НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
Наталія АУШЕВА

“___”_____202__р.

Дипломна робота

на здобуття ступеня бакалавра
за освітньо-професійною програмою
“Цифрові технології в енергетиці”
зі спеціальності *122 “Комп’ютерні науки”*

на тему: Система менеджменту бізнес процесів, моніторингу та аналізу відносин з клієнтом

Виконав (-ла):
студент (-ка) IV курсу, групи ТР-03

БОЙКО Нікіта Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: *доцент каф. цифрових технологій в енергетиці*
доц. Полягушко Любов Григорівна _____
(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Рецензент:_____
_____
_____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім’я, по батькові) (підпис)

Н.контроль: *асистент ГОЛОВАКІН Микита Андрійович* _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____
(підпис)

Київ – 2024

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

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
Наталія АУШЕВА

«__»_____202___р.

ЗАВДАННЯ
на дипломну роботу студенту

БОЙКО Нікіті Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи **Система менеджменту бізнес процесів, моніторингу та аналізу відносин з клієнтом**

керівник роботи
Полягушко Любов Григорівна, доцент
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 202___р. №_____

2. Термін подання роботи студентом ***07.06.2024 року***

3. Вихідні дані до роботи мова програмування Java, мова програмування JavaScript, мова програмування TypeScript, фреймворки Spring, Hibernate, бібліотека React, СКБД PostgreSQL, Redis, середовище розробки Intellij IDEA, інструмент тестування Postman

4. Перелік питань, які потрібно розробити
1) проаналізувати існуючі рішення, їх функціональні можливості та обмеження

2) розробити архітектуру системи, спроектувати схему бази даних та визначити схему взаємодії компонентів застосунку

3) розробити спроєктовану серверну та інтерфейс частини застосунку

4) піддати систему тестуванню для перевірки функціональності

5) створити простий та інтуїтивний програмний веб-інтерфейс

6) представити процес застосування веб-інтерфейсу та інструкцію запуску системи

5. Орієнтований перелік ілюстративного матеріалу діаграми, що демонструють функціонал, архітектуру системи, бази даних, взаємодію користувачів із системою, класи програмного забезпечення; приклади роботи з програмним продуктом

6. Дата видачі завдання «15» вересня *2023 р.*

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі	17-20.04.24	
3.	Розробка архітектури та загальної структури системи	21-25.04.24	
4.	Розробка частин окремих підсистем	25.04-02.05.24	
5.	Програмна реалізація системи	03-07.05.24	
6.	Оформлення пояснювальної записки	08-12.05.24	
7.	Захист програмного продукту	13-17.05.23	
8.	Передзахист	03-06.06.23	
9.	Подання готової роботи на кафедру	07.06.2024	
10.	Захист	17-31.06.2024	

Студент _____ **Нікіта БОЙКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Любов ПОЛЯГУШКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

АНОТАЦІЯ

Дипломна робота виконана на 63 сторінках, містить 41 ілюстрацій, 1 додаток, 33 джерел в переліку посилань.

Мета роботи – створення системи менеджменту бізнес процесів, моніторингу та аналізу відносин з клієнтом.

Методи та засоби: мова програмування Java, мова програмування JavaScript, мова програмування TypeScript, фреймворки Spring, Hibernate, бібліотека React, СКБД PostgreSQL, Redis, середовище розробки Intellij IDEA.

Результат – програмний інструментарій для управління бізнес процесами, моніторингу і аналізу відносин з клієнтами.

Ключові слова: МЕНЕДЖМЕНТ БІЗНЕС ПРОЦЕСІВ, КЛІЄНТСЬКІ ВІДНОСИНИ, АНАЛІЗ ДАНИХ, JAVA, JAVASCRIPT, SPRING, HIBERNATE, REACT, POSTGRESQL, REDIS, INTELLIJ IDEA, POSTMAN.

ABSTRACT

The diploma work is written on 63 pages, contains 41 illustrations, 1 appendix, 33 sources in the reference list.

The aim of the work is the development of a business process management, monitoring, and customer relationship analysis system.

Methods and tools: Java programming language, JavaScript programming language, TypeScript programming language, Spring framework, Hibernate framework, React library, PostgreSQL DBMS, Redis, development environment Intellij IDEA.

Result – software tools for managing business processes, monitoring, and analyzing customer relationships.

Keywords: BUSINESS PROCESS MANAGEMENT, CUSTOMER RELATIONSHIPS, DATA ANALYSIS, JAVA, JAVASCRIPT, SPRING, HIBERNATE, REACT, POSTGRESQL, REDIS, INTELLIJ IDEA, POSTMAN

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Java – об’єктно-орієнтована мова програмування

PostgreSQL – вільна система керування реляційними базами даних

СУБД – система управління базами даних (англ. Database Management System)

IDE – інтегроване середовище розробки (англ. Integrated Development Environment)

E-commerce – електронна комерція (англ. Electronic commerce)

HTML – мова розмітки гіпертексту (англ. HyperText Markup Language)

CSS – каскадні таблиці стилів (англ. Cascading Style Sheets)

Intellij IDEA – редактор вихідного коду

Apache Kafka – це розподілене сховище подій і платформа для їх багатопотокового оброблення

Elasticsearch – вільне програмне забезпечення, пошуковий сервер, розроблений на базі Lucene

Мікросервіси – архітектурний стиль, за яким єдиний застосунок будується як сукупність невеличких сервісів, кожен з яких працює у своєму власному процесі та спілкується з рештою, використовуючи прості та швидкі протоколи передачі даних

JS – JavaScript, мова програмування для створення інтерактивних елементів веб-сторінок

Spring – фреймворк для розробки додатків на Java

Spring Cloud – набір інструментів для створення розподілених систем на базі Spring

Hibernate – фреймворк для об'єктно-реляційного відображення в Java

Netflix Eureka – сервіс реєстрації для балансування навантаження та виявлення сервісів у хмарних системах

Spring Boot – фреймворк для швидкої розробки додатків на базі Spring

Spring Boot Admin – інструмент для адміністрування та моніторингу додатків на базі Spring Boot

React JS – бібліотека для створення користувацьких інтерфейсів на JavaScript

JWT - стандарт токена доступу на основі JSON, стандартизованого в RFC 7519

GitHub - один з найбільших вебсервісів для спільної розробки програмного забезпечення

API - це набір чітко визначених методів для взаємодії різних компонентів (англ. Application programming interface)

Docker - інструментарій для управління ізольованими Linux-контейнерами

ВСТУП

У сучасному світі, де конкуренція на ринку постійно зростає, ефективне управління бізнес-процесами та взаємодія з клієнтами є ключовими складовими успіху будь-якої компанії. Розуміння та впровадження систем управління бізнес-процесами сприяє оптимізації внутрішніх процесів, підвищенню продуктивності та забезпеченню високої якості продуктів чи послуг. Ефективна комунікація та взаємодія з клієнтами, можливість розширення торгівельної діяльності для малого та середнього бізнесу, легка інтеграція з додатковими функціональними вимогами є важливими аспектами сучасного підприємства. В умовах постійних змін та швидкого розвитку технологій, забезпечення задоволення клієнтів та збереження їх лояльності вимагає від компаній не лише якісних продуктів чи послуг, але й вивчення їх потреб, вчасну реакцію на зміни та побудову довгострокових стосунків.

Метою роботи є розробка програмного забезпечення для керування бізнес-процесами, моніторингу та аналізу відносин з клієнтами. Це програмне забезпечення має забезпечити підвищення ефективності підприємства та задоволення потреб сучасного споживача шляхом інтеграції різних систем управління бізнес-процесами з моніторинговими та аналітичними методами.

Для досягнення цієї мети, робота передбачає вирішення наступних завдань: аналіз підходів, методів та алгоритмів розв’язання задачі управління бізнес-процесами та взаємодії з клієнтами, аналіз програмних засобів для реалізації програмної системи управління бізнес-процесами та відносин з клієнтами, розробка програмного забезпечення відповідно до теми дипломної роботи, тестування функціоналу програмного забезпечення на предмет відповідності поставленим задачам та коректності роботи, демонстрація розробленого програмного забезпечення.

Робота складається з вступу, п’яти розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто задачі

розроблюваної системи, описано схему взаємодії її частин та функціональні вимоги. Другий розділ присвячений аналізу програмних засобів, необхідних для реалізації цієї системи. Третій розділ зосереджено на засобах розробки програмного забезпечення. Четвертий – на програмній реалізації. Останній присвячено роботі користувача з системою. Робота містить 1 додаток, 41 рисунок. Список використаних джерел налічує 33 найменувань. Загальний обсяг роботи становить 94 сторінки.

Дипломна робота спрямована на розробку ефективного програмного забезпечення для управління бізнес-процесами та взаємодії з клієнтами, що є актуальним завданням для сучасних підприємств в умовах швидкого розвитку технологій та підвищення конкуренції на ринку.

Задача системи менеджменту та моніторингу, аналізу клієнтської взаємодії. У цьому розділі буде проведено постановку задачі для реалізації застосунку. Буде описано схему взаємодії підсистем

Аналіз проблеми електронної комерційної діяльності. У цьому розділі буде проведено огляд веб-застосунків для схожих систем менеджменту. Буде проведений аналіз їхніх переваг та недоліків.

Засоби розробки. У даному розділі будуть описані загальні технології розроблюваного веб-застосунку, які були обрані для програмної реалізації.

Опис програмної реалізації. У цьому розділі будуть описані деталі реалізації веб-застосунку, включаючи використані технології та методи програмування. Буде проведено огляд архітектури, функціональних можливостей для користувачів по ролям, наведено основні діаграми та опис реалізації окремих частин застосунку.

Робота користувача з програмною системою. У цьому розділі буде описано вимоги для встановлення, запуску системи. Також буде наведено ілюстрації роботи з системою та основна інструкція для користування.

Висновки. У заключному розділі будуть сформульовані висновки щодо результатів роботи, надані рекомендації щодо подальшого вдосконалення веб-застосунку та визначені перспективи подальших досліджень у даній області.

1 ЗАДАЧА СИСТЕМИ МЕНЕДЖМЕНТУ ТА МОНІТОРНИГУ, АНАЛІЗУ КЛІЄНТСЬКОЇ ВЗАЄМОДІЇ

Вивчення задач системи менеджменту та моніторингу, аналізу клієнтської взаємодії є першим і дуже важливим етапом розробки, оскільки без чіткого розуміння як має виглядати кінцевий продукт неможливо буде отримати задовільний результат. Тому маємо визначити декілька основних пунктів:

- Призначення програмного забезпечення.
- Задачі програмного забезпечення.
- Визначити яка буде вхідна та вихідна інформація.
- Опис підсистем.
- Опис схеми взаємодії програмних модулів.

1.1 Призначення програмного забезпечення та потенційні користувачі

Цей веб-застосунок розробляється для допомоги різним користувачам, що є дрібними e-commerce компаніями, або компаніям, діяльність яких так чи інакше пов’язана з процесом продажу певної продукції.

Потенційні користувачі включають в себе будь-яку особу або підприємство, що займається продажем товарів, та має бажання розширити свою торгівельну площу, за допомогою виставлення своєї продукції на продаж через веб-додаток, також даний застосунок може слугувати гарною базою для більших компаній за обсягом, обігом товарів, так як розробка була спланована шляхом, що дозволяє легко інтегрувати нові потрібні комунікації, інтеграції з різними партнерами, що надають певні послуги, у яких клієнт може бути зацікавленим.

Реалізоване програмне забезпечення повинне надавати весь необхідний функціонал для зручного заповнення, каталогування та категоризації продукції, з метою кращих пошукових результатів, спрощеного процесу придбання продукції та

систематизації статистичних даних на основі останніх продажів. Також повинно бути надано можливість рольового користувацького підходу до роботи з системою.

Можливості перегляду статистики, оновлення, актуалізації даних, моніторингу стану системи та перевірки останніх змін в її роботі.

Весь функціонал повинен не лише задовольняти потреби комерційної діяльності різних компаній, але й бути дружнім до розширення та інтеграцій зі сторонніми партнерами.

1.2 Функціональні вимоги до програмного забезпечення

Ключові задачі програмного забезпечення сформульовані наступним чином:

- Авторизація та реєстрація користувачів у системі з рольовим режимом доступу та функціоналу, відновлення втраченого доступу
- Можливість базової реалізації e-commerce процесів для малого та середнього бізнесу, або інших форм підприємницької діяльності пов’язаної з комерційною діяльністю, а саме:
- Перегляд інформації про прибуток
- Перегляд статистики продажів певного продукту
- Перегляд найбільш прибуткових клієнтів та їх контактної інформації, яка існує в системі
- Сповіщення адміністратора про процеси необхідні до реагування (нові замовлення, оновлення складу товарів)
- Сповіщення клієнта про оновлення інформації по його замовленням
- Можливість файлового експорту даних про наявні замовлення та продукти
- Можливість замовлення продукції, різні варіанти оплати, вказання місця доставки
- Можливість категоризації продукції
- Можливість додавання та оновлення інформації про продукти

- Можливість внесення даних про оплату замовлення адміністратором
- Можливість моніторингу загального стану системи на предмет доступності, останніх подій, статистики з приводу процесу роботи, конфігурації, навантаження та можливість повноцінної взаємодії усіх компонентів один з одним

1.3 Вхідна і вихідна інформація

Вхідна інформація:

Для функціонування частини системи потрібен ряд даних, які заповнюють користувачі залежно від їх ролі.

Дані про продукти, категоризацію, дані для оформлення замовлень, дані для списку очікування, дані користувачів для користування системою.

Вихідна інформація:

Звіти, статистика обігу продуктів та платежів, історія замовлень, список найбільш прибуткових клієнтів, моніторинг стану системи та її доступності у реальному часі.

Вхідні дані умовно діляться на три типи:

- Заповнення можливе лише клієнтом
- Заповнення можливе лише менеджером
- Заповнення доступне для всіх

Клієнт заповнює інформацію, яка необхідна для створення замовлення – обирає продукти, їх кількість, варіант оплати та доставки. Також клієнт може заповнювати інформацію для персонального списку очікування товарів, на випадок, якщо він не має змоги придбати їх на даний момент. Менеджер заповнює інформацію про продукти, категорії продуктів, може вносити дані про оплату замовлення.

Вихідну інформацію формує закладена у програмному забезпеченні логіка

збереження, створення нових даних на основі існуючих, відображення даних збережених у системі на вимогу користувача.

1.4 Опис підсистеми

Цей веб-застосунок включає в себе такі підсистеми:

1) Застосунок визначення інших сервісів.

Призначений для надання моніторингових панелей, збереження та конфігурації режиму доступу до цих даних. Містить конфігурації для обробки підключень інших сервісів. Основна задача, бути отримувачем підключень та збором моніторингових даних підключених мікросервісів, з можливістю перегляду, а також налаштуванням сценаріїв у випадку зміни внутрішнього стану мікросервісного застосунку [1] та інших використаних технологій

2) Застосунок визначення конфігурацій.

Призначений для отримання внутрішніх налаштувань, таких як підключення до бази даних, назва застосунку, підключення до кешу, налаштування двигуна міграцій, підключення до моніторингового сервісу, що збирає метрики та інші аналітичні дані, конфігурація фреймворку для роботи з базою даних, налаштування безпеки, тощо

3) Застосунок, що володіє бізнес-логікою для обробки користувачів.

Даний сервіс призначений для реалізації функціоналу необхідного для збереження даних про усіх існуючих користувачів, та також оновлення та отримання цих відомостей, проведення авторизації результатом якої є JWT [2] – спеціальний токен генерація якого проходить усередині власноруч створеної інфраструктурної бібліотеки, з метою використання даного аутентифікаційного підходу при роботі з рештою сервісів, незважаючи на те, що авторизація була проведена лише з залученням одного

4) Застосунок для роботи з замовленнями.

Даний сервіс надає весь необхідний функціонал для збереження та обробки

інформації про категорії продуктів, продукти, замовлення користувачів, платежі проведені у системі, зберігає, формує та передає статистику про продажі, найбільш прибуткових та вигідних до співпраці клієнтів.

Реалізує корисні інтеграції, а саме: інтеграція з платіжною системою DataTrans [3], інтеграція з сервісом доставки, сервісом надання актуальних курсів валют для конвертації платіжної інформації для розділу статистики. Виступає основним сервісом з точки зору бізнес операцій, навколо якого побудовані решта сервісів

5) Застосунок для роботи з повідомленнями.

Даний сервіс реалізує інтеграцію з поштовим API NovaPost [4], для відправки електронних листів з використанням Gmail [5], також підтримує роботу з Telegram Bot API [6], для відправки повідомлень користувачам цієї платформи.

Сервіс є уніфікованим, тобто може бути використаний для відправки будь-яких повідомлень, на будь-які, коректні з точки зору вимог зі сторони інтеграції, адреси.

6) Застосунок верифікації дій користувача.

Даний сервіс необхідний для реалізації складних процесів верифікації, а саме тих, для яких необхідно забезпечення збереження стану незалежно від можливих збоїв, чи інших проблем, які можуть ускладнити цей процес.

Отримує запити від інших сервісів через брокер повідомлень, будує необхідні моделі, в яких визначена інформація про тип дії, наприклад відновлення паролю або підтвердження реєстрації користувача, платформа для надсилання, код підтвердження, чай його дії, повідомлення певного формату в залежності від платформи, якщо вона підтримує спеціалізоване форматування. Виконує підтвердження верифікації, забезпечує унікальність, узгодженість.

Сервіс уніфіковано, тип операцій та решта деталей визначається лише фантазією та вимогами клієнта.

7) Застосунок надання користувацького інтерфейсу

Даний застосунок необхідний для відображення графічних елементів, використання яких надає візуалізацію усього функціоналу серверної частини.

Спеціальні сторінки та форми для авторизації, реєстрації, відновлення доступу до акаунту. Перегляд категорій, продуктів, фільтрація їх на основі введених даних, оформлення замовлення, перегляду деталізованої інформації про конкретний продукт, графічні моделі кошику покупок, тощо

Візуалізація форм та списків, графіків, для менеджмент процесів, можливість переадресації між сторінками, надсилання та отримання інформації від серверної частини, збереження тимчасових даних на клієнтській стороні, тобто браузері.

1.5 Схема взаємодії програмних модулів

Загальну архітектуру програмного забезпечення можна описати як клієнт-серверний застосунок [7], з мікросервісною структурою [8] сервісної частини (рис. 1.5.1).

Всього існує 6 повноцінних застосунків, 4 з них напряму пов’язані з наданням функціоналу для реалізації бізнес-логіки, решта службові, необхідні для підтримання загальних патернів та концепцій у реалізації застосунків з такою архітектурою для обраних технологій.

Структурна схема застосунку реалізує принцип одного сховища для сервісу, це значить, що кожен з них має персональне сховище даних – база даних або кеш, якщо цим сервісом передбачене збереження певних даних. Такий підхід забезпечує високу незалежність та ізольованість сервісів один від одного, що підвищує гнучкість системи, спрощує її масштабування і обслуговування, а також покращує безпеку даних, оскільки кожен сервіс відповідає тільки за свої власні дані. Цей підхід також сприяє дотриманню принципів мікросервісної архітектури, забезпечуючи незалежний розвиток і розгортання кожного компонента системи. Крім того, завдяки цьому підходу можна легше інтегрувати нові технології та компоненти, не впливаючи на загальну функціональність системи, що є важливою перевагою в умовах швидкого розвитку технологій і змінюваних вимог бізнесу.

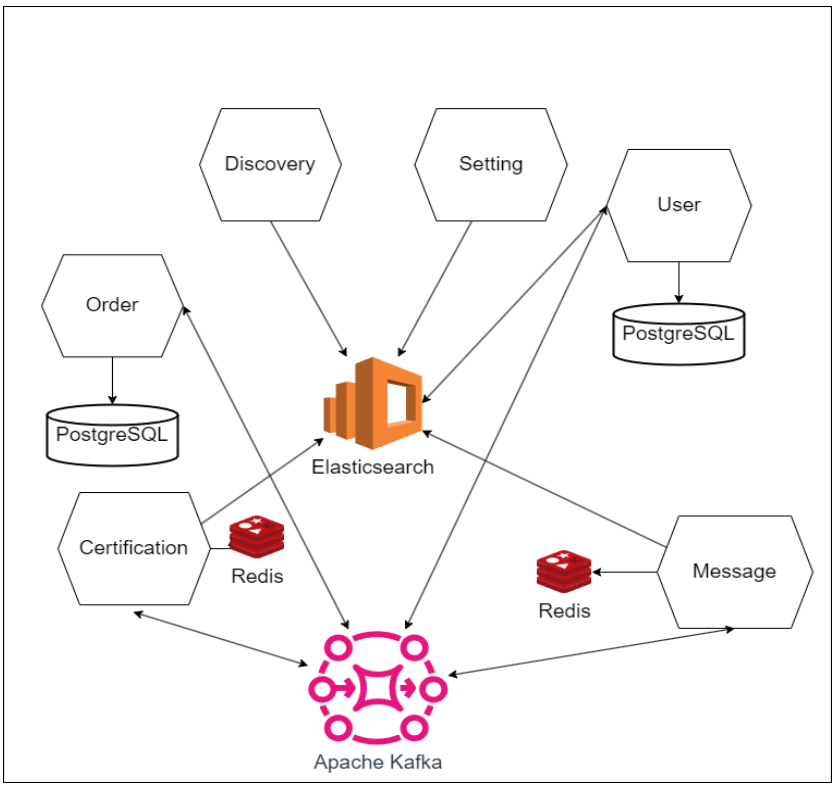

Рисунок 1.5.1 - структурна схема взаємодії серверної частини

Кожен з бізнес сервісів взаємодіє з брокером повідомлень на основі AMQP [9] протоколу, з залученням Apache Kafka. Комунікація відбувається за принципом черг, усі повідомлення які отримує брокер, потрапляють у конкретний канал повідомлень по його унікальній назві, де в свою чергу брокер надсилає ці повідомлення усім приймачам, таким чином принцип передачі повідомлень кінцевому отримувачу відбувається за умови його наявності. Це вирішує проблему з узгодженістю та надійністю системи, брокер не зможе надіслати повідомлення неіснуючому отримувачу, також реалізована поведінка самоочищення, якщо приймачі відсутні занадто довго з метою звільнення черг, кожне повідомлення має персональний час життя на стороні Apache Kafka [10].

Кожен з сервісів, пов’язаний з логікою функціоналу має певне сховище даних, сервіси для роботи з користувачами та комерційними процесами зберігають усі дані до баз даних, так як інформація з якою вони працюють потрібна постійно і має бути збережена та доступна у максимально надійній формі.

Уніфіковані службові сервіси не мають жорсткої прив’язаності до конкретних даних, вони використовують виключно кеш на основі Redis [11], так як дані, якими ці сервіси володіють та оброблюють не є необхідними до довготривалого зберігання, часто змінюються і читаються.

Для додаткового аналізу та моніторингу обробки певних подій, кожен з сервісів підключено до централізованої зовнішньої системи логування в основі якої є Elasticsearch [12] – вільне програмне забезпечення створене для пошуку та індексації різних документів, за своєю основою найбільш наближений до NoSQL [13] - сховищ.

Принцип взаємодії серверної та клієнтської частини базується на клієнт-серверній архітектурі (рис.1.5.2)

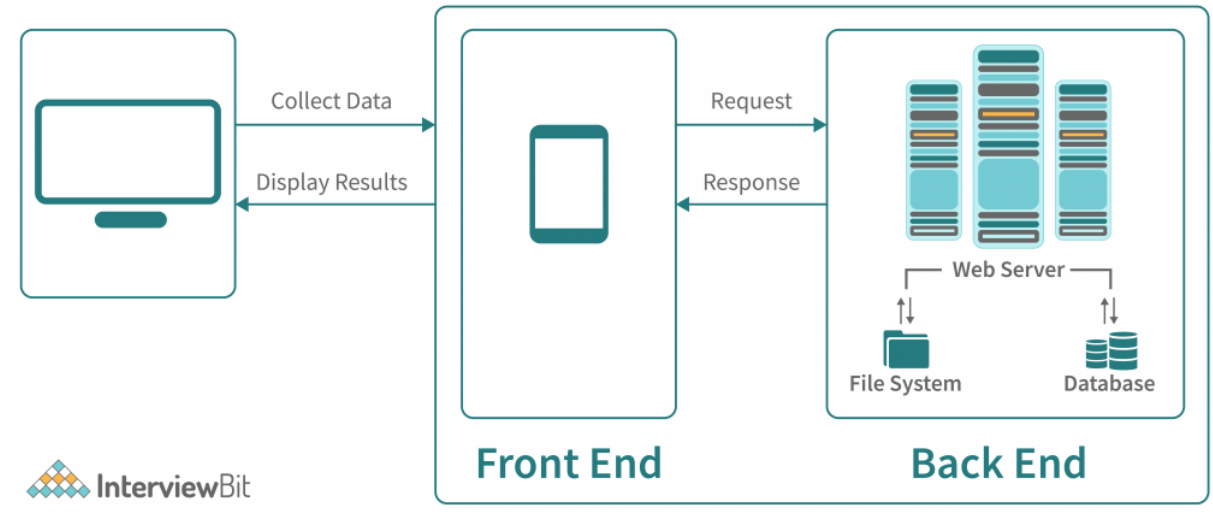

Рисунок 1.5.2 – схема роботи веб-додатку [7]

Для даного програмного забезпечення клієнтська частина це застосунок в основі якого React–сучасна бібліотека для веб-розробки.

Комунікація між клієнтською реалізацією та серверною відбувається на основі HTTP-протоколу, в поодиноких випадках також залучено використання WebSocket [14] для передачі даних від сервера в односторонньому форматі

2 АНАЛІЗ ПРОБЛЕМИ ЕЛЕКТРОННОЇ КОМЕРЦІЙНОЇ ДІЯЛЬНОСТІ

Аналіз проблем системи менеджменту та моніторингу, аналізу клієнтської взаємодії є першим і дуже важливим етапом розробки, оскільки без порівняння та оцінки існуючих рішень отримати задовільний результат буде тяжко. Тому маємо визначити декілька основних пунктів:

- Загальний огляд предметної області
- Огляд існуючих програмних систем
- Аналіз функціональних можливостей та особливостей

2.1 Загальний огляд предметної області

Сучасна електронна комерція стала неодмінною частиною бізнесу в цифрову еру. Її зростання в останні десятиліття дало змогу компаніям здійснювати торгівлю та здійснювати комунікацію з клієнтами через Інтернет. Однак разом зі зростанням можливостей електронної комерції з'явилися й нові виклики та проблеми, які потребують аналізу та вирішення.

Проблеми електронної комерції можуть бути різноманітні і виникати на кожному етапі процесу від замовлення товару до його доставки та обслуговування після продажу. Деякі з найпоширеніших проблем включають:

1. Безпека: Однією з найбільших проблем електронної комерції є безпека транзакцій та конфіденційність особистих даних клієнтів. Крадіжки ідентичності, кібератаки та шахрайство - серйозні загрози для електронних магазинів та їх клієнтів.

2. Інтеграція з системами: Часто компанії стикаються з проблемами інтеграції своїх електронних магазинів з іншими бізнес-системами, такими як системи управління запасами, бухгалтерські програми та CRM-системи [15], служби

доставки, онлайн-оплати, тощо.

3. Конкуренція: Зростання кількості електронних магазинів призводить до посилення конкуренції, що може зробити складнішими зусилля з приваблення та утримання клієнтів.

4. Доставка: Ефективна організація доставки товарів є ключовою складовою успіху електронного бізнесу. Проблеми з доставкою можуть включати затримки, втрату посилок та проблеми зі співпрацею з службами доставки придбаних товарів.

5. Обслуговування клієнтів: Забезпечення якісного обслуговування до та після продажу може бути важкою задачею в електронній комерції, особливо коли мова йде про вирішення проблем з продуктами або повернення товарів.

Аналіз цих проблем, а також розробка стратегій їх вирішення, є важливою частиною успішної електронної комерційної діяльності.

2.2 Огляд існуючих програмних систем

Наразі існує велика кількість програмних систем, які призначені для полегшення та покращення електронної комерційної діяльності [16]. Ось загальний порівняльний огляд деяких з найпопулярніших систем:

1. Magento – це одна з найпопулярніших платформ електронної комерції, яка надає широкі можливості для створення та управління онлайн-магазинами.

Плюси:

- Велика кількість функцій та можливостей для налаштування.
- Гнучкість у розширеннях та інтеграціях.
- Підтримка магазинів будь-якого розміру та складності.

Мінуси:

- Вимагає достатньо великих знань для налаштування та управління.
- Високі витрати на розробку та підтримку.

2. Shopify – це хмарна платформа для створення та управління електронними магазинами, яка відома своєю простотою та широким вибором шаблонів та додатків.

Плюси:

- Простий у використанні та швидкий у налаштуванні.
- Великий вибір шаблонів та додатків.
- Хороша підтримка та безпека.

Мінуси:

- Обмежені можливості розширення в порівнянні з Magento.
- Місячні платежі за використання.

3. WooCommerce – це розширення для WordPress, яке перетворює його в повнофункціональний онлайн-магазин і відоме своєю інтеграцією з екосистемою WordPress.

Плюси:

- Інтеграція з WordPress та його екосистемою.
- Безкоштовний основний функціонал.
- Широкий вибір розширень та тем.

Мінуси:

- Потребує наявності веб-сайту на WordPress.
- Завантаженість сервера може вплинути на продуктивність.

4. BigCommerce – це хмарна платформа електронної комерції, яка надає широкий спектр функцій для створення та управління магазинами будь-якого розміру.

Плюси:

- Широкий функціонал, включаючи маркетинг та аналітику.
- Швидке розгортання магазину та підтримка.
- Інтеграція зі сторонніми сервісами.

Мінуси:

- Обмежені можливості налаштування в порівнянні з Magento.
- Вартість підписки може бути високою для малих бізнесів

5. OpenCart – це відкрита система управління контентом для електронної комерції, яка надає можливості для створення і налаштування онлайн-магазинів.

Плюси:

- Відкритий код та безкоштовний доступ.
- Простий інтерфейс та легка установка.
- Широкий вибір розширень.

Мінуси:

- Менш розвинуті можливості порівняно з Magento або Shopify.
- Може вимагати більше часу для налаштування та підтримки.

Кожна з цих систем має свої унікальні переваги та недоліки, тому вибір залежить від конкретних потреб та обмежень цільового бізнесу.

2.3 Аналіз функціональних можливостей та особливостей

Наступним розглянемо аналіз функціональних можливостей та особливостей кожної з програмних систем електронної комерції, які були надані:

1. Magento

Функціональні можливості:

- Розширені засоби налаштування товарів, категорій та промоцій.
- Можливість створення різних типів товарів та варіацій.
- Широкі можливості управління замовленнями та клієнтськими обліковими записами.
- Потужні інструменти аналітики та звітності.

Особливості:

- Велика кількість розширень та тем для розширення функціоналу.
- Можливість інтеграції з різними системами платежів та доставки

2. Shopify

Функціональні можливості:

- Простий у використанні інтерфейс для створення та налаштування магазину.
- Інструменти маркетингу та SEO для залучення клієнтів.
- Швидке розгортання та безпека за замовчуванням.

Особливості:

- Можливість створення магазину без необхідності знання коду.
- Велика кількість додатків для розширення функціоналу магазину

3. WooCommerce

Функціональні можливості:

- Інтеграція з WordPress, що дозволяє використовувати всі можливості платформи.
- Гнучке управління та налаштуванням товарів і категорій.
- Можливість використання різних методів оплати та доставки.

Особливості:

- Безкоштовна основна версія з можливістю розширення функціоналу за допомогою різноманітних розширень.
- Велика спільнота користувачів та розробників

4. BigCommerce

Функціональні можливості:

- Широкий набір інструментів для створення, редагування та керування каталогом товарів.
- Вбудовані засоби маркетингу, такі як знижки, промокоди та програми лояльності.
- Аналітика та звітність для відстеження продажів та поведінки покупців.

Особливості:

- Швидке розгортання магазину та підтримка інтеграції з різними платіжними системами та доставкою.
- Готові шаблони для швидкого старту та можливість налаштування дизайну.

5. OpenCart

Функціональні можливості:

- Управління каталогом товарів та категоріями зі зручним інтерфейсом.
- Різноманітні методи оплати та доставки, що включаються за замовчуванням.
- Модульна система для розширення функціоналу магазину.

Особливості:

- Відкритий код, що дозволяє розробникам вносити будь-які зміни та

розширювати можливості.

- Наявність великої кількості безкоштовних та платних розширень для розширення функціоналу магазину.

Кожна з цих систем має свої унікальні функціональні можливості та особливості, що можуть впливати на їх вибір для конкретного бізнесу

Однак вибір оптимальної платформи має враховувати як сильні, так і слабкі сторони кожного рішення

У зв'язку з розробкою застосунку електронної комерції ключовим аспектом стане здатність системи ефективно взаємодіяти з користувачами, забезпечуючи зручний та безпроблемний процес покупок та оплати. Основною ідеєю розробленого застосунку буде підтримка та стимулювання користувачів у здійсненні покупок, надання їм зручного інтерфейсу та високоякісного обслуговування.

Отже, розроблений застосунок електронної комерції відзначатиметься своєю спрямованістю на максимальне задоволення потреб клієнтів, забезпечуючи їм зручний та простий процес покупок, а також підтримуючи розвиток бізнесу через стимулювання продажів та збільшення лояльності клієнтів. Основними перевагами такого застосунку стануть інтуїтивно зрозумілий інтерфейс, можливість легкої інтеграції з існуючими системами управління бізнесом та надання широкого спектра функціональних можливостей для покращення взаємодії з клієнтами. Це дозволить не лише підвищити ефективність роботи підприємства, але й забезпечити стабільне зростання клієнтської бази та підтримку високого рівня задоволеності споживачів. Зокрема, застосунок забезпечить гнучке налаштування та управління товарним каталогом, інтеграцію з популярними платіжними системами, а також можливість використання інструментів аналітики для оптимізації бізнес-процесів.

3 ЗАСОБИ РОЗРОБКИ

У цьому розділі досліджено використані інструменти розробки, програмне забезпечення, мови програмування, а також використані бібліотеки та фреймворки. Розглядаючи засоби розробки, їх можна логічно поділити на категорії відповідно до виконуваних завдань. Зокрема, розглянемо наступні компоненти веб-застосунку, які необхідно буде розробити:

- Візуальна частина
- Серверна частина
- База даних та кеш

Також необхідно буде обрати базу даних та інші системи збереження інформації за потреби, додаткові елементи, наприклад брокери повідомлень, різні технології необхідні для повноцінної реалізації застосунку.

Для реалізації кожної з цих складових необхідно обрати та використовувати відповідні інструменти та мови програмування.

3.1 Візуальна частина

Створене програмне забезпечення є веб-застосунком, який реалізує собою клієнт-серверний варіант взаємодії між частинами.

Для реалізації інтерфейс частини було використано мову гіпертекстової розмітки HTML [17], стилізацію для надання певного візуального відображення для кращого сприйняття користувачем засобами CSS [18], процеси ініціації та обробки динамічного функціоналу, а саме взаємодія з сервером, збереження даних на клієнтській стороні, робота з запитами, тощо – засобами мови програмування JavaScript, для спрощення процесу розробки також було залучено бібліотеку React [19], яка дозволяє використовувати динамічне оновлення сторінки, процес передачі та відображення даних, а також компонентний підхід, що дозволяє повторно

використовувати раніше написану розмітку та логіку в різних частинах програми.

Для розробки інтерфейсу було обрано базові технології властиві веб-застосункам, а саме HTML – гіпертекстова мова розмітки, що є стандартом для перегляду сторінок у веб-браузері, CSS – каскадна таблиця стилів яка описує зовнішній вигляд сторінки, для реалізації динамічних процесів було залучено мову програмування JavaScript [20]

JavaScript - це високорівнева, інтерпретована мова програмування, що використовується для розробки веб-додатків та додатків на стороні клієнтського браузера. Одна з основних мов програмування для створення інтерактивних елементів на веб-сторінках. JavaScript спрощує взаємодію з HTML і CSS, дозволяючи змінювати їх вміст, стилі, інтерактивність та багато іншого.

Основні особливості JavaScript:

1. Динамічність: JavaScript є мовою програмування з динамічною типізацією, що дозволяє змінювати типи даних залежно від контексту.

2. Клієнтська сторона: JavaScript використовується для створення веб-додатків та веб-сайтів, які виконуються в браузері клієнта.

3. Асинхронний код: JavaScript підтримує асинхронне програмування, що дозволяє виконувати операції без блокування інтерфейсу користувача.

4. Крос-платформенність: JavaScript може виконуватися на будь-якому пристрої з веб-браузером, незалежно від операційної системи.

5. Широке застосування: JavaScript використовується не лише на веб-сторінках, а й у серверному програмуванні, мобільних додатках та навіть для розробки настільних програм.

JavaScript є однією з найпопулярніших мов програмування і використовується мільйонами розробників по всьому світу для створення різноманітних веб-додатків та сервісів.

Також для спрощення процесу розробки було залучено спеціалізовану бібліотеку React.

React - це бібліотека JavaScript для створення інтерфейсів користувача, яка розроблялася компанією Facebook. Основна ідея React полягає в тому, щоб

розбивати інтерфейс на невеликі компоненти, якими зручно керувати. React використовує концепцію віртуального DOM, що дозволяє оптимізувати оновлення інтерфейсу, шляхом мінімізації кількості дій, які потрібно здійснити для відображення змін.

Ось деякі переваги використання React:

1) Компонентний підхід: React базується на розбитті інтерфейсу на невеликі компоненти, що спрощує розробку та підтримку коду.

2) Віртуальний DOM: React використовує віртуальний DOM для ефективного оновлення інтерфейсу, що дозволяє підтримувати високу продуктивність навіть за наявності великої кількості компонентів.

3) Односторінкові додатки: React чудово підходить для розробки односторінкових додатків (SPA), де зміни на сторінці відбуваються без перезавантаження всієї сторінки, проте є можливість використання різної маршрутизації за необхідності.

React Router – це бібліотека для React, яка надає можливість створювати односторінкові додатки з різними маршрутами та відображенням вмісту на відповідних шляхах URL. Вона дозволяє організовувати навігацію між сторінками та визначати, який компонент повинен бути відображений для кожного URL-шляху.

4) Реактивність: React дозволяє створювати реактивні та інтерактивні інтерфейси, що реагують на дії користувачів без зайвого перероблення DOM.

5) Широке співтовариство та екосистема: React має велике співтовариство розробників і багато сторонніх бібліотек і фреймворків, що допомагають в розробці.

У цілому, використання React може значно спростити процес розробки веб-додатків, забезпечуючи швидку, ефективну та модульну структуру коду

3.2 Серверна частина

Серверна частина застосунку реалізована з залученням мікросервісної архітектури для побудови інфраструктури, а саме набором різних сервісів, які як

доступні для взаємодії напряму з клієнтської сторони через протоколи обміну даними HTTP, та у виключних ситуаціях, коли необхідно отримати інформацію від сервера по факту її появи, WebSocket, так і з інших сервісів з метою реалізації комплексної логіки, використовуючи брокер повідомлень на основі протоколу асинхронного обміну повідомленнями AMQP.

Для розробки цієї частини було використано мову програмування Java, одна з найбільш свіжих та стабільних версій, яка дозволяє інтегруватися з фреймворками різного характеру призначення.

В основі реалізації веб-архітектури закладено використання Spring Framework та Spring Boot [21] (рис. 3.2.1)

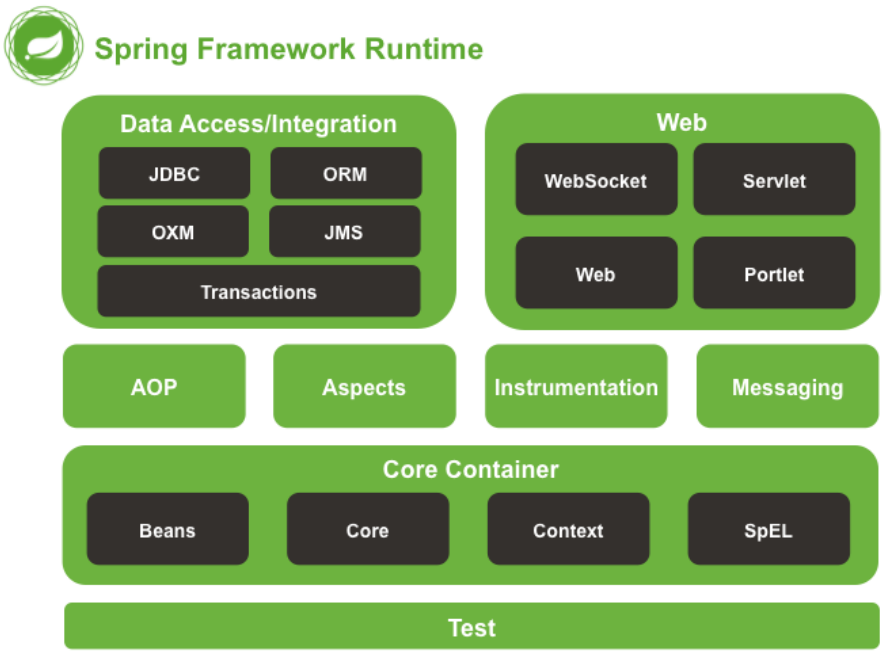

Рисунок 3.2.1 – Основа структури та наявні модулі Spring Framework

Використання цих технологій для мікросервісного веб-додатка має кілька причин.

1. Масштабованість

Java має довгу історію в розробці підприємницьких додатків та має потужні

інструменти для масштабування. Spring Framework надає широкий спектр інструментів та бібліотек для розробки веб-додатків, що полегшує масштабування мікросервісів.

2. Легкість розробки

Spring Framework забезпечує простий та ефективний спосіб розробки мікросервісів. Він має багато готових модулів, які спрощують роботу з базами даних, веб-службами, безпекою та іншими аспектами розробки.

3. Широкий спектр інструментів

Екосистема Spring має багато інструментів, таких як Spring Boot для швидкого створення мікросервісів, Spring Cloud [22] для управління конфігурацією, балансуванням навантаження та іншими аспектами, що допомагають у побудові розподілених систем.

4. Багатофункціональність

Java має багатий набір бібліотек та фреймворків, що сприяє розробці різноманітних функцій у мікросервісах, таких як обробка розподілених транзакцій, кешування, робота з різними форматами даних тощо.

5. Надійність та стабільність

Java відома своєю надійністю та стабільністю. Використання Java та Spring може забезпечити високу якість та стабільність мікросервісів, що є критичним для підприємницьких додатків.

Загалом, використання Java та Spring для мікросервісного веб-додатка може спростити розробку, забезпечити широкі можливості функціональності та забезпечити високу надійність та масштабованість.

Фреймворк Spring відомий своєю модульністю, що дозволяє використовувати лише необхідні компоненти для конкретного проекту та легко розширювати функціональність за допомогою різних модулів. Розглянемо детальніше, які модулі були задіяні для створення бекенд-частини застосунку:

1. Spring Core

Це основний модуль, який надає базові функціональність інверсії управління та інтеграції залежностей (IoC/DI), а також інші ключові можливості, такі як

конфігурація, обробка подій та інші.

2. Spring AOP (Aspect-Oriented Programming)

Цей модуль використовується для аспектно-орієнтованого програмування, що дозволяє розділяти логіку додатку на окремі аспекти, такі як логування, транзакції тощо, і використовувати їх повторно.

3. Spring Data

Цей модуль спрощує роботу з реляційними та нереляційними базами даних, надаючи різноманітні інтерфейси та шаблони для взаємодії з базою даних та кешом.

4. Spring Security

Модуль для забезпечення безпеки веб-додатків, включаючи аутентифікацію, авторизацію та керування доступом до ресурсів.

5. Spring Messaging

Цей модуль дозволяє створювати рішення для обміну повідомленнями на основі різних протоколів, включаючи WebSocket, STOMP тощо.

6. Spring Aspects

Даний модуль дозволяє використовувати аспекти для внесення змін в поведінку додатку, такі як виконання додаткових операцій перед або після виклику методу, тісно пов’язаний з використанням АОП, та є його сабмодулем, який вже містить в собі певні готові рішення та звільняє розробника від необхідності самостійної реалізації.

7. Spring Web

Цей модуль містить інструменти для створення веб-додатків, включаючи підтримку контролерів, обробників запитів, обробку шаблонів тощо.

Це перелік ключових модулів Spring Framework, які були задіяні в розробці цієї частини застосунку. Кожен з цих модулів надає різноманітну функціональність та допомагає забезпечити стабільність, безпеку та ефективність роботи застосунку.

