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)
"___"_____2023 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Мобільний застосунок для організації та планування спільних спортивних подій

Виконав студент IV курсу, групи IT-94
(шифр групи)

Хоменко Олександр Сергійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Крамар Ю. М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації старший викладач, Вітковська І. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент доцент, к.т.н., доц., Ткач М. М. _____
(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 –2023

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 інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2023 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Хоменку Олександру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Мобільний застосунок для організації та планування спільних спортивних подій

керівник проєкту Крамар Юлія Михайлівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «31» травня 2023 р. №2101-с

2. Термін подання студентом проєкту «17» червня 2023 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі.
2) Інформаційне забезпечення: вхідні дані, опис структури бази даних.
3) Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів.
4) Технологічний розділ: керівництво користувача, методика випробувань програмного продукту.

5. Перелік графічного матеріалу

1) Схема структурна класів програмного забезпечення

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «10» березня 2023 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	10.03.2023	
2	Аналіз існуючих методів розв’язання задачі	20.03.2023	
3	Постановка та формалізація задачі	23.03.2023	
4	Розробка інформаційного забезпечення	14.04.2023	
5	Алгоритмізація задачі	16.04.2023	
6	Обґрунтування вибору використаних технічних засобів	27.04.2023	
7	Розробка програмного забезпечення	10.05.2023	
8	Налагодження програми	24.05.2023	
9	Виконання графічних документів	01.06.2023	
10	Оформлення пояснювальної записки	02.06.2023	
11	Подання ДП на попередній захист	06.06.2023	
12	Подання ДП рецензенту	12.06.2023	
13	Подання ДП на основний захист	17.06.2023	

Студент _____ Олександр ХОМЕНКО

(підпис) (ініціали, прізвище)

Керівник _____ Юлія КРАМАР

(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 35 таблиць, 18 рисунків та 10 джерел – загалом 55 сторінки.

Дипломний проєкт присвячений розробці мобільного застосунку для організації та планування спільних спортивних подій.

Метою роботи є збільшення мотивації людей до зайняття спортом, шляхом створення зручної платформи для пошуку партнерів для спортивних занять та зміцнення спортивної спільноти.

Об'єкт дослідження: мобільний застосунок для організації та планування спільних спортивних подій.

Предмет дослідження: проектування та розробка мобільного застосунку, який дозволить користувачам легко знаходити і приєднуватися до спільних спортивних подій.

У розділі "Аналіз вимог до програмного забезпечення" представлено змістовний опис і аналіз предметної області. Також виконано аналіз існуючих технологій та успішних IT-проєктів. Поставлено задачу та сформульовано цілі розробки.

У другому розділі "Моделювання та конструювання програмного забезпечення" розглянуто мову програмування, а також представлено структуру проєкту та опис класів. Розглянуто архітектуру програмного забезпечення та проведено конструювання програмного забезпечення.

У третьому розділі "Аналіз якості та тестування програмного забезпечення" описано процеси тестування та представлено контрольний приклад.

У четвертому розділі "Впровадження та супровід програмного забезпечення" описано процес розгортання програмного забезпечення та підтримки його функціонування.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, IOS, SWIFT, FIREBASE, XCODE, СПОРТ, БАЗА ДАНИХ.

ABSTRACT

Explanatory note of the diploma project consists of four chapters, includes 35 tables, 18 figures, and 10 sources - totaling 55 pages.

The diploma project is dedicated to the development of a mobile application for organizing and planning joint sports events.

The aim of the work is to increase people's motivation for engaging in sports by creating a convenient platform for finding partners for sports activities and strengthening the sports community.

Research Object: mobile application for organizing and planning joint sports events.

Research Subject: designing and developing a mobile application that will allow users to easily find and join in on shared sports events.

The chapter "Analysis of software requirements" presents a comprehensive description and analysis of the subject area. An analysis of existing technologies and successful IT projects has also been conducted. The tasks are defined, and the development goals are formulated.

In the second chapter, "Modeling and Designing Software," the programming language is discussed, and the project structure and class descriptions are presented. The software architecture is examined, and software construction is performed.

The third chapter, "Quality Analysis and Software Testing," describes the testing processes and provides a sample test case.

In the fourth chapter, "Implementation and Software Support," the deployment process and support for the software's functioning are described.

