НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмний застосунок для отримання послуг у фітнес центрах

Виконав студент IV курсу, групи ІП-з81
(шифр групи)
Гордійчук Владислав Валерійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник асистент кафедри ІПІ, Храмченко М. С. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації доцент, к.т.н., доц., Ліщук К. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент кафедри ІСТ, к.т.н., доц., Сперкач М. О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ –2022

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення комп'ютеризованих систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2022 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Гордійчук Владислав Валерійович
(прізвище, ім'я, по батькові)

1. Тема проєкту Програмний застосунок для отримання послуг у фітнес центрах

керівник проєкту Храмченко Микола Сергійович, асистент кафедри ІПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « 8 » червня 2022 р. №980-с

2. Термін подання студентом проєкту « 19 » червня 2022 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог.
2) Вибір технологій і середовища розробки: засоби розробки, технічні рішення, архітектура програмного забезпечення.
3) Конструювання програмного забезпечення.
4) Аналіз якості та тестування програмного забезпечення.
5) Впровадження та супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Креслення вигляду екранних форм

2) Схема структурна варіантів використань

3) Схема структурна класів програмного забезпечення

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» березня 2022 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2022	
2	Аналіз існуючих методів розв’язання задачі	17.03.2022	
3	Постановка та формалізація задачі	24.03.2022	
4	Розробка інформаційного забезпечення	31.03.2022	
5	Алгоритмізація задачі	07.04.2022	
6	Обґрунтування вибору використаних технічних засобів	14.04.2022	
7	Розробка програмного забезпечення	21.04.2022	
8	Налагодження програми	07.05.2022	
9	Виконання графічних документів	15.05.2022	
10	Оформлення пояснювальної записки	23.05.2022	
11	Подання ДП на попередній захист	06.06.2022	
12	Подання ДП рецензенту	12.06.2022	
13	Подання ДП на основний захист	19.06.2022	

Студент _____ Владислав ГОРДІЙЧУК
(підпис) (ініціали, прізвище)

Керівник _____ Микола ХРАМЧЕНКО
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 27 таблиць, 43 рисунків та 15 джерел – загалом 79 сторінок.

Об'єкт дослідження: мобільний застосунок, завдяки якому можна отримати послуги у фітнес-центрах.

Мета дипломного проєкту: створення мобільного програмного застосунку для створення альтернативного, цифрового досвіду для комунікації користувача та фітнес центру.

В першому розділі було виконано аналіз предметної області, відомих технічних рішень, сформульовано функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення.

У другому розділі розглянуто моделювання програмного забезпечення. Проаналізована безпека даних.

Третій розділ присвячений розгортанню та впровадженню програмного забезпечення.

В додатках указано: технічне завдання, керівництво користувача та графічний матеріал.

КЛЮЧОВІ СЛОВА: iOS, МОБІЛЬНИЙ ДОДАТОК, MVVM, GOOGLE FIREBASE, STRIPE.

ABSTRACT

The explanatory note of the diploma project consists of four sections, contains 27 tables, 43 figures and 15 sources – in total 79 pages.

The object of study: a mobile application that allows you to get services in fitness centers.

The aim of the diploma project: create a mobile software application to provide an alternative and digital experience for communication between the user and the fitness center.

In the first section, the subject area was analyzed, known technical solutions, functional and non-functional requirements for the developed software were formulated.

In the second section, a software modeling was described. Data security is analyzed.

The third section is devoted to software deployment and implementation.

In the annexes, the terms of reference, the user's guide and the graphic material.

KEYWORDS: iOS, MOBILE APP, MVVM, GOOGLE FIREBASE, STRIPE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-8109.045440.01.91	Технічне завдання	10	
3	А4	КПІ.ІП-8109.045440.02.81	Пояснювальна записка	79	
4	А4	КПІ.ІП-8109.045440.03.13	Опис програми	44	
5	А4	КПІ.ІП-8109.045440.04.34	Керівництво користувача	12	
6	А3	КПІ.ІП-8109.045440.05.99	Креслення вигляду екранних форм	1	
7	А3	КПІ.ІП-8109.045440.05.99	Схема структурна варіантів	1	
			використань		
8	А3	КПІ.ІП-8109.045440.05.99	Схема структурна класів	1	
			програмного забезпечення		

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІП-8109.045440.00.90		
Розроб.		Гордійчук В.В.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Храмченко М.С				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІП-з81	
Н.контр.		Ліщук К.І.					
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ ОТРИМАННЯ ПОСЛУГ У ФІТНЕС ЦЕНТРАХ

Технічне завдання

КПІ.ІП-8109.045440.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Владислав ГОРДІЙЧУК

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ ОТРИМАННЯ ПОСЛУГ У ФІТНЕС ЦЕНТРАХ.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного застосунку під платформу iOS «Програмний застосунок для отримання послуг у фітнес центрах» Gean Sports [КПІ.ІП-8109.045440.01.91], котра використовується для пошуку і оплати, з метою надання, послуг у фітнес центрах та призначена для отримання альтернативного, цифрового, клієнтського досвіду.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки Gean Sports є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для вирішення задач пов’язаних з отриманням банківських платежів.

Метою розробки є розширення можливостей надання цифрового клієнтського досвіду фітнес центрами.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

– створювати обліковий запис у системі;
– відображення останніх новин;
– пошук тренувань та/або фітнес центрів;
– оплата доступу до тренувань та/або фітнес центрів;
– генерація унікального QR коду по результату оплати доступу;

4.1.2 Розробку виконати на платформі macOS Monterey.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації за допомогою, призначених для цієї мети, інтерактивних елементів інтерфейсу програмного застосунку.

4.2.2 Передбачити захист від некоректних дій користувача.

4.2.3 Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2 Обслуговування

4.3.3 Обслуговуючий персонал

Немає вимог до обслуговуючого персоналу.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на мобільних пристроях під керуванням операційної системи iOS.

4.4.2 Мінімальна конфігурація технічних засобів:

4.4.2.1 Версія операційної системи ..... iOS 15.0 і вище.

4.5 Вимоги до інформаційної та програмної сумісності

4.5.1 Програмне забезпечення повинно працювати під управлінням операційної системи сімейства iOS.

4.5.2 Вхідні дані повинні бути представлені в наступному форматі: взаємодія з елементами користувацького інтерфейсу

4.5.3 Результати повинні бути представлені в наступному форматі: віртуальний інтерфейс користувача.

4.5.4 Програмне забезпечення повинно бути створено у середовищі розробці xCode на мові програмування Swift, використовуючи фреймворк SwiftUI.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8 Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2 Програмне забезпечення повинно мати довідникову систему.

5.3 У склад супроводжувальної документації повинні входити наступні документи:

5.3.1 Пояснювальна записка не менше ніж на 55 аркушах формату А4 (без додатків 5.3.2 - 5.3.3).

5.3.2 Технічне завдання.

5.3.3 Керівництво користувача.

5.4 Графічна частина повинна бути виконана не менше ніж на 3 аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1 Креслення вигляду екранних форм.

5.4.2 Схема структурна варіантів використання

5.4.3 Схема структурна класів програмного забезпечення

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	10.03.2022	
2.	Розробка технічного завдання	17.03.2022	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	31.03.2022	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	14.04.2022	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	21.04.2022	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	07.05.2022	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	15.05.2022	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	15.05.2022	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05.2022	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Програмний застосунок для отримання послуг у фітнес центрах

КПІ.ІП-8109.045440.02.81

Київ – 2022

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс
SDK	–	Software development kit
CRUD	–	Операції Create, Read, Update та Delete
ОС	–	Операційна система.

ВСТУП

У сучасному світі людину супроводжує безліч проблем. Проте, за часи існування прогрес не стояв на місці, і людство зробило величезний стрибок вперед в сфері технологій, що дозволило змінити спосіб взаємодії і вирішення багатьох сучасних проблем.

В теперішній час, коли майже у кожного є доступ до інформації за пару кліків в телефоні, багато компаній і бізнесів також намагаються зловити сучасний тренд і перейти у цифру для кращого користувацького досвіду у вигляді створення сайтів, або мобільних програмних застосунків.

Використання мобільного додатку має безліч переваг. Головним чинником - є доступність: дуже важно знайти людину, у якої може не бути сенсорного телефону. Інший важливий чинник – можливість спілкуватися зі своїм клієнтом, та бути у житті клієнта не лише тоді, коли людина контактує з бізнесом в реальному житті, але також і віртуально – у пристрої, яким людина може скористатись будь якої митті.

На ринку вже представлені безліч програмних застосунків, які вирішують проблеми людей, проте є ще багато сфер, проблеми які можна допомогти вирішити за допомогою програмного застосунку.

Отже, створення програмного застосунку для отримання послуг у фітнес центрах можна вважати ще одним ключем для вирішення сучасних проблем сучасних людей, і який допоможе генерувати більше виручки для бізнесів, які працюють у сфері спортивних центрів.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Електронний торговельний майданчик, або маркетплейс – це сукупність декількох інтернет магазинів, які керуються різними продавцями. Електронно торговельним майданчиком можна назвати будь який Інтернет-ресурс, через який укладаються договори купівлі-продажу між підприємцями та покупцями.

Технічно, першою компанією електронної комерції була Boston Computer Exchange, яка була запущена ще в 1982 році. Це був передусім онлайн-ринок, який обслуговував людей, які хотіли продати свої старі комп'ютери. З приходом Інтернету дебютував інший, більш звичний вид магазину. Book Stacks Unlimited, який відкрив свої віртуальні двері в 1992 році, був онлайн-книжковим магазином, який фактично передував компанії, яка швидко стала синонімом терміну на цілі два роки: Amazon.

Бізнес модель маркетплейсів дуже проста. Власник онлайн-майданчика бере відсоток від продажу за використання його сайту, адже завдяки електронним торговельним майданчикам є можливість об'єднати постачальника різних товарів та послуг зі своїм споживачем у єдиному інформаційно-торговельному просторі. Саме такі майданчики дають можливість підвищити ефективність бізнесу та вийти на цифровий ринок зі своїми продуктами.

1.2 Змістовний опис і аналіз предметної області

Існує два типи маркетплейсів: ті, які продають все, що можна уявити, наприклад Aliexpress (рисунок 1.1) або Craigslist. Там можна знайти косметику, одяг (уживаний і новий), домашніх тварин, меблі та навіть перуки. Інша ж фокусуються на певних речах, наприклад, Airbnb (рисунок 1.2) працює лише з житлом, або Zipcar і Getaround, які працюють у сфері винайму автомобілю.

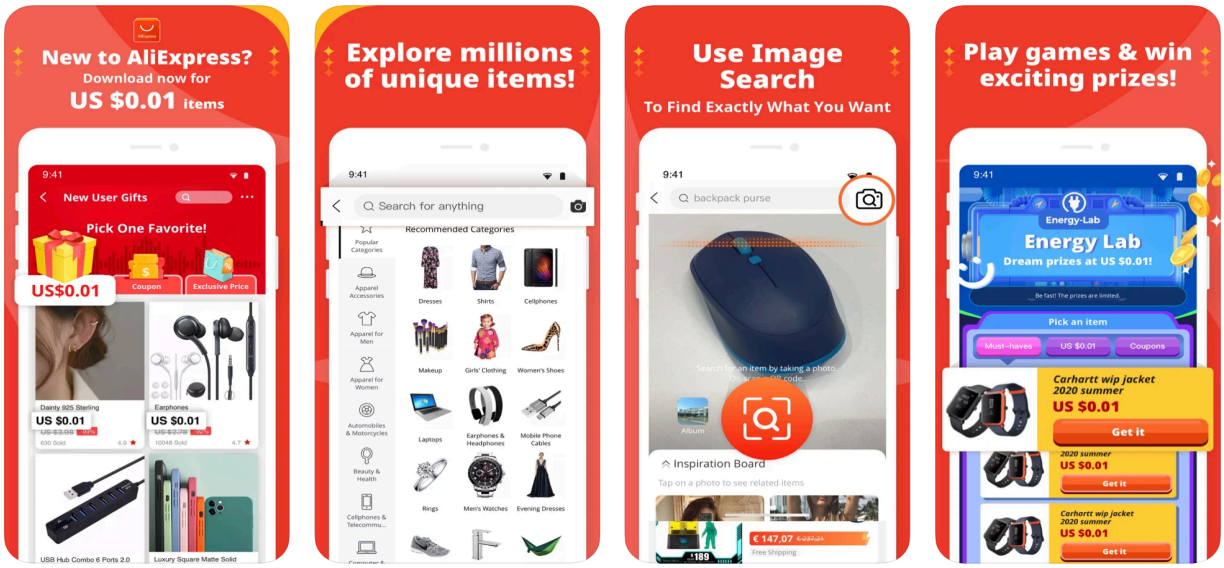


Рисунок 1.1 – Мобільний програмний додаток Aliexpress

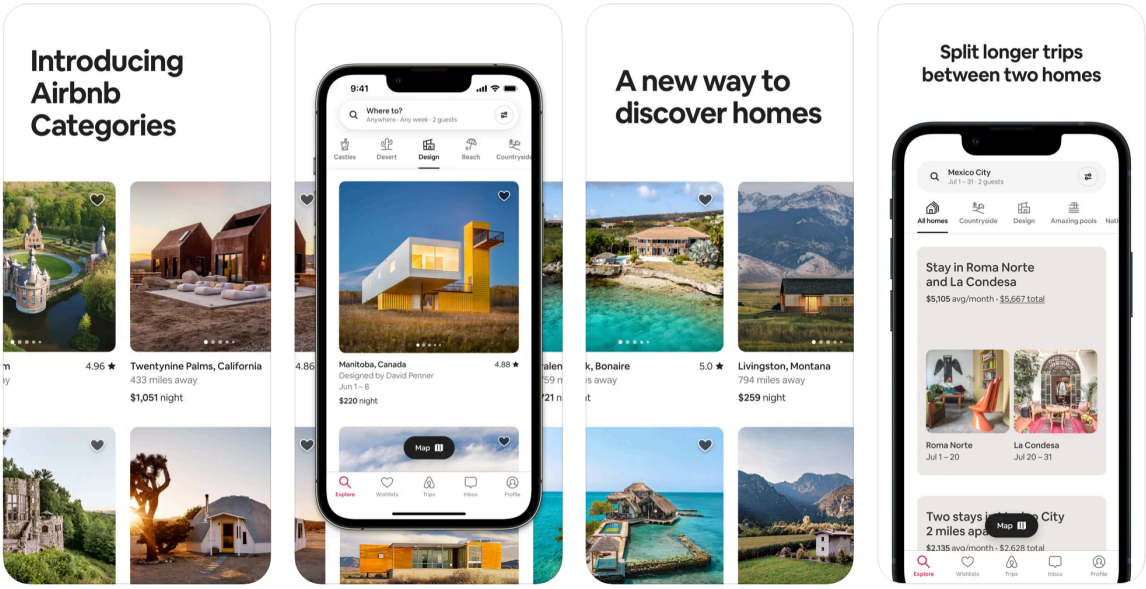


Рисунок 1.2 – Мобільний програмний додаток Airbnb

Як правило, доступ до маркетплейсу представляються за допомогою веб-сайту, або мобільного додатку. Останній має перевагу в офлайн доступі, та кращу оптимізацію на пристрої, під який мобільний додаток створений. Більшість маркетплейсів є безкоштовними для користувачів, а прибуток самого маркетплейсу надходить або від відсотку від продажу, або від пропонування додаткових послуг споживачам і продавцям, або від розміщення платної зовнішньої реклами.

Переважна більшість маркетплейсів зберігають і відображають інформацію про розміщений товар, або послугу та персональні дані продавця, або компанії. Наповнення маркетплейсу продуктами та послугами відбувається за допомогою кабінету продавця, де в багатьох випадках вказується найменування товару, або послуги, фото, опис та ціна. Для кращих результатів продажів – деякі маркетплейси дають можливість просувати товари, або послуги, за певну плату. При знаходженні необхідного товару, покупець може зв’язатися з продавцем за допомогою номеру телефону, пошти, або виконати авторизацію у власний кабінет і залишити заявку с необхідними даними для контакту, або отримання замовлення.

1.3 Аналіз успішних IT-проектів

Розглянемо існуючі мобільні додатки на платформі iOS для отримання послуг у фітнес центрах.

1.3.1 Hussle Flexible Gym Membership

Hussle – це платформа, яка об’єднує безліч фітнес-центрів і з’єднує їх з людьми, які шукають додаткову гнучкість і доступ до кількох фітнес-клубів у Великобританії, щоб відповідати своєму стилю життя. Сервіс допомагає клубам бути конкурентоспроможними в сегменті клієнтів, які вони наразі не можуть завоювати (клієнти, яким не потрібне традиційне членство в тренажерному залі). Використовуючи масштаби партнерської мережі, компанія інвестує у високоефективні національні та місцеві маркетингові кампанії, щоб залучити нових членів і збільшити прибуток для партнерських операторів.

Це сервіс, який є посередником між фітнес центром і клієнтом, і який надає можливість лістингу, отримання детальної інформації, та оплату підписки за доступ в фітнес центр.

Як працює дана платформа (рисунок 1.3):

– Hussle використовує масштаби мережі для інвестування в маркетинг;
– клієнти реагують на маркетинг і роблять покупки;
– клієнти надають свій 8-значний код для доступу до фітнес центрів;
– Hussle отримує комісію і реінвестує в маркетинг.

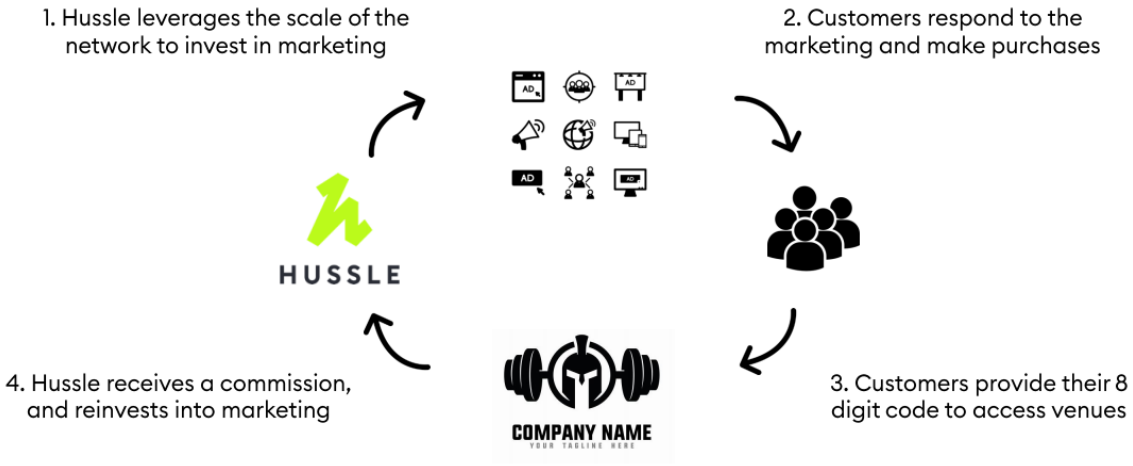


Рисунок 1.3 – Бізнес модель платформи Hussle

Коли користувач вперше запускає додаток – він бачить форму логіну (рисунок 1.4). Проте, йому зовсім не обов'язково авторизуватись у сервісі для початку базового використання, адже можна продовжити використання додатку у лімітованому режимі без авторизації, але з можливістю лістингу фітнес центрів (рисунок 1.5), отримання детальної інформації і цін на підписку (рисунок 1.6).

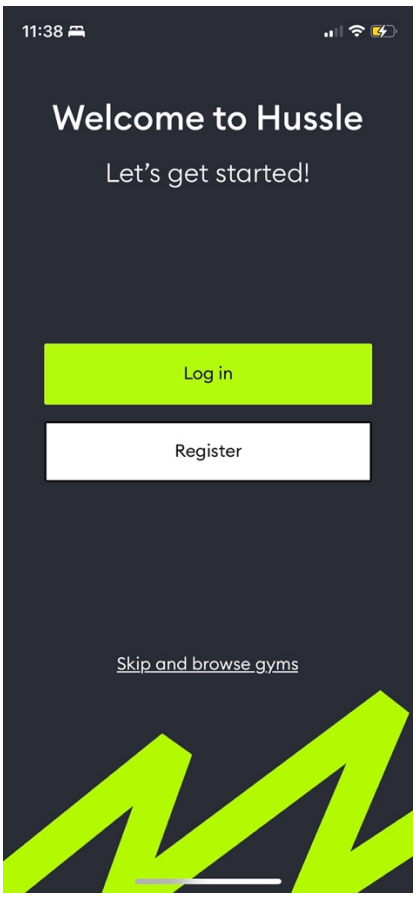


Рисунок 1.4 – Форма логіну в мобільному додатку Hussle

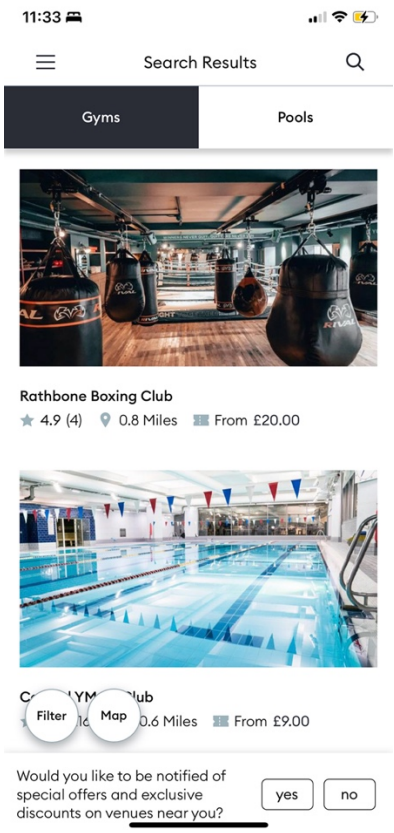


Рисунок 1.5 – Лістинг фітнес центрів в мобільному додатку Hussle

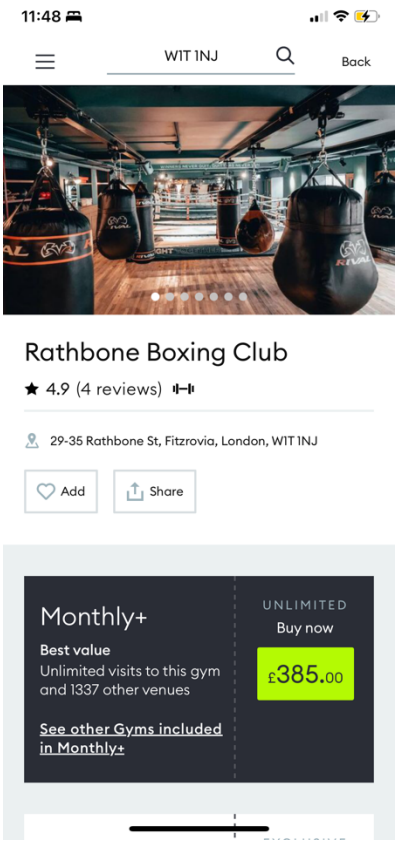


Рисунок 1.6 – Сторінка фітнес центру з деталями і цінами в мобільному додатку Hussle

У користувача є можливість купити обрати підписку орієнтуючись на ціну. Мобільний додаток показує 3 типи підписки з окремими цінами для кожного (рисунок 1.7):

– DayPass – доступ до одного, обраного, фітнес центру на один день;
– Monthly – доступ до одного фітнес центру на один місяць;
– Monthly+ – доступ до багатьох фітнес центрів, які є в одній категорії з обраним фітнес центром.

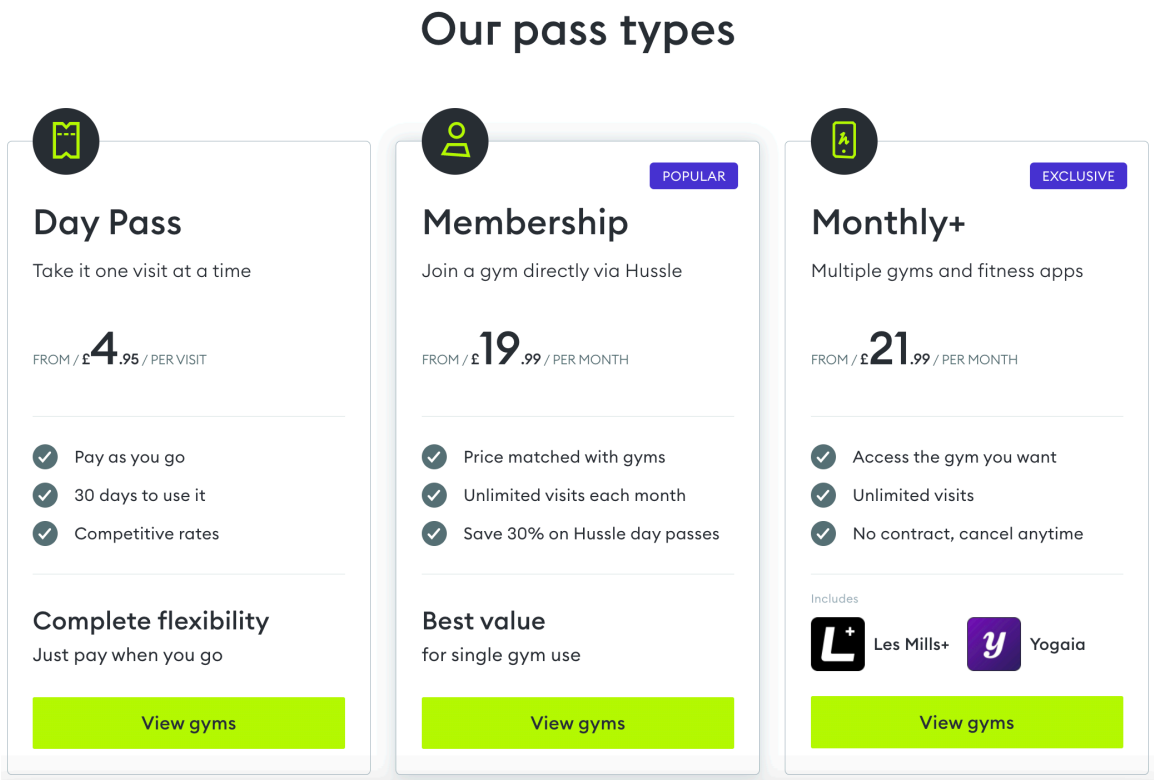


Рисунок 1.7 – Типи підписок в мобільному додатку Hussle

Коли користувач обирає тип підписки – з’являється необхідність у створені нового облікового запису на платформі Hussle, або авторизація в існуючий. Як можна побачити на рисунку 1.8, із доступних типів оплати підписки, користувач може вибрати лише оплату банківською карткою із самостійним заповненням даних банківської карти.

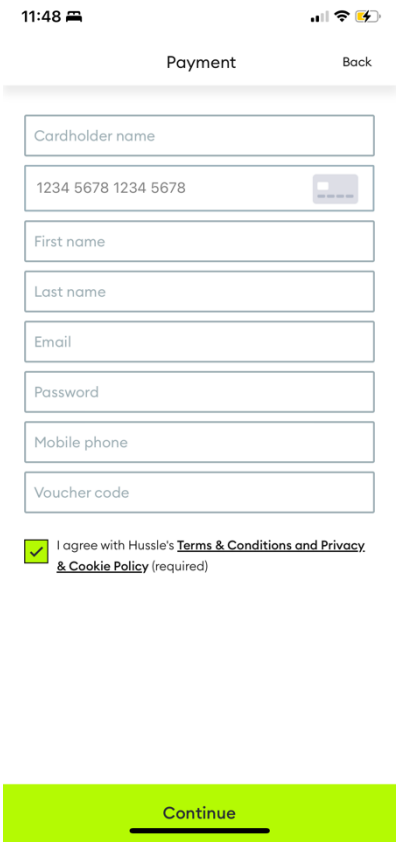


Рисунок 1.8 – Сторінка оплати підписки і створення облікового запису в мобільному додатку Hussle

Варто зазначити, що сервіс є досить перспективним, адже перебуває у стадії росту і розширення власної партнерської мережі фітнес центрів.

1.3.2 [solidcore]

Мобільний застосунок [solidcore] був створений для використання лише в мережевих фітнес центрах [solidcore], які розташовуються у багатьох містах США та налічують базу із 80+ фітнес студій.

Купуючи підписку, користувач отримає доступ до наступних можливостей (рисунок 1.9):

– Ранній доступ до розкладу фітнес студії з можливостю бронювання місця в сесії за 24 години до початку сесії;

– Можливість відмовитись від сесії без додаткової плати;

– Можливість відвідування будь якої фітнес студії мережі без додаткової плати.

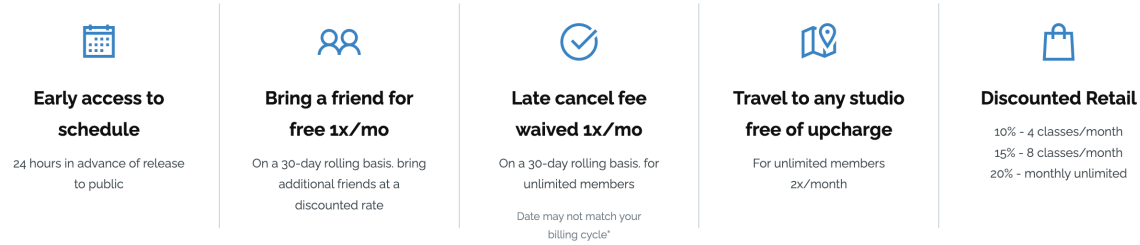


Рисунок 1.9 – Можливості підписки в мобільному додатку [solidcore]

Коли користувач вперше запускає додаток – він одразу побачить форму авторизації (рисунок 1.10). Без авторизації, у користувача не буде можливості використати програмний застосунок навіть на базовому рівні.

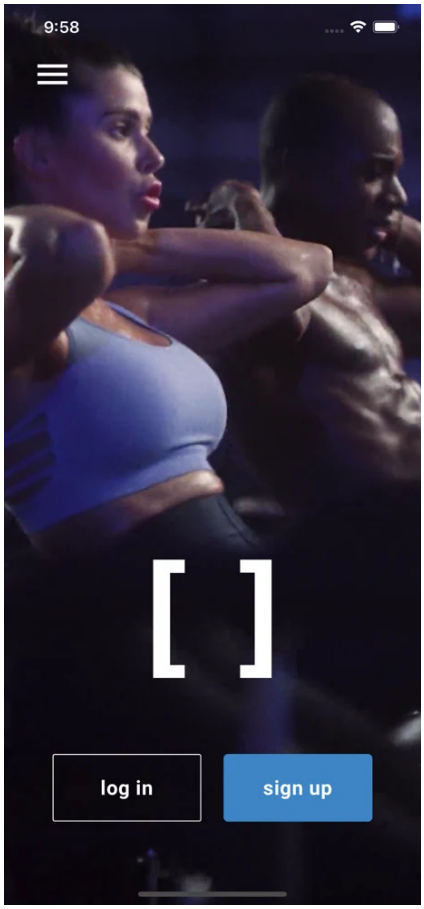


Рисунок 1.10 – Форма авторизації в мобільному додатку [solidcore]

Після авторизації, або реєстрації, користувачу мобільного додатку необхідно обрати бажану підписку (рисунок 1.11). Оплата підписки можлива лише банківською картою із ручним введенням даних банківської карти.

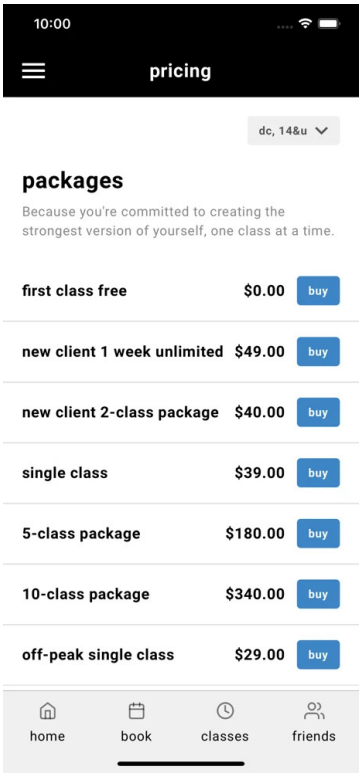


Рисунок 1.10 – Вибір підписки в мобільному додатку [solidcore]

В результаті оплати підписки, користувачу відкривається можливість отримати список сесій (рисунок 1.11), відфільтрувати список (рисунок 1.12), обрати бажану сессію та забронювати свою участь.

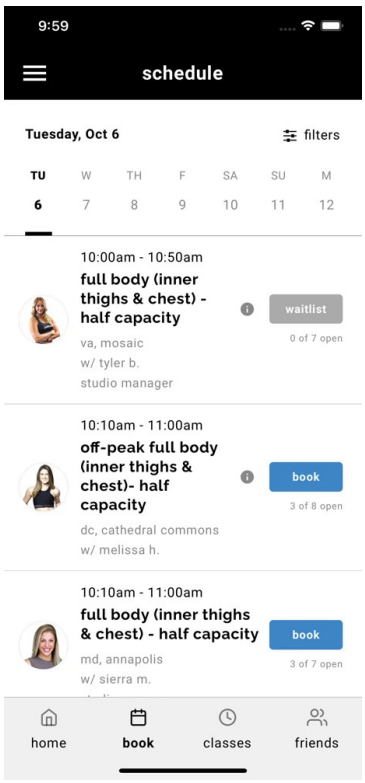


Рисунок 1.11 – Список сесій в мобільному додатку [solidcore]

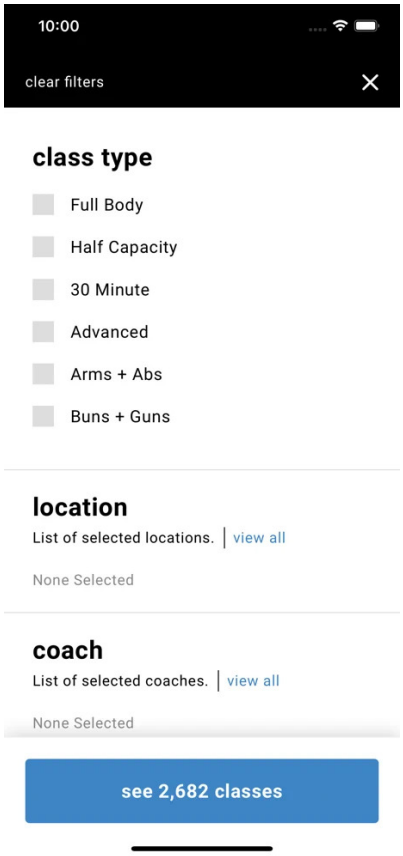


Рисунок 1.12 – Фільтри в мобільному додатку [solidcore]

Отже, мобільний застосунок [solidcore], що відіграє роль членського застосунку, та має примітивний базовий функціонал, закриває цифрові бізнес-проблеми лише однієї мережі фітнес студій.

1.3.3 Urban Sports Club

Urban Sports Club – це багатофункціональна платформа, яка допомагає користувачу знайти активність, яка підходить саме йому, в конкретний час (рисунок 1.13):

- Заняття на місці – бронювання групового заняття в одному з партнерських фітнес центрів;
- Соло заняття – звичайних доступ до партнерського фітнес центру з метою самостійного заняття;
- Online – бронювання online заняття і відвідини його у віртуальному форматі – через телефон, або планшет.

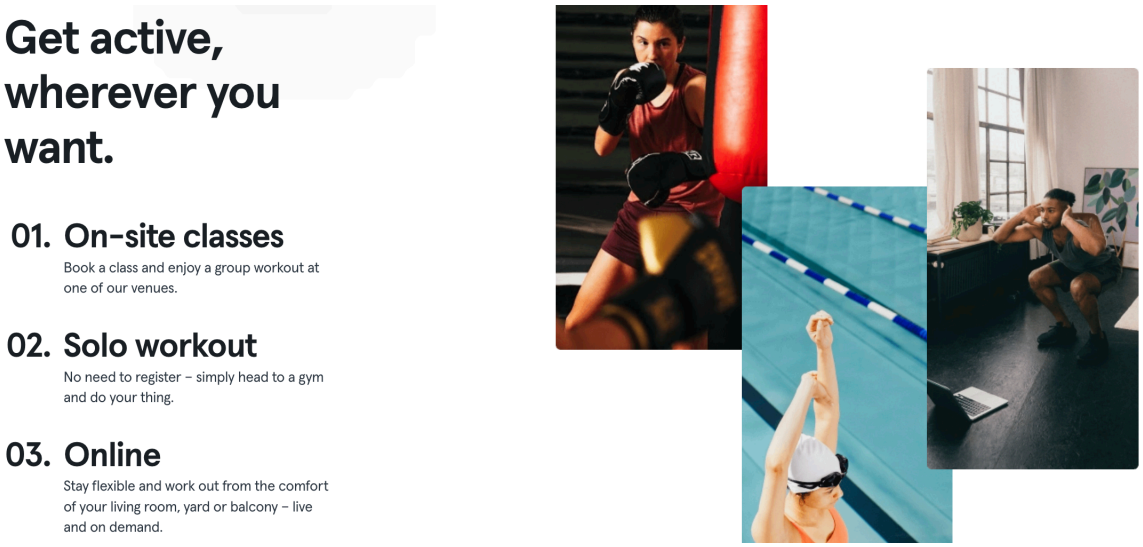


Рисунок 1.13 – Можливості мобільного додатку Urban Sports Club

Опис програмного додатку почнеться з самої першої сторінки, яка відображається користувачу після встановлення додатку, а саме – сторінки авторизації (рисунок 1.14). Ця платформа дає можливість користувачу використовувати базовий функціонал без авторизації. В цей базовий функціонал входить наступне:

– лістниг можливих сессій (рисунок 1.15);
– сторінка з деталями сесії (рисунок 1.16);
– лістниг графіку проведення сессій (рисунок 1.17).

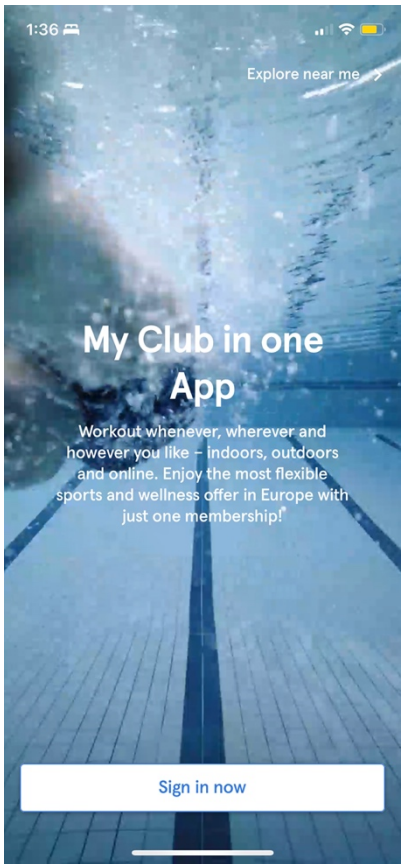


Рисунок 1.14 – Сторінка авторизації в мобільному додатку Urban Sports Club

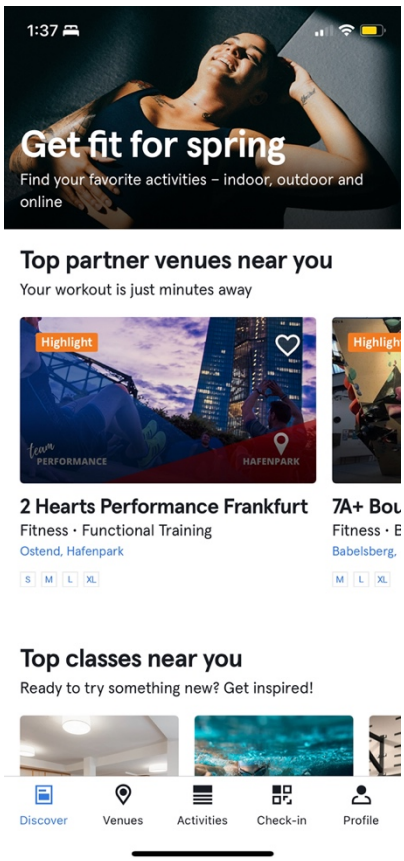


Рисунок 1.15 – Сторінка авторизації в мобільному додатку Urban Sports Club

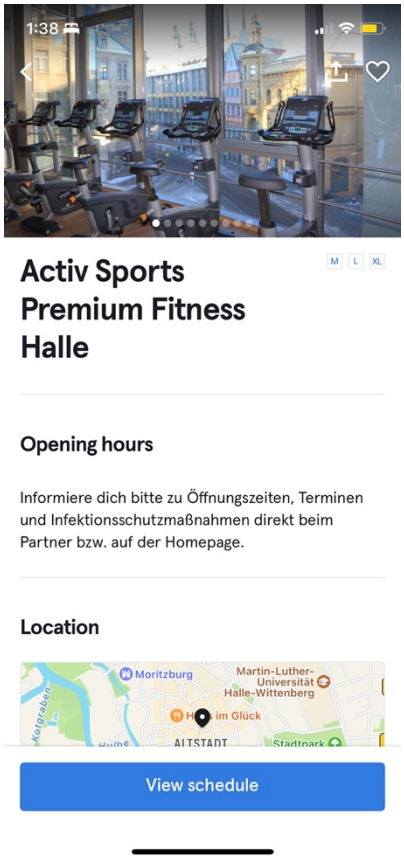


Рисунок 1.16 – Сторінка сесії в мобільному додатку Urban Sports Club

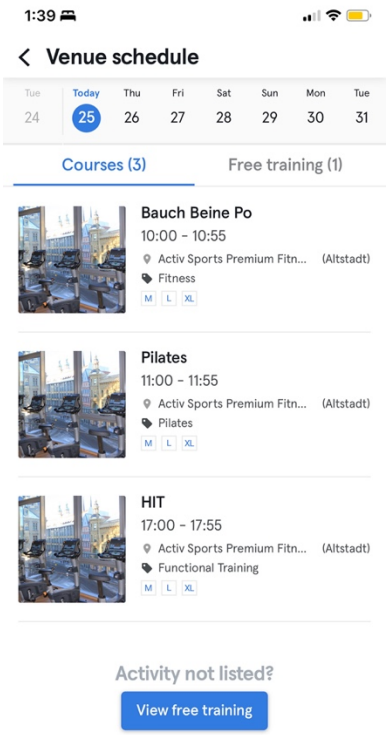


Рисунок 1.17 – Сторінка графіку для сесії в мобільному додатку Urban Sports Club

Після авторизації в обліковий запис – у користувача з'являється можливість зареєструватися на сесію та оплатити участь. Як тільки оплата пройшла успішно – створиться унікальний QR Code (рисунок 1.18) учасника за допомогою якого можна буде ідентифікувати учасника на самій сесії. Також, цей самий QR Code можна буде знайти у власному кабінеті користувача (рисунок 1.19).

Рисунок 1.18 – Сторінка унікального QR Code сесії в мобільному додатку Urban Sports Club

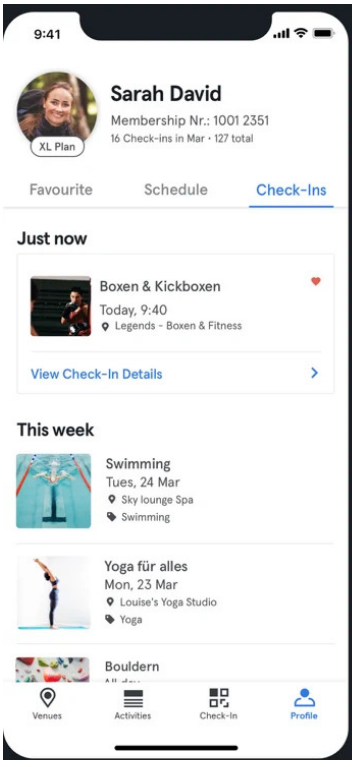


Рисунок 1.18 – Сторінка кабінету користувача в мобільному додатку Urban Sports Club

Також хотів зазначити, що сервіс має географічні обмеження в роботі і не дозволяє використовувати свій продукт за межами визначених країн.

1.4 Аналіз вимог до програмного забезпечення

Виділимо загальні ролі користувачів для створення мобільного застосунку для отримання послуг у фітнес-центрах.

Користувач, який не має власного облікового запису – користувач мобільного пристрою, що має доступ до мережі Інтернет, що не зареєстрований у системі.

Користувач, який має власний обліковий запис – користувач мобільного пристрою, що має доступ до мережі Інтернет та вже зареєструвався у системі за допомогою сервісу Apple Sign In.

Система повинна містити наступні можливості:

- авторизація та деавторизація користувача;
- перегляд списку подій та фітнес-центрів;
- перегляд деталей обраної події або фітнес-центру;
- оплата за допомогою Apple Pay;
- перегляд списку придбаних пропусків до подій та фітнес-центрів;
- пошук подій та фітнес-центрів за їх назвою.

1.4.1 Розроблення функціональних вимог

Розглянемо варіанти використання мобільного додатку:

Таблиця 1.1 – Варіант використання UC001

Назва	Створення облікового запису.
Опис	Користувач без облікового запису має можливість зареєструватись у системі за допомогою Apple Sign In. В результаті, користувач отримає api token.

Продовження таблиці 1.1

Учасники	Користувач, який не має облікового запису.
Передумови	Користувач без облікового запису, застосунок запущений.
Постумови	Користувач зареєстрований.
Основний сценарій	1. користувач запускає додаток; 2. користувач натискає на кнопку входу в особистий кабінет; 3. система відображає вікно Apple Sign In; 4. користувач підтверджує свої біометричні дані; 5. система надсилає apple token на сервер; 6. система перевіряє apple token на наявність в базі даних та реєструє при необхідності; 7. система та надає користувачеві api token.

Таблиця 1.2 – Варіант використання UC002

Назва	Вихід із власного облікового запису.
Опис	Користувач з обліковоим записов має можливість вийти із нього.
Учасники	Користувач, який має обліковий запис.
Передумови	Користувач з обліковим записов, застосунок запущений.
Постумови	Користувач вийшов з облікового запису.
Основний сценарій	1. користувач запускає додаток; 2. користувач натискає на кнопку виходу; 3. система знищує api token.

Таблиця 1.3 – Варіант використання UC003

Назва	Перегляд списку новин.
Опис	Користувач без облікового запису має можливість переглянути список новин.
Учасники	Користувач, який не має облікового запису.
Передумови	Користувач без облікового запису, застосунок запущений.
Постумови	Користувач бачить список новин.
Основний сценарій	1. користувач запускає додаток; 2. система показує список новин.

Таблиця 1.4 – Варіант використання UC004

Назва	Перегляд списку подій.
Опис	Користувач без облікового запису має можливість переглянути список подій.
Учасники	Користувач, який не має облікового запису.
Передумови	Користувач без облікового запису, застосунок запущений.
Постумови	Користувач бачить список подій.
Основний сценарій	1. користувач запускає додаток; 2. система показує список подій.

Таблиця 1.5 – Варіант використання UC005

Назва	Перегляд списку фітнес-центрів.
Опис	Користувач без облікового запису має можливість переглянути список фітнес-центрів.
Учасники	Користувач, який не має облікового запису.

Продовження таблиці 1.5

Передумови	Користувач без облікового запису, застосунок запущений.
Постумови	Користувач бачить список фітнес-центрів.
Основний сценарій	1. користувач запускає додаток; 2. система показує список фітнес-центрів.

Таблиця 1.6 – Варіант використання UC006

Назва	Пошук подій.
Опис	Користувач без облікового запису має можливість шукати подію за назвою.
Учасники	Користувач, який не має облікового запису.
Передумови	Користувач без облікового запису, застосунок запущений.
Постумови	Користувач бачить знайдену подію.
Основний сценарій	1. користувач запускає додаток; 2. користувач натискає на поле пошуку; 3. користувач вводить назву шуканої події; 4. система відображає знайдену подію.

Таблиця 1.7 – Варіант використання UC007

Назва	Пошук фітнес-центру.
Опис	Користувач без облікового запису має можливість шукати фітнес-центр за назвою.
Учасники	Користувач, який не має облікового запису.
Передумови	Користувач без облікового запису, застосунок запущений.

Продовження таблиці 1.7

Постумови	Користувач бачить знайдену подію.
Основний сценарій	1. користувач запускає додаток; 2. користувач натискає на поле пошуку; 3. користувач вводить назву фітнес-центру; 4. система відображає знайдений фітнес-центр.

Таблиця 1.8 – Варіант використання UC008

Назва	Перегляд деталей новини.
Опис	Користувач без облікового запису має можливість переглянути деталі новини.
Учасники	Користувач, який не має облікового запису.
Передумови	Користувач без облікового запису, застосунок запущений.
Постумови	Користувач бачить сторінку з деталями новини.
Основний сценарій	1. користувач запускає додаток; 2. користувач обирає новину; 3. система відкриває сторінку з деталями новини.

Таблиця 1.9 – Варіант використання UC009

Назва	Перегляд деталей події.
Опис	Користувач без облікового запису має можливість переглянути деталі події.
Учасники	Користувач, який не має облікового запису.
Передумови	Користувач без облікового запису, застосунок запущений.

Продовження таблиці 1.9

Постумови	Користувач бачить сторінку з деталями події.
Основний сценарій	1. користувач запускає додаток; 2. користувач обирає подію; 3. система відкриває сторінку з деталями події.

Таблиця 1.10 – Варіант використання UC010

Назва	Перегляд деталей фітнес-центру.
Опис	Користувач без облікового запису має можливість переглянути деталі фітнес-центру.
Учасники	Користувач, який не має облікового запису.
Передумови	Користувач без облікового запису, застосунок запущений.
Постумови	Користувач бачить сторінку з деталями фітнес-центру.
Основний сценарій	1. користувач запускає додаток; 2. користувач обирає подію; 3. система відкриває сторінку з деталями події.

Таблиця 1.11 – Варіант використання UC011

Назва	Оплата доступу.
Опис	Користувач з обліковим записом має можливість переглянути оплатити доступ до фітнес центру або події.
Учасники	Користувач, який має обліковий запис.
Передумови	Користувач з обліковим записом, застосунок запущений.

Продовження таблиці 1.11

Постумови	Користувач бачить QR код доступу до фітнес-центру або події.
Основний сценарій	1. користувач запускає додаток; 2. користувач обирає подію або фітнес-центр; 3. система відкриває сторінку з деталями події і кнопкою оплати; 4. користувач натискає на кнопку оплати; 5. система відображає вікно Apple Pay; 6. користувач підтверджує свої біометричні дані; 7. система перевіряє статус оплати; 8. користувач бачить згенерований QR код.

Таблиця 1.12 – Варіант використання UC012

Назва	Перегляд оплачених подій.
Опис	Користувач з обліковим записом має можливість переглянути свої оплачені події.
Учасники	Користувач, який має обліковий запис.
Передумови	Користувач з обліковим записом, застосунок запущений.
Постумови	Користувач бачить список з оплаченими подіями.
Основний сценарій	1. користувач запускає додаток; 2. користувач натискає на навігаційну кнопку Active Passes; 3. система відкриває сторінку зі списком оплачених подій.

Таблиця 1.13 – Варіант використання UC013

Назва	Перегляд оплачених фітнес-центрів.
Опис	Користувач з обліковим записом має можливість переглянути свої оплачені фітнес-центри.
Учасники	Користувач, який має обліковий запис.
Передумови	Користувач з обліковим записом, застосунок запущений.
Постумови	Користувач бачить список з оплаченими фітнес-центрами.
Основний сценарій	1. користувач запускає додаток; 2. користувач натискає на навігаційну кнопку Active Passes; 3. система відкриває сторінку зі списком фітнес-центрів.

Функціональні вимоги наведені в таблицях.

Таблиця 1.14 – Варіант використання REQ001

Номер	REQ001
Назва	Створення облікового запису у системі.
Опис	Реєстрація нових користувачів у додатку за допомогою сервісу Apple Sign In.

Таблиця 1.15 – Варіант використання REQ002

Номер	REQ002
Назва	Деавторизація з системи.
Опис	Видалення API key з системи.

Таблиця 1.16 – Варіант використання REQ003

Номер	REQ003
Назва	Відображення списку новин.
Опис	Будь які новини повинні відображатись користувачу. Новини мають бути відсортовані за датою, починаючи з найсвіжішої новини.

Таблиця 1.17 – Варіант використання REQ004

Номер	REQ004
Назва	Відображення списку подій.
Опис	Останні 10 подій мають відображатись користувачу. Події мають бути відсортовані за датою, починаючи з найближчої події. Виконати фільтр по імені при необхідності.

Таблиця 1.18 – Варіант використання REQ005

Номер	REQ005
Назва	Відображення списку фітнес-центрів.
Опис	Усі фітнес-центри мають відображатись користувачу. Виконати фільтр по імені при необхідності.

Таблиця 1.19 – Варіант використання REQ006

Номер	REQ006
Назва	Відображення списку оплачених доступів.
Опис	Користувач має побачити усі події та фітнес-центри, за які була здійснена оплата.

Таблиця 1.20 – Варіант використання REQ007

Номер	REQ007
Назва	Оплата доступу.
Опис	Система має опрацювати платіж і повернути результат

Матриця відносин між функціональними вимогами та варіантами використання додатку зображені на рисунку 1.19.

	REQ001 Створення облікового запису у системі	REQ002 Деавторизація з системи	REQ003 Відображення списку новин	REQ004 Відображення списку подій	REQ005 Відображення списку фітнес-центрів	REQ006 Відображення списку оплачених доступів	REQ007 Оплата доступу
UC001 Створення облікового запису	■						
UC001 Вихід із власного облікового запису		■					
UC001 Перегляд списку новин			■				
UC001 Перегляд списку подій				■			
UC001 Перегляд списку фітнес-центрів					■		
UC001 Пошук подій							
UC001 Пошук фітнес-центру							
UC001 Перегляд деталей новини			■				
UC001 Перегляд деталей події				■			
UC001 Перегляд деталей фітнес-центру					■		
UC001 Оплата доступу							■
UC012 Перегляд оплачених подій						■	
UC013 Перегляд оплачених фітнес-центрів						■	

Рисунок 1.19 – Матриця відносин між функціональними вимогами та варіантами використання додатку

1.4.2 Розроблення нефункціональних вимог

Мобільний програмний застосунок має відповідати наступним нефункціональним вимогам:

- мова користувацького інтерфейсу – англійська;
- відображення дат в додатку – локальна. Дата встановлюється автоматично, але може бути модифікована в налаштуваннях операційної системи;
- тривалість життя доступу до події – включно до дати наступного дня після дати події;
- тривалість життя доступу до фітнес-центру – рівно 30 днів з моменту оплати доступу;
- обмін даними між мобільним застосунком та сервером має бути реалізовано використовуючи захищений протокол HTTPS.

1.4.3 Постановка комплексу завдань модулю

Програмний застосунок, який розробляється в рамках даної дипломної роботи призначений для отримані послуг у фітнес-центрах.

Мета створення даної роботи – впровадження цифрового досвіду отримання послуг у фітнес-центрах за допомогою мобільного застосунку.

Для досягнення мети проєкту мобільний додаток має вирішувати наступні задачі:

- створення облікового кабінету користувача;
- лістинг доступних подій та фітнес-центрів;
- пошук серед доступних подій та фітнес-центрів;
- оплата доступу до подій та фітнес-центрів;
- відображення списку новин;
- відображення деталей новин, події або фітнес-центру.

1.5 Висновки по розділу

У цьому розділі мною було вивчено декілька існуючих мобільних програмних рішень для вирішення проблеми на отримання послуг у фітнес центрах. Усі з вищезазначених платформ працюють на ринок західних країн, і в той самий час, в Україні ще немає якогось аналогічного рішення, яке би працювало на ринон України.

В результаті аналізу інших сервісів – мною була створена порівняльна таблиця 1.20, в якій можна ясно розгледіти переваги і недоліки. З даних таблиці можна зробити висновок, що покращений користувацький досвід можна отримати в результаті інтуїтивно зрозумілого інтерфейсу з можливістю використання нативних сервісів, та елементів графічного інтерфейсу. Саме тому, було прийнято використання нативного сервісу Apple Pay, що дозволить клієнту не вводити вручну дані карти, та економити час при оформлені замовлення, і також використовувати нативний сервіс Apple Sign In для перевикористання раніше наданої інформації користувача компанії Apple, що дозволить економити час під час створення нового облікового запису, або авторизації на платформі.

Таблиця 1.21 – Порівняння існуючих сервісів

Функціонал	Hussle	[solidcore]	Urban Sports Club	Gean Sports
Створення облікового запису	+	+	+	+
Швидка реєстрація за допомогою Apple Sign In	+	-	-	+
Лістинг фітнес центрів без авторизації	+	-	+	+
Запис на сесії	-	+	+	+

Продовження таблиці 1.21

Оплата збереженою картою за допомогою Apple Pay	-	-	-	+
Лістинг новин	-	+	-	+
Створення QR Code для ідентифікації учасника	-	-	+	+

2 ВИБІР ТЕХНОЛОГІЙ І СЕРЕДОВИЩА РОЗРОБКИ

2.1 Мова програмування Swift

Swift – це дуже потужна та інтуїтивно зрозуміла мова програмування, яка була створена корпорацією Apple у 2014 році і використовується для створення програм, які працюють на платформах iOS, macOS, watchOS і tvOS. Код, написаний на Swift [1], є має чистий і сучасний синтаксис, пропонує безперешкодний доступ до існуючого коду та фреймворків C і Objective-C. Приклад коду програми, написаній на Swift, показаний на рисунку 2.1.

```swift
// оголошення масиву даних
var fruits = ["mango", "kiwi", "avocado"]

// приклад оператора if; функцій .isEmpty, та .count
if fruits.isEmpty {
    print("Фрукти відсутні у масиві даних.")
} else {
    print("В масиві даних є \(fruits.count) фруктів")
}

// приклад оголошення  "словника" (dictionary ) з 4 елементів, кожен із яких містить ім'я та вік
let people = ["Anna": 67, "Beto": 8, "Jack": 33, "Sam": 25]

// використовуючи можливості мови Swift, ми надрукуємо обидва значення в єдиному циклі
for (name, age) in people {
    print("\(name) is \(age) years old.")
}

// оголошення методів починається із службового слова "func"
// тип результату описується після "->"
func sayHello(personName: String) -> String {
    let greeting = "Hello, " + personName + "!"
    return greeting
}

// як вивести в консолі словосполучення "Hello, Jane!", використовуючи вище описаний метод
print(sayHello("Jane"))
```

Рисунок 2.1 – Приклад програми на мові Swift

Незважаючи на те, що Swift був натхненний Objective-C та багатьма іншими мовами, він сам по собі не є мовою, похідною від C. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація. Будучи повною та незалежною мовою, Swift пропонує основні функції, такі як керування потоками, структури даних і функції, з конструкціями високого

рівня, такими як об’єкти, протоколи, замикання та генерики. Swift охоплює модулі, усуваючи потребу в заголовках і дублюванні коду, які вони спричиняють.

Swift використовує автоматичний підрахунок посилань (ARC) для керування пам'яттю. Раніше Apple вимагала ручного керування пам’яттю в Objective-C, але в 2011 році представила ARC, щоб полегшити розподіл і звільнення пам’яті. Однією з проблем ARC є можливість створення сильного циклу посилань, коли об’єкти посилаються один на одного таким чином, що ви можете досягти об’єкта, з якого ви почали, дотримуючись посилання. Наприклад: A посилається на B, B посилається на A (рисунок 2.2). Це призводить до того, що вони застрягають в пам’яті оскільки вони ніколи не звільняються. Swift надає ключові слова «weak» і «unowned», щоб запобігти сильні цикли посилань. Як правило, зв’язок «батько-дитина» використовують сильне посилання, тоді як дитина-батько використовує або слабке посилання, коли батьки та діти можуть бути не пов’язаними, або невласними, коли дитина завжди має батьків, але батько може не мати дитини. Слабкі посилання повинні бути «weak» змінними, оскільки вони можуть змінюватися і ставати нульовими. Замикання всередині класу також може створити сильний цикл посилань, захоплюючи посилання на себе, тому тут важливо використовувати тільки «weak» або «unowned» посилання (рисунок 2.3). Посилання на себе, які розглядатимуться як «weak» або «unowned», можна вказати за допомогою списку захоплення.

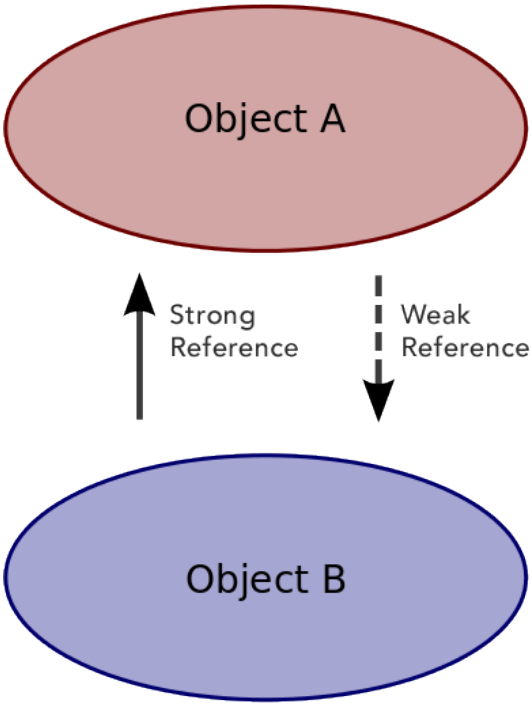


Рисунок 2.2 – Приклад сильного і слабкого посилання у зв’язку «батько-дитина»

Рисунок 2.3 – Приклад правильного слабкого посилання на себе в замикані

Оскільки Swift використовується для створення програм – він є відкритим для всіх і тому, протягом 8 років його існування, він придбав велику спільноту підтримки та велику кількість сторонніх інструментів.

2.2 Порівняння Swift та Objective-C

Swift покликаний зробити революцію і повністю замінити Objective-C, оскільки це нова і більш зручна мова програмування. Більш детальне порівняння наведено у таблиці 2.1.

Таблиця 2.1 – Порівняння Objective-C та Swift

Параметр	Objective-C	Swift
Читабельність	Objective-C, окрім legacy коду, ще й вимагає створення header в окремому файлі, в якому декларуються змінні та функції	Код, написаний на Swift є більш чистим, оскільки Swift відкидає багато застарілих умов, таких як крапка з комою в кінці рядків або дужки, які оточують умовні вирази всередині операторів if/else
Підтримка	Неможливість розвитку Objective-C без розвитку C, оскільки є пряма залежність. C вимагає від програмістів підтримувати два файли коду, щоб покращити час компіляції та ефективність коду, що також переноситься на Objective-C	Swift відкидає вимогу щодо двох файлів, об'єднуючи заголовок Objective-C (.h) і файли реалізації (.m) в один файл коду (.swift). Тому створення програми на Swift дозволяє програмістам приділити більше часу створенню логіки
Менше коду та менше Legacy коду	З Objective-C існує багато проблем, які викликають збої в роботі програми. Крім того, класи поділені на дві частини: інтерфейс і реалізація. Це збільшує кількість файлів у проекті вдвічі	Swift надає код, який менш схильний до помилок через його вбудовану підтримку для маніпулювання текстовими рядками та даними

Продовження таблиці 2.1

Швидкість	Objective-C у 2,8 рази швидше, ніж версія Python, проте він значно програє Swift	Swift також надає різні переваги швидкості під час розробки, у свою чергу, заощаджуючи витрати. Наприклад, складне сортування об’єктів буде працювати в 3,9 рази швидше, ніж реалізація того ж алгоритму в Python. Його продуктивність наближається до C++, який вважається найшвидшим алгоритмом обчислення арифметики
Об’єм пам’яті	Objective-C пропонує статичні бібліотеки, які не оновлюються автоматично, а застрягли у версії, в якій вони були зібрані. Під час компіляції програми статичні бібліотеки блокуються в коді. Це збільшує розмір програми та збільшує час її завантаження	Swift пропонує динамічні бібліотеки, що є великою перевагою. Динамічні бібліотеки, які існують поза кодом, завантажуються лише тоді, коли це необхідно. Оскільки стандартні бібліотеки Swift включені в кожну macOS, iOS, tvOS і watchOS, це допомагає зменшити загальний розмір програми

Продовження таблиці 2.1

Контроль пам'яті програми	Керування пам'яттю програми вручну може бути громіздким і складним завданням, головним чином тому, що програміст повинен пам’ятати про вивільнення, коли закінчив роботу з об’єктом. Це необхідна, але не завжди практична рутина. На практиці це означає, що програмісту потрібно збалансувати кожен виклик виділення, збереження та копіювання з вивільненням або автоматичним вивільненням на одному об’єкті	Swift пропонує простий і легкий спосіб керування пам'яттю програми. Ця функція допомагає запобігти витоку пам’яті завдяки автоматичному підрахунку посилань. За допомогою автоматичного підрахунку пам’яті (ARC) Swift визначає, які екземпляри більше не потрібні. ARC автоматично позбавиться від них, допомагаючи покращити продуктивність програми, не впливаючи на пам’ять або центральний процесор

2.2.1 Що краще: Swift або Objective-C?

Новачку, який планує працювати з продуктами Apple, вже немає сенсу приділяти час вивченню Objective-C тому що ця мова програмування вже пережила своє. Для початківця дуже важливим є той факт, що мова програмування Swift підтримується і оновлюється, що не можна сказати про Objective-C. Інша велика перевага – час на вивчення і освоєння мови програмування Swift є набагато низьким через свою простоту.

Багато компаній у всьому світі скористалися перевагами розробки додатків Swift, і список, як правило, неосяжний. Мова вже займає лідируючі

позиції, і попереду є багатообіцяюче майбутнє, оскільки Apple продовжує безкінечно розвивати мову, щоб розкрити її повний потенціал.

2.3 Фреймворк SwiftUI

SwiftUI — це абсолютно новий фреймворк від Apple для створення інтерфейсів користувача для iOS, tvOS, macOS і watchOS. Apple представила SwiftUI у 2019 році, і відтоді фреймворк розвивається швидкими темпами. На відміну від UIKit, SwiftUI [2] є кросплатформним фреймворком. Завдяки SwiftUI Apple пропонує розробникам рішення для швидкого створення додатків.

SwiftUI абсолютно новий, але він виглядає знайомим тим, хто розробляв на UIKit (рисунок 2.4). Незважаючи на те, що концепції, що лежать в основі SwiftUI, дуже відрізняються від UIKit, Apple не винайшла велосипед. Фреймворк пропонує все необхідне для створення користувацьких інтерфейсів, таких як списки, стеки, кнопки, пікери та багато інших компонентів, з якими розробники на UIKit вже знайомі. Він також надає інструменти, необхідні для створення сторінок, додавання анімації та інтеграції жестів керування.

```swift
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Your time is limited, so don't waste it living someone else's life. Don't be trapped by dogma—which is living with the results of other people's thinking. Don't let the noise of others' opinions drown out your own inner voice. And most important, have the courage to follow your heart and intuition.")
            .fontWeight(.bold)
            .font(.title)
            .foregroundColor(.gray)
            .multilineTextAlignment(.center)
            .lineSpacing(10)
            .padding()
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

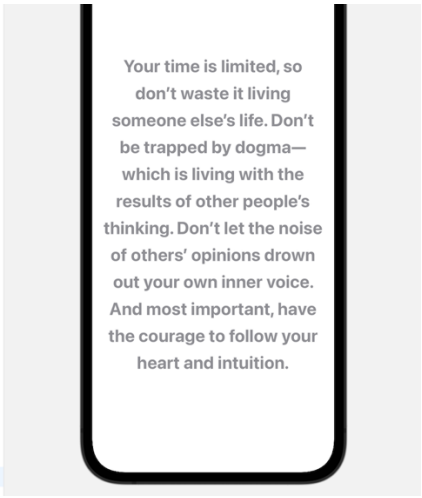


Рисунок 2.4 – Приклад програми написаний за допомогою фреймворку SwiftUI

SwiftUI — це набір інструментів користувальницького інтерфейсу, який дозволяє нам декларативно проектувати програми. Можна сказати, що ми розказуємо SwiftUI, як ми хочемо, щоб наш інтерфейс користувача виглядав і працював, і він з'ясовує, як це зробити, коли користувач взаємодіє з ним.

Декларативний інтерфейс користувача найкраще зрозуміти в порівнянні з імперативним інтерфейсом користувача, що робили розробники iOS до iOS 13. У імперативному інтерфейсі користувача ми можемо зробити так, щоб функція викликалася при натисканні кнопки, всередині функції зчитається значення і показується текст – ми регулярно змінюємо зовнішній вигляд і роботу інтерфейсу користувача залежно від того, що відбувається.

Імперативний інтерфейс користувача спричиняє всілякі проблеми, більшість з яких обертається навколо стану, що є ще одним модним терміном, що означає «значення, які ми зберігаємо в нашому коді». Нам потрібно відстежити, в якому стані знаходиться наш код, і переконатися, що інтерфейс користувача правильно відображає цей стан (рисунок 2.5).

```
// Imperative Programming

let array = [1, 2, 3, 4, 5, 6]

var evenNumbers: [Int] = []

for i in 0..<array.count {
    if array[i] % 2 == 0 {
        evenNumbers.append(array[i])
    }
}
```

Рисунок 2.5 – Приклад імперативного стилю написання програми

Якщо у нас є один екран з однією властивістю булевого значення, яка впливає на інтерфейс користувача, у нас є два стани: булеве значення може бути увімкненим або вимкненим. Якщо у нас є два булеві значення A і B, то ми маємо чотири стани:

– A вимкнено і B вимкнено;
– A увімкнено і B вимкнено;
– A вимкнено і B увімкнено;
– A увімкнено і B увімкнено.

А якщо у нас три булеві значення? Або п'ять? Або цілі числа, рядки, дати тощо? Ну, тоді у нас все набагато складніше.

Якщо ви коли-небудь користувалися програмою, яка повідомляє, що у вас є 1 непрочитане повідомлення, незалежно від того, скільки разів ви натискали на читання повідомлення, то це проблема стану, а точніше - проблема імперативного інтерфейсу користувача.

На відміну від цього, декларативний інтерфейс користувача дозволяє нам повідомляти iOS про всі можливі стани нашої програми одночасно (рисунок 2.6). Ми можемо сказати, що якщо ми ввійшли - потрібно показувати привітальне повідомлення, але якщо ми вийшли - потрібно показувати кнопку входу. Нам не потрібно писати код, щоб переміщатися між цими двома станами вручну – це і є проблемою імперативного інтерфейсу, яку Apple намагається залишити в минулому.

```
// Declarative

let evenNumbers2 = array.filter { $0 % 2 == 0 }
```

Рисунок 2.6 – Приклад декларативного стилю написання програми

Замість цього ми дозволяємо SwiftUI переміщатися між елементами інтерфейсу користувача, коли стан змінюється. Ми вже сказали йому, що потрібно показувати залежно від того, чи увійшов користувач, чи вийшов із системи. Тому, коли ми змінюємо стан аутентифікації - SwiftUI може оновлювати інтерфейс користувача без додаткового коду.

Ось що це означає під декларативним інтерфейсом: ми не прописуємо кодом відображення і приховування компонентів SwiftUI, а просто задаємо

йому всі правила, яких ми хочемо, щоб він дотримувався, і очікуємо, що SwiftUI ці всі правила дотримуються.

Але SwiftUI не зупиняється на досягнутому – він також діє як міжплатформний шар інтерфейсу користувача, який працює на iOS, macOS, tvOS і навіть watchOS. Це означає, що тепер можна вивчити одну мову та одну структуру макета, а потім розгорнути свій код на декількох платформах.

2.4 Порівняння SwiftUI та UIKit

В таблиці 2.2 наведені порівняння основних параметрів між UIKit та SwiftUI.

Таблиця 2.2 – Порівняння UIKit та SwiftUI

Параметр	UIKit	SwiftUI
Конструктор інтерфейсу	Є можливість визначити весь інтерфейс в коді, або в призначеному для цього – Interface Builder	Надається можливість визначити інтерфейс в коді і одночасно бачити результат в Canvas, який оновлюється в режимі реального часу
Оновлення інтерфейсу користувача	За допомогою UIKit розробники повинні визначити, що відображатиметься на екрані та коли екран оновиться. Вони також повинні визначити, як відбуватиметься перехід між різними станами UI	SwiftUI функціонує реактивно. За допомогою SwiftUI все, що бачить користувач на екрані, є ефектом оновлення структури елементу інтерфейсу і дозволяє системі зрозуміти, що потрібно зробити
Документація та підтримка	UIKit вважається кращим, ніж SwiftUI, оскільки UIKit є старшим фреймворком і тому має більшу документацію і можливості	Оскільки SwiftUI є достатньо молодим, тому його можливості можуть не повністю задовільнити розробників. Не зважаючи на це – розробники все одно переходять на SwiftUI

Продовження таблиці 2.2

Сумісність з різними версіями iOS	UIKit дозволяє підтримувати починаючи з iOS 9.0. Це робить UIKit більш гнучким. Він безсумнівно виграє проти SwiftUI, коли справа доходить до зворотної сумісності	SwiftUI був представлений для роботи з iOS 13, проте він обмежений і працює краще з iOS 14+
Швидкість розробки програми	UIKit вимагає написання більшої кількості коду і бути більш пильним до оновлення інтерфейсу користувача при зміні стану	SwiftUI є найшвидшим способом створення функцій у програмі. Він вимагає набагато менше коду і може допомогти вам досягти майже тих самих результатів за набагато менший час порівняно з UIKit. Крім того, функція попереднього перегляду SwiftUI в реальному часі має тенденцію прискорити розробку додатків
Функціональність анімації	За допомогою UIKit розробникам потрібно визначити анімацію у своєму коді прописуючи набагато більше коду, аніж в SwiftUI	SwiftUI надає розробникам безліч анімацій. Крім того, переходи також стають набагато легшими за допомогою SwiftUI
Мультиплатформенна сумісність	Присутня обмежена мультиплатформеність	SwiftUI працює на пристроях Apple, включаючи Apple Watch, iPhone або MacBook. Елементи інтерфейсу SwiftUI адаптуються до будь-якого розміру екрана

Продовження таблиці 2.2

Віджети	UIKit не буде корисним при плануванні розробки віджета	SwiftUI легко вирішує проблему розробки віджетів для різних платформ.

2.4.1 Що краще: SwiftUI або UIKit?

На сьогоднішній день, поки неможливо сказати, який один з фреймворків краще вчити початківцю. З однієї сторони – SwiftUI дозволяє створювали програмні додатки набагато швидше, аніж UIKit. З іншої сторони - він є досить обмеженим і деякі задачі все ж потрібно буде вирішувати за допомогою UIKit. Але SwiftUI є досить молодим і буде оновлюватись в найближчій перспективі. Одного дня, SwiftUI повністю витіснить UIKit, але зараз повна розробка комплексного програмного застосунку на SwiftUI з особливим інтерфейсом неможлива без використання комбінації SwiftUI та UIKit.

2.5 xCode – IDE

xCode - це IDE [3], інтегроване середовище розробки, створене Apple для розробки програмного забезпечення для macOS, iOS, watchOS і tvOS. Це єдиний офіційно підтримуваний інструмент для створення та публікації програм у магазині додатків Apple, призначений для використання початківцями та досвідченими розробниками (рисунок 2.7).

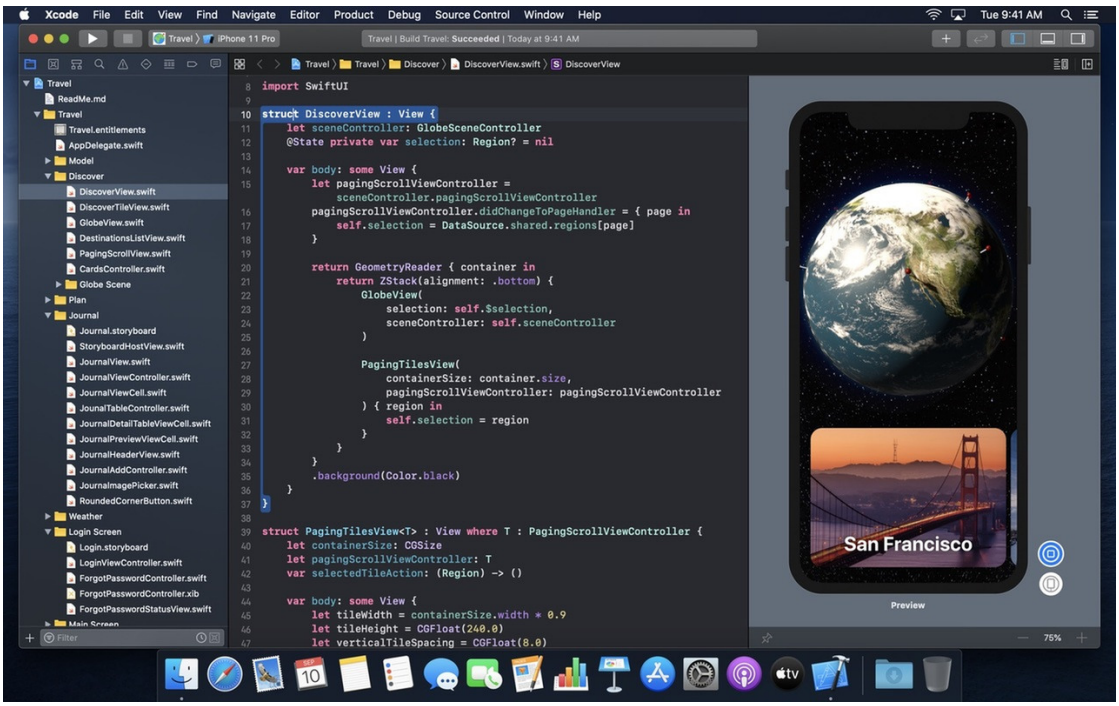


Рисунок 2.7 – Вигляд xCode IDE

xCode включає в себе всі інструменти, необхідні для створення програми, а саме: текстовий редактор і компілятор. За допомогою xCode можливо писати, компілювати, налагоджувати програму і надіслати її до Apple App Store. Він містить ряд інструментів, які допомагають швидко рухатися по процесу розробки, тому досвідчені розробники можуть створювати програми блискавично, а початківці стикаються з меншою плутаниною та перешкодами для створення своєї програми.

Як редактор коду, xCode підтримує величезну різноманітність мов програмування – C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit і Swift. Він використовує моделі програмування Cocoa, Carbon і Java.

xCode розроблено, щоб надати розробнику одне вікно для роботи. Він має перевірку вихідного коду та функцію автозаповнення, що значно полегшить написання вихідного коду. Коли створюється новий проект, розробник може вибрати один із доступних шаблонів, щоб отримати базову структуру, яку можна розширити. Ці функції корисні для новачків-розробників, оскільки вони дають їм опору під час навчання. Досвідчені

розробники знайдуть ці функції корисними для оптимізації робочого процесу та значно пришвидшеного процесу розробки додатків.

xCode також має шаблони та збережені фрагменти коду, щоб зробити розробку набагато більш гладкою. Є можливість створювати власні шаблони, зрозумієте, що часто передруковуєте часто використовуваний код. Ця функція дозволяє новачкам використовувати ці шаблони для створення своїх додатків навіть з невеликими знаннями про розробку додатків.

Редактор xCode дозволяє переглядати кілька файлів одночасно. Замість того, щоб відкривати кілька файлів, щоб зробити невелику зміну, є можливість переглянути їх усі та скористатися інструментом пошуку та заміни, щоб оновити рядки коду. Це заощадить величезну кількість часу на розробку. Робота автоматично зберігається, тому не потрібно турбуватися про втрату будь-яких оновлень або внесених змін.

xCode Interface Builder дозволяє створювати різні візуальні елементи. Є можливість розробити їх самостійно або використовувати бібліотеку xCode. Автоматичний макет дозволяє створити адаптивну програму, яка буде прив’язуватися до потрібного розміру та положення для екрана, на якому вони перебувають.

2.6 Шаблон проєктування MVVM

MVVM означає «Model View ViewModel» [4], і це архітектура програмного забезпечення, яку часто використовують розробники Apple для заміни MVC.

У MVVM UViewController вважаються частиною рівня View (таблиця 2.3), що означає, що їхня робота полягає в тому, щоб зосередитися саме на макеті та життєвому циклі сторінки – viewDidLoad() тощо. На його місці створюється новий об’єкт, який називається View Model, який фактично є більшою частиною коду. Він повинен бути здатним відповідати на запити

даних тощо, за винятком того, що він не повинен посилатися на елементи керування інтерфейсом користувача (рисунок 2.8).

Таблиця 2.3 – Структура і опис MVVM

Компонент	Опис
View Controller	View Controller виконує лише речі, пов’язані з UI - показувати/отримувати інформацію. Тобто, це частина шару для відображення.
View Model	View Model отримує інформацію від View Controller, обробляє всю цю інформацію та надсилає її назад до View Controller
Model	Це лише модель об’єкту. Він використовується у View Model і оновлюється щоразу, коли View Model надсилає нові оновлення

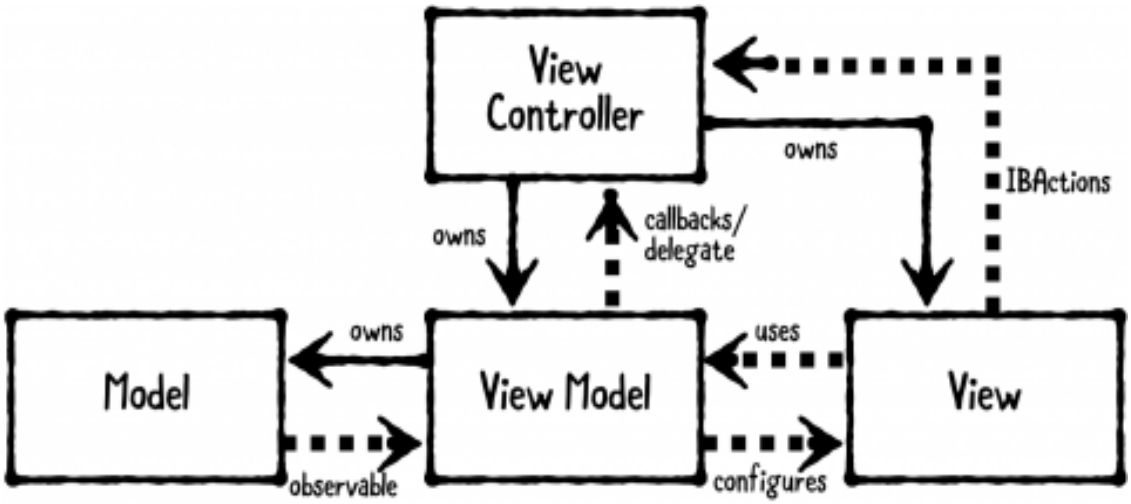


Рисунок 2.7 – Структура MVVM

Модель представляє прості дані (рисунок 2.8). Вона просто зберігає дані і не має нічого спільного з будь-якою бізнес-логікою. Можна сказати, що це проста структура даних, яку ми очікуємо від нашого API.

```swift
// MARK: - Employee
struct Employees: Decodable {
    let status: String
    let data: [EmployeeData]
}

// MARK: - EmployeeData
struct EmployeeData: Decodable {
    let id, employeeName, employeeSalary, employeeAge: String
    let profileImage: String

    enum CodingKeys: String, CodingKey {
        case id
        case employeeName = "employee_name"
        case employeeSalary = "employee_salary"
        case employeeAge = "employee_age"
        case profileImage = "profile_image"
    }
}
```

Рисунок 2.8 – Компонент Model в структурі MVVM

View Model є основним компонентом цього шаблону архітектури. View Model ніколи не знає, що таке інтерфейс і ніколи з ним не взаємодіє. Це робить цю архітектуру більш тестовою та усуває складність через елементи інтерфейсу.

```swift
import Foundation

class APIService :  NSObject {

    private let sourcesURL = URL(string: "http://dummy.restapiexample.com/api/v1/employees

    func apiToGetEmployeeData(completion : @escaping (Employees) -> ()){
        URLSession.shared.dataTask(with: sourcesURL) { (data, urlResponse, error) in
            if let data = data {

                let jsonDecoder = JSONDecoder()

                let empData = try! jsonDecoder.decode(Employees.self, from: data)
                    completion(empData)
            }
        }.resume()
    }
}
```

Рисунок 2.8 – Компонент View Model в структурі MVVM

Порядок роботи структури MVVM:

- визивається View Controller і цей View буде мати посилання на View Model;
- користувач виконає якусь дію над View Controller, який в свою чергу – надішле запит до View Model;
- View Model створить запит до API Service, та дочекається відповіді;
- як тільки відповідь надана – View Model сповістить View Controller, який в свою чергу – оновить інтерфейс користувача з отриманим Model.

2.7 Безсерверна архітектура

Простіше кажучи, безсерверна архітектура — це парадигма програмування, яка зосереджується на створення різних функцій для певних web-endpoints безпосередньо, а не на проходженні всього процесу налаштування сервера, маршрутизації запитів, визначення ресурсів REST, а потім створення методів для GET, POST і PUT. Тобто, просто необхідно створити код тіла методу, все інше надається від постачальника послуг.

Найпоширенішим постачальником безсерверних послуг є Firebase з їх службою Firebase Cloud Functions. Це високо-масштабне рішення, яке широко використовується в мобільних додатках.

Крім простоти, безсерверна архітектура має кілька інших переваг. Оскільки розробник програмує лише логіку веб-сервісу, масштабованість більше не буде проблемою (рисунок 2.9). З розробника стягуватиметься плата лише за час виконання запиту, що означає, що за невикористані функції не стягуватиметься сума.

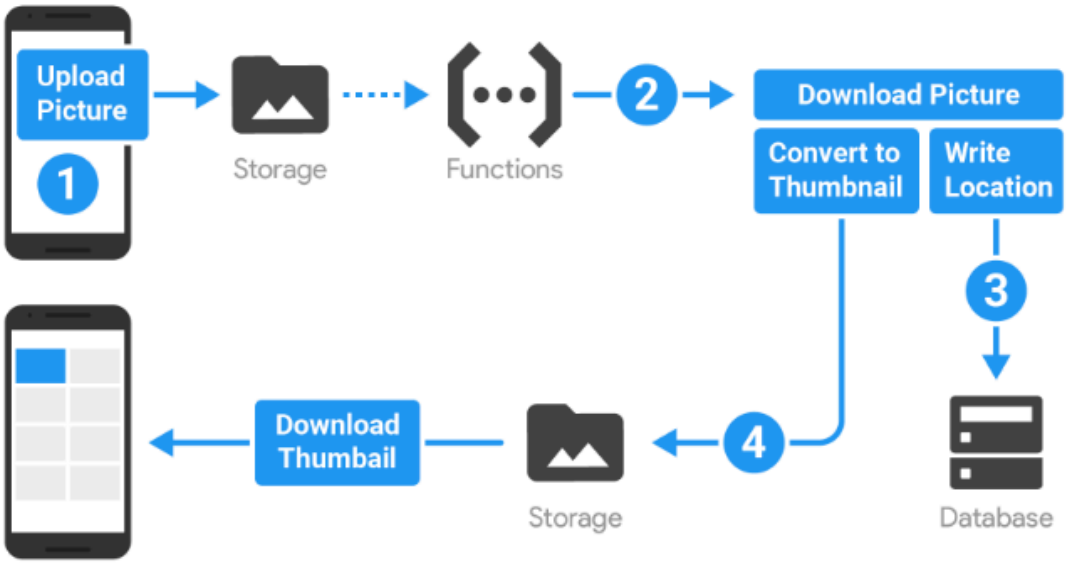


Рисунок 2.9 – Приклад використання безсерверної архітектури

2.7.1 Firebase

Firebase — це набір інструментів для створювання, покращення та розвитку мобільного додатка, а інструменти, які він надає, охоплюють значну частину сервісів, які розробникам зазвичай доводиться створювати самостійно. Це включає такі речі [6], як аналітика, аутентифікація, бази даних, конфігурація, зберігання файлів, push-повідомлення тощо. Служби розміщуються в хмарі [7] та масштабуються без будь-яких зусиль з боку розробника.

Розміщено в хмарі – мається на увазі, що продукти мають бекенд-компоненти, які повністю обслуговуються та керуються Google [8]. Клієнтські SDK, надані Firebase (рисунок 2.10), взаємодіють із цими серверними службами безпосередньо, без необхідності встановлювати проміжного програмного забезпечення між мобільним додатком і службою.

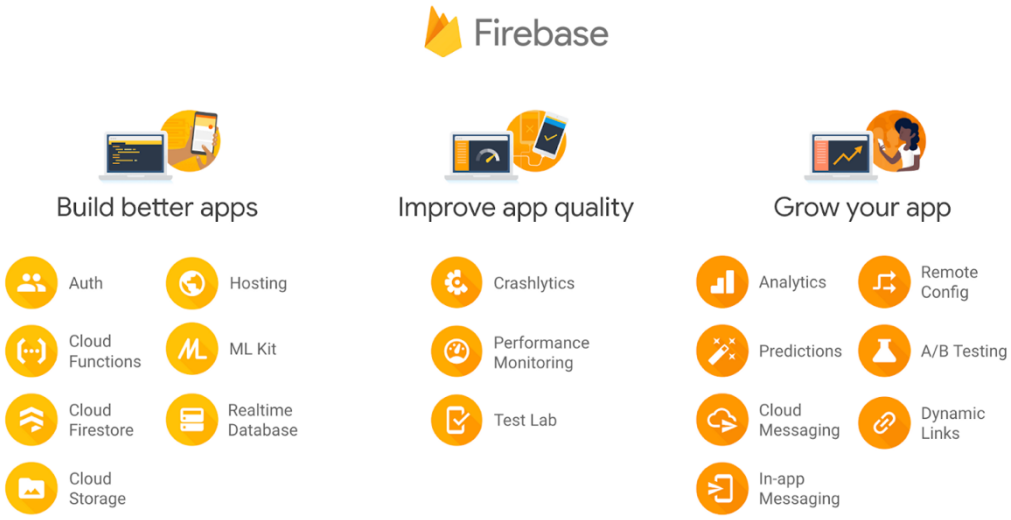


Рисунок 2.10 – Сервіси, які пропонує Firebase

Це відрізняється від традиційної розробки додатків, яка зазвичай включає в себе написання як інтерфейсного, так і бекендового програмного забезпечення. Код інтерфейсу просто викликає кінцеві точки API, надані бекендом, а бекенд-код фактично виконує роботу. Однак у продуктах Firebase традиційний бекенд обходиться, переклавши роботу на клієнта (рисунок 2.11). Адміністративний доступ до кожного з цих продуктів надає консоль Firebase.

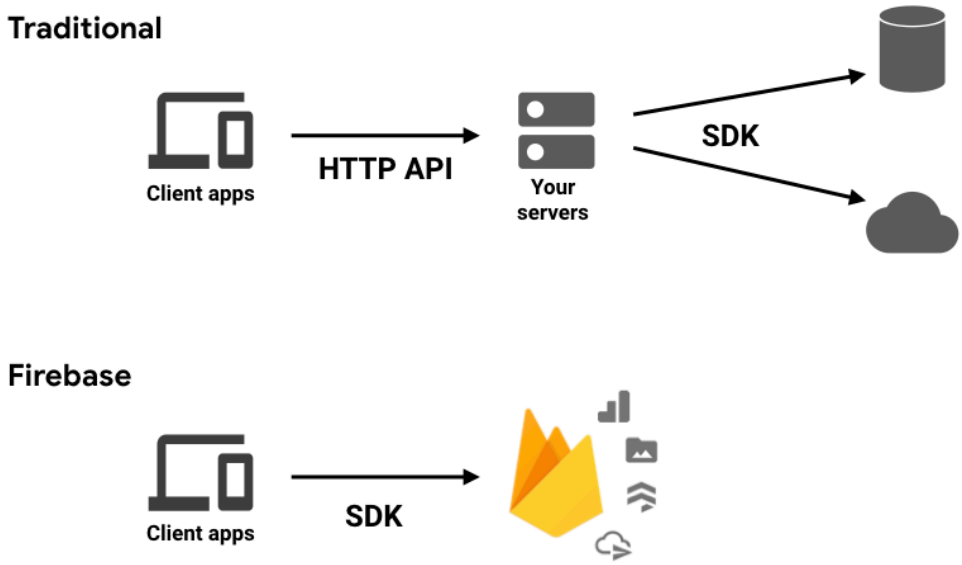


Рисунок 2.11 – Різниця між традиційним підходом для роботи з сервером і безсерверним підходом

2.7.2 Firebase Authentication

Firebase Authentication надає серверні послуги, прості у використанні пакети SDK і готові бібліотеки інтерфейсу користувача для аутентифікація користувачів у мобільному додатку. Він підтримує автентифікацію за допомогою паролів, номерів телефонів, популярних постачальників ідентифікації, таких як Google, Facebook, Twitter тощо.

Firebase Authentication тісно інтегрується з іншими службами Firebase і використовує стандарти, такі як OAuth 2.0 і OpenID Connect, тому її можна легко інтегрувати з вашим власним бекендом.

Після успішного входу ви є можливість отримати доступ до основної інформації профілю користувача, а також є можливість контролювати доступ користувача до даних, що зберігаються в інших продуктах Firebase (рисунок 2.12).

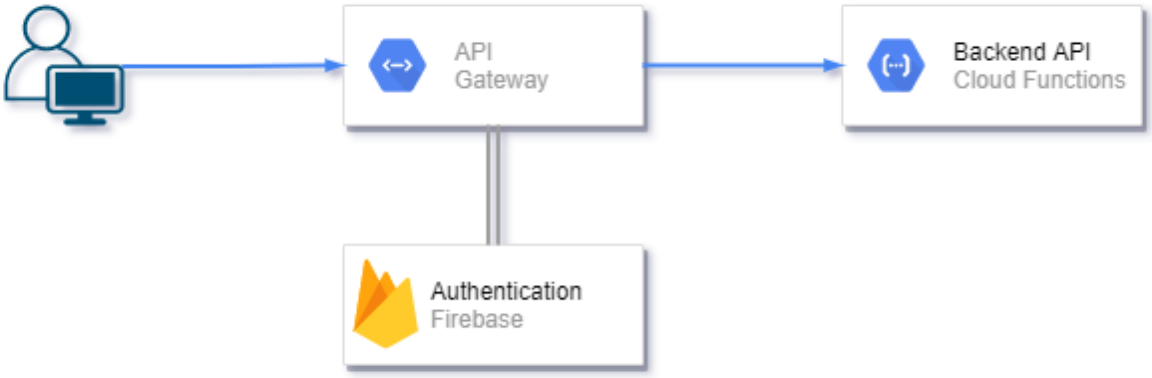


Рисунок 2.12 – Схема аутентифікації користувача використовуючи Firebase Authentication

2.7.3 Firebase Firestore

Firebase Firestore є невід'ємною частиною платформи Google Firebase. Він приймає форму хмарного сервера баз даних NoSQL, який чудово справляється зі зберіганням та синхронізацією даних. Насправді веб та мобільні програми можуть безпосередньо взаємодіяти з Firestore за допомогою вбудованих SDK. Firestore — це високопродуктивна база даних, яка підтримує

автоматичне масштабування. Крім того, він досить простий у використанні і дуже надійний. Розробники можуть працювати з Firebase, використовуючи широкий спектр технологій, таких як Java, C++, Unity, Go, Node.js SDK, REST і RPC API.

Однією з унікальних функцій є синхронізація даних між кількома клієнтськими додатками за допомогою прослуховувачів у реальному часі. Firestore використовує технологію Cloud Identity та Access Management від Google для обробки автентифікації. Firestore буквально зберігає дані як документи, які логічно класифікуються в колекції (рисунок 2.13). Документ Firestore пропонує підтримку різних типів файлів, чисел, рядків і вкладених об'єктів. Він безпечний, надійний, а також легко інтегрується з Firebase та Google Cloud Platform. Ці чудові функції пояснюють причину, чому багато компаній вибрали Firestore як свою бажану базу даних.

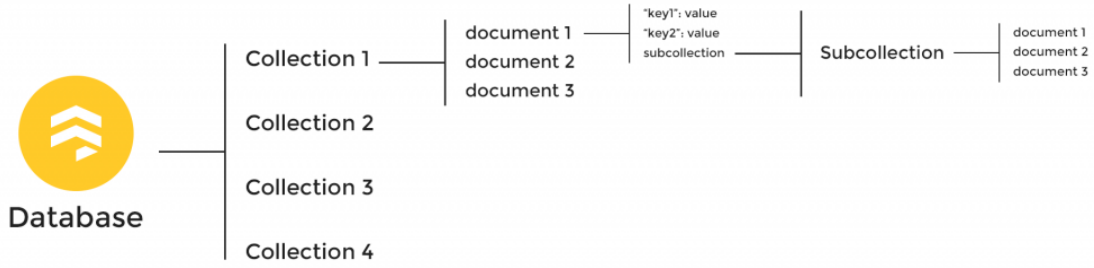


Рисунок 2.13 – Структура збереження даних в Firestore

2.7.4 Безпека в Firebase Firestore

За безпеку в Firestore відповідає Firebase Security Rules [11], які є шаром між даними та користувачами. Вони вирішують, кому дозволено отримати доступ до даних, що зберігаються в Firestore. Кожен запит на кожну операцію CRUD над даними проходить через них, і вони приймають остаточне рішення, чи слід цей запит виконувати чи ні (рисунок 2.14).

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read: if isSignedIn() && isOwner();
    }

    function isOwner(){
      return request.auth.uid == resource.data.userId;
    }

    function isSignedIn(){
      return request.auth != null;
    }

  }
}
```

Рисунок 2.14 – Приклад задання Firebase Security Rules

2.7.5 Firebase Cloud Functions

Cloud Functions — це ще один продукт Google Cloud, який добре працює з іншими продуктами Firebase та Cloud. Пакети Firebase SDK для Cloud Functions дозволяє писати та розгортати код, який працює на «безсерверній» інфраструктурі Google і який автоматично реагує на події, що надходять від інших продуктів Firebase. Хмарні функції для Firebase — це єдиний продукт усього пакету Firebase, який насправді дозволяє писати бекенд-код.

Список речей, які ви можете робити з Cloud Functions [12], величезний. Але їх поєднує одна основна концепція: продукти Firebase (база даних, сховище, авторизація тощо) видають події, коли дані змінюються, і код, розгорнутий у Cloud Functions, запускається у відповідь на ці події.

2.8 Висновки до розділу

У цьому розділі були обрані основні інструменти і технології для розробки мобільного програмного застосунку для отримання послуг у фітнес центрах. Описані основні переваги мови програмування Swift і фреймворку

SwiftUI, також основні особливості середовища розробки xCode - IDE. Були наведені і описані обрані продукти Firebase, використання яких допоможе в реалізації проєкту: Authentication, Firestore, Security Rules, Cloud Functions.

3 КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Структура проєкту

Для реалізації поставленої задачі – було обрано наступну схему навігації в програмному застосунку (рис. 3.1), яка складається з наступних сторінок:

- Main Tab Screen – головна сторінка, яка містить одразу 2 сторінки всередині та елемент інтерфейсу, при натискання на який – відбувається переключення між цими двома сторінками;
- Explore Screen – сторінка, на якій відображаються останні новини, список доступних фітнес центрів та подій;
- User Activity Passes Screen – сторінка, на якій користувач може отримати доступ до придбаних пропусків;
- Search Activities Screen – сторінка, на якій знаходиться пошуковий елемент інтерфейсу та список знайдених результатів;
- Activity Details Screen – сторінка, на якій користувач може побачити назву, дату, адресу, кнопка оплати, QR код та місцезнаходження фітнес центру, або події
- Sign in/Sign up Screen – сторінка, на якій користувач має можливість увійти у свій обліковий запис, або зареєструватись при відсутності облікового запису;
- News Details Screen – сторінка, на якій користувач побачить деталі новини, а саме: зображення, дату та текст новини.

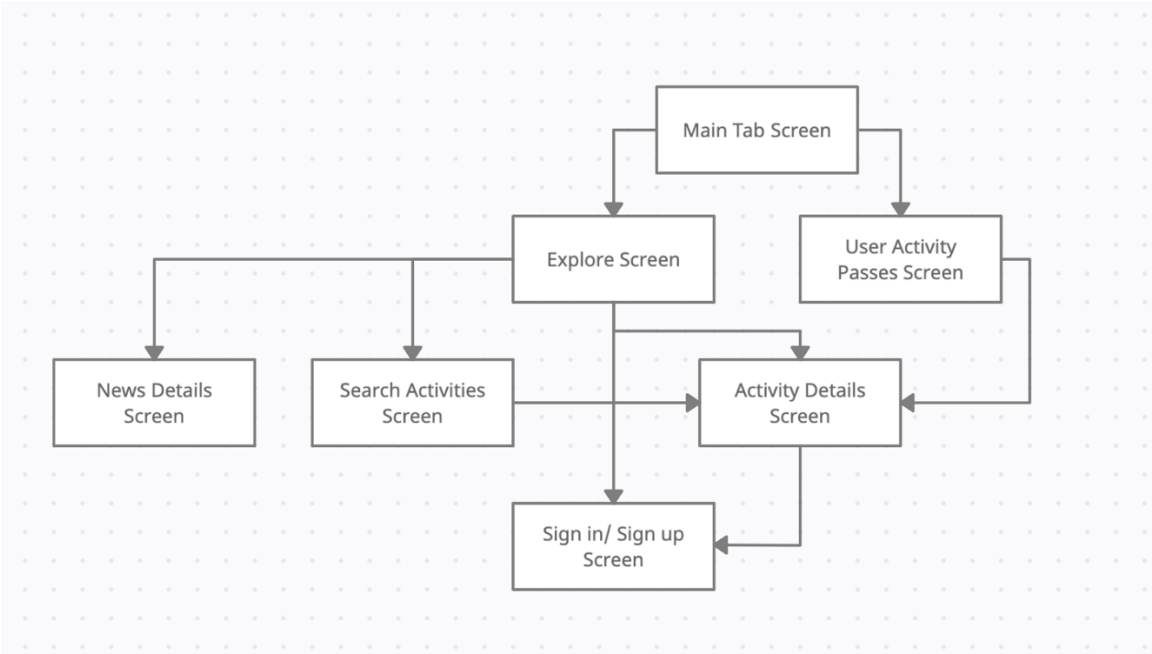


Рисунок 3.1 – Схема навігації в додатку

Файлова структура проєкту (рис. 3.2) була розроблена і структурована за допомогою директорій наступним чином:

- User Stories – директорія, в якій зберігаються внутрішні папки з файлами, які належать до програмного коду користувацького інтерфейсу та сторінок;
- Services – директорія, в якій зберігаються папки і файли з програмним кодом, які відносяться до різних сервісів, які використовує проєкт: сервіс роботи з сервером, сервіс бази даних, сервіс карт і локації, сервіс для роботи з даними користувача та сервіс виконання оплати;
- Extensions – директорія, в якій зберігаються файли з програмним кодом, який прискорює та допомагає в написанні програмного коду сторінок додатку.
- Supporting Files – директорія, в якій зберігаються різні допоміжні файли проєкту, який генерує xCode для роботи з внутрішніми сервісами Apple.

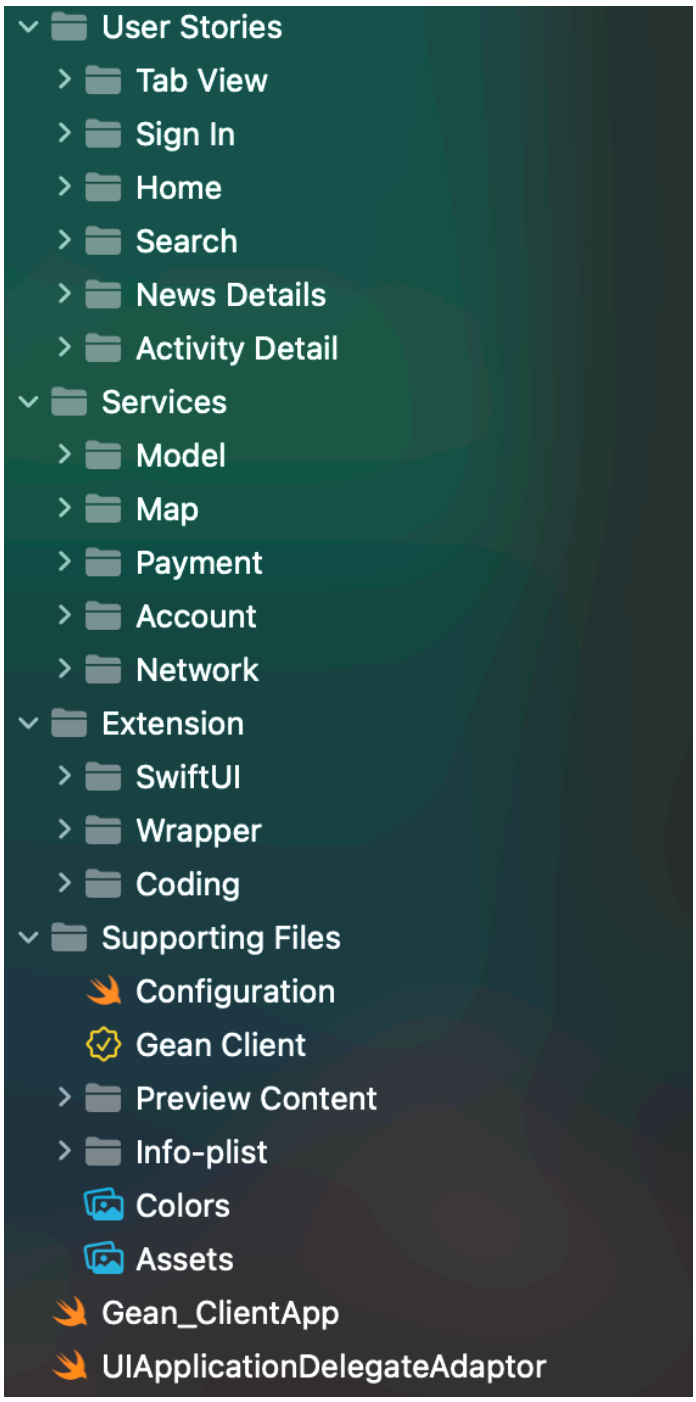


Рисунок 3.2 – Основна файлова структура проєкту

Оскільки, для проєкту була обрана архітектура MVVM (Model-View-View Model), то для кожної піддиректорії для User Stories, в якій міститься програмний код сторінок, було створено 3 директорії:

– Model – директорія, в якій зберігаються моделі та об’єкти необхідні для функціонування сторінки;

– View – директорія, в якій зберігаються файли з програмним кодом сторінки додатка;
– View Model – директорія, я в якій зберігаються файли, програмний код, яких необхідний для обробки та обчислення запитів від файлу з кодом користувацького інтерфейсу.

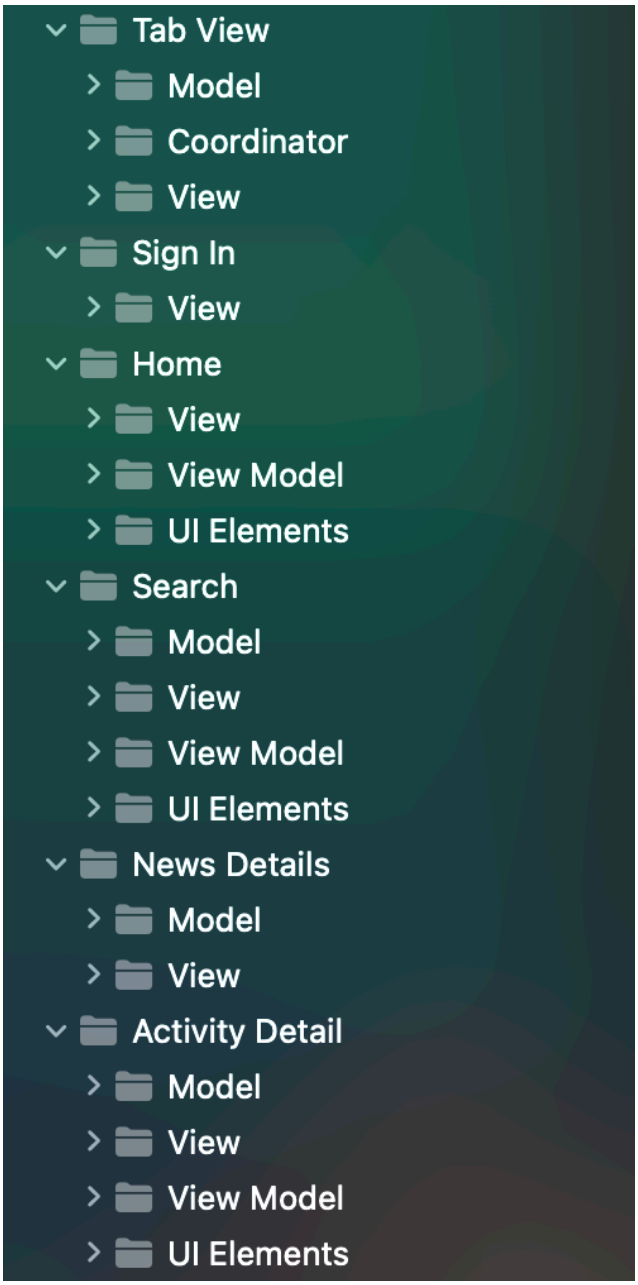


Рисунок 3.3 – Файлова структура сторінок додатку, використовуючи архітектуру MVVM

3.2 Опис класів

Детальний опис класів відображений у таблиці 3.1.

Таблиця 3.1 – Опис класів проєкту

Клас	Опис
AppDelegate	Стандартний клас, який містить методи для оповіщення про зміну життєвого циклу мобільного додатку
News	Клас, що описує структуру новини
Event	Клас, що описує структуру події
FitnessCenter	Клас, що описує структуру фітнес-центру
ActivityAddress	Клас, що описує структуру адреси
ActivityPass	Клас, що описує структуру доступу
ApplePayService	Клас, який відповідає за проведення оплати
NetworkService	Клас, який відповідає за роботу з мережею
AccountService	Клас, який відповідає за роботу з моделлю поточного користувача, його авторизацію та деавторизацію
ActivityDetailsView	Клас, що описує вікно деталей події, або фітнес-центру
ActivityDetailDescriptionSection	Клас, що описує вікно опису події, або фітнес-центру
ActivityDetailAccessSection	Клас, що описує вікно з кнопкою для оплати події, або фітнес-центру
ActivityDetailHeader	Клас, що описує вікно з назвою та зображенням події, або фітнес-центру
ActivityMapSectionSection	Клас, що описує вікно карти
ActivityDetailQRCodeSection	Клас, що описує вікно з QR кодом для доступу до події, або фітнес-центру
AcitvityDetailViewModel	Клас, що описує бізнес логіку вікна деталей подій
HomeView	Клас, що описує вікно початкової сторінки
HomeExploreSection	Клас, що описує вікно з пошуковим полем
HomeFitnessCentersSection	Клас, що описує вікно зі списком фітнес-центрів
HomeFitnessCentersSectionCell	Клас, що описує елемент зі списка фітнес-центрів

Продовження таблиці 3.1

HomeNavBar	Клас, що описує вікно з кнопкою логіну
HomeNewsSection	Клас, що описує вікно зі списком новин
HomeNewsSectionCell	Клас, що описує елемент зі списка новин
HomeUpcomingEvents Section	Клас, що описує вікно зі списком подій
HomeUpcomingEvents SectionCell	Клас, що описує елемент зі списка подій
HomeViewModel	Клас, що описує бізнес логіку вікна початкової сторінки
NewsDetailView	Клас, що описує вікно деталей новини
SearchView	Клас, що описує вікно пошуку події, або фітнес центру
SearchItemCell	Клас, що описує елемент зі списку пошуку
SearchTextFieldView	Клас, що описує текстове поле пошуку
SearchViewModel	Клас, який відповідальний за бізнес логіку пошукувого вікна
SignInView	Клас, що описує вікно авторизації
MainTabView	Клас, що описує вікно нижнього бара
MainTabBarItem	Клас, що описує елемент нижнього бара
MainTabBar	Клас, що описує вид нижнього бара

Таблиця 3.2 - Опис методів класів

Клас	Назва	Вхідні параметри	Опис
AppDelegate	applicationDidFinishLaunchingWithOptions	application : UIApplication launchOptions : [UILaunchOptionsKey : Any]	Метод, який викликається після повного завантаження додатку
AppDelegate	configureFirebase		Метод, який налаштовує API ключ Firebase

Продовження таблиці 3.2

AppDelegate	configureStripe		Метод, який налаштовує API ключ Stripe
News	setDocumentId		Метод, який встановлює унікальний ключ
News	getTitle		Метод, який повертає назву новини
News	getDescription		Метод, який повертає опис новини
News	getBackroundImageUrl		Метод, який повертає зображення новини
News	getCreatedAt		Метод, який повертає дату створення новини
Event	setDocumentId		Метод, який встановлює унікальний ключ
Event	getName		Метод, який повертає назву події
Event	getDescription		Метод, який повертає опис події
Event	getBackroundImageUrl		Метод, який повертає зображення події
Event	getAddress		Метод, який повертає адресу події
Event	getPrice		Метод, який повертає ціну події
Event	getDate		Метод, який повертає дату події
Event	getLocale		Метод, який повертає локалізацію події
FitnessCenter	setDocumentId		Метод, який встановлює унікальний ключ
FitnessCenter	getName		Метод, який повертає назву фітнес-центру
FitnessCenter	getDescription		Метод, який повертає опис фітнес-центру
FitnessCenter	getBackroundImageUrl		Метод, який повертає зображення фітнес-центру
FitnessCenter	getAddress		Метод, який повертає адресу фітнес-центру

Продовження таблиці 3.2

FitnessCenter	getPrice		Метод, який повертає ціну фітнес-центру
FitnessCenter	getLocale		Метод, який повертає локалізацію фітнес-центру
ActivityAddress	getDescription		Метод, який повертає назву адреси
ActivityAddress	getCoordinate		Метод, який повертає координати адреси
ActivityPass	setDocumentId		Метод, який встановлює унікальний ключ
ActivityPass	getQRCodeValue		Метод, який повертає значення QR коду
ActivityPass	getFitnessCenter		Метод, який повертає модель фітнес-центру
ActivityPass	setFitnessCenter	value: FitnessCenter	Метод, який встановлює модель фітнес-центру
ActivityPass	getFitnessCenterReference		Метод, який повертає зв’язок на фітнес-центр
ActivityPass	getEvent		Метод, який повертає модель події
ActivityPass	setEvent	value: Event	Метод, який встановлює модель події
ActivityPass	getEventReference		Метод, який повертає зв’язок на подію
ActivityPass	getEndDate		Метод, який повертає останій день доступу
ApplePayService	pay	activity: Activity?	Метод, який починає здійснювати оплату

Продовження таблиці 3.2

ApplePayService	applePayContext	context: STPApplePayContext, paymentMethod: STPPaymentMethod, completion: @escaping STPIntentClientSecretCompletionBlock	Метод, який відповідальний за створення Payment Intent
ApplePayService	applePayContext	context: STPApplePayContext, status: STPPaymentStatus, error: Error?	Метод, який відповідальний за обробку результату оплати
NetworkService	getFitnessCenters	searchQuery: String? = nil, limit: Int = 15	Метод, який відповідає за отримання фітнес-центрів з Firestore
NetworkService	getEvents	searchQuery: String? = nil, limit: Int = 15	Метод, який відповідає за отримання подій з Firestore
NetworkService	getNews		Метод, який відповідає за отримання новин з Firestore
NetworkService	getActivityPassList		Метод, який відповідає за отримання доступів з Firestore

Продовження таблиці 3.2

NetworkService	getActivityPass	fitnessCenterId: String? = nil, eventId: String? = nil	Метод, який відповідає за отримання доступу з Firestore
NetworkService	createStripePaymentIntent	userUid: String, fitnessCenterId: String?, eventId: String?	Метод, який відповідає за створення Payment Intent у Stripe
NetworkService	fetchCollectionFromFirestore	path: String, queryLimit: FirestoreQueryLimit? = nil, queryItems: [FirestoreQueryItem] = []	Метод, який відповідає за отримання моделей з Firestore

3.3 Архітектура бази даних

В якості сховища даних було вирішено використати Firestore, оскільки він дозволяє розв’язати поставлені задачі без заглиблення в розробку серверної частини. Firestore призначений для зберігання ієрархічних даних, які мають гнучку нереляційну схему. Розумний підхід до оцінки потенційного рішення в Firestore грає важливу роль, оскільки потрібно враховувати доступні квоти та певні обмеження. Враховуючи вищесказане, було створено архітектуру бази даних (рис. 3.4), яка необхідна для розв’язання поставлених задач.

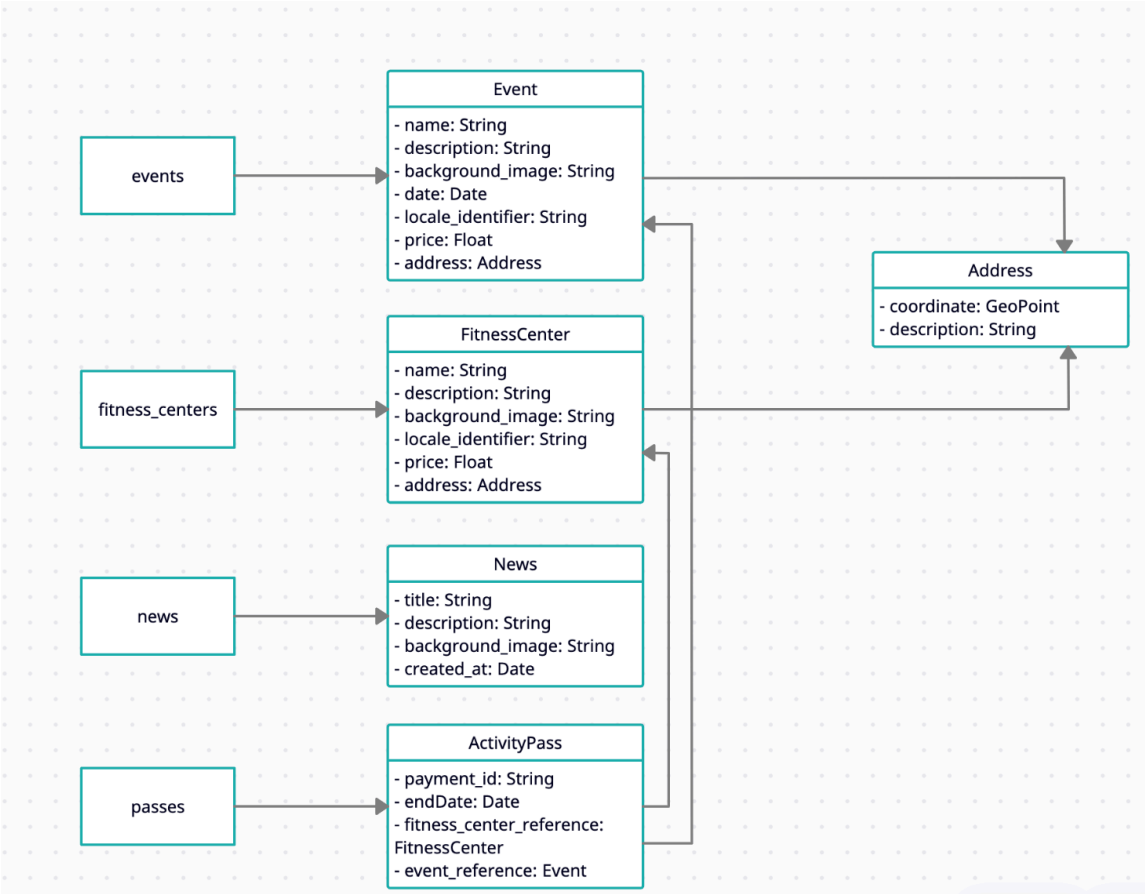


Рисунок 3.4 – Структура таблиць і записів в Firestore

Детальний опис класів відображений у таблиці 3.3.

Таблиця 3.3 – Опис бази даних

Назва колекції	Опис
events	Колекція, що відповідає за збереження подій.
fitness_centers	Колекція, що відповідає за фітнес-центрів.
news	Колекція, що відповідає за збереження новин.
passses	Колекція, що відповідає за збереження доступів до подій, або фітнес-центрів. Кожен елемент має зв’язок з колекцією events, або fitness_centers.

3.4 Робота з мережею

Для реалізації комунікації програмного застосунку з базою даних Firestore був використаний офіційний Firestore SDK. Попри простоту використання SDK, також був створений проміжний шар Network Layer, який

має покращити і пришвидшити повторне використання одного і того самого запиту до сховища даних.

Для розв’язання проблеми оплати – був використаний сервіс Stripe. Він дозволяє приймати оплату від клієнтів різними методами та валютами. Більш того, Stripe пропонує багатий на можливості API.

Процес створення запиту на оплату за допомогою Apple Pay та бібліотеки Stripe виглядає наступним чином:

1. після підтвердження користувачем наміру створити платіж Apple Pay генерує спеціальний унікальний токен банківської картки, яку обрав користувач;
2. мобільний додаток робить запит на створення Payment Intent в Stripe і передає токен банківської картки;
3. мобільний застосунок переходить в режим очікування і починає запити до Firestore для перевірки статусу оплати з інтервалом 1 секунди;
4. якщо на банківській картці присутня достатня сума для списання, Stripe надсилає запит на Firestore, що оплата пройшла успішно, і Firestore змінює статус оплати;
5. як тільки мобільний програмний застосунок побачить успішну оплату – він одразу вийде зі статусу очікування та сповістить користувача про успішну оплату.

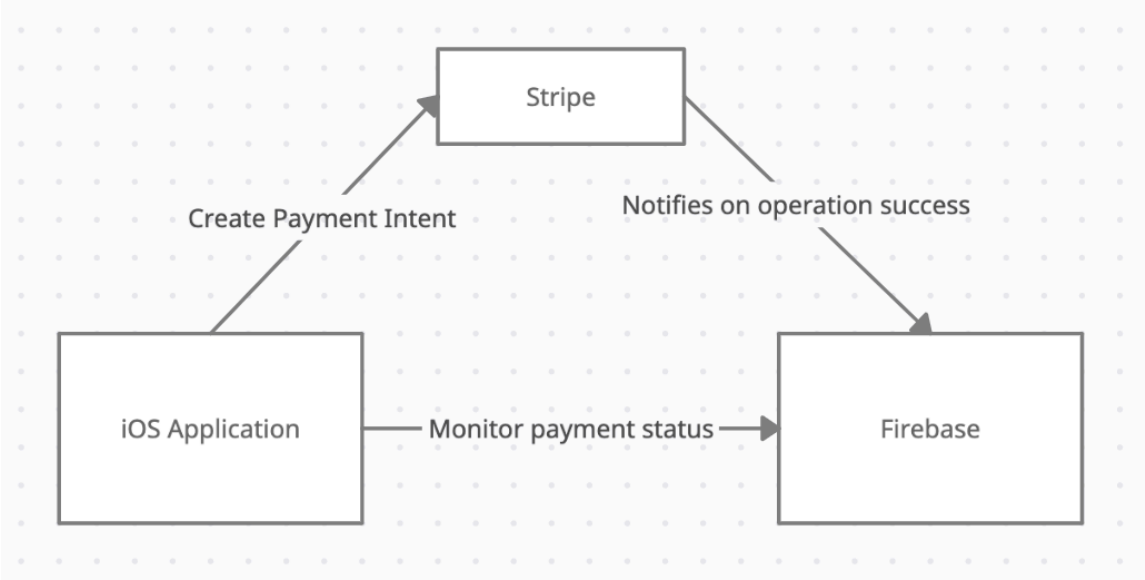


Рисунок 3.5 – Схема виконання оплати програмним застосунком

3.5 Безпека даних в базі даних

Для вирішення питання безпеки даних у Firestore – ми звернемо увагу на Security Rules. Цей інструмент дозволить нам зберегти конфіденціальність ресурсу та записів.

На рисунку 3.4 можна побачити записи, які зберігає Firestore. Правила безпеки для цих записів будуть виглядати наступним чином (рис. 3.6):

- всі записи мають заборону на редагування будь-яким користувачем, не залежно від статусу авторизації;
- записи fitness_centers дозволені для перегляду будь-якому користувачу, не залежно від статусу авторизації;
- записи events дозволені для перегляду будь-якому користувачу, не залежно від статусу авторизації;
- записи news дозволені для перегляду будь-якому користувачу, не залежно від статусу авторизації;
- записи passes дозволені для перегляду-лише авторизованому користувачу, і лише власнику запису.

```
rules_version = '2';
service cloud.firestore {

  match /databases/{database}/documents {

    match /news/{news} {
      allow read: if true;
    }

    match /fitness_centers/{fitness_center} {
      allow read: if true;
    }

    match /events/{event} {
      allow read: if true;
    }

    match /passes/{pass} {
      allow read: if request.auth.uid != null && get(resource.data.user_reference).id == request.auth.uid;
    }
  }
}
```

Рисунок 3.6 – Програмний код правил безпеки Firestore

Редагування записів news, fitness_centers та events заборонено усім користувачам, бо це є спільними записами, які доступні усім користувачам. Записи passes заборонені для редагування через те, що ці записи зберігають платіжні дані й дату валідності. Отже, будь-які маніпуляції з записами passes можуть нашкодити бізнес-логіці.

3.6 Висновки до розділу

У цьому розділі було представлено короткий розбір реалізації мобільного програмного застосунку для отримання послуг у фітнес-центрах.

Було описано варіанти використання технологій та представлені наочні приклади програмного коду.

Завдяки використанню певних підходів та технологій була реалізована і описана програмна частина диплома.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Способи тестування

Тестування буде проводитись методом Black Box. При тестуванні таким методом перевіряються функціональні можливості програмного додатка без знання внутрішньої структури коду та деталей реалізації. Тестування методом Black Box в основному зосереджується на вхідних даних та вихідних даних програмного додатка і повністю базується на вимогах і специфікаціях програмного забезпечення.

Основні типи тестування чорної скриньки:

- функціональне тестування - тип тестування чорного ящика пов’язаний з функціональними вимогами системи;
- нефункціональне тестування - тип тестування чорного ящика пов’язаний не з тестуванням певної функціональності, а з нефункціональними вимогами, такими як продуктивність, масштабованість, зручність використання;

Кроки для проведення тестування чорної скриньки:

- спочатку вивчаються вимоги та технічні характеристики системи;
- обирається дійсні вхідні дані (позитивний тестовий сценарій), щоб перевірити, чи програмний застосунок обробляє їх правильно. Крім того, деякі недійсні вхідні дані (негативний тестовий сценарій) вибираються для перевірки того, що програмний додаток здатний їх виявити;
- визначаються очікувані результати для всіх цих входів;
- створюються тестові випадки з вибраними входами;
- тестові випадки виконуються;
- виконується порівняння фактичних результатів з очікуваними;
- дефекти, якщо такі є, усуваються та перевіряються.

4.2 Опис тестування покупки доступу до фітнес-центру

Для початку тестування і перевірки позитивного сценарію – потрібно переконатись у правильних вхідних даних:

- користувач має стабільне підключення до інтернету;
- користувач авторизувався у свій обліковий запис;
- користувач виконує покупку у фітнес-центрі, до якого ще немає доступу.

Вихідні дані для позитивного сценарію:

- користувач побачить QR код на сторінці фітнес-центру.

Після того, як програмний застосунок запустився – необхідно авторизуватись у свій обліковий запис та перейти на сторінку фітнес центру, де ми очікуємо побачити кнопку «Get access» (рис. 4.1).

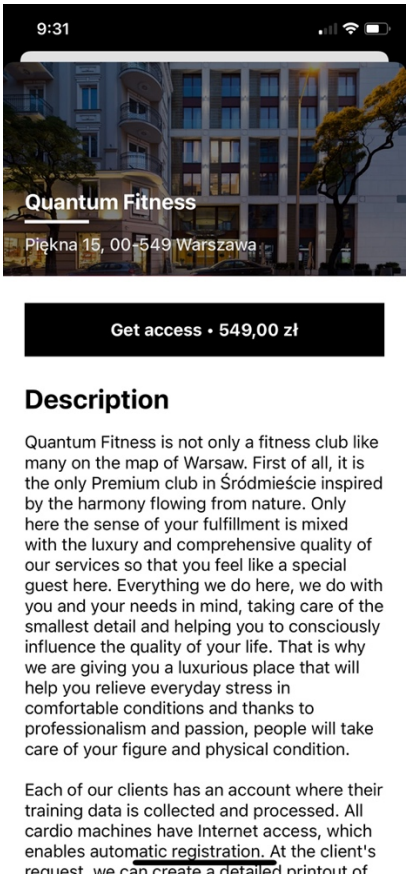


Рисунок 4.1 – Сторінка фітнес-центру, до якого немає доступу

Як тільки ми впевнимось у задовільнені вхідних даних – можна почати процес оплати доступу в фітнес-центр. Отже після натискання на кнопку «Get

access» у нас з’явиться вікно Apple Pay, де користувач підтверджує свій намір оплатити (рис. 4.2).

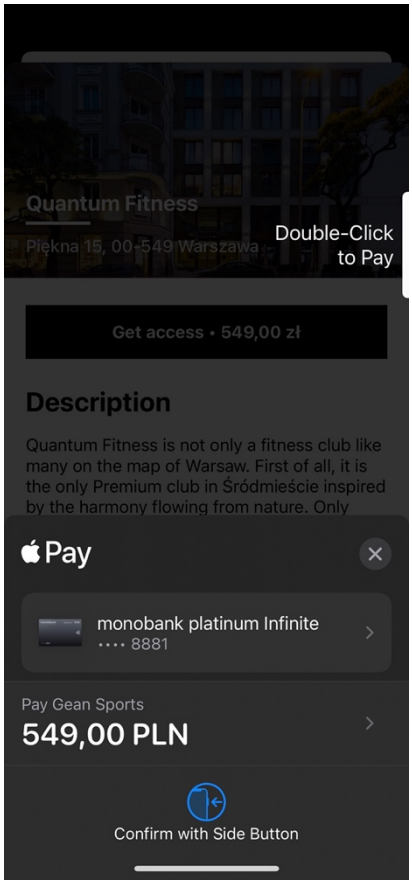


Рисунок 4.2 – Вікно оплати Apple Pay

Як тільки оплата пройде успішно – ми зможемо побачити, що на сторінці фітнес центру кнопка «Get access» замінилась на QR код, що є доступом до фітнес центру (рис. 4.3). Отже, вихідні дані є теж задовільненими, і цим завершується тестування успішного сценарію.

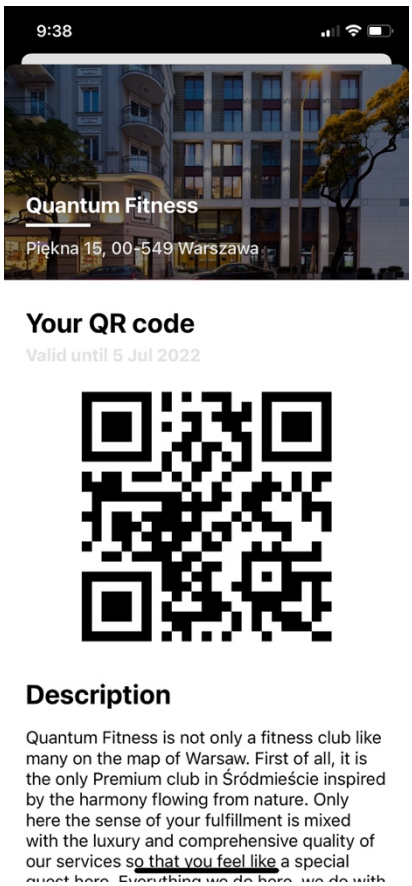


Рисунок 4.3 – Сторінка фітнес-центру, до якого є доступу

Для перевірки негативного сценарію, коли користувач має намір оплатити доступ до фітнес центру без актуальної авторизації, у нас будуть наступні вхідні дані:

- користувач має стабільне підключення до інтернету;
- користувач ще не авторизувався у свій обліковий запис;
- користувач виконує покупку у фітнес-центрі, до якого ще немає доступу.

Вихідні дані для негативного сценарію:

- користувач побачить сторінку авторизації.

Після того, як програмний застосунок запустився – необхідно одразу перейти на сторінку фітнес центру, де ми очікуємо побачити кнопку «Get access» (рис. 4.1).

Спробуємо натиснути на кнопку «Get access» та одразу бачимо сторінку авторизації (рис. 4.4), тим самим ми бачимо, що програма не пропустила не авторизованого користувача до вікна оплати. Отже, вихідні дані є задовільненими, і цим завершується тестування негативного сценарію.

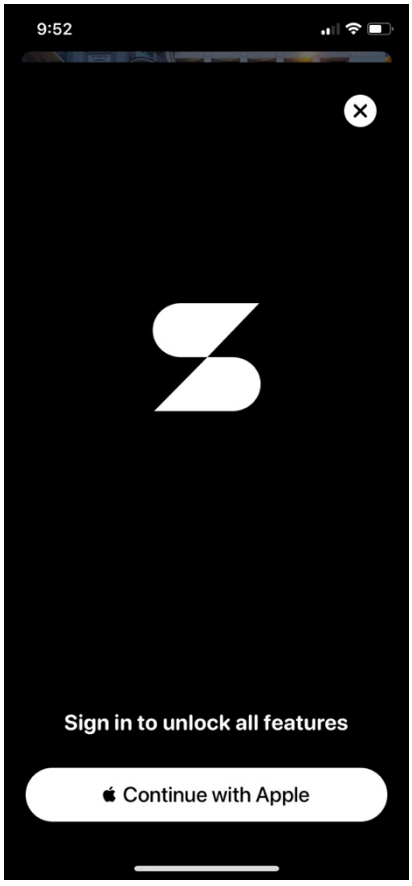


Рисунок 4.4 – Сторінка авторизації

4.3 Висновки до розділу

У цьому розділі був розглянутий метод тестування Black Box, та приведений наочний приклад його застосування для успішного та негативного сценаріїв.

Тестування програмного застосунку було проведено успішно, що свідчить про те, що мобільний додаток був реалізований згідно із заявленими раніше вимогам та має простий у використанні інтерфейс.

5 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання даного програмного додатку необхідно згенерувати .ipa файл засобами середовища розробки xCode і відправити в офіційний магазин мобільних програмних застосунків App Store. Файл програмного забезпечення .ipa являє собою упакований і скомпільований код програми та зазвичай зберігається в App Store.

Для завантаження додатку необхідно мати інтернет підключення, та операційну систему iOS 15 і вище.

5.2 Робота з програмним забезпеченням

Інструкція з користуванням програмного забезпечення описана у додатку.

5.3 Висновки до розділу

В даному розділі було описано умови розгортання iOS-застосунку.

ВИСНОВКИ

В результаті виконання дипломної роботи було створено мобільний програмний застосунок для отримання послуг у фітнес центрах. Кожен з етапів розробки дипломної роботи був описаний у відповідному розділі.

У першому розділі були наведені загальні положення та представлені вже існуючі, на ринку, рішення. В результаті аналізу, можна стверджувати, що на сьогоднішній день є альтернативні рішення, хоча кожне з них не повторює один одного, і кожне з рішень має свої переваги та недоліки.

У другому розділі було обрано технології, які були використані при розробці мобільного додатка, в результаті аналізу вимог до системи.

У третьому розділі було виконано моделювання та аналіз розробленого програмного застосунку.

У четвертому розділі було описано метод Black Box Testing, який був використаний у створенні тестового плану і виконанні тестування програмного продукту.

П’ятий розділ містить інструкцію розгортання програмного застосунку.

Отже, розроблений програмний застосунок має сучасний інтерфейс, враховує недоліки і переваги існуючих рішень та задовольняє поставлені вимоги.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Swift - The powerful programming language that is also easy to learn [Електронний ресурс]. – Режим доступу: https://developer.apple.com/swift/

2) SwiftUI [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/swiftui/

3) Xcode [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/xcode

4) Introducing MVVM into your SwiftUI project [Електронний ресурс]. – Режим доступу: https://www.hackingwithswift.com/books/ios-swiftui/introducing-mvvm-into-your-swiftui-project

5) How to use the coordinator pattern in iOS apps [Електронний ресурс]. – Режим доступу: https://www.hackingwithswift.com/articles/71/how-to-use-the-coordinator-pattern-in-ios-apps

6) Building scalable applications with Firestore [Електронний ресурс]. – Режим доступу: https://cloud.google.com/architecture/building-scalable-apps-with-cloud-firestore

7) Cloud Firestore [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/firestore

8) Get started with Cloud Firestore [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/firestore/quickstart

9) Implementing User Authentication with Sign in with Apple [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/authenticationservices/implementing_user_authentication_with_sign_in_with_apple

10) Authenticate Using Apple [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/auth/ios/apple

11) Firebase Security Rules [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/rules

12) Get started: write, test, and deploy your first functions [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/functions/get-started

13) Best Practices in SwiftUI Composition [Електронний ресурс]. – Режим доступу: https://betterprogramming.pub/best-practices-in-swiftui-composition-282b02772a24

14) SwiftUI Microservices [Електронний ресурс]. – Режим доступу: https://betterprogramming.pub/swiftui-microservices-c7002228710

15) How to Create QR Codes in Swift [Електронний ресурс]. – Режим доступу: https://medium.com/codex/qr-codes-are-simple-in-swift-6d203ebc3f5b

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ ОТРИМАННЯ ПОСЛУГ У ФІТНЕС ЦЕНТРАХ

Опис програми

КПІ.ІП-8109.045440.03.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Владислав Гордійчук

Київ – 2022

ТЕКСТ ПРОГРАМНОГО КОДУ

Тексти програмного коду

Програмний застосунок для отримання послуг у фітнес-центрах

(Найменування програми (документа))

CD-R

(Вид носія даних)

42 арк, 1126 Кб

(Обсяг програми, арк., Кб)

Київ – 2022

```swift
//
//  UIApplicationDelegateAdaptor.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import SwiftUI
import FirebaseCore
import Stripe

final class AppDelegate: NSObject, UIApplicationDelegate {

    // MARK: - Methods

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool {

        configureFirebase()

        configureStripe()

        return true
    }

    // MARK: - Configure

    private func configureFirebase() {

        FirebaseApp.configure()
    }

    private func configureStripe() {

        StripeAPI.defaultPublishableKey =
            "pk_test_51L0k7EKdWiti1dx9x5E3o0Yh6wvil3556XsXppuqbQatIpQPc5EgoeZPhrvZ1nsExUrloByS8qj
            i6YCwq4R1MZZB00Zhnd6Wj5"
    }
}
```

```swift
//
//  Gean_ClientApp.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/7/22.
//

import SwiftUI

@main
struct Gean_ClientApp: App {

    @UIApplicationDelegateAdaptor(AppDelegate.self) var appDelegate

    var body: some Scene {

        WindowGroup {

            MainTabView()
        }
    }
}
```

```swift
//
//  News.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/22/22.
//

import Foundation
import FirebaseService

final class News: Identifiable, FirestoreObject, Codable {

    internal var id: String? {

        return documentId
    }

    private var documentId: String?

    private var title: String?

    private var description: String?

    private var backgroundImageUrl: URL?

    private var createdAt: Date?

    // MARK: - Coding Keys

    enum CodingKeys: String, CodingKey {

        case title, description
        case backgroundImageUrl = "background_image"
        case createdAt = "created_at"
    }
```

```swift
    // MARK: - Interaction Methods

    public func setDocumentId(_ value: String) {

        documentId = value
    }

    public func getTitle() -> String? {

        return title
    }

    public func getDescription() -> String? {

        return description
    }

    public func getBackgroundImageUrl() -> URL? {

        return backgroundImageUrl
    }

    public func getCreatedAt() -> Date? {

        return createdAt
    }
}
```

```swift
//
//  Activity.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import Foundation

protocol Activity: AnyObject {

    func getId() -> String?

    func getName() -> String?

    func getDescription() -> String?

    func getBackgroundImageUrl() -> URL?

    func getDate() -> Date?

    func getAddress() -> ActivityAddress?

    func getPrice() -> Float?

    func getLocale() -> Locale?
}

extension Activity {

    func getDate() -> Date? { return nil }

    func getLocalizedDate() -> String? {

        guard let date = getDate()
        else { return nil }

        let dateFormatter = DateFormatter()
        dateFormatter.setLocalizedDateFormatFromTemplate("dLLLjmma")

        return dateFormatter.string(from: date)
    }

    func getLocalizedPrice() -> String? {

        guard let price = getPrice(),
              let locale = getLocale()
        else { return nil }

        let numberFormatter = NumberFormatter()
        numberFormatter.numberStyle = .currency
        numberFormatter.locale = locale

        return numberFormatter.string(from: NSNumber(value: price))
    }
}
```

```swift
//
//  FitnessCenter.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import Foundation

final class FitnessCenter: Identifiable, FirestoreObject, Activity, Codable {

    // MARK: - Variables

    internal var id: String? {

        return documentId
    }

    private var documentId: String?

    private var name: String?

    private var description: String?

    private var backgroundImageUrl: URL?

    private var address: ActivityAddress?

    private var price: Float?

    private var localeIdentifier: String?

    // MARK: - Coding Keys

    enum CodingKeys: String, CodingKey {

        case documentId, name, description
        case backgroundImageUrl = "background_image"
        case address, price
        case localeIdentifier = "locale_identifier"
    }

    // MARK: - Interaction Methods

    func setDocumentId(_ value: String) {

        documentId = value
    }

    func getId() -> String? {

        return id
    }

    func getName() -> String? {

        return name
    }

    func getDescription() -> String? {

        return description
    }

    func getBackgroundImageUrl() -> URL? {

        return backgroundImageUrl
    }

    func getAddress() -> ActivityAddress? {

        return address
    }
```

```swift
    func getPrice() -> Float? {

        return price
    }

    func getLocale() -> Locale? {

        guard let localeIdentifier = localeIdentifier
        else { return nil }

        return Locale(identifier: localeIdentifier)
    }
}
```

```swift
//
//  Event.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import FirebaseFirestore

final class Event: Identifiable, FirestoreObject, Activity, Codable {

    // MARK: - Variables

    internal var id: String? {

        return documentId
    }

    private var documentId: String?

    private var name: String?

    private var description: String?

    private var backgroundImageUrl: URL?

    private var date: Timestamp?

    private var address: ActivityAddress?

    private var price: Float?

    private var localeIdentifier: String?

    // MARK: - Coding Keys

    enum CodingKeys: String, CodingKey {

        case documentId, name, description
        case backgroundImageUrl = "background_image"
```

```
        case date, address, price
        case localeIdentifier = "locale_identifier"
    }

    // MARK: - Interaction Methods

    func setDocumentId(_ value: String) {

        documentId = value
    }

    func getId() -> String? {

        return id
    }

    func getName() -> String? {

        return name
    }

    func getDescription() -> String? {

        return description
    }

    func getBackgroundImageUrl() -> URL? {

        return backgroundImageUrl
    }

    func getAddress() -> ActivityAddress? {

        return address
    }

    func getPrice() -> Float? {

        return price
    }
```

```
    func getDate() -> Date? {

        return date?.dateValue()
    }

    func getLocale() -> Locale? {

        guard let localeIdentifier = localeIdentifier
        else { return nil }

        return Locale(identifier: localeIdentifier)
    }
}
```

```swift
//  ActivityAddress.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import FirebaseFirestore
import CoreLocation

final class ActivityAddress: Codable {

    private var description: String?

    private var coordinate: GeoPoint?

    // MARK: - Coding Keys

    enum CodingKeys: String, CodingKey {

        case description, coordinate
    }

    // MARK: - Interaction Methods

    func getDescription() -> String? {

        return description
    }

    func getCoordinate() -> CLLocationCoordinate2D? {

        guard let latitude = coordinate?.latitude,
              let longitude = coordinate?.longitude
        else { return nil }

        return CLLocationCoordinate2D(
            latitude: latitude,
            longitude: longitude
        )
    }
}
```

```swift
//
//  ActivityPass.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import Foundation
import FirebaseFirestore

final class ActivityPass: Identifiable, FirestoreObject, Codable {

    // MARK: - Variables

    internal var id: String? {

        return documentId
    }

    private var documentId: String?

    private var fitnessCenter: FitnessCenter?

    private var fitnessCenterReference: DocumentReference?

    private var event: Event?

    private var eventReference: DocumentReference?

    private var endDate: Date?

    // MARK: - Coding Keys

    enum CodingKeys: String, CodingKey {

        case documentId
        case fitnessCenterReference = "fitness_center_reference"
        case eventReference = "event_reference"
        case endDate = "end_date"
    }
```

```swift
    // MARK: - Interaction Methods

    func setDocumentId(_ value: String) {

        documentId = value
    }

    func getQRCodeValue() -> String? {

        return documentId
    }

    func getFitnessCenter() -> FitnessCenter? {

        return fitnessCenter
    }

    func setFitnessCenter(_ value: FitnessCenter) {

        fitnessCenter = value
    }

    func getFitnessCenterReference() -> DocumentReference? {

        return fitnessCenterReference
    }

    func getEvent() -> Event? {

        return event
    }

    func setEvent(_ value: Event) {

        event = value
    }

    func getEventReference() -> DocumentReference? {

        return eventReference
    }
```

```
84      func getEndDate() -> Date? {
85
86          return endDate
87      }
88  }
```

```
1   //
2   //  AuthStatus.swift
3   //  Gean Client
4   //
5   //  Created by Vlad Gordiichuk on 6/4/22.
6   //
7
8   import Foundation
9
10  extension AccountService {
11
12      // MARK: - AuthStatus
13
14      public enum AuthStatus {
15
16          case authenticated, notAuthenticated
17      }
18  }
```

```
1   //
2   //  AccountService.swift
3   //  Gean Client
4   //
5   //  Created by Vlad Gordiichuk on 5/22/22.
6   //
7
8   import SwiftUI
9   import Combine
10  import FirebaseAuth
11  import FirebaseService
12
13  final class AccountService: ObservableObject {
14
15      // MARK: - Variables
16
17      static let shared = AccountService()
18
19      private var cancellables = Set<AnyCancellable>()
20
21      // MARK: - Binding Variables
22
23      @Published var authStatus: AccountService.AuthStatus?
24
25      @Published var activityPassList: [ActivityPass]?
26
27      // MARK: - Init
28
29      private init() {
30
31          setInitialValues()
32
33          startAuthListener()
34      }
```

```
36      // MARK: - Interaction Methods
37
38      public func signOut() throws {
39
40          try Auth.auth().signOut()
41
42          refreshAccountData()
43      }
44
45      // MARK: - Methods
46
47      private func setInitialValues() {
48
49          setAuthStatus(Auth.auth().currentUser)
50
51          refreshAccountData()
52      }
53
54      private func refreshAccountData() {
55
56          if authStatus == .authenticated {
57
58              Task(priority: .background) {
59
60                  async let refreshActivityPasses: ()? = try? refreshActivityPasses()
61
62                  await (refreshActivityPasses)
63              }
64          } else {
65
66              activityPassList = nil
67          }
68      }
69
70      private func startAuthListener() {
71
72          let subject = PassthroughSubject<User?, Error>()
73
74          Auth.auth().addStateDidChangeListener { (auth, user) in
```

```
75
76              subject.send(user)
77          }
78
79          subject.sink { _ in } receiveValue: { [weak self] user in
80
81              guard let self = self
82              else { return }
83
84              self.setAuthStatus(user)
85
86              self.refreshAccountData()
87          }.store(in: &cancellables)
88      }
89
90      private func setAuthStatus(_ user: User?) {
91
92          authStatus = user != nil ? .authenticated : .notAuthenticated
93      }
94  }
```

```swift
//
//  AccountService+ActivityPasses.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import Foundation

extension AccountService {

    // MARK: - Interaction Methods

    public func refreshActivityPasses() async throws {

        let activityPassList = try await NetworkService.getActivityPassList()

        await MainActor.run {

            self.activityPassList = activityPassList
        }
    }

    public func appendActivityPass(_ activityPass: ActivityPass) {

        if activityPassList == nil {

            activityPassList = [activityPass]
        } else {

            activityPassList?.append(activityPass)
        }
    }
}
```

```swift
//
//  MapService.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import UIKit
import CoreLocation

final class MapService {

    // MARK: - Interaction Methods

    static func canOpen(provider: MapService.Provider) -> Bool {

        guard let url = provider.getHostURL()
        else { return false }

        return UIApplication.shared.canOpenURL(url)
    }

    static func open(provider: MapService.Provider, _ coordinate: CLLocationCoordinate2D) {

        guard let url = provider.getUrl(coordinate)
        else { return }

        UIApplication.shared.open(url)
    }
}
```

```swift
//
//  MapService+Provider.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import Foundation
import CoreLocation

extension MapService {

    enum Provider: CaseIterable {

        // MARK: - Cases

        case appleMaps
        case googleMaps

        // MARK: - Interaction Methods

        public func getTitle() -> String {

            switch self {
            case .appleMaps:

                return "Apple Maps"
            case .googleMaps:

                return "Google Maps"
            }
        }

        public func getHostURL() -> URL? {

            switch self {
            case .appleMaps:

                return URL(string: "maps://")
```

```
            case .googleMaps:

                return URL(string: "comgooglemaps://")
            }
        }

        public func getUrl(_ coordinate: CLLocationCoordinate2D) -> URL? {

            guard let hostUrlRaw = getHostURL()?.absoluteString
            else { return nil }

            let rawUrl: String = {

                switch self {
                case .appleMaps:

                    return String(
                        format: "%@?saddr=&daddr=%f,%f",
                        hostUrlRaw,
                        coordinate.latitude,
                        coordinate.longitude
                    )
                case .googleMaps:

                    return String(
                        format: "%@?saddr=&daddr=%f,%f&directionsmode=driving",
                        hostUrlRaw,
                        coordinate.latitude,
                        coordinate.longitude
                    )
                }
            }()

            return URL(string: rawUrl)
        }
    }
}
```

```
//
//  StripePaymentIntent.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/29/22.
//

import Foundation

final class StripePaymentIntent: Codable {

    // MARK: - Variables

    var id: String

    var clientSecret: String

    // MARK: - Coding Keys

    enum CodingKeys: String, CodingKey {

        case id
        case clientSecret = "client_secret"
    }
}
```

```swift
//
//  ApplePayError.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import Foundation

enum ApplePayError: Error {

    case malformedRequest
}

extension ApplePayError: LocalizedError {

    public var errorDescription: String? {

        switch self {
        case .malformedRequest:

            return NSLocalizedString(
                "There was some issue with Apple Pay. Try again later.",
                comment: "Some request parameters are missing or invalid."
            )
        }
    }
}
```

```swift
//
//  ApplePayService.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/22/22.
//

import FirebaseAuth
import Stripe
import PassKit

final class ApplePayService: NSObject {

    // MARK: - Variables

    private var activity: Activity?

    private var applePayCheckedThrowingContinuation: UnsafeContinuation<Void, Error>?

    // MARK: - Interaction Methods

    func pay(for activity: Activity?) async throws {

        guard let activity = activity,
              let price = activity.getPrice(),
              let regionCode = activity.getLocale()?.regionCode,
              let currencyCode = activity.getLocale()?.currencyCode
        else { throw ApplePayError.malformedRequest }

        self.activity = activity

        try await withUnsafeThrowingContinuation { (continuation: UnsafeContinuation<Void, Error>)
            in

            applePayCheckedThrowingContinuation = continuation

            let paymentRequest = StripeAPI.paymentRequest(
                withMerchantIdentifier: Configuration.Services.Stripe.merchantIdentifier,
                country: regionCode,
                currency: currencyCode
```

```
            let decimalNumber = NSDecimalNumber(value: price)
                .rounding(
                    accordingToBehavior: NSDecimalNumberHandler(
                        roundingMode: .plain,
                        scale: 2,
                        raiseOnExactness: false,
                        raiseOnOverflow: false,
                        raiseOnUnderflow: false,
                        raiseOnDivideByZero: true
                    )
                )

            paymentRequest.requiredShippingContactFields = []
            paymentRequest.requiredBillingContactFields = []

            paymentRequest.paymentSummaryItems = [
                PKPaymentSummaryItem(
                    label: "Gean Sports",
                    amount: decimalNumber
                )
            ]

            DispatchQueue.main.async {

                let applePayContext = STPApplePayContext(paymentRequest: paymentRequest, delegate:
                    self)
                applePayContext?.presentApplePay()
            }
        }
    }
}

extension ApplePayService: STPApplePayContextDelegate {

    // MARK: - STPApplePayContextDelegate

    func applePayContext(_ context: STPApplePayContext, didCreatePaymentMethod paymentMethod:
        STPPaymentMethod, paymentInformation: PKPayment, completion: @escaping
        STPIntentClientSecretCompletionBlock) {

        Task {

            do {

                let eventId = (activity as? Event)?.getId()
                let fitnessCenterId = (activity as? FitnessCenter)?.getId()

                guard let userUid = Auth.auth().currentUser?.uid,
                      (eventId != nil || fitnessCenterId != nil)
                else { throw ApplePayError.malformedRequest }

                let paymentIntent = try await NetworkService.createStripePaymentIntent(
                    userUid: userUid,
                    fitnessCenterId: fitnessCenterId,
                    eventId: eventId
                )

                completion(paymentIntent.clientSecret, nil)
            } catch let error {

                completion(nil, error)
            }
        }
    }

    func applePayContext(_ context: STPApplePayContext, didCompleteWith status: STPPaymentStatus,
        error: Error?) {

        switch status {
        case .success:

            applePayCheckedThrowingContinuation?.resume()
        case .userCancellation:

            return
        default:

            let error = error ?? NSError(domain: "com.stripe.apple-pay", code: 404)

            applePayCheckedThrowingContinuation?.resume(throwing: error)
```

```swift
//
//  AppUrl.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import Foundation

enum AppUrl {

    static let stripeCreatePaymentIntent =
        "https://us-central1-gean-sports.cloudfunctions.net/stripeCreatePaymentIntent"
}
```

```swift
//
//  FirestoreError.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import Foundation

enum FirestoreError: Error {

    case malformedRequest,
         authRequired
}

extension FirestoreError: LocalizedError {

    public var errorDescription: String? {

        switch self {
        case .malformedRequest:

            return NSLocalizedString(
                "There was some issue. Try again later.",
                comment: "Some request parameters are missing or invalid."
            )
        case .authRequired:

            return NSLocalizedString(
                "You should be signed in.",
                comment: "Authorization is required to fetch this data."
            )
        }
    }
}
```

```swift
//
//  FirestoreObject.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import Foundation

protocol FirestoreObject {

    func setDocumentId(_ value: String)
}
```

```swift
//
//  FirestoreQueryItem.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import Foundation
import FirebaseFirestore

public struct FirestoreQueryItem {

    public let fieldPath: FieldPath?

    public let key: String?

    public let type: FirestoreQueryItemType

    public let values: [Any]

    public init(_ fieldPath: FieldPath, _ type: FirestoreQueryItemType, value: Any) {

        self.fieldPath = fieldPath
        self.key = nil
        self.type = type
        self.values = [value]
    }

    public init(_ fieldPath: FieldPath, _ type: FirestoreQueryItemType, values: [Any]) {

        self.fieldPath = fieldPath
        self.key = nil
        self.type = type
        self.values = values
    }

    public init(_ key: String, _ type: FirestoreQueryItemType, value: Any) {

        self.fieldPath = nil
        self.key = key
        self.type = type
        self.values = [value]
    }

    public init(_ key: String, _ type: FirestoreQueryItemType, values: [Any]) {

        self.fieldPath = nil
        self.key = key
        self.type = type
        self.values = values
    }
}
```

```swift
//
//  FirestoreQueryItemType.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import Foundation

public enum FirestoreQueryItemType {

    case isEqualTo,
         isNotEqualTo,
         isLessThan,
         isLessThanOrEqualTo,
         isGreaterThan,
         isGreaterThanOrEqualTo,
         arrayContains,
         arrayContainsAny,
         `in`,
         notIn
}
```

```swift
//
//  FirestoreQueryLimit.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import SwiftUI
import FirebaseFirestore

public struct FirestoreQueryLimit {

    public var limit: Int

    public var orderBy: String

    public var descending: Bool

    @Binding public var lastDocumentSnapshot: DocumentSnapshot?

    public init(
        limit: Int,
        orderBy: String,
        descending: Bool,
        lastDocumentSnapshot: Binding<DocumentSnapshot?>
    ) {

        self.limit = limit

        self.orderBy = orderBy

        self.descending = descending

        self._lastDocumentSnapshot = lastDocumentSnapshot
    }
}
```

```swift
//
//  NetworkService.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import FirebaseFirestore
import FirebaseAuth

final class NetworkService {

    // MARK: - Fitness Centers

    static public func getFitnessCenters(searchQuery: String? = nil, limit: Int = 15) async throws -> [FitnessCenter] {

        var queryItems = [FirestoreQueryItem]()

        if let searchQuery = searchQuery {

            queryItems += [
                FirestoreQueryItem("name", .isGreaterThanOrEqualTo, value: searchQuery),
                FirestoreQueryItem("name", .isLessThanOrEqualTo, value: searchQuery + "~")
            ]
        }

        return try await fetchCollectionFromFirestore(
            path: "fitness_centers",
            queryLimit: FirestoreQueryLimit(
                limit: limit,
                orderBy: FitnessCenter.CodingKeys.name.stringValue,
                descending: true,
                lastDocumentSnapshot: .constant(nil)
            ),
            queryItems: queryItems
        )
    }

    // MARK: - Events

    static public func getEvents(searchQuery: String? = nil, limit: Int = 15) async throws -> [Event] {

        if let searchQuery = searchQuery {

            return try await fetchCollectionFromFirestore(
                path: "events",
                queryLimit: FirestoreQueryLimit(
                    limit: limit,
                    orderBy: Event.CodingKeys.name.stringValue,
                    descending: false,
                    lastDocumentSnapshot: .constant(nil)
                ),
                queryItems: [
                    FirestoreQueryItem("name", .isGreaterThanOrEqualTo, value: searchQuery),
                    FirestoreQueryItem("name", .isLessThanOrEqualTo, value: searchQuery + "~")
                ]
            )
        } else {
```

```
            return try await fetchCollectionFromFirestore(
                path: "events",
                queryLimit: FirestoreQueryLimit(
                    limit: limit,
                    orderBy: Event.CodingKeys.date.stringValue,
                    descending: true,
                    lastDocumentSnapshot: .constant(nil)
                ),
                queryItems: [
                    FirestoreQueryItem("date", .isGreaterThanOrEqualTo, value: Date.now)
                ]
            )
        }
    }

    // MARK: - News

    static public func getNews() async throws -> [News] {

        return try await fetchCollectionFromFirestore(
            path: "news",
            queryLimit: FirestoreQueryLimit(
                limit: 15,
                orderBy: News.CodingKeys.createdAt.stringValue,
                descending: true,
                lastDocumentSnapshot: .constant(nil)
            )
        )
    }

    // MARK: - Passes

    static public func getActivityPassList() async throws -> [ActivityPass] {

        guard let userUid = Auth.auth().currentUser?.uid
        else { throw FirestoreError.authRequired }

        let reference = Firestore
            .firestore()
            .collection("users")
            .document(userUid)

        let activityPassCollection = try await fetchCollectionFromFirestore(
            path: "passes",
            queryItems: [
                FirestoreQueryItem("user_reference", .isEqualTo, value: reference),
                FirestoreQueryItem("end_date", .isGreaterThan, value: Date.now)
            ]
        ) as [ActivityPass]

        let fitnessCenterQueryItems: [FirestoreQueryItem] = activityPassCollection.compactMap {

            guard let documentId = $0.getFitnessCenterReference()?.documentID
            else { return nil }
```

```
            return FirestoreQueryItem(FieldPath.documentID(), .isEqualTo, value: documentId)
        }

        let fitnessCenterReferences = activityPassCollection
            .compactMap { $0.getFitnessCenterReference() }

        let eventReferences = activityPassCollection
            .compactMap { $0.getEventReference() }

        if !fitnessCenterQueryItems.isEmpty {

            let fitnessCenterCollection = try await fetchCollectionFromFirestore(
                path: "fitness_centers",
                queryItems: [
                    FirestoreQueryItem(FieldPath.documentID(), .in, values: fitnessCenterReferences)
                ]
            ) as [FitnessCenter]

            for activityPass in activityPassCollection {

                guard let fitnessCenterId = activityPass.getFitnessCenterReference()?.documentID,
                      let fitnessCenter = fitnessCenterCollection.first(where: { $0.getId() == fitnessCenterId })
                else { continue }

                activityPass.setFitnessCenter(fitnessCenter)
            }
        }

        if !eventReferences.isEmpty {

            let eventCollection = try await fetchCollectionFromFirestore(
                path: "events",
                queryItems: [
                    FirestoreQueryItem(FieldPath.documentID(), .in, values: eventReferences)
                ]
            ) as [Event]

            for activityPass in activityPassCollection {

                guard let eventId = activityPass.getEventReference()?.documentID,
                      let event = eventCollection.first(where: { $0.getId() == eventId })
                else { continue }

                activityPass.setEvent(event)
            }
        }

        return activityPassCollection
    }

    static public func getActivityPass(fitnessCenterId: String? = nil, eventId: String? = nil) async throws ->
ActivityPass? {

        let queryItem: FirestoreQueryItem? = {
```

```
            if let fitnessCenterId = fitnessCenterId {

                let reference = Firestore
                    .firestore()
                    .collection("fitness_centers")
                    .document(fitnessCenterId)

                return FirestoreQueryItem("fitness_center_reference", .isEqualTo, value: reference)
            } else if let eventId = eventId {

                let reference = Firestore
                    .firestore()
                    .collection("events")
                    .document(eventId)

                return FirestoreQueryItem("event_reference", .isEqualTo, value: reference)
            } else {

                return nil
            }
        }()

        guard let queryItem = queryItem,
              let userUid = Auth.auth().currentUser?.uid
        else { throw FirestoreError.malformedRequest }

        let userReference = Firestore
            .firestore()
            .collection("users")
            .document(userUid)

        let collection = try await fetchCollectionFromFirestore(
            path: "passes",
            queryItems: [
                queryItem,
                FirestoreQueryItem("user_reference", .isEqualTo, value: userReference)
            ]
        ) as [ActivityPass]

        if let pass = collection.first {

            if let fitnessCenterReference = pass.getFitnessCenterReference() {

                let fitnessCenter = try await fitnessCenterReference.getDocument().data(as: FitnessCenter.self)
                fitnessCenter.setDocumentId(fitnessCenterReference.documentID)

                pass.setFitnessCenter(fitnessCenter)
            } else if let eventReference = pass.getEventReference() {

                let event = try await eventReference.getDocument().data(as: Event.self)
                event.setDocumentId(eventReference.documentID)

                pass.setEvent(event)
            }
```

```
            return pass
        } else {

            return nil
        }
    }

    // MARK: - Stripe

    static public func createStripePaymentIntent(userUid: String, fitnessCenterId: String?, eventId: String?) async
throws -> StripePaymentIntent {

        guard var urlComponents = URLComponents(string: AppUrl.stripeCreatePaymentIntent)
        else { throw ApplePayError.malformedRequest }

        urlComponents.queryItems = [
            URLQueryItem(name: "user_uid", value: userUid)
        ]

        if let fitnessCenterId = fitnessCenterId {

            let queryItem = URLQueryItem(name: "fitness_center_id", value: fitnessCenterId)

            urlComponents.queryItems?.append(queryItem)
        }

        if let eventId = eventId {

            let queryItem = URLQueryItem(name: "event_id", value: eventId)

            urlComponents.queryItems?.append(queryItem)
        }

        guard let url = urlComponents.url
        else { throw ApplePayError.malformedRequest }

        let (data, _) = try await URLSession.shared.data(from: url)
        return try JSONDecoder().decode(StripePaymentIntent.self, from: data)
    }

    // MARK: - Methods

    static private func fetchCollectionFromFirestore<T: FirestoreObject & Decodable>(path: String, queryLimit:
FirestoreQueryLimit? = nil, queryItems: [FirestoreQueryItem] = []) async throws -> [T] {

        var query: Query = Firestore
            .firestore()
            .collection(path)

        if let queryLimit = queryLimit {

            query = query
                .limit(to: queryLimit.limit)
                .order(
                    by: queryLimit.orderBy,
```

```
                descending: queryLimit.descending
            )
        }

        for queryItem in queryItems {

            guard let value = queryItem.values.first
            else { continue }

            switch queryItem.type {
            case .isEqualTo:

                if let key = queryItem.key {

                    query = query.whereField(key, isEqualTo: value)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, isEqualTo: value)
                }
            case .isNotEqualTo:

                if let key = queryItem.key {

                    query = query.whereField(key, isNotEqualTo: value)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, isNotEqualTo: value)
                }
            case .isLessThan:

                if let key = queryItem.key {

                    query = query.whereField(key, isLessThan: value)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, isLessThan: value)
                }
            case .isLessThanOrEqualTo:

                if let key = queryItem.key {

                    query = query.whereField(key, isLessThanOrEqualTo: value)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, isLessThanOrEqualTo: value)
                }
            case .isGreaterThan:

                if let key = queryItem.key {

                    query = query.whereField(key, isGreaterThan: value)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, isGreaterThan: value)
                }
```

```
            case .isGreaterThanOrEqualTo:

                if let key = queryItem.key {

                    query = query.whereField(key, isGreaterThanOrEqualTo: value)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, isGreaterThanOrEqualTo: value)
                }
            case .arrayContains:

                if let key = queryItem.key {

                    query = query.whereField(key, arrayContains: queryItem.values)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, arrayContains: queryItem.values)
                }
            case .arrayContainsAny:

                if let key = queryItem.key {

                    query = query.whereField(key, arrayContainsAny: queryItem.values)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, arrayContainsAny: queryItem.values)
                }
            case .in:

                if let key = queryItem.key {

                    query = query.whereField(key, in: queryItem.values)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, in: queryItem.values)
                }
            case .notIn:

                if let key = queryItem.key {

                    query = query.whereField(key, notIn: queryItem.values)
                } else if let fieldPath = queryItem.fieldPath {

                    query = query.whereField(fieldPath, notIn: queryItem.values)
                }
            }
        }

        return try await query.getDocuments().documents.compactMap { document in

            guard let data = try? document.data(as: T.self)
            else { return nil }

            data.setDocumentId(document.documentID)
```

```
            return data
        }
    }
}
```

```swift
//
//  MainTabCoordinator.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/19/22.
//

import SwiftUI

final class MainTabCoordinator: ObservableObject {

    @Published var isShowingSignInView: Bool = false

    @Published var isShowingSearchViewWithConfiguration: SearchViewConfiguration?

    @Published var isShowindNewsDetailsViewWithConfiguration: NewsDetailsViewConfiguration?

    @Published var isShowingActivityDetailViewWithConfiguration: ActivityDetailViewConfiguration?
}
```

```swift
//
//  MainTabItem.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/8/22.
//

import SwiftUI

enum MainTabItem: Int, CaseIterable {

    // MARK: Cases

    case home = 0
    case myPasses

    // MARK: - Interaction Methods

    func getButtonImage() -> Image {
        switch self {
        case .home:

            return Image(systemName: "globe.europe.africa.fill")
        case .myPasses:

            return Image(systemName: "square.stack.fill")
        }
    }

    @ViewBuilder
    func getView() -> some View {

        switch self {
        case .home:

            HomeView()
        case .myPasses:

            let configuration = SearchViewConfiguration(
                scope: [.userFitnessCenters, .userEvents],
                isShowingSearchBar: false,
                isLocationEnabled: false,
                isShowingTabBarPadding: true
            )

            SearchView(viewModel: SearchView.ViewModel(configuration: configuration))
        }
    }
}
```

```swift
//
//  MainTabView.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/8/22.
//

import SwiftUI
import FirebaseService

struct MainTabView: View {

    // MARK: - Variables

    @State private var selectedTab = MainTabItem.home

    @ObservedObject private var mainTabCoordinator = MainTabCoordinator()

    // MARK: - Body

    var body: some View {

        ZStack(alignment: .bottom) {

            TabViewWrapper(
                currentTab: $selectedTab,
                controllers: MainTabItem.allCases.map {
                    UIHostingController(
                        rootView: AnyView(
                            $0.getView()
                                .environmentObject(mainTabCoordinator)
                        )
                    )
                }
            )
            .ignoresSafeArea(.all, edges: .bottom)
            .background(Color.Interface.white)

            VStack {

                Spacer()

                MainTabBar(selectedTab: $selectedTab)
            }
            .padding(.bottom, 8)
            .ignoresSafeArea(.keyboard)
        }
        .sheet(
            isPresented: $mainTabCoordinator.isShowingSignInView,
            onDismiss: { mainTabCoordinator.isShowingSignInView = false },
            content: {

                SignInView()
```

```
                SignInView()
            }
        )
        .sheet(
            item: $mainTabCoordinator.isShowingSearchViewWithConfiguration,
            onDismiss: { mainTabCoordinator.isShowingSearchViewWithConfiguration = nil },
            content: { configuration in

                SearchView(viewModel: SearchView.ViewModel(configuration: configuration))
            }
        )
        .sheet(
            item: $mainTabCoordinator.isShowindNewsDetailsViewWithConfiguration,
            onDismiss: { mainTabCoordinator.isShowindNewsDetailsViewWithConfiguration = nil },
            content: { item in

                NewsDetailsView(news: item.news)
            }
        )
        .sheet(
            item: $mainTabCoordinator.isShowingActivityDetailViewWithConfiguration,
            onDismiss: { mainTabCoordinator.isShowingActivityDetailViewWithConfiguration = nil },
            content: { _ in

                ActivityDetailView(
                    viewModel: ActivityDetailView.ViewModel(
                        mainTabCoordinator
                            .isShowingActivityDetailViewWithConfiguration?
                            .activity
                    )
                )
            }
        )
    }
}

struct MainTabBar_Previews: PreviewProvider {
    static var previews: some View {
        MainTabView()
    }
}
```

```
//
//  MainTabBarItem.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/8/22.
//

import SwiftUI

struct MainTabBarItem: View {

    // MARK: - Variables

    @Binding var selectedTab: MainTabItem

    var tabItem: MainTabItem

    // MARK: - Body

    var body: some View {

        Button {

            selectedTab = tabItem
        } label: {

            let tintColor = selectedTab == tabItem ? Color.Interface.black : Color.Interface.platinum

            tabItem.getButtonImage()
                .resizable()
                .scaledToFit()
                .frame(width: 30, height: 30)
                .tint(tintColor)
        }
        .padding(10)
    }
}
```

```
//
//  MainTabBar.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/8/22.
//

import SwiftUI

struct MainTabBar: View {

    // MARK: - Variables

    @Binding var selectedTab: MainTabItem

    // MARK: - Body

    var body: some View {

        HStack(spacing: 10) {

            ForEach(MainTabItem.allCases, id: \.rawValue) { tabItem in

                MainTabBarItem(
                    selectedTab: $selectedTab,
                    tabItem: tabItem
                )
            }
        }
        .padding([.leading, .trailing], 15)
        .background(Color.Interface.white)
        .cornerRadius(.infinity)
        .shadow(color: Color.Interface.black.opacity(0.3), radius: 5)
    }
}
```

```
//
//  SignInView.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import AuthenticationServices
import FirebaseService
import FirebaseAuth

struct SignInView: View {

    // MARK: - Variables

    @Environment(\.dismiss) var dismiss

    @State private var isAlertShown: Bool = false
    @State private var alertMessage: String?

    // MARK: - Body

    var body: some View {

        ZStack {

            Color.Interface.black
                .ignoresSafeArea()

            VStack(spacing: 30) {

                HStack {

                    Spacer()

                    Button {

                        dismiss()
                    } label: {

                        Image(systemName: "xmark.circle.fill")
                            .resizable()
                            .foregroundColor(Color.Interface.white)
                            .frame(width: 30, height: 30)
                    }
                    .padding(10)
                }


                Spacer()

                Image("logo.medium")
                    .resizable()
                    .frame(width: 100, height: 100)

                Spacer()
```

```
                Spacer()

                Text("Sign in to unlock all features")
                    .font(.system(size: 20, weight: .bold, design: .rounded))
                    .foregroundColor(Color.Interface.white)
                    .multilineTextAlignment(.center)
                    .frame(maxWidth: .infinity)

                FirebaseSignInWithAppleButton(
                    label: .continue,
                    requestedScopes: [.fullName, .email]
                ) { result in

                    switch result {
                    case .success:

                        dismiss()
                    case .failure(let error):

                        guard error._domain == AuthErrorDomain
                        else { return }

                        isAlertShown = true
                        alertMessage = error.localizedDescription
                    }
                }
                .signInWithAppleButtonStyle(.white)
                .clipShape(Capsule())
                .frame(height: 50)
                .frame(maxWidth: 400)
            }
            .padding(20)
        }
        .alert("Something went wrong", isPresented: $isAlertShown, actions: {

        }, message: {
            Text(alertMessage ?? "Try again, please")
                .onAppear { alertMessage = nil }
        })
    }
}

struct SignInView_Previews: PreviewProvider {
    static var previews: some View {
        SignInView()
            .environmentObject(AuthState())
    }
}
```

```
//
//  HomeView.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/8/22.
//

import SwiftUI
import SwiftUIPullToRefresh

struct HomeView: View {

    // MARK: - Variables

    @StateObject var viewModel = ViewModel()

    // MARK: - Body

    var body: some View {

        VStack() {

            HomeNavBar()
                .padding(.horizontal, 20)
                .padding(.top, 10)

            Divider()
                .background(Color.Interface.platinum)
                .padding(.horizontal, 20)

            RefreshableScrollView(showsIndicators: false) {

                await viewModel.refreshNews()
            } progress: { state in

                RefreshActivityIndicator(isAnimating: [.primed, .loading, .waiting].contains(state))
            } content: {

                LazyVStack(alignment: .leading, spacing: 25) {

                    if !viewModel.news.isEmpty {

                        HomeNewsSection(news: viewModel.news)
                    }

                    HomeExploreSection()

                    if !viewModel.events.isEmpty {

                        HomeUpcomingEventsSection(events: viewModel.events)
                    }

                    if !viewModel.fitnessCenters.isEmpty {

                        HomeFitnessCentersSection(fitnessCenters: viewModel.fitnessCenters)
                    }
                }
                .animation(
```

```
                    .easeInOut,
                    value: viewModel.news.count +
                        viewModel.events.count +
                        viewModel.fitnessCenters.count
                )
                .padding(.top, 25)
                .padding(.bottom, 80)
            }
        }
        .background(Color.Interface.white)
        .onTapGesture(perform: hideKeyboard)
    }
}

struct HomeView_Previews: PreviewProvider {
    static var previews: some View {
        HomeView()
    }
}
```

```
//
//  HomeViewModel.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/22/22.
//

import Foundation

extension HomeView {

    @MainActor
    final class ViewModel: ObservableObject {

        // MARK: - Variables

        @Published var news = [News]()

        @Published var events = [Event]()

        @Published var fitnessCenters = [FitnessCenter]()

        // MARK: - Init

        init() {

            Task(priority: .background) {

                await refreshNews()
            }
        }

        // MARK: - Interaction Methods

        public func refreshNews() async {

            async let newsRequest = NetworkService.getNews()
            async let eventsRequest = NetworkService.getEvents()
            async let fitnessCentersRequest = NetworkService.getFitnessCenters()

            let (retrievedNews, retrievedEvents, retrievedFitnessCenters) = try! await (newsRequest, eventsRequest,
fitnessCentersRequest)

            news = retrievedNews

            events = retrievedEvents

            fitnessCenters = retrievedFitnessCenters
        }
    }
}
```

```
//
//  HomeNavBar.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI

struct HomeNavBar: View {

    // MARK: - Variables

    @EnvironmentObject private var mainTabCoordinator: MainTabCoordinator

    @StateObject var accountService = AccountService.shared

    // MARK: - Body

    var body: some View {

        HStack(alignment: .center, spacing: 20) {
```

```
            Text("Gean Sports")
                .font(.system(size: 20, weight: .bold, design: .rounded))
                .tint(Color.Interface.black)
                .lineLimit(1)
                .frame(height: 40)

            Spacer()

            if accountService.authStatus == .authenticated {

                Menu {

                    Button("Sign out") {

                        try? AccountService.shared.signOut()
                    }
                } label: {

                    Image(systemName: "person.crop.circle.fill")
                        .resizable()
                        .scaledToFit()
                        .frame(width: 35, height: 35)
                        .tint(Color.Interface.black)
                }
            } else {

                Button {

                    mainTabCoordinator.isShowingSignInView = true
                } label: {

                    Image(systemName: "person.crop.circle.fill.badge.plus")
                        .resizable()
                        .scaledToFit()
                        .frame(width: 40, height: 40)
                        .tint(Color.Interface.black)
                }
            }
        }
    }
}
```

```
//
//  HomeNewsSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import ASCollectionView

struct HomeNewsSection: View {

    // MARK: - Variables

    @EnvironmentObject private var mainTabCoordinator: MainTabCoordinator

    var news: [News]

    // MARK: - Body

    var body: some View {

        ASCollectionView(
            data: news,
            dataID: \.id
        ) { item, _ in

            HomeNewsCarouselCell(news: item)
                .onTapGesture {

                    mainTabCoordinator.isShowindNewsDetailsViewWithConfiguration = NewsDetailsViewConfiguration(news:
item)
                }
        }
        .layout(layout: getCollectionLayout)
        .frame(height: 300)
    }

    // MARK: - Methods

    func getCollectionLayout() -> ASCollectionLayoutSection {

        return ASCollectionLayoutSection {

            let item = NSCollectionLayoutItem(
                layoutSize: NSCollectionLayoutSize(
                    widthDimension: .fractionalWidth(1.0),
                    heightDimension: .fractionalHeight(1.0)
                )
            )

            let group = NSCollectionLayoutGroup.vertical(
                layoutSize: NSCollectionLayoutSize(
                    widthDimension: .absolute(UIScreen.main.bounds.width - 40),
                    heightDimension: .fractionalHeight(1)
                ),
                subitems: [item]
            )

            let section = NSCollectionLayoutSection(group: group)
            section.orthogonalScrollingBehavior = .groupPaging
            section.interGroupSpacing = 15
            section.contentInsets = NSDirectionalEdgeInsets(
                top: 0,
                leading: 20,
                bottom: 0,
                trailing: 20
            )

            return section
        }
    }
}
```

```
//
//  HomeNewsCarouselCell.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import NukeUI

struct HomeNewsCarouselCell: View {

    // MARK: - Variables

    var news: News

    // MARK: - Body

    var body: some View {

        VStack(alignment: .leading, spacing: 15) {

            VStack(alignment: .leading, spacing: 5) {

                Text(getNewsTitle())
                    .font(.system(size: 20, weight: .bold))
                    .foregroundColor(Color.Interface.black)
                    .lineLimit(1)

                Text(getUpdateAtText())
                    .font(.system(size: 16, weight: .bold))
                    .foregroundColor(Color.Interface.platinum)
                    .lineLimit(1)
            }

            LazyImage(
                source: news.getBackgroundImageUrl()?.absoluteString,
                resizingMode: .aspectFill
            )
            .cornerRadius(5)
        }
    }

    // MARK: - Interaction Methods

    private func getNewsTitle() -> String {

        return news.getTitle() ?? ""
    }

    private func getUpdateAtText() -> String {

        let date = news.getCreatedAt() ?? Date()

        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "dd MMM yyyy"
        let dateText = dateFormatter.string(from: date)

        return String(
            format: "Published on %@",
            dateText
        )
    }
}
```

```
//
//  HomeExploreSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import NukeUI

struct HomeExploreSection: View {

    // MARK: - Variables

    @EnvironmentObject private var mainTabCoordinator: MainTabCoordinator

    // MARK: - Body

    var body: some View {

        VStack(alignment: .leading, spacing: 15) {

            Text("Explore 🗺")
                .font(.system(size: 25, weight: .bold))
                .foregroundColor(Color.Interface.black)
                .lineLimit(1)

            Button {

                mainTabCoordinator.isShowingSearchViewWithConfiguration = SearchViewConfiguration(
                    scope: [.fitnessCenters, .events]
                )
            } label: {

                SearchTextFieldView(
                    shouldShowLocationButton: false,
                    scope: [.fitnessCenters, .events],
                    searchQuery: .constant("")
                )
                .allowsHitTesting(false)
            }
        }
        .padding(.horizontal, 20)
    }
}
```

```swift
//
//  HomeFitnessCentersSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI

struct HomeFitnessCentersSection: View {

    // MARK: - Variables

    @EnvironmentObject private var mainTabCoordinator: MainTabCoordinator

    var fitnessCenters: [FitnessCenter]

    // MARK: - Body

    var body: some View {

        VStack(alignment: .leading, spacing: 15) {

            Text("Our fitness centers 🌍")
                .font(.system(size: 25, weight: .bold))
                .foregroundColor(Color.Interface.black)
                .lineLimit(1)

            LazyVStack(spacing: 20) {

                ForEach(fitnessCenters) { fitnessCenter in

                    HomeFitnessCentersSectionCell(fitnessCenter: fitnessCenter)
                        .onTapGesture {

                            let configuration = ActivityDetailViewConfiguration(activity: fitnessCenter)

                            mainTabCoordinator.isShowingActivityDetailViewWithConfiguration = configuration
                        }
                }
            }
        }
        .padding(.horizontal, 20)
    }
}
```

```swift
//
//  HomeFitnessCentersSectionCell.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import NukeUI

struct HomeFitnessCentersSectionCell: View {

    // MARK: - Variables

    var fitnessCenter: FitnessCenter

    // MARK: - Body

    var body: some View {

        ZStack(alignment: .bottomLeading) {

            LazyImage(
                source: fitnessCenter.getBackgroundImageUrl(),
                resizingMode: .aspectFill
            )

            Color.Interface.black.opacity(0.5)


            HStack(spacing: 10) {

                VStack(alignment: .leading, spacing: 10) {

                    if let name = fitnessCenter.getName() {

                        Text(name)
                            .foregroundColor(Color.Interface.white)
                            .font(.system(size: 20, weight: .bold))
                    }

                    Divider()
                        .frame(height: 3)
                        .frame(maxWidth: 60)
                        .background(Color.Interface.white)

                    if let addressDescription = fitnessCenter.getAddress()?.getDescription() {

                        Text(addressDescription)
                            .foregroundColor(Color.Interface.platinum)
                            .font(.system(size: 16, weight: .semibold))
                    }
                }
                .lineLimit(1)

                Spacer()

                Image(systemName: "arrow.right.circle.fill")
```

```
                    .resizable()
                    .foregroundColor(.white)
                    .frame(width: 30, height: 30)
            }
            .padding(.vertical, 15)
            .padding(.horizontal, 10)
        }
        .cornerRadius(5)
        .frame(height: 130)
    }
}
```

```
//
//  HomeUpcomingEventsSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI

struct HomeUpcomingEventsSection: View {

    // MARK: - Variables

    @EnvironmentObject private var mainTabCoordinator: MainTabCoordinator

    var events: [Event]

    // MARK: - Body

    var body: some View {

        VStack(alignment: .leading, spacing: 15) {

            Text("Upcoming events 😍")
                .font(.system(size: 25, weight: .bold))
                .foregroundColor(Color.Interface.black)
                .lineLimit(1)
                .padding(.horizontal, 20)

            ScrollView(.horizontal, showsIndicators: false) {

                LazyHStack(spacing: 20) {

                    ForEach(events) { event in

                        HomeUpcomingEventsSectionCell(event: event)
                            .onTapGesture {

                                let configuration = ActivityDetailViewConfiguration(activity: event)

                                mainTabCoordinator.isShowingActivityDetailViewWithConfiguration = configuration
                            }
                    }
                }
                .padding(.horizontal, 20)
            }
        }
        .frame(maxWidth: .infinity, alignment: .leading)
        .padding(.vertical, 20)
        .background(Color.Interface.cultured)
    }
}
```

```
//
//  HomeUpcomingEventsCellSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import NukeUI

struct HomeUpcomingEventsSectionCell: View {

    // MARK: - Variables

    var event: Event

    // MARK: - Body

    var body: some View {

        ZStack(alignment: .bottomLeading) {

            LazyImage(
                source: event.getBackgroundImageUrl(),
                resizingMode: .aspectFill
            )

            Color.black.opacity(0.5)

            VStack(alignment: .leading, spacing: 10) {

                if let name = event.getName() {

                    Text(name)
                        .foregroundColor(Color.Interface.white)
                        .font(.system(size: 20, weight: .bold))
                }

                Divider()
                    .frame(height: 3)
                    .frame(maxWidth: 60)
                    .background(Color.Interface.white)

                if let localizedDate = event.getLocalizedDate() {

                    Text(localizedDate)
                        .foregroundColor(Color.Interface.platinum)
                        .font(.system(size: 16, weight: .bold))
                }
            }
            .padding(.vertical, 15)
            .padding(.horizontal, 10)
        }
        .frame(width: 140, height: 215)
        .cornerRadius(5)
    }
}
```

```swift
//
//  SearchScope.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/19/22.
//

import Foundation

enum SearchScope: Identifiable {

    // MARK: - Cases

    case events
    case fitnessCenters

    case userEvents
    case userFitnessCenters

    // MARK: - Identifiable

    var id: Int { hashValue }

    // MARK: - Interaction Methods

    func getTextFieldPlaceholder() -> String {

        switch self {
        case .events, .userEvents:

            return "events"
        case .fitnessCenters, .userFitnessCenters:

            return "fitness centers"
        }
    }
}
```

```swift
//
//  SearchViewConfiguration.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/19/22.
//

import Foundation

struct SearchViewConfiguration: Identifiable {

    var id: Int { 0 }

    // MARK: - Variables

    var scope = [SearchScope]()

    var isShowingSearchBar = true

    var isLocationEnabled = true

    var isShowingTabBarPadding = false
}
```

```swift
//
//  SearchView.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import SwiftUIPullToRefresh

struct SearchView: View {

    // MARK: - Configuration Variables

    private var configuration: SearchViewConfiguration {

        viewModel.configuration
    }

    // MARK: - Variables

    @StateObject var viewModel: SearchView.ViewModel

    @Environment(\.presentationMode) var presentationMode

    @EnvironmentObject private var mainTabCoordinator: MainTabCoordinator


    @State var isShowingActivityDetailViewWithConfiguration: ActivityDetailViewConfiguration?

    // MARK: - Body

    var body: some View {

        VStack {

            if configuration.isShowingSearchBar {

                SearchTextFieldView(
                    shouldShowLocationButton: configuration.isLocationEnabled,
                    scope: configuration.scope,
                    searchQuery: $viewModel.searchQuery
                )
                .padding(.top, 20)
                .padding([.horizontal], 20)
            }

            RefreshableScrollView(showsIndicators: false) {

                await viewModel.refreshActivityPasses()
            } progress: { state in
```

```
                RefreshActivityIndicator(isAnimating: true)
            } content: {

                LazyVStack(
                    alignment: .leading,
                    spacing: 0
                ) {

                    if !viewModel.fitnessCenterList.isEmpty {

                        Text("Fitness centers")
                            .font(.system(size: 25, weight: .bold))
                            .padding(.top, 20)
                            .padding(.bottom, 5)

                        ForEach(viewModel.fitnessCenterList) { fitnessCenter in

                            SearchItemCell(activity: fitnessCenter)
                                .onTapGesture {

                                    if presentationMode.wrappedValue.isPresented {

                                        isShowingActivityDetailViewWithConfiguration =
ActivityDetailViewConfiguration(activity: fitnessCenter)
                                    } else {

                                        mainTabCoordinator.isShowingActivityDetailViewWithConfiguration =
ActivityDetailViewConfiguration(activity: fitnessCenter)
                                    }
                                }
                        }
                    }

                    if !viewModel.eventList.isEmpty {

                        Text("Events")
                            .font(.system(size: 25, weight: .bold))
                            .padding(.top, 20)
                            .padding(.bottom, 5)

                        ForEach(viewModel.eventList) { event in

                            SearchItemCell(activity: event)
                                .onTapGesture {

                                    if presentationMode.wrappedValue.isPresented {

                                        isShowingActivityDetailViewWithConfiguration =
ActivityDetailViewConfiguration(activity: event)
                                    } else {
```

```
                                        mainTabCoordinator.isShowingActivityDetailViewWithConfiguration =
ActivityDetailViewConfiguration(activity: event)
                                    }
                                }
                        }
                    }
                }
                .animation(.easeInOut(duration: 0.15), value: viewModel.fitnessCenterList.count)
                .animation(.easeInOut(duration: 0.15), value: viewModel.eventList.count)
                .padding(.horizontal, 20)
                .padding(.bottom, configuration.isShowingTabBarPadding ? 80 : 0)
            }
        }
        .background(Color.Interface.white)
        .sheet(
            item: $isShowingActivityDetailViewWithConfiguration,
            onDismiss: { isShowingActivityDetailViewWithConfiguration = nil },
            content: { configuration in

                ActivityDetailView(
                    viewModel: ActivityDetailView.ViewModel(
                        configuration.activity
                    )
                )
            }
        )
    }
}
```

```
//
//  SearchViewModel.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 6/4/22.
//

import Foundation
import Combine

extension SearchView {

    final class ViewModel: ObservableObject {

        // MARK: - Variables

        var configuration: SearchViewConfiguration

        @Published var fitnessCenterList = [FitnessCenter]()

        @Published var eventList = [Event]()

        @Published var searchQuery: String = ""

        private var cancellables = Set<AnyCancellable>()

        init(configuration: SearchViewConfiguration) {

            self.configuration = configuration

            bindSearchQuery()

            bindActivityPassList()
        }

        // MARK: - Interaction Methods

        public func refreshActivityPasses() async {

            try? await AccountService.shared.refreshActivityPasses()

            await refreshBindingActivityList()
        }

        public func fetchActivityList(with searchQuery: String) {

            Task(priority: .userInitiated) {

                async let getFitnessCenters = try? NetworkService.getFitnessCenters(
                    searchQuery: searchQuery,
                    limit: 3
                )
```

```
                let (fitnessCenterList, eventList) = await (getFitnessCenters, getEvents)

                await MainActor.run {

                    self.fitnessCenterList = fitnessCenterList ?? []

                    self.eventList = eventList ?? []
                }
            }
        }

        // MARK: - Methods

        @MainActor
        private func refreshBindingActivityList() async {

            if configuration.scope.contains(.userFitnessCenters) {

                fitnessCenterList = AccountService.shared.activityPassList?
                    .compactMap { $0.getFitnessCenter() }
                    .sorted {

                        guard let lhsName = $0.getName(),
                              let rhsName = $1.getName()
                        else { return false }

                        return lhsName < rhsName
                    } ?? []
            }

            if configuration.scope.contains(.userEvents) {

                eventList = AccountService.shared.activityPassList?
                    .compactMap { $0.getEvent() }
                    .sorted {

                        guard let lhsDate = $0.getDate(),
                              let rhsDate = $1.getDate()
                        else { return false }

                        return lhsDate < rhsDate
                    } ?? []
            }
        }

        private func bindSearchQuery() {

            $searchQuery
                .debounce(for: 0.3, scheduler: RunLoop.main)
                .sink { [weak self] searchQuery in
```

```
                    guard let self = self
                    else { return }

                    if searchQuery.isEmpty {

                        if self.configuration.scope.contains(.fitnessCenters) {

                            self.fitnessCenterList = []
                        }

                        if self.configuration.scope.contains(.events) {

                            self.eventList = []
                        }
                    } else {

                        self.fetchActivityList(with: searchQuery)
                    }
                }
                .store(in: &cancellables)
        }

        private func bindActivityPassList() {

            AccountService.shared.$activityPassList.sink { [weak self] _ in

                Task { [weak self] in

                    guard let self = self
                    else { return }

                    await self.refreshBindingActivityList()
                }
            }.store(in: &cancellables)
        }
    }
}
```

```
//
//  SearchTextFieldView.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import CoreLocationUI

struct SearchTextFieldView: View {

    // MARK: - Configuration Variables

    var shouldShowLocationButton: Bool = true

    var scope = [SearchScope]()

    // MARK: - Variables

    @Binding var searchQuery: String

    @FocusState private var isTextFieldFocused: Bool

    // MARK: - Body

    var body: some View {

        HStack(spacing: 10) {

            HStack(spacing: 10) {

                Image(systemName: "magnifyingglass")
                    .resizable()
                    .frame(width: 20, height: 20)
                    .foregroundColor(isTextFieldFocused || !searchQuery.isEmpty ? Color.Interface.black :
Color.Interface.platinum)
                    .animation(.easeIn(duration: 0.1), value: isTextFieldFocused)

                ZStack(alignment: .leading) {

                    let placeholderText = String(
                        format: "Search for %@",
                        scope.map { $0.getTextFieldPlaceholder() }
                            .uniqued()
                            .joined(separator: ", ")
                    )

                    Text(placeholderText)
                        .font(.system(size: 14, weight: .bold))
                        .lineLimit(1)
                        .opacity(searchQuery.isEmpty ? 1 : 0)
                        .foregroundColor(Color.Interface.platinum)
                        .animation(.easeInOut(duration: 0.1), value: searchQuery)
```

```
                    TextField("", text: $searchQuery)
                        .tint(Color.Interface.black)
                        .accentColor(Color.Interface.black)
                        .font(.system(size: 14, weight: .bold))
                        .focused($isTextFieldFocused)
                        .disableAutocorrection(true)
                        .textInputAutocapitalization(.words)
                        .frame(height: 50)
                }
            }
            .padding(.leading, 15)
            .padding(.trailing, 10)
            .background(Color.Interface.cultured)
            .onTapGesture {

                isTextFieldFocused = true
            }

            /*if shouldShowLocationButton {

                LocationButton {
                    print(9999)
                }
                .symbolVariant(.fill)
                .labelStyle(.iconOnly)
                .foregroundColor(.white)
                .clipShape(Capsule())
            }*/
        }
    }
}
```

```
//
//  SearchItemCell.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import NukeUI

struct SearchItemCell: View {

    // MARK: - Variables

    var activity: Activity

    // MARK: - Body

    var body: some View {


        VStack(
            alignment: .leading,
            spacing: 10
        ) {

            HStack(
                alignment: .center,
                spacing: 15) {

                    LazyImage(
                        source: activity.getBackgroundImageUrl(),
                        resizingMode: .aspectFill
                    )
                    .frame(width: 70, height: 70)

                    VStack(
                        alignment: .leading,
                        spacing: 1
                    ) {

                        if let mainText = activity.getName() {

                            Text(mainText)
                                .foregroundColor(Color.Interface.black)
                                .font(.system(size: 20, weight: .bold))
                                .lineLimit(1)
                        }

                        Color.Interface.black
                            .frame(width: 60, height: 3)

                        if let secondaryText = activity.getLocalizedDate() ?? activity.getAddress()?.getDescription()
                        {
```

```
                            Text(secondaryText)
                                .foregroundColor(Color.Interface.platinum)
                                .font(.system(size: 16, weight: .medium))
                                .padding(.top, 10)
                                .lineLimit(1)
                        }
                    }

                    Spacer()

                    Image(systemName: "arrow.right.circle.fill")
                        .resizable()
                        .frame(width: 30, height: 30)
                }

            Divider()
                .background(Color.Interface.platinum)
        }
        .padding(.top, 10)
    }
}
```

```
//
//  NewsDetailsViewConfiguration.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/22/22.
//

import Foundation

struct NewsDetailsViewConfiguration: Identifiable {

    // MARK: - Variables

    var id: Int?

    var news: News
}
```

```
//
//  NewsDetailsView.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/22/22.
//

import SwiftUI
import NukeUI

struct NewsDetailsView: View {

    // MARK: - Variables

    var news: News

    // MARK: - Body

    var body: some View {

        ScrollView(
            .vertical,
            showsIndicators: false
        ) {

            VStack(alignment: .leading, spacing: 15) {

                if let backgroundImageUrl = news.getBackgroundImageUrl()?.absoluteString {

                    LazyImage(
                        source: backgroundImageUrl,
                        resizingMode: .aspectFill
                    )
                    .frame(height: 250)
                    .cornerRadius(5)
                }

                VStack(alignment: .leading, spacing: 5) {

                    Text(getNewsTitle())
                        .font(.system(size: 20, weight: .bold))
                        .foregroundColor(Color.Interface.black)

                    Text(getUpdateAtText())
                        .font(.system(size: 16, weight: .bold))
                        .foregroundColor(Color.Interface.platinum)
                }

                if let description = news.getDescription() {

                    Text(description)
                        .font(.system(size: 16, weight: .medium))
                        .foregroundColor(Color.Interface.black)
                }
            }
        }
        .padding(20)
    }
```

```
    // MARK: - Interaction Methods

    private func getNewsTitle() -> String {

        return news.getTitle() ?? ""
    }

    private func getUpdateAtText() -> String {

        let date = news.getCreatedAt() ?? Date()

        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "dd MMM yyyy"
        let dateText = dateFormatter.string(from: date)

        return String(
            format: "Published on %@",
            dateText
        )
    }
}
```

```swift
//
//  ActivityDetailViewConfiguration.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/19/22.
//

import Foundation

struct ActivityDetailViewConfiguration: Identifiable {

    // MARK: - Variables

    var id: Int { 0 }

    var activity: Activity
}
```

```swift
//
//  DetailView.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/15/22.
//

import SwiftUI
import NukeUI
import MapKit
import CoreImage.CIFilterBuiltins
import StripeUICore

struct ActivityDetailView: View {

    // MARK: - Variables

    @Environment(\.presentationMode) var presentationMode

    @StateObject var viewModel: ViewModel

    // MARK: - Body

    var body: some View {

        ScrollView(
            .vertical,
            showsIndicators: false
        ) {

            VStack(
                alignment: .leading,
                spacing: 25
            ) {

                ActivityDetailHeader(viewModel: viewModel)
                    .frame(height: 200)

                Group {

                    if viewModel.activityPass == nil {

                        ActivityDetailGetAccessSection(viewModel: viewModel)
                    } else {

                        ActivityDetailQRCodeSection(viewModel: viewModel)
                    }

                    if let description = viewModel.activity?.getDescription() {

                        ActivityDetailDescriptionSection(decription: description)
                    }

                    if let address = viewModel.activity?.getAddress() {

                        ActivityDetailMapSection(address: address)
                    }
                }
```

```swift
                .padding([.horizontal], 20)
            }
            .animation(.easeInOut, value: (viewModel.activityPass == nil))
        }
        .edgesIgnoringSafeArea(.top)
    }
}
```

```swift
//
//  ActivityDetailView+ViewModel.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import SwiftUI
import Combine

extension ActivityDetailView {

    @MainActor
    final class ViewModel: ObservableObject {

        // MARK: - Variables

        @Published var activity: Activity?

        @Published var activityPass: ActivityPass?

        @Published var isFetchingActivityPass: Bool = false

        private var applePayService = ApplePayService()

        private var cancellables = Set<AnyCancellable>()

        // MARK: - Init

        init(_ activity: Activity?) {

            self.activity = activity

            self.activityPass = getLocalActivityPass()

            bindActivityPassList()
        }

        // MARK: - Interaction Methods

        public func performPayment() {

            Task(priority: .userInitiated) {

                do {

                    try await applePayService.pay(for: activity)

                    await monitorActivityPass()
                } catch let error {

                    print(error.localizedDescription)
                }
            }
        }
```

```swift
        private func bindActivityPassList() {

            AccountService.shared.$activityPassList.sink { [weak self] activityPassList in

                guard let self = self
                else { return }

                self.activityPass = self.getLocalActivityPass(from: activityPassList)
            }.store(in: &cancellables)
        }

        private func monitorActivityPass() async {

            let activityPass = try? await NetworkService.getActivityPass(
                fitnessCenterId: (activity as? FitnessCenter)?.getId(),
                eventId: (activity as? Event)?.getId()
            )

            if let activityPass = activityPass {

                AccountService.shared.appendActivityPass(activityPass)

                await MainActor.run {

                    self.activityPass = activityPass
                }
            } else {

                try? await Task.sleep(nanoseconds: 1_000_000_000)

                await monitorActivityPass()
            }
        }

        private func getLocalActivityPass(from activityPassList: [ActivityPass]? = nil) -> ActivityPass? {

            guard let activity = activity
            else { return nil }

            return (activityPassList ?? AccountService.shared.activityPassList)?.first { activityPass in

                let fitnessCenterId = activityPass.getFitnessCenter()?.getId()
                let eventId = activityPass.getEvent()?.getId()

                if let thisFitnessCenterId = (activity as? FitnessCenter)?.getId() {

                    return thisFitnessCenterId == fitnessCenterId
                } else if let thisEventId = (activity as? Event)?.getId() {

                    return thisEventId == eventId
                } else {

                    return false
                }
            }
        }
    }
}
```

```swift
//
//  ActivityDetailHeader.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import SwiftUI
import NukeUI

struct ActivityDetailHeader: View {

    // MARK: - Variables

    @ObservedObject var viewModel: ActivityDetailView.ViewModel

    // MARK: - Body

    var body: some View {

        ZStack(alignment: .bottomLeading) {

            LazyImage(
                source: viewModel.activity?.getBackgroundImageUrl(),
                resizingMode: .aspectFill
            )

            Color.Interface.black.opacity(0.5)

            VStack(
                alignment: .leading,
                spacing: 5
            ) {

                if let mainText = viewModel.activity?.getName() {

                    Text(mainText)
                        .foregroundColor(Color.Interface.white)
                        .font(.system(size: 20, weight: .bold))
                        .lineLimit(1)
                }

                Color.Interface.white
                    .frame(width: 60, height: 3)

                if
                    let activity = viewModel.activity,
                    let secondaryText = activity.getLocalizedDate() ?? activity.getAddress()?.getDescription()
                {

                    Text(secondaryText)
                        .foregroundColor(Color.Interface.platinum)
                        .font(.system(size: 16, weight: .medium))
                        .padding(.top, 5)
                        .lineLimit(1)
                }
            }
            .padding(20)
```

```swift
//
//  ActivityDetailGetAccessSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import SwiftUI

struct ActivityDetailGetAccessSection: View {

    // MARK: - Variables

    @ObservedObject var viewModel: ActivityDetailView.ViewModel

    @State private var isShowingSignInPage: Bool = false

    // MARK: - Body

    var body: some View {

        Button {

            if AccountService.shared.authStatus == .authenticated {

                viewModel.performPayment()
            } else {

                isShowingSignInPage = true
            }
        } label: {

            Text(getButtonTitle())
                .lineLimit(1)
                .font(.system(size: 16, weight: .bold))
                .truncationMode(.middle)
                .foregroundColor(Color.Interface.white)
                .padding(.vertical, 13)
                .padding(.horizontal, 20)
                .frame(maxWidth: .infinity, minHeight: 50)
                .background(Color.Interface.black)
                .foregroundColor(Color.Interface.white)
        }
        .sheet(isPresented: $isShowingSignInPage) {

            SignInView()
        }
    }

    // MARK: - Methods

    private func getButtonTitle() -> String {

        return [
            "Get access",
            viewModel.activity?.getLocalizedPrice()
        ].compactMap { $0 }.joined(separator: " • ")
    }
}
```

```swift
//
//  ActivityDetailQRCodeSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import SwiftUI

struct ActivityDetailQRCodeSection: View {

    // MARK: - Variables

    @ObservedObject var viewModel: ActivityDetailView.ViewModel

    @State private var qrCodeImage: UIImage?

    // MARK: - Body

    var body: some View {

        VStack(alignment: .leading, spacing: 15) {

            VStack(alignment: .leading, spacing: 5) {

                Text("Your QR code")
                    .font(.system(size: 25, weight: .bold))
                    .foregroundColor(Color.Interface.black)

                if let validUntilText = generateValidUntilText() {

                    Text(validUntilText)
                        .font(.system(size: 16, weight: .bold))
                        .foregroundColor(Color.Interface.platinum)
                        .lineLimit(1)
                }
            }

            VStack {

                Image(uiImage: qrCodeImage ?? UIImage())
                    .interpolation(.none)
                    .resizable()
                    .scaledToFit()
                    .frame(width: 250, height: 250)
                    .background(Color.Interface.cultured)
                    .task {

                        qrCodeImage = generateQRCodeImage(
                            from: viewModel.activityPass?.getQRCodeValue()
                        )
                    }
            }
            .frame(maxWidth: .infinity)
        }
    }
```

```swift
    // MARK: - Methods

    private func generateValidUntilText() -> String? {

        guard let endDate = viewModel.activityPass?.getEndDate()
        else { return nil }

        let dateFormatter = DateFormatter()
        dateFormatter.dateStyle = .medium

        return [
            "Valid until",
            dateFormatter.string(from: endDate)
        ].compactMap { $0 }.joined(separator: " ")
    }

    private func generateQRCodeImage(from string: String?) -> UIImage? {

        guard let string = string
        else { return nil }

        let filter = CIFilter.qrCodeGenerator()
        filter.message = Data(string.utf8)

        let context = CIContext()

        guard let outputImage = filter.outputImage,
              let cgImage = context.createCGImage(outputImage, from: outputImage.extent)
        else { return nil }

        return UIImage(cgImage: cgImage)
    }
}
```

```swift
//
//  ActivityDetailDescriptionSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import SwiftUI

struct ActivityDetailDescriptionSection: View {

    // MARK: - Variables

    var decription: String

    // MARK: - Body

    var body: some View {

        VStack(alignment: .leading, spacing: 15) {

            Text("Description")
                .font(.system(size: 25, weight: .bold))
                .foregroundColor(Color.Interface.black)

            Text(decription)
                .font(.system(size: 16, weight: .regular))
                .foregroundColor(Color.Interface.black)
        }
    }
}

struct ActivityDetailDescriptionSection_Previews: PreviewProvider {
    static var previews: some View {
        ActivityDetailDescriptionSection(decription: "")
    }
}
```

```swift
//
//  ActivityDetailMapSection.swift
//  Gean Client
//
//  Created by Vlad Gordiichuk on 5/21/22.
//

import SwiftUI
import CoreLocation
import MapKit
import UniformTypeIdentifiers

struct ActivityDetailMapSection: View {

    // MARK: - Variables

    var address: ActivityAddress?

    @State private var mapSnapshotImage: UIImage?

    @State private var isSeeOnMapActionsShown = false

    // MARK: - Body

    var body: some View {

        VStack(alignment: .leading, spacing: 15) {

            VStack(alignment: .leading, spacing: 10) {

                Text("Where to find")
                    .font(.system(size: 25, weight: .bold))
                    .foregroundColor(Color.Interface.black)

                if let addressDescription = address?.getDescription() {

                    Text(addressDescription)
                        .font(.system(size: 16, weight: .bold))
                        .foregroundColor(Color.Interface.platinum)
                        .lineLimit(1)
                }
            }

            Button {

                isSeeOnMapActionsShown = true
            } label: {

                ZStack {

                    Color.Interface.cultured

                    GeometryReader { proxy in

                            Image(uiImage: mapSnapshotImage ?? UIImage())
                                .animation(.none)
                                .task {
```

```
                                if let coordinate = address?.getCoordinate() {

                                    generateMapImage(
                                        coordinate: coordinate,
                                        imageSize: CGSize(
                                            width: proxy.size.width,
                                            height: proxy.size.height
                                        )
                                    )
                                }
                            }
                    }
                    .opacity(mapSnapshotImage == nil ? 0 : 1)
                    .animation(.easeInOut, value: mapSnapshotImage)
                }
            }
            .frame(height: 150)
            .confirmationDialog(
                "",
                isPresented: $isSeeOnMapActionsShown,
                titleVisibility: .hidden
            ) {

                let providers = MapService.Provider.allCases
                    .filter { MapService.canOpen(provider: $0) }

                ForEach(providers, id: \.hashValue) { provider in

                    if
                        let coordinate = address?.getCoordinate(),
                        MapService.canOpen(provider: provider)
                    {

                        let buttonTitle = String(
                            format: "See on %@",
                            provider.getTitle()
                        )

                        Button(buttonTitle) {

                            MapService.open(
                                provider: provider,
                                coordinate
                            )
                        }
                    }
                }

                if let addressDescription = address?.getDescription() {

                    Button("Copy address") {

                        UIPasteboard.general.setValue(
                            addressDescription,
                            forPasteboardType: UTType.plainText.identifier
                        )
                    }
                }
```

```
                }
            }
        }
    }

    private func generateMapImage(coordinate: CLLocationCoordinate2D, imageSize: CGSize) {

        let span = CLLocationDegrees(0.001)

        let region = MKCoordinateRegion(
            center: coordinate,
            span: MKCoordinateSpan(
                latitudeDelta: span,
                longitudeDelta: span
            )
        )

        let options = MKMapSnapshotter.Options()
        options.region = region
        options.size = imageSize
        options.showsBuildings = true

        let snapshotter = MKMapSnapshotter(options: options)
        snapshotter.start { (snapshot, _) in

            guard let snapshot = snapshot
            else { return }

            let mapImage = snapshot.image

            let annotationView = MKMarkerAnnotationView(annotation: nil, reuseIdentifier: nil)
            annotationView.contentMode = .scaleAspectFit
            annotationView.bounds = CGRect(
                x: 0,
                y: 0,
                width: 40,
                height: 40
            )

            UIGraphicsBeginImageContextWithOptions(mapImage.size, true, mapImage.scale)

            mapImage.draw(at: CGPoint.zero)

            annotationView.drawHierarchy(
                in: CGRect(
                    origin: CGPoint(
                        x: (imageSize.width / 2) - (annotationView.bounds.width / 2),
                        y: (imageSize.height / 2) - (annotationView.bounds.height / 2)
                    ),
                    size: annotationView.bounds.size
                ),
                afterScreenUpdates: true
            )

            let compositeImage = UIGraphicsGetImageFromCurrentImageContext()

            UIGraphicsEndImageContext()
```

```
                mapSnapshotImage = compositeImage
            }
        }
    }
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ ОТРИМАННЯ ПОСЛУГ У ФІТНЕС ЦЕНТРАХ

Керівництво користувача

КПІ.ІП-8109.045440.04.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Владислав ГОРДІЙЧУК

Київ – 2022

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

«Gean Sports» - це мобільний додаток для отримання послуг у фітнес-центрах. Завдяки цьому програмному застосунку, фітнес-центри мають можливість пропонувати альтернативний, цифровий рівень комунікації клієнта зі своїм бізнесом.

Користувацький інтерфейс було розроблено максимально доступно для клієнта та задовольняє потреби сучасного користувача додатку. Мобільний застосунок має сучасний вид та інтуїтивно зрозумілий підхід до інтеракції з елементами інтерфейсу.

2 ПІДГОТОВКА ДО РОБОТИ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність мобільного пристрою з операційною системою на базі iOS з версією 15.0 або вище;

– наявність доступу до Інтернету;

– для встановлення додатку на мобільному пристрої повинно бути не менше 50МБ вільної пам’яті.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, за допомогою xCode-IDE. Для початку – необхідно підключити iOS-девайс до комп’ютера, дочекатись налаштування режиму розробника та натиснути на кнопку «Run».

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку. У разі, якщо дана іконка не з’явилась, то встановлення відбулось не успішно.

3 РОБОТА ІЗ ЗАСТОСУНКОМ

3.1 Сторінка Explore

При запуску програмного застосунку, першим, що побачить користувач – сторінка Explore (рис. 5.1). Ця сторінка призначена для перегляду новин, фітнес-центрів та подій. Користувач має можливість оновити дані на сторінці Explore потягнувши зверху-вниз (рис. 5.2). При натисканні на елементи новини, фітнес-центру, або події – користувачу відкриється сторінка з деталями елементу, який він переглядає. Коли користувач натисне на іконку картинку чоловічка у верхньому правому куту – він побачить сторінку авторизації, або кнопку виходу із облікового запису, залежно від його статусу авторизації. Якщо користувач натисне на пошукове текстове поле – за екрані з’явиться сторінка пошуку фітнес-центрів та подій.

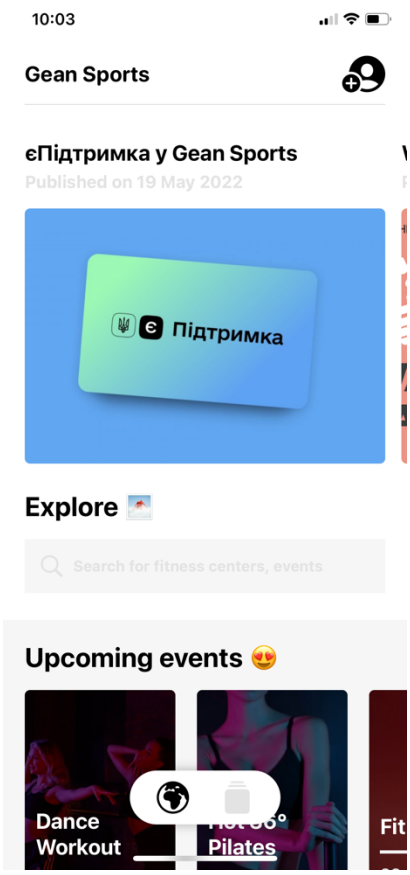


Рисунок 3.1 – Сторінка Explore

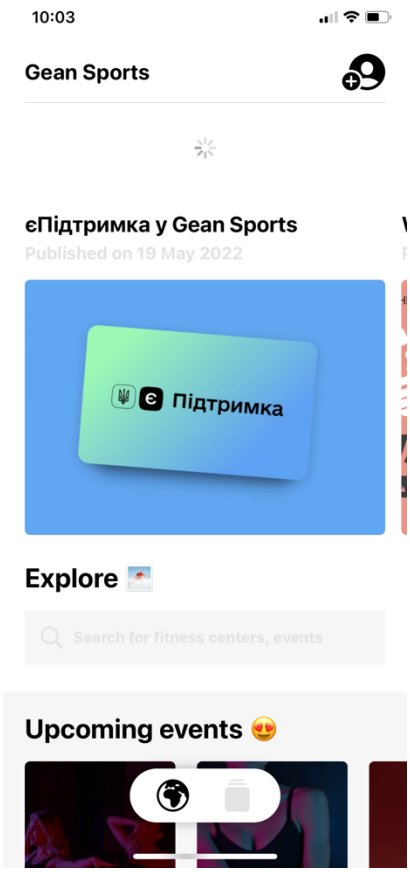


Рисунок 3.2 – Сторінка Explore, яка оновлює дані

3.2 Сторінка Sign in/Sign up

Ця сторінка одночасно відповідає за вхід та створення облікового запису. Можна побачити лише одну кнопку «Continue with Apple» (рис. 5.3), при натисканні на яку у користувача з’явиться вікно авторизації (рис. 5.4). Після успішної авторизації, сторінка Sign in/Sign up автоматично закриється.

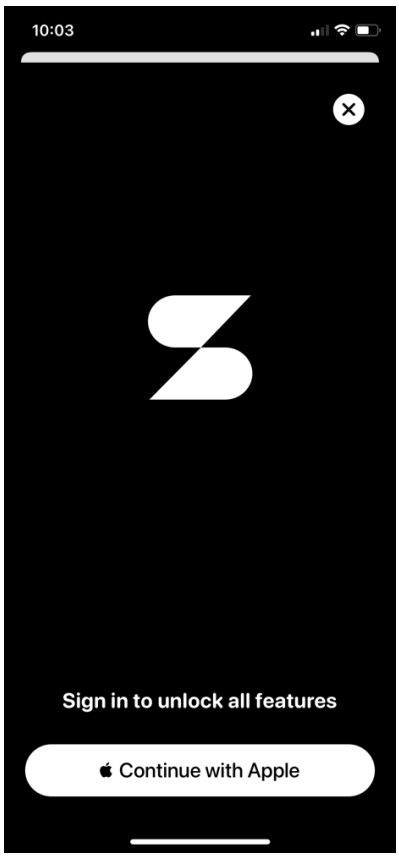


Рисунок 3.3 – Сторінка Sign In/Sign up

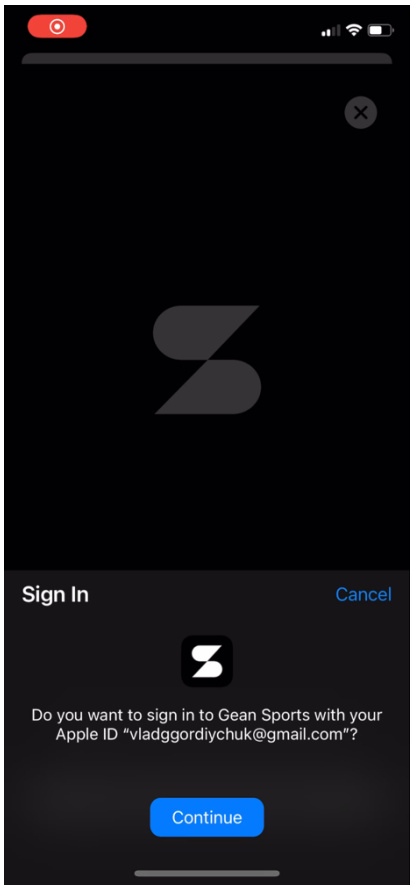


Рисунок 3.4 – Вікно «Continue with Apple» на сторінці Sign in/Sign up

3.3 Сторінка News

Ця сторінка є повністю інформативною (рис 5.5), та не має прихованого функціоналу. Елементи інтерфейсу включають:

– баннер новини;

– назва новини;

– дата публікація новини;

– опис новини.

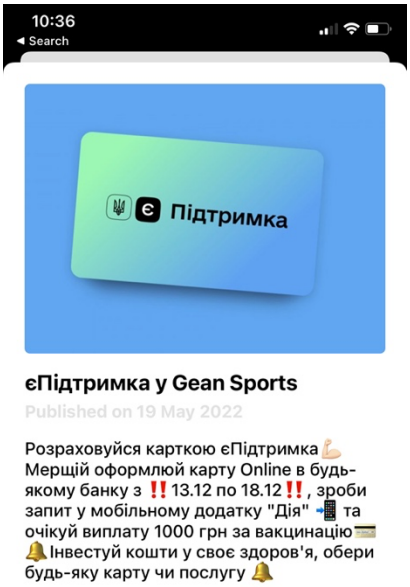


Рисунок 3.5 – Сторінка News

3.4 Сторінка Activity Details

Сторінка Activity Details призначена для перегляду детальної інформації фітнес-центру, або події (рис. 5.6). Окрім перегляду інформації, на цій сторінці можна побачити QR код (рис. 5.8), або купити доступ (рис. 5.7) до фітнес-центру, або події, яка переглядається.

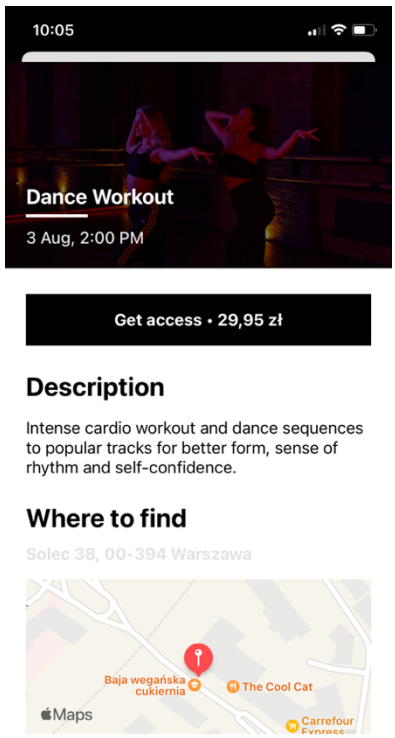


Рисунок 3.6 – Сторінка Activity Details з кнопкою «Get Access»

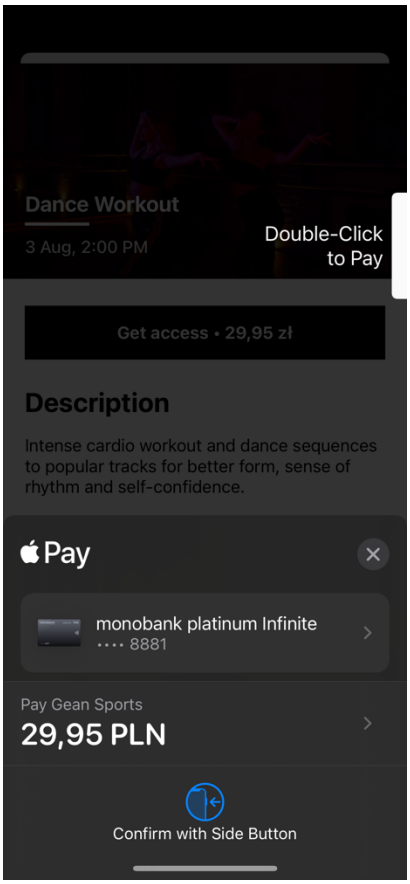


Рисунок 3.7 – Сторінка Activity Details з вікном оплати

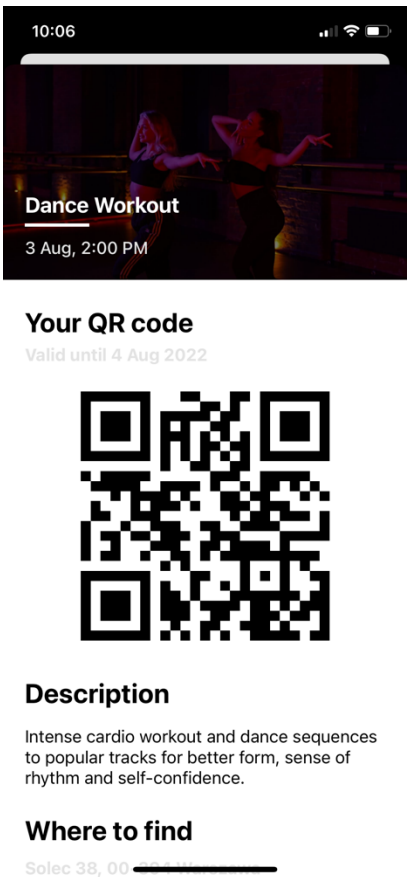


Рисунок 3.8 – Сторінка Activity Details з QR кодом

3.5 Сторінка Activity Passes

Ця сторінка призначена для перегляду вже придбаних доступів до фітнес-центрів та подій (рис. 5.9). Є можливість оновити контент потягнувши зверху-вниз. Якщо натиснути на будь який елемент із списку – користувач побачить сторінку з деталями обраного елемента.

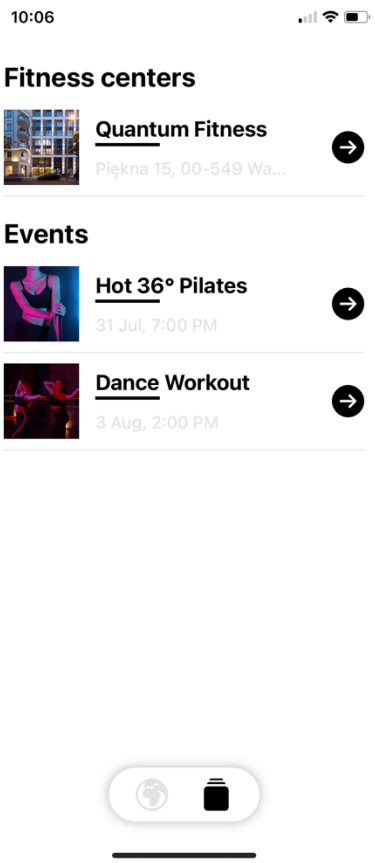


Рисунок 3.9 – Сторінка Activity Passes

3.6 Сторінка Search

Ця сторінка призначена для пошуку фітнес-центрів та подій (рис. 5.10). Для початку пошуку достатньо ввести будь яке значення в текстове поле і результат з’явиться майже в той самий час. Якщо натиснути на будь який елемент із списку – користувач побачить сторінку з деталями обраного елемента.

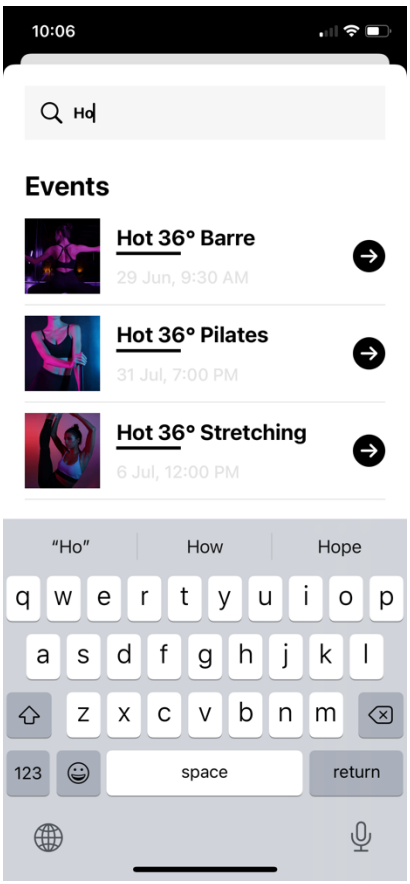


Рисунок 3.10 – Сторінка Search

3.7 Висновки

В рамках цього розділу було описано роботу програмного застосунку, який було розроблено під час виконання бакалаврської роботи.

Були розглянуті основні можливості та сценарії мобільного застосунку.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2022 р.

ПРОГРАМНИЙ ЗАСТОСУНОК ДЛЯ ОТРИМАННЯ ПОСЛУГ У ФІТНЕС ЦЕНТРАХ

Графічний матеріал

КПІ.ІП-8109.045440.05.99

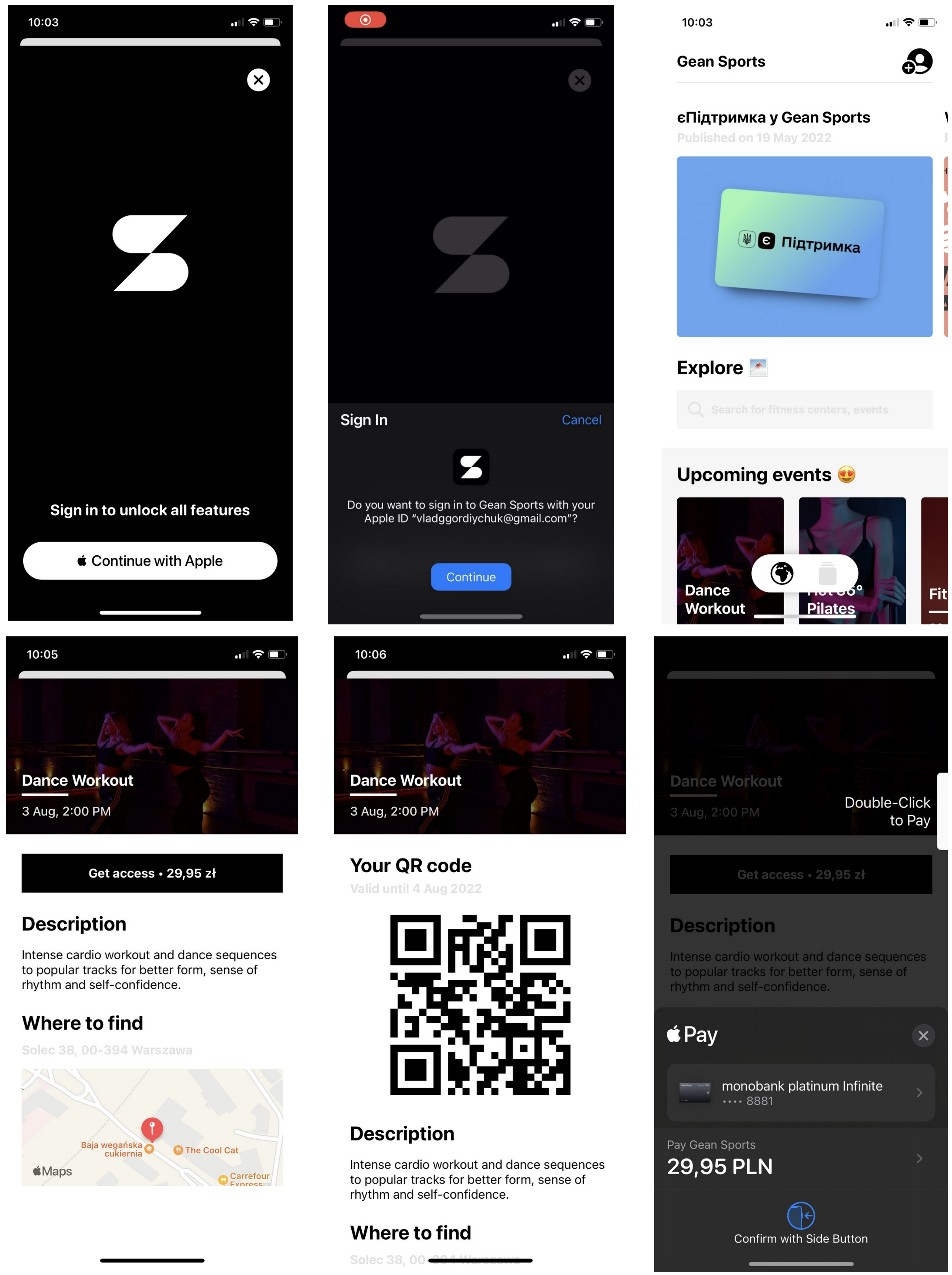
“ПОГОДЖЕНО”
Керівник проєкту:
_____ Микола ХРАМЧЕНКО

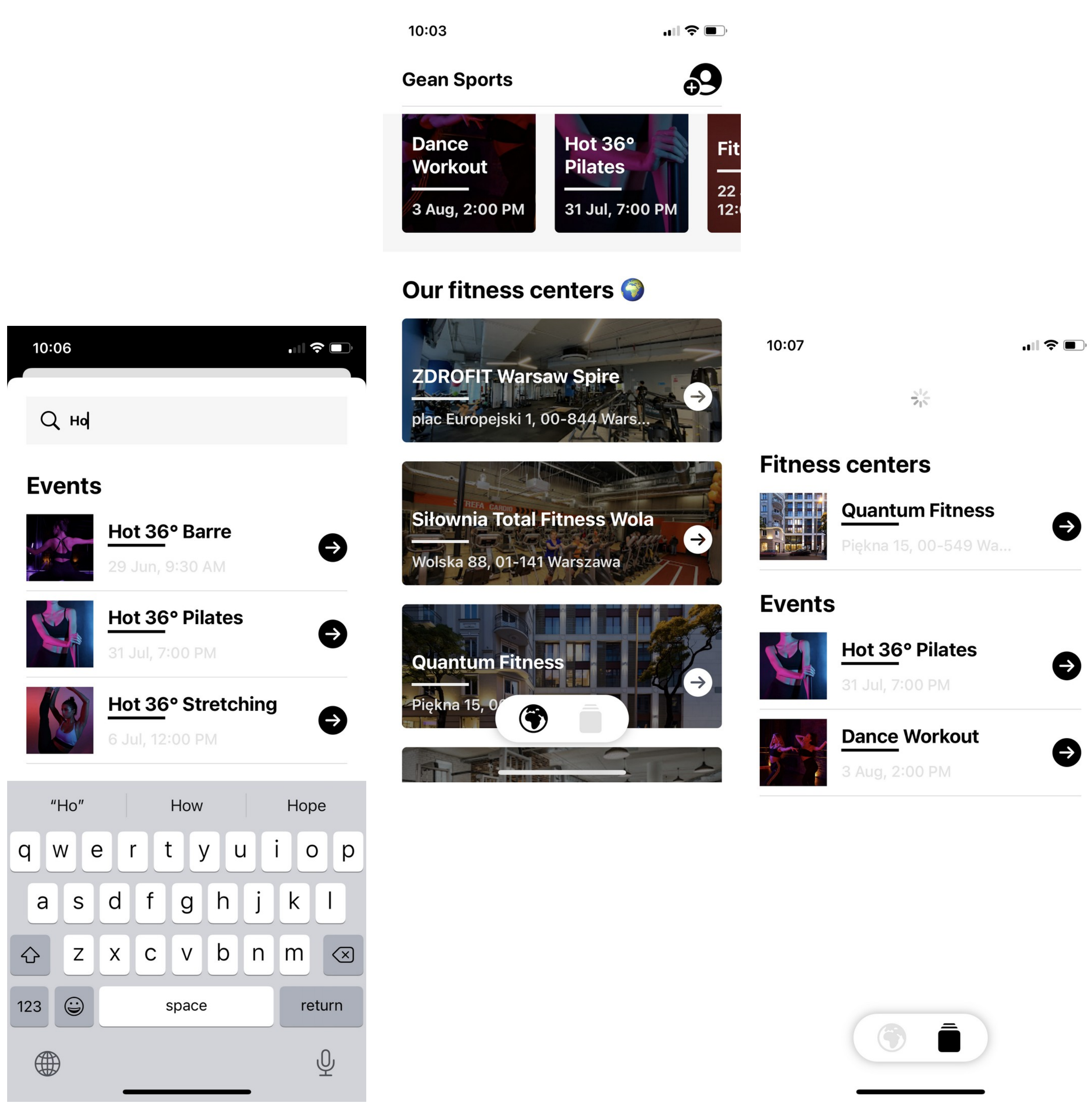
Нормоконтроль:
_____ Катерина ЛІЩУК

Виконавець:
_____ Владислав ГОРДІЙЧУК

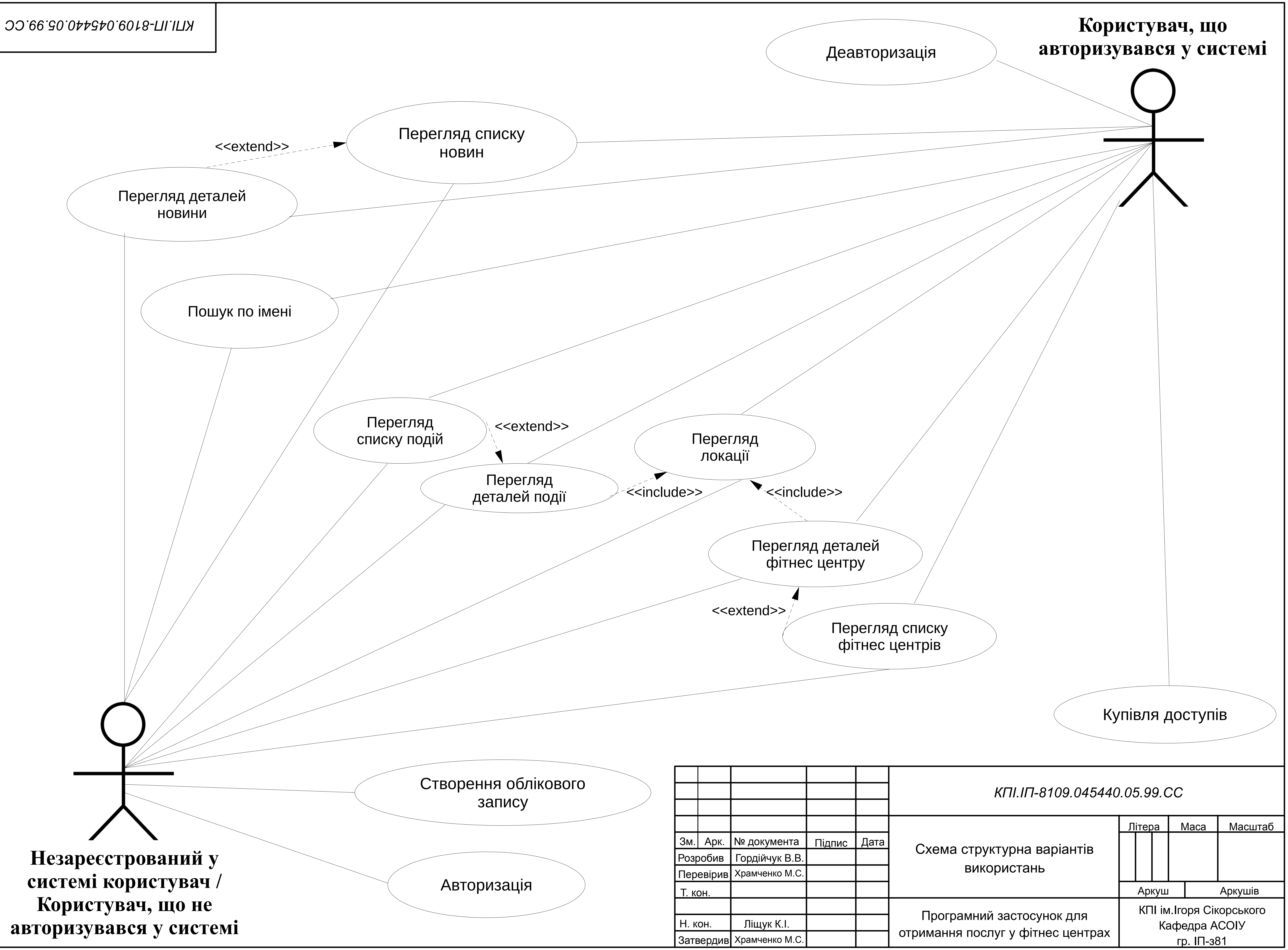
Київ – 2022

КПІ.ІП-8109.045440.05.99.КЕ

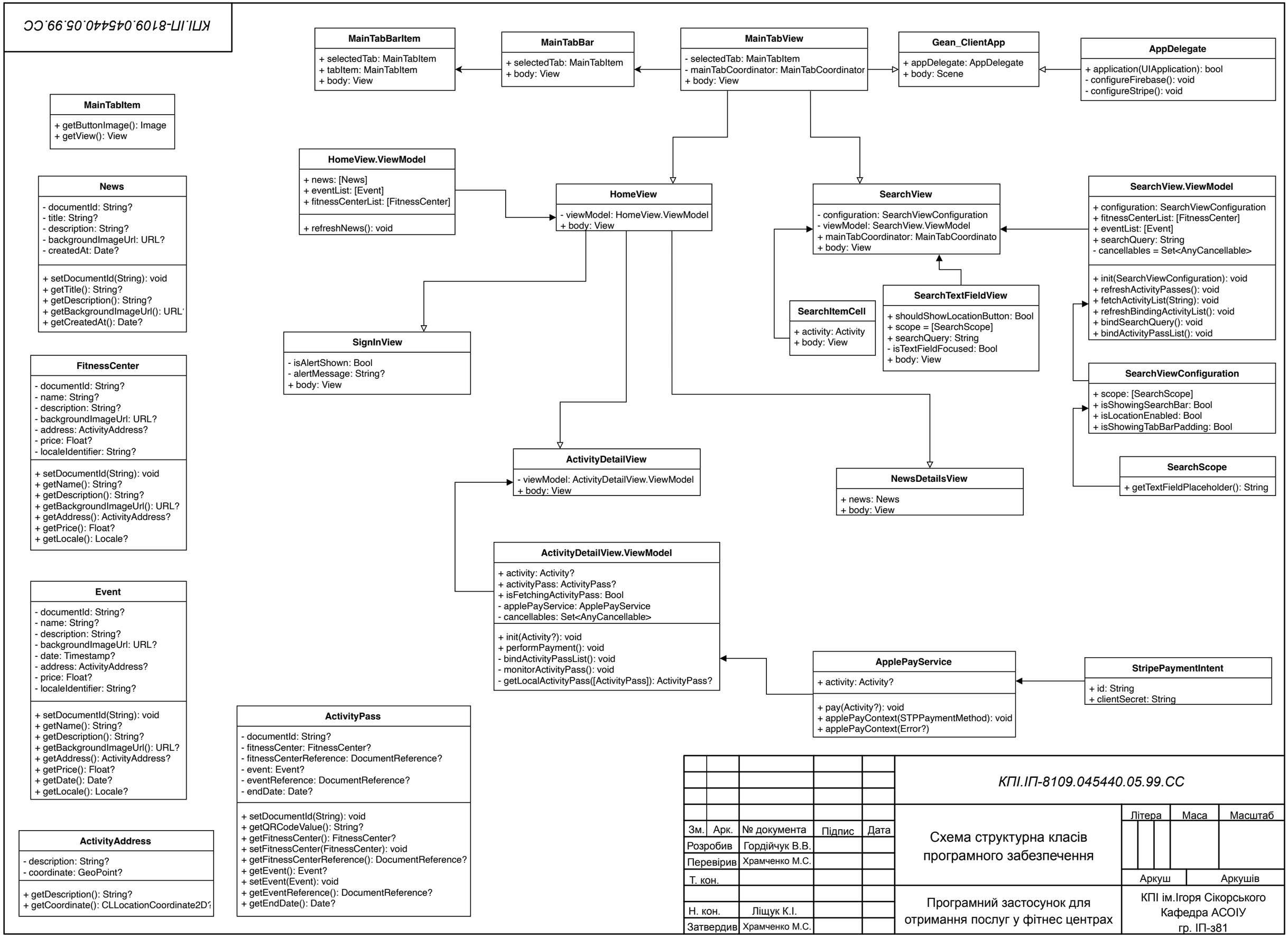




					КПІ.ІП-8109.045440.05.99.КЕ			
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера	Маса	Масштаб
Розробив		Гордійчук В.В.						
Перевірив		Храмченко М.С.						
Т. кон.						Аркуш		Аркушів
					Програмний застосунок для отримання послуг у фітнес центрах	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-381		
Н. кон.		Ліщук К.І.						
Затвердив		Храмченко М.С.						



					КПІ.ІП-8109.045440.05.99.СС			
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера	Маса	Масштаб
Розробив		Гордійчук В.В.						
Перевірив		Храмченко М.С.						
Т. кон.						Аркуш		Аркушів
Н. кон.		Ліщук К.І.			Програмний застосунок для отримання послуг у фітнес центрах	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-з81		
Затвердив		Храмченко М.С.						



					КПІ.ІП-8109.045440.05.99.СС
Зм.	Арк.	№ документа	Підпис	Дата	Літера / Маса / Масштаб
Розробив		Гордійчук В.В.			Схема структурна класів програмного забезпечення
Перевірив		Храмченко М.С.			Аркуш / Аркушів
Т. кон.					
Н. кон.		Ліщук К.І.			Програмний застосунок для отримання послуг у фітнес центрах
Затвердив		Храмченко М.С.			КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-з81

UNICHECK

Имя пользователя:
Лісовиченко Олег Іванович

ID проверки:
1011469537

Дата проверки:
06.06.2022 07:45:54 EEST

Тип проверки:
Doc vs Internet + Library

Дата отчета:
06.06.2022 07:47:40 EEST

ID пользователя:
76913

Название файла: **ІП-з81_Гордійчук_ПЗ**

Количество страниц: **66** Количество слов: **6921** Количество символов: **50685** Размер файла: **14.70 MB** ID файла: **1011347647**

Обнаружены модификации текста (могут влиять на процент совпадений)

13.7%
Совпадения

Наибольшее совпадение: 3.84% с источником из Библиотеки (ID файла: **1009700928**)

0.94% Источники из Интернета	24	Страница 68
13.7% Источники из Библиотеки	145	Страница 68

0% Цитат

Исключение цитат выключено

Исключение списка библиографических ссылок выключено

0% Исключений

Нет исключенных источников

Модификации

Обнаружены модификации текста. Подробная информация доступна в онлайн-отчете.

Подозрительное форматирование	20 страниц