Окрім використання Spring, залучені також інші бібліотеки та фреймворки для роботи з базою даних з залученням ORM підходу, у тому числі Hibernate [23], обробки даних різних форматів JSON, XML, їх парсингу та запису.

Бібліотеки для роботи з конкретною базою даних – PostgreSQL [24],

підтримкою міграцій – контроль версій бази даних, можливість програмної вимоги відповідності певній структурі, а саме наявності конкретних таблиць, атрибутів, посилань, схем, тощо.

Модуль для роботи з AMQP брокером, зовнішньою системою збереження та відображення записів логування, бібліотека для роботи з JWT та певні внутрішні бібліотеки залучені для спрощення процесу розробки.

Також використані технології пов’язані з вибором архітектури застосунку, а саме Spring Cloud, Netflix Eureka [25] та Spring Boot Admin, що забезпечує можливість централізації усіх окремих сервісів, підтримка процесу взаємного виявлення, візуалізації даних про стан системи, централізоване збереження та видача налаштувань для кожного сервісу, що є необхідним для зручної та надійної роботи з мікросервісами [26]

Окрім цього, для уникнення повторення конфігурацій, моделей, можливості використання спільних налаштувань, які властиві кожному сервісу інтегрованому у бізнес-логіку, було розроблено застосунок, який виступає у якості бібліотеки для імпорту, та опубліковано на платформі GitHub.

Для забезпечення процесу старту, компіляції та побудови окремого проекту, а також імпорту раніше перелічених модулів та бібліотек різних фреймворків, використано засіб автоматизації роботи зі складними проектами Maven.

Кожен з проектів та інших самостійних одиниць, таких як брокер, чи система зовнішнього збереження записів логування, реалізовано у тому числі для запуску у контейнеризованій формі з залученням системи управління UNIX-контейнерами, Docker [27].

Використання контейнерів дозволить легко запустити та провести публікацію проекту на сервер, доступний користувачам, без необхідності профільного налаштування, так як усі сучасні хмарі рішення підтримують цю технологію.

3.3 База даних та кеш

Для повноцінного функціонування, збереження потрібних даних необхідно визначитися з базою даних.

Розглянемо можливі варіанти за відношенням даних у базі, існує декілька варіантів:

- SQL
- NoSQL

SQL (реляційні бази даних):

Використовують реляційну модель для організації даних у вигляді таблиць зі зв'язками між ними. Вони надійні, стабільні та добре підходять для схематизованих даних з чітко визначеними відносинами

NoSQL (нереляційні бази даних):

Використовуються для роботи з неструктурованими або півструктурованими даними, які можуть змінюватися з часом.

NoSQL бази даних, такі як MongoDB або Cassandra, мають високу масштабованість та гнучкість, що робить їх ідеальними для сучасних, розподілених систем.

У контексті обраної предметної області краще звернути увагу на реляційну модель даних, так як дані чітко пов’язані між собою, у деяких випадках залежні від існування інших і вимагають посилання на них.

Ось декілька причин, чому для e-commerce додатку з мікросервісною архітектурою, може бути кращим вибором SQL база даних:

1. Структурованість даних

У сфері електронної комерції важливо мати чітку структуру даних для зберігання інформації про продукти, замовлення, клієнтів тощо. SQL бази даних надають реляційну модель, яка дозволяє легко організувати дані в таблицях з взаємозв'язками між ними.

2. Складні запити

Ваш e-commerce додаток, ймовірно, буде потребувати складних запитів до

бази даних для аналізу даних, створення звітів, рекомендаційних систем тощо. SQL бази даних забезпечують потужність SQL мови для виконання таких запитів.

3. Індекси та референси у SQL базі даних

Покращення продуктивності запитів

Індекси дозволяють ефективно шукати та фільтрувати дані, що особливо важливо для електронної комерції з великим обсягом інформації про товари та замовлення.

Забезпечення цілісності даних: Використання зовнішніх ключів допомагає у підтримці зв'язків між таблицями та у встановленні правил для збереження цілісності даних, що є ключовим аспектом електронної комерції.

В контексті NoSQL баз даних, такі можливості, як індекси та референси, зазвичай відсутні. Це може зробити SQL базу даних більш привабливою для використання в електронній комерції, де забезпечення продуктивності та цілісності даних є критично важливими аспектами

4. Транзакційна підтримка

В електронній комерції важливо мати підтримку транзакцій для забезпечення консистентності та цілісності даних, особливо під час операцій замовлення та оплати. SQL бази даних надають сильну підтримку ACID (Atomicity, Consistency, Isolation, Durability) для керування транзакціями

5. Стабільність та надійність

Більшість SQL баз даних, зокрема такі як PostgreSQL, MySQL або MariaDB, відомі своєю стабільністю та надійністю. Для електронної комерції, де велика кількість транзакцій та важлива доступність системи, це дуже важливо

6. Масштабованість

Хоча NoSQL бази даних часто відомі своєю масштабованістю, багато SQL баз даних також мають механізми для горизонтального масштабування, такі як реплікація та кластеризація, що дозволяє їм працювати з великими обсягами даних.

Зважаючи на ці фактори, SQL база даних, зокрема PostgreSQL, може бути оптимальним вибором для e-commerce додатку з мікросервісною архітектурою.

Порівняння SQL баз даних:

- PostgreSQL: Має широкий набір функціональності та гнучкість, ідеальний для складних проектів з великим обсягом даних.

- MySQL/MariaDB: Простіший та легший у використанні, підходить для менших проектів або прототипування.

- SQLite: Вбудована база даних, хороша для розробки та тестування.Порівняння кеш-систем:

- Redis: Швидкий, масштабований та з великим набором функцій, ідеальний для кешування та роботи з розподіленими системами.

- Memcached: Простий та швидкий, використовується головним чином для кешування проміжних результатів та зменшення навантаження на сервер.

- Ehcache: Використовується для локального кешування на рівні додатка, менш підходить для роботи з розподіленими системами.

Щоб вибрати оптимальний варіант, слід врахувати вимоги проекту щодо продуктивності, масштабованості та надійності. У даному випадку мова йде про потенційно масштабовані та навантажені e-commerce додатки, тому PostgreSQL та Redis є оптимальними виборами. Крім того, варто звернути увагу на можливості інтеграції та підтримки цих технологій. PostgreSQL забезпечує високу продуктивність і надійність завдяки потужному механізму транзакцій і можливості горизонтального масштабування через шардинг та реплікацію. Redis пропонує надзвичайно швидкий доступ до даних, що є критично важливим для обробки великої кількості одночасних запитів у реальному часі. Разом ці технології створюють міцну основу для побудови стійких, швидких і масштабованих e-commerce додатків, здатних ефективно обробляти високі навантаження та забезпечувати безперебійну роботу системи механізмом кешування даних та підтримки розподіленого середовища. Elasticsearch та Kibana надають засоби для потужного пошуку, аналізу та візуалізації даних, що дозволяє ефективно моніторити та аналізувати роботу системи. Docker та Docker Compose [28] дозволяють легко створювати, розгортати та керувати контейнеризованими додатками, що спрощує розробку та розгортання складних систем.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В цьому розділі описано програмну реалізацію розробленого веб-застосунку, що включає в себе огляд наступних пунктів:

- Діаграми
- Огляд створення інтерфейсу
- Огляд створення сервісів
- Огляд бази даних

4.1 Діаграми

Першочергово варто описати функціонал користувача, додаток передбачає рольовий підхід при розподілі функціоналу. Застосунок визнає користувачів двох ролей – клієнт, та адміністратор. Будь-який користувач системи повинен мати змогу авторизуватися у системі та відновити пароль для доступу до акаунту у випадку втрати

Клієнт повинен мати змогу:

1) Зареєструвати акаунт
2) Переглядати доступні категорії продуктів та їх продукти
3) Придбати продукти – оформити замовлення
4) Мати можливість провести оплату відразу під час оформлення замовлення
5) Переглядати свої замовлення та їх історію змін
6) Експортувати дані про замовлення чи історію до файлу, наприклад для звітності
7) Можливість додавати та видаляти продукти зі списку очікування

Менеджер повинен мати змогу:

1) Мати доступ до списку зареєстрованих користувачів та створення нових

2) Можливість створення, редагування та вимкнення чи увімкнення категорій

3) Можливість створення, редагування та блокування чи деблокування продуктів

4) Переглядати та створювати вручну дані про проведені платіжні операції

5) Мати доступ до статистики продажу кожного продукту

6) Мати доступ до статистики виручки

7) Мати доступ до списку найприбутковіших користувачів та їх контактної інформації

8) Мати доступ до метрик, моніторингових даних та записів логування системи

Для опису функціоналу варто використати побудову різних діаграм [29]

Опишемо функціонал кожної ролі за допомогою use-case діаграми (рис. 4.4.1)

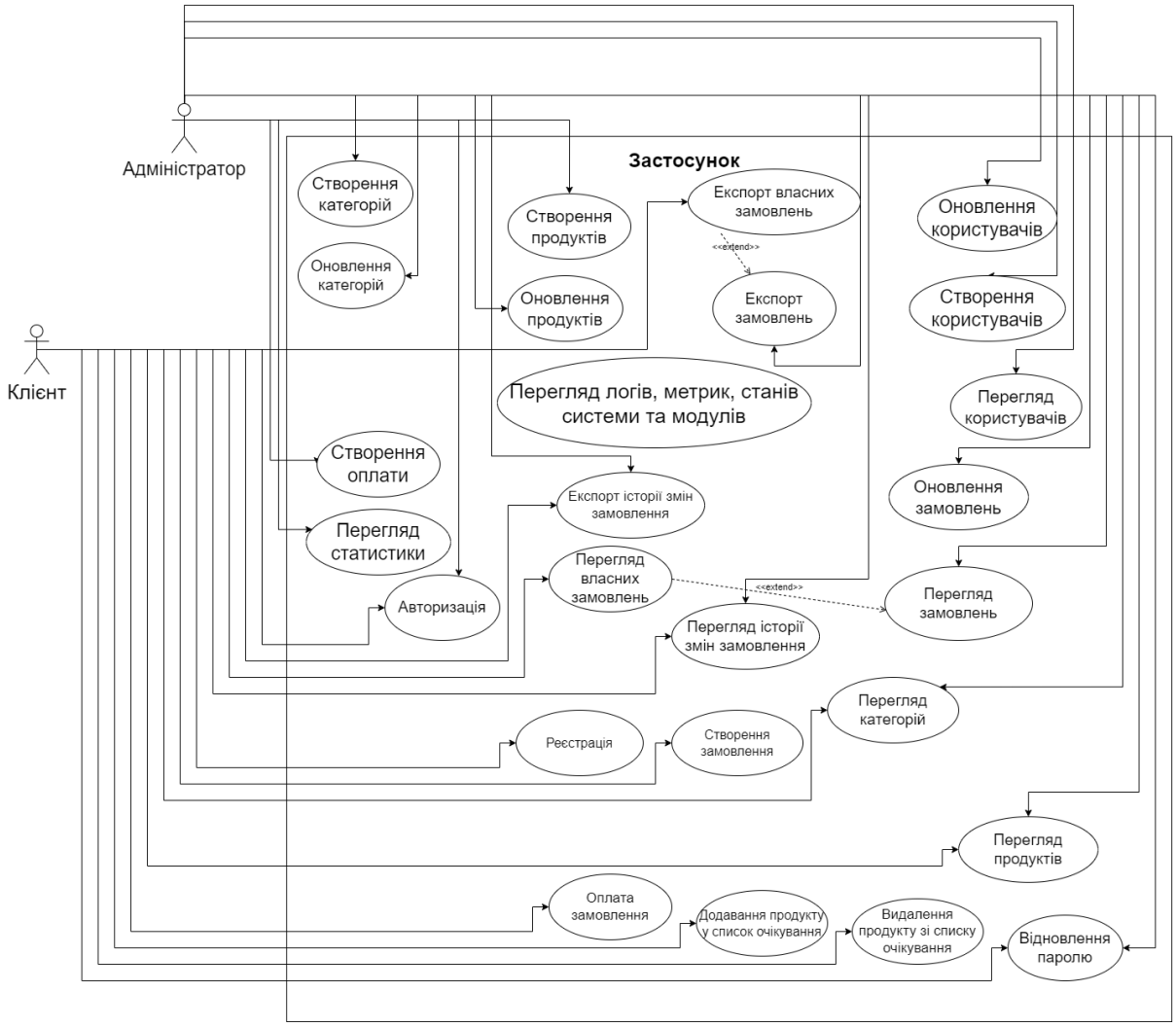

Рисунок 4.1.1 – Use - case діаграма застосунку

Для програмної реалізації використаємо діаграми класів, розглянемо основні частини застосунку:

- робота з замовленнями (рис. 4.4.2)
- робота з категоріями (рис. 4.4.3)
- робота з адресами (рис. 4.4.4)
- робота з продукцією (рис. 4.4.5)
- робота з транзакціями (рис. 4.4.6)

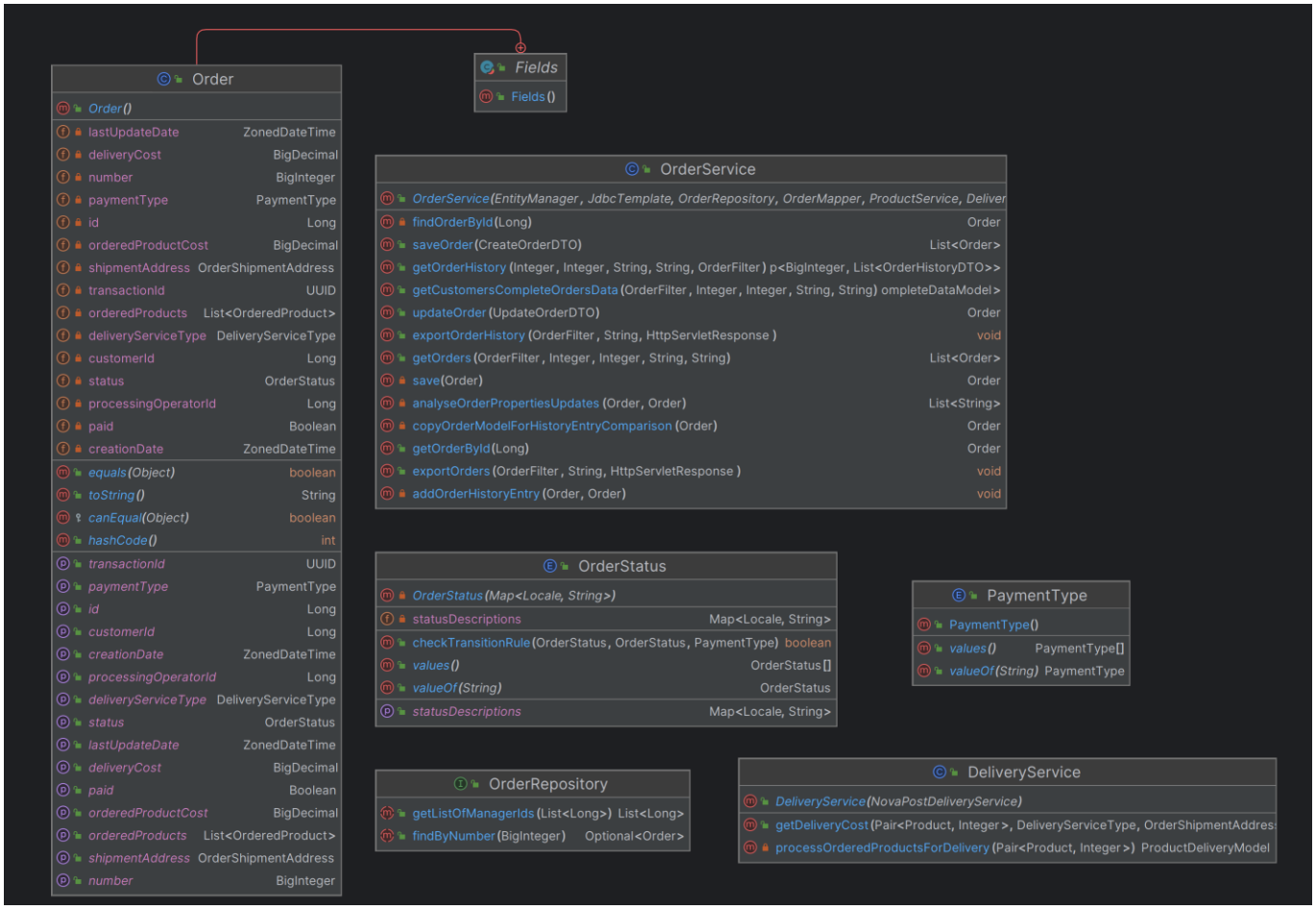

Рисунок 4.1.2 – Діаграма класів логіки роботи з замовленням

Основний клас є клас, що описує саму модель замовлення, які поля та яких типів даних буде містити. Фактично дані ті ж самі, що і у відповідній таблиці бази даних, проте використаний фреймворк Hibernate підтримує асоціації – тому замість описання атрибутів, які посилаються на іншу таблицю, можна прописати логіку отримання повного запису по ідентифікатору і отримати разом з об’єктом замовлення.

Та службові класи, які надають функціонал для роботи з базою даних, а саме репозиторій і клас який реалізує усі методи необхідні для бізнес-логіки, що виступає як сервіс.

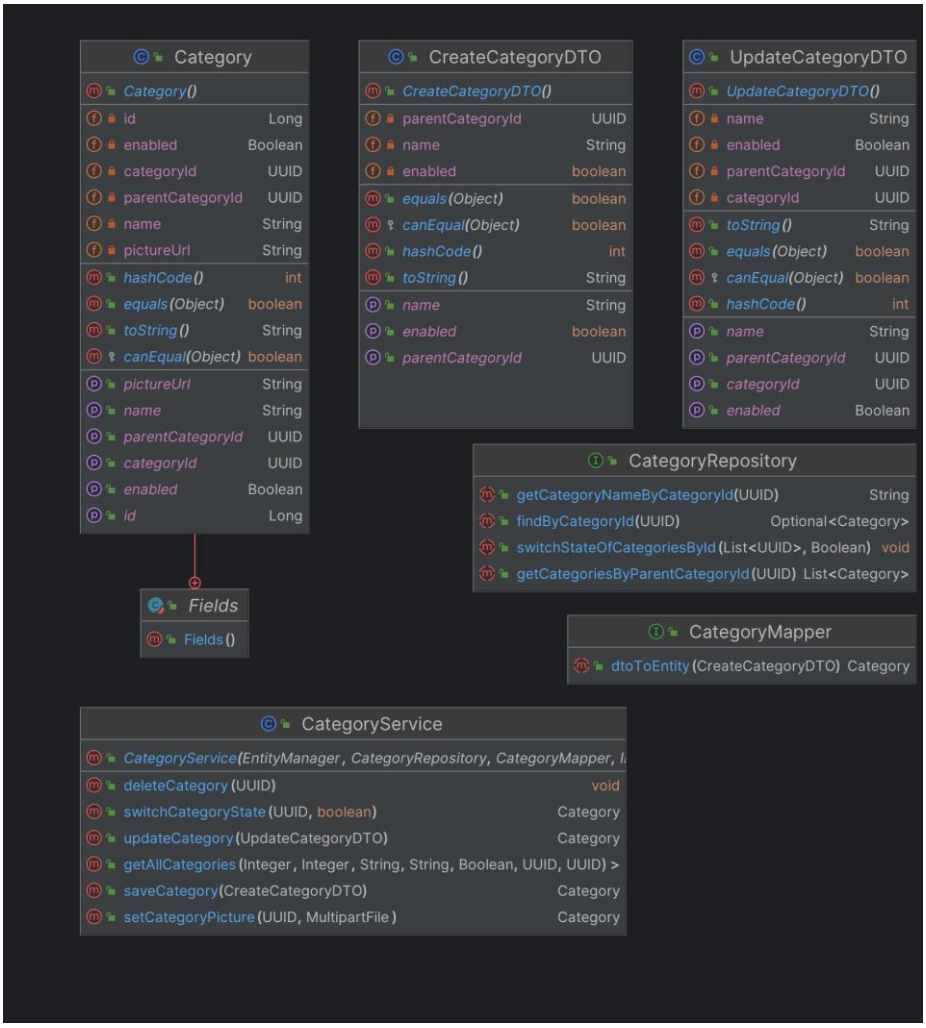

Рисунок 4.1.3 – Діаграма класів логіки роботи з категоріями

Дана діаграма демонструє основні класи для реалізації логіки роботи з моделлю категорій. А саме клас, що описує саму категорію, всі дані з відповідної таблиці бази даних у форматі об’єктно-реляційної моделі.

Службові класи для роботи з базою даних та реалізації бізнес логіки, а також внутрішнього перетворення однієї моделі в іншу.

Додатково є допоміжні моделі, які слугують для отримання інформації від користувача у певних запитах до серверної частини, для операцій створення чи оновлення категорії.

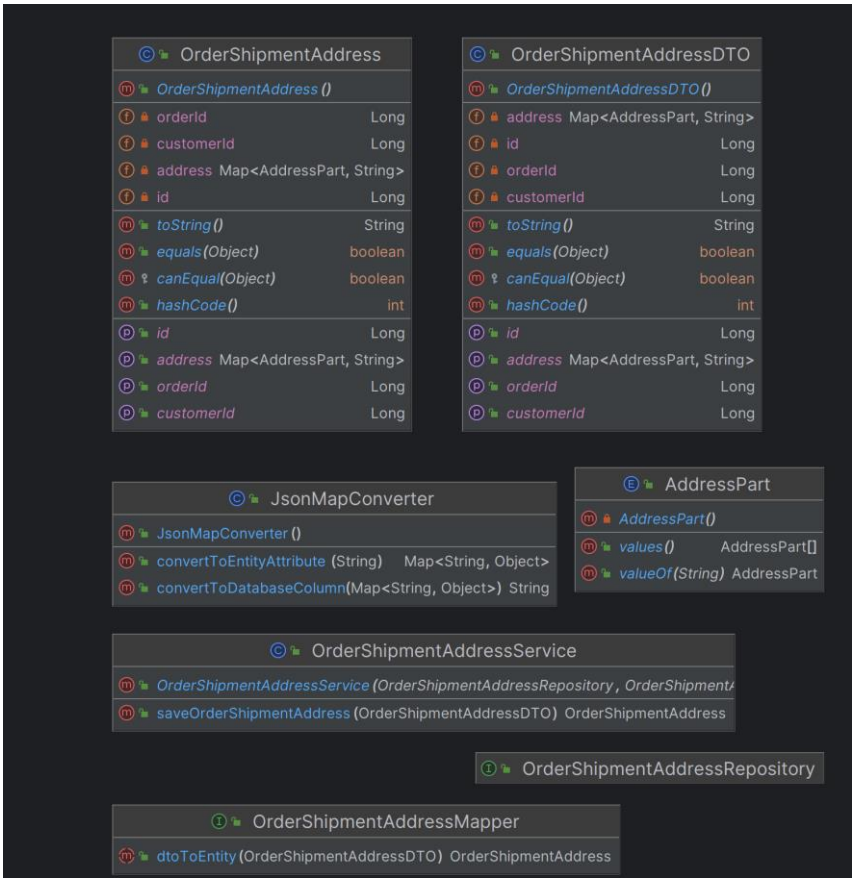

Рисунок 4.1.4 – Діаграма класів логіки роботи з адресами доставки

Ця група класів відповідає за роботу з адресами замовлення, на випадок, якщо користувач забажав замовити продукти з доставкою то певної точки.

Основна модель містить в собі ті ж дані, що і відповідна таблиця в базі даних.

У наявності службові класи для роботи з базою даних та бізнес-логікою.

Також допоміжні моделі, для роботи з адресами, для отримання інформації від користувача і службові класи, для перетворення однієї моделі в іншу, та для запису динамічних адрес у базу даних та відповідно читання цих даних з неї – JsonMapConverter.

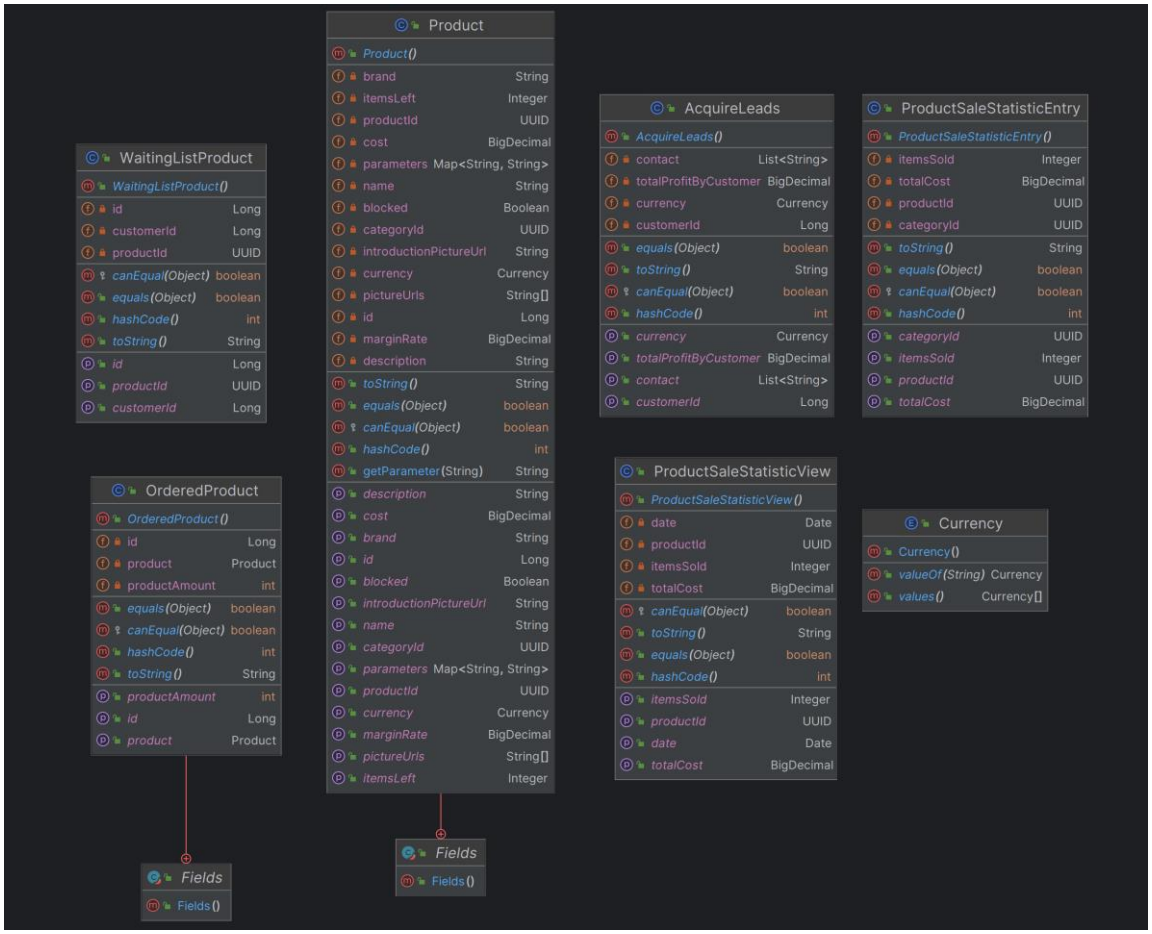

Рисунок 4.1.5 – Діаграма класів моделей пов’язаних з продукцією

Існують декілька основних моделей. Головна це модель продукту, вона містить в собі дані з відповідної таблиці бази даних. Також створено моделі для реалізації логіки роботи зі списком очікування, визначенням замовленого продукту, моделі пов’язані зі статистикою та інші необхідні моделі для процесу роботи з продуктами, валютна інформація, модель збереження інформації про наявність продукту у замовленнях.

Рисунок 4.1.6 – Діаграма класів логіки роботи з транзакціями

Ця діаграма описує основні моделі та службові класи для роботи з даними про транзакції, інтеграція з базою даних та бізнес-логіка, у тому числі методи інтеграції з API платіжного провайдера DataTrans.

4.2 Огляд створення інтерфейсу

Для розробки клієнтського інтерфейсу було залучено HTML, CSS, JavaScript, TypeScript, React та Bootstrap

Спочатку опишемо загальну концепцію, базуючись на вказаному необхідному функціоналу, розділ 1, пункт 1.2.

Застосунок побудовано навколо можливості автоматичного оновлення сторінки засобами бібліотеки React, проект має головну стартову сторінку, головний компонент Application, конфігураційні файли для запуску, збереження змінних оточення, підключених бібліотек.

Також застосунок використовує деякі статичні ресурси, наприклад картинки, список міст України.

Все інше напряму відноситься до реалізації конкретного функціоналу застосунку.

Під час розробки було використано технологію роутингу (рис. 4.1.1) – відображення конкретного вмісту залежно від адреси, та інших обставин, наприклад статусу авторизації користувача у системі.

```
<LanguageProvider>
  <Router>
    <Routes>
      <Route exact path="/home" element={<UI />} />
      <Route exact path="/" element={<UI />} />
      <Route exact path="/sign/up" element={isLoggedIn() ? <UI /> : <SignUpForm />} />
      <Route exact path="/sign/in" element={isLoggedIn() ? <UI /> : <LoginForm />} />
      <Route exact path="/product" element={<ProductComponent />} />
      <Route exact path="/purchase" element={<CheckoutOrder />} />
      <Route
        exact
        path="/customer/personal"
        element={isCustomerLoggedIn() ? <CustomerPersonalCabinet /> : <Unauthorized />}
      />
      <Route
        exact
        path="/manager/personal"
        element={isManagerLoggedIn() ? <ManagerPersonalCabinet /> : <Unauthorized />}
      />
      <Route exact path="/product/edit" element={isManagerLoggedIn() ? <EditProductWrapper /> : <Unauthorized />} />
      <Route exact path="/product/new" element={isManagerLoggedIn() ? <AddProduct /> : <Unauthorized />} />
      <Route exact path="/password/change" element={isLoggedIn() ? <UI /> : <PasswordChange />} />
      <Route path="*" element={<NotFound />} />
    </Routes>
  </Router>
</LanguageProvider>
```

Рисунок 4.2.1 – Роутинг застосунку

Це необхідно для більш зручного користування додатком, можливість переходу відразу до потрібної сторінки.

Основа додатку це реалізація функціоналу для користувача.

Перш за все користувач повинен бути забезпечений можливістю авторизації в системі, також створення персонального акаунту та відновлення доступу до нього у випадку втрати.

Для цього, як і для всіх інших сторінок було використано базову структуру.

Кожен компонент, який виступає повноцінною сторінкою використовує структурні компоненти Header – є навігаційним основним компонентом, Footer – містить додаткову контактну інформацію, посилання на політику використання, посилання на політику конфіденційності.

Окрім цих найпопулярніших компонентів було створено також декілька службових сторінок, а саме сторінка відповіді при помилці 404 NotFound – значить що шуканий ресурс не знайдено, у тому числі сайту невідома дана URL-адреса, та помилка 401 Unauthorized – для випадків, коли користувач намагається перейти на сторінку, доступу до якої він не має.

Також кожен користувач має доступ до секції авторизації, яка містить форми для – авторизації, реєстрації та відновлення паролю.

Для клієнта було реалізовано набір компонентів (рис. 4.2.2) що забезпечують функціонал роботи з системою: для оформлення замовлень, панель доступу до меню, головний екран системи, компоненти необхідні для роботи з адресами, переглядом замовлень, продуктів, кошика та списку очікування продуктів.

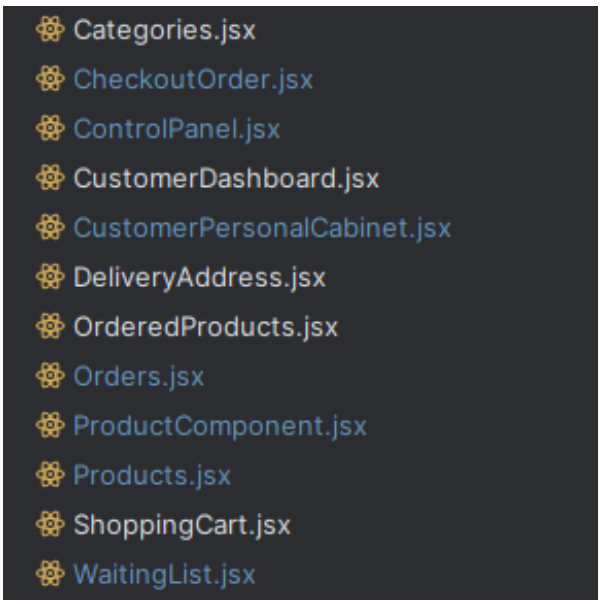

Рисунок 4.2.2 – Перелік деяких основних компонентів

Деякі з компонентів виступають у якості самостійних сторінок, наприклад CustomerPersonalCabinet, ProductComponent

Концепція реалізації в такому випадку використання інших компонентів, які виступають частинами сторінок, наприклад компоненти для відображення замовлень, категорій, списку очікування, тощо.

Загальний принцип реалізації клієнтського інтерфейсу для усіх користувачів наступний: існують декілька основних сторінок, від однієї, решта компонентів відповідають за конкретний функціонал, залежно від адреси сторінки або деяких параметрів запиту обирається конкретний компонент для відображення на цій сторінці.

Логіка всередині компонентів побудована навколо використання хуків – спосіб описання внутрішнього стану, або з залученням переданих у компонент даних ззовні.

Принцип комунікації з сервісами застосунку побудований навколо використання Fetch API, реалізація асинхронних HTTP – запитів.

Інфраструктурно проект поділено на декілька частин, а саме перелік констант, таких як адреси, функції запитів до сервісів, фотографії та інші ресурси для

відображення, опис основних моделей – класів, виконано з залученням TypeScript, для зручної робити з даними запиту та відповіді від сервера.

Стилізація в основному зроблена за рахунок готових рішень бібліотеки Bootstrap, проте деякі специфічні класи стилів реалізовані вручну.

4.3 Огляд створення сервісів

Для розробки сервісів було використано мову програмування Java, та супутні фреймворки для реалізації веб-застосунків, роботи з базою даних, організацією роботи мікросервісного застосунку.

Першочергово необхідно визначитися з архітектурою, головна мета мікросервісного підходу при розробці застосунків, це розділити усю необхідну логіку на зони відповідальності, зробивши сервіси якомога більш уніфікованими та незалежними.

Специфіка розробленого забезпечення полягає у роботі з системою, яка дозволяє керувати процесами пов’язаними з електронною торгівлею.

Відповідно головними операціями є можливість придбання продукції користувачем, можливість її перегляду для вибору, зручність у процесі підбору та інші споріднені операції.

Цільовою моделлю є модель користувача, всі замовлення та операції прив’язані до нього.

Першим сервісом буде користувацький, він міститиме усі операції, що стосуються процесу роботи з користувачем.

Список функціоналу закладеного в основі сервісу користувачів наступна:

- Перегляд списку користувачів
- Реєстрація клієнтів та створення менеджерів
- Авторизація
- Відновлення втраченого паролю

Для реалізації реєстрації та відновлення паролю потрібен процес верифікації,

проте потенційно список функцій може поповнитися, відповідно варто створити окремий сервіс, який буде слугувати для процесу верифікацій певних дій.

Список функціоналу закладеного в основі верифікаційного сервісу:

- Отримання запиту на формування запису про верифікацію та ініціювання повідомлення користувачеві
- Верифікація на основі коду отриманого від користувача з поверненням результату – успіх або ні

Окрім верифікаційних повідомлень існує потреба в інших сповіщеннях, тому було створено окремий сервіс для надсилання повідомлень з наступною функціональністю:

- Надсилання повідомлення на задану платформу, з заданим текстом при отриманні запиту через брокер повідомлень від інших сервісів

На даний момент підтримує електронні листи та повідомлення у телеграм через бота.

Далі потрібна реалізація сервісу для роботи з замовленнями, він буде найбільшим за обсягом функціоналу:

- Створення, перегляд, оновлення категорій продуктів
- Створення, перегляд, оновлення продуктів
- Створення, перегляд, оновлення, експорт замовлень
- Створення, перегляд транзакцій
- Інтеграція з поштовим API

Кожна група функціоналу поділена на підгрупи у програмні реалізації. Усі сценарії роботи поділені за логічним змістом, кожен окремий має свій персональний контролер, який оброблює запити до сервісу, також залежно від цільової моделі є реалізація сервісу, репозиторію, моделі для роботи з базою даних та моделей для роботи з проміжними даними, спеціальні допоміжні реалізації за потреби.

Цей принцип діє на всі сервіси, це зроблено з метою підтримування чистоти, адаптивності, розширюваності програми та написаного коду з найменшою витратою часу та ресурсів.

Окрім описаних чотирьох сервісів є ще три окремі проекти.

Перший службовий сервіс направлений на централізацію усієї конфігурації, а саме головних базових налаштувань для кожного сервісу, які можуть відрізнятися.

Працює на основі Spring Cloud технологій, головна задача сервісу це роздача конфігурації усім іншим, перед їх запуском, тобто кожен сервіс при запуску надсилає запит до сервісу конфігурацій з метою отримання конфігураційного файлу і його вмісту.

Інший сервіс створено для визначення поточного стану мікросервісної системи, цей сервіс займається збором метрик, аналітик, веде журнал підключень та збоїв у процесі роботи та надає вичерпну інформацію кінцевому користувачеві, за рахунок наявності декількох панелей моніторингу, які візуально дають змогу навіть без цільових знань, оцінити стан системи та повідомити про помилку у разі її виникнення.

Останній проект є бібліотекою, це необхідно для використання спільної конфігурації безпеки, роботи з брокером повідомлень, кешем та іншими налаштуваннями, які є спільними для усіх інших сервісів. Окрім конфігурації для усіх сервісів, що імпортують цю бібліотеку також наявні спільні моделі для передачі даних між сервісами за допомогою брокера повідомлень, для комунікації за допомогою WebSocket, спеціальні сервіси для роботи з SQL - запитами до бази даних, створення QR-кодів. Усі службові сервіси для роботи з файловим експортом чи форматуванням даних, сервіси спільного використання, щоб забезпечити доступ до інформації про поточно авторизованого користувача, чи роботи з JWT, сервіси для роботи зі спільними сторонніми API для збереження медіа.

Для імпорту та поширення використано GitHub платформу, яка надає функціонал для побудови бібліотеки за допомогою інструменту збірки Maven, після налаштування та публікації релізу – проект стає доступним для імпорту.

Усі службові сервіси використовують ці налаштування (рис. 4.3.1)

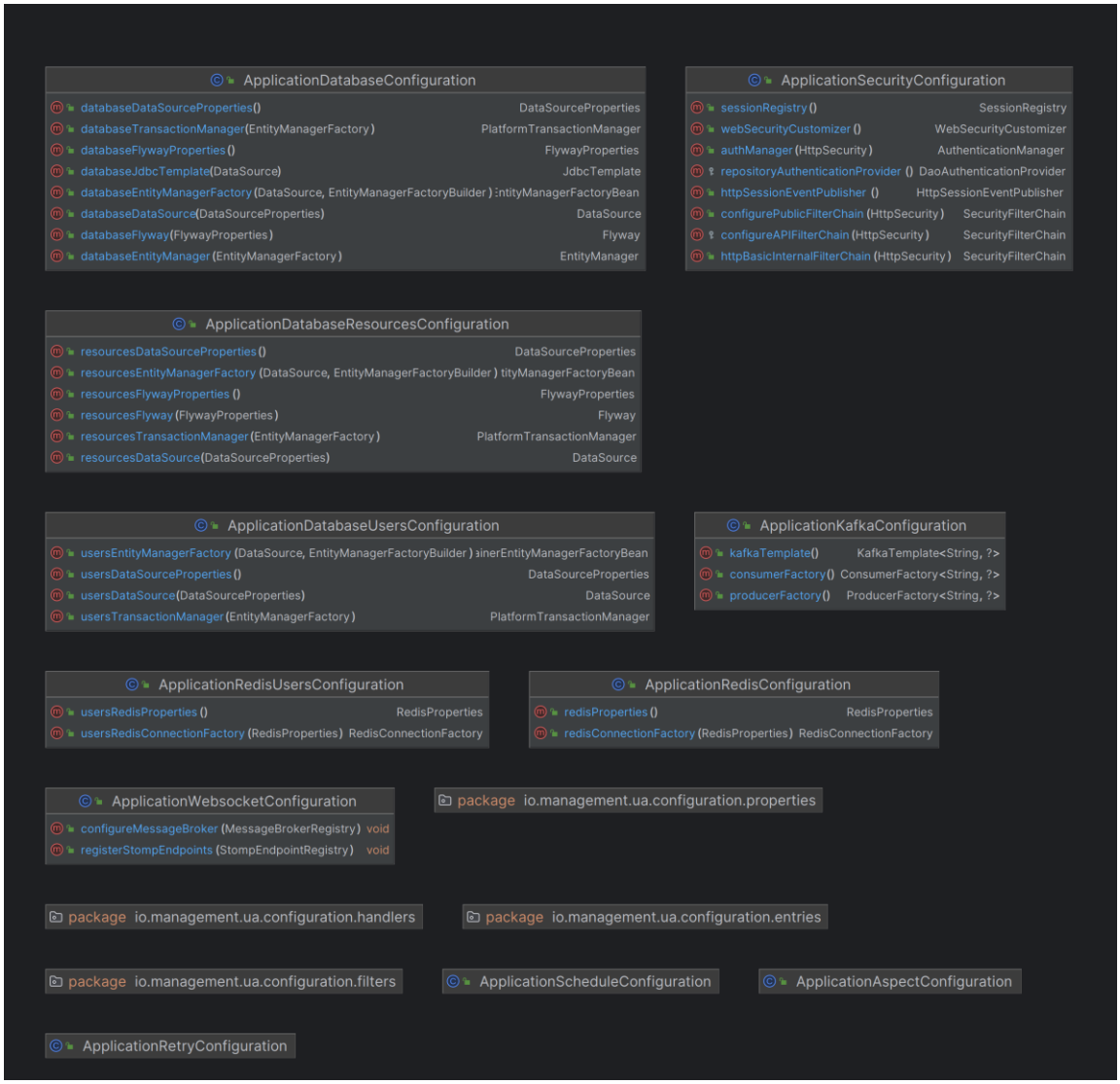

Рисунок 4.3.1 – Основні конфігураційні класи

Ця схема демонструє лише головні конфігураційні класи, весь проект налічує більше сотні різних класів, які відрізняються за призначенням

4.4 Бази даних

Розглянемо структуру використовуваних баз даних.

Бази даних даного застосунку використовуються для збереження даних про користувачів (рис. 4.4.1) , категорії, продукти, статистику продажів, замовлення,

адреси доставки, транзакції, історію змін замовлення та списку очікування товарів для кожного користувача, фотографії продуктів та категорій (рис. 4.4.3)

Тобто тих даних які передбачають довготривале зберігання, та є критично важливими для системи.

Також існує окрема база даних для збереження файлових ресурсів (рис. 4.4.2) – фотографій для категорій та продуктів.

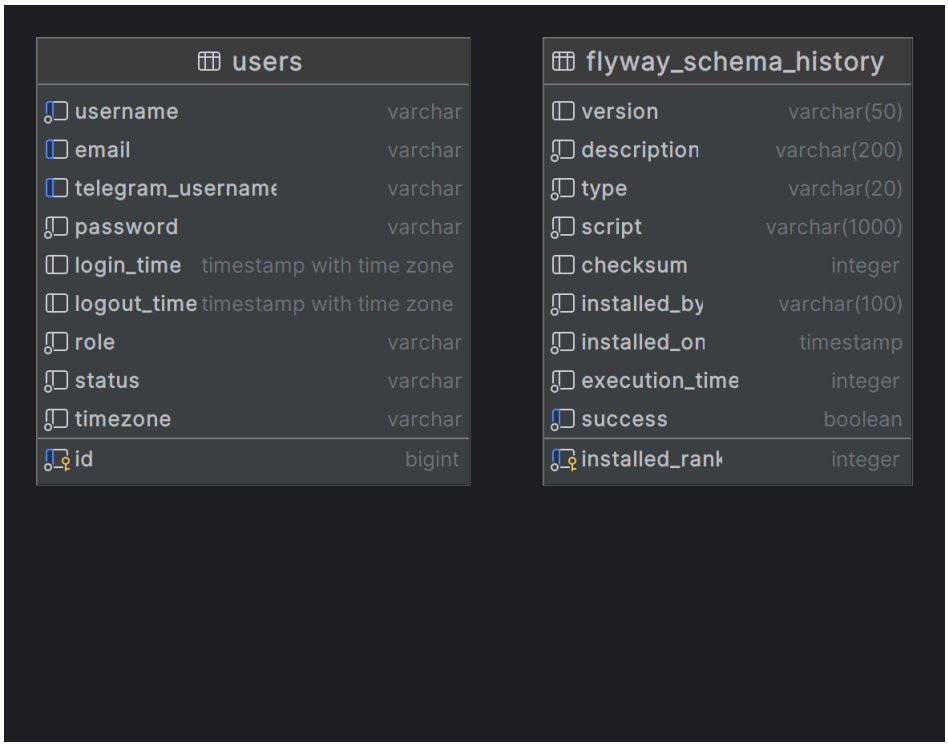

Рисунок 4.4.1 – Схема бази даних користувачів

Для користувачів використовується лише одна таблиця, так як логіка не вимагає створення певних асоціативних моделей, чи відношень, які були б напряму пов’язані лише з користувачем.

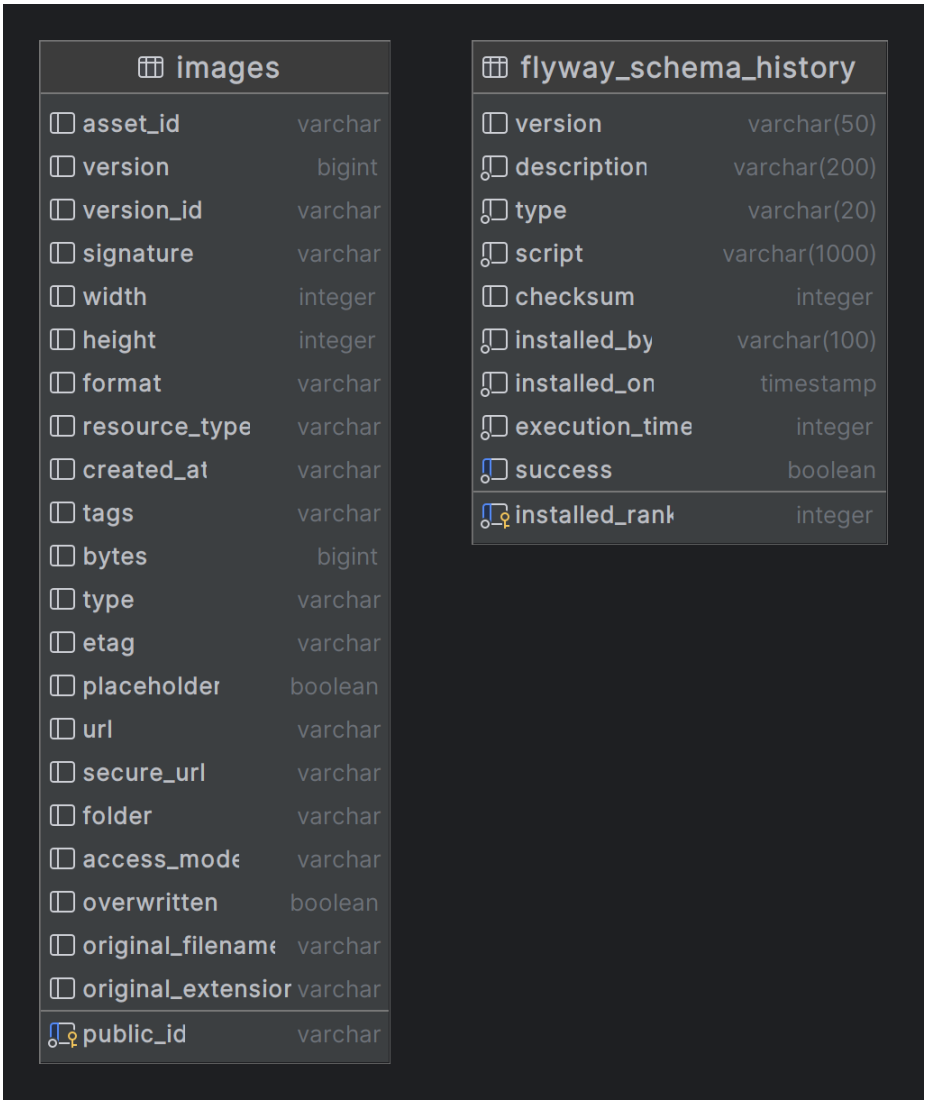

Рисунок 4.4.2 – Схема бази даних ресурсів

Для збереження фотографій використовується API Cloudinary [30], в базі даних зберігаються необхідні дані для виконання запитів до API, та відображення цих фото на стороні користувацького інтерфейсу.

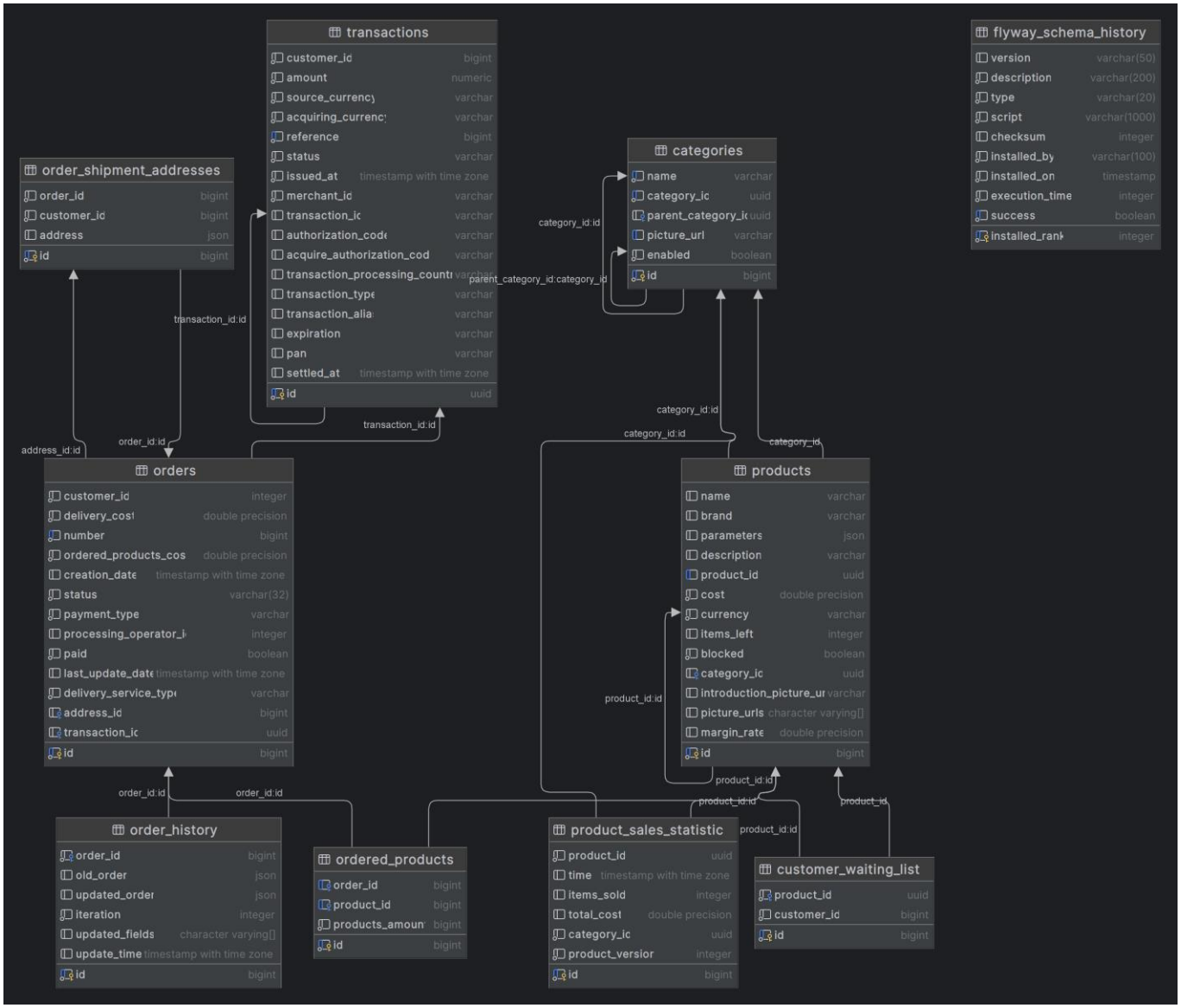

Рисунок 4.4.3 – Схема бази даних сервісу замовлень

Збереження даних, що відносяться до бізнес-процесів сервісу замовлень структурно набагато складніше порівняно з попередніми. Для забезпечення логіки роботи функціоналу необхідно налаштувати взаємодію та узгодженість даних про різні споріднені сутності. Почнемо з найбільш незалежної таблиці – categories. Ця таблиця слугує для збереження даних про категорії продуктів. Кожна категорія має свій індентифікатор запису у таблиці, та окреме унікальне поле типу UUID, яке також виступає ідентифікатором запису. Аналогічну ситуацію демонструє таблиця products. Це зумовлено комбінацією двох факторів, простота генерації первинного ключа та безпека при роботі з записами на стороні клієнтського застосунку.

На певних сторінках в якості параметрів адреси сторінки фігурують ідентифікатори, для застосунку було використано посилання та пошук даних саме за UUID - ідентифікатором. Такий підхід майже унеможливлює можливість підбору такого ідентифікатора для якогось іншого запису вручну.

1) Таблиця "Категорії" (categories): зберігає інформацію про категорії товарів. Включає атрибути для назви категорії, ідентифікатора, URL титульного фото, статусу доступності та посилання на батьківську категорію.

2) Таблиця "Продукти" (products): зберігає дані про товари. Включає атрибути для назви, бренду, опису, ідентифікатора, ціни, кількості на складі, статусу доступності, посилання на категорію та фотографії продукту.

3) Таблиця "Лист очікування користувача" (customer_waiting_list): зберігає посилання на користувача та продукт, на який він чекає.

4) Таблиця "Статистика продажів продукту" (product_sales_statistic): зберігає дані про продажі товарів, включаючи ідентифікатор продукту, час продажу, кількість проданих товарів та загальну вартість.

5) Таблиця "Замовлення" (orders): зберігає дані про замовлення користувачів, включаючи ідентифікатор користувача, вартість доставки, номер замовлення, загальну вартість продуктів, статус замовлення та інші деталі.

6) Таблиця "Продукти замовлення" (ordered_products): зберігає інформацію про продукти у кожному замовленні, включаючи ідентифікатор продукту, ідентифікатор замовлення та кількість продуктів.

7) Таблиця "Адреса доставки замовлення" (order_shipment_addresses): зберігає адреси доставки замовлень у форматі JSON для різних перевізників.

8) Таблиця "Історія змін замовлення" (order_history): зберігає історію змін у статусі замовлення, включаючи старий та оновлений стан замовлення, номер оновлення та інші деталі.

9) Таблиця "Транзакції" (transactions): зберігає дані про фінансові транзакції, включаючи ідентифікатор користувача, суму, валюту, посилання, статус та інші деталі

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Після опису програмної реалізації перейдемо до взаємодії користувача із системою, що включає в себе:

- Опис системних вимог
- Інсталяція та запуск застосунку
- Інструкція для користувача
- Приклади результатів роботи

5.1 Опис системних вимог

Для запуску застосунку необхідно інсталювати Docker. Основні вимоги до системи включають наступні компоненти:

1. Операційна система: Linux, macOS або Windows
2. Процесор: Багатоядерний процесор (рекомендується мінімум 4 ядра)
3. Оперативна пам'ять: Мінімум 16 ГБ
4. Місце на диску: Мінімум 50 ГБ вільного місця
5. Docker: Версія 20.10 або новіша
6. Docker Compose: Версія 1.29 або новіша

Контейнери та їх вимоги

Система використовує наступні контейнери:

1. ELK Stack (Elasticsearch, Logstash, Kibana):
   Elasticsearch: мінімум 4 ГБ RAM
   Logstash [31]: мінімум 1 ГБ RAM
   Kibana [32]: мінімум 1 ГБ RAM
2. Kafka: мінімум 2 ГБ RAM
3. Redis: мінімум 512 МБ RAM
4. Java Spring Boot сервіси (6 екземплярів): мінімум 1 ГБ RAM на кожен сервіс

5.2 Інсталяція та запуск за стосунку

1. Інсталяція Docker:

Завантажте Docker з офіційного сайту і встановіть його відповідно до інструкцій для вашої операційної системи

2. Клонування репозиторію:

git clone <URL-вашого-репозиторію>

Усі сервіси опубліковані на платформі GitHub

3. Запуск контейнерів:

Виконайте команду для запуску всіх контейнерів [33]:

docker-compose up -d

5.3 Інструкція для користувача

1. Запуск та перевірка стану контейнерів:

Переконайтесь, що всі контейнери запущені та працюють за допомогою команди docker ps.

Усі 6 сервісів для серверної частини мають бути запущені, 7 контейнер містить інтерфейс – доступ до нього можливий через веб-браузер за адресою http://YOUR_SERVER_HOST:3333 або налаштувати інший порт у налаштуваннях.

У результаті запуску усіх сервісів ви отримаєте доступ до програмного інтерфейсу, користування яким надає усі необхідні можливості реалізації бізнес-процесів. Сам інтерфейс виконано в інтуїтивній формі, кожна сторінка буде розглянута у прикладах результатів роботи.

Так як зареєструватися як менеджер неможливо, система передбачає створення менеджера за замовчуванням з логіном manager та паролем password, після першого запуску системи можна створити нові акаунти для менеджера і відмовитися від цього або змінити пароль.

5.4 Приклади результатів роботи

Інструкція для інтерфейсу

Неавторизований перегляд передбачає можливість перегляду доступних продуктів, категорій, перегляд конкретних продуктів та детальну інформацію про них, можливість додавання продуктів до кошика (рис. 5.4.1)

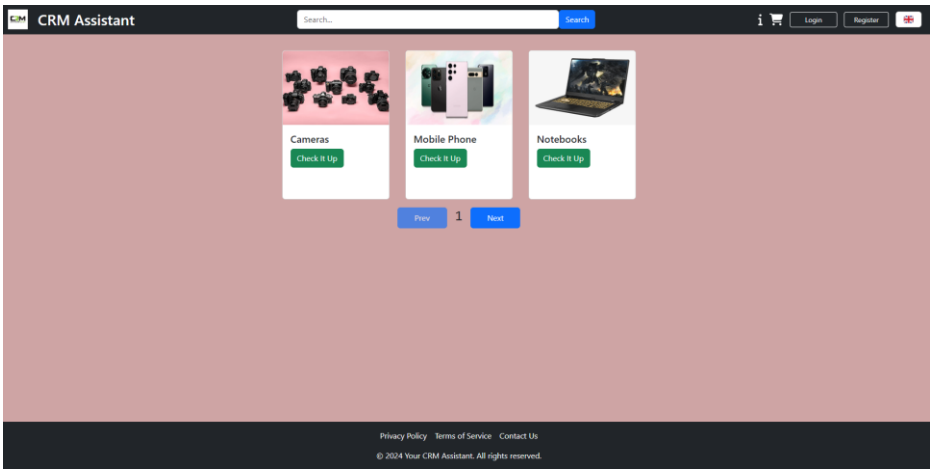

Рисунок 5.4.1 – Стартова сторінка системи при неавторизованому перегляді

Також доступна сторінка з формами для авторизації, відновлення паролю та реєстрація (рис. 5.4.2)

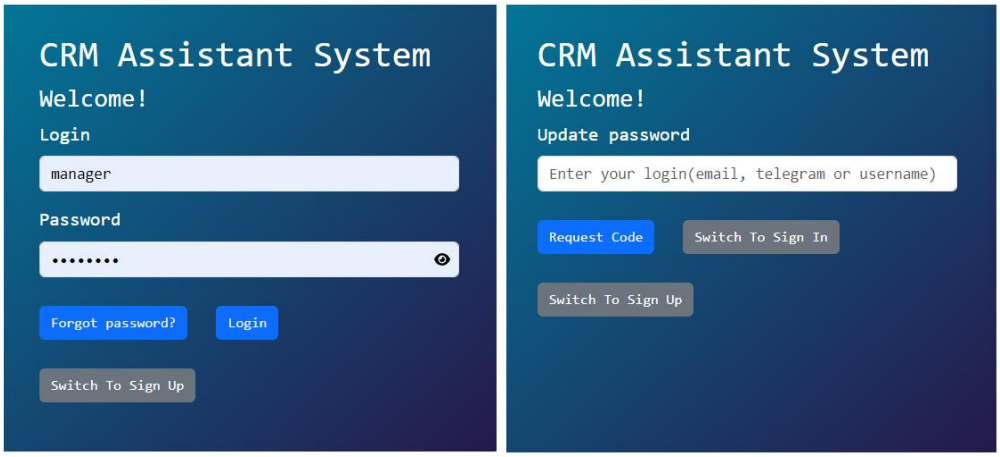

Рисунок 5.4.2 – Сторінки авторизації та відновлення паролю

Частина візуалізації для клієнта після авторизації – з’являється можливість перегляду меню, додавання продуктів до списку очікування та оформлення замовлення (рис. 5.4.3)

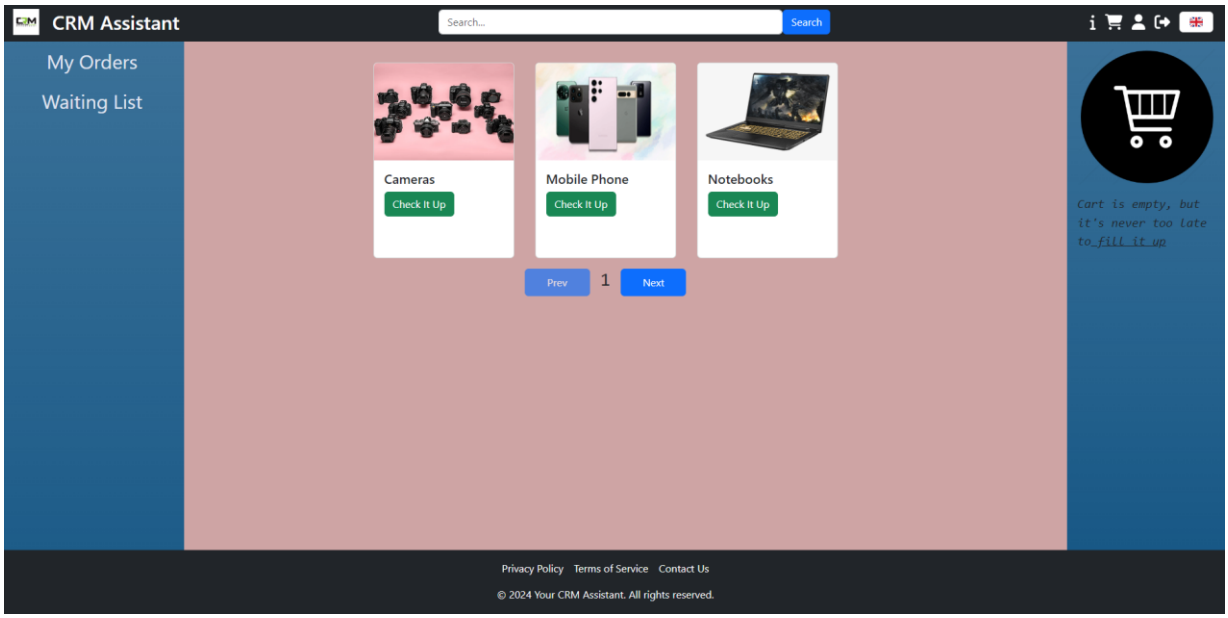

Рисунок 5.4.3 – Зміст користувацького меню та кошику

Меню та кошик оформлені у якості спливаючих вікон, які анімовано з’являються на сторінці при натисканні на відповідні кнопки, щоб закрити ці спливаючі елементи варто натиснути на вільну частину екрану, тобто початковий екран, або нижню панель чи ті самі кнопки.

Клієнт має можливість роботи зі списком очікування товарів (рис. 5.4.4)

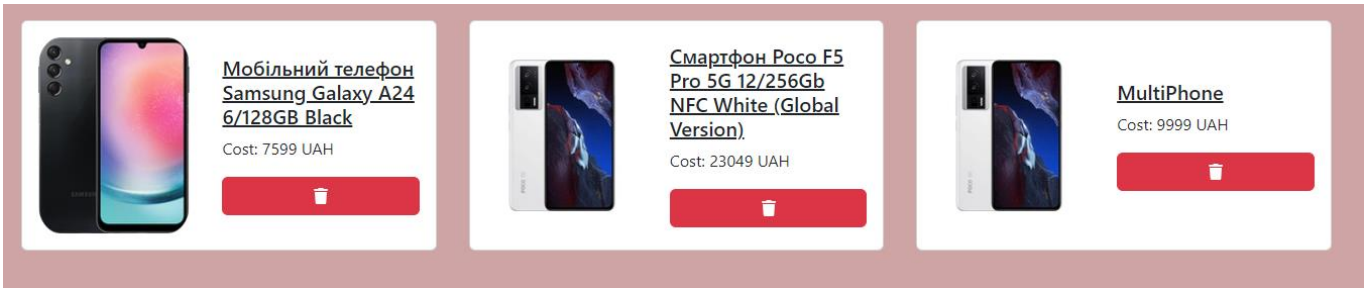

Рисунок 5.4.4 – Зміст списку очікування клієнта

Аналогічно, доступна можливість перегляду даних про існуючі замовлення (рис. 5.4.5)

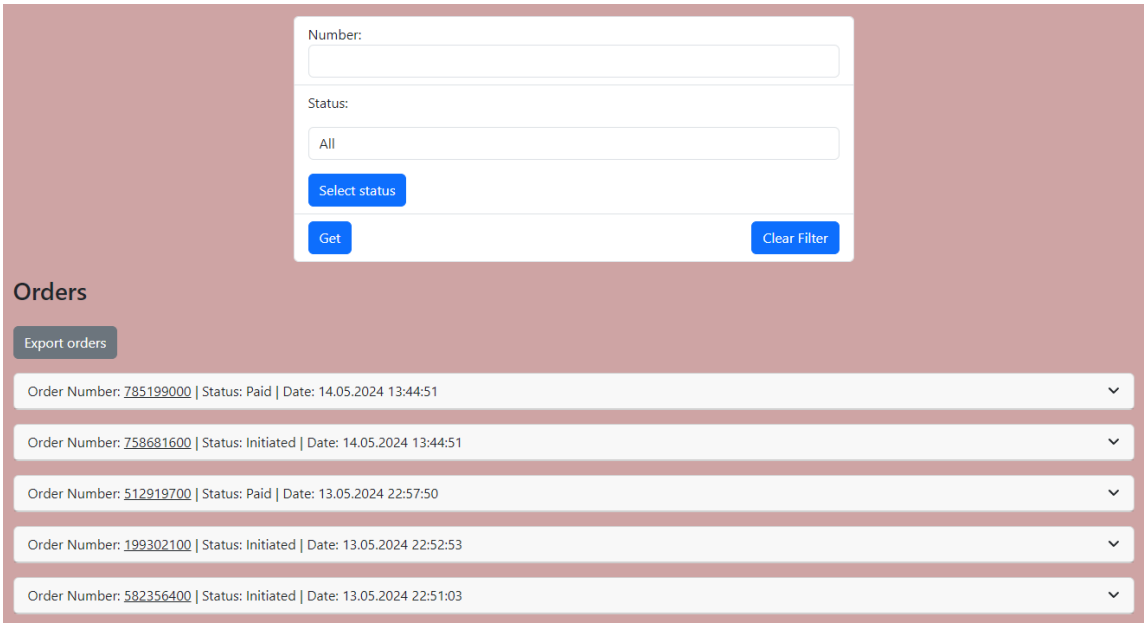

Рисунок 5.4.5 – Приклад перегляду існуючих замовлень

Для клієнта доступна якомога деталізована інформація про кожне замовлення (рис. 5.4.6)

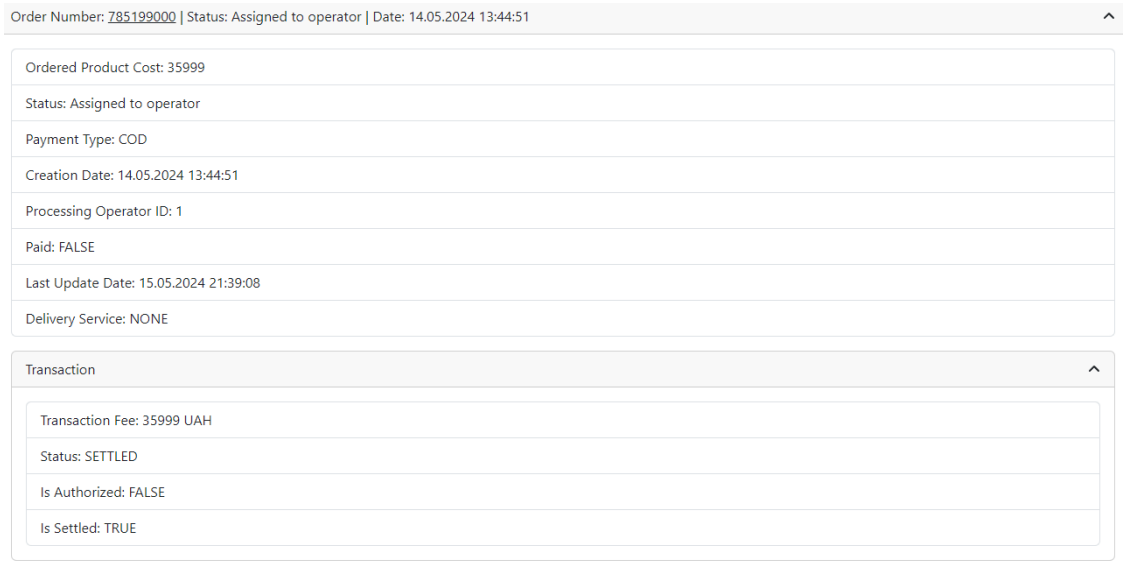

Рисунок 5.4.6 – Приклад перегляду існуючого замовлення

В тому числі є можливість перегляду історії змін кожного замовлення (рис. 5.4.7)

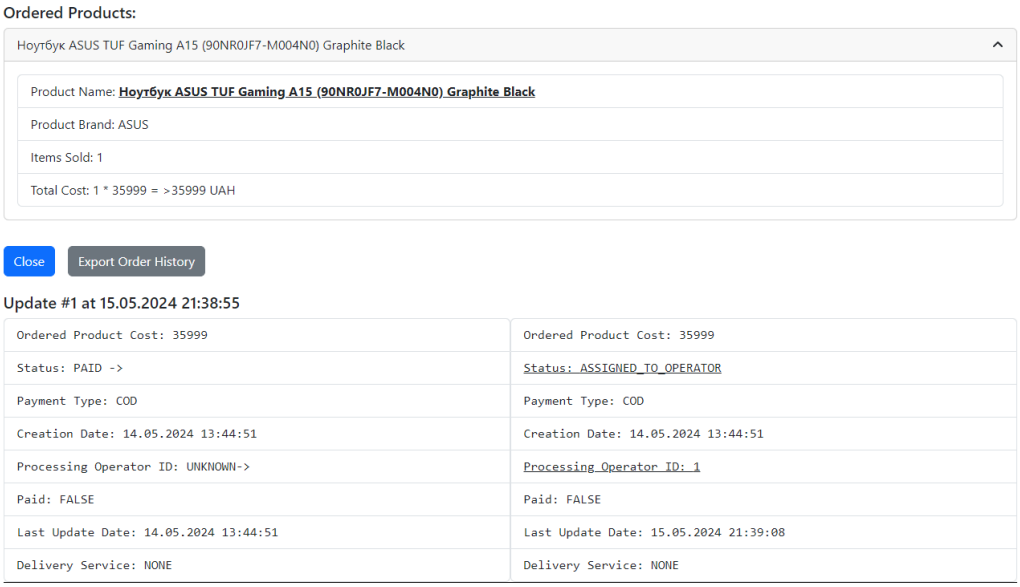

Рисунок 5.4.7 – Приклад перегляду історії замовлення та замовленого продукту

Також кожен клієнт має можливість експортувати до Excel файлу як список замовлень, так і історію змін конкретного.

Усі клієнти можуть переглянути сторінку продукту з детальною інформацією (рис. 5.4.8)

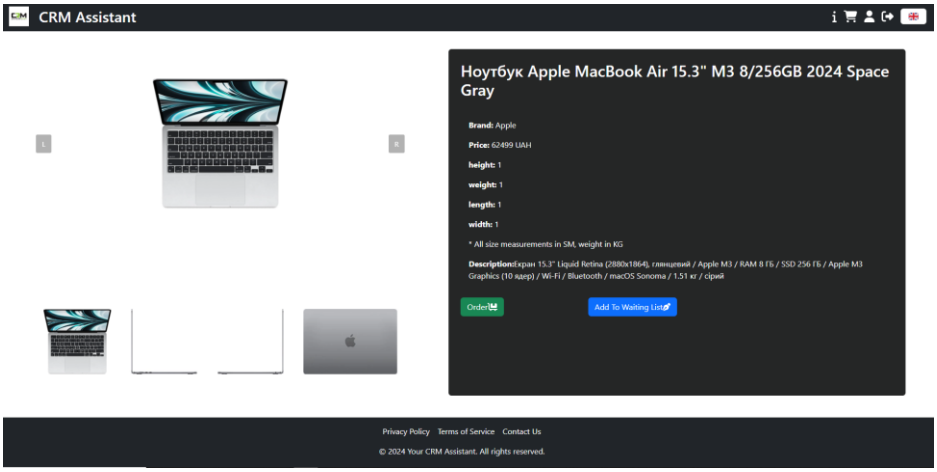

Рисунок 5.4.8 – Приклад перегляду продукту користувачем

Після вибору продуктів користувач має змогу оформити замовлення з вибором типу оплати, доставки, тощо (рис. 5.4.9).

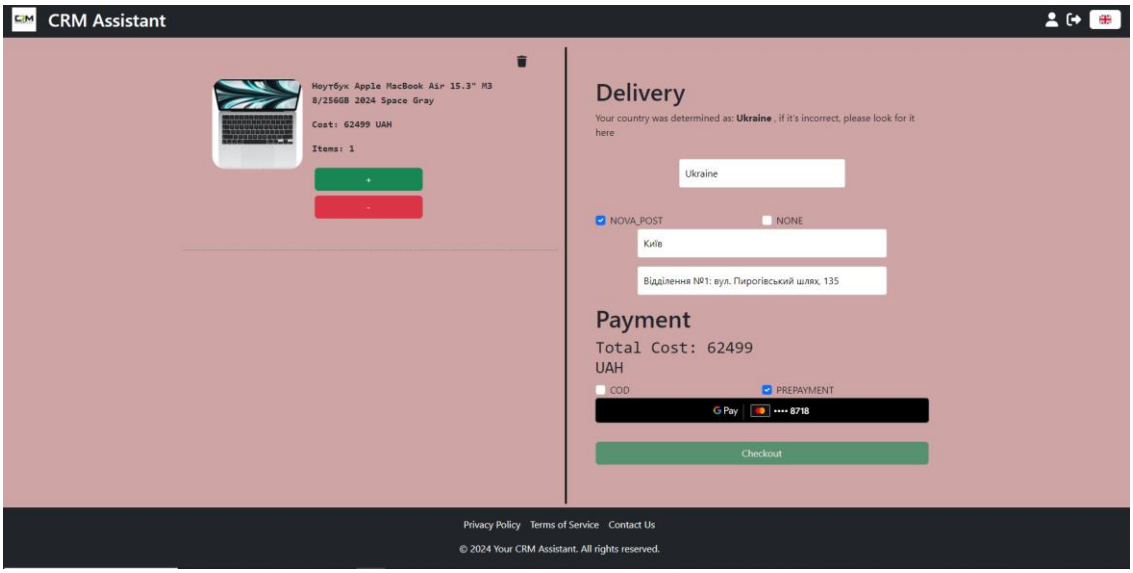

Рисунок 5.4.9 – Приклад перегляду сторінки оформлення замовлення

Окрім користувача існує окрема візуалізація функціоналу для менеджера (рис. 5.4.10).

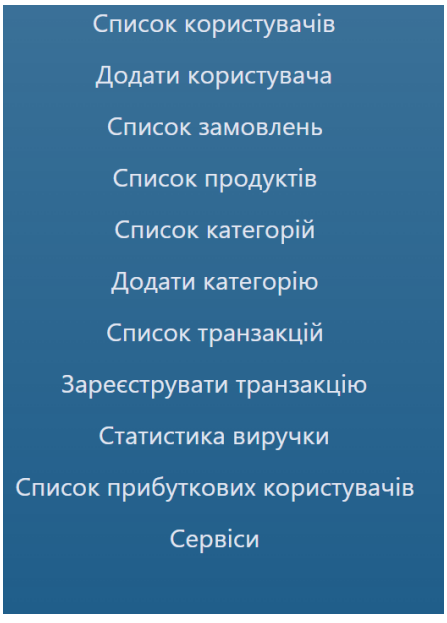

Рисунок 5.4.10 – Зміст меню адміністратора

Користувач з даною роллю має можливість повного керування усіма процесами та записами напряму пов’язаними з бізнес-логікою застосунку.

Перегляд користувачів, з можливістю пошуку по ключовим параметрам, можливість блокування доступу, та форма для додавання нового менеджеру за виникнення такої потреби для бізнесу (рис 5.4.11).

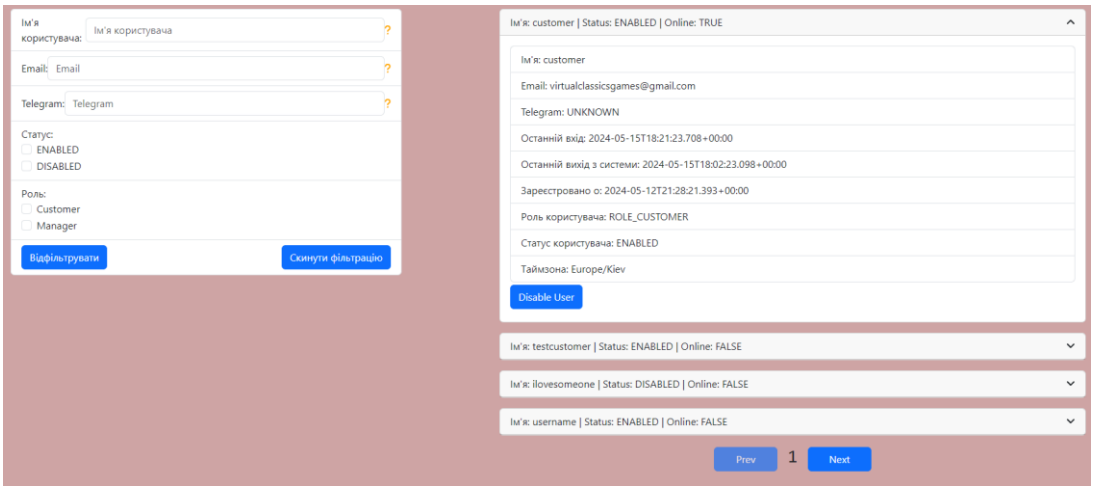

Рисунок 5.4.11 – Сторінка перегляду користувачів

Доступна та зрозуміла форма створення менеджерів (рис 5.4.12)

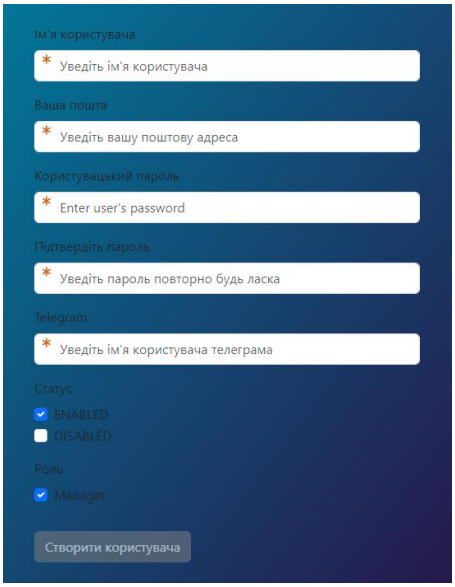

Рисунок 5.4.12 – Форма створення нового менеджера

Абсолютно так само, як і користувач, адміністратор має змогу переглядати список замовлень, проте він має можливість бачити усі замовлення та оновлювати їх статуси, залежно від ситуації (рис. 5.4.13)

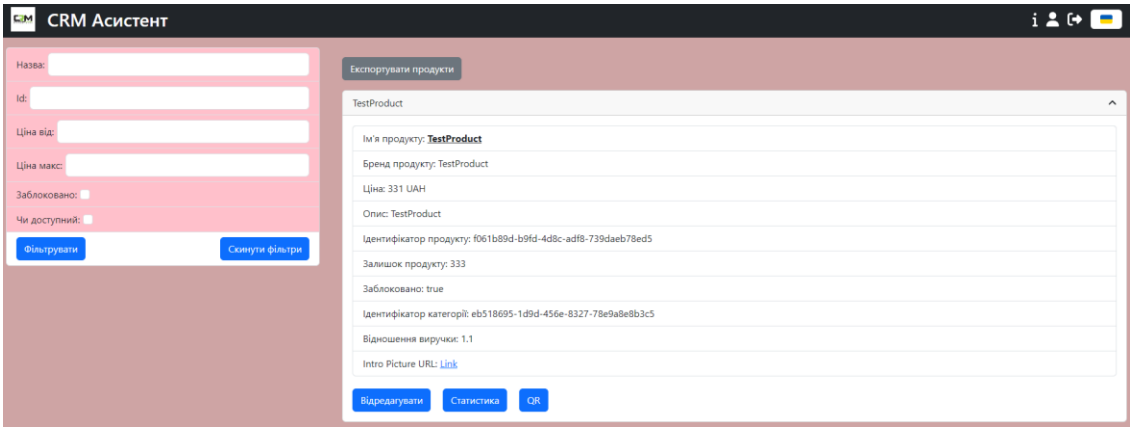

Рисунок 5.4.13 – Сторінка перегляду існуючих продуктів

Менеджер має можливість змінювати дані про продукцію, переглянути статистику продажів (рис. 5.4.14) – скільки одиниць за який день було продано, також є можливість завантажити QR-код з посиланням на сторінку продукту, це зроблено з орієнтацією на роботу зі складом, можливість маркування продукції спеціалізованими наліпками водночас виступатимуть маркетингом електронної платформи, тощо.

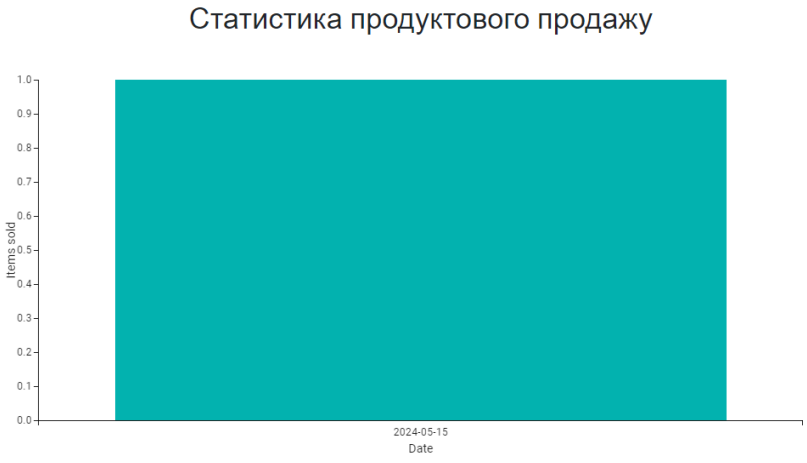

Рисунок 5.4.14 – Приклад статистики продажу(модальне вікно сторінки)

Під час редагування продукту менеджер має змогу змінювати назву, бренд, опис, вартість, кількість товарів на складі, також замінювати чи додавати фотографії товарів, є окрема панель для створення динамічних параметрів опису, наприклад розміри, чи країна походження, гарантія, тощо (рис 5.4.15).

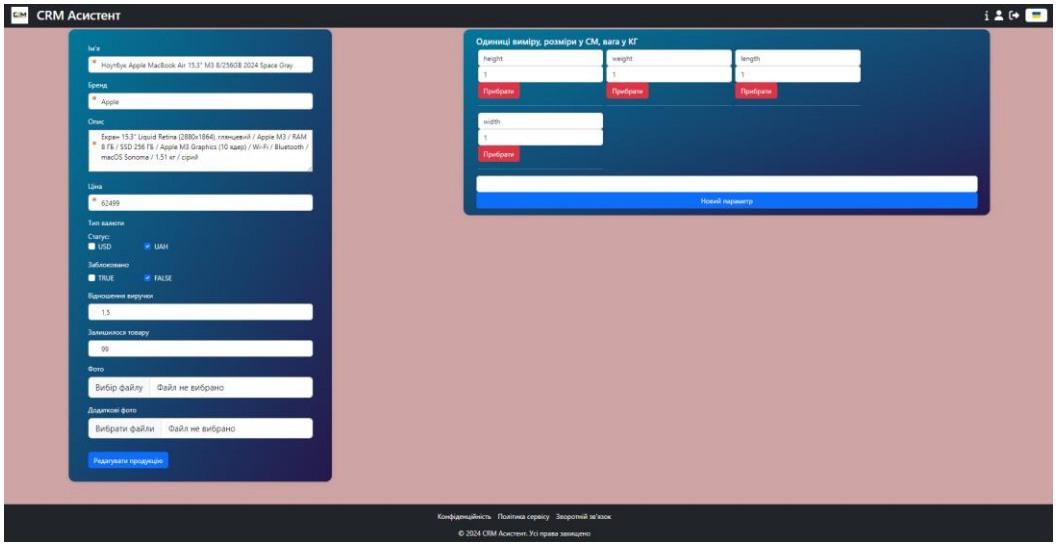

Рисунок 5.4.15 – Сторінка зміни та формування продукції

Однією з ключових сутностей є категорії продуктів, вони дозволяють класифікувати та розділити продукцію по групам на основі користувацьких потреб (рис. 5.4.16).

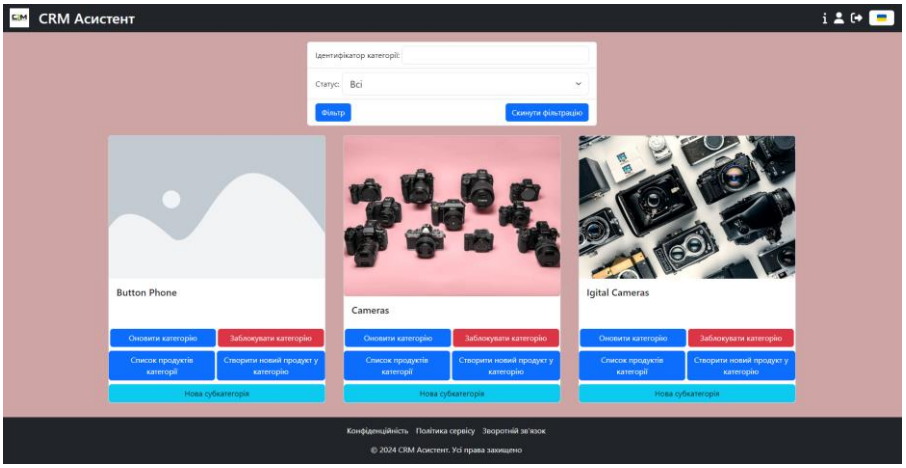

Рисунок 5.4.16 – Сторінка перегляду категорій

Сторінка перегляду надає можливість пошуку категорії по ідентифікатору, статусу, а також надає ряд функціональних кнопок для переходу до оновлення категорії, блокування, перегляду продуктів конкретної категорії, додавання нового продукту до категорії та створення підкатегорії (рис. 5.4.17).

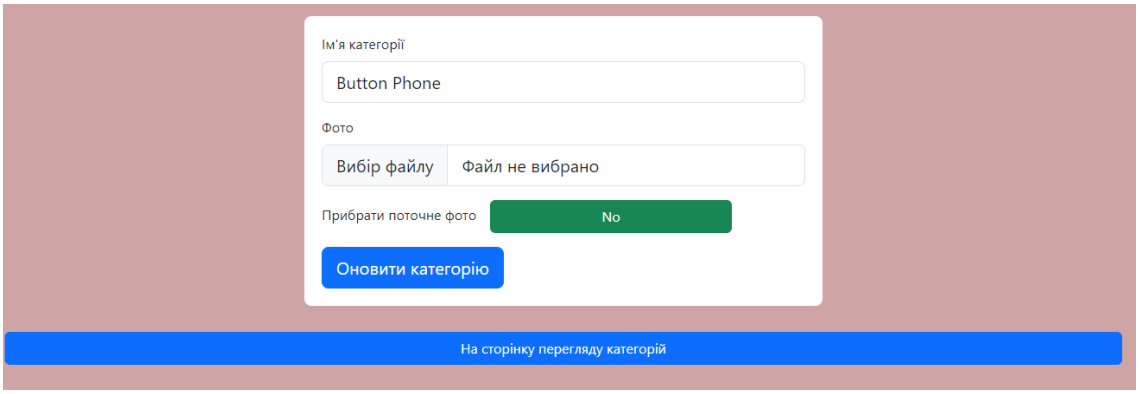

Рисунок 5.4.17 – Сторінка оновлення та створення категорії

Також менеджер має змогу переглядати історію платежів (рис. 5.4.18) , якщо під час замовлення користувач обрав варіант передплати, або ж вносити платіжну інформацію самостійно, у випадку оплати клієнтом в точці видачі, чи поштовому відділенні.

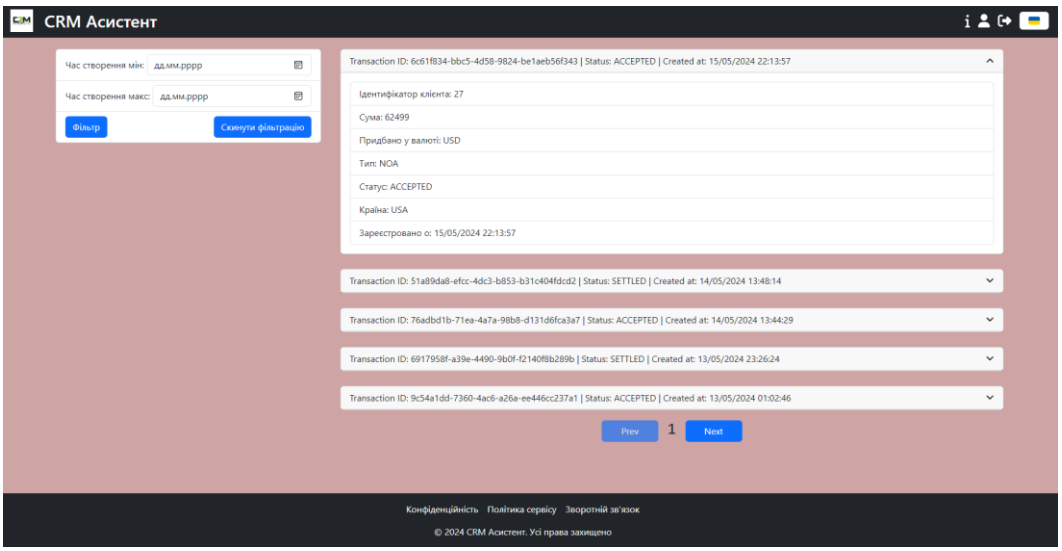

Рисунок 5.4.18 – Сторінка перегляду транзакцій

Форма створення транзакції вручну (рис. 5.4.19), використовується для внесення даних про оплату, також автоматично оновлює статус замовлення на оплачене, якщо не задіяно процес доставки. У випадку оплати у віддаленому пункту видачі, буде лише зареєстрована нова платіжна операція.

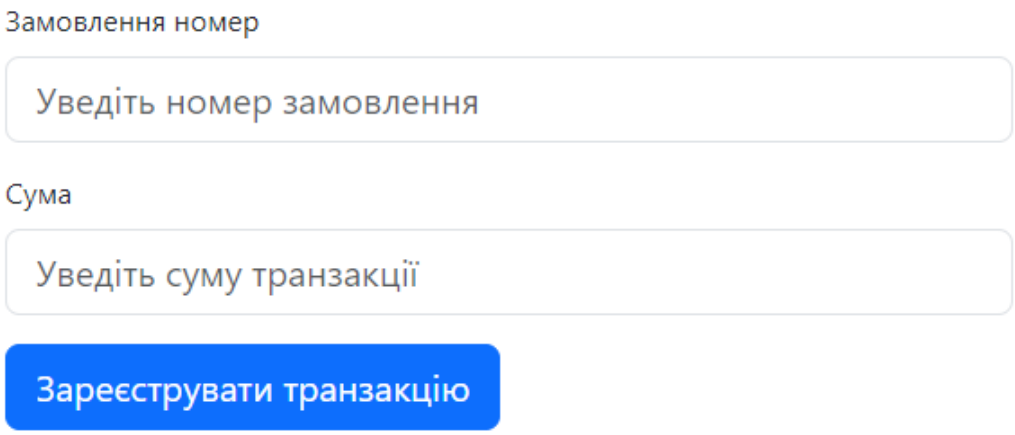

Рисунок 5.4.19 – Сторінка створення транзакцій

Окрім CRUD-операцій з основними моделями та сутностями, доступний перегляд статистики по зароблених коштам (рис. 5.4.20) з можливістю автоматичної конвертації відображуваної інформації в потрібну валюту за вибором користувача, контактна інформація клієнтів (рис. 5.4.21), необхідна для швидкого та зручного сповіщення про нові наявні позиції чи для спеціальних пропозицій від магазину, впорядкована по найбільшій прибутковості, та посилання на зовнішні ресурси інтеграцій, а саме сторінка керування платіжного процесору DataTrans з усією інформацією про проведені онлайн платежі під час оплати замовлень, моніторингова інформація надана службовим сервісом власної розробки про стан сервісів, кількість ресурсів споживану кожним сервісом, стан серверу на якому знаходяться сервіси, також перегляд статусу усієї системи та усіх записів логування з можливістю зручного фільтрування по часу, назві сервісу, типу події та іншими критеріями для гнучкого та ефективного аналізу системних змін чи процесів у режимі реального часу.

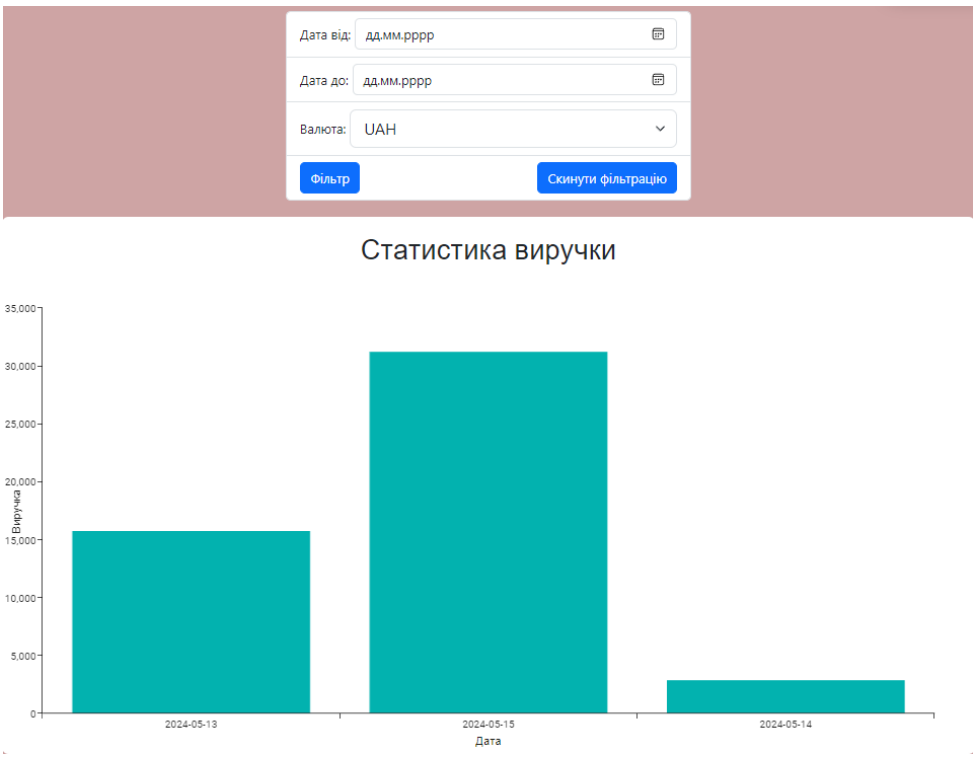

Рисунок 5.4.20 – Сторінка перегляду статистики прибутку

Дана сторінка підтримує вибірку за конкретний часовий проміжок, а також конвертацію UAH / USD.

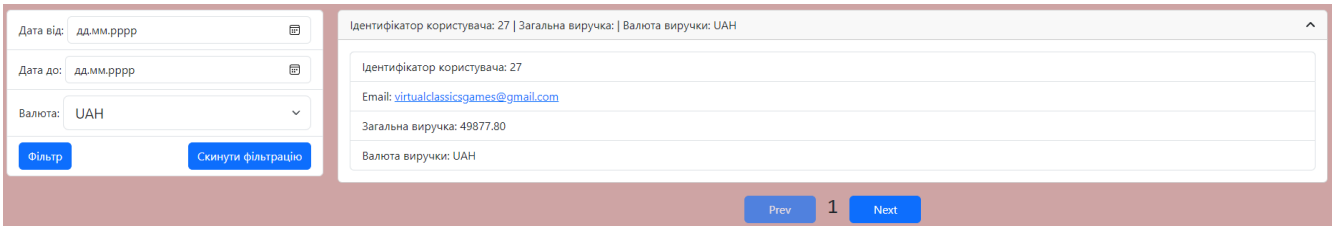

Рисунок 5.4.21 – Сторінка перегляду прибуткових користувачів

Аналогічно до попередньої сторінки є можливість конвертації, також фільтрація по часовому проміжку. Остання сторінка дозволяє менеджеру відвідати побічні сторінки (рис. 5.4.22) за допомогою спеціальних кнопок, які переадресують менеджера на відповідні сторінки окремо від системної.

Рисунок 5.4.22 – Сторінка для переходу до сторонніх моніторингів та інтеграцій

Розглянемо спочатку панель керування, що надає платіжний провайдер для своїх користувачів (рис. 5.4.23)

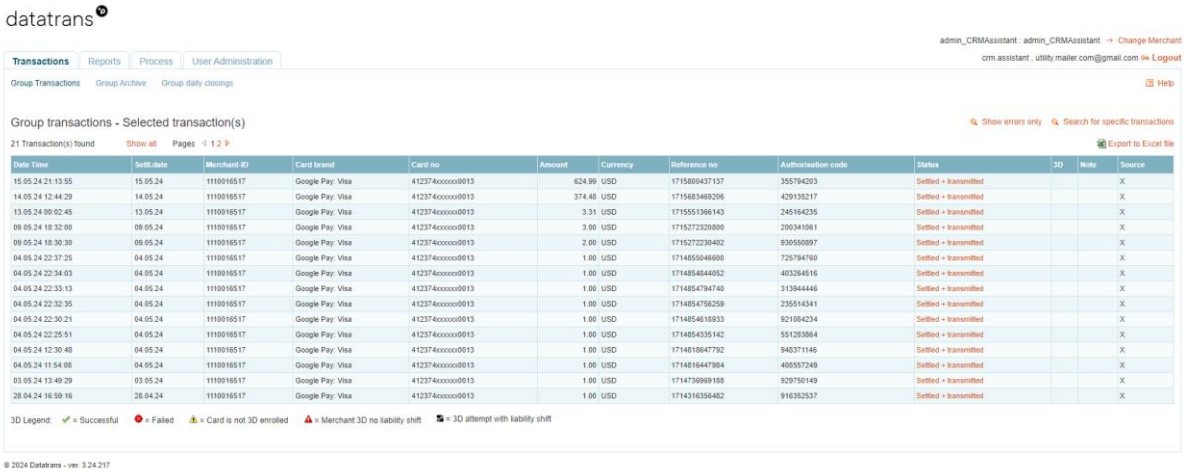

Рисунок 5.4.23 – Сторінка API платіжного провайдера

Дана реалізація надає можливість перевірки та керування платежами у більш вільній та деталізованій формі, також є експорт різних даних, фільтрація та інші корисні функції. Також наявні інтеграції з Kibana (рис. 5.4.24), Netflix Eureka (рис. 5.4.25) та Spring Boot Admin (рис. 5.4.26).

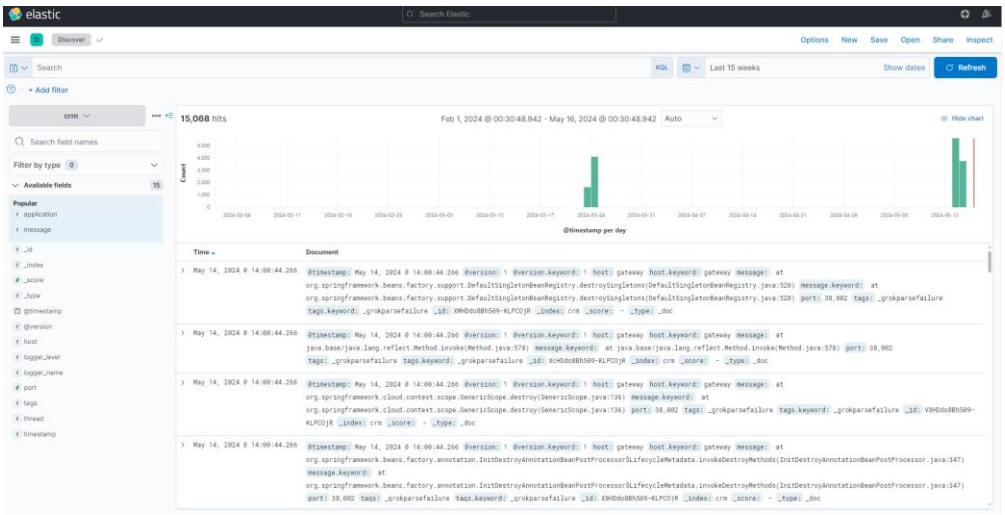

Рисунок 5.4.24 – Сторінка відображення записів логування

Дана інтеграція дозволяє слідкувати за останніми подіями в кожному з сервісів застосунку, у випадку виникнення помилок, чи інших проблем, використання системних записів логування може допомогти у вирішенні цих питань та уникнути їх повторення у майбутньому.

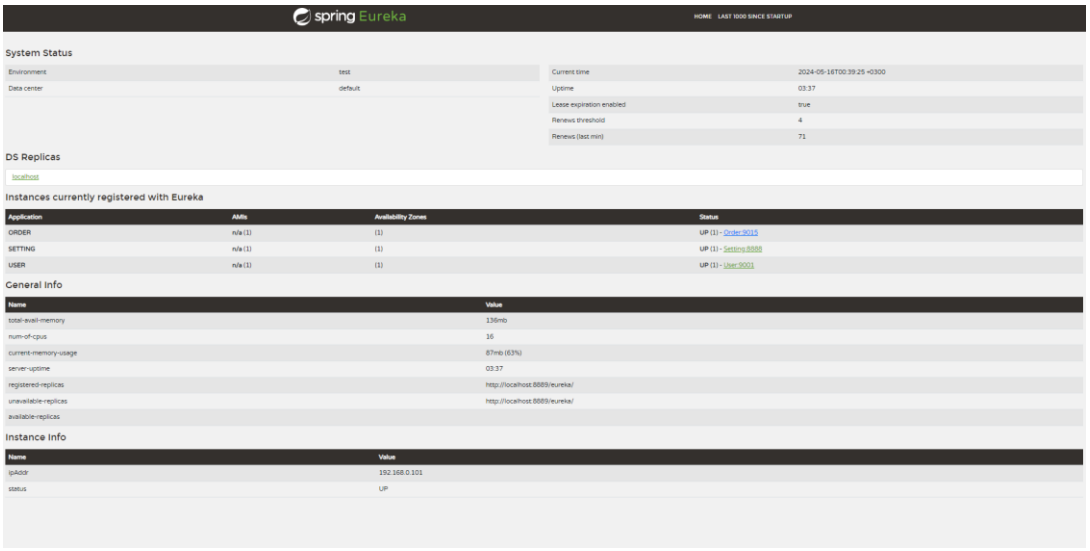

Рисунок 5.4.25 – Сторінка Netflix Eureka

Ця панель керування дозволяє отримувати відомості про стан підключених сервісів, конфігурацію сервісу визначення інших застосунків, тощо.

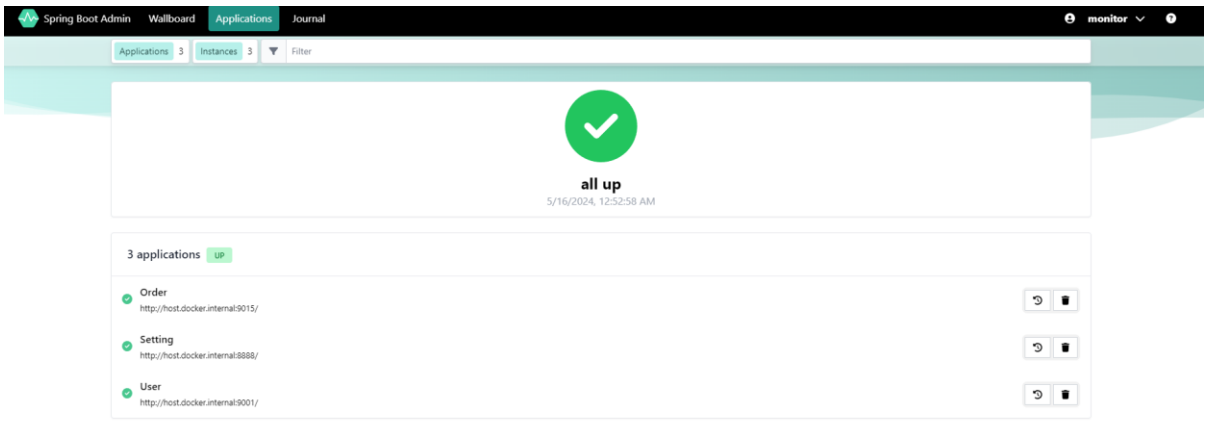

Рисунок 5.4.26 – Сторінка Spring Boot Admin

Ця інтеграція також забезпечує візуалізацію різноманітних метрик та показників у зручному графічному вигляді. Користувач може переглядати статистику з навантаження на процесор, обсягу використаної оперативної пам'яті та інші параметри, що допомагають оцінити продуктивність та ефективність системи. Крім того, важливою функцією є здатність моніторити та аналізувати журнали змін, що дозволяє оперативно виявляти проблеми та вирішувати їх. Таким чином, ця інтеграція робить процес моніторингу та аналізу роботи системи більш прозорим та ефективним для розробників та адміністраторів.

Інтеграція надає повний інструментарій для моніторингу, аналізу та управління e-commerce системою. Інтерфейс включає всі необхідні функції для зручного користування, такі як моніторинг стану сервісів і серверів, доступ до змінних оточення, журналів змін, кешу, а також можливість зміни профілю і взаємодії з контролерами та об'єктами. За допомогою цієї інтеграції користувач може ефективно виконувати аналіз продуктивності, виявляти та вирішувати проблеми, а також забезпечувати безперебійну роботу системи електронної комерції. Усі компоненти системи були успішно протестовані мануальним шляхом та з залученням Postman для тестування серверної частини.

ВИСНОВКИ

У ході виконання дипломної роботи було розглянуто та проаналізовано існуючі системи менеджменту та моніторингу клієнтської взаємодії, на основі яких було побудовано оптимальні вимоги та стратегію для розробки програмного забезпечення. Дослідження охопило кілька ключових аспектів, включаючи аналіз поточних проблем електронної комерційної діяльності, огляд існуючих програмних систем, а також детальний опис засобів розробки та програмної реалізації системи.

У рамках дипломної роботи було проведено комплексний аналіз існуючих рішень в сфері електронної комерції, що дозволило визначити основні тенденції та потреби сучасного ринку. На основі отриманих даних було розроблено архітектуру системи з використанням мікросервісів, що забезпечує гнучкість і масштабованість. Проєктування структурної схеми баз даних забезпечило ефективне зберігання та управління даними кожного окремого сервісу.

Успішно розроблено як серверну частину, так і інтерфейс користувача, що забезпечує зручну та інтуїтивно зрозумілу взаємодію з системою. Проведено всебічне тестування системи, яке підтвердило відповідність розробленого функціоналу всім заданим вимогам. Особливу увагу було приділено розробці інтерфейсу застосунку, який виконано у максимально простій та зрозумілій формі для користувача, що сприяє зручності в експлуатації.

Крім того, для полегшення впровадження системи було розроблено детальну інструкцію для її запуску, що забезпечує безперешкодний старт та ефективне використання розробленого програмного забезпечення.

Отже, виконана дипломна робота дозволила набути цінних знань та навичок у розробці програмних систем для менеджменту, аналізу та моніторингу клієнтської взаємодії. Отримані результати можуть бути використані для подальшого вдосконалення системи та її впровадження у реальну бізнес-практику, що сприятиме підвищенню ефективності управління та покращенню взаємодії з клієнтами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microservices Architecture – Introduction and best practices. Amazon Microservices – веб-сайт. URL: https://aws.amazon.com/microservices (дата звернення 21.05.2024)
2. JWT – JSON Web Token Introduction and Documentation. JWT: веб-сайт. URL: https://jwt.io/introduction/ (дата звернення 21.05.2024)
3. Datatrans API – Integration and documentation. Datatrans: веб-сайт. URL: https://docs.datatrans.ch/docs (дата звернення 21.05.2024)
4. Nova Post API – Документація для інтеграції з Новою Поштою. Нова Пошта: веб-сайт. URL: https://developers.novaposhta.ua/documentation (дата звернення 21.05.2024)
5. Spring Boot Email Integration – Sending emails using Gmail with Spring Boot. Spring: веб-сайт. URL: https://spring.io/guides/gs/spring-boot-email/ (дата звернення 21.05.2024)
6. Telegram Bots – Creating and managing Telegram bots. Telegram Bots: веб-сайт. URL: https://core.telegram.org/bots (дата звернення 21.05.2024)
7. Web Application Architecture – Detailed Explanation. Web Application Architecture: веб-сайт. URL: https://www.interviewbit.com/blog/web-application-architecture (дата звернення 15.05.2024)
8. Microservices – Architectural style for structuring an application as a collection of services. Microservices: веб-сайт. URL: https://martinfowler.com/articles/microservices.html (дата звернення 15.05.2024)
9. AMQP – Advanced Message Queuing Protocol documentation. AMQP: веб-сайт. URL: https://www.amqp.org/resources/specifications (дата звернення 21.05.2024)
10. Apache Kafka – Documentation and Resources. Apache Kafka: веб-сайт. URL: https://kafka.apache.org/documentation/ (дата звернення 21.05.2024)

11. Introduction to Redis. Redis: веб-сайт. URL: https://www.tutorialspoint.com/redis/index.htm (дата звернення 18.05.2024)
12. Elasticsearch – Distributed search and analytics engine. Elasticsearch: веб-сайт. URL: https://www.elast[ic.co/elasticsearch/ (дата звернення 21.05.2024)
13. NoSQL Databases – Overview and Documentation. NoSQL Databases: веб-сайт. URL: https://www.mongodb.com/nosql-explained (дата звернення 21.05.2024)
14. WebSocket – API Documentation and Overview. MDN Web Docs: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (дата звернення 21.05.2024)
15. CRM Software – Customer Relationship Management solutions. Salesforce CRM – веб-сайт. URL: https://www.salesforce.com/crm/what-is-crm/ (дата звернення 16.05.2024)
16. E-commerce – Overview and Best Practices. Shopify: веб-сайт. URL: https://www.shopify.com/enterprise/what-is-ecommerce (дата звернення 21.05.2024)
17. HTML & CSS – Building blocks of the web. HTML & CSS: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення 20.05.2024)
18. CSS – Cascading Style Sheets. CSS: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення 20.05.2024)
19. React – A JavaScript library for building user interfaces. React: веб-сайт. URL: https://reactjs.org (дата звернення 13.05.2024)
20. JavaScript – The programming language of the web. JavaScript: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 20.05.2024)
21. Spring Boot – Spring framework for rapid application development. Spring Boot: веб-сайт. URL: https://spring.io/projects/spring-boot (дата звернення 13.05.2024)
22. Spring Cloud – Tools for building cloud-native applications. Spring Cloud: веб-

сайт. URL: https://spring.io/projects/spring-cloud (дата звернення 13.05.2024)

23. Hibernate – A framework for Java applications. Hibernate ORM: веб-сайт. URL: https://hibernate.org/orm/ (дата звернення 20.05.2024)
24. Introduction to PostgreSQL. PostgreSQL: веб-сайт. URL: https://www.postgresql.org/ (дата звернення 18.05.2024)
25. Spring Cloud Netflix – Service discovery and routing. Spring Cloud Netflix: веб-сайт. URL: https://cloud.spring.io/spring-cloud-netflix (дата звернення 13.05.2024)
26. Building Microservices with Spring Boot. Spring IO: веб-сайт. URL: https://spring.io/microservices (дата звернення 18.05.2024)
27. Docker: веб-сайт. URL: https://www.docker.com (дата звернення 19.05.2024)
28. Docker Compose – Defining and running multi-container Docker applications. Docker Compose: веб-сайт. URL: https://docs.docker.com/compose/ (дата звернення 20.05.2024)
29. How to Create Use Case, Class, and Other Diagrams. Visualising with UML: веб-сайт. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml (дата звернення 19.05.2024)
30. Cloudinary API Documentation. Cloudinary: веб-сайт. URL: https://cloudinary.com/documentation (дата звернення 21.05.2024)
31. Logstash – Ingest, transform, and send your data. Logstash: веб-сайт. URL: https://www.elastic.co/logstash (дата звернення 21.05.2024)
32. Kibana – Explore, visualize, and analyze your data. Kibana: веб-сайт. URL: https://www.elastic.co/kibana (дата звернення 21.05.2024)
33. DevOps – Introduction to Docker. Docker: веб-сайт. URL: https://www.tutorialspoint.com/docker/index.htm (дата звернення 21.05.2024)

Додаток А

Програмне забезпечення системи менеджменту бізнес процесів, моніторингу та аналізу відносин з клієнтом

Текст програми

Аркушів 12

Київ – 2024

OrderService.java

```
/**
 * Order Processing Logic
 * For each unique product item new order will be created
 * Reasons: customer can select different products from different
vendors, with different delivery settings, for correct processing they
are going to be split
 */
@Transactional
@CachePut(cacheNames = "ordersCache", key = "#result.id")
public List<Order> saveOrder(@Valid CreateOrderDTO createOrderDTO) {
    if (createOrderDTO.getOrderedProducts() == null ||
createOrderDTO.getOrderedProducts().isEmpty()) {
        throw new RuntimeException("Invalid order entity, no ordered
products");
    }

    if (createOrderDTO.getPaymentType() == PaymentType.COD &&
createOrderDTO.isPaid()) {
        throw new DefaultException("Invalid parameters combination,
order payment type can not be {} with order paid status {}",
                createOrderDTO.getPaymentType(), true);
    }

    if (createOrderDTO.getDeliveryServiceType() == null) {

createOrderDTO.setDeliveryServiceType(DeliveryServiceType.NONE);
    }

    if
(userDetailsImplementationService.getUserRoleById(createOrderDTO.getCu
stomerId()) == null) {
        throw new NotFoundException("Customer with ID {} was not
found", createOrderDTO.getCustomerId());
    }

    List<Order> orders = new ArrayList<>();

    for (OrderedProductDTO orderedProductDTO :
createOrderDTO.getOrderedProducts()) {
        Order order = orderMapper.dtoToEntity(createOrderDTO);

        if (!createOrderDTO.isPaid()) {
            order.setStatus(OrderStatus.INITIATED);
        } else {
            order.setStatus(OrderStatus.PAID);
            order.setTransactionId(createOrderDTO.getTransactionId());
        }

        order = save(order);

        Pair<Product, Integer> orderedProduct =
productService.getProductOrderingPair(orderedProductDTO);
```

```
        productService.orderProduct(order.getId(),
orderedProduct.getFirst().getId(), orderedProductDTO.getAmount());

        BigDecimal productCost = orderedProduct.getFirst().getCost()

.multiply(BigDecimal.valueOf(orderedProduct.getSecond()));

        if (createOrderDTO.getDeliveryServiceType() !=
DeliveryServiceType.NONE) {
            BigDecimal deliveryCost =
                    deliveryService.getDeliveryCost(orderedProduct,
createOrderDTO.getDeliveryServiceType(),
createOrderDTO.getOrderShipmentAddress());
            order.setDeliveryCost(deliveryCost);
        }

        order.setOrderedProductCost(productCost);

        if (createOrderDTO.getDeliveryServiceType() !=
DeliveryServiceType.NONE) {
            OrderShipmentAddressDTO orderShipmentAddressDTO =
createOrderDTO.getOrderShipmentAddress();

orderShipmentAddressDTO.setCustomerId(order.getCustomerId());
            orderShipmentAddressDTO.setOrderId(order.getId());

            OrderShipmentAddress orderShipmentAddress =
orderShipmentAddressService.saveOrderShipmentAddress(orderShipmentAddr
essDTO);
            order.setShipmentAddress(orderShipmentAddress);
        }

        ProductSaleStatisticEntry entry = new
ProductSaleStatisticEntry();

        entry.setProductId(orderedProduct.getFirst().getProductId());
        entry.setItemsSold(orderedProduct.getSecond());
        entry.setTotalCost(productCost);

entry.setCategoryId(orderedProduct.getFirst().getCategoryId());

        productService.addProductSaleStatisticEntry(entry);

        order = save(order);

        orders.add(order);
    }

    return orders;
}
@Transactional
@CachePut(cacheNames = "ordersCache", key = "#result.id")
```

```
public Order updateOrder(@Valid UpdateOrderDTO updateOrderDTO) {
    Order order = findOrderById(updateOrderDTO.getOrderId());

    if (updateOrderDTO.getNextStatus() == order.getStatus()) {
        log.debug("Order status update is redundant, new status {} is
already registered", updateOrderDTO.getNextStatus());
        return order;
    }

    Order copy = copyOrderModelForHistoryEntryComparison(order);

    if (OrderStatus.checkTransitionRule(order.getStatus(),
updateOrderDTO.getNextStatus(), order.getPaymentType())) {
        order.setStatus(updateOrderDTO.getNextStatus());
        if (updateOrderDTO.getNextStatus() == OrderStatus.PAID) {
            order.setPaid(true);
        }

        if (order.getStatus() == OrderStatus.ASSIGNED_TO_OPERATOR) {
            List<UserDetailsModel> managers =
userDetailsImplementationService.getUsersByRole(UserSecurityRole.ROLE_
MANAGER);

            if (managers != null && !managers.isEmpty()) {
                List<Long> uwu =
managers.stream().map(UserDetailsModel::getId).toList();

                if (managers.size() == 1) {

order.setProcessingOperatorId(managers.get(0).getId());
                } else {
                    List<Long> managerIds =
orderRepository.getListOfManagerIds(uwu);

                    List<Long> availableManager = new
ArrayList<>(uwu);
                    availableManager.removeAll(managerIds);

                    if (!availableManager.isEmpty()) {

order.setProcessingOperatorId(availableManager.get(0));
                    } else {

order.setProcessingOperatorId(managerIds.get(0));
                    }
                }
            }
        }

        order = save(order);

        try {
            UserDetailsModel userDetailsModel =
```

```
userDetailsImplementationService.getUserById(order.getCustomerId());

            if (userDetailsModel != null) {
                Order finalOrder = order;

                Runnable task = () -> {
                    MessageModel messageModel = new MessageModel();

                    messageModel.setSender("CRM");

                    if (userDetailsModel.getEmail() != null) {

messageModel.setMessagePlatform(MessageModel.MessagePlatform.EMAIL);

messageModel.setMessageType(MessageModel.MessageType.PLAIN_TEXT);

messageModel.setReceiver(userDetailsModel.getEmail());
                    } else if (userDetailsModel.getTelegramUsername()
!= null) {

messageModel.setMessagePlatform(MessageModel.MessagePlatform.TELEGRAM)
;

messageModel.setMessageType(MessageModel.MessageType.PLAIN_TEXT);

messageModel.setReceiver(userDetailsModel.getTelegramUsername());
                    }

                    messageModel.setSubject("Notification from CRM");
                    messageModel.setContent("Your order " +
finalOrder.getNumber() + " was updated to status " +
finalOrder.getStatus().name().replaceAll("_", " "));

                    messageProducer.produce(messageModel);
                };

                executorService.schedule(task, 3, TimeUnit.SECONDS);
            }
        } catch (Exception e) {
            log.error(e.getMessage(), e);
        }
    } else {
        throw new ActionRestrictedException(String.format("Order can
not be transferred to the status named: %s",
updateOrderDTO.getNextStatus()));
    }

    addOrderHistoryEntry(copy, order);

    return order;
}
```

OrderController.java

```
@RestController
@RequestMapping("/api/v1/orders")
@RequiredArgsConstructor
public class OrderController {
    private final OrderService orderService;

    @PostMapping
    public Response<?> getOrders(@RequestParam(required = false)
Integer page,
                                 @RequestParam(required = false)
Integer size,
                                 @RequestParam(required = false)
String sortBy,
                                 @RequestParam(required = false)
String direction,
                                 @RequestBody(required = false)
OrderFilter orderFilter) {
        return Response.ok(orderService.getOrders(orderFilter, page,
size, sortBy, direction));
    }

    @PostMapping("/complete")
    public Response<?> getCompleteOrdersData(@RequestParam(required =
false) Integer page,
                                 @RequestParam(required = false)
Integer size,
                                 @RequestParam(required = false)
String sortBy,
                                 @RequestParam(required = false)
String direction,
                                 @RequestBody(required = false)
OrderFilter orderFilter) {
        return
Response.ok(orderService.getCustomersCompleteOrdersData(orderFilter,
page, size, sortBy, direction));
    }

    @PostMapping("/save")
    public Response<?> saveOrder(@RequestBody CreateOrderDTO
createOrderDTO) {
        return Response.ok(orderService.saveOrder(createOrderDTO));
    }

    @PutMapping("/update")
    public Response<?> updateOrder(@RequestBody UpdateOrderDTO
updateOrderDTO) {
        return Response.ok(orderService.updateOrder(updateOrderDTO));
    }

    @PostMapping("/history")
    public Response<?> getOrderHistory(@RequestParam(required = false)
Integer page,
                                       @RequestParam(required = false)
```

```
Integer size,
                                        @RequestParam(required = false)
String sortBy,
                                        @RequestParam(required = false)
String direction,
                                        @RequestBody(required = false)
OrderFilter orderFilter) {
        return Response.ok(orderService.getOrderHistory(page, size,
sortBy, direction, orderFilter));
    }

    @PostMapping("/export")
    public void exportOrderData(@RequestBody(required = false)
OrderFilter orderFilter,
                                @RequestParam(required = false) String
filename,
                                HttpServletResponse
httpServletResponse) {
        orderService.exportOrders(orderFilter, filename,
httpServletResponse);
    }

    @PostMapping("/export/history")
    public void exportOrderHistoryData(@RequestBody(required = false)
OrderFilter orderFilter,
                                        @RequestParam(required = false)
String filename,
                                        HttpServletResponse
httpServletResponse) {
        orderService.exportOrderHistory(orderFilter, filename,
httpServletResponse);
    }
}
```

TransactionService.java

```
@Service
@RequiredArgsConstructor
@Slf4j
@Validated
public class TransactionService {
    @PersistenceContext(unitName = "database")
    private final EntityManager entityManager;
    private final DataTransTransactionService
dataTransTransactionService;
    private final TransactionRepository transactionRepository;
    private final DataTransConfiguration dataTransConfiguration;
    private final WebSocketService webSocketService;
    private final TransactionMapper transactionMapper;
    private final OrderRepository orderRepository;

    public List<Transaction> getTransactions(@Nullable @Valid
TransactionFilter transactionFilter,
                                             @DefaultNumberValue
Integer page,
```

```
@DefaultNumberValue(number = 100) Integer size,

@DefaultStringValue(string = "issuedAt") String sortBy,

@DefaultStringValue(string = "DESC") String direction) {
        CriteriaBuilder criteriaBuilder =
entityManager.getCriteriaBuilder();
        CriteriaQuery<Transaction> query =
criteriaBuilder.createQuery(Transaction.class);
        Root<Transaction> root = query.from(Transaction.class);

        List<Predicate> predicates = new ArrayList<>();

        if (transactionFilter != null) {
            if (transactionFilter.getIssuedAtFrom() != null) {

predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get(Transacti
on.Fields.issuedAt), transactionFilter.getIssuedAtFrom()));
            }
            if (transactionFilter.getIssuedAtTo() != null) {

predicates.add(criteriaBuilder.lessThanOrEqualTo(root.get(Transaction.
Fields.issuedAt), transactionFilter.getIssuedAtTo()));
            }
        }

        query.where(predicates.toArray(Predicate[]::new));

        if (direction != null) {
            switch (Sort.Direction.valueOf(direction)) {
                case DESC ->
query.orderBy(criteriaBuilder.desc(root.get(sortBy)));
                case ASC ->
query.orderBy(criteriaBuilder.asc(root.get(sortBy)));
            }
        } else {

query.orderBy(criteriaBuilder.desc(root.get(Transaction.Fields.issuedA
t)));
        }

        return entityManager.createQuery(query).setFirstResult((page -
1) * size).setMaxResults(size).getResultList();
    }

    private Transaction
processIncomingTransactionData(TransactionInitiativeDTO
transactionInitiative) {
        Transaction transaction = new Transaction();

        transaction.setId(UUID.randomUUID());
```

```
transaction.setCustomerId(transactionInitiative.getCustomerId());
        transaction.setAmount(transactionInitiative.getAmount());

transaction.setSourceCurrency(transactionInitiative.getCurrency());

transaction.setAcquiringCurrency(Currency.valueOf(dataTransConfigurati
on.getAcquiringCurrency()));

transaction.setReference(transactionInitiative.getReference());
        transaction.setStatus("CREATED");
        transaction.setIssuedAt(TimeUtil.getCurrentDateTime());

transaction.setMerchantId(dataTransConfiguration.getMerchantId());

        return transaction;
    }

    @Transactional
    public Transaction processIncomingTransaction(@Valid
TransactionInitiativeDTO transactionInitiativeDTO) {
        Transaction transaction =
processIncomingTransactionData(transactionInitiativeDTO);

        TransactionAuthorizationAPIModel
transactionAuthorizationAPIModel =
dataTransTransactionService.processTransaction(transactionInitiativeDT
O);
        TransactionStateMessage transactionStateMessage;

        if
((!transactionAuthorizationAPIModel.getAuthorizationBody().getStatus()
.equalsIgnoreCase("error")
                && transactionAuthorizationAPIModel.getError() ==
null)
                || (transactionAuthorizationAPIModel.getError() !=
null && transactionAuthorizationAPIModel.getError().getErrorMessage()
== null)) {
            AuthorizationResponse authorizationResponse =
transactionAuthorizationAPIModel.getAuthorizationBody().getTransaction
().getAuthorizationResponse();

transaction.setStatus(transactionAuthorizationAPIModel.getAuthorizatio
nBody().getStatus().toUpperCase());

transaction.setTransactionId(authorizationResponse.getUppTransactionId
());

transaction.setAuthorizationCode(authorizationResponse.getAuthorizatio
nCode());

transaction.setAcquireAuthorizationCode(authorizationResponse.getAcqAu
thorizationCode());
```

```
transaction.setTransactionProcessingCountry(authorizationResponse.getR
eturnCustomerCountry());

transaction.setTransactionType(transactionAuthorizationAPIModel.getAut
horizationBody().getTransaction().getAuthorizationRequest().getRequest
Type());

transaction.setTransactionAlias(authorizationResponse.getAliasCC());
            transaction.setExpiration(String.format("%s/%s",
authorizationResponse.getExpirationMonth(),
authorizationResponse.getExpirationYear()));
            transaction.setPan(authorizationResponse.getPan());

            transactionStateMessage =
transactionMapper.modelToStateMessage(transaction);
            transactionStateMessage.setAuthorized(true);

            TransactionPaymentAPIModel transactionPaymentAPIModel =
dataTransTransactionService.settleTransaction(transactionAuthorization
APIModel);

            if
(!transactionAuthorizationAPIModel.getAuthorizationBody().getStatus().
equals("ERROR")
                    && transactionAuthorizationAPIModel.getError() ==
null
                    ||
transactionAuthorizationAPIModel.getError().getErrorMessage() == null)
{
                if (transactionPaymentAPIModel.getError() == null ||
transactionPaymentAPIModel.getError().getErrorMessage() == null) {

transaction.setSettledAt(TimeUtil.getCurrentDateTime());
                    transactionStateMessage.setSettled(true);
                } else {
                    TransactionProcessingError
transactionProcessingError = transactionPaymentAPIModel.getError();
                    transactionStateMessage.setSettled(false);

log.error(transactionProcessingError.getErrorMessage(),
transactionProcessingError);
                }
            }
        } else {
            TransactionProcessingError transactionProcessingError =
transactionAuthorizationAPIModel.getError();
            if (transactionProcessingError != null) {

log.error(transactionProcessingError.getErrorMessage(),
transactionProcessingError);
            }

            transactionStateMessage =
```

```
transactionMapper.modelToStateMessage(transaction);
            transactionStateMessage.setAuthorized(false);
        }


webSocketService.sendMessage(WebSocketTopics.TRANSACTION_STATE.getTopi
c() + "/" + transactionStateMessage.getCustomerId(),
transactionStateMessage);

        return transactionRepository.saveAndFlush(transaction);
    }

    @Transactional
    public Transaction addManualPayment(@Valid
TransactionManualInitiativeModel transactionManualInitiativeModel) {
        BigInteger num;

        try {
            num = new
BigInteger(transactionManualInitiativeModel.getNumber());
        } catch (NumberFormatException e) {
            log.error(e.getMessage(), e);
            throw new DefaultException("Order can not be found because
num is ivalid {}", transactionManualInitiativeModel.getNumber());
        }

        Order order = orderRepository.findByNumber(num)
                .orElseThrow(() -> new NotFoundException("Order with
number {} was not found",
transactionManualInitiativeModel.getNumber()));

        if (order.getOrderedProductCost().doubleValue() >
transactionManualInitiativeModel.getPaymentAmount().doubleValue()) {
            throw new ActionRestrictedException("Payment amount can
not be less than cost of the order");
        }

        if (order.getTransactionId() != null) {
            throw new ActionRestrictedException("Order with number {}
is already paid", num);
        }

        Transaction transaction = new Transaction();

        transaction.setId(UUID.randomUUID());
        transaction.setCustomerId(order.getCustomerId());

transaction.setAmount(transactionManualInitiativeModel.getPaymentAmoun
t());
        transaction.setReference(BigInteger.valueOf(new
Date().getTime() * new Random().nextInt(3, 9) -
                new Random().nextInt(10000) * new
Random().nextInt(333) +
```

```
                new Random().nextInt(33333)));
        if (order.getOrderedProducts() != null &&
!order.getOrderedProducts().isEmpty() &&
order.getOrderedProducts().get(0).getProduct() != null) {

transaction.setSourceCurrency(order.getOrderedProducts().get(0).getPro
duct().getCurrency());

transaction.setAcquiringCurrency(order.getOrderedProducts().get(0).get
Product().getCurrency());
        } else {
            transaction.setSourceCurrency(Currency.USD);
            transaction.setAcquiringCurrency(Currency.USD);
        }
        transaction.setStatus("SETTLED");
        transaction.setIssuedAt(TimeUtil.getCurrentDateTime());
        transaction.setSettledAt(TimeUtil.getCurrentDateTime());
        transaction.setMerchantId("MANUAL");
        transaction.setTransactionId(UUID.randomUUID().toString());

        if (!OrderStatus.completedStatus.contains(order.getStatus()))
{
            order.setStatus(OrderStatus.PAID);
        }

        transaction = transactionRepository.save(transaction);

        order.setTransactionId(transaction.getId());

        orderRepository.save(order);

        return transaction;
    }
}
```

TransactionController.java

```
@RequestMapping("/api/v1/transactions")
@RestController
@RequiredArgsConstructor
public class TransactionController {
    private final TransactionService transactionService;

    @PostMapping
    public Response<?> getTransactions(@RequestBody(required = false)
TransactionFilter transactionFilter,
                                       @RequestParam(required = false)
Integer page,
                                       @RequestParam(required = false)
Integer size,
                                       @RequestParam(required = false)
String sortBy,
                                       @RequestParam(required = false)
String direction) {
        return
```

```
Response.ok(transactionService.getTransactions(transactionFilter,
page, size, sortBy, direction));
    }

    @PostMapping("/initiate")
    public Response<?> initiateTransaction(@RequestBody
TransactionInitiativeDTO transactionInitiativeDTO) {
        return
Response.ok(transactionService.processIncomingTransaction(transactionI
nitiativeDTO));
    }


@PreAuthorize("hasAuthority(T(io.management.ua.utility.models.UserSecu
rityRole).ROLE_MANAGER)")
    @PostMapping("/manual/payment")
    public Response<?> addManualPayment(@RequestBody
TransactionManualInitiativeModel transactionManualInitiativeModel) {
        return
Response.ok(transactionService.addManualPayment(transactionManualIniti
ativeModel));
    }
}
```