KEYWORDS: MOBILE APP, IOS, SWIFT, FIREBASE, XCODE, SPORT, DATABASE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-9428.045490.01.91	Технічне завдання	13	
3	А4	КПІ.ІТ-9428.045490.02.81	Пояснювальна записка	55	
4	А4	КПІ.ІТ-9428.045490.03.12	Текст програми	46	
5	А4	КПІ.ІТ-9428.045490.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-9428.045490.05.34	Керівництво користувача	8	
7	А3	КПІ.ІТ-9428.045490.06.99	Схема бази даних	1	
8	А3	КПІ.ІТ-9428.045490.07.99	Схема структурна класів	1	
			програмного забезпечення		
9	А3	КПІ.ІТ-9428.045490.08.99	Креслення вигляду екранних форм	1	

Змін	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9428.045490.00.90		
Розроб.		Хоменко О.С.			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Крамар Ю.М.					
Н.контр.		Вітковська І.І.				КПІ ім. Ігоря Сікорського ФІОТ каф.	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ТА ПЛАНУВАННЯ СПІЛЬНИХ СПОРТИВНИХ ПОДІЙ

Технічне завдання

КПІ.ІТ-9428.045490.01.91

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Олександр ХОМЕНКО

Київ – 2023

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ТА ПЛАНУВАННЯ СПІЛЬНИХ СПОРТИВНИХ ПОДІЙ.

Галузь застосування:

Наведене технічне завдання поширюється на розробку програмного застосунку під платформу iOS «Мобільний застосунок для організації та планування спільних спортивних подій» [КПІ.ІТ-9428.045490.01.91], котра використовується та пошук партнерів для занять спортом та призначена для людей будь-якого віку та рівня фізичної підготовки, які бажають займатися спортом в компанії інших людей.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки програмного застосунку під платформу iOS «Мобільний застосунок для організації та планування спільних спортивних подій» 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 створення мобільного застосунку, який допоможе користувачам знайти і приєднатися до спільних спортивних подій, що відбуваються у їхньому окрузі.

Метою розробки є збільшення мотивації людей до зайняття спортом, створення зручної платформи для пошуку партнерів для спортивних занять та зміцнення спортивної спільноти.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– Форма реєстрації для введення необхідних даних користувача, таких як ім'я, електронна пошта, пароль тощо.

– Форма авторизації для введення ідентифікатора користувача.

– Можливість деавторизуватися, щоб вийти з облікового запису.

– Форма для створення нової спортивної події, включаючи назву, дату, час, місцезнаходження, тип спорту.

– Відображення списку доступних спортивних подій з основними деталями, такими як назва, дата, час, місцезнаходження.

– Фільтрація подій за типом спорту та місцезнаходженням:

– Мапа, на якій можна шукати та переглядати спортивні події в конкретному регіоні або на певній місцевості.

– Можливість видалити спортивну подію, якщо вона стала неактуальною або скасованою.

– Відображення розширеної інформації про обрану спортивну подію.

– Перехід до чату з іншим користувачем зі сторінки спортивної події або профілю користувача.

Прототипи екранних форм зображено на рисунках 4.1 – 4.2:

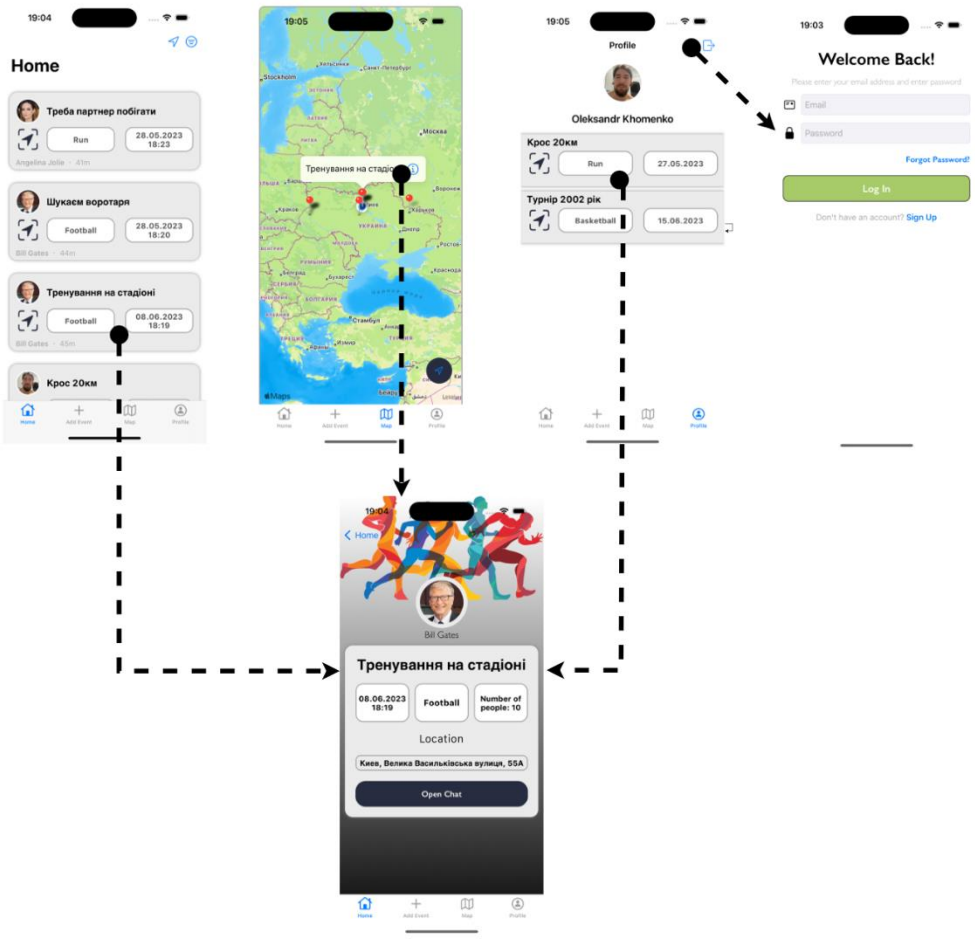


Рисунок 4.1 – Прототип головних екранів

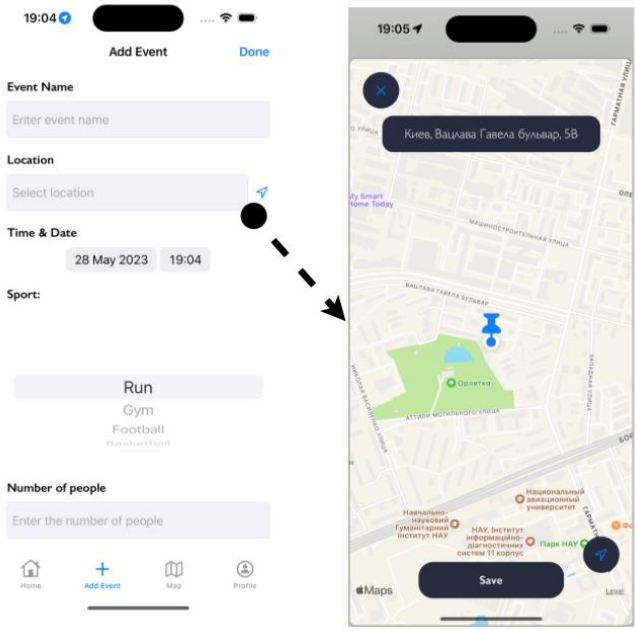


Рисунок 4.2 – Прототип екрану створення події

4.1.2 Для користувача

- створення облікового запису;
- авторизація та деавторизація користувача;
- створення спортивних подій;
- перегляд списку спортивних подій;
- фільтрація подій за типом спорту та місцезнаходженням;
- пошук спортивних подій на мапі;
- видалення події в разі неактуальності;
- перегляд детальної інформації події;
- перехід до чату з іншим користувачем.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних. В разі виникнення помилок програма повинна бути здатна відновлюватися та продовжувати свою роботу та працювати на різних обсягах даних, а також змінювати свою продуктивність відповідно до потреб користувачів.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються.

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються.

4.4 Вимоги до складу і параметрів технічних засобів

Мінімальна конфігурація технічних засобів:

- тип процесору: Apple A8;

– об‘єм ОЗП: 1 ГБ.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем iOS 15.0 або вище. Розробку виконати на платформі macOS Monterey.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі:

– назва події (наприклад «Тренування»);
– вид тренування (наприклад «Біг»);
– локація (наприклад «Центральний стадіон, Київ»);
– час.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі:

– відображення події на мапі у вигляді анотації або у вигляді списку.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Swift, використовуючи фреймворк UIKit для створення інтерфейсу користувача.

4.5.4 Вимоги до середовища розробки

Розробку виконати на платформі XCode.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема бази даних;
– схема структурна класів програмного забезпечення;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

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

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.03	
2.	Розробка технічного завдання	03.04	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	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: Мобільний застосунок для організації та планування спільних спортивних подій

КПІ.ІТ-9428.045490.02.81

Київ – 2023

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	–	Application programming interface - прикладний програмний інтерфейс.
ER	–	Entity-Relation diagram.
IDE	–	Integrated Development Environment – інтегроване середовище розробки.
IT	–	Інформаційні технології.
MVVM	–	Model View ViewModel.
SDK	–	Software development kit - набір для розробки програмного забезпечення.
UI	–	User Interface – інтерфейс користувача.
БД	–	База даних.
ОС	–	Операційна система.
ПК	–	Персональний комп’ютер.

ВСТУП

Зважаючи на постійний розвиток технологій та збільшення числа смартфонів в користуванні, стає дедалі важливішим розробляти мобільні застосунки для полегшення різних сфер життя людей. Однією з таких сфер є спорт. Відсутність мотивації та недостатній рівень організації можуть стати перешкодою для людей, які бажають займатися спортом.

Крім того, сьогодні велика увага приділяється здоровому способу життя. Зайнятість, стрес та відсутність вільного часу часто стають причинами, чому люди не займаються спортом. Але якщо людина знаходить собі партнерів для спільних занять, то її мотивація зростає і вона більш активно займається спортом.

Мобільний застосунок для організації та планування спільних спортивних подій може бути ефективним інструментом для підвищення рівня фізичної активності серед людей. Завдяки йому користувачі зможуть знайти нових друзів, знайомих чи колег, які так само бажають займатися спортом, і разом з ними досягнути своїх спортивних цілей.

Також варто зазначити, що мобільні застосунки стали невід'ємною частиною нашого повсякденного життя. Вони дозволяють нам ефективніше управляти своїм часом, спілкуватися з іншими людьми та здійснювати безліч інших різноманітних дій. Розробка мобільного застосунку для організації та планування спільних спортивних подій є актуальною задачею, яка може привернути увагу багатьох користувачів та стати успішним інструментом для залучення людей до здорового способу життя.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

У сучасному світі зайнятість та повсякденні справи часто заважають людям займатися спортом та брати участь в спортивних подіях. Брак мотивації, важкість знаходження спортивних партнерів та обмежений доступ до інформації про спортивні події можуть стати перешкодами для активного та здорового способу життя.

З метою вирішення цих проблем та стимулювання спортивної активності серед населення, пропонується розробка мобільного застосунку. Цей застосунок буде служити зручною та легко доступною платформою для пошуку та приєднання до спільних спортивних подій, що відбуваються у окрузі користувача.

Для досягнення поставленої мети буде використовуватись комплексний підхід, включаючи вивчення існуючих рішень та технологій, розробку функціональності та інтерфейсу застосунку, а також впровадження тестування та забезпечення безпеки та якості.

Результати розробки даного мобільного застосунку можуть сприяти популяризації спорту, створенню активної спортивної спільноти та поліпшенню здоров'я та життєвого стилю користувачів.

1.2 Змістовний опис і аналіз предметної області

Аналізуючи предметну область, пов'язану з розробкою мобільного застосунку для пошуку та приєднання до спортивних подій, важливо досліджувати мотиваційні фактори, які впливають на залучення людей до зайняття спортом, а також враховувати існуючі розробки і технології в сфері мобільних застосунків для спортивних заходів. Основні цілі розробки полягають у збільшенні мотивації людей до зайняття спортом, створенні зручної платформи для пошуку спортивних партнерів та зміцненні спортивної спільноти. Мобільний застосунок може забезпечити розширення доступу до

інформації про спортивні події, що відбуваються у конкретному окрузі або регіоні. