldPasswordField(),
            SizedBox(height: 26),
            _newPasswordField(),
            SizedBox(height: 26),
            _confirmPasswordField(),
            SizedBox(height: 12),
            DesignButton(
                onTap: _saveChanges,
                child: Text(
                  S.current.saveLabel,
                  style: radencyButtonTextStyle(),
                )),
          ],
        ),
      ),
    );
  }

  Widget _oldPasswordField() {
    return CustomPasswordFormField(
      formKey: _oldPasswordFormKey,
      title:
fieldTitle(S.current.oldPasswordField),
      onChangedCallback: (value) => oldPassword =
value,
      autovalidateMode: autovalidateMode,
    );
  }

  Widget _newPasswordField() {
    return CustomPasswordFormField(
      formKey: _newPasswordFormKey,
      title:
fieldTitle(S.current.newPasswordField),
      onChangedCallback: (value) => newPassword =
value,
      autovalidateMode: autovalidateMode,
      validator: (String? value) {
        return validatePassword(
          password: value,
          shouldValidateComplexity: true,
        );
      },
    );
  }

  Widget _confirmPasswordField() {
    return CustomPasswordFormField(
      formKey: _passwordRepeatFormKey,
      title:
fieldTitle(S.current.repeatNewPasswordField),
      onChangedCallback: (value) =>
confirmPassword = value,
      autovalidateMode: autovalidateMode,
      validator: (String? value) {
        return validatePasswordRepeat(
            passwordRepeat: value, password:
newPassword);
      },
    );
  }

  void _saveChanges() {
    setState(() {
      autovalidateMode =
AutovalidateMode.onUserInteraction;
    });
    if
(_newPasswordFormKey.currentState!.validate() &&

_passwordRepeatFormKey.currentState!.validate()) {
      Navigator.pop(
          context,
          SavePasswordChanges(
              oldPassword: oldPassword,
              newPassword: newPassword,
              confirmPassword: confirmPassword));
    }
  }
}

class EditLoginWithBiometricsPage extends
StatefulWidget {
  const EditLoginWithBiometricsPage({Key? key}) :
super(key: key);

  @override
  _EditLoginWithBiometricsPageState createState()
=> _EditLoginWithBiometricsPageState();
}

class _EditLoginWithBiometricsPageState extends
State<EditLoginWithBiometricsPage> {
  final LocalAuthentication auth =
LocalAuthentication();

  loginWithBiometrics() async {
    final LocalAuthentication auth =
LocalAuthentication();
    bool authenticated = false;
    try {
      authenticated = await auth.authenticate(
          localizedReason: 'Let OS determine
authentication method',
          useErrorDialogs: true,
          stickyAuth: true);
    } on PlatformException catch (e) {
      print(e);
      return;
    }
    return authenticated;
  }

  AutovalidateMode autovalidateMode =
AutovalidateMode.disabled;
  bool isSwitched = false;

  @override
  void initState() {
    super.initState();
    getSwitchValues();
  }

  getSwitchValues() async {
    isSwitched = (await getSwitchState())!;
    setState(() {});
  }

  Future<bool> saveSwitchState(bool value) async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    prefs.setBool("switchState", value);
    print('Switch Value saved $value');
    return prefs.setBool("switchState", value);
  }

  Future<bool?> getSwitchState() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
```

```
    bool? isSwitched =
prefs.getBool("switchState");
    print(isSwitched);

    return isSwitched;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: designAppBar(
          title: 'Settings',
          leading: IconButton(
            icon: Icon(Icons.arrow_back, key:
Key('back-arrow-button')),
            onPressed: () =>
Navigator.pop(context),
          ),
        ),
        body: Container(
          padding:
EdgeInsets.symmetric(horizontal: 30, vertical:
20),
          child: Column(
            children: [
              _touchId(),
            ],
          ),
        )
    );
  }

  Widget _touchId() {
    return Row(
      mainAxisAlignment:
MainAxisAlignment.spaceBetween,
      children: [
        Padding(
          padding: const EdgeInsets.fromLTRB(5, 5,
5, 5),
          child: Icon(
            Icons.fingerprint,
            color: AppColors.darkAccentColor,
          ),
        ),
        Column(
          children: [
            Text('Biometrics'),
            Text('Log in app, using fingerprint'),
          ],
        ),
        Switch(
          activeColor: AppColors.darkAccentColor,
          value: isSwitched,
          onChanged: (value){
            setState(() {
              isSwitched = value;
              saveSwitchState(value);
              if(isSwitched == true){
                loginWithBiometrics();
              };
            });
          },
        ),
      ],
    );
  }
}



class EditProfileContent extends StatefulWidget {
  const EditProfileContent({Key? key}) :
super(key: key);

  @override
  _EditProfileContentState createState() =>
_EditProfileContentState();
}

class _EditProfileContentState extends
State<EditProfileContent> {
  late String firstName;
  late String lastName;
  late String email;
  late String phoneNumber;
  late String birthDate;

  List<bool> haveChanges = List.generate(5,
(index) => false);

  final GlobalKey<FormState> _firstNameFormKey =
      GlobalKey<FormState>(debugLabel: 'first name
key');
  final GlobalKey<FormState> _lastNameFormKey =
      GlobalKey<FormState>(debugLabel: 'last name
key');
  final GlobalKey<FormState> _emailFormKey =
      GlobalKey<FormState>(debugLabel: 'register
email key');
  final GlobalKey<FormState> _phoneNumberKey =
      GlobalKey<FormState>(debugLabel: 'phone
number key');

  AutovalidateMode autovalidateMode =
AutovalidateMode.disabled;
  String phoneNumberIsoCode = 'UA';

  @override
  Widget build(BuildContext context) {
    return WillPopScope(
      onWillPop: _showConfirmChangesAlertDialog,
      child: Scaffold(
        appBar: designAppBar(
            title: S.current.editProfileTitle,
            leading: IconButton(
              icon: Icon(Icons.arrow_back, key:
Key('back-arrow-button')),
              onPressed:
_showConfirmChangesAlertDialog,
            ),
            rightButtonLabel:
S.current.saveButton,
            onRightButtonPressed: _saveChanges),
        body: _editProfileBody(),
      ),
    );
  }

  Widget _editProfileBody() {
    return SingleChildScrollView(
      child: Container(
        padding: EdgeInsets.symmetric(horizontal:
30),
        child: Column(
          children: [
            SizedBox(height: 16),
            _profileAvatar(),
            SizedBox(height: 26),
            Row(
              children: [
                Flexible(child:
_firstNameField()),
                Flexible(child: _lastNameField()),
              ],
            ),
            _emailField(),
            _phoneNumberField(),
            _birthDateField(),
            SizedBox(height: 12),
            _editLoginBiometricsButton(),
            _editPasswordButton(),
            _editAddressButton(),
```

```
            _editFamilyMarriageStatusButton(),
            SizedBox(height: 12),
            _joinedLabel(),
            SizedBox(height: 12),
          ],
        ),
      ),
    );
  }

  Widget _profileAvatar() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      return ProfileAvatar(
        onClicked: () {

context.read<EditProfileBloc>().add(ChangeAvatarRe
quest());
        },
        imagePath: state.avatarUrl,
        changeable: true,
        size: 100,
      );
    });
  }

  Widget _firstNameField() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      firstName = state.firstName;
      return CustomTextFormField(
        key: Key('first-name-field'),
        initialValue: firstName,
        formKey: _firstNameFormKey,
        autovalidateMode: autovalidateMode,
        title:
fieldTitle(S.current.firstNameField),
        onChangedCallback: (value) {
          firstName = value;
          haveChanges[0] = firstName !=
state.firstName;
        },
        textInputType: TextInputType.name,
        validator: (String? value) {
          return validateName(name: value);
        },
        titleSpace: 2.0,
      );
    });
  }

  Widget _lastNameField() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      lastName = state.lastName;
      return CustomTextFormField(
        key: Key('last-name-field'),
        initialValue: lastName,
        formKey: _lastNameFormKey,
        autovalidateMode: autovalidateMode,
        title:
fieldTitle(S.current.lastNameField),
        onChangedCallback: (value) {
          lastName = value;
          haveChanges[1] = lastName !=
state.lastName;
        },
        textInputType: TextInputType.name,
        validator: (String? value) {
          return validateName(name: value);
        },
        titleSpace: 2.0,
      );
    });
  }

  Widget _emailField() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      email = state.email;
      return CustomTextFormField(
        key: Key('email-field'),
        initialValue: email,
        formKey: _emailFormKey,
        autovalidateMode: autovalidateMode,
        title: fieldTitle(S.current.emailField),
        onChangedCallback: (value) {
          email = value;
          haveChanges[2] = email != state.email;
        },
        textInputType: TextInputType.emailAddress,
        validator: (String? value) {
          return validateEmail(email: value);
        },
        titleSpace: 2.0,
      );
    });
  }

  Widget _phoneNumberField() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      phoneNumber = state.phoneNumber;
      return CustomPhoneNumberFormField(
        key: Key('phone-number-field'),
        initialValue: phoneNumber,
        isoCode: phoneNumberIsoCode,
        formKey: _phoneNumberKey,
        autovalidateMode: autovalidateMode,
        title:
fieldTitle(S.current.phoneNumberField),
        onChangedCallback: (value) {
          phoneNumber = value.toString();
          haveChanges[3] = phoneNumber !=
state.phoneNumber;
        },
        titleSpace: 2.0,
      );
    });
  }

  Widget _birthDateField() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      birthDate = state.birthDate ??
DateTime(2000).toString();
      return ChangeableDate(
        initialDate: DateTime.parse(birthDate),
        label: S.current.changeBirthDateLabel,
        onChangedCallback: (String value) {
          birthDate = value;
          haveChanges[4] = birthDate !=
state.birthDate;
        },
      );
    });
  }

  Widget _editPasswordButton() {
    return EditFieldButton(
      key: Key('password-edit-button'),
      onTap: () async {
        EditProfileEvent? resultEvent =
            await Navigator.pushNamed(context,
Routes.editPasswordPage)
                as EditProfileEvent?;
        if (resultEvent != null) {
```

```
context.read<EditProfileBloc>().add(resultEvent);
        }
      },
      title: S.current.changePasswordLabel,
    );
  }

  Widget _editAddressButton() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      return EditFieldButton(
        key: Key('address-edit-button'),
        onTap: () async {
          EditProfileEvent? resultEvent = await
Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) =>
EditAddressPage(
                    initPostalCode:
state.user.postalCode,
                    initStreetAddress:
state.user.streetAddress,
                    initStateAddress:
state.user.stateAddress,
                    initCityAddress:
state.user.cityAddress,
                ),
              ));
          if (resultEvent != null) {

context.read<EditProfileBloc>().add(resultEvent);
          }
        },
        title: S.current.changeAddressLabel,
      );
    });
  }

  Widget _editFamilyMarriageStatusButton() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      return EditFieldButton(
        key: Key('family-status-edit-button'),
        onTap: () async {
          EditProfileEvent? resultEvent = await
Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) =>
EditFamilyStatusPage(
                    initMarriageStatus:
state.user.marriageStatus,
                    initDependantsAmount:
state.user.dependantsAmount,
                ),
              ));
          if (resultEvent != null) {

context.read<EditProfileBloc>().add(resultEvent);
          }
        },
        title: S.current.changeFamilyStatusLabel,
      );
    });
  }

  Widget _editLoginBiometricsButton() {
    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
          return EditFieldButton(
            key: Key('login-biometrics-button'),
            onTap: () async {
              EditProfileEvent? resultEvent =
await Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) =>
EditLoginWithBiometricsPage(),
                  ));
              if (resultEvent != null) {

context.read<EditProfileBloc>().add(resultEvent);
              }
            },
            title:
S.current.changeLoginBiometricsLabel,
          );
        },
    );
  }

  Widget _joinedLabel() {
    var dateFormat = DateFormat("dd MMM, yyy");

    return BlocBuilder<EditProfileBloc,
EditProfileState>(
        builder: (context, state) {
      String formattedDate =

dateFormat.format(DateTime.parse(state.joinedDate)
);
      return Text(S.current.joinedLabel +
formattedDate);
    });
  }

  Future<bool> _showConfirmChangesAlertDialog()
async {
    if (haveChanges.any((element) => element)) {
      await showConfirmChangesAlertDialog(
          context: context,
          exitLabel: S.current.exitLabel,
          saveChangesLabel: S.current.saveLabel,
          title:
S.current.exitWithoutSavingChanges,
          onSavePressed: () {
            _saveChanges();
            Navigator.pop(context, true);
          },
          onExitPressed: () {
            Navigator.pop(context, true);
          });
    }
    Navigator.pop(context);
    return false;
  }

  void _saveChanges() {
    setState(() {
      autovalidateMode =
AutovalidateMode.onUserInteraction;
    });
    if (_firstNameFormKey.currentState!.validate()
&&
        _lastNameFormKey.currentState!.validate()
&&
        _emailFormKey.currentState!.validate() &&
        _phoneNumberKey.currentState!.validate())
{

context.read<EditProfileBloc>().add(SaveProfileCha
ngesPressed(
          firstName: firstName,
          lastName: lastName,
          email: email,
          birthDate: birthDate,
          phoneNumber: phoneNumber));
      haveChanges = List.generate(5, (index) =>
false);
```

```
    }
  }
}

class EditFamilyStatusPage extends StatefulWidget
{
  EditFamilyStatusPage(
      {Key? key,
      required this.initMarriageStatus,
      required this.initDependantsAmount})
      : super(key: key);

  final bool initMarriageStatus;
  final int initDependantsAmount;

  @override
  State<EditFamilyStatusPage> createState() =>
_EditFamilyStatusPageState();
}

class _EditFamilyStatusPageState extends
State<EditFamilyStatusPage> {
  late bool marriageStatus;
  late int dependantsAmount;

  @override
  void initState() {
    marriageStatus = widget.initMarriageStatus;
    dependantsAmount =
widget.initDependantsAmount;
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: designAppBar(
        title: S.current.changeFamilyStatusLabel,
        leading: IconButton(
          icon: Icon(Icons.arrow_back, key:
Key('back-arrow-button')),
          onPressed: () => Navigator.pop(context),
        ),
      ),
      body: _editMarriageStatusBody(),
    );
  }

  Widget _editMarriageStatusBody() {
    return SingleChildScrollView(
      child: Container(
        padding: EdgeInsets.symmetric(horizontal:
30, vertical: 30),
        child: Column(
          children: [
            _dependantAmountField(),
            SizedBox(height: 26),
            _marriageStatusChoiceField(),
            SizedBox(height: 12),
            DesignButton(
                onTap: _saveChanges,
                child: Text(
                  S.current.saveLabel,
                  style: radencyButtonTextStyle(),
                )),
          ],
        ),
      ),
    );
  }

  Widget _dependantAmountField() {
    return CustomTextFormField(
      initialValue: dependantsAmount.toString(),
      inputFormatters: [
FilteringTextInputFormatter.allow(RegExp(r'[0-
9]')),
      ],
      title:
fieldTitle(S.current.dependantsAmountField),
      textInputType:
          TextInputType.numberWithOptions(signed:
false, decimal: true),
      onChangedCallback: (value) {
        try {
          dependantsAmount = int.parse(value);
        } catch (_) {
          dependantsAmount = 0;
        }
      },
    );
  }

  Widget _marriageStatusChoiceField() {
    return Column(
      crossAxisAlignment:
CrossAxisAlignment.start,
      mainAxisAlignment: MainAxisAlignment.start,
      children: <Widget>[
        Container(
            padding:
EdgeInsets.symmetric(horizontal: 8),
            child:
Text(S.current.marriageStatusField,
                style: TextStyle(fontWeight:
FontWeight.bold))),
        ListTile(
          title: Text(S.current.marriedLabel),
          leading: Radio<bool>(
            value: true,
            groupValue: marriageStatus,
            onChanged: (bool? value) {
              setState(() => marriageStatus =
true);
            },
          ),
        ),
        ListTile(
          title: Text(S.current.unmarriedLabel),
          leading: Radio<bool>(
            value: false,
            groupValue: marriageStatus,
            onChanged: (bool? value) {
              setState(() => marriageStatus =
false);
            },
          ),
        ),
      ],
    );
  }

  void _saveChanges() {
    if (_dataIsChanged()) {
      Navigator.pop(
          context,
          SaveFamilyStatusChanges(
            marriageStatus: marriageStatus,
            dependantsAmount: dependantsAmount,
          ));
    } else {
      Navigator.pop(context);
    }
  }

  bool _dataIsChanged() {
    return widget.initMarriageStatus !=
marriageStatus ||
        widget.initDependantsAmount !=
dependantsAmount;
  }
```

```
}


class MenuWidget extends StatelessWidget {
  const MenuWidget({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(
      child: ListView(
        padding: EdgeInsets.zero,
        children: [
          MenuHeaderDrawer(),
          ListTile(
            title: MenuItemsWidget(
              icon: Icons.store,
              itemName: 'Merchants',
            ),
            onTap: () {
              Navigator.pushNamed(context,
Routes.merchantsPage);
            },
          ),
          ListTile(
            title: MenuItemsWidget(
              icon: Icons.account_balance,
              itemName: 'Dashboard',
            ),
            onTap: () {
              Navigator.pushNamed(context,
Routes.dashboardPage);
            },
          ),
          ListTile(
            title: MenuItemsWidget(
              icon: Icons.shopping_cart,
              itemName: 'Orders',
            ),
            onTap: () {
              Navigator.pushNamed(context,
Routes.ordersPage);
            },
          ),
          SizedBox(height: 28),
          ListTile(
            title: MenuItemsWidget(
              icon: Icons.logout,
              itemName: 'Log out',
            ),
            onTap: () {

context.read<AuthenticationBloc>().add(Authenticat
ionLogOut());
            },
          ),
        ],
      ),
    );
  }
}

class MenuItemsWidget extends StatelessWidget {
  final IconData icon;
  final String itemName;

  MenuItemsWidget({
    required this.icon,
    required this.itemName,
  });

  @override
  Widget build(BuildContext context) {
    return Container(
      child: Row(
        children: [
          Icon(icon, color:
AppColors.darkAccentColor),
          Padding(
            padding: const EdgeInsets.fromLTRB(7,
0, 0, 0),
            child: Text(
              itemName,
              style: TextStyle(color:
Colors.black.withOpacity(0.6)),
            ),
          )
        ],
      ),
    );
  }
}

class MenuHeaderDrawer extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      child: DrawerHeader(
        decoration:
designBoxDecorator(hasGradient: true,
hasCircularBorders: false),
        child: Container(
          width: double.infinity,
          alignment: Alignment.centerLeft,
          child: BlocBuilder<AuthenticationBloc,
AuthenticationState>(
              builder: (context, state) {
            if (state is
AuthenticationAuthenticated) {
              return GestureDetector(
                onTap: () =>
                    Navigator.pushNamed(context,
Routes.editProfilePage),
                child: Column(
                  children: [
                    Row(
                      mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                      children: [
                        Text(
                          'Your profile',
                          key: const Key('your-
profile-key'),
                          style: TextStyle(
                            color:
Colors.white.withOpacity(0.9),
                              fontSize: 18,
                          ),
                        ),
                        Icon(
                          Icons.edit_outlined,
                          color:
AppColors.lightContrastColor,
                        ),
                      ],
                    ),
                    Container(
                        padding: const
EdgeInsets.fromLTRB(0, 10, 0, 0),
                        alignment:
Alignment.centerLeft,
                        child:
ProfileAvatar(imagePath: state.user.avatar)),
                    Padding(
                      padding: const
EdgeInsets.fromLTRB(0, 10, 0, 0),
                      child: Align(
                        alignment:
Alignment.bottomLeft,
                        child: Container(
                          child: Text(
                            state.user.username,
                            style: TextStyle(
                              color:
Colors.white.withOpacity(0.9),
```

```
                            fontSize: 15,
                          ),
                        ),
                      ),
                    ),
                  ),
                ],
              ),
            );
          } else {
            return SizedBox();
          }
        }),
      ),
    ),
  );
 }
}


class OrdersPage extends StatefulWidget {
  const OrdersPage({Key? key}) : super(key: key);

  @override
  _OrdersPageState createState() =>
_OrdersPageState();
}

class _OrdersPageState extends State<OrdersPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      drawer: Drawer(
        child: MenuWidget(),
      ),
      appBar: designAppBar(
        title: 'Your orders',
        actions: <Widget>[
          IconButton(
            icon: Icon(Icons.search),
            onPressed: () {},
          ),
          IconButton(
            icon:
Icon(Icons.shopping_cart_outlined),
            onPressed: () {},
          ),
        ],
      ),
      body: FutureBuilder<List<Order>>(
        future:
UserDataRepository().getProducts(),
        builder: (BuildContext context,
AsyncSnapshot<List<Order>> snapshot) {
          if (!snapshot.hasData) {
           return Container(
             child: Text('Error'),
           );
          } else {
            return Container(
                padding: const
EdgeInsets.fromLTRB(0, 11, 0, 8),
                child: ListView.separated(
                  itemCount: orders.length,
                  separatorBuilder: (_, __) =>
SizedBox(height: 10),
                  itemBuilder: (context, index) {
                    return OrderWidget(
                      name: orders[index].name,
                      image_url:
orders[index].image_url,
                      price: orders[index].price,
                      created_at:
orders[index].created_at,
                      numberOfPayments:
orders[index].numberOfPayments,
                    );
                  },
                ),
              );
            }
          },
        ),
      );
  }
}

class OrderWidget extends StatelessWidget {
  String name;
  String image_url;
  int price;
  DateTime created_at;
  int numberOfPayments;

  OrderWidget({
    required this.numberOfPayments,
    required this.number,
    required this.image_url,
    required this.price,
    required this.created_at,
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        Navigator.pushNamed(context,
Routes.orderTransactions, arguments: {
          'name': name.capitalize(),
          'image': _getImage(60),
          'price': price,
          'createdAt': created_at,
          'numberOfPayments': numberOfPayments,
        });
      },
      child: Container(
        margin: EdgeInsets.symmetric(horizontal:
14),
        padding: EdgeInsets.symmetric(vertical:
14, horizontal: 8),
        decoration: _boxDecoration(),
        child: Stack(
          children: <Widget>[
            Container(
              child: Row(
                mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                children: [
                  _orderImage(),
                  _orderNameAndDate(),
                  _orderPriceAndDiscount(),
                ],
              ),
            ),
          ],
        ),
      ),
    );
  }

  Widget _orderImage() {
    return Container(
      margin: const EdgeInsets.fromLTRB(10, 2, 10,
2),
      child: _getImage(80)
    );
  }

  Widget _getImage(double size){
    return CachedNetworkImage(
      width: size,
      height: size,
      fit: BoxFit.cover,
      imageUrl: image_url,
```

```
      placeholder: (context, url) =>
          Center(child:
CircularProgressIndicator()),
      errorWidget: (context, url, error) => Icon(
        Icons.image,
        size: 35,
        color: AppColors.accentColor,
      ),
    );
  }

  Widget _orderNameAndDate() {
    return Column(
      children: [
        Padding(
          padding: const EdgeInsets.only(bottom:
8),
          child: Text(
            name.capitalize(),
            style: TextStyle(fontSize: 16,
fontWeight: FontWeight.bold),
          ),
        ),
        Text(
          created_at.day.toString() +
              '-' +
              created_at.month.toString() +
              '-' +
              created_at.year.toString(),
          style: TextStyle(color:
Colors.black.withOpacity(0.7)),
        ),
      ],
    );
  }

  Widget _orderPriceAndDiscount() {
    return Padding(
      padding: const EdgeInsets.only(right: 15),
      child: Row(
        children: [
          Text(
            price.toString() + ' \$',
            style: TextStyle(
                fontWeight: FontWeight.bold,
                fontSize: 18,
                color: Colors.orange),
          ),
          Icon(
            Icons.chevron_right,
            size: 28,
          ),
          // Text('Your discount is: ' +
discount.toString() + '\$'),
        ],
      ),
    );
  }

  Decoration _boxDecoration() {
    return BoxDecoration(
      border: Border.all(width: 1.0, color:
Colors.grey.shade400),
      color: Colors.white,
      borderRadius:
BorderRadius.all(Radius.circular(8)),
      boxShadow: [
        BoxShadow(
          color: Colors.grey.withOpacity(0.5),
          blurRadius: 8,
          offset: Offset(0, 2),
        ),
      ],
    );
  }
}
```

```
class OrderTransactionsPage extends StatefulWidget
{
  const OrderTransactionsPage({Key? key}) :
super(key: key);

  @override
  _OrderTransactionsPageState createState() =>
_OrderTransactionsPageState();
}

class _OrderTransactionsPageState extends
State<OrderTransactionsPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: designAppBar(
        title: S.current.orderTransactionsTitle,
        leading: IconButton(
          icon: Icon(Icons.arrow_back, key:
Key('back-arrow-button')),
          onPressed: ()=> Navigator.pop(context),
        ),
      ),
      body: orderTransactionsBody(),
    );
  }

  Widget orderTransactionsBody() {
    return MultiBlocProvider(
      providers: [
        BlocProvider(
            create: (context) =>

OrderTransactionsBloc()..add(RefreshOrderTransacti
ons())),
      ],
      child: BlocBuilder<OrderTransactionsBloc,
OrderTransactionsState>(
        builder: (context, state) {
          switch (state.status) {
            case OrderTransactionsStatus.failure:
              return RefreshButton(
                  onPressed: () => setState(() {

context.read<OrderTransactionsBloc>()

..add(RefreshButtonPressed());
                    }),
                  label: S.current.loadingFailed);
            case OrderTransactionsStatus.success:
              return
orderTransactionsContent(state.orderTransactions!)
;
            default:
              return const Center(child:
CircularProgressIndicator());
          }
        },
      ),
    );
  }

  Widget
orderTransactionsContent(List<OrderTransactions>
orderTransactions) {
    final arguments =
ModalRoute.of(context)!.settings.arguments as Map;

    return Column(
      children: [
        NumberTitledBox(
            number: arguments['price'].toString(),
            label: S.current.totalLeftToPayLabel,
            scale: 0.4,
            textColor: Colors.white,
            hasIndicator: true,
```

```
            currentPayment:
arguments['numberOfPayments'] - 2,
            numberOfPayments:
arguments['numberOfPayments']),
        SizedBox(height: 10),
        Expanded(
            child: ListView.builder(
          itemCount:
arguments['numberOfPayments'],
          itemBuilder: (context, index) {
            print(arguments['numberOfPayments']);
            return Column(
              children: [

DateListHeader(orderTransactions[index].paymentDat
es,
                    todayHeader:
S.current.todayHeader,
                    mark: S.current.upcomingLabel,
                    isPaid:
orderTransactions[index].isPaid),
                TransactionListItem(
                  image: arguments['image'],
                  title: '${arguments['name']}
(${orderTransactions[0].brandName})',
                  subtitle:
S.current.loanRateLabel +
_getLoanRate(arguments['numberOfPayments'],
index),
                  price: (arguments['price'] /
arguments['numberOfPayments']).toStringAsFixed(2),
                  isPaid:
orderTransactions[index].isPaid,
                ),
                SizedBox(height: 10),
              ],
            );
          },
        )),
      ],
    );
  }
  String _getLoanRate(int total, int index){
    return ': ${total - index}/$total';
  }
}


class AppView extends StatefulWidget {
  const AppView({Key? key}) : super(key: key);

  @override
  _AppViewState createState() => _AppViewState();
}

class _AppViewState extends State<AppView> {
  final _navigatorKey =
GlobalKey<NavigatorState>();

  NavigatorState? get _navigator =>
_navigatorKey.currentState;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      navigatorKey: _navigatorKey,
      localizationsDelegates: [
        S.delegate,
        GlobalMaterialLocalizations.delegate,
        GlobalWidgetsLocalizations.delegate,
        GlobalCupertinoLocalizations.delegate,
      ],
      supportedLocales: [
        const Locale('en', ''),
      ],
      routes: {
        Routes.splashScreen: (context) =>
SplashScreen(),
        Routes.loginPage: (context) =>
LoginPage(),
        Routes.registerPage: (context) =>
RegisterPage(),
        Routes.restorePasswordPage: (context) =>
RestorePasswordPage(),
        Routes.forgotPasswordPage: (context) =>
ForgotPasswordPage(),
        Routes.merchantsPage: (context) =>
MerchantsPage(),
        Routes.dashboardPage: (context) =>
DashboardPage(),
        Routes.orderTransactions: (context) =>
OrderTransactionsPage(),
        Routes.editProfilePage: (context) =>
EditProfilePage(),
        Routes.editPasswordPage: (context) =>
EditPasswordPage(),
        Routes.ordersPage: (context) =>
OrdersPage(),
      },
      builder: (context, child) {
        return MultiBlocListener(
          listeners: [
            BlocListener<AuthenticationBloc,
AuthenticationState>(
              listener: (context, state) {
                print("her");
                if (state is
AuthenticationAuthenticated) {

_navigator?.pushNamedAndRemoveUntil(
                      Routes.merchantsPage,
(route) => false);
                } else if (state is
AuthenticationUnauthenticated) {

_navigator?.pushNamedAndRemoveUntil(
                      Routes.merchantsPage,
(route) => false);
                } else {

_navigator?.pushNamedAndRemoveUntil(
                      Routes.splashScreen, (route)
=> false);
                }
              },
            ),
          ],
          child: child!,
        );
      },
    );
  }
}

class ForgotPasswordPage extends StatefulWidget {
  const ForgotPasswordPage({Key? key}) :
super(key: key);

  @override
  _ForgotPasswordPageState createState() =>
_ForgotPasswordPageState();
}

class _ForgotPasswordPageState extends
State<ForgotPasswordPage> {
  String email = '';

  final GlobalKey<FormState> _emailFormKey =
      GlobalKey<FormState>(debugLabel:
'forgotPassword email key');

  AutovalidateMode autovalidateMode =
AutovalidateMode.disabled;
```

```
  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) => ForgotPasswordBloc(),
      child: Scaffold(
        body: forgotPasswordContents(),
      ),
    );
  }

  Widget forgotPasswordContents() {
    return BlocListener<ForgotPasswordBloc,
ForgotPasswordState>(
      listener: (context, state) {
        if (state.errorMessage != null) {
          ScaffoldMessenger.of(context)
            ..hideCurrentSnackBar()
            ..showSnackBar(
              SnackBar(content:
Text(state.errorMessage!)),
            );
        }
      },
      child: CenteredScrollView(
        child: Column(
          children: [

pageTitle(S.current.forgotPasswordScreenTitle),
            emailField(),
            sendRestoreLinkButton(),
            registerRedirect(),
            loginRedirect(),
          ],
        ),
      ),
    );
  }

  Widget emailField() {
    return BlocBuilder<ForgotPasswordBloc,
ForgotPasswordState>(
      builder: (context, state) {
        return CustomTextFormField(
          formKey: _emailFormKey,
          autovalidateMode: autovalidateMode,
          title: fieldTitle(S.current.emailField,
showFieldRequired: false),
          onChangedCallback: (value) => email =
value,
          textInputType:
TextInputType.emailAddress,
          validator: (String? value) {
            return validateEmail(email: value);
          },
        );
      },
    );
  }

  Widget sendRestoreLinkButton() {
    return BlocBuilder<ForgotPasswordBloc,
ForgotPasswordState>(
      builder: (context, state) {
        return DesignButton(
          isLoading: state.isLoading,
          child: Text(

S.current.sendRestoreLinkButton.toUpperCase(),
            style: radencyButtonTextStyle(),
          ),
          onTap: () {
            setState(() {
              autovalidateMode =
AutovalidateMode.onUserInteraction;
            });
            if
(_emailFormKey.currentState!.validate()) {
              context
                  .read<ForgotPasswordBloc>()

.add(SendRestoreLinkPressed(email: email));
            }
          },
        );
      },
    );
  }

  Widget registerRedirect() {
    return Wrap(
      crossAxisAlignment:
WrapCrossAlignment.center,
      children: [
        Text(S.current.registerRedirectQuestion),
        SizedBox(
          width: 5,
        ),
        Container(
          child: TextButton(
              style: TextButton.styleFrom(
                padding: EdgeInsets.zero,
              ),
              onPressed: () {
                Navigator.pushNamed(context,
Routes.registerPage);
              },
              child:
Text(S.current.registerRedirectButton)),
        )
      ],
    );
  }

  Widget loginRedirect() {
    return Wrap(
      crossAxisAlignment:
WrapCrossAlignment.center,
      children: [
        Text(S.current.logInRedirectQuestion),
        SizedBox(width: 5),
        Container(
          child: TextButton(
              style: TextButton.styleFrom(padding:
EdgeInsets.zero),
              onPressed: () {
                Navigator.pushNamed(context,
Routes.loginPage);
              },
              child:
Text(S.current.logInRedirectButton)),
        )
      ],
    );
  }
}

class RestorePasswordPage extends StatefulWidget {
  const RestorePasswordPage({Key? key}) :
super(key: key);

  @override
  _RestorePasswordPageState createState() =>
_RestorePasswordPageState();
}

class _RestorePasswordPageState extends
State<RestorePasswordPage> {
  String password = '';
  String passwordRepeat = '';

  final GlobalKey<FormState> _passwordFormKey =
```

```
      GlobalKey<FormState>(debugLabel:
'restorePassword password key');
  final GlobalKey<FormState>
_passwordRepeatFormKey =
      GlobalKey<FormState>(debugLabel:
'restorePassword password repeat key');

  AutovalidateMode autovalidateMode =
AutovalidateMode.disabled;

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) => RestorePasswordBloc(
          authenticationBloc:
context.read<AuthenticationBloc>()),
      child: Scaffold(
        body: restorePasswordContents(),
      ),
    );
  }

  Widget restorePasswordContents() {
    return BlocListener<RestorePasswordBloc,
RestorePasswordState>(
      listener: (context, state) {
        if (state.errorMessage != null) {
          ScaffoldMessenger.of(context)
            ..hideCurrentSnackBar()
            ..showSnackBar(
              SnackBar(content:
Text(state.errorMessage!)),
            );
        }
      },
      child: CenteredScrollView(
        child: Column(
          children: [

pageTitle(S.current.resetPasswordScreenTitle),
            passwordField(),
            passwordRepeatField(),
            confirmPasswordButton(),
          ],
        ),
      ),
    );
  }

  Widget passwordField() {
    return BlocBuilder<RestorePasswordBloc,
RestorePasswordState>(
      builder: (context, state) {
        return CustomPasswordFormField(
          formKey: _passwordFormKey,
          title:
fieldTitle(S.current.passwordField,
showFieldRequired: true),
          onChangedCallback: (value) => password =
value,
          autovalidateMode: autovalidateMode,
          validator: (String? value) {
            return validatePassword(
              password: value,
              shouldValidateComplexity: true,
            );
          },
        );
      },
    );
  }

  Widget passwordRepeatField() {
    return BlocBuilder<RestorePasswordBloc,
RestorePasswordState>(
      builder: (context, state) {
        return CustomPasswordFormField(
          formKey: _passwordRepeatFormKey,
          title:
fieldTitle(S.current.passwordRepeatField,
              showFieldRequired: true),
          onChangedCallback: (value) =>
passwordRepeat = value,
          autovalidateMode: autovalidateMode,
          validator: (String? value) {
            return validatePasswordRepeat(
                passwordRepeat: value, password:
password);
          },
        );
      },
    );
  }

  Widget confirmPasswordButton() {
    return BlocBuilder<RestorePasswordBloc,
RestorePasswordState>(
      builder: (context, state) {
        return DesignButton(
          isLoading: state.isLoading,
          child: Text(

S.current.confirmPasswordButton.toUpperCase(),
            style: radencyButtonTextStyle(),
          ),
          onTap: () {
            setState(() {
              autovalidateMode =
AutovalidateMode.onUserInteraction;
            });

            if
(_passwordFormKey.currentState!.validate() &&

_passwordRepeatFormKey.currentState!.validate()) {

context.read<RestorePasswordBloc>().add(ConfirmNew
PasswordPressed(
                  password: password,
confirmPassword: passwordRepeat));
            }
          },
        );
      },
    );
  }
}

class RegisterPage extends StatefulWidget {
  const RegisterPage({Key? key}) : super(key:
key);

  @override
  _RegisterPageState createState() =>
_RegisterPageState();
}

class _RegisterPageState extends
State<RegisterPage> {
  String firstName = '';
  String lastName = '';
  String email = '';
  String phoneNumber = '';
  String password = '';
  String passwordRepeat = '';

  final GlobalKey<FormState> _firstNameFormKey =
      GlobalKey<FormState>(debugLabel: 'first name
key');
  final GlobalKey<FormState> _lastNameFormKey =
      GlobalKey<FormState>(debugLabel: 'last name
key');
  final GlobalKey<FormState> _emailFormKey =
```

```
      GlobalKey<FormState>(debugLabel: 'register
email key');
  final GlobalKey<FormState> _phoneNumberKey =
      GlobalKey<FormState>(debugLabel: 'phone
number key');
  final GlobalKey<FormState> _passwordFormKey =
      GlobalKey<FormState>(debugLabel: 'register
password key');
  final GlobalKey<FormState>
_passwordRepeatFormKey =
      GlobalKey<FormState>(debugLabel: 'register
password repeat key');

  AutovalidateMode autovalidateMode =
AutovalidateMode.disabled;
  String phoneNumberIsoCode = 'UA';

  @override
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) =>
          RegisterBloc(),
      child: Scaffold(
        body: registerContents(),
      ),
    );
  }

  Widget registerContents() {
    return BlocListener<RegisterBloc,
RegisterState>(
      listener: (context, state) {
        if (state.errorMessage != null) {
          ScaffoldMmessenger.of(context)
            ..hideCurrentSnackBar()
            ..showSnackBar(SnackBar(content:
Text(state.errorMessage!)));
        }
      },
      child: CenteredScrollView(
        child: Column(
          children: [

pageTitle(S.current.registerScreenTitle),
            registerForm(),
            registerButton(),
            loginRedirect(),
          ],
        ),
      ),
    );
  }

  Widget registerForm() {
    return Column(
      children: [
        firstNameField(),
        lastNameField(),
        emailField(),
        phoneNumberField(),
        passwordField(),
        passwordRepeatField(),
      ],
    );
  }

  Widget firstNameField() {
    return BlocBuilder<RegisterBloc,
RegisterState>(
      builder: (context, state) {
        return CustomTextFormField(
          formKey: _firstNameFormKey,
          autovalidateMode: autovalidateMode,
          title:
fieldTitle(S.current.firstNameField,
showFieldRequired: true),
          onChangedCallback: (value) => firstName
= value,
          textInputType: TextInputType.name,
          validator: (String? value) {
            return validateName(name: value);
          },
        );
      },
    );
  }

  Widget lastNameField() {
    return BlocBuilder<RegisterBloc,
RegisterState>(
      builder: (context, state) {
        return CustomTextFormField(
          formKey: _lastNameFormKey,
          autovalidateMode: autovalidateMode,
          title:
fieldTitle(S.current.lastNameField,
showFieldRequired: true),
          onChangedCallback: (value) => lastName =
value,
          textInputType: TextInputType.name,
          validator: (String? value) {
            return validateName(name: value);
          },
        );
      },
    );
  }

  Widget emailField() {
    return BlocBuilder<RegisterBloc,
RegisterState>(
      builder: (context, state) {
        return CustomTextFormField(
          formKey: _emailFormKey,
          autovalidateMode: autovalidateMode,
          title: fieldTitle(S.current.emailField,
showFieldRequired: true),
          onChangedCallback: (value) => email =
value,
          textInputType:
TextInputType.emailAddress,
          validator: (String? value) {
            return validateEmail(email: value);
          },
        );
      },
    );
  }

  Widget phoneNumberField() {
    return BlocBuilder<RegisterBloc,
RegisterState>(
      builder: (context, state) {
        return CustomPhoneNumberFormField(
          formKey: _phoneNumberKey,
          isoCode: phoneNumberIsoCode,
          autovalidateMode: autovalidateMode,
          title:

fieldTitle(S.current.phoneNumberField,
showFieldRequired: true),
          onChangedCallback: (value) =>
phoneNumber = value.toString(),
          errorMessage:
S.current.phoneNumberValidatorIncorrect,
        );
      },
    );
  }

  Widget passwordField() {
    return BlocBuilder<RegisterBloc,
RegisterState>(
```

```
      builder: (context, state) {
        return CustomPasswordFormField(
          formKey: _passwordFormKey,
          title:
fieldTitle(S.current.passwordField,
showFieldRequired: true),
          onChangedCallback: (value) => password =
value,
          autovalidateMode: autovalidateMode,
          validator: (String? value) {
            return validatePassword(
              password: value,
              shouldValidateComplexity: true,
            );
          },
        );
      },
    );
  }

  Widget passwordRepeatField() {
    return BlocBuilder<RegisterBloc,
RegisterState>(
      builder: (context, state) {
        return CustomPasswordFormField(
          formKey: _passwordRepeatFormKey,
          title:
fieldTitle(S.current.passwordRepeatField,
              showFieldRequired: true),
          onChangedCallback: (value) =>
passwordRepeat = value,
          autovalidateMode: autovalidateMode,
          validator: (String? value) {
            return validatePasswordRepeat(
                passwordRepeat: value, password:
password);
          },
        );
      },
    );
  }

  Widget registerButton() {
    return BlocBuilder<RegisterBloc,
RegisterState>(
      builder: (context, state) {
        return DesignButton(
          isLoading: state.isLoading,
          child: Text(

S.current.registerButton.toUpperCase(),
            style: radencyButtonTextStyle(),
          ),
          onTap: () {
            setState(() {
              autovalidateMode =
AutovalidateMode.onUserInteraction;
            });

            if
(_firstNameFormKey.currentState!.validate() &&

_lastNameFormKey.currentState!.validate() &&

_emailFormKey.currentState!.validate() &&

_phoneNumberKey.currentState!.validate() &&

_passwordFormKey.currentState!.validate() &&

_passwordRepeatFormKey.currentState!.validate()) {

context.read<RegisterBloc>().add(RegisterPressed(
                  firstName: firstName,
                  lastName: lastName,
                  email: email,
                  phoneNumber: phoneNumber,
                  password: password,
                  confirmPassword:
passwordRepeat));
            }
          },
        );
      },
    );
  }

  Widget loginRedirect() {
    return Wrap(
      crossAxisAlignment:
WrapCrossAlignment.center,
      children: [
        Text(S.current.logInRedirectQuestion),
        SizedBox(width: 5),
        Container(
          child: TextButton(
              style: TextButton.styleFrom(padding:
EdgeInsets.zero),
              onPressed: () {
                Navigator.pushNamed(context,
Routes.loginPage);
              },
              child:
Text(S.current.logInRedirectButton)),
        )
      ],
    );
  }
}

class LoginPage extends StatefulWidget {
  const LoginPage({Key? key}) : super(key: key);

  @override
  _LoginPageState createState() =>
_LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  String email = '';
  String password = '';

  var isSwitched = false;

  void initState() {
    super.initState();
    getSwitchValues();
  }

  getSwitchValues() async {
    isSwitched = (await getSwitchState())!;
    setState(() {});
  }

  Future<bool?> getSwitchState() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    bool? isSwitched =
prefs.getBool("switchState");

    return isSwitched;
  }

  final GlobalKey<FormState> _emailFormKey =
      GlobalKey<FormState>(debugLabel: 'login
email key');
  final GlobalKey<FormState> _passwordFormKey =
      GlobalKey<FormState>(debugLabel: 'login
password key');

  AutovalidateMode autovalidateMode =
AutovalidateMode.disabled;

  @override
```

```
  Widget build(BuildContext context) {
    return BlocProvider(
      create: (context) =>
          LoginBloc(authenticationBloc:
context.read<AuthenticationBloc>()),
      child: Scaffold(
        body: loginContents(),
      ),
    );
  }

  Widget loginContents() {
    return BlocListener<LoginBloc, LoginState>(
      listener: (context, state) {
        if (state.message != null) {
          ScaffoldMessenger.of(context)
            ..hideCurrentSnackBar()
            ..showSnackBar(
              SnackBar(
                  key: const Key('login-snack-
bar'),
                  content: Text(state.message!)),
            );
        }
      },
      child: CenteredScrollView(
        child: Column(
          children: [
            pageTitle(S.current.loginScreenTitle),
            loginForm(),
            loginButton(),
            if (isSwitched == true)
biometricsForm(),
            registerRedirect(),
          ],
        ),
      ),
    );
  }

  Widget loginForm() {
    return Column(
      children: [
        emailField(),
        passwordField(),
      ],
    );
  }

  Widget emailField() {
    return BlocBuilder<LoginBloc, LoginState>(
      builder: (context, state) {
        return CustomTextFormField(
          key: const Key('email-login-form-
field'),
          formKey: _emailFormKey,
          autovalidateMode: autovalidateMode,
          title: fieldTitle(S.current.emailField,
showFieldRequired: false),
          onChangedCallback: (value) => email =
value,
          textInputType:
TextInputType.emailAddress,
          validator: (String? value) {
            return validateEmail(email: value);
          },
        );
      },
    );
  }

  Widget passwordField() {
    return BlocBuilder<LoginBloc, LoginState>(
      builder: (context, state) {
        return CustomPasswordFormField(
          key: const Key('password-login-form-
field'),
          formKey: _passwordFormKey,
          title: Row(
            mainAxisAlignment:
MainAxisAlignment.spaceBetween,
            children: [
              fieldTitle(S.current.passwordField,
showFieldRequired: false),
              Container(
                height: 20,
                child: TextButton(
                  key: const Key('forgot-password-
redirect-button'),
                  style: TextButton.styleFrom(
                    padding: EdgeInsets.zero,
                  ),
                  child:
Text(S.current.forgotPasswordButton),
                  onPressed: () {
                    Navigator.pushNamed(context,
Routes.forgotPasswordPage);
                  },
                ),
              ),
            ],
          ),
          onChangedCallback: (value) => password =
value,
          autovalidateMode: autovalidateMode,
          validator: (String? value) {
            return validatePassword(
              password: value,
              shouldValidateComplexity: false,
            );
          },
        );
      },
    );
  }

  Widget loginButton() {
    return BlocBuilder<LoginBloc, LoginState>(
      builder: (context, state) {
        return DesignButton(
          key: const Key('login-button'),
          isLoading: state.isLoading,
          child: Text(
            S.current.loginButton.toUpperCase(),
            style: radencyButtonTextStyle(),
          ),
          onTap: () {
            setState(() {
              autovalidateMode =
AutovalidateMode.onUserInteraction;
            });

            if
(_emailFormKey.currentState!.validate() &&

_passwordFormKey.currentState!.validate()) {
              context
                  .read<LoginBloc>()
                  .add(LoginPressed(email: email,
password: password));
            }
          },
        );
      },
    );
  }

  Widget biometricsForm() {
    return BlocBuilder<LoginBloc,
LoginState>(builder: (context, state) {
      return DesignButton(
          isLoading: state.isLoading,
          child: Icon(Icons.fingerprint),
          onTap: () async {
```

```
            final result = await
loginWithBiometrics();
            setState(() {
              autovalidateMode =
AutovalidateMode.onUserInteraction;
            });
            if (result == true) {
              context
                  .read<LoginBloc>()
                  .add(LoginPressed(email: email,
password: password));
            } else {

ScaffoldMessenger.of(context).showSnackBar(SnackBa
r(
                  content: Text(
                      'Failed to login with
biometrics. Please login manually.')));
            }
          });
    });
  }

  loginWithBiometrics() async {
    final LocalAuthentication auth =
LocalAuthentication();
    bool authenticated = false;
    try {
      authenticated = await auth.authenticate(
          localizedReason: 'Let OS determine
authentication method',
          useErrorDialogs: true,
          stickyAuth: true);
    } on PlatformException catch (e) {
      print(e);
      return;
    }
    if (authenticated) {
      final sharedPreferences = await
SharedPreferences.getInstance();
      email =
sharedPreferences.getString('email')!;
      password =
sharedPreferences.getString('password')!;
      if (email == null || password == null) {
        print('very bad reqest');
        return false;
      }
      return true;
    }
  }

  Widget registerRedirect() {
    return Wrap(
      crossAxisAlignment:
WrapCrossAlignment.center,
      children: [
        Text(S.current.registerRedirectQuestion),
        SizedBox(width: 5),
        Container(
          child: TextButton(
              key: const Key('register-redirect-
from-login'),
              style: TextButton.styleFrom(
                padding: EdgeInsets.zero,
              ),
              onPressed: () {
                Navigator.pushNamed(context,
Routes.registerPage);
              },
              child:
Text(S.current.registerRedirectButton)),
        )
      ],
    );
  }
}
```

```
PreferredSizeWidget designAppBar({
  String title = "",
  Widget? titleWidget,
  Widget? leading,
  String? rightButtonLabel,
  VoidCallback? onRightButtonPressed,
  bool rightBtnIsLoading = false,
  PreferredSizeWidget? bottom,
  List<Widget>? actions,
}) {
  var actionsButton = rightButtonLabel != null
      ? [
          GestureDetector(
            onTap: rightBtnIsLoading ? null :
onRightButtonPressed,
            child: Container(
              key: Key('save-button'),
              margin:
EdgeInsets.symmetric(vertical: 8, horizontal: 2),
              width: 80,
              alignment: Alignment.center,
              decoration:
designBoxDecorator(hasGradient: true),
              child: rightBtnIsLoading
                  ? CircularProgressIndicator()
                  : Text(rightButtonLabel),
            ),
          )
        ]
      : null;

  return AppBar(
    iconTheme: IconThemeData(color:
AppColors.darkAccentColor),
    shadowColor: AppColors.accentColor,
    elevation: 1,
    leading: leading,
    bottom: bottom,
    actions: actionsButton ?? actions,
    backgroundColor: AppColors.backgroundColor,
    title: titleWidget == null
        ? Text(title, style: TextStyle(color:
AppColors.defTextColor))
        : titleWidget,
    centerTitle: true,
  );
}

class BottomLoader extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.symmetric(vertical: 10),
      alignment: Alignment.center,
      child: const SizedBox(
        height: 32,
        width: 32,
        child:
CircularProgressIndicator(strokeWidth: 2.5),
      ),
    );
  }
}

class DateListHeader extends StatelessWidget {
  const DateListHeader(this.date,
      {Key? key, required this.todayHeader,
this.mark, this.isPaid = true})
      : super(key: key);

  final DateTime date;
  final String todayHeader;
  final String? mark;
  final bool isPaid;

  @override
```

```
  Widget build(BuildContext context) {
    return Container(
      padding: EdgeInsets.symmetric(vertical: 8,
horizontal: 20),
      alignment: Alignment.centerLeft,
      child: Text(
        formattedDate,
        style: TextStyle(
          color: Colors.grey.shade700,
        ),
      ),
    );
  }

  String get formattedDate {
    final today =
        DateTime(DateTime.now().year,
DateTime.now().month, DateTime.now().day);
    if (date.isAtSameMomentAs(today)) return
todayHeader;
    final formatter = DateFormat('MMM. dd, yyyy');
    String formattedDate = formatter.format(date);
    return isPaid ? formattedDate :
'$formattedDate - $mark';
  }
}


class TransactionListItem extends StatelessWidget
{
  const TransactionListItem({
    Key? key,
    this.icon,
    this.image,
    required this.title,
    required this.subtitle,
    required this.price,
    this.isPaid = true,
  }) : super(key: key);

  final IconData? icon;
  final Widget? image;
  final String title;
  final String subtitle;
  final String price;
  final bool isPaid;
  static const double iconSize = 30.0;

  Color get _textColor => isPaid ? Colors.black :
Colors.grey;

  @override
  Widget build(BuildContext context) {
    return Container(
      margin: EdgeInsets.symmetric(horizontal:
14),
      padding: EdgeInsets.symmetric(vertical: 14,
horizontal: 8),
      decoration: designBoxDecorator(),
      child: Row(
        children: <Widget>[
          Container(
            child: _iconWidget(),
            padding:
EdgeInsets.symmetric(horizontal: 10),
            alignment: Alignment.center,
          ),
          Expanded(
            child: Container(
              padding:
EdgeInsets.symmetric(horizontal: 4),
              child: Column(
                crossAxisAlignment:
CrossAxisAlignment.start,
                children: <Widget>[
                  Text(subtitle,
                      style: TextStyle(color:
_textColor, fontSize: 20.0)),
                  SizedBox(height: 6),
                  Text(title,
                      style: TextStyle(fontSize:
14.0, color: _textColor)),
                ],
              ),
            ),
          ),
          Text('$price \$ ',
              textScaleFactor: 1.7,
              style: TextStyle(
                color: _textColor,
              ),
          ),
          SizedBox(width: 4)
        ],
      ),
    );
  }

  Widget _iconWidget() {
    if (icon != null) {
      return Icon(
        icon,
        size: iconSize,
        color: AppColors.darkAccentColor,
      );
    } else if (image != null) {
      return image!;
    } else {
      return Container();
    }
  }
}

class EditFieldButton extends StatelessWidget {
  const EditFieldButton({
    Key? key,
    required this.onTap,
    required this.title,
    this.isLoading = false,
    this.hasIcon = true,
  }) : super(key: key);

  final String title;
  final VoidCallback? onTap;
  final bool isLoading;
  final bool hasIcon;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: isLoading ? null : onTap,
      child: Container(
        margin: EdgeInsets.only(bottom: 16),
        padding: const EdgeInsets.all(6.0),
        decoration:
designBoxDecorator(hasGradient: true),
        child: isLoading
            ? CircularProgressIndicator()
            : Row(
                children: <Widget>[
                  Expanded(
                    child: Container(
                        padding:
EdgeInsets.symmetric(horizontal: 4),
                        child: Text(title, style:
radencyButtonTextStyle(),)),
                  ),
                  hasIcon ?
Icon(Icons.chevron_right, size: 28) : Container(),
                ],
              ),
      ),
    );
```

```
  }
}

class RefreshButton extends StatelessWidget {
  const RefreshButton({
    Key? key,
    required this.onPressed,
    this.size = 42,
    this.label,
  }) : super(key: key);

  final VoidCallback? onPressed;
  final double size;
  final String? label;

  @override
  Widget build(BuildContext context) {
    return label != null
        ? Center(
            child: Column(children: [
              SizedBox(height: 100),
              _iconButton(),
              SizedBox(height: 10),
              Text(label!),
            ]),
          )
        : _iconButton();
  }

  Widget _iconButton() {
    return IconButton(
      iconSize: size,
      icon: Icon(Icons.refresh),
      onPressed: onPressed,
    );
  }
}


class StylizedElevatedButton extends
StatelessWidget {
  final Widget child;
  final Color? backgroundColor;
  final Color? foregroundColor;
  final VoidCallback? onPressed;

  StylizedElevatedButton(
      {required this.child,
      this.backgroundColor = Colors.transparent,
      required this.onPressed,
      this.foregroundColor});

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: designBoxDecorator(hasGradient:
true),
      child: ElevatedButton(
        child: child,
        style: ButtonStyle(
            elevation:
MaterialStateProperty.resolveWith((states) => 10),
            backgroundColor:
MaterialStateProperty.resolveWith(
                (states) => getColor(states,
backgroundColor!)),
            foregroundColor:
MaterialStateProperty.all<Color>(
                foregroundColor == null
                    ?
Theme.of(context).colorScheme.secondary
                    : foregroundColor!),
            textStyle:
MaterialStateProperty.all<TextStyle>(TextStyle(
                color: foregroundColor == null
                    ?
Theme.of(context).secondaryHeaderColor
                    : foregroundColor)),
            shape:
MaterialStateProperty.all<OutlinedBorder>(
              RoundedRectangleBorder(
                borderRadius:
BorderRadius.circular(8),
              ),
            ),
            shadowColor:
MaterialStateProperty.all(backgroundColor)),
        onPressed: onPressed,
      ),
    );
  }

  Color getColor(Set<MaterialState> states, Color
defaultColor) {
    if (states.any(<MaterialState>{
      MaterialState.disabled,
    }.contains)) {
      return Colors.grey;
    }
    return defaultColor;
  }
}

class DesignButton extends StatelessWidget {
  const DesignButton({
    Key? key,
    required this.onTap,
    required this.child,
    this.backgroundColor = Colors.transparent,
    this.isLoading = false,
    this.height = 50.0,
  }) : super(key: key);

  final Widget child;
  final VoidCallback? onTap;
  final Color backgroundColor;
  final bool isLoading;
  final double height;

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(8.0),
      child: Container(
        height: height,
        width: double.infinity,
        child: StylizedElevatedButton(
          child: isLoading ?
CircularProgressIndicator() : child,
          onPressed: isLoading ? null : onTap,
          backgroundColor: backgroundColor,
          foregroundColor: Colors.orange,
        ),
      ),
    );
  }
}

class MerchantWidget extends StatelessWidget {
  final Merchant merchant;
  final double? givenSize;

  MerchantWidget({
    required this.merchant,
    this.givenSize,
  });

  @override
  Widget build(BuildContext context) {
    double size = givenSize ??
MediaQuery.of(context).size.width / 3 - 6;
    return Container(
      alignment: Alignment.center,
      margin: const EdgeInsets.all(3),
```

```
      child: GestureDetector(
        onTap: () async => await
launch(merchant.link),
        child: Stack(
          fit: StackFit.passthrough,
          children: [
            ClipRRect(
              borderRadius:
BorderRadius.circular(8.0),
              child: CachedNetworkImage(
                width: size,
                height: size,
                fit: BoxFit.fitHeight,
                imageUrl: merchant.imageUrl,
                placeholder: (context, url) =>
                    Center(child:
CircularProgressIndicator()),
                errorWidget: (context, url, error)
=> CachedNetworkImage(
                    imageUrl:

'https://images.unsplash.com/photo-1496171367470-
9ed9a91'

'ea931?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufD
B8fHx8'
                        '&ixlib=rb-
1.2.1&auto=format&fit=crop&w=270&q=80',
                    placeholder: (context, url) =>
                        Center(child:
CircularProgressIndicator()),
                    errorWidget: (context, url,
error) =>
                        Icon(Icons.image, color:
AppColors.shadowColor)),
              ),
            ),
            Positioned(
              top: size / 2,
              child: ClipRRect(
                borderRadius:
BorderRadius.circular(8.0),
                child: Container(
                  width: size,
                  height: size / 2,
                  decoration: BoxDecoration(
                      gradient:
LinearGradient(colors: [
                    Colors.black.withOpacity(0.8),
                    Colors.white.withOpacity(0)
                  ], begin:
Alignment.bottomCenter, end:
Alignment.topCenter)),
                  padding: const
EdgeInsets.all(8),
                  alignment: Alignment.bottomLeft,
                  child: Text(
                    merchant.name,
                    style: TextStyle(
                      color:
Colors.white.withOpacity(0.8),
                      fontSize: 16,
                    ),
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}

class ProfileAvatar extends StatelessWidget {
  final String imagePath;
  final VoidCallback? onClicked;
  final bool changeable;
  final double size;

  const ProfileAvatar({
    Key? key,
    required this.imagePath,
    this.onClicked,
    this.changeable = false,
    this.size = 50.0,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      height: size + 16,
      width: size + 16,
      child: changeable
          ? Stack(
              children: <Widget>[
                buildImage(),
                Positioned(
                  child: GestureDetector(
                    onTap: onClicked,
                    child: CircleAvatar(
                      backgroundColor:
AppColors.darkAccentColor,
                      child:
Icon(Icons.edit_outlined,
                          size: 20, color:
AppColors.lightContrastColor),
                      radius: 14,
                    ),
                  ),
                  right: size / 20,
                  bottom: size / 20,
                ),
              ],
            )
          : Container(
              decoration:
designBoxDecorator(borderRadius: 100),
              padding: EdgeInsets.all(0.5),
              child: buildImage()),
    );
  }

  Widget buildImage() {
    return CachedNetworkImage(
      imageUrl: imagePath,
      imageBuilder: (context, imageProvider) =>
CircleAvatar(
        radius: size - 16,
        backgroundImage: imageProvider,
        backgroundColor: AppColors.defTextColor,
      ),
      placeholder: (context, url) => Center(child:
CircularProgressIndicator()),
      errorWidget: (context, url, error) => Icon(
        Icons.account_circle,
        key: Key('error-icon'),
        color: AppColors.shadowColor,
        size: size + 15,
      ),
    );
  }
}

class CustomPhoneNumberFormField extends
StatefulWidget {
  const CustomPhoneNumberFormField(
      {Key? key,
      required this.formKey,
      required this.title,
      this.hintText,
      required this.onChangedCallback,
      this.errorText,
      this.initialValue,
```

```
      this.isoCode,
      this.autovalidateMode =
AutovalidateMode.disabled,
      this.focusNode,
      this.prefix,
      this.enabled = true,
      this.errorMessage,
      this.titleSpace = 10.0})
      : super(key: key);

  final Key formKey;
  final Widget title;
  final String? hintText;
  final Function(PhoneNumber) onChangedCallback;
  final String? errorText;
  final String? initialValue;
  final String? isoCode;
  final AutovalidateMode autovalidateMode;
  final FocusNode? focusNode;
  final Widget? prefix;
  final bool enabled;
  final String? errorMessage;
  final double titleSpace;

  @override
  State<CustomPhoneNumberFormField> createState()
=>
      _CustomPhoneNumberFormFieldState();
}

class _CustomPhoneNumberFormFieldState
    extends State<CustomPhoneNumberFormField> {
  PhoneNumber? _phoneNumber;

  @override
  void initState() {

WidgetsBinding.instance!.addPostFrameCallback((_)
async {
      _phoneNumber = widget.initialValue == null
          ? PhoneNumber(
              phoneNumber: widget.initialValue,
isoCode: widget.isoCode)
          : await
PhoneNumber.getRegionInfoFromPhoneNumber(
              widget.initialValue!,
widget.isoCode!);
    });
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.symmetric(
          horizontal: 8.0, vertical:
widget.titleSpace - 2.0),
      child: Column(
        crossAxisAlignment:
CrossAxisAlignment.start,
        children: [
          widget.title,
          SizedBox(
            height: widget.titleSpace,
          ),
          Form(
            key: widget.formKey,
            child: InternationalPhoneNumberInput(
              selectorConfig: SelectorConfig(
                selectorType:
PhoneInputSelectorType.DIALOG,
              ),
              isEnabled: widget.enabled,
              focusNode: widget.focusNode,
              autoValidateMode:
widget.autovalidateMode,
              inputDecoration:
formFieldDecoration(
                context,
                hintText: widget.hintText,
                errorText: widget.errorText,
                prefix: widget.prefix,
              ),
              onInputChanged:
widget.onChangedCallback,
              initialValue: _phoneNumber == null
                  ? PhoneNumber(
                      phoneNumber:
widget.initialValue, isoCode: widget.isoCode)
                  : _phoneNumber,
              errorMessage: widget.errorMessage,
              formatInput: false,
            ),
          ),
        ],
      ),
    );
  }
}

class CustomTextFormField extends StatelessWidget
{
  const CustomTextFormField({
    Key? key,
    this.formKey,
    required this.title,
    this.hintText,
    required this.onChangedCallback,
    this.validator,
    this.errorText,
    this.textInputType,
    this.obscureText = false,
    this.initialValue,
    this.autovalidateMode,
    this.inputFormatters = const [],
    this.focusNode,
    this.prefix,
    this.enabled = true,
    this.titleSpace = 10.0,
  }) : super(key: key);

  final Key? formKey;
  final Widget title;
  final String? hintText;
  final Function(String) onChangedCallback;
  final String? errorText;
  final TextInputType? textInputType;
  final bool obscureText;
  final String? initialValue;
  final String? Function(String?)? validator;
  final AutovalidateMode? autovalidateMode;
  final List<TextInputFormatter> inputFormatters;
  final FocusNode? focusNode;
  final Widget? prefix;
  final bool enabled;
  final double titleSpace;

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding:
          EdgeInsets.symmetric(horizontal: 8.0,
vertical: titleSpace - 2.0),
      child: Column(
        crossAxisAlignment:
CrossAxisAlignment.start,
        children: [
          title,
          SizedBox(height: titleSpace),
          Form(
            key: formKey,
            child: TextFormField(
              enabled: enabled,
```

```
              focusNode: focusNode,
              autovalidateMode: autovalidateMode,
              keyboardType: textInputType,
              decoration: formFieldDecoration(
                context,
                hintText: hintText,
                errorText: errorText,
                prefix: prefix,
              ),
              onChanged: onChangedCallback,
              obscureText: obscureText,
              initialValue: initialValue,
              validator: validator,
              inputFormatters: inputFormatters,
            ),
          ),
        ],
      ),
    );
  }
}

class CustomPasswordFormField extends
StatefulWidget {
  const CustomPasswordFormField({
    Key? key,
    required this.formKey,
    required this.title,
    this.hintText,
    required this.onChangedCallback,
    this.validator,
    this.errorText,
    this.initialValue,
    this.autovalidateMode,
    this.inputFormatters = const [],
    this.focusNode,
    this.prefix,
    this.enabled = true,
  }) : super(key: key);

  final Key formKey;
  final Widget title;
  final String? hintText;
  final Function(String) onChangedCallback;
  final String? Function(String?)? validator;
  final String? errorText;
  final String? initialValue;
  final AutovalidateMode? autovalidateMode;
  final List<TextInputFormatter> inputFormatters;
  final FocusNode? focusNode;
  final Widget? prefix;
  final bool enabled;

  @override
  _CustomPasswordFormFieldState createState() =>
      _CustomPasswordFormFieldState();
}

class _CustomPasswordFormFieldState extends
State<CustomPasswordFormField> {
  bool _isObscure = true;

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(8.0),
      child: Column(
        crossAxisAlignment:
CrossAxisAlignment.start,
        children: [
          widget.title,
          SizedBox(
            height: 10,
          ),
          Form(
            key: widget.formKey,
            child: TextFormField(
              enabled: widget.enabled,
              focusNode: widget.focusNode,
              autovalidateMode:
widget.autovalidateMode,
              obscureText: _isObscure,
              decoration: formFieldDecoration(
                context,
                hintText: widget.hintText,
                errorText: widget.errorText,
                prefix: widget.prefix,
                suffixIcon:
                    LimitedBox(
                  maxHeight: 20,
                  child: IconButton(
                      padding:
EdgeInsets.all(0.0),
                      icon: Icon(
                          _isObscure ?
Icons.visibility : Icons.visibility_off),
                      iconSize: 20.0,
                      onPressed: () => setState(()
=> _isObscure = !_isObscure),
                      splashRadius: 1.0),
                ),
              ),
              onChanged: widget.onChangedCallback,
              initialValue: widget.initialValue,
              validator: widget.validator,
            ),
          ),
        ],
      ),
    );
  }
}


class NumberTitledBox extends StatelessWidget {
  const NumberTitledBox(
      {Key? key,
      required this.number,
      required this.label,
      this.scale = 0.0,
      this.textColor = AppColors.backgroundColor,
      this.hasIndicator = false,
      this.numberOfPayments,
      this.currentPayment,
      Widget belowWidget = const Center()})
      : super(key: key);

  final String number;
  final String label;
  final double scale;
  final Color textColor;
  final bool hasIndicator;
  final int? currentPayment;
  final int? numberOfPayments;

  @override
  Widget build(BuildContext context) {
    double distanceScale = 1 + scale * 3;
    double textScale = 1 + scale / 2;
    return Container(
        padding: EdgeInsets.only(
            top: 14 * distanceScale,
            bottom: 10 * distanceScale,
            left: 18 * distanceScale,
            right: 18 * distanceScale),
        margin: EdgeInsets.only(top: 14, left: 12,
right: 12),
        decoration:
designBoxDecorator(hasGradient: true),
        child: Column(children: [
          Text(
            label,
            style: TextStyle(color: textColor),
          ),
```

```
          FittedBox(
            child: Text("\$${number.toString()}",
                style: TextStyle(fontSize: 34 *
textScale, color: textColor)),
          ),
          hasIndicator ? progressIndicator() :
Container(),
        ]));
  }

  Widget progressIndicator() {
    return Container(
      padding: EdgeInsets.only(top: 10, bottom:
10),
      child: Row(children: [
        Expanded(
          child: StepProgressIndicator(
              totalSteps: numberOfPayments!,
              currentStep: currentPayment!,
              size: 10,
              padding: 0,
              selectedColor:
AppColors.lightContrastColor,
              unselectedColor:
AppColors.lightColor,
              roundedEdges: Radius.circular(30)),
        ),
        Container(
            padding: EdgeInsets.only(left: 10),
            child:
Text('$currentPayment/$numberOfPayments',
                style: TextStyle(color:
textColor)))
      ]),
    );
  }
}

Widget pageTitle(String title) {
  return Padding(
    padding: const EdgeInsets.all(8.0),
    child: Container(
        height: 50,
        alignment: Alignment.center,
        width: double.infinity,
        child: Text(title,
            style: TextStyle(
              fontSize: 30,
            ))),
  );
}

Widget fieldTitle(String title, {bool
showFieldRequired = false}) {
  return RichText(
    text: TextSpan(
      style: formFieldTitleStyle(),
      children: <TextSpan>[
        TextSpan(text: title),
        if (showFieldRequired)
          TextSpan(text: ' *', style: new
TextStyle(color: Colors.red)),
      ],
    ),
  );
}

Future<bool> showConfirmChangesAlertDialog(
    {required BuildContext context,
    required String exitLabel,
    required String saveChangesLabel,
    required String title,
    required VoidCallback onSavePressed,
    required VoidCallback onExitPressed,
    }) async {
    return await showDialog(
        context: context,
        barrierDismissible: true,
        builder: (_) => AlertDialog(
              title: Text(title),
              actions: [
                TextButton(
                  child: Text(exitLabel),
                  onPressed: onExitPressed,
                ),
                TextButton(
                  child: Text(saveChangesLabel),
                  onPressed: onSavePressed,
                )
              ],
            ));
}


void showImageSourceActionSheet({
  required Function(ImageSource)
selectImageSource,
  required String galleryLabel,
  required String cameraLabel,
  required BuildContext context
}) {
  if (Platform.isIOS) {
    showCupertinoModalPopup(
      context: context,
      builder: (context) => CupertinoActionSheet(
        actions: [
          CupertinoActionSheetAction(
            child: Text(cameraLabel),
            onPressed: () {
              Navigator.pop(context);

selectImageSource(ImageSource.camera);
            },
          ),
          CupertinoActionSheetAction(
            child: Text(galleryLabel),
            onPressed: () {
              Navigator.pop(context);

selectImageSource(ImageSource.gallery);
            },
          )
        ],
      ),
    );
  } else {
    showModalBottomSheet(
      context: context,
      builder: (context) => Wrap(children: [
        ListTile(
          leading: Icon(Icons.camera_alt),
          title: Text(cameraLabel),
          onTap: () {
            Navigator.pop(context);
            selectImageSource(ImageSource.camera);
          },
        ),
        ListTile(
          leading: Icon(Icons.photo_album),
          title: Text(galleryLabel),
          onTap: () {
            Navigator.pop(context);

selectImageSource(ImageSource.gallery);
          },
        ),
      ]),
    );
  }
}
```

```
class CenteredScrollView extends StatelessWidget {
  const CenteredScrollView({Key? key, required
this.child}) : super(key: key);

  final Widget child;

  @override
  Widget build(BuildContext context) {
    return  SafeArea(
      child: Container(
        alignment: Alignment.topCenter,
        child: SingleChildScrollView(
          padding:
EdgeInsets.symmetric(horizontal: 20, vertical:
15),
          child: Container(
            constraints: BoxConstraints(maxWidth:
min(400, MediaQuery.of(context).size.width *
0.9)),
            child: this.child,
          ),
        ),
      ),
    );
  }

class ChangeableDate extends StatefulWidget {
  ChangeableDate({
    Key? key,
    required this.initialDate,
    required this.label,
    required this.onChangedCallback,
  }) : super(key: key);

  final DateTime initialDate;
  final String label;
  final Function(String) onChangedCallback;

  @override
  State<ChangeableDate> createState() =>
_ChangeableDateState();
}

class _ChangeableDateState extends
State<ChangeableDate> {
  late DateTime _initialDate = widget.initialDate;

  @override
  Widget build(BuildContext context) {
    var monthDateFormat = DateFormat("MMMM");
    return Container(
      child: Column(
        crossAxisAlignment:
CrossAxisAlignment.start,
        children: [
          Text(widget.label, style:
TextStyle(fontWeight: FontWeight.bold)),
          SizedBox(height: 8),
          Row(
            children: [

_dateItemButton(_initialDate.day.toString(),
context),
              SizedBox(width: 16),
              Expanded(
                  child: _dateItemButton(

monthDateFormat.format(_initialDate), context)),
              SizedBox(width: 16),

_dateItemButton(_initialDate.year.toString(),
context),
            ],
          ),
          SizedBox(height: 12),
        ],
      ),
    );
  }

  Widget _dateItemButton(String label,
BuildContext context) {
    return Container(
      child: ElevatedButton(
        style: ElevatedButton.styleFrom(
          primary: Colors.white,
          onPrimary: Colors.black,
        ),
        onPressed: () {
          _selectDate(context);
        },
        child: Text(label),
      ),
    );
  }

  _selectDate(BuildContext context) async {
    var lastDate = DateTime(
        DateTime.now().year - 18,
DateTime.now().month, DateTime.now().day);
    var initialDate =
        widget.initialDate.isBefore(lastDate) ?
widget.initialDate : lastDate;
    final DateTime? picked = await showDatePicker(
      context: context,
      initialDate: initialDate,
      firstDate: DateTime(1930),
      lastDate: lastDate,
    );
    if (picked != null && picked !=
widget.initialDate) {
      widget.onChangedCallback(picked.toString());
      setState(() {
        _initialDate = picked;
      });
    }
  }
}


BoxDecoration designBoxDecorator({
  bool hasGradient = false,
  Color color = AppColors.backgroundColor,
  bool hasCircularBorders = true,
  double borderRadius = 8.0,
  bool hasShadow = true,
}) {
  return BoxDecoration(
    gradient: hasGradient ? AppColors.boxGradient
: null,
    border:
        hasShadow ? null : Border.all(width: 1.0,
color: AppColors.shadowColor),
    color: color,
    borderRadius: hasCircularBorders
        ?
BorderRadius.all(Radius.circular(borderRadius))
        : null,
    boxShadow: [
      BoxShadow(
        color: hasShadow ? AppColors.shadowColor :
AppColors.backgroundColor,
        blurRadius: 8,
        offset: Offset(0, 2),
      ),
    ],
  );
}
class Routes {
  static const splashScreen = '/';
  static const loginPage = '/login';
  static const registerPage = '/register';
  static const restorePasswordPage =
'/password_restore';
```

```
  static const forgotPasswordPage =
'/forgot_password';
  static const merchantsPage = '/all_goods';
  static const dashboardPage = '/dashboard';
  static const orderTransactions =
'/order_transactions';
  static const editProfilePage = '/edit_profile';
  static const editPasswordPage =
'/edit_password';
  static const ordersPage = '/orders';
}
final String emailRegExp =
    r"^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-
9](?:[a-zA-Z0-9-]{0,253}[a-zA-Z0-9])?(?:\.[a-zA-
Z0-9](?:[a-zA-Z0-9-]{0,"
    r"253}[a-zA-Z0-9])?)*$";


TextStyle formFieldTitleStyle() {
  return TextStyle(fontWeight: FontWeight.bold,
color: Colors.black);
}

TextStyle coloredFormFieldTitleStyle() {
  return TextStyle(fontWeight: FontWeight.bold,
color: Colors.white);
}

TextStyle companyNameStyle() {
  return TextStyle(
    color: Colors.red,
    fontWeight: FontWeight.bold,
    fontSize: 22,
  );
}

TextStyle authViewTitleStyle() {
  return TextStyle(fontSize: 30, fontWeight:
FontWeight.bold);
}

TextStyle authViewMessageStyle() {
  return TextStyle(fontSize: 18, );
}

TextStyle radencyButtonTextStyle() {
  return TextStyle(fontSize: 20, color:
Colors.white);
}

extension StringExtension on String {
  String capitalize() {
    return
"${this[0].toUpperCase()}${this.substring(1)}";
  }

InputDecoration formFieldDecoration(context,
    {String? hintText,
    bool focused = false,
    String? errorText,
    Widget? prefix,
    Widget? suffixIcon}) {
  return InputDecoration(
    helperText: '',
    hintText: hintText,
    errorText: errorText,
    prefix: prefix,
    suffix: suffixIcon,
  );
}

class AppColors {
  // static const Gradient boxGradient = const
LinearGradient(
  //     colors: [Color(0xff0092a3),
Color(0xff00657e)], tileMode: TileMode.decal);
  //
  // static const Color shadowColor =
Color(0xff919191);
  // static const Color accentColor =
Color(0xff0084a3);
  //
  // static const Color lightContrastColor =
Color(0xffebeedc);
  // static const Color lightColor =
Color(0xffea8b7a);
  // static const Color darkAccentColor =
Color(0xff11687d);
  // static const backgroundColor =
Color(0xffffffff);
  // static const defTextColor =
Color(0xff000000);
  static const Gradient boxGradient = const
LinearGradient(
      colors: [Color(0xff5300ee),
Color(0xff8200ee)]);

  static const Color shadowColor =
Color(0xff919191);
  static const Color accentColor =
Color(0xff6739fe);

  static const Color lightContrastColor =
Color(0xffebeedc);
  static const Color lightColor =
Color(0xfffddabd);
  static const Color darkAccentColor =
Color(0xff272b90);
  static const backgroundColor =
Color(0xffffffff);
  static const defTextColor = Color(0xff000000);
}

String? validatePasswordRepeat({
  required String? passwordRepeat,
  required String password,
}) {
  if (passwordRepeat?.trim().length == 0) {
    return S.current.passwordRepeatValidatorEmpty;
  } else if (passwordRepeat != password) {
    return
S.current.passwordRepeatValidatorNotMatch;
  }

  return null;
}

String? validateName({required String? name}) {
  final maximumLength = 70;

  if (name?.trim().length == 0) {
    return S.current.firstNameValidatorEmpty;
  } else if (name != null && name.length >
maximumLength) {
    return S.current.firstNameValidatorTooLong;
  }

  return null;
}

String? validateEmail({required String? email}) {
  if (email?.trim().length == 0) {
    return S.current.emailValidatorEmpty;
  } else if
(!RegExp(emailRegExp).hasMatch(email!)) {
    return S.current.emailValidatorIncorrect;
  }

  return null;
}

String? validatePassword({
  required String? password,
  required bool shouldValidateComplexity,
```

```
}) {
  final minimalLength = 8;
  final maximumLength = 100;

  if (password?.trim().length == 0) {
    return S.current.passwordValidatorEmpty;
  } else if (shouldValidateComplexity) {
    if (password!.length < minimalLength) {
      return
S.current.passwordValidatorShort(minimalLength);
    }
    if (password.length > maximumLength) {
      return
S.current.passwordValidatorLong(maximumLength);
    }
    if
(!RegExp(passwordRegExp).hasMatch(password)) {
      return S.current.passwordValidatorIncorrect;
    }
  }

  return null;
}

class BaseAPI {
  const BaseAPI();

  static String? base;
  static String api = '$base/api/v1';
  static String users = '$api/users';
  static String auth = '$api/auth';
  static String products = '$api/products';

  String get usersPath => users;
  Uri get usersUri => Uri.parse(usersPath);

  String get profilePath => '$users/profile';
  Uri get profileUri => Uri.parse(profilePath);

  String get securityPath =>
'$users/profile/security';
  Uri get securityUri => Uri.parse(securityPath);

  String get loginPath => '$auth/login';
  Uri get loginUri => Uri.parse(loginPath);

  String get resetPath => '$auth/reset';
  Uri get resetUri => Uri.parse(resetPath);

  String get refreshTokenPath => '$auth/refresh-
token';
  Uri get refreshTokenUri =>
Uri.parse(refreshTokenPath);

  String get fileUploadPath => '$api/file-upload';
  Uri get fileUploadUri =>
Uri.parse(fileUploadPath);

  String get productsPath => products;
  Uri get productsUri => Uri.parse(productsPath);

  String get transactionsPath =>
"$api/transactions/merchant=1";

  String merchantsPath(Categories? category, bool
getPopular,
      String? searchQuery, int start, int limit) {
    if (searchQuery != null) {
      searchQuery = searchQuery[0].toUpperCase() +
searchQuery.substring(1);
      return
'$api/merchants/name_contains=$searchQuery';
    } else if (category != null && category !=
Categories.All) {
      return
'$api/merchants/category=${category.toString().rep
laceFirst('Categories.', '').toLowerCase()}';
    } else if (getPopular) {
      return '$api/merchants/popular';
    } else {
      return '$api/merchants';
    }
  }

  Map<String, String> headers() => {
        'Content-Type': 'application/json;
charset=UTF-8',
        'Authorization': 'Bearer',
        'Accept': 'application/json, text/plain,
*/*',
      };

  Map<String, String> authorisationHeaders(String
token,
      {Map<String, String>? additionalHeader}) {
    return <String, String>{
      'Accept': 'application/json, text/plain,
*/*',
      'Authorization': 'Bearer $token',
      ...?additionalHeader
    };
  }

  static initialise() async {
    base = 'http://' + await HostConfig.getHost();
  }
}

class ChangeProfileAPI extends BaseAPI {
  final Dio _dio = Dio();

  Future<void> editProfile(
      {required String body, required String
token}) async {
    var changeUserDataRequest = await
http.put(super.profileUri,
        headers: super.authorisationHeaders(token,
            additionalHeader: {"Content-Type":
"application/json"}),
        body: body);
    _verifyResponse(changeUserDataRequest);
  }

  Future<void> changePassword(
      {required String body, required String
token}) async {
    var changePasswordRequest = await
http.put(super.securityUri,
        headers: super.authorisationHeaders(token,
            additionalHeader: {"Content-Type":
"application/json;charset=UTF-8"}),
        body: body);
    _verifyResponse(changePasswordRequest);
  }

  Future<String> uploadAvatar(String filePath,
{required String token}) async {
    _dio.interceptors.clear();

_dio.interceptors.add(InterceptorsWrapper(onReques
t: (request, handler) {
      request.headers['Authorization'] = 'Bearer
$token';
      return handler.next(request);
    }));

    FormData formData = FormData.fromMap({
          filename: filePath.split('/').last),
    });
    var response = await
_dio.post(super.fileUploadPath, data: formData);
    return response.statusCode != 200
        ? throw Exception(response.statusMessage)
        : response.data['url'];
```

```
  }

  void _verifyResponse(http.Response response) {
    if (response.statusCode != 200) {
      throw Exception(response.reasonPhrase);
    }
  }
}



class EditProfileRepository {
  EditProfileRepository({
    TokenSecureStorage? tokenSecureStorage,
    ChangeProfileAPI? changeProfileAPI,
  })  : _tokenSecureStorage = tokenSecureStorage
?? TokenSecureStorage(),
        _changeProfileAPI = changeProfileAPI ??
ChangeProfileAPI();

  final TokenSecureStorage _tokenSecureStorage;
  final ChangeProfileAPI _changeProfileAPI;

  Future<void> editProfile({required User user})
async {
    final String? accessToken = await
_tokenSecureStorage.getAccessToken();
    try {
      _changeProfileAPI.editProfile(body:
jsonEncode(user), token: accessToken!);
    } catch (exception){
      final refreshedAccessToken = await
_tokenSecureStorage.getRefreshedAccessToken();
      _changeProfileAPI.editProfile(body:
jsonEncode(user), token: refreshedAccessToken!);
    }
  }

  Future<void> changePassword({
    required String oldPassword,
    required String newPassword,
    required String confirmPassword,
  }) async {
    String passwordsJson = jsonEncode({
      'password': oldPassword,
      'verifiedPassword': newPassword,
      'confirmPassword': confirmPassword
    });
    final String? accessToken = await
_tokenSecureStorage.getAccessToken();
    try {
      _changeProfileAPI.changePassword(body:
passwordsJson, token: accessToken!);
    } catch (exception){
      final refreshedAccessToken = await
_tokenSecureStorage.getRefreshedAccessToken();
      _changeProfileAPI.changePassword(body:
passwordsJson, token: refreshedAccessToken!);
    }
  }

  Future<String> uploadAvatar(String filePath)
async {
    String? accessToken = await
_tokenSecureStorage.getAccessToken();
    try {
      return
_changeProfileAPI.uploadAvatar(filePath, token:
accessToken!);
    } catch (exception){
      final refreshedAccessToken = await
_tokenSecureStorage.getRefreshedAccessToken();
      return
_changeProfileAPI.uploadAvatar(filePath, token:
refreshedAccessToken!);
    }
  }
}


class UserDataAPI extends BaseAPI {
  Dio _dio = Dio();

  Future<List> fetchMerchants(
      {Categories? category,
      required int limit,
      int startIndex = 0,
      bool getPopular = false,
      String? searchQuery}) async {
    var response = await _dio.get(super
        .merchantsPath(category, getPopular,
searchQuery, startIndex, limit));

    return response.statusCode != 200
        ? throw Exception(response.statusMessage)
        : response.data;
  }

  Future<List<Transaction>> fetchTransactions(int
transactionLimit,
      [int startIndex = 0]) async {
    try {
      var response = await
_dio.get(super.transactionsPath);
      List<Transaction> tr = [];
      (response.data as List).forEach((element)
{tr.add(Transaction.fromJson(element));});
      return tr;
    } catch (exception){
      await Future.delayed(Duration(milliseconds:
1000));
      return
List<Transaction>.generate(transactionLimit,
              (index) =>
Transaction.defaultTransaction(index +
startIndex));
    }

  }

  Future<Dashboard> fetchDashboard() async {
    // todo http request
    await Future.delayed(Duration(milliseconds:
1000));

    return Dashboard.defaultData();
  }

  Future<List<OrderTransactions>>
fetchOrderTransactions() async {
    // todo http request
    await Future.delayed(Duration(milliseconds:
1000));

    OrderTransactions.tempDateCounter =
DateTime.now().add(Duration(days: 50));
    return List.generate(12, (index) =>
OrderTransactions.defaultData());
  }

  Future<List<Order>> getProducts(String token)
async {
    var response = await
http.get(super.productsUri,
        headers:
super.authorisationHeaders(token));
    _verifyResponse(response);
    List<dynamic> orderList =
jsonDecode(response.body);
    return orderList.map((e) =>
Order.fromJson(e)).toList();
  }

  void _verifyResponse(http.Response response) {
```

```
    if (response.statusCode != 200) {
      throw Exception(response.reasonPhrase);
    }
  }
}


class UserDataRepository {
  UserDataRepository({
    TokenSecureStorage? tokenSecureStorage,
    UserDataAPI? userDataAPI,
  })  : _tokenSecureStorage = tokenSecureStorage
?? TokenSecureStorage(),
        _userDataAPI = userDataAPI ??
UserDataAPI();

  final TokenSecureStorage _tokenSecureStorage;
  final UserDataAPI _userDataAPI;

  Future<List<Merchant>> fetchMerchants(
      {required int limit,
      int startIndex = 0,
      Categories? category,
      bool getPopular = false,
      String? searchQuery}) async {
    var merchants = await
_userDataAPI.fetchMerchants(
        category: category,
        limit: limit,
        startIndex: startIndex,
        getPopular: getPopular,
    searchQuery: searchQuery);

    return merchants.map((e) =>
Merchant.fromJson(e)).toList();
  }

  Future<List<Transaction>> fetchTransactions(
      {required int transactionLimit, int
startIndex = 0}) async {
    return
_userDataAPI.fetchTransactions(transactionLimit,
startIndex);
  }

  Future<Dashboard> fetchDashboard() async {
    return _userDataAPI.fetchDashboard();
  }

  Future<List<OrderTransactions>>
fetchOrderTransactions() async {
    return _userDataAPI.fetchOrderTransactions();
  }

  Future<List<Order>> getProducts() async {
    final String? accessToken = await
_tokenSecureStorage.getAccessToken();
    try {
      return
_userDataAPI.getProducts(accessToken!);
    } catch (exception) {
      final refreshedAccessToken =
          await
_tokenSecureStorage.getRefreshedAccessToken();
      return
_userDataAPI.getProducts(refreshedAccessToken!);
    }
  }
}


class TokenSecureStorage {
  const TokenSecureStorage({RefreshTokenAPI?
refreshTokenAPI})
  : _refreshTokenAPI = refreshTokenAPI ?? const
RefreshTokenAPI();

  final _secureStorage = const
FlutterSecureStorage();
  static const String _accessTokenKey =
'access_token';
  static const String _refreshTokenKey =
'refresh_token';
  final RefreshTokenAPI _refreshTokenAPI;

  Future<String?> getAccessToken() async {
    return await _secureStorage.read(key:
_accessTokenKey);
  }

  Future<String?> getRefreshedAccessToken() async
{
    String? refreshToken = await
_secureStorage.read(key: _refreshTokenKey);
    final tokensResponse = await
_refreshTokenAPI.refreshTokens(refreshToken!);
    var tokens = jsonDecode(tokensResponse);
    saveTokens(tokens);
    return tokens['accessToken'];
  }

  Future<void> saveTokens(Map<String, dynamic>
response) async {
    await _secureStorage.write(
        key: _accessTokenKey, value:
response['accessToken']);
    await _secureStorage.write(
        key: _refreshTokenKey, value:
response['refreshToken']);
  }

  void logout() {
    _secureStorage.deleteAll();
  }
}


class AuthenticationAPI extends BaseAPI {
  const AuthenticationAPI();

  Future<String> makeRegisterRequest(
      String firstName,
      String lastName,
      String email,
      String phoneNumber,
      String password,
      String confirmPassword) async {
    final registerResponse = await
http.post(super.usersUri,
        headers: super.headers(),
        body: jsonEncode(<String, dynamic>{
          "user": {
            "firstName": firstName,
            "lastName": lastName,
            "email": email,
            "phoneNumber": phoneNumber,
            "password": password,
            "confirmPassword": confirmPassword
          },
          "role": {"roleId": 1}
        }));
    _verifyResponse(registerResponse);
    return registerResponse.body;
  }

  Future<http.Response> makeLoginRequest(String
email, String password) async {
    final loginResponse = await http.post(
      Uri.parse(super.loginPath),
      headers: super.headers(),
      body: jsonEncode(<String, String>{"email":
email, "password": password}),
    );
    _verifyResponse(loginResponse);
```

```
    return loginResponse;
  }

  Future<String> makeForgotPasswordRequest(String
email) async {
    final forgotPasswordResponse = await
http.post(
      super.resetUri,
      headers: super.headers(),
      body: jsonEncode(<String, String>{'email':
email}),
    );
    _verifyResponse(forgotPasswordResponse);
    return forgotPasswordResponse.body;
  }

  Future<String> makeRestorePasswordRequest(
      String password, String confirmPassword)
async {
    final restorePasswordResponse = await
http.post(
      super.usersUri,
      headers: super.headers(),
      body: jsonEncode(<String, String>{
        "password": password,
        "confirmPassword": confirmPassword
      }),
    );
    _verifyResponse(restorePasswordResponse);
    return(restorePasswordResponse.body);
  }

  Future<User> fetchUserData(String accessToken,
{String? etag}) async {
    if (etag == null) {
      final idResponse = await
http.get(super.loginUri,
          headers:
super.authorisationHeaders(accessToken));
      _verifyResponse(idResponse);

      etag = idResponse.headers['etag'];
    }

    final profileResponse = await
http.get(super.profileUri,
        headers:
super.authorisationHeaders(accessToken,
            additionalHeader: {'If-None-Match':
etag!}));
    _verifyResponse(profileResponse);

    return User.fromJson(profileResponse.body);
  }

  void _verifyResponse(http.Response response){
    if (response.statusCode != 200) {
      throw Exception(response.reasonPhrase);
    }
  }
}



class AuthenticationRepository {
  AuthenticationRepository({
    TokenSecureStorage? tokenSecureStorage,
    AuthenticationAPI? authenticationAPI,
  })  : _tokenSecureStorage = tokenSecureStorage
?? TokenSecureStorage(),
        _authenticationAPI = authenticationAPI ??
AuthenticationAPI();

  final TokenSecureStorage _tokenSecureStorage;
  final AuthenticationAPI _authenticationAPI;

  Future<User> getAuthorisedUser() async {
    String? accessToken = await
_tokenSecureStorage.getAccessToken();
    if (accessToken == null) {
      throw Exception("No authorised user");
    }
    try {
      return
_authenticationAPI.fetchUserData(accessToken);
    } catch (_) {
      final refreshedAccessToken = await
_tokenSecureStorage.getRefreshedAccessToken();
      return
_authenticationAPI.fetchUserData(refreshedAccessTo
ken!);
    }
  }

  Future<User> getUserByLogin(String email, String
password) async {
    var loginResponse =
        await
_authenticationAPI.makeLoginRequest(email,
password);

    var tokens =
jsonDecode(loginResponse.body)['tokens'];
    _tokenSecureStorage.saveTokens(tokens);
    return await
_authenticationAPI.fetchUserData(tokens['accessTok
en'],
        etag: loginResponse.headers['etag']);
  }

  Future<User> getUserByRestoredPassword(
      String password, String confirmPassword)
async {
    var restoreResponse =
        await
_authenticationAPI.makeLoginRequest(password,
confirmPassword);

    if (restoreResponse.statusCode != 200) {
      throw
Exception(restoreResponse.reasonPhrase);
    }

    var tokens =
jsonDecode(restoreResponse.body)['tokens'];
    _tokenSecureStorage.saveTokens(tokens);
    return await
_authenticationAPI.fetchUserData(tokens['accessTok
en'],
        etag: restoreResponse.headers['etag']);
  }

  Future<String> registerUser(String firstName,
String lastName, String email,
      String phoneNumber, String password, String
confirmPassword) async {
    return await
_authenticationAPI.makeRegisterRequest(
        firstName, lastName, email, phoneNumber,
password, confirmPassword);
  }

  Future<String> forgotPasswordRequest(String
email) async {
    return await
_authenticationAPI.makeForgotPasswordRequest(email
);
  }

  void logout() async =>
_tokenSecureStorage.logout();
}

class RefreshTokenAPI extends BaseAPI {
```

```
  const RefreshTokenAPI();

  Future<String> refreshTokens(String
refreshToken) async {
    var response = await
http.post(super.refreshTokenUri,
        headers:
super.authorisationHeaders(refreshToken));
    if(response.statusCode != 200){
      throw Exception("Failed to get access token
using refresh token");
    }
    return response.body;
  }
}

class App extends StatelessWidget {
  const App({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MultiBlocProvider(
      providers: [
        BlocProvider(
            create: (context) =>

AuthenticationBloc()..add(AuthenticationInitialize
())),
      ],
      child: AppView(),
    );
  }
}


void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  BaseAPI.initialise();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return App();
  }
}
```

```
// pubsec.yaml file
name: flutter_project
description: A new Flutter project.


version: 1.0.0+1

environment:
  sdk: ">=2.12.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter


  http: 0.13.3
  bloc: ^7.2.1
  flutter_bloc: 7.1.0
  equatable: 2.0.3
  envify: 2.0.2
  intl: 0.17.0
  mask_text_input_formatter: 2.0.0
  intl_phone_number_input: ^0.7.0+2
  device_info: ^2.0.2
  flutter_secure_storage: ^4.2.1
  grouped_list: ^4.1.0
  bloc_concurrency: ^0.1.0
  step_progress_indicator: ^1.0.1
  image_picker: ^0.8.4+4
  image_cropper: ^1.3.1
  cached_network_image: ^3.1.0+1
  dio: ^4.0.1
  url_launcher: ^6.0.13
  local_auth: ^1.1.8
  shared_preferences: ^2.0.8
  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter
  integration_test:
    sdk: flutter
  bloc_test: ^8.5.0

  json_serializable:
  envify_generator: any
  build_runner: an
```

ДОДАТОК 5

Сервіс для створення та керування відкладеними платежами за покупки в інтернеті

Текст програмного коду серверної частини

ІАЛЦ.467200.008 Д5

Аркушів 37

Київ 2022 р

```
const roleValidation = (permissions: string[]):
RequestHandler => {
  const validator: RequestHandler = (req, res, next)
=> {
    if (req.method === 'OPTIONS') {
      return next();
    }
    try {
      const token =
req.headers.authorization?.split(' ')[1];
      if (!token) {
        return res
          .status(HttpCode.UNAUTHORIZED)
          .json({ message:
ResponseMessages.NOT_AUTHORIZED });
      }

      jwt.verify(token, secret as Secret, (err,
decoded) => {
        if (err instanceof TokenExpiredError) {
          return res
            .status(HttpCode.UNAUTHORIZED)
            .send({ message:
ResponseMessages.TOKEN_EXPIRED });
        }
        if (err) {
          return res
            .status(HttpCode.UNAUTHORIZED)
            .send({ message:
ResponseMessages.WRONG_TOKEN_REQUEST });
        }

        const userPermissions = decoded?.permissions
|| [];
        let hasPermission = false;

        userPermissions.forEach((userPermission:
string) => {
          if (permissions.includes(userPermission)) {
            hasPermission = true;
          }
        });

        if (!hasPermission) {
          return res
            .status(HttpCode.FORBIDDEN)
            .send({ message:
ResponseMessages.NO_PERMISSION });
        }

        next();
      });
    } catch {
      res
        .status(HttpCode.UNAUTHORIZED)
        .json({ message:
ResponseMessages.NOT_AUTHORIZED });
    }
  };

  return validator;
};
```

```
const roleCheck = (): RequestHandler => {
  const validator: RequestHandler = async (req,
res) => {

    const { userEmail, expectedRole } = req.body

    try {
      // Finding user in DB.

      const user = await
userRepository.getByEmail(userEmail)
      if (!user) {
        return res
          .status(HttpCode.NOT_FOUND)
          .send({ message:
ResponseMessages.USER_NOT_FOUND })
      }

      // Finding user role in DB.

      const userRole = await
userRolesRepository.getByUserId(user.id)
      if (!userRole) {
        return res
          .status(HttpCode.NOT_FOUND)
          .send({ message:
ResponseMessages.USER_ROLE_NOT_FOUND })
      }

      // Finding role by roleId in user-role table.

      const role = await
roleRepository.getById(userRole.roleId.toString())
      if (!role) {
        return res
          .status(HttpCode.NOT_FOUND)
          .send({ message:
ResponseMessages.ROLE_NOT_FOUND })
      }

      // Check if user role matching expected one.

      if (role.key !== expectedRole) {
        return res
          .status(HttpCode.OK)
          .send({ message:
ResponseMessages.ROLE_NOT_MATCH })
      } else {
        return res
          .status(HttpCode.OK)
          .send({ message:
ResponseMessages.ROLE_MATCH })
      }
    } catch (err) {
      logger.error(err as string)
      return res
        .status(HttpCode.INTERNAL_SERVER_ERROR)
        .send({ message:
ResponseMessages.INTERNAL_SERVER_ERROR })
    }
  }

  return validator
}

export { roleValidation, roleCheck };
```

```
const handleError: ErrorRequestHandler = (err: HttpError, _req, res, _next) => {
  const { status = HttpCode.INTERNAL_SERVER_ERROR, message, stack } = err;

  logger.error(message, stack);

  return res.status(status).send({
    message,
  });
};

export { handleError };

const setTraceId: RequestHandler = async (_req, _res, next) => {
  const traceId = getRandomId();
  const store = new Map().set(AppAsyncStorageKey.TRACE_ID, traceId);

  return appAsyncStorage.run(store, () => {
    return next();
  });
};

export { setTraceId };

const yupValidation = (resourceSchema: AnyObjectSchema): RequestHandler => async (req, res, next) => {
  const resource = req.body;
  try {
    await resourceSchema.validate(resource);
    next();
  } catch (e) {
    console.error(e);
    res.status(400).json({ message: e.errors.join(', ') });
  }
};

export { yupValidation };
export {};

const refreshTokenValidation: RequestHandler = (req, res, next) => {
  const refreshToken = req.headers.authorization?.split(' ')[1];

  try {
    if (!refreshToken) {
      return res
        .status(HttpCode.BAD_REQUEST)
        .json({ message: ResponseMessages.WRONG_TOKEN_REQUEST });
    }

    jwt.verify(
      refreshToken,
      secretRefresh as Secret,
      (err: VerifyErrors | null, decoded: JwtPayload | undefined) => {
        if (err) {
          return res
            .status(HttpCode.UNAUTHORIZED)
            .send({ message: ResponseMessages.NOT_AUTHORIZED });
        }
        if (decoded?.userId) {

redis_client.GET(decoded?.userId, (err, data) => {
            if (err) {
              return res

.status(HttpCode.BAD_REQUEST)
                .send({ message: ResponseMessages.REDIS_REQUEST_ERROR });
            }

            if (data === null) {
              return res
```

```
.status(HttpCode.NOT_FOUND)
                .json({            message:
ResponseMessages.TOKEN_NOT_IN_STORE });
            }

            if (data !== refreshToken) {
              return res

.status(HttpCode.NOT_FOUND)
                .json({            message:
ResponseMessages.TOKEN_NOT_IN_STORE });
            }
            req.body.userId               =
decoded?.userId;
            next();
          });
        }
      },
    );
  } catch {
    res
      .status(HttpCode.UNAUTHORIZED)
      .json({                      message:
ResponseMessages.NOT_AUTHORIZED });
  }
};

export { refreshTokenValidation };
```

```
const jwtValidation: RequestHandler = (req,
res, next) => {
  if (req.method === 'OPTIONS') {
    return next();
  }
  try {
    const              token              =
req.headers.authorization?.split(' ')[1];
    if (!token) {
      return res
        .status(HttpCode.UNAUTHORIZED)
        .json({                      message:
ResponseMessages.NOT_AUTHORIZED });
    }

    jwt.verify(token,  secret  as  Secret,
(err, decoded) => {
      if (err instanceof TokenExpiredError)
{
        return res
          .status(HttpCode.UNAUTHORIZED)
          .send({                    message:
ResponseMessages.TOKEN_EXPIRED });
      }
      if (err) {
        return res
          .status(HttpCode.UNAUTHORIZED)
          .send({                    message:
ResponseMessages.WRONG_TOKEN_REQUEST });
      }
      req.user = decoded;
      next();
    });
  } catch {
    res
      .status(HttpCode.UNAUTHORIZED)
      .json({                      message:
ResponseMessages.NOT_AUTHORIZED });
  }
};

export { jwtValidation };
export    *    from    './log-request/log-
request.middleware';
export  *  from  './set-trace-id/set-trace-
id.middleware';
export   *   from   './handle-error/handle-
error.middleware';
export    *    from    './get-platform/get-
platform.middelware';

const  logRequest:  RequestHandler  =  async
(req, _res, next): Promise<void> => {
  logger.log(`METHOD:       ${req.method},
PATH:${req.path}`);
```

```
  return next();
};

export { logRequest };

const getPlatform: RequestHandler = async (req, res, next) => {
  try {
    const referer = urlHelper.parseURL(req.headers.referer).host;
    urlHelper.appHost = referer;
    next();
  } catch (e) {
    next(e);
  }
};

export { getPlatform };
export { ConnectionError } from 'sequelize';
export { ConnectionError as DbConnectionError } from './sequelize';
export { HttpError } from 'shared/exceptions';
export * from './packages';

type AppAsyncStorage = Map<AppAsyncStorageKey, unknown>;

export type { AppAsyncStorage };
export * from './app-async-storage.types';
// export * from './sequelize-atributes'
export type MailOptions = {
  mailTheme: string;
  data: {
    link: string;
  };
};
export * from "./mail-options.types"
export * from './app';
export * from './mail';
enum ResponseMessages {
  NOT_AUTHORIZED = 'Not authorized!',
  NON_EXISTING_EMAIL = 'Non-existing email',
  NON_MATCH_PASSWORDS = 'Passwords do not match',
  CONFIRMED = 'Confirmed',
  TOKEN_EXPIRED = 'Access Token was expired!',
  REDIS_REQUEST_ERROR = 'Redis request error!',
  WRONG_TOKEN_REQUEST = 'Wrong token request',
  TOKEN_NOT_IN_STORE = 'Refresh token is not in store!',
  NO_PERMISSION = "You don't have access",
  WRONG_PASSWORD = 'Wrong password',
  NOT_ACTIVATED = 'Account is not activated!',
  USER_NOT_FOUND = 'User not found!',
  USER_ROLE_NOT_FOUND = 'User role not found for that user!',
  ROLE_NOT_FOUND = 'Role with that ID not found!',
  ROLE_NOT_MATCH = 'Role you provide doesn\'t match role that user has!',
  ROLE_MATCH = 'User has role that you provided!',
  INTERNAL_SERVER_ERROR = 'Unexpected error happens in server!'
}

export { ResponseMessages };
enum AppConfig {
  API_V1_PREFIX = '/api/v1/',
}

export { AppConfig };
enum LogLevel {
  FATAL = 'fatal',
  ERROR = 'error',
  WARN = 'warn',
  INFO = 'info',
  DEBUG = 'debug',
```

```
  TRACE = 'trace'
}

export { LogLevel };

const { NODE_ENV, APP_SERVER_PORT } =
process.env;

const ENV = {
  APP: {
    NODE_ENV: NODE_ENV as AppEnvironment,
    SERVER_PORT: APP_SERVER_PORT ?? 3001,
  },
};

export { ENV };
enum AppEnvironment {
  DEVELOPMENT = 'development',
}

export { AppEnvironment };
export * from './app-environment.enum';
export * from './env.enum';
export * from './log-level.enum';
export * from './app-async-storage-
key.enum';
export * from './app-config.enum';
enum AppAsyncStorageKey {
  TRACE_ID = 'traceId',
}

export { AppAsyncStorageKey };
enum ModelName {
  USER = 'user',
  ROLE = 'role',
  PERMISSION = 'permission',
  USER_ROLES = 'user_roles',
  PRODUCT = 'product',
  MERCHANT = 'merchant',
  TRANSACTIONS = 'transactions',
  CATEGORY = 'category',
}
```

```
export { ModelName };
export * from './model-name.enum';
export { UserSex, UserType } from
'shared/common/enums';
export { HttpCode } from
'shared/common/enums';
export * from 'shared/common/enums/api';

export * from './app';
export * from './http';
export * from './api';
export * from './model-name';
export * from './user';
export * from './email';
export * from './permission';
export { Permission } from
'shared/common/enums';
enum EmailType {
  ACTIVATION = 'activation',
  RESET_PASSWORD = 'resetpass',
}

export { EmailType };
export interface ITransactionExtended
extends ITransaction {
  merchant_id?: number;
  id?: number;
  processed?: boolean;
  authorization?: string;
  expiration_date?: Date;
  createdAt?: Date;
  updatedAt?: Date;
}

export interface ITransaction {
  user_id: number;
  order_id: string;
  amount: number;
  merchant_sid: number;
  order_details: string;
  payments: number;
}
```

```
export type { IRole } from 'shared/common/interfaces';
export interface ICategory {
  id: number;
  name: string;
  createdAt: Date;
  updatedAt: Date;
}
export * from './auth-service.interface';
export interface AuthServiceRes<T>  {
  code: number,
  data: string | T
}
export type {IUserRoles} from 'shared/common/interfaces';
export type {IUser} from 'shared/common/interfaces';
export type { IProduct } from 'shared/common/interfaces';
export * from './user';
export * from './tokens';
export * from './auth';
export * from './permission';
export * from './role';
export * from './user-roles';
export * from './product';
export * from './merchant';
export * from './transaction';
export * from './category';
export type { IPermission } from 'shared/common/interfaces';
export type { ITokens } from 'shared/common/interfaces';

export interface IMerchant {
  id: number;
  sid: number;
  name: string;
  link: string;
  category?: ICategory;
}

const swaggerOptions = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'API',
      version: '1.0.0',
      description: 'Some desc :)',
    },
    servers: [
      {
        url: 'http://localhost:' + ENV.APP.SERVER_PORT + '/api/v1',
      },
    ],
  },
  apis: ['./src/swagger-options/swagger-docs/*.ts'],
};

const specs = swaggerJSDoc(swaggerOptions);

export { specs };

const initTransactionApi = (apiRouter: Router): Router => {
  const transactionRouter = Router();

  apiRouter.use(ApiPath.TRANSACTIONS, transactionRouter);


transactionRouter.get(TransactionApiPath.$MERCHANT, async (_req, res) => {
    try {
      const transactions = await transactionService.getTransactionsByMerchantId(
        _req.params.merchant as unknown as number,
      );

res.status(HttpCode.OK).json(transactions);
    } catch (error) {
```

```
res.status(HttpCode.BAD_REQUEST).json(erro
r);
    }
  });


transactionRouter.get(TransactionApiPath.R
OOT, async (_req, res) => {
    try {
      const    transactions    =    await
transactionService.getAllTransactions();

res.status(HttpCode.OK).json(transactions)
;
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });


transactionRouter.post(TransactionApiPath.
CREATE, async (_req, res) => {
    try {
      const    newTransaction    =    await
transactionService.createTransaction(
        _req.body,
      );

res.status(HttpCode.OK).send(newTransactio
n);
    } catch (error) {

res.status(HttpCode.BAD_REQUEST).json(erro
r);
    }
  });

  return transactionRouter;
};

export { initTransactionApi };

const  initRoleApi  =  (apiRouter:  Router):
Router => {
  const roleRouter = Router();

  apiRouter.use(ApiPath.ROLES,
roleRouter);

  roleRouter.get(RolesApiPath.ROOT,   async
(_req, res, next) => {
    try {
      const       roles       =       await
roleService.getAllRoles();
      res.status(HttpCode.OK).json(roles);
    } catch (error) {
      next(error);
    }
  });

  roleRouter.get(RolesApiPath.$ID,    async
(_req, res, next) => {
    try {
      const       role       =       await
roleService.getByIdWithPermissions(_req.pa
rams.id);
      res.status(HttpCode.OK).json(role);
    } catch (error) {
      next(error);
    }
  });

  roleRouter.post(RolesApiPath.ROOT,  async
(_req, res, next) => {
    try {
      await
roleSchema.validate(_req.body);
      const       role       =       await
roleService.createNewRole(_req.body);
      res.status(HttpCode.OK).json(role);
    } catch (error) {
      next(error);
    }
```

```
  });

  roleRouter.put(RolesApiPath.$ID,    async
(_req, res, next) => {
    try {
      await
roleSchema.validate(_req.body);
      const    roleUpdateInfo    =    await
roleService.updateRole(
        _req.params.id,
        _req.body,
      );

res.status(HttpCode.OK).json(roleUpdateInf
o[0]);
    } catch (error) {
      next(error);
    }
  });

  roleRouter.delete(RolesApiPath.$ID,
async (_req, res, next) => {
    try {
      await
roleService.deleteRole(_req.params.id);

res.status(HttpCode.NO_CONTENT).json();
    } catch (error) {
      next(error);
    }
  });

  return roleRouter;
};

export { initRoleApi };

const roleSchema = Yup.object().shape({
  name:  Yup.string().trim().required('Role
name is required'),
});

export { roleSchema }
```

```
const  initAuthApi  =  (apiRouter:  Router):
Router => {
  const authRouter = Router();
  apiRouter.use(ApiPath.AUTH, authRouter);
  authRouter.post(
    AuthApiPath.LOGIN,
    yupValidation(loginSchema),
    async (_req, _res, next) => {
      try {
        const  {  data,  code  }  =  await
authService.loginUser(_req.body);
        _res
          .status(code)
          .json(
            typeof  data  ===  'string'  ?  {
message: data } : { tokens: data },
          );
      } catch (error) {
        next(error);
      }
    },
  );

  authRouter.post(
    AuthApiPath.REFRESH_TOKEN,
    refreshTokenValidation,
    async (_req, _res) => {
      const      tokens      =      await
getTokens(_req.body.userId);

_res.status(HttpCode.OK).json(tokens);
    },
  );

  authRouter.get(AuthApiPath.LOGIN,
jwtValidation, async (req, res) => {
    // endpoint for testing jwtValidation
    res.json(req.user.userId);
  });

  authRouter.post(
    AuthApiPath.RESET_PASSWORD,
```

```
    getPlatform,
    yupValidation(resetSchema),
    async (_req, res) => {
      try {
        const { code, data } = await
authService.resetPassword(_req.body);
        res.status(code).json({ message:
data });
      } catch (error) {

res.status(HttpCode.BAD_REQUEST).json(erro
r);
      }
    },
  );



  authRouter.post(
    AuthApiPath.VERIFY_PASSWORD,
    yupValidation(verifySchema),
    async (_req, res) => {
      try {
        const { code, data } = await
authService.verifyPassword(_req.body);
        res.status(code).json({ message:
data });
      } catch (error) {

res.status(HttpCode.BAD_REQUEST).json(erro
r);
      }
    },
  );

  return authRouter;
};

export { initAuthApi };

const initUserApi = (apiRouter: Router):
Router => {
  const userRouter = Router();

  apiRouter.use(ApiPath.USERS,
userRouter);

  userRouter.get(UsersApiPath.ROOT, async
(_req, res) => {
    try {
      const users = await
userService.getAllUsers();
      res.status(HttpCode.OK).json(users);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });

  // -- get auth profile --
  userRouter.get(UsersApiPath.PROFILE,
jwtValidation, async (_req, res) => {
    try {
      const user = await
userService.getUserById(_req.user.userId);
      res.status(HttpCode.OK).json(user);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });

  // -- update auth profile --
  userRouter.put(UsersApiPath.PROFILE,
jwtValidation, async (_req, res) => {
    try {
      const result = await
userService.updateUserProfile(_req.body);
      if (!result) {
        return
res.status(HttpCode.BAD_REQUEST).send();
      }

res.status(HttpCode.OK).json(result[0]);
```

```
    } catch (error) {
      console.log(error);

res.status(HttpCode.BAD_REQUEST).json(erro
r);
    }
  });

  userRouter.post(UsersApiPath.ROOT,
getPlatform, async (_req, res, next) => {
    try {
      await
userSchema.validate(_req.body.user, {
        context: { required: true },
      });
      const  {  code,  data  }  =  await
userService.createNewUser(
        _req.body.user,
        _req.body.role,
      );
      if (code === HttpCode.OK) {
        const   userToActivate   =   await
userService.setUserActivation(data);
        res.status(code).json({   message:
'success', user: userToActivate });
      } else {
        res.status(code).json({   message:
data });
      }
    } catch (error) {
      next(error);
    }
  });

  userRouter.put(UsersApiPath.$ID,   async
(_req, res) => {
    try {
      await
userSchema.validate(_req.body);
      const       user       =       await
userService.updateUser(_req.params.id,
_req.body);
      res.status(HttpCode.OK).json(user);
    } catch (error) {

res.status(HttpCode.BAD_REQUEST).json(erro
r);
    }
  });

  userRouter.delete(UsersApiPath.$ID,
async (_req, res) => {
    try {
      await
userService.deleteUser(_req.params.id);

res.status(HttpCode.NO_CONTENT).json();
    } catch (error) {

res.status(HttpCode.BAD_REQUEST).json(erro
r);
    }
  });

  userRouter.put(
    UsersApiPath.ACTIVATION_REQUEST,
    getPlatform,
    async (_req, res, next) => {
      try {
        const      user      =      await
userService.getUserByEmail(_req.body.email
);
        if (user) {
          await
userService.setUserActivation(user);
          const          message          =
createActivationMessage(
            ActivationStatus.SENT,
            'Mail was sent',
          );

res.status(HttpCode.OK).json(message);
        }

res.status(HttpCode.NOT_FOUND).json('User
not found.');
```

```
      } catch (error) {
        next(error);
      }
    },
  );

  userRouter.put(UsersApiPath.$ACTIVATION,
async (_req, res, next) => {
    try {
      const      outcome      =      await
userService.activateUser(_req.params.token
);
      return res.json(outcome);
    } catch (error) {
      next(error);
    }
  });

  userRouter.post(
    UsersApiPath.PAG_USERS,
    jwtValidation,
    async (_req, _res, next) => {
      try {
        const      info      =      await
userService.getUsersWithPagination(
          _req.body as ISearchFilter,
        );
        _res.status(200).json({      count:
info.count, data: info.rows });
      } catch (error) {
        next(error);
      }
    },
  );

  userRouter.post(
    UsersApiPath.MANAGE,
    jwtValidation,
    // yupValidation(createUserSchema),
    async (_req, _res, next) => {
      try {
        await
userService.createNewUser(_req.body.user,
_req.body.role);
        _res.status(HttpCode.OK).json({
message: ResponseMessages.CONFIRMED });
      } catch (e) {
        next(e);
      }
    },
  );
  userRouter.put(
    UsersApiPath.$MANAGE,
    jwtValidation,
    // yupValidation(updateUserSchema),
    async (_req, _res, next) => {
      try {
        await
userService.updateUserManage(
          _req.body.user,
          _req.body.role,
          _req.params.id,
        );
        _res.status(HttpCode.OK).json({
message: ResponseMessages.CONFIRMED });
      } catch (e) {
        next(e);
      }
    },
  );

  userRouter.put(
    UsersApiPath.$MANAGE_ACTIVE,
    jwtValidation,
    async (_req, _res, next) => {
      try {
        await
userService.setUserActive(_req.params.id);
        _res.status(HttpCode.OK).json({
message: ResponseMessages.CONFIRMED });
      } catch (e) {
        next(e);
      }
    },
```

```
  );

  userRouter.get(UsersApiPath.$ID,    async
(_req, res) => {
    try {
      const       user       =       await
userService.getUserById(_req.params.id);
      res.status(HttpCode.OK).json(user);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });

  userRouter.put(
    UsersApiPath.SECURITY_PASSWORD,
    jwtValidation,
    yupValidation(updatePasswordSchema),
    async (_req, _res, next) => {
      try {
        const      result      =      await
userService.updatePassword(
          _req.body,
          _req.user.userId,
        );
        _res.status(result.code).json({
message: result.data });
      } catch (e) {
        next(e);
      }
    },
  );

  userRouter.post(UsersApiPath.CHECK_ROLE,
roleCheck());

  return userRouter;
};

export { initUserApi };

const userSchema = Yup.object().shape({
  firstName: Yup.string()
    .when('$required', (required: boolean,
schema: StringSchema) => (
      required   ?   schema.required('First
Name is required') : schema
    )),
  lastName: Yup.string()
    .when('$required', (required: boolean,
schema: StringSchema) => (
      required ? schema.required('Last Name
is required') : schema
    )),
  email:    Yup.string().email('Email    is
invalid')
    .when('$required', (required: boolean,
schema: StringSchema) => (
      required ? schema.required('Email is
required') : schema
    )),
  phoneNumber: Yup.string()
    .matches(/^\+?\d{10,12}$/,       'Phone
Number is invalid')
    .when('$required', (required: boolean,
schema: StringSchema) => (
      required ? schema.required('Phone is
required') : schema
    )),
  password: Yup.string()
    .matches(
      /^.*(?=.{6,})((?=.*[!@#$%^&*()\-
_=+{};:,<.>]){1})(?=.*\d)((?=.*[a-
z]){1})((?=.*[A-Z]){1}).*$/,
      'Password must contain at least 6
characters, one uppercase, one number and
one special case character',
    )
    .min(6,   'Password   Must   Contain   6
Characters')
    .when('$required', (required: boolean,
schema: StringSchema) => (
      required ? schema.required('Password
is required') : schema
    )),
```

```
});

export { userSchema }

const apis = [
  initUserApi,
  initAuthApi,
  initFileUploadApi,
  initRoleApi,
  initPermissionApi,
  initProductApi,
  initMerchantApi,
  initTransactionApi
];

const initApi = (app: Router): Router => {
  const apiRouter = Router();
  app.use(AppConfig.API_V1_PREFIX,
apiRouter);

  apis.forEach((api) => api(apiRouter));

  return apiRouter;
};

export { initApi };

const initProductApi = (apiRouter: Router):
Router => {
  const productRouter = Router();

  apiRouter.use(ApiPath.PRODUCTS,
productRouter);

  productRouter.get(ProductApiPath.ROOT,
async (_req, res) => {
    try {
      const      products      =      await
productService.getAllProducts();

res.status(HttpCode.OK).json(products);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });

  productRouter.get(ProductApiPath.NEWEST,
async (_req, res) => {
    try {
      const      products      =      await
productService.getNewestProducts();

res.status(HttpCode.OK).json(products);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });


productRouter.get(ProductApiPath.WITH_DISC
OUND, async (_req, res) => {
    try {
      const      products      =      await
productService.getProductsWithDiscount();

res.status(HttpCode.OK).json(products);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });


productRouter.get(ProductApiPath.CHEAPEST,
async (_req, res) => {
    try {
      const      products      =      await
productService.getCheapestProducts();

res.status(HttpCode.OK).json(products);
```

```
    } catch (error) {
res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });

  return productRouter;
};

export { initProductApi };

const initFileUploadApi = (apiRouter:
Router): Router => {
  const fileUploadRouter = Router();

  apiRouter.use(ApiPath.FILE_UPLOAD,
fileUploadRouter);


fileUploadRouter.post(FileUploadApiPath.RO
OT, jwtValidation, (_req, _res) => {
fileUploadService.getSingleParser('image')
(_req, _res, (err: unknown) => {
      if (err) {
        console.log(err);
        return
_res.status(HttpCode.BAD_REQUEST).json({
error: err });
      }
      _res.status(HttpCode.OK).json({  url:
_req.file?.path });
    });
  });

  return fileUploadRouter;
};

export { initFileUploadApi };

const initPermissionApi = (apiRouter:
Router): Router => {
  const permissionRouter = Router();

  apiRouter.use(ApiPath.PERMISSIONS,
permissionRouter);


permissionRouter.get(PermissionsApiPath.RO
OT, async (_req, res, next) => {
    try {
      const permissions = await
permissionService.getAllPermissions();

res.status(HttpCode.OK).json(permissions);
    } catch (error) {
      next(error);
    }
  });


permissionRouter.get(PermissionsApiPath.$I
D, async (_req, res, next) => {
    try {
      const permission = await
permissionService.getPermissionById(
        _req.params.id,
      );

res.status(HttpCode.OK).json(permission);
    } catch (error) {
      next(error);
    }
  });


permissionRouter.post(PermissionsApiPath.R
OOT, async (_req, res, next) => {
    try {
      await
permissionSchema.validate(_req.body);
      const permission = await
permissionService.createNewPermission(_req
.body);
```

```
res.status(HttpCode.OK).json(permission);
    } catch (error) {
      next(error);
    }
  });


permissionRouter.put(PermissionsApiPath.$ID, async (_req, res, next) => {
    try {
      await permissionSchema.validate(_req.body);
      const permissionUpdateInfo = await permissionService.updatePermission(
        _req.params.id,
        _req.body,
      );

res.status(HttpCode.OK).json(permissionUpdateInfo[0]);
    } catch (error) {
      next(error);
    }
  });


permissionRouter.delete(PermissionsApiPath.$ID, async (_req, res, next) => {
    try {
      await permissionService.deletePermission(_req.params.id);

res.status(HttpCode.NO_CONTENT).json();
    } catch (error) {
      next(error);
    }
  });

  return permissionRouter;
};

export { initPermissionApi };

const permissionSchema = Yup.object().shape({
  name: Yup.string().trim().required('Permission name is required'),
});

export { permissionSchema };

const initMerchantApi = (apiRouter: Router): Router => {
  const merchantRouter = Router();

  apiRouter.use(ApiPath.MERCHANTS, merchantRouter);

  merchantRouter.get('/', async (_req, res) => {
    const startId: number = _req.query.startId as unknown as number;
    const limit: number = _req.query.limit as unknown as number;
    try {
      const merchants = await merchantService.getAllMerchants(startId, limit);

res.status(HttpCode.OK).json(merchants);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error);
    }
  });

  merchantRouter.get('/filter', async (_req, res) => {
    try {
      const merchants = await merchantService.getMerchantsByProperty(
        _req.query,
```

```
      );

res.status(HttpCode.OK).json(merchants);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });


merchantRouter.get('/name_contains=:contai
ns', async (_req, res) => {
    const      contains:      string      =
_req.params.contains;
    try {
      const      merchants      =      await
merchantService.getMerchantsThatMatch(cont
ains);

res.status(HttpCode.OK).json(merchants);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });


merchantRouter.get('/category=:category',
async (_req, res) => {
    try {
      const      merchants      =      await
merchantService.getMerchantsByCategory(
        _req.params.category,
      );

res.status(HttpCode.OK).json(merchants);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });

  merchantRouter.get('/popular',      async
(_req, res) => {
    try {
      const      merchants      =      await
merchantService.getPopularMerchants();

res.status(HttpCode.OK).json(merchants);
    } catch (error) {

res.status(HttpCode.NOT_FOUND).json(error)
;
    }
  });

  return merchantRouter;
};

export { initMerchantApi };
  redisPort,
  redisHost,
  redisPassword,
} from '../../../config/redis.config';

const port = parseInt(redisPort || '') ||
0;
const            redis_client            =
redis.createClient(port, redisHost);

if (redisPassword) {
  redis_client.auth(redisPassword);
}
redis_client.on('connect', function () {
  logger.log('Redis client connected');
});

redis_client.on('error', function () {
  logger.log('Redis connection failed');
});

export default redis_client;
```

```
class UserRolesRepository {
  public getAll():Promise<IUserRoles[]>{
    return UserRoleModel.findAll()
  }
  public
getById(id:string):Promise<IUserRoles     |
null>{
    return UserRoleModel.findByPk(id)
  }

  public
getByUserId(userId:string):Promise<IUserRo
les | null>{
    return UserRoleModel.findOne({ where: {
userId } })
  }

  public
createUserRole(role:IUserRoles):Promise<IU
serRoles>{
    return UserRoleModel.create(role)
  }
  public    async    updateById(id:string,
data:IUserRoles):Promise<IUserRoles[]>{
    const       result       =       await
UserRoleModel.update(data, {
      where: { id },
      returning: true
    });
    return result[1];
  }
  public
deleteById(id:string):Promise<number>{
    return UserRoleModel.destroy({
      where: { id }
    });
  }
}

export { UserRolesRepository };

class RoleRepository {
  public getAll(): Promise<IRole[]> {
    return RoleModel.findAll({
      attributes: ['name'],
      include: [
        {
          model: PermissionModel,
          attributes: ['key'],
          through: {
            attributes: [],
          },
        },
      ],
    });
  }
  public getById(id: string): Promise<IRole
| null> {
    return RoleModel.findByPk(id);
  }
  public     createRole(role:     IRole):
Promise<IRole> {
    return RoleModel.create(role);
  }
  public async updateById(id: string, data:
IRole): Promise<IRole[]> {
    const       result       =       await
RoleModel.update(data, {
      where: { id },
      returning: true,
    });
    return result[1];
  }
  public      deleteById(id:      string):
Promise<number> {
    return RoleModel.destroy({
      where: { id },
    });
  }
  public  async  getByIdWithPermissions(id:
string): Promise<IRole | null> {
    return await RoleModel.findByPk(id, {
      attributes: ['name'],
      include: [
        {
          model: PermissionModel,
```

```
          attributes: ['key'],
          through: {
            attributes: [],
          },
        },
      ],
    });
  }
}

export { RoleRepository };

const attributes: string[] = ['id', 'sid',
'image_url', 'name', 'link'];

class MerchantRepository {
  public getAll(startId?: number, limit?:
number): Promise<IMerchant[]> {
    const condition =
      startId && limit
        ? {
            id: {
              [Op.between]:       [startId,
startId + limit],
            },
          }
        : {};
    return MerchantModel.findAll({
      attributes,
      where: condition,
      include: {
        model: UserModel,
      },
    });
  }

  public               getWithCategories():
Promise<IMerchant[]> {
    return MerchantModel.findAll({
      attributes,
      include: {
        model: CategoryModel,
        attributes: ['name'],
      },
    });
  }

  public     getByProperty(prop:     any):
Promise<IMerchant[]> {
    return MerchantModel.findAll({
      attributes,
      where: prop,
      include: {
        model: CategoryModel,
        attributes: ['name'],
      },
    });
  }

  public       getThoseThatMatch(substring:
string): Promise<IMerchant[]> {
    return MerchantModel.findAll({
      attributes,
      where: {
        name: {
          [Op.substring]: substring,
        },
      },
      include: {
        model: CategoryModel,
        attributes: ['name'],
      },
    });
  }

  public       getBySid(sid:       number):
Promise<IMerchant | null> {
    return MerchantModel.findOne({
      attributes,
      where: {
        sid,
      },
      include: {
        model: CategoryModel,
        attributes: ['name'],
      },
```

```
    });
  }
}

export { MerchantRepository };

class PermissionRepository {
  public getAll(): Promise<IPermission[]> {
    return PermissionModel.findAll();
  }

  public getById(id: string): Promise<IPermission | null> {
    return PermissionModel.findByPk(id);
  }

  public createPermission(role: IPermission): Promise<IPermission> {
    return PermissionModel.create(role);
  }

  public async updateById(
    id: string,
    data: IPermission,
  ): Promise<IPermission[]> {
    const result = await PermissionModel.update(data, {
      where: { id },
      returning: true,
    });
    return result[1];
  }

  public deleteById(id: string): Promise<number> {
    return PermissionModel.destroy({
      where: { id },
    });
  }
}

export { PermissionRepository };
  MerchantModel,
  PermissionModel,
  RoleModel,
  UserModel,
} from '../models';

const include = [
  {
    model: RoleModel,
    attributes: ['name', 'key', 'id'],
    through: {
      attributes: [],
    },
  },
  {
    model: MerchantModel,
    attributes: ['id', 'name', 'image_url'],
  },
];

class UserRepository {
  public getAll(): Promise<IUser[]> {
    return UserModel.findAll({
      include,
    });
  }

  public async getById(id: string): Promise<IUser | null> {
    const user = await UserModel.findByPk(id, {
      include,
    });

    return user;
  }

  public createUser(user: IUser): Promise<IUser> {
    return UserModel.create(user);
  }
```

```
  public async updateById(id: string, data: IUser): Promise<IUser[]> {
    const result = await UserModel.update(data, {
      where: { id },
      returning: true,
    });
    return result[1];
  }

  public deleteById(id: string): Promise<number> {
    return UserModel.destroy({
      where: { id },
    });
  }

  public async getByEmail(email: string): Promise<IUser | null> {
    return await UserModel.findOne({
      include: [
        {
          model: RoleModel,
          attributes: ['name', 'key'],
          through: {
            attributes: [],
          },
          include: [
            {
              model: PermissionModel,
              attributes: ['name', 'key'],
              through: {
                attributes: [],
              },
            },
          ],
        },
      ],
      where: { email },
    });
  }

  public async getByToken(activationToken: string): Promise<IUser | null> {
    return await UserModel.findOne({ where: { activationToken } });
  }

  public async activateUser(
    activationToken: string,
    data: IUser,
  ): Promise<IUser[]> {
    const result = await UserModel.update(data, {
      where: { activationToken },
      returning: true,
    });
    return result[1];
  }

  public async getUsersWithPagination(
    data: ISearchFilter,
  ): Promise<{ rows: IUser[]; count: number }> {
    const { filter, limit, page, search } = data;
    const offset = (page - 1) * limit;
    const isActive = filter === 'all' ? null : filter !== 'online';
    return await UserModel.findAndCountAll({
      where: {
        [Op.or]: [
          { firstName: { [Op.iLike]: `%${search.trim()}%` } },
          { lastName: { [Op.iLike]: `%${search.trim()}%` } },
          { phoneNumber: { [Op.substring]: search.trim() } },
        ],
        isActive: { [Op.not]: isActive },
      },
      offset: +offset,
      limit: +limit,
    });
```

```
  }
}

export { UserRepository };

const userRepository = new UserRepository();
const roleRepository = new RoleRepository();
const permissionRepository = new PermissionRepository();
const userRolesRepository = new UserRolesRepository();
const productRepository = new ProductRepository();
const transactionRepository = new TransactionRepository();
const merchantRepository = new MerchantRepository();

export {
  userRepository,
  roleRepository,
  permissionRepository,
  userRolesRepository,
  productRepository,
  merchantRepository,
  transactionRepository
};

const attributes: string[] = [
  'name',
  'image_url',
  'partner_code',
  'price',
  'created_at',
];

class ProductRepository {
  public getAll(): Promise<IProduct[]> {
    return ProductModel.findAll({
      attributes,
      include: {
        model: MerchantModel,
      },
    });
  }

  public getNewest(): Promise<IProduct[]> {
    return ProductModel.findAll({
      attributes,
      order: [['created_at', 'DESC']],
    });
  }

  public getWithDiscount(): Promise<IProduct[]> {
    return ProductModel.findAll({
      attributes,
      where: {
        discount: {
          [Op.gt]: 0,
        },
      },
    });
  }

  public getCheapest(): Promise<IProduct[]> {
    return ProductModel.findAll({
      attributes,
      order: [['price', 'ASC']],
    });
  }
}

export { ProductRepository };

const attributes: string[] = [
  'user_id',
  'merchant_id',
  'processed',
  'amount',
  'order_details',
  'expiration_date',
  'order_id',
```

```
];

class TransactionRepository {
  public getAll(): Promise<ITransaction[]>
{
    return TransactionModel.findAll({
      attributes,
    });
  }

  public       getByMerchantId(merchant_id:
number): Promise<ITransaction[]> {
    return TransactionModel.findAll({
      attributes,
      where: {
        merchant_id,
      },
    });
  }

  public  getByOrderId(order_id:  number):
Promise<ITransaction[]> {
    return TransactionModel.findAll({
      attributes,
      where: {
        order_id,
      },
    });
  }

  public createTransaction(
    transaction: ITransactionExtended,
  ): Promise<ITransaction | null> {
    return
TransactionModel.create(transaction);
  }
}

export { TransactionRepository };


interface     CategoryInstance     extends
ICategory, Model {}

const    createCategoryModel    =    (orm:
Sequelize): ModelCtor<CategoryInstance> =>
{
  const          CategoryModel          =
orm.define<CategoryInstance>(
    ModelName.CATEGORY,
    {
      id: {
        field: 'id',
        autoIncrement: true,
        allowNull: false,
        primaryKey: true,
        type: DataTypes.NUMBER,
      },
      name: {
        field: 'name',
        allowNull: false,
        type: DataTypes.STRING,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    },
    {
      tableName: 'categories',
    },
  );
  return CategoryModel;
};

export default createCategoryModel;

interface ProductInstance extends IProduct,
Model {}

const    createProductModel    =    (orm:
Sequelize): ModelCtor<ProductInstance> => {
  const           ProductModel           =
orm.define<ProductInstance>(
    ModelName.PRODUCT,
    {
      name: {
        field: 'name',
```

```
    allowNull: false,
    type: DataTypes.STRING,
  },
  merchant_id: {
    field: 'merchant_id',
    allowNull: false,
    type: DataTypes.NUMBER,
  },
  imageUrl: {
    field: 'image_url',
    allowNull: true,
    type: DataTypes.STRING,
  },
  partnerCode: {
    field: 'partner_code',
    allowNull: false,
    type: DataTypes.NUMBER,
  },
  price: {
    field: 'price',
    allowNull: false,
    type: DataTypes.NUMBER,
  },
  discount: {
    field: 'discount',
    allowNull: true,
    type: DataTypes.NUMBER,
  },
  expirationDate: {
    field: 'expiration_date',
    allowNull: false,
    type: DataTypes.DATE,
  },
  createdAt: {
    field: 'created_at',
    type: DataTypes.DATE,
  },
  updateddAt: {
    field: 'updated_at',
    type: DataTypes.DATE,
  },
},
{
    tableName: 'products',
  },
);
return ProductModel;
};

export default createProductModel;

export interface UserRoleInstance extends
IUserRoles, Model {
}

const createUserRoleModel = (orm:
Sequelize): ModelCtor<UserRoleInstance> =>
{
  const UserRoleModel =
orm.define<UserRoleInstance>(
    ModelName.USER_ROLES, {
      roleId: {
        allowNull: false,
        type: DataTypes.NUMBER,
      },
      userId: {
        allowNull: false,
        type: DataTypes.NUMBER,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    }, {
      tableName: 'user_roles',
    });
  return UserRoleModel;
};
export default createUserRoleModel;

export interface RoleInstance extends
IRole, Model {}

const createRoleModel = (orm: Sequelize):
ModelCtor<RoleInstance> => {
  const RoleModel =
orm.define<RoleInstance>(
    ModelName.ROLE,
```

```
    {
      name: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      key: {
        type: DataTypes.STRING,
        allowNull: true,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    },
    {
      tableName: 'roles',
    },
  );
  return RoleModel;
};

export default createRoleModel;
  UserModel,
  RoleModel,
  PermissionModel,
  MerchantModel,
  CategoryModel,
} from './index';

export function createRolePermissionAssociations(): void {
  RoleModel.belongsToMany(PermissionModel, { through: 'role_permissions' });
  PermissionModel.belongsToMany(RoleModel, { through: 'role_permissions' });
}

export function createUserRolesAssociations(): void {
  UserModel.belongsToMany(RoleModel, { through: 'user_roles' });
  RoleModel.belongsToMany(UserModel, { through: 'user_roles' });
}

export function createMerchantCategoryAssociations(): void {
  MerchantModel.belongsTo(CategoryModel, { foreignKey: 'category_id' });
}

export function createMerchantUserAssociations(): void {
  MerchantModel.belongsTo(UserModel, { foreignKey: 'user_id' });
  UserModel.hasOne(MerchantModel, { foreignKey: 'user_id' });
}

interface TransactionInstance extends ITransactionExtended, Model {}

const createTransactionModel = (
  orm: Sequelize,
): ModelCtor<TransactionInstance> => {
  const ProductModel = orm.define<TransactionInstance>(
    ModelName.TRANSACTIONS,
    {
      id: {
        field: 'id',
        autoIncrement: true,
        allowNull: false,
        primaryKey: true,
        type: DataTypes.NUMBER,
      },
      processed: {
        field: 'processed',
        allowNull: false,
        type: DataTypes.BOOLEAN,
      },
      merchant_id: {
        field: 'merchant_id',
        allowNull: false,
        type: DataTypes.NUMBER,
      },
      order_id: {
```

```
    field: 'order_id',
    allowNull: false,
    type: DataTypes.STRING,
  },
  user_id: {
    field: 'user_id',
    allowNull: false,
    type: DataTypes.NUMBER,
  },
  amount: {
    field: 'amount',
    allowNull: false,
    type: DataTypes.NUMBER,
  },
  order_details: {
    field: 'order_details',
    allowNull: false,
    type: DataTypes.STRING,
  },
  authorization: {
    field: 'authorization',
    allowNull: true,
    type: DataTypes.STRING,
  },
  expiration_date: {
    field: 'expiration_date',
    allowNull: false,
    type: DataTypes.DATE,
  },
  createdAt: {
    field: 'createdAt',
    allowNull: false,
    type: DataTypes.DATE,
  },
  updatedAt: {
    field: 'updatedAt',
    allowNull: false,
    type: DataTypes.DATE,
  },
},
{
  tableName: 'transactions',
},
```

```
  );
  return ProductModel;
};

export default createTransactionModel;

export interface PermissionInstance extends
IPermission, Model {}

const createPermissionModel = (
  orm: Sequelize,
): ModelCtor<PermissionInstance> => {
  const          PermissionModel          =
orm.define<PermissionInstance>(
    ModelName.PERMISSION,
    {
      name: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      key: {
        type: DataTypes.STRING,
        allowNull: true,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    },
    {
      tableName: 'permissions',
    },
  );
  return PermissionModel;
};

export default createPermissionModel;

  createMerchantCategoryAssociations,
  createRolePermissionAssociations,
  createUserRolesAssociations,
  createMerchantUserAssociations,
} from './associations';
```

```
const              UserModel              =
createUserModel(sequelize);
const              RoleModel              =
createRoleModel(sequelize);
const           PermissionModel           =
createPermissionModel(sequelize);
const            UserRoleModel            =
createUserRoleModel(sequelize);
const             ProductModel            =
createProductModel(sequelize);
const            MerchantModel            =
createMerchantModel(sequelize);
const           TransactionModel          =
createTransactionModel(sequelize);
const            CategoryModel            =
createCategoryModel(sequelize);

createRolePermissionAssociations();
createUserRolesAssociations();
createMerchantCategoryAssociations();
createMerchantUserAssociations();

export {
  UserModel,
  RoleModel,
  PermissionModel,
  UserRoleModel,
  ProductModel,
  MerchantModel,
  TransactionModel,
  CategoryModel,
};


interface UserInstance extends IUser, Model
{}

const createUserModel = (orm: Sequelize):
ModelCtor<UserInstance> => {
  const              UserModel              =
orm.define<UserInstance>(
    ModelName.USER,
    {
      sid: {
        field: 'sid',
        allowNull: false,
        type: DataTypes.NUMBER,
      },
      firstName: {
        field: 'first_name',
        allowNull: false,
        type: DataTypes.STRING,
      },
      lastName: {
        field: 'last_name',
        allowNull: false,
        type: DataTypes.STRING,
      },
      phoneNumber: {
        field: 'phone_number',
        allowNull: false,
        type: DataTypes.STRING,
      },
      email: {
        allowNull: false,
        type: DataTypes.STRING,
      },
      password: {
        allowNull: false,
        type: DataTypes.STRING,
        unique: true,
      },
      birthDate: {
        field: 'birth_date',
        allowNull: true,
        type: DataTypes.DATE,
      },
      avatar: {
        field: 'avatar',
        allowNull: true,
        type: DataTypes.STRING,
      },
      isActive: {
        field: 'is_active',
        type: DataTypes.BOOLEAN,
        defaultValue: false,
```

```
      },
      activationTokenExpiration: {
        field:
'activation_token_expiration',
        type: DataTypes.DATE,
        defaultValue: Date.now() + 1000 *
60 * 60,
      },
      activationToken: {
        field: 'activation_token',
        type: DataTypes.STRING,
      },
      postalCode: {
        field: 'postal_code',
        type: DataTypes.STRING,
        defaultValue: '',
      },
      stateAddress: {
        field: 'state_address',
        type: DataTypes.STRING,
        defaultValue: '',
      },
      cityAddress: {
        field: 'city_address',
        type: DataTypes.STRING,
        defaultValue: '',
      },
      streetAddress: {
        field: 'street_address',
        type: DataTypes.STRING,
        defaultValue: '',
      },
      marriageStatus: {
        field: 'marriage_status',
        type: DataTypes.BOOLEAN,
        defaultValue: false,
      },
      dependantsAmount: {
        field: 'dependants_amount',
        type: DataTypes.NUMBER,
        defaultValue: 0,
      },
      balance: {
        field: 'balance',
        type: DataTypes.NUMBER,
        defaultValue: 0,
      },
      createdAt: DataTypes.DATE,
      updatedAt: DataTypes.DATE,
    },
    {
      tableName: 'users',
    },
  );
  return UserModel;
};

export default createUserModel;

interface     MerchantInstance     extends
IMerchant, Model {}

const    createMerchantModel    =    (orm:
Sequelize): ModelCtor<MerchantInstance> =>
{
  const           MerchantModel           =
orm.define<MerchantInstance>(
    ModelName.MERCHANT,
    {
      sid: {
        field: 'sid',
        allowNull: false,
        type: DataTypes.NUMBER,
      },
      category_id: {
        field: 'category_id',
        allowNull: false,
        type: DataTypes.NUMBER,
      },
      user_id: {
        field: 'user_id',
        allowNull: false,
        type: DataTypes.NUMBER,
      },
      image_url: {
        field: 'image_url',
```

```
        allowNull: false,
        type: DataTypes.STRING,
      },
      name: {
        field: 'name',
        allowNull: false,
        type: DataTypes.STRING,
      },
      link: {
        field: 'link',
        allowNull: false,
        type: DataTypes.STRING,
      },
      created_at: {
        field: 'created_at',
        type: DataTypes.DATE,
      },
      updated_at: {
        field: 'updated_at',
        type: DataTypes.DATE,
      },
    },
    {
      tableName: 'merchants',
    },
  );

  return MerchantModel;
};

export default createMerchantModel;

const env = (process.env.NODE_ENV as keyof
typeof envConfigs) || 'development';
const config = envConfigs[env];

const { database, username, password, host,
port, dialect } = config;

const sequelize = new Sequelize({
  port: Number(port),
  dialect: dialect as Dialect,
  database,
  username,
  password,
  host,
  logging: false,
});

export { sequelize };

require('dotenv').config();

const app = express();

sequelize
  .authenticate()
  .then(() => {
    return logger.log('Database connection
was successful');
  })
  .catch(({     message,     stack     }:
DbConnectionError) => {
    return logger.error(message, stack);
  });

app.use(setTraceId);
app.use(logRequest);
app.use(json());
app.use(urlencoded({ extended: true }));
app.use(cors());

initApi(app);
app.use('/api-docs',       swaggerUI.serve,
swaggerUI.setup(specs));

app.use(express.static(join(__dirname,
'../public')));
app.use('*', (_req, res) => {
  return       res.sendFile(join(__dirname,
'../public', 'index.html'));
});
app.use(handleError);

const               server               =
app.listen(ENV.APP.SERVER_PORT, () => {
```

```
  logger.log(`Listening to connections on port — ${ENV.APP.SERVER_PORT}`);
});

export { server, app };

const verifyJWT = (token: string, secret = '') => {
  return jwt.verify(token, secret as Secret);
};

export { verifyJWT };
export * from './create-jwt/create-jwt.helper';
export * from './verify-jwt/verify-jwt.helper';
export * from './create-refresh-token/create-refresh-token.helper';

const createJWT = (id: string, permissions: string[] = []) => {
  return jwt.sign(
    {
      userId: id,
      permissions,
    },
    secret as Secret,
    { expiresIn: expire },
  );
};

export { createJWT };

const createRefreshToken = (userId: string): Promise<string> => {
  return new Promise((resolve, reject) => {
    const payload = { userId: userId };
    const options = { expiresIn: expireRefresh };
    const secret = secretRefresh as Secret;
    const refreshToken = jwt.sign(payload, secret, options);
    redis_client.SET(userId, refreshToken, (err) => {
      if (err) {
reject(HttpCode.INTERNAL_SERVER_ERROR);
        return;
      }
      resolve(refreshToken);
    });
  });
};

export { createRefreshToken };
const calculateAmountToPay = (payPerUnit: number, unitsAmount: number): number => {
  return payPerUnit * unitsAmount;
};

export { calculateAmountToPay };
export * from './calculate-amountToPay.helper';

const createMail = (theme: string, link: string): MailOptions => {
  return {
    mailTheme: theme,
    data: {
      link: link,
    },
  };
};

export { createMail };
export * from './create-email.helper';

const hashToken = (data: string): string => {
  return createHash('sha256').update(data).digest('hex');
};
```

```
export { hashToken };

const isMatchPassword = async (reqPassword: string, dbPassword: string): Promise<boolean> => {
  return await bcrypt.compare(reqPassword, dbPassword);
};

export { isMatchPassword };

const hashPassword = async (password: string): Promise<string> => {
  return await bcrypt.hash(password, 12);
};

export { hashPassword };
export * from './bcrypt-password/bcrypt-compare-password.helper';
export * from './bcrypt-password/bcrypt-hash-password.helper';
export * from './bcrypt-token/bcrypt-hash-token.helper'


const createActivationMessage = (
  status: ActivationStatus,
  message: string,
  email: string | null = null
) : IActivationMessage => ({
  status,
  message,
  email
});

export { createActivationMessage };
export * from './create-activation-message/create-activation-message.helper';

const getUpdatedUser = (user: IUser, newUserData: IProfile): IUser => {
  return {
    id: user.id,
    sid: user.sid,
    balance: newUserData.balance,
    firstName: newUserData.firstName,
    lastName: newUserData.lastName,
    email: newUserData.email,
    phoneNumber: newUserData.phoneNumber,
    birthDate: new Date(newUserData.birthDate),
    avatar: newUserData.avatar,
    cityAddress: user.cityAddress,
    dependantsAmount: user.dependantsAmount,
    marriageStatus: user.marriageStatus,
    postalCode: user.postalCode,
    stateAddress: user.stateAddress,
    streetAddress: user.streetAddress,
    updatedAt: user.updatedAt,
    activationTokenExpiration: user.activationTokenExpiration,
    activationToken: user.activationToken,
    password: user.password,
    isActive: user.isActive,
    createdAt: user.createdAt,
  };
};

export { getUpdatedUser };
const adminUpdatedUser = (user: IUser, newUserData: IUser): IUser => {
  return {
    id: user.id,
    sid: user.sid,
    balance: newUserData.balance,
    firstName: newUserData.firstName,
    lastName: newUserData.lastName,
    email: newUserData.email,
    phoneNumber: newUserData.phoneNumber,
    birthDate: new Date(newUserData.birthDate as Date),
    avatar: user.avatar,
    cityAddress: newUserData.cityAddress,
    dependantsAmount: newUserData.dependantsAmount,
```

```
    marriageStatus:
newUserData.marriageStatus,
    postalCode: newUserData.postalCode,
    stateAddress:
newUserData.stateAddress,
    streetAddress:
newUserData.streetAddress,
    updatedAt: user.updatedAt,
    activationTokenExpiration:
user.activationTokenExpiration,
    activationToken: user.activationToken,
    password: user.password,
    isActive: user.isActive,
    createdAt: user.createdAt,
  };
};

export { adminUpdatedUser };
interface IPermission {
  name: string;
  key: string;
}

interface IRole {
  name: string;
  key: string;
  permissions: IPermission[];
}

const getUserPermissions = (user: any):
string[] => {
  const { roles } = user;

  const allPermissions: string[] =
roles.reduce(
    (acc: string[], role: IRole) => {
      const { permissions } = role;

      const rolePermissions =
permissions.reduce((acc: string[], { key })
=> {
        return [...acc, key];
      }, []);
      return [...acc, ...rolePermissions];
    },
    [],
  );

  return [...new Set(allPermissions)];
};

export { getUserPermissions };
export * from './get-updated-user/get-
updated-user';
export * from './get-user-permissions/get-
user-permisssions';

class UrlHelper {
  private _appHost: string;

  constructor() {
    this._appHost = '';
  }

  public set appHost(host: string) {
    this._appHost = host;
  }

  public getFullUrl(path: string): string {
    const trimmedPath =
path.replace(/(^\/)|(\/$)/g, '');
    return protocol + '://' + this._appHost
+ '/' + trimmedPath
  }

  public getFullApiUrl(path: string):
string {
    return protocol + '://' + host + ':' +
port + path;
  }

  public strToUrl(url: string): URL {
    return new URL(url);
  }
```

```
  public parseURL (url: string | undefined): URL {
    if(url) {
      return new URL(url);
    }
    throw new TypeError(`UrlHelper: Cannot parse non-url ${url} with type ${typeof url}`)
  }
}

const urlHelper = new UrlHelper();

export { urlHelper }
export * from './url.helper';
export { checkIsOneOf } from 'shared/helpers';
const comparePasswords = (pass: string, verPass: string): boolean => pass.trim() === verPass.trim();

export {comparePasswords}
export * from './compare-strings.helper';

const getRandomId = (size = DEFAULT_ID_LENGTH): string => nanoid(size);

export { getRandomId };
const DEFAULT_ID_LENGTH = 10;

export { DEFAULT_ID_LENGTH };
export * from './get-random-id/get-random-id.helper';
export * from './compare-strings';
export * from './boolean';
export * from './string';
export * from './tokens';
export * from './jwt';
export * from './bcrypt';
export * from './mail';
export * from './url';
export * from './activation';
export * from './user';
```

```
export * from './calculate';
export * from './get-tokents/get-tokens';

const getTokens = async (
  userId: string,
  permissions: string[] = [],
): Promise<ITokens> => {
  const accessToken = createJWT(userId, permissions);
  const refreshToken = await createRefreshToken(userId);
  return {
    accessToken,
    refreshToken,
  };
};

export { getTokens };

const createMail = (theme: string, link: string): MailOptions => {
  return {
    mailTheme: theme,
    data: {
      link: link,
    },
  };
};

export { createMail };
export * from './create-email/create-email.helper'

class ProductService {
  public getAllProducts(): Promise<IProduct[]> {
    return productRepository.getAll();
  }

  public getNewestProducts(): Promise<IProduct[]> {
    return productRepository.getNewest();
  }
```

```
  public         getProductsWithDiscount():
Promise<IProduct[]> {
    return
productRepository.getWithDiscount();
  }

  public             getCheapestProducts():
Promise<IProduct[]> {
    return
productRepository.getCheapest();
  }
}

export { ProductService };

class PermissionService {
  public               getAllPermissions():
Promise<IPermission[]> {
    return permissionRepository.getAll();
  }
  public   getPermissionById(id:   string):
Promise<IPermission | null> {
    return
permissionRepository.getById(id);
  }
  public    createNewPermission(permission:
IPermission): Promise<IPermission> {
    return
permissionRepository.createPermission(perm
ission);
  }
  public async updatePermission(
    id: string,
    data: IPermission,
  ): Promise<IPermission[]> {
    return
permissionRepository.updateById(id, data);
  }
  public    deletePermission(id:    string):
Promise<number> {
    return
permissionRepository.deleteById(id);
  }
}

export { PermissionService };

type Constructor = {
  asyncStorage:
AsyncLocalStorage<AppAsyncStorage>;
  logLevel: LogLevel;
};

class Logger {
  #instance: pino.Logger;

  #asyncStorage:
AsyncLocalStorage<AppAsyncStorage>;

  constructor({  asyncStorage,  logLevel  }:
Constructor) {
    this.#asyncStorage = asyncStorage;
    this.#instance = pino({
      prettyPrint: true,
      level: logLevel,
    });
  }

  error(message:  string,  stack?:  string):
void {
    this.#instance.error({         traceId:
this.traceId }, message);

    if (stack) {
      this.#instance.error(stack);
    }
  }

  log(message: string): void {
    return  this.#instance.info({  traceId:
this.traceId }, message);
  }

  warn(message: string): void {
```

```
    return  this.#instance.warn({  traceId:
this.traceId }, message);
  }

  private get traceId(): string | null {
    const traceId = this.#asyncStorage
      .getStore()
      ?.get(AppAsyncStorageKey.TRACE_ID)
as string | null;

    return traceId ?? null;
  }
}

export { Logger };

class UserRolesService {
  public
getAllUsersRoles():Promise<IUserRoles[]>{
    return userRolesRepository.getAll()
  }
  public
getUserRolesById(id:string):Promise<IUserR
oles | null>{
    return userRolesRepository.getById(id)
  }
  public
createNewUserRole(user:IUserRoles):Promise
<IUserRoles>{
    return
userRolesRepository.createUserRole(user)
  }
  public  async  updateUserRole(id:string,
data:IUserRoles):Promise<IUserRoles[]>{
    return
userRolesRepository.updateById(id, data)
  }
  public
deleteUserRole(id:string):Promise<number>{
    return
userRolesRepository.deleteById(id)
  }
}

export { UserRolesService };

export { AsyncLocalStorage };
/*  eslint-disable  @typescript-eslint/no-
var-requires */

const nodemailer = require('nodemailer');

class MailService {
  transporter: Transporter;

  constructor() {
    this.transporter                        =
nodemailer.createTransport({
      host,
      port,
      auth: {
        user,
        pass,
      },
      secure: port === '465',
    });
  }

  async  sendMail(type:  string,  email:
string,       options:       MailOptions):
Promise<Transporter> {
    return this.transporter.sendMail(
      {
        from: user,
        to: email,
        subject: options?.mailTheme,
        html:
compileFile(path.resolve(`${__dirname}/tem
plates/${type}.pug`))({
          ...options?.data,
        }),
      },
    );
  }
```

```
  async sendActivationMail(email: string, token: string): Promise<Transporter> {
    const link = urlHelper.getFullUrl(`/email-activation/${token}`);
    const mailOptions = createMail('Confirm registration', link);
    return await this.sendMail(EmailType.ACTIVATION, email, mailOptions);
  }

  async sendResetPasswordMail(email: string, token: string): Promise<Transporter> {
    const link = urlHelper.getFullUrl(`/verify-refresh/${token}`);
    const mailOptions = createMail(
      `Reset password from ${email} account`,
      link
    );
    return await this.sendMail(EmailType.RESET_PASSWORD, email, mailOptions);
  }
}

export { MailService };

class MerchantService {
  public getAllMerchants(
    startId?: number,
    limit?: number,
  ): Promise<IMerchant[]> {
    return merchantRepository.getAll(startId, limit);
  }

  public async getMerchantsByCategory(category: string): Promise<IMerchant[]> {
    return (await merchantRepository.getWithCategories()).filter(
      (merchant) => merchant.category?.name === category,
    );
  }

  public getMerchantsByProperty(prop: any): Promise<IMerchant[]> {
    return merchantRepository.getByProperty(prop);
  }

  public getMerchantsThatMatch(name: string): Promise<IMerchant[]> {
    return merchantRepository.getThoseThatMatch(name);
  }

  public async getPopularMerchants(): Promise<IMerchant[]> {
    const merchants = await this.getAllMerchants(undefined, undefined);

    const popular: any[] = [];

    for (let i = 0; i < merchants.length; i++) {
      popular.push({
        merchant: merchants[i],
        transactions: (
          await transactionService.getTransactionsByMerchantId(merchants[i].id)
        ).length,
      });
    }

    popular.sort((m1, m2) => m2.transactions - m1.transactions);
```

```
    return popular.map((m) => m.merchant).slice(0, 10);
  }

  public async getMerchantIdBySid(sid: number): Promise<number | null> {
    const merchant: IMerchant | null = await merchantRepository.getBySid(sid);
    if (merchant) {
      return merchant.id;
    } else {
      return null;
    }
  }
}

export { MerchantService };

class FileUploadService {
  parser: multer.Multer;

  constructor() {
    Cloudinary.config({
      cloud_name,
      api_key,
      api_secret,
    });

    const storage = new CloudinaryStorage({
      cloudinary: Cloudinary,
      params: async (req, file) => {
        return {
          folder: 'img',
          format: 'png',
          public_id: nanoid(),
        };
      },
    });

    this.parser = multer({ storage, limits: { fileSize: image_size_limit } });
  }

  getSingleParser(fieldName: string): RequestHandler {
    return this.parser.single(fieldName);
  }
}

export { FileUploadService };
class RoleService {
  public getAllRoles(): Promise<IRole[]> {
    return roleRepository.getAll();
  }
  public getRoleById(id: string): Promise<IRole | null> {
    return roleRepository.getById(id);
  }
  public getRoleByIdWithPermissions(id: string): Promise<IRole | null> {
    return roleRepository.getById(id);
  }
  public createNewRole(industry: IRole): Promise<IRole> {
    return roleRepository.createRole(industry);
  }
  public async updateRole(id: string, data: IRole): Promise<IRole[]> {
    return roleRepository.updateById(id, data);
  }
  public deleteRole(id: string): Promise<number> {
    return roleRepository.deleteById(id);
  }
  public getByIdWithPermissions(id: string): Promise<IRole | null> {
    return roleRepository.getByIdWithPermissions(id);
  }
}

export { RoleService };
  IUser,
```

```
  IActivationMessage,
  IProfile,
  ISearchFilter,
  IUserRoles,
} from 'shared/common/interfaces';
  comparePasswords,
  createActivationMessage as message,
  getUpdatedUser,
} from '~/helpers';

interface IResponse {
  code: HttpCode;
  data: any;
}

class UserService {
  public getAllUsers(): Promise<IUser[]> {
    return userRepository.getAll();
  }

  public      getUserById(id:      string):
Promise<IUser | null> {
    return userRepository.getById(id);
  }

  public async createNewUser(
    user: IUser,
    role: IUserRoles,
  ): Promise<IResponse> {
    console.log(role);
    const { password } = user;
    const     passwordHash     =     await
hashPassword(password);
    const userData: IUser = {
      ...user,
      sid: Math.floor(+Date.now() / 10000),
      password: passwordHash,
    };
    try {
      const     createdUser     =     await
userRepository.createUser(userData);
      await
userRolesRepository.createUserRole({
        ...role,
        userId: +createdUser.id,
      });
      return  {  code:  HttpCode.OK,  data:
createdUser };
    } catch (err) {
      return
this.handleRegistrationError(err);
    }
  }

  public    async    setUserActivation(data:
IUser): Promise<IUser[]> {
    const { id, email } = data;
    const            tokenHash            =
hashToken(Date.now().toString());
    await
mailService.sendActivationMail(email,
tokenHash);
    return await this.updateUser(id, {
      ...data,
      activationToken: tokenHash,
      activationTokenExpiration:        new
Date(Date.now() + 1000 * 60 * 60),
    });
  }

  public async updateUser(id: string, data:
IUser): Promise<IUser[]> {
    return    userRepository.updateById(id,
data);
  }

  public async updateUserProfile(
    userProfile: IProfile,
  ): Promise<IUser[] | null> {
    try {
      const       user       =       await
userService.getUserById(userProfile.id.toS
tring());
      if (!user) {
        return null;
      }
```

```
      const          updatedUser          =
getUpdatedUser(user, userProfile);
      await
userSchema.validate(updatedUser, { context:
{ required: true } });
      const      result      =      await
userRepository.updateById(
        userProfile.id.toString(),
        updatedUser,
      );
      return result;
    } catch {
      return null;
    }
  }

  public      deleteUser(id:      string):
Promise<number> {
    return userRepository.deleteById(id);
  }

  public   getUserByEmail(email:   string):
Promise<IUser | null> {
    return
userRepository.getByEmail(email);
  }

  public   getUserByToken(token:   string):
Promise<IUser | null> {
    return
userRepository.getByToken(token);
  }

  public async activateUser(token: string):
Promise<IActivationMessage> {
    const        user        =        await
this.getUserByToken(token);
    if (!user) {
      return
message(ActivationStatus.NOT_FOUND,
'Activation failed');
    }
    if  (user.activationTokenExpiration  <
new Date(Date.now())) {
      return message(
        ActivationStatus.EXPIRED,
        'Activation time expired',
        user.email,
      );
    }
    const data = {
      isActive: true,
      activationToken: '',
    };
    userRepository.activateUser(token,    {
...user, ...data });
    return
message(ActivationStatus.SUCCESS,
'Successful activation');
  }

  public async getUsersWithPagination(data:
ISearchFilter) {
    return                            await
userRepository.getUsersWithPagination(data
);
  }

  public    async    updateUserManage(data:
IUser, role: IUserRoles, id: string) {
    try {
      const       user       =       await
userService.getUserById(id);
      if (!user) {
        return null;
      }
      const          updatedUser          =
adminUpdatedUser(user, data);
      await
userSchema.validate(updatedUser, { context:
{ required: true } });
      const      result      =      await
userRepository.updateById(id,
updatedUser);
      return result;
```

```
    } catch {
      return null;
    }
  }

  public async setUserActive(id: string) {
    try {
      const user = await userService.getUserById(id);
      if (!user) {
        return null;
      }
      const upUser = { ...user, isActive: !user.isActive };
      await userRepository.updateById(id, upUser);
    } catch (e) {
      throw new Error(e);
    }
  }

  public async updatePassword(
    data: IUserPassword,
    id: string,
  ): Promise<AuthServiceRes<string>> {
    const { password, verifiedPassword, confirmPassword } = <IUserPassword>data;
    const isEqual = comparePasswords(verifiedPassword, confirmPassword);
    if (!isEqual) {
      return {
        code: HttpCode.BAD_REQUEST,
        data: ResponseMessages.NON_MATCH_PASSWORDS,
      };
    } else {
      const user = await userService.getUserById(id);
      if (user) {
        const isMatch = await isMatchPassword(password, user.password);
        if (!isMatch) {
          return {
            code: HttpCode.BAD_REQUEST,
            data: ResponseMessages.WRONG_PASSWORD,
          };
        }
        const verifiedPasswordHash = await hashPassword(verifiedPassword);
        const newPassword = { ...user, password: verifiedPasswordHash };
        await userRepository.updateById(id, newPassword);

        return {
          code: HttpCode.OK,
          data: ResponseMessages.CONFIRMED,
        };
      } else {
        return {
          code: HttpCode.BAD_REQUEST,
          data: ResponseMessages.NOT_AUTHORIZED,
        };
      }
    }
  }

  private handleRegistrationError(err: any): IResponse {
    if (!err.errors) {
      return { code: HttpCode.INTERNAL_SERVER_ERROR, data: err };
    }
    if (err.errors[0].type === 'unique violation') {
      return {
        code: HttpCode.BAD_REQUEST,
        data: `Email already exists in the system`,
      };
```

```
    } else {
      return {
        code:
HttpCode.INTERNAL_SERVER_ERROR,
        data: err.errors[0].message,
      };
    }
  }
}

export { UserService };
  comparePasswords,
  createJWT,
  getTokens,
  getUserPermissions,
  hashPassword,
  isMatchPassword,
  verifyJWT,
} from '~/helpers';

interface IAuthData extends ITokens {
  permissions: string[];
}

class AuthService {
  async      loginUser(data:      ILogin):
Promise<AuthServiceRes<IAuthData>> {
    const { email, password } = data;
    const        user        =        await
userService.getUserByEmail(email);
    if (!user) {
      return {
        code: HttpCode.BAD_REQUEST,
        data:
ResponseMessages.NON_EXISTING_EMAIL,
      };
    } else {
      const      isMatch      =      await
isMatchPassword(password, user.password);
      if (!isMatch) {
        return {
          code: HttpCode.BAD_REQUEST,
          data:
ResponseMessages.NON_MATCH_PASSWORDS,
        };
      }
      if(!user.isActive) {
        return {
          code: HttpCode.BAD_REQUEST,
          data:
ResponseMessages.NOT_ACTIVATED
        }
      }

      const          permissions          =
getUserPermissions(user);
      const      tokens      =      await
getTokens(user.id, permissions);
      const data = {
        ...tokens,
        permissions,
      };

      return { code: HttpCode.OK, data };
    }
  }

  async resetPassword(data: {
    email: string;
  }): Promise<AuthServiceRes<string>> {
    const { email } = data;
    const mailService = new MailService();
    const        user        =        await
userService.getUserByEmail(email);
    if (user) {
      const token = createJWT(user.id);
      await
mailService.sendResetPasswordMail(email,
token);
      return  {  code:  HttpCode.OK,  data:
ResponseMessages.CONFIRMED };
    } else {
      return {
        code: HttpCode.BAD_REQUEST,
```

```
        data:
ResponseMessages.NON_EXISTING_EMAIL,
      };
    }
  }

  async verifyPassword(data: IVerPassword):
Promise<AuthServiceRes<string>> {
    const { password, verifiedPassword,
token } = data;
    const isCompare =
comparePasswords(password,
verifiedPassword);
    if (!isCompare) {
      return {
        code: HttpCode.BAD_REQUEST,
        data:
ResponseMessages.NON_MATCH_PASSWORDS,
      };
    } else {
      const decoded = verifyJWT(token,
secret) as { userId: string };

      if (decoded) {
        const user =
userService.getUserById(decoded.userId);

        if (user) {
          const hashedPassword =
hashPassword(password);
          await
userService.updateUser(user.id, {
            ...user,
            password: hashedPassword,
          });
          return { code: HttpCode.OK, data:
ResponseMessages.CONFIRMED };
        } else {
          return {
            code: HttpCode.BAD_REQUEST,
            data:
ResponseMessages.NON_EXISTING_EMAIL,
          };
        }
      } else {
        return {
          code: HttpCode.FORBIDDEN,
          data:
ResponseMessages.TOKEN_EXPIRED,
        };
      }
    }
  }
}

export { AuthService };

const appAsyncStorage =
AsyncLocalStorage<AppAsyncStorage>();

const logger = new Logger({
  logLevel: LogLevel.DEBUG,
  asyncStorage: appAsyncStorage,
});

const userService = new UserService();
const mailService = new MailService();
const authService = new AuthService();
const fileUploadService =
FileUploadService();
const roleService = new RoleService();
const permissionService =
PermissionService();
const userRolesService =
UserRolesService();
const productService =
ProductService();
const merchantService =
MerchantService();
const transactionService =
TransactionService();

export {
  appAsyncStorage,
  logger,
  userService,
```

```
  mailService,
  authService,
  fileUploadService,
  roleService,
  permissionService,
  userRolesService,
  productService,
  merchantService,
  transactionService
};

const        transactionValSchema        =
Yup.object().shape({
  merchant_sid: Yup.number().required(),
  user_id: Yup.number().required(),
  amount: Yup.number().required(),
  order_details: Yup.string(),
  order_id: Yup.string(),
});

export { transactionValSchema };

class TransactionService {
  public              getAllTransactions():
Promise<ITransaction[]> {
    return transactionRepository.getAll();
  }

  public getTransactionsByMerchantId(
    merchant_id: number,
  ): Promise<ITransaction[]> {
    return
transactionRepository.getByMerchantId(merc
hant_id);
  }

  public
getTransactionsByOrderId(order_id:
number): Promise<ITransaction[]> {
    return
transactionRepository.getByOrderId(order_i
d);
  }

  public async createTransaction(
    transaction: ITransaction,
  ): Promise<ITransaction[] | null> {
    try {
      const               newTransactions:
ITransactionExtended[] = [];

      await
transactionValSchema.validate(transaction)
;

      const merchantId: number | null =
        await
merchantService.getMerchantIdBySid(transac
tion.merchant_sid);
      if (merchantId) {
        const today = new Date();

        for    (let    i    =    0;    i    <
transaction.payments; i++) {
          const             newTransaction:
ITransactionExtended = {
            ...transaction,
            amount:
Math.floor(transaction.amount            /
transaction.payments),
            authorization: 'auth-string',
            merchant_id: merchantId,
            expiration_date:            new
Date(today.setMonth(today.getMonth() + i +
1)),
            processed: false,
            createdAt: today,
            updatedAt: today,
          };

          const                    created:
ITransactionExtended | null =
            await
transactionRepository.createTransaction(ne
wTransaction);
```

```
    if (created) {
      newTransactions.push(created);
    }
  }

  return newTransactions;
} else {
  return null;
}
```

```
    } catch (e) {
      console.log(e);
      return null;
    }
  }
}

export { TransactionService };
```

ДОДАТОК 6

Сервіс для створення та керування відкладеними платежами за покупки в інтернеті

Текст програмного коду сервісної веб-частини

ІАЛЦ.467200.009 Д6

Аркушів 37

Київ 2022 р

```
export { HttpError } from 'shared/exceptions';
import { AppRoute } from 'common/enums';
import { LinkProps } from 'react-router-dom';

type Props = LinkProps & {
  to: AppRoute;
};

export type { Props };
import Link from './link/link';
import RBAC from './RBAC/rbac';
import RBACRoute from './RBACRoute/RBACRoute';

export { Link, RBAC, RBACRoute };
export { default as FormField } from './FormField';
export * from './Uploader'
export interface IListItem {
  label: string,
  link?: string,
  onClick?: () => void
}

export interface IHeaderProps {
  user?: {
    firstName: string,
    lastName: string,
    avatar: string,
  }
  callback?: () => void,
  isCollapsed?: boolean
}
export * from './root-state.type';
export * from './app-thunk.type';
import { ThunkAction, Action } from
'@reduxjs/toolkit';
import { RootState } from 'common/types/app/root-
state.type';

type AppThunk<ReturnType = void> = ThunkAction<
  ReturnType,
  RootState,
  unknown,
  Action<string>
>;

export type { AppThunk };
import { store } from 'store/store';

type RootState = ReturnType<typeof store.getState>;

export type { RootState };
export type LoginResponse = {
  accessToken: string;
  refreshToken: string;
  permissions: string[];
};

export * from './login-response.types';
import { Action, PayloadAction } from
'@reduxjs/toolkit';
import {IProfile, IUserPassword} from 'shared';

export interface SagaAction extends Action,
PayloadAction {
  type: string;
}

export interface SagaUserAction
  extends Action,
    PayloadAction<{ user: IProfile } & { token:
string }> {
  type: string;
}

export interface SagaActionWithTokenAndPayload<T>
extends Action, PayloadAction<T & { token: string
}> {
  type: string
}

export interface SagaActionWithToken extends
Action, PayloadAction<{ token: string }> {
  type: string
}

export interface SagaUpdatePasswordAction
  extends Action,
    PayloadAction<{ passwordData: IUserPassword } &
{ token: string }> {
  type: string;
}
export * from './saga-action.type';
export * from './saga-action';
export * from './auth';
export * from './profile-transactions.types';
type IDataTransactions = {
  transaction_id: number;
  title: string;
  created: string;
  currency: string;
  amount: number;
  avatar?: string;
};

type IAmountOfMoney = {
  amount: number;
  currency: string;
};

type ITransactions = {
  title: string;
  time: number;
  data: [IDataTransactions];
};

type ICurencies = {
  symbol: string;
  name: string;
  rate: number;
};

export type { IDataTransactions, IAmountOfMoney,
ITransactions, ICurencies };
type Message = {
  message: string;
}

export type ApiResponse<T> = Message | T
```

```
type HttpOptions = {
  method: HttpMethod;
  contentType: ContentType;
  payload: Record<string, unknown> |
string;
};export { roleValidation, roleCheck };

export type { HttpOptions };
export * from './http-options.type';
export * from './app';
export * from './http';
export * from './formik';
export * from './saga';
type FormError = {
  email?: string;
};

export type { FormError };
export * from './login-formik.types';
enum AppRoute {
  MAIN = '/main',
  SIGN_IN = '/sign-in',
  SIGN_UP = '/sign-up',
  REFRESH = '/reset',
  VERIFY_REFRESH = '/verify-
refresh/:token',
  EMAIL_ACTIVATION = '/email-
activation/:token',
  USER_MANAGE = '/user-manage',
  PLATFORM_EDIT = '/platform-edit',
  USER_PROFILE = '/profile',
  UPDATE_USER = '/update-user/:id',
  CREATE_USER = '/create-user',
  ACTIVATION = 'users/activation',
  TENANT_DETERMINE = 'tenants/platform',
  EDIT_PROFILE = '/profile-edit',
  TRANSCATION_PROFILE =
'/profile/transactions',
  NOTIFICATION_PROFILE =
'/profile/notification',
  SECURITY_PROFILE = '/profile-security',
  ERROR = '/error',
  BOOKINGS_MANAGE =
'/bookings/:action/:id?',
  TIMECARDS_MANAGE = '/timecards/:action',
  BOOKING_REVIEW = '/booking/review/:id',
  GOODS = '/goods',
  BOOKINGS_CREATE = '/bookings/:action',
  TIMECARD_REVIEW = '/timecard/review/:id',
  TIMECARD = '/timecard',
  MAIL_SENT = '/mail-sent',
  PAYMENTS = '/payments',
  CREATE_PAYMENT = '/create-payment',
  USER_DASHBOARD = '/user-dashboard',
  MERCHANT_DASHBOARD = '/merchant-
dashboard',
  CHARGES = '/charges',
  MERCHANT_HUB = '/merchant-hub',
  PURCHASE = '/purchase',
  TRANSACTIONS = '/transactions',
}

export { AppRoute };
const { REACT_APP_API_ORIGIN_URL } =
process.env;

const ENV = {
  API_PATH: REACT_APP_API_ORIGIN_URL,
} as const;

export { ENV };
export * from './reducer-name.enum';
export * from './app-route.enum';
export * from './env.enum';
enum ReducerName {
  REG = 'registration',
  USER = 'user',
  ACTIVATION = 'activation',
  FILE = 'file',
  USER_MANAGE = 'user_manage',
  USER_DATA = 'userData',
  USER_DASHBOARD = 'userDashboard',
  MERCHANT = 'merchant',
  TENANT = 'tenant',
  BOOKING = 'booking',
```

```
  TIMECARD = 'timecard',
  BOOKING_MANAGE = 'booking_manage',
  TIMECARD_MANAGE = 'timecard_manage',
  PAYMENTS = 'payments',
  ORDERS = 'orders',
  USER_CHARGES = 'userCharges',
  MERCHANT_HUB = 'merchant-hub',
  TRANSACTIONS = 'transactions',
}

export { ReducerName };
export { ContentType } from 'shared/common/enums';
export * from "./tenant-sagas-types.enum"
enum TenantSagasTypes {
  UPDATE_TENANT = 'UPDATE_TENANT',
  GET_ALL_INDUSTRIES = 'GET_ALL_INDUSTRIES',
}

export { TenantSagasTypes };
export * from './payments-types';
enum PaymentsSagaTypes {
  PAYMENTS_GET_WITH_PAGINATION = 'PAYMENT_GET_WITH_PAGINATION',
  PAYMENT_GET_USER_TO_UPDATE = 'PAYMENT_GET_USER_TO_UPDATE',
  PAYMENT_UPDATE_USER = 'PAYMENT_UPDATE_USER',
  PAYMENT_CREATE_USER = 'PAYMENT_CREATE_USER',
  SET_PAYMENT_ACTIVE = 'SET_PAYMENT_ACTIVE',
  CREATE_PAYMENT = 'CREATE_PAYMENT',
  GET_SINGLE = 'GET_SINGLE'
}

export {PaymentsSagaTypes};
enum UserDashboardSagaTypes {
  GET_USER_DATA = 'GET_USER_DATA',
  GET_ORDERS = 'GET_ORDERS',
}
export { UserDashboardSagaTypes };
export * from './user-dashboard-sagas-types.enum';

enum FileSagasTypes {
  LOAD_TO_CLOUD = 'LOAD_TO_CLOUD'
}

export { FileSagasTypes };
export * from "./file-sagas-types.enum"
export * from "./auth-sagas-types.enum"
enum AuthSagasTypes {
  LOGIN_USER = 'LOGIN_USER',
  REFRESH_PASSWORD = 'REFRESH_PASSWORD',
  VERIFY_PASSWORD_CHANGE = 'VERIFY_PASSWORD_CHANGE',
  REGISTER_USER = 'REGISTER_USER',
  LOGOUT_USER = 'LOGOUT_USER',
}

export { AuthSagasTypes };
enum BookingSagaTypes {
  BOOKING_GET_USER_WITH_PAGINATION = 'BOOKING_GET_USER_WITH_PAGINATION',
  BOOKING_GET_BOOKING_REVIEW = 'BOOKING_GET_BOOKING_REVIEW',
  BOOKING_UPDATE_BOOKING_STATUS = 'BOOKING_UPDATE_BOOKING_STATUS',
  BOOKING_ACCEPT_BOOKING = 'BOOKING_ACCEPT_BOOKING',
  BOOKING_CANCEL_BOOKING = 'BOOKING_CANCEL_BOOKING'
}

export { BookingSagaTypes };
export * from './booking.types';

export * from './user-charges-sagas-types.enum';

enum UserChargesSagaTypes {
    GET_USER_CHARGES = 'GET_USER_CHARGES',
  }
  export { UserChargesSagaTypes };
```

```
enum TimecardSagaTypes {
  TIMECARD_GET_USER_WITH_PAGINATION       =
'TIMECARD_GET_USER_WITH_PAGINATION',
  TIMECARD_GET_TIMECARD_REVIEW            =
'TIMECARD_GET_TIMECARD_REVIEW',
  TIMECARD_UPDATE_TIMECARD_STATUS         =
'TIMECARD_UPDATE_TIMECARD_STATUS',
}

export { TimecardSagaTypes };
export * from "./timecard-saga-types.enum"
export enum MerchantSagaTypes {
  LOAD_ALL_MERCHANTS                      =
'LOAD_ALL_MERCHANTS',
  LOAD_NEWEST_MERCHANTS                   =
'LOAD_NEWEST_MERCHANTS',
  LOAD_CHEAPEST_MERCHANTS                 =
'LOAD_CHEAPEST_MERCHANTS',
  LOAD_DISCOUNTED_MERCHANTS               =
'LOAD_DISCOUNTED_MERCHANTS',
}
export * from './merchant-saga-types-enum';
enum TransactionsSagaTypes {
  LOAD_TRANSACTIONS = 'LOAD_TRANSACTIONS',
}

export { TransactionsSagaTypes };
export  *  from  './transaction-saga-types-
enum';
enum UserManageSagaTypes {
  MANAGE_GET_USERS_WITH_PAGINATION        =
'MANAGE_GET_USERS_WITH_PAGINATION',
  MANAGE_GET_USER_TO_UPDATE               =
'MANAGE_GET_USER_TO_UPDATE',
  MANAGE_UPDATE_USER                      =
'MANAGE_UPDATE_USER',
  MANAGE_CREATE_USER                      =
'MANAGE_CREATE_USER',
  SET_USER_ACTIVE = 'SET_USER_ACTIVE'

}

export { UserManageSagaTypes };
export * from './user-manage-types';
enum BookingManageSagasTypes {
  GET_BOOKING = 'GET_BOOKING',
  CREATE_BOOKING = 'CREATE_BOOKING',
  UPDATE_BOOKING = 'UPDATE_BOOKING',
  GET_ALL_TEMPLATES = 'GET_ALL_TEMPLATES',
  GET_ALL_TIMECARD_TEMPLATES              =
'GET_ALL_TIMECARD_TEMPLATES',
  GET_ALL_TENANTS = 'GET_ALL_TENANTS'
}

export { BookingManageSagasTypes };
export   *   from   "./booking-manage-sagas-
types.enum"
enum UserDataSagaTypes {
  GET_USER = 'GET_USER',
  UPDATE_USER = 'UPDATE_USER',
  UPDATE_USER_PASSWORD                    =
'UPDATE_USER_PASSWORD'
}

export { UserDataSagaTypes };
export    *    from    './user-data-sagas-
types.enum';
export * from './auth';
export * from './file';
export * from './user-data';
export * from './user-dashboard';
export * from './user-manage';
export * from './tenant';
export * from './booking';
export * from './booking-manage';
export * from './timecard';
export * from './timecard-manage';
export * from './timecard-manage';
export * from './payments';
export * from './merchants';
export * from './orders';
export * from './merchant-hub';
enum TimecardManageSagasTypes {
  GET_TIMECARD = 'GET_TIMECARD',
  CREATE_TIMECARD = 'CREATE_TIMECARD',
  UPDATE_TIMECARD = 'UPDATE_TIMECARD',
```

```
}

export { TimecardManageSagasTypes };
export * from "./timecard-manage-sagas-types.enum"
export enum OrdersSagaTypes {
  LOAD_ALL_ORDERS = 'LOAD_ALL_ORDERS',
  SET_ALL_ORDERS = 'SET_ALL_ORDERS',
  CREATE_ORDER = 'CREATE_ORDER',
}
export * from './orders-saga-types-enum'
export enum MerchantHubSagaTypes {
  LOAD_MERCHANTS = 'LOAD_MERCHANTS',
  SET_MERCHANTS = 'SET_MERCHANTS',
}
export * from './merchant-hub';
export { HttpHeader, HttpMethod } from 'shared/common/enums';
export enum LocalstorageKeys {
  AUTH = 'AUTH'
}
export * from './localstorage.enum';
export * from './app';
export * from './http';
export * from './file';
export * from './saga';
export * from './localstorage';
export * from './permission';
export { Permission } from 'shared/common/enums';



export const useTypedSelector: TypedUseSelectorHook<RootState> = useSelector;

export const useDetectOutsideClick = (
  closestSelector: string,
  initialState: boolean,
) => {
  const [isVisible, setIsVisible] = useState(initialState);

  function hide({target}: Event) {
    if (!(target instanceof Element)) return;
    if (!target?.closest(closestSelector)) {
      setIsVisible(false);
    }
  }

  useEffect(() => {
    if (isVisible) {
      window.addEventListener('mousedown', hide);
    }
    return () => {

window.removeEventListener('mousedown', hide);
    }

  }, [isVisible, closestSelector]);

  return [isVisible, setIsVisible] as const;
}
/// <reference types="react-scripts" />

export const SignUpSchema = Yup.object().shape({
  firstName: Yup.string()
    .max(70, "The name must not exceed 70 characters in length")
    .matches(/^[a-zA-Z0-9-/']*([[a-zA-Z0-9]*]?)$/, 'Enter correct data')
    .required('First Name is required'),
  lastName: Yup.string()
    .max(70, "The name must not exceed 70 characters in length")
    .matches(/^[a-zA-Z0-9-/']*([[a-zA-Z0-9]*]?)$/, 'Enter correct data')
    .required('Last Name is required'),
  email: Yup.string().email('Email is invalid')
```

```
    .matches(/^[a-zA-Z0-9.]+@[a-zA-Z0-9]+(?:\.[a-zA-Z0-9-]+)*$/, 'Enter correct data')
    .required('Email is required'),
  phoneNumber: Yup.string()
    .matches(/^\+?\d{10,12}$/, 'Enter correct data')
    .required('Phone Number is required'),
  password: Yup.string()
    .matches(/^[a-zA-Z0-9.\-!@#$%^&*()-_+=?><,~]*$/, 'letters must be latin')
    .matches(/^[A-Z]/, 'First letter must be capital')
    .matches(/[0-9]/, 'Must match some number')
    .matches(/[a-zA-Z0-9]*[.\-!@#$%^&*()-_+=?><,~][a-zA-Z0-9]*$/, 'Must match some special case character')
    .min(6, 'Min length 6 symbols')
    .required('Must be completed'),
  confirmPassword: Yup.string()
    .oneOf([Yup.ref('password'), null], 'Passwords must match')
    .required('Confirm Password is required'),
});

export const chartOptions = [
  { title: 'last week', value: timePeriods.LAST_WEEK },
  { title: 'last month', value: timePeriods.LAST_MONTH },
  { title: 'all times', value: timePeriods.ALL_TIMES },
];

export function getBusinessSnapshotElements(orders: Order[]) {
  return [
    { title: 'Loan volume', getValue: () => '$' + loanVolume(orders) },
    {
      title: 'Average order value',
      getValue: () => '$' + avgOrderValue(orders),
    },
    { title: 'Orders made', getValue: () => orders.length },
    {
      title: 'Covered loans',
      getValue: () => percentageOfFullfilled(orders) + '%',
    },
  ];
}
enum timePeriods {
  LAST_MONTH = 30,
  LAST_WEEK = 7,
  ALL_TIMES = 5000,
}

export { timePeriods };

function nDaysAgo(n: number): Date {
  const today = new Date();
  return new Date(today.getFullYear(), today.getMonth(), today.getDate() - n);
}

function daysBetweenTwoDates(date1: Date, date2: Date): number {
  return (date1.getTime() - date2.getTime()) / (1000 * 3600 * 24);
}

function lastNDays(n: number): Date[] {
  const today = new Date();
  const days: Date[] = [];
  for (let i = 0; i < n; i++) {
    days.push(
      new Date(today.getFullYear(), today.getMonth(), today.getDate() - i),
    );
  }
  return days.reverse();
```

```
}

function firstOrderDate(orders: Order[]): Date {
  return orders.reduce((firstOrderDate, order) => {
    if (new Date(order.created_at) < firstOrderDate)
      return new Date(order.created_at);
    return firstOrderDate;
  }, new Date());
}

function loanedByDate(date: Date, orders: Order[]): number {
  return orders.reduce((count: number, order: Order) => {
    if (
      date.getDate() === new Date(order.created_at).getDate() &&
      date.getMonth() === new Date(order.created_at).getMonth() &&
      date.getFullYear() === new Date(order.created_at).getFullYear()
    ) {
      count += order.price;
    }
    return count;
  }, 0);
}

export {
  nDaysAgo,
  daysBetweenTwoDates,
  lastNDays,
  firstOrderDate,
  loanedByDate,
};

const dataTransactions: IDataTransactions[] = [
  {
    transaction_id: 1,
    title: 'Bank transfer to SoundCloud',
    created: '2021-10-17T13:41:43.921Z',
    currency: 'EUR',
    amount: 1200,
    avatar:
      'https://play-lh.googleusercontent.com/lvYCdrPNFU0Ar_lXln3JShoE-NaYF_V-DNlp4eLRZhUVkj00wAseSIm-60OoCKznpw',
  },
  {
    transaction_id: 2,
    title: 'From RUKI',
    created: '2021-09-21T13:41:43.921Z',
    currency: 'EUR',
    amount: 5,
    avatar:

'https://pbs.twimg.com/profile_images/736207719519162368/7UxTM9o-.jpg',
  },
  {
    transaction_id: 3,
    title: 'Uber ride',
    created: '2021-10-21T13:41:43.921Z',
    currency: 'EUR',
    amount: 20,
    avatar:
      'https://seeklogo.com/images/U/uber-logo-2BB8EC4342-seeklogo.com.png',
  },
  {
    transaction_id: 4,
    title: 'Starucks',
    created: '2021-09-21T13:41:43.921Z',
    currency: 'EUR',
    amount: 50,
    avatar:

'https://upload.wikimedia.org/wikipedia/ru/thumb/d/d3/Starbucks_Corporation_Logo_2011.svg/1200px-Starbucks_Corporation_Logo_2011.svg.png',
```

```
  },
  {
    transaction_id: 5,
    title: 'Exchange from USD',
    created: '2020-08-21T13:41:43.921Z',
    currency: 'EUR',
    amount: 10,
    avatar:
'https://images.pngnice.com/download/2007/
USD-PNG-Transparent.png',
  },
  {
    transaction_id: 6,
    title: 'Transport for London',
    created: '2020-08-21T13:41:43.921Z',
    currency: 'EUR',
    amount: 10,
    avatar:
'https://cdn.worldvectorlogo.com/logos/tra
nsport-for-london-1.svg',
  },
];

const currencies: ICurencies[] = [
  { symbol: '€', name: 'EUR', rate: 0.033
},
  { symbol: '₴', name: 'UAH', rate: 1 },
  { symbol: '₽', name: 'RUB', rate: 2.67 },
  { symbol: '$', name: 'USD', rate: 0.038
},
];

export { dataTransactions, currencies };

function    parseDateString(value:    any,
originalValue: any) {
  const parsedDate = isDate(originalValue)
    ? originalValue
    :  parse(originalValue,  'yyyy-MM-dd',
new Date());

  return parsedDate;
}

const adult = new Date();
adult.setFullYear(adult.getFullYear()    -
18);

const  uppercaseName  =  (value:  string,
originalValue: string) => {
  return
originalValue.charAt(0).toUpperCase()    +
originalValue.slice(1);
};

export    const    userProfileSchema    =
Yup.object().shape({
  firstName: Yup.string()
    .trim()
    .max(70, 'The name must not exceed 70
characters in length')
    .matches(/^[a-zA-Z0-9-/']*([[a-zA-Z0-
9]*]?)$/, 'Enter correct data')
    .required('First Name is required'),
  lastName: Yup.string()
    .trim()
    .max(70, 'The name must not exceed 70
characters in length')
    .matches(/^[a-zA-Z0-9-/']*([[a-zA-Z0-
9]*]?)$/, 'Enter correct data')
    .required('Last Name is required'),
  email: Yup.string()
    .email('Email is invalid')
    .trim()
    .matches(
      /^[a-zA-Z0-9.]+@[a-zA-Z0-9]+(?:\.[a-
zA-Z0-9-]+)*$/,
      'Enter correct data',
    )
    .required('Email is required'),
  phoneNumber: Yup.string()
    .matches(/^\+?\d{10,12}$/,       'Enter
correct data')
    .required('Phone Number is required'),
  birthDate:
Yup.date().transform(parseDateString).max(
```

```
adult, "Your date is not correct. You must be over 18 years old"),
  avatar: Yup.string(),
});
export * from './profile-page';
export * from './notification-page';
export * from './security-page';
export * from './edit-page';
export * from './transactions-page';

function useQuery() {
  const { search } = useLocation();
  return useMemo(() => new URLSearchParams(search), [search]);
}

export { useQuery };
export * from './useQuery';

const apiService = new ApiService();

export async function createTransaction(transaction: ITransaction) {
  const data: ITransaction = {
    ...transaction,
  };
  const response = await apiService.httpRequest(
    '/transactions/create',
    'post',
    {
      body: { ...data },
    },
  );
  console.log(response);
}

const options: IOption[] = [
  {
    period: 3,
    apr: 0,
  },
  {
    period: 6,
    apr: 0.05,
  },
  {
    period: 12,
    apr: 0.1,
  },
];

export default options;
interface IOption {
  period: number; //months
  apr: number; //percent
}

export type { IOption };
export * from './option.interface';
export * from './product.interface';
export * from './transaction.interface';
interface ITransaction {
  merchant_sid: number;
  order_id: string;
  user_id: number;
  amount: number;
  order_details: string;
  payments: number;
}

export type { ITransaction };
interface IProduct {
  price: number;
  name: string;
  merchant_id: number;
}

export type { IProduct };
export * from './auth/registration';
export * from './auth/login';
export * from './auth/reset';
export * from './auth/verify-refresh';
export * from './auth/mail-sent-message';
export * from './main/main';
```

```
export * from './email-activation';
export * from './main/user-manage';
export * from './user-profile';
export * from './charges';
export * from './main/user-dashboard';
export * from './goods';
export * from './error';
export * from './merchant-dashboard';
export * from './merchant-hub';
export * from './purchase';
export * from './transactions';
export interface IOrderInput {
  name: string;
  number_of_payments: number;
  payments_covered: number;
  price: string;
  expires_at: string;
}
const addSpaces = (number: number) => {
  return number
    .toString()
    .split('')
    .reverse()
    .join('')
    .replace(/[^\dA-Z]/g, '')
    .replace(/(.{3})/g, '$1 ')
    .split('')
    .reverse()
    .join('')
    .trim();
};
export { addSpaces };
export * from './update';
export * from './create';
export * from './user-manage';

function searchMerchants(merchants:
IMerchant[], query: string): IMerchant[] {
  if (query === '') return merchants;
  return merchants.filter(
    (merch) =>
      merch.name.substring(0,
query.length).toLowerCase() ===
      query.toLowerCase(),
  );
}

export { searchMerchants };
export * from './parse-data.helpers';
export const parseDateToTime = (time:
string) => {
  return new Date(time).toLocaleString('en-
Us', {
    minute: 'numeric',
    hour: 'numeric',
    hour12: false,
  });
};

export const parseDataToYMD = (value?:
string) => {
  return new
Date(value!).toLocaleString('en-us', {
    month: 'long',
    year: 'numeric',
    day: 'numeric',
  });
};

export const dateToString = (date: string)
=> {
  const currentDate = new Date(date);

  const days = Math.floor(
    (new Date().getTime() -
currentDate.getTime()) / (1000 * 60 * 60 *
24),
  );

  switch (true) {
    case days === 0:
      return 'Today';
    case days === 1:
      return 'Yesterday';
    case new Date().getFullYear() ===
currentDate.getFullYear(): {
```

```
      return
currentDate.toLocaleString('en-us', {
        day: 'numeric',
        month: 'long',
      });
    }
    default:
      return date;
  }
};
export    {    checkIsOneOf    }    from
'shared/helpers';
export    *    from    './space-through-
three.helper';
export * from './boolean';
export * from './date';
export * from './string';

class Http {
  load<T = unknown>(url: string, options:
Partial<HttpOptions> = {}): Promise<T> {
    const  {  method  =  HttpMethod.GET,
payload = {}, contentType } = options;

    const            headers            =
this._getHeaders(contentType);
    const            isJSON             =
checkIsOneOf(contentType,
ContentType.JSON);

    return fetch(url, {
      method,
      headers,
      body:            isJSON            ?
JSON.stringify(payload)  :  (payload  as
string),
    })
      .then(this._checkStatus)
      .then((res)                        =>
this._parseJSON<T>(res))
      .catch(this._throwError);
  }

  _getHeaders(contentType?:  ContentType):
Headers {
    const headers = new Headers();

    if (contentType) {

headers.append(HttpHeader.CONTENT_TYPE,
contentType);
    }

    return headers;
  }

  _checkStatus(response:        Response):
Response {
    if (!response.ok) {
      throw new HttpError({
        status: response.status,
      });
    }

    return response;
  }

  _parseJSON<T>(response:       Response):
Promise<T> {
    return response.json();
  }

  _throwError(err: Error): never {
    throw err;
  }
}

export { Http };
class LocalstorageService {
  setItem(key: string, data: any) {
    const jsonData = JSON.stringify(data);
    localStorage.setItem(key, jsonData);
  }

  getItem(key:  string):  {  accessToken:
string; refreshToken: string } {
```

```
    const data = localStorage.getItem(key);
    return data ? JSON.parse(data) : null;
  }

  removeItem(key: string) {
    localStorage.removeItem(key);
  }
}

export default LocalstorageService;
```

```
const { REACT_APP_ORDERS_HOST, REACT_APP_ORDERS_API_ORIGIN_URL } = process.env;

class ApiOrdersService {
  _url = `${REACT_APP_ORDERS_HOST}${REACT_APP_ORDERS_API_ORIGIN_URL}`;
  instance = axios.create({
    baseURL: this._url,
  });

  httpRequest = async (
    url: string,
    method: Method = 'GET',
    options: IHttpRequestOptions = {
      body: null,
      params: null,
      headers: {},
      token: '',
    },
  ): Promise<any> => {
    const { body } = options;
    try {
      const res = await this.instance.request({
        url,
        data: body,
        headers: {
          Authorization: `Bearer ${tokens?.accessToken || ''}`,
          ...headers,
        },
        method,
        params,
      });
      return res.data;
    } catch (e) {
      console.log(e);
    }
  };
}

export default ApiOrdersService;
```

```
const { REACT_APP_BACKEND_HOST, REACT_APP_API_ORIGIN_URL } = process.env;

const BACKEND_HOST: string = REACT_APP_BACKEND_HOST ?? 'http://localhost:3001/';
const API_ORIGIN_URL: string = REACT_APP_API_ORIGIN_URL ?? 'api/v1/';

export interface IHttpRequestOptions {
  token?: string;
  params?: any;
  body?: any;
  headers?: any;
}

interface ITokens {
  accessToken: string;
  refreshToken: string;
}

class ApiService {
  _url = `${BACKEND_HOST}${API_ORIGIN_URL}`;
  instance = axios.create({
    baseURL: this._url,
  });

  localstorageService = new LocalstorageService();
```

```
  httpRequest = async (
    url: string,
    method: Method = 'GET',
    options: IHttpRequestOptions = {
      body: null,
      params: null,
      headers: {},
      token: '',
    },
  ): Promise<any> => {
    const { body, params, headers } = options;
    const tokens = await this.refreshTokens();
    try {
      const res = await this.instance.request({
        url,
        data: body,
        headers: {
          Authorization: `Bearer ${tokens?.accessToken || ''}`,
          ...headers,
        },
        method,
        params,
      });
      return res.data;
    } catch (e) {
      const { data } = e.response;
      console.dir(e);
      if (!Object.keys(data).length) {
        throw e;
      }
      if (data.message) {
        throw new Error(data.message);
      }
      if (data.error) {
        throw new Error(data.error);
      }
      throw new Error(data);
    }
  };

  async refreshTokens(): Promise<ITokens | undefined> {
    const state = store.getState();
    const userLoggedIn = state.user.authState;

    if (!userLoggedIn) {
      return;
    }
    const tokens = this.localstorageService.getItem(LocalstorageKeys.AUTH);

    try {
      // check if access token still valid
      await this.instance.request({
        url: `${ApiPath.AUTH}${AuthApiPath.LOGIN}`,
        headers: {
          Authorization: `Bearer ${tokens?.accessToken || ''}`,
        },
        method: 'GET',
      });
      return tokens;
    } catch (e) {
      if (e.message.includes(HttpCode.UNAUTHORIZED)) {
        try {
          // refresh tokens
          const tokensData = await this.instance.request({
            url: `${ApiPath.AUTH}${AuthApiPath.REFRESH_TOKEN}`,
            headers: {
              Authorization: `Bearer ${tokens?.refreshToken || ''}`,
            },
            method: 'POST',
```

```
          });

          const newTokens: ITokens = {
            refreshToken:
tokensData.data.refreshToken,
            accessToken:
tokensData.data.accessToken,
          };


this.localstorageService.setItem(Localstor
ageKeys.AUTH, newTokens);
          return newTokens;
        } catch (e) {

this.localstorageService.removeItem(Locals
torageKeys.AUTH);
          return;
        }
      }
    }
  }
}

export default ApiService;

export { ApiOrdersService, ApiService };
export * from './http/http.service';

interface IUserData {
  orders: Order[];
  loan: number;
  balance: number;
  processed: number;
  firstName: string;
  lastName: string;
  sid: number | null;
  requestStatus: boolean;
  requestPasswordStatus: boolean;
  error: string | null;
  loading: boolean;
  requestFulfilled: boolean;
}
```

```
const initialState: IUserData = {
  orders: [],
  loan: 0,
  processed: 0,
  balance: 0,
  firstName: '',
  lastName: '',
  sid: null,
  requestStatus: false,
  requestPasswordStatus: false,
  error: null,
  loading: false,
  requestFulfilled: false,
};

const { reducer, actions } = createSlice({
  name: ReducerName.USER_DASHBOARD,
  initialState,
  reducers: {
    requestStart: (state) => {
      state.error = null;
      state.loading = true;
    },

    setUserData:      (state,      action:
PayloadAction<IUser>) => {
      const  {  firstName,  lastName,  sid,
balance } = action.payload as IUser;
      state.firstName = firstName;
      state.lastName = lastName;
      state.sid = sid;
      state.balance = balance;
    },

    setOrders:       (state,       action:
PayloadAction<Order[]>) => {
      const     orders:     Order[]     =
action.payload;
      state.orders = orders;
    },
```

```
    setLoan:        (state,        action:
PayloadAction<number>) => {
      state.loan = action.payload;
    },

    setProcessed:      (state,      action:
PayloadAction<number>) => {
      state.processed = action.payload;
    },

    requestFail: (state) => {
      state.requestStatus = false;
      state.loading = false;
    },

    requestSuccess: (state) => {
      state.requestStatus = true;
      state.loading = false;
      state.requestFulfilled = true;
    },

    clearError: (state) => {
      state.error = null;
      state.requestPasswordStatus = false;
    },
  },
});

export const getUserDataAction = (payload:
string) => ({
  type:
UserDashboardSagaTypes.GET_USER_DATA,
  payload: payload,
});

export  const  getOrdersAction  =  (payload:
number) => ({
  type: UserDashboardSagaTypes.GET_ORDERS,
  payload: payload,
});

const userDashboardActionCreator = {
  ...actions,
};

export    {    userDashboardActionCreator,
reducer };

type FileState = {
  loading: boolean;
  error: string | null;
  cloudURL: string;
};

const initialState: FileState = {
  loading: false,
  error: null,
  cloudURL: '',
};

const { reducer, actions } = createSlice({
  name: ReducerName.FILE,
  initialState,
  reducers: {
    requestStart: (state) => {
      state.error = null;
      state.loading = true;
    },
    requestFailed: (state, action) => {
      state.error = action.payload;
      state.loading = false;
    },
    clearError: (state) => {
      state.error = null;
    },
    clearURL: (state) => {
      state.cloudURL = '';
    },
    cloudUploadSucceed: (state, action) =>
{
      state.loading = false;
      state.cloudURL = action.payload;
    },
  },
});
```

```
export const loadFileToCloudAction = (data:
any) => ({
  type: FileSagasTypes.LOAD_TO_CLOUD,
  payload: data,
});

const FileActionCreator = {
  ...actions,
};

export { FileActionCreator, reducer };

type UserState = {
  verifySucceed: boolean;
  authState: boolean;
  loading: boolean;
  registerState: boolean;
  resetState: string;
  error: string | null;
  permissions: string[];
  accessToken: string;
  refreshToken: string;
};

const initialState: UserState = {
  verifySucceed: false,
  authState: false,
  resetState: '',
  registerState: false,
  loading: false,
  error: null,
  permissions: [],
  accessToken: '',
  refreshToken: '',
};

const { reducer, actions } = createSlice({
  name: ReducerName.USER,
  initialState,
  reducers: {
    requestStart: (state) => {
      state.error = null;
      state.loading = true;
      state.resetState = '';
      state.verifySucceed = false;
    },
    loginSucceed: (state, action) => {
      const { accessToken, refreshToken,
permissions } =
        action.payload as LoginResponse;

      state.accessToken = accessToken;
      state.refreshToken = refreshToken;
      state.permissions = permissions;
      state.loading = false;
      state.authState = true;
    },
    resetSucceed: (state, action) => {
      state.resetState = action.payload;
      state.loading = false;
    },
    requestFailed: (state, action) => {
      state.error = action.payload;
      state.loading = false;
    },
    verifySucceed: (state) => {
      state.verifySucceed = true;
      state.loading = false;
    },
    clearError: (state) => {
      state.error = null;
    },
    clearResetState: (state) => {
      state.resetState = '';
    },
    clearRegisterState: (state) => {
      state.registerState = false;
    },
    signUpSucceed: (state) => {
      state.registerState = true;
      state.loading = false;
    },
    logoutSucceed: (state) => {
      state.authState = false;
      state.accessToken = '';
      state.refreshToken = '';
```

```
      state.loading = false;
    },
  },
});

export const signUpUserAction = (data: any)
=> ({
  type: AuthSagasTypes.REGISTER_USER,
  payload: data,
});
export  const  loginUserAction  =  (data:
ILogin) => ({
  type: AuthSagasTypes.LOGIN_USER,
  payload: data,
});
export const resetPasswordAction = (email:
{ email: string }) => ({
  type: AuthSagasTypes.REFRESH_PASSWORD,
  payload: email,
});
export const verifyPasswordAction = (data:
IVerPassword) => ({
  type:
AuthSagasTypes.VERIFY_PASSWORD_CHANGE,
  payload: data,
});
export const logoutUserAction = () => ({
  type: AuthSagasTypes.LOGOUT_USER,
});

const UserActionCreator = {
  ...actions,
};

export { UserActionCreator, reducer };

interface ChargesRow {
    id: string
    date: string
    loanId: string
    name: string
    status: boolean
    total: number
}

interface IChargesData {
    chargesUserData: Array<ChargesRow>
    requestStatus: boolean
    requestPasswordStatus: boolean
    error: string | null
}

const testData= [
    {
        id: "kjdkjnd",
        date: "2021-10-05",
        loanId: "dkl-kndo",
        name: "Name of something",
        status: true,
        total: 3121516,
    },
    {
        id: "kjdknd",
        date: "2021-10-06",
        loanId: "dkl-kndo",
        name: "Name of something",
        status: false,
        total: 3121516,
    }
]

const initialState: IChargesData = {
    chargesUserData: [],
    requestStatus: false,
    requestPasswordStatus: false,
    error: null,
};

const { reducer, actions } = createSlice({
  name: ReducerName.USER_DASHBOARD,
  initialState,
  reducers: {
    requestStart: (state) => {
      state.error = null;
    },
```

```
    setChargesUserData: (state, action:
PayloadAction<Array<ChargesRow>>) => {
      state.chargesUserData               =
action.payload;
    },

    requestFail: (state) => {
      state.requestStatus = false;
    },

    requestSuccess: (state) => {
      state.requestStatus = true;
    },

    clearError: (state) => {
      state.error = null;
      state.requestPasswordStatus = false;
    },
  },
});



export    const    getUserChargesAction    =
(payload: string | null) => ({
  type:
UserChargesSagaTypes.GET_USER_CHARGES,
  payload: payload,
});

const userChargesdActionCreator = {
  ...actions,
};

export { userChargesdActionCreator, reducer
};

const initialState: IActivationMessage = {
  status: ActivationStatus.SUCCESS,
  email: '',
  message: '',
};

const { reducer, actions } = createSlice({
  name: ReducerName.ACTIVATION,
  initialState,
  reducers: {
    request: (state, action) => state,
    activate: (state, action) => state,
    setStatus: (state, action) => (
      state = {...state, ...action.payload}
    )
  },
});

const ActivationActionCreator = {
  ...actions,
};

export  {  ActivationActionCreator,  reducer
};

export interface Merchant {
  name: string;
  imageUrl: string;
  partnerCode: number;
  price: number;
  discount: number;
  created_at: Date;
}

interface MerchantStateType {
  merchants: Merchant[];
  error: string | null;
  loading: boolean;
  requestStatus: boolean;
}

const initialState: MerchantStateType = {
  merchants: [],
  error: null,
  loading: false,
  requestStatus: false,
};

const { reducer, actions } = createSlice({
  name: ReducerName.MERCHANT,
```

```
  initialState,

  reducers: {
    requestStart: (state) => {
      state.loading = true;
      state.error = null;
    },
    resetSucceed: (state) => {
      state.loading = false;
    },
    requestFailed: (state, action) => {
      state.error = action.payload;
      state.loading = false;
    },
    setMerchants:      (state,      action:
PayloadAction<Merchant[]>) => {
      state.merchants = action.payload;
    },
  },
});

export const loadMerchAction = () => ({
  type:
MerchantSagaTypes.LOAD_ALL_MERCHANTS,
});

export const loadNewestMerchAction = () =>
({
  type:
MerchantSagaTypes.LOAD_NEWEST_MERCHANTS,
});

export const loadCheapestMerchAction = ()
=> ({
  type:
MerchantSagaTypes.LOAD_CHEAPEST_MERCHANTS,
});

export const loadMerchWithDiscountAction =
() => ({
  type:
MerchantSagaTypes.LOAD_DISCOUNTED_MERCHANT
S,
});

const merchantActionCreator = {
  ...actions,
};

export { merchantActionCreator, reducer };

export interface ITransaction {
  user_id: number;
  merchant_id: number;
  order_id: number | null;
  timeStamp: Date;
  amount: number;
  order_details: string;
  authorization: string;
  expiration_date: Date;
  processed: boolean;
}

export interface ITransactionState {
  transactions: ITransaction[];
  error: string | null;
  loading: boolean;
  requestStatus: boolean;
}

const initialState: ITransactionState = {
  transactions: [],
  error: null,
  loading: false,
  requestStatus: false,
};

const { reducer, actions } = createSlice({
  name: ReducerName.MERCHANT,
  initialState,

  reducers: {
    requestStart: (state) => {
      state.loading = true;
      state.error = null;
    },
```

```
    resetSucceed: (state) => {
      state.loading = false;
    },
    requestFailed: (state, action) => {
      state.error = action.payload;
      state.loading = false;
    },
    setTransactions:    (state,    action:
PayloadAction<ITransaction[]>) => {
      state.transactions = action.payload;
    },
  },
});

export   const   loadTransactionsAction   =
(merchant_id: number) => ({
  type:
TransactionsSagaTypes.LOAD_TRANSACTIONS,
  payload: merchant_id,
});

const transactionsActionCreator = {
  ...actions,
};

export { transactionsActionCreator, reducer
};

type Filter = 'all' | 'online' | 'offline'



interface IState {
  loading: boolean,
  error: string | null,
  count: number,
  activePage: number,
  search: string,
  useFilter: Filter,
  setActive: boolean,
  itemsPerPage: number,
  data: any,
  upDelStatus: boolean,
  user: any
}

const initialState: IState = {
  loading: false,
  error: null,
  activePage: 1,
  data: [],
  useFilter: 'all',
  setActive: false,
  itemsPerPage: 2,
  search: '',
  count: 0,
  user: {},
  upDelStatus: false,
};



const { reducer, actions } = createSlice({
  name: ReducerName.USER_MANAGE,
  initialState,
  reducers: {
    changePage: (state, action) => {
      state.activePage = action.payload;
    },
    changeItemsPerPage: (state, action) =>
{
      state.itemsPerPage = action.payload;
    },
    changeFormState: (state, action) => {
      state.search = action.payload;
    },
    changeUseFilter: (state, action) => {
      state.useFilter = action.payload;
    },
    startRequest: (state) => {
      state.count = 0;
      state.loading = true;
      state.error = null;
      state.data = [];
      state.upDelStatus = false;
      state.user = {};
    },
    succeedRequest: (state, action) => {
```

```
      const { count, data } =
action.payload;
      state.count = count;
      state.data = data;
      state.loading = false;
    },
    startSetActiveUser: (state) => {
      state.setActive = false;
      state.loading = true;
      state.error = null;
    },
    setActiveUser: (state) => {
      state.loading = false;
      state.setActive = true;
    },
    startGetUserRequest: (state) => {
      state.loading = true;
      state.error = null;
      state.user = {};
    },
    succeedGetUser: (state, action) => {
      state.user = action.payload;
      state.loading = false;
    },
    succeedUpDel: (state) => {
      state.upDelStatus = true;
      state.loading = false;
    },
    failedRequest: (state, action) => {
      state.error = action.payload;
      state.loading = false;
    },
  },
});

export const getPaginationUsersAction =
(data: any) => ({
  type:
UserManageSagaTypes.MANAGE_GET_USERS_WITH_
PAGINATION,
  payload: data,
});

export const getUserForUpdateAction =
(data: any) => ({
  type:
UserManageSagaTypes.MANAGE_GET_USER_TO_UPD
ATE,
  payload: data,
});

export const updateUserAction = (data: any)
=> ({
  type:
UserManageSagaTypes.MANAGE_UPDATE_USER,
  payload: data,
});

export const createUserAction = (data: any)
=> ({
  type:
UserManageSagaTypes.MANAGE_CREATE_USER,
  payload: data,
});

export const setUserActiveAction = (data:
any) => ({
  type:
UserManageSagaTypes.SET_USER_ACTIVE,
  payload: data,
});

const UserManageActionCreator = {
  ...actions,
};

export { UserManageActionCreator, reducer
};

interface IMerchant {
  id: number;
  name: string;
}

interface UserData {
  user: IProfile;
```

```
  data: any;
  requestStatus: boolean;
  requestPasswordStatus: boolean;
  error: string | null;
  role: string;
}

const initialState: UserData = {
  user: {
    id: 0,
    sid: 0,
    balance: 0,
    firstName: '',
    lastName: '',
    phoneNumber: '',
    birthDate: '',
    email: '',
    avatar: '',
    role: '',
    cityAddress: '',
    isActive: false,
    merchant: null,
  },
  data: [],
  role: '',
  requestStatus: false,
  requestPasswordStatus: false,
  error: null,
};

const { reducer, actions } = createSlice({
  name: ReducerName.USER_DATA,
  initialState,
  reducers: {
    requestStart: (state) => {
      state.error = null;
    },
    setUser:        (state,        action:
PayloadAction<IProfile>) => {
      state.user = action.payload;
      if (state.user.birthDate) {
        state.user.birthDate             =
state.user.birthDate.slice(0, 10);
      }
    },
    setAvatar:       (state,       action:
PayloadAction<string>) => {
      state.user.avatar = action.payload;
    },
    setRole: (state, action) => {
      state.role = action.payload;
    },
    requestFail: (state) => {
      state.requestStatus = false;
    },

    requestSuccess: (state) => {
      state.requestStatus = true;
    },

    requestSuccessPassword: (state, action)
=> {
      const { data } = action.payload;
      state.data = data;
      state.requestPasswordStatus = true;
    },

    requestFailPassword: (state, action) =>
{
      state.error = action.payload;
      state.requestStatus = false;
      state.requestPasswordStatus = false;
    },

    clearError: (state) => {
      state.error = null;
      state.requestPasswordStatus = false;
    },
  },
});

export  const  getUserAction  =  (headers:
string | null) => ({
  type: UserDataSagaTypes.GET_USER,
  payload: headers,
});
```

```
export const updateUserAction = (
  data: { user: IProfile } & { token: string
},
) => ({
  type: UserDataSagaTypes.UPDATE_USER,
  payload: data,
});

export const updateUserPasswordAction = (
  data: { passwordData: IUserPassword } & {
token: string },
) => ({
  type:
UserDataSagaTypes.UPDATE_USER_PASSWORD,
  payload: data,
});

const UserDataActionCreator = {
  ...actions,
};

export { UserDataActionCreator, reducer };
export    {    reducer    as    userReducer,
UserActionCreator          }          from
'./auth/auth.slice';

export {
  reducer as activationReducer,
  ActivationActionCreator,
} from './activation/activation.slice';

export    {    reducer    as    fileReducer,
FileActionCreator          }          from
'./file/file.slice';

export {
  reducer as userDataReducer,
  UserDataActionCreator,
} from './user-data/user-data.slice';

export {
  reducer as userDashboardReducer,
  userDashboardActionCreator,
}       from       './user-dashboard/user-
dashboard.slice';

export {
  reducer as userChargesReducer,
  userChargesdActionCreator,
} from './user-charges/user-charges.slice';

export {
  reducer as userManageReducer,
  UserManageActionCreator,
} from './user-manage/user-manage.slice';

export {
  reducer as merchantReducer,
  merchantActionCreator,
} from './merchants/merchant.slice';

export {
  reducer as ordersReducer,
  ordersActionCreator,
} from './orders/order.slice';

export {
  reducer as merchantHubReducer,
  merchantHubActionCreator,
} from './merchant-hub/merchant-hub.slice';

export {
  reducer as transactionsReducer,
  transactionsActionCreator,
} from './transactions/transactions.slice';

export interface Order {
  name: string;
  price: number;
  created_at: string;
  expires_at: string;
  fulfilled: boolean;
  number_of_payments: number;
  payments_covered: number;
}
```

```
interface OrdersStateType {
  newOrder: IOrder | null;
  orders: Order[];
  error: string | null;
  loading: boolean;
  requestStatus: boolean;
}

const initialState: OrdersStateType = {
  newOrder: null,
  orders: [],
  error: null,
  loading: false,
  requestStatus: false,
};

const { reducer, actions } = createSlice({
  name: ReducerName.ORDERS,
  initialState,

  reducers: {
    requestStart: (state) => {
      state.loading = true;
      state.error = null;
    },
    resetSucceed: (state) => {
      state.loading = false;
    },
    setNewOrder: (state, action) => {
      state.newOrder = action.payload;
    },
    requestFailed: (state, action) => {
      state.error = action.payload;
      state.loading = false;
    },
    setOrders:       (state,       action:
PayloadAction<Order[]>) => {
      state.orders = action.payload;
    },
  },
});
```

```
export  const  loadOrdersAction  =  (data:
number) => ({
  type: OrdersSagaTypes.LOAD_ALL_ORDERS,
  payload: data,
});

export interface IOrder {
  _id?: number;
  user_id: number;
  merchant_id: number;
  name: string;
  image_url: string;
  price: number;
  number_of_payments: number;
  fulfilled: boolean;
  expires_at: Date;
}

export  const  createOrderAction  =  (data:
IOrder) => ({
  type: OrdersSagaTypes.CREATE_ORDER,
  payload: data,
});

const ordersActionCreator = {
  ...actions,
};

export { ordersActionCreator, reducer };

export interface IMerchant {
  sid: number;
  name: string;
  image_url: string;
  link: string;
}

interface MerchantStateType {
  merchants: IMerchant[];
  filteredMerchants: IMerchant[];
  error: string | null;
  loading: boolean;
  requestStatus: boolean;
```

```
}

const initialState: MerchantStateType = {
  merchants: [],
  filteredMerchants: [],
  error: null,
  loading: false,
  requestStatus: false,
};

const { reducer, actions } = createSlice({
  name: ReducerName.MERCHANT,
  initialState,

  reducers: {
    requestStart: (state) => {
      state.loading = true;
      state.error = null;
    },
    resetSucceed: (state) => {
      state.loading = false;
    },
    requestFailed: (state, action) => {
      state.error = action.payload;
      state.loading = false;
    },
    setMerchants:      (state,      action:
PayloadAction<IMerchant[]>) => {
      state.merchants = action.payload;
    },
    setFilteredMerchants:  (state,  action:
PayloadAction<IMerchant[]>) => {
      state.filteredMerchants            =
action.payload;
    },
  },
});

export const loadMerchantsAction = () => ({
  type:
MerchantHubSagaTypes.LOAD_MERCHANTS,
});

export    const    setMerchantsAction    =
(merchants: IMerchant[]) => ({
  type:
MerchantHubSagaTypes.SET_MERCHANTS,
  payload: merchants,
});

const merchantHubActionCreator = {
  ...actions,
};

export { merchantHubActionCreator, reducer
};

  SagaAction,
  SagaUpdatePasswordAction,
  SagaUserAction,
} from '../../common/types';



const apiService = new ApiService();

function* getUser(data: PayloadAction) {
  const  accessToken  =  (data.payload  as
unknown) as string;
  if (!accessToken) return;
  try {
    const user = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.PROFILE}`,
      'GET',
      {
        token: accessToken,
      },
    );
    yield
put(FileActionCreator.cloudUploadSucceed(u
ser.avatar));
    yield
put(UserDataActionCreator.setUser(user));
```

```
    yield
put(UserDataActionCreator.setRole(user.rol
es[0].key));
    yield
put(UserDataActionCreator.requestSuccess()
);
  } catch (e) {
    yield
put(UserDataActionCreator.requestFail());
  }
}

function* updateUser(
  data: PayloadAction<{ user: IProfile } &
{ token: string }>,
) {
  const accessToken = data.payload.token;
  if (!accessToken) return;
  try {
    const user = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.PROFILE}`,
      'PUT',
      {
        body: data.payload.user,
        token: accessToken,
      },
    );
    yield
put(UserDataActionCreator.setUser(user));
    yield
put(UserDataActionCreator.requestSuccess()
);
  } catch (e) {
    yield
put(UserDataActionCreator.requestFail());
  }
}

function* updateUserPassword(
  data:    PayloadAction<{    passwordData:
IUserPassword } & { token: string }>,
) {
  try {
    const response = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.SECURITY_P
ASSWORD}`,
      'PUT',
      {
        token: data.payload.token,
        body: data.payload.passwordData,
      },
    );
    yield
put(UserDataActionCreator.requestSuccessPa
ssword(String('Confirm')));
  } catch (e) {
    yield
put(UserDataActionCreator.requestFailPassw
ord(String(e)));
  }
}

function* userDataSagaWatcher() {
  yield
takeEvery<SagaAction>(UserDataSagaTypes.GE
T_USER, getUser);
  yield
takeEvery<SagaUserAction>(UserDataSagaType
s.UPDATE_USER, updateUser);
  yield
takeEvery<SagaUpdatePasswordAction>(

UserDataSagaTypes.UPDATE_USER_PASSWORD,
    updateUserPassword,
  );
}

export default userDataSagaWatcher;

// const apiService = new ApiService();
```

```
const delayDashboardData = (ms : number) =>
new Promise(res => setTimeout(res, ms));


const testData= [
    {
        id: "kjdkjnd",
        date: "2021-08-10",
        loanId: "dk5-rzd",
        name:    "Money    transfer    from
Vladimir",
        status: true,
        total: 516,
    },
    {
        id: "kjdknd",
        date: "2021-10-08",
        loanId: "dk6-rzd",
        name: "Money transfer from Vladimir
",
        status: false,
        total: 312,
    },
    {
        id: "It works",
        date: "2021-11-08",
        loanId: "dk6-rzd",
        name:    "Money    transfer    from
Vladimir",
        status: false,
        total: 300,
    }
]

function*              getUserCharges(data:
PayloadAction) {
  const  accessToken  =  (data.payload  as
unknown) as string;
  if (!accessToken) return;
  try {
    const userCharges = testData;
    yield delayDashboardData(1000);
    const userDashboard = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.PROFILE}`,
      'GET',
      {
        token: accessToken,
      },
    );
    console.log(userCharges)
    yield
put(userChargesdActionCreator.setChargesUs
erData(userCharges));
    yield
put(userChargesdActionCreator.requestSucce
ss());
  } catch (e) {
    yield
put(userChargesdActionCreator.requestFail(
));
  }
}

function* userDataChargesSagaWatcher() {
  yield
takeEvery<SagaAction>(UserChargesSagaTypes
.GET_USER_CHARGES, getUserCharges);
}

export default userDataChargesSagaWatcher;

const apiService = new ApiService();

function*            loadFileToCloud(data:
PayloadAction) {
  const ls = new LocalstorageService();
  const     {     accessToken     }     =
ls.getItem(LocalstorageKeys.AUTH);

  try {
    yield
put(FileActionCreator.requestStart());
    const { url }: Record<string, string> =
yield call(
```

```
      apiService.httpRequest,
      '/file-upload',
      'POST',
      {
        body: data.payload,
        headers:     {     'Content-Type':
'multipart/form-data' },
        token: accessToken,
      },
    );
    yield
put(UserDataActionCreator.setAvatar(url));
    yield
put(FileActionCreator.cloudUploadSucceed(u
rl));
  } catch (e) {
    yield
put(FileActionCreator.requestFailed(String
(e)));
  }
}

function* fileSagaWatcher(): Generator {
  yield
takeEvery<SagaAction>(FileSagasTypes.LOAD_
TO_CLOUD, loadFileToCloud);
}

export default fileSagaWatcher;

const apiService = new ApiService();

function*              activateUser(action:
Record<string, string>) {
  const { payload: token } = action;
  const                url                =
`${ApiPath.USERS}${UsersApiPath.ACTIVATION
}/${token}`;

  const                 activationResponse:
IActivationMessage          =          yield
call(apiService.httpRequest,           url,
HttpMethod.PUT);
  yield
put(ActivationActionCreator.setStatus(acti
vationResponse));
}

function*     sendActivationRequest(action:
Record<string, string>) {
  const { payload: email } = action;
  const                url                =
`${ApiPath.USERS}${UsersApiPath.ACTIVATION
_REQUEST}`;
  apiService.httpRequest(url,
HttpMethod.PUT);
  const                 activationResponse:
IActivationMessage = yield call(
    apiService.httpRequest,
    url,
    HttpMethod.PUT,
    { body: { email } }
  );
  yield
put(ActivationActionCreator.setStatus(acti
vationResponse));
}

function* activationSaga(): Generator {
  yield
takeEvery(ActivationActionCreator.activate
, activateUser);
  yield
takeEvery(ActivationActionCreator.request,
sendActivationRequest);
}

export default activationSaga;

const apiService = new ApiService();

function* getMerchants() {
  try {
    yield
put(merchantHubActionCreator.requestStart(
));
```

```
    const merchants: IMerchant[] = yield
call(
      apiService.httpRequest,
      `${ApiPath.MERCHANTS}`,
    );
    yield
put(merchantHubActionCreator.setMerchants(
merchants));
    yield
put(merchantHubActionCreator.setFilteredMe
rchants(merchants));
    yield
put(merchantHubActionCreator.resetSucceed(
));
  } catch (e) {
    yield
put(merchantHubActionCreator.requestFailed
(e));
  }
}

function* setMerchants(data: PayloadAction)
{
  yield put(

merchantHubActionCreator.setFilteredMercha
nts(
      data.payload     as     unknown     as
IMerchant[],
    ),
  );
}

function* merchantHubSagaWatcher() {
  yield takeEvery<SagaAction>(
    MerchantHubSagaTypes.LOAD_MERCHANTS,
    getMerchants,
  );
  yield
takeEvery<SagaAction>(MerchantHubSagaTypes
.SET_MERCHANTS, setMerchants);
}

export default merchantHubSagaWatcher;

const apiService = new ApiOrdersService();

function* getOrders(data: PayloadAction) {
  try {
    yield
put(ordersActionCreator.requestStart());
    const orders: Order[] = yield call(
      apiService.httpRequest,

`orders/merchant_id=${data.payload}`,
    );
    yield
put(ordersActionCreator.setOrders(orders))
;
    yield
put(ordersActionCreator.resetSucceed());
  } catch (e) {
    yield
put(ordersActionCreator.requestFailed(e));
  }
}

function* createOrder(data: PayloadAction)
{
  try {
    yield
put(ordersActionCreator.requestStart());

    const response: IOrder = yield call(
      apiService.httpRequest,
      `orders/`,
      'POST',
      {
        body: data.payload,
      },
    );

    console.log(response);
```

```
    yield
put(ordersActionCreator.setNewOrder(respon
se));

    yield
put(ordersActionCreator.resetSucceed());
  } catch (e) {
    yield
put(ordersActionCreator.requestFailed(e));
  }
}

function* ordersSagaWatcher() {
  yield
takeEvery<SagaAction>(OrdersSagaTypes.LOAD
_ALL_ORDERS, getOrders);
  yield
takeEvery<SagaAction>(OrdersSagaTypes.CREA
TE_ORDER, createOrder);
}

export default ordersSagaWatcher;

const apiService = new ApiService();

function* getUsersWithPagination(
  data:  PayloadAction<{  pagination:  any;
token: string }>,
) {
  try {
    yield
put(UserManageActionCreator.startRequest()
);
    const response = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.PAG_USERS}
`,
      'POST',
      {
        body: data.payload.pagination,
        token: data.payload.token,
      },
    );
    yield
put(UserManageActionCreator.succeedRequest
(response));
  } catch (e) {
    yield
put(UserManageActionCreator.failedRequest(
String(e)));
  }
}

function*           getUserForUpdate(data:
PayloadAction<{ id: number; token: string
}>) {
  try {
    yield
put(UserManageActionCreator.startRequest()
);
    const response = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}/${data.payload.id}`,
      'GET',
      {
        token: data.payload.token,
      },
    );
    const newUser = {
      ...response,
      birthDate: response.birthDate
        ? response.birthDate.slice(0, 10)
        : response.birthDate,
      roles: response.roles[0].id,
    };
    yield
put(UserManageActionCreator.succeedGetUser
(newUser));
  } catch (e) {
    yield
put(UserManageActionCreator.failedRequest(
String(e)));
  }
}
```

```
function* updateUser(
  data: PayloadAction<{
    data: { id: number; token: string };
    form: { user: any; role: { roleId:
number } };
  }>,
) {
  console.log('UPDATE USER');
  try {
    yield
put(UserManageActionCreator.startRequest()
);
    const response = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.MANAGE}/${
data.payload.data.id}`,
      'PUT',
      {
        token: data.payload.data.token,
        body: data.payload.form,
        params: { id: data.payload.data.id
},
      },
    );
    console.log('response', response);
    yield
put(UserManageActionCreator.succeedUpDel()
);
  } catch (e) {
    yield
put(UserManageActionCreator.failedRequest(
String(e)));
  }
}

function* createUser(
  data: PayloadAction<{
    data: { id: number; token: string };
    form: { user: { user: any }; role: {
roleId: number } };
  }>,
) {
  try {
    yield
put(UserManageActionCreator.startRequest()
);
    const response = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.MANAGE}`,
      'POST',
      {
        token: data.payload.data.token,
        body: data.payload.form,
      },
    );
    yield
put(UserManageActionCreator.succeedUpDel()
);
  } catch (e) {
    yield
put(UserManageActionCreator.failedRequest(
String(e)));
  }
}

function*              setUserActive(data:
PayloadAction<{ id: number; token: string
}>) {
  try {
    yield
put(UserManageActionCreator.startSetActive
User());
    const response = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.MANAGE_ACT
IVE}/${data.payload.id}`,
      'PUT',
      {
        token: data.payload.token,
      },
    );
```

```
    yield
put(UserManageActionCreator.setActiveUser(
));
  } catch (e) {
    yield
put(UserManageActionCreator.failedRequest(
String(e)));
  }
}

function* userManageSagaWatcher() {
  yield
takeEvery<SagaActionWithTokenAndPayload<{
pagination: any }>>(

UserManageSagaTypes.MANAGE_GET_USERS_WITH_
PAGINATION,
    getUsersWithPagination,
  );
  yield
takeEvery<SagaActionWithTokenAndPayload<{
id: number }>>(

UserManageSagaTypes.MANAGE_GET_USER_TO_UPD
ATE,
    getUserForUpdate,
  );
  yield takeEvery<
    SagaActionWithTokenAndPayload<{
      data: { id: number; token: string };
      form: { user: { user: any }; role: {
roleId: number } };
    }>

>(UserManageSagaTypes.MANAGE_CREATE_USER,
createUser);
  yield takeEvery<
    SagaActionWithTokenAndPayload<{
      data: { id: number; token: string };
      form: { user: { user: any }; role: {
roleId: number } };
    }>

>(UserManageSagaTypes.MANAGE_UPDATE_USER,
updateUser);
  yield
takeEvery<SagaActionWithTokenAndPayload<{
id: number }>>(
    UserManageSagaTypes.SET_USER_ACTIVE,
    setUserActive,
  );
}

export default userManageSagaWatcher;

const apiService = new ApiService();

function*            getTransactions(data:
PayloadAction) {
  try {
    yield
put(transactionsActionCreator.requestStart
());
    const  transactions:  ITransaction[]  =
yield call(
      apiService.httpRequest,

`transactions/merchant=${data.payload}`,
    );
    console.log(transactions);
    yield
put(transactionsActionCreator.setTransacti
ons(transactions));
    yield
put(transactionsActionCreator.resetSucceed
());
  } catch (e) {
    yield
put(transactionsActionCreator.requestFaile
d(e));
  }
}

function* transactionsSagaWatcher() {
  yield takeEvery<SagaAction>(
```

```
TransactionsSagaTypes.LOAD_TRANSACTIONS,
    getTransactions,
  );
}

export default transactionsSagaWatcher;

function* rootSaga(): Generator {
  yield all([
    authSagaWatcher(),
    activationSaga(),
    fileSagaWatcher(),
    userDataSagaWatcher(),
    userManageSagaWatcher(),
    userDataDashboardSagaWatcher(),
    userDataChargesSagaWatcher(),
    merchantSagaWatcher(),
    ordersSagaWatcher(),
    merchantHubSagaWatcher(),
    transactionsSagaWatcher(),
  ]);
}

export default rootSaga;

const apiService = new ApiService();
const      apiOrdersService      =      new
ApiOrdersService();

function* getUserData(data: PayloadAction)
{
  const   accessToken   =   data.payload   as
unknown as string;
  if (!accessToken) return;
  try {
    yield
put(userDashboardActionCreator.requestStar
t());
    const user: IUser = yield call(
      apiService.httpRequest,

`${ApiPath.USERS}${UsersApiPath.PROFILE}`,
      'GET',
      { token: accessToken },
    );
    const orders: Order[] = yield call(
      apiOrdersService.httpRequest,
      `orders/user_id=${user.id}`,
      'GET',
    );
    const loan: number = yield call(
      apiOrdersService.httpRequest,
      `orders/loan/user_id=${user.id}`,
    );
    const processed: number = yield call(
      apiOrdersService.httpRequest,

`orders/processed/user_id=${user.id}`,
    );

    console.log('orders', orders);

    yield
put(userDashboardActionCreator.setUserData
(user));
    yield
put(userDashboardActionCreator.setOrders(o
rders));
    yield
put(userDashboardActionCreator.setLoan(loa
n));
    yield
put(userDashboardActionCreator.setProcesse
d(processed));
    yield
put(userDashboardActionCreator.requestSucc
ess());
  } catch (error) {
    yield console.log(error);
    yield
put(userDashboardActionCreator.requestFail
());
  }
}
```

```
function* userDataSagaWatcher() {
  yield takeEvery<SagaAction>(
    UserDashboardSagaTypes.GET_USER_DATA,
    getUserData,
  );
}

export default userDataSagaWatcher;

const apiService = new ApiService();

export interface ResponseGenerator {
  tokens?: {
    accessToken: string;
    refreshToken: string;
  };
  message?: string;
}

function* signUpUser(data: PayloadAction) {
  try {
    yield
put(UserActionCreator.requestStart());
    const confirm: ResponseGenerator =
yield call(
      apiService.httpRequest,
      ApiPath.USERS,
      'POST',
      {
        body: data.payload,
      },
    );
    yield
put(UserActionCreator.signUpSucceed());
  } catch (e) {
    yield
put(UserActionCreator.requestFailed(String
(e)));
  }
}

function* loginUser(data: PayloadAction) {
  try {
    yield
put(UserActionCreator.requestStart());
    const authResult: ResponseGenerator =
yield call(
      apiService.httpRequest,

`${ApiPath.AUTH}${AuthApiPath.LOGIN}`,
      'POST',
      {
        body: data.payload,
      },
    );
    yield
put(UserActionCreator.loginSucceed(authRes
ult.tokens));
  } catch (e) {
    yield
put(UserActionCreator.requestFailed(String
(e)));
  }
}

function*              resetPassword(data:
PayloadAction) {
  try {
    yield
put(UserActionCreator.requestStart());
    const confirm: ResponseGenerator =
yield call(
      apiService.httpRequest,

`${ApiPath.AUTH}${AuthApiPath.RESET_PASSWO
RD}`,
      'POST',
      { body: data.payload },
    );
    yield
put(UserActionCreator.resetSucceed(confirm
.message));
  } catch (e) {
    yield
put(UserActionCreator.requestFailed(String
(e)));
```

```
  }
}

function*              verifyPassword(data:
PayloadAction) {
  try {
    yield
put(UserActionCreator.requestStart());
    const   confirm:   ResponseGenerator   =
yield call(
      apiService.httpRequest,

`${ApiPath.AUTH}${AuthApiPath.VERIFY_PASSW
ORD}`,
      'POST',
      { body: data.payload },
    );
    yield
put(UserActionCreator.verifySucceed());
  } catch (e) {
    yield
put(UserActionCreator.requestFailed(String
(e)));
  }
}

function* logoutUser() {
  yield
put(UserActionCreator.requestStart());
  const    localstorageService    =    new
LocalstorageService();

localstorageService.removeItem(Localstorag
eKeys.AUTH);
  const               token               =
localstorageService.getItem(LocalstorageKe
ys.AUTH);
  if (!token) {
    yield
put(UserActionCreator.logoutSucceed());
  } else {
    yield
put(UserActionCreator.requestFailed('Logou
t failed'));
  }
}

function* authSagaWatcher() {
  yield
takeEvery<SagaAction>(AuthSagasTypes.LOGIN
_USER, loginUser);
  yield
takeEvery<SagaAction>(AuthSagasTypes.REFRE
SH_PASSWORD, resetPassword);
  yield takeEvery<SagaAction>(
    AuthSagasTypes.VERIFY_PASSWORD_CHANGE,
    verifyPassword,
  );
  yield
takeLatest<SagaAction>(AuthSagasTypes.REGI
STER_USER, signUpUser);
  yield
takeEvery<SagaAction>(AuthSagasTypes.LOGOU
T_USER, logoutUser);
}

export default authSagaWatcher;

const apiService = new ApiService();

function* getMerchants() {
  try {
    yield
put(merchantActionCreator.requestStart());
    const  merchants:  Merchant[]  =  yield
call(
      apiService.httpRequest,

`${ApiPath.PRODUCTS}${ProductApiPath.ROOT}
`,
    );
    yield
put(merchantActionCreator.setMerchants(mer
chants));
```

```
    yield
put(merchantActionCreator.resetSucceed());
  } catch (e) {
    yield
put(merchantActionCreator.requestFailed(e)
);
  }
}

function* getNewestMerchants() {
  try {
    yield
put(merchantActionCreator.requestStart());
    const  merchants:  Merchant[]  =  yield
call(
      apiService.httpRequest,

`${ApiPath.PRODUCTS}${ProductApiPath.NEWES
T}`,
    );
    yield
put(merchantActionCreator.setMerchants(mer
chants));
    yield
put(merchantActionCreator.resetSucceed());
  } catch (e) {
    yield
put(merchantActionCreator.requestFailed(e)
);
  }
}

function* getMerchantsWithDiscount() {
  try {
    yield
put(merchantActionCreator.requestStart());
    const  merchants:  Merchant[]  =  yield
call(
      apiService.httpRequest,

`${ApiPath.PRODUCTS}${ProductApiPath.WITH_
DISCOUND}`,
    );
    yield
put(merchantActionCreator.setMerchants(mer
chants));
    yield
put(merchantActionCreator.resetSucceed());
  } catch (e) {
    yield
put(merchantActionCreator.requestFailed(e)
);
  }
}

function* getCheapestMerchants() {
  try {
    yield
put(merchantActionCreator.requestStart());
    const  merchants:  Merchant[]  =  yield
call(
      apiService.httpRequest,

`${ApiPath.PRODUCTS}${ProductApiPath.CHEAP
EST}`,
    );
    yield
put(merchantActionCreator.setMerchants(mer
chants));
    yield
put(merchantActionCreator.resetSucceed());
  } catch (e) {
    yield
put(merchantActionCreator.requestFailed(e)
);
  }
}

function* merchantSagaWatcher() {
  yield takeEvery<SagaAction>(
    MerchantSagaTypes.LOAD_ALL_MERCHANTS,
    getMerchants,
  );
  yield takeEvery<SagaAction>(

MerchantSagaTypes.LOAD_NEWEST_MERCHANTS,
```

```
    getNewestMerchants,
  );
  yield takeEvery<SagaAction>(
    MerchantSagaTypes.LOAD_CHEAPEST_MERCHANTS,
    getCheapestMerchants,
  );
  yield takeEvery<SagaAction>(
    MerchantSagaTypes.LOAD_DISCOUNTED_MERCHANTS,
    getMerchantsWithDiscount,
  );
}

export default merchantSagaWatcher;
  userReducer,
  activationReducer,
  fileReducer,
  userDataReducer,
  userManageReducer,
  userDashboardReducer,
  userChargesReducer,
  merchantReducer,
  ordersReducer,
  merchantHubReducer,
  transactionsReducer,
} from './slices';

const sagaMiddleware = createSagaMiddleware();

const store = configureStore({
  reducer: {
    [ReducerName.USER]: userReducer,
    [ReducerName.ACTIVATION]: activationReducer,
    [ReducerName.FILE]: fileReducer,
    [ReducerName.USER_DATA]: userDataReducer,
    [ReducerName.USER_MANAGE]: userManageReducer,
    [ReducerName.USER_DASHBOARD]: userDashboardReducer,
    [ReducerName.MERCHANT]: merchantReducer,
    [ReducerName.ORDERS]: ordersReducer,
    [ReducerName.USER_CHARGES]: userChargesReducer,
    [ReducerName.MERCHANT_HUB]: merchantHubReducer,
    [ReducerName.TRANSACTIONS]: transactionsReducer,
  },
  middleware: (cdm) =>
    cdm({
      serializableCheck: {
        ignoredActions: [FileSagasTypes.LOAD_TO_CLOUD],
      },
    }).concat(sagaMiddleware),
});

sagaMiddleware.run(rootSaga);

export { store };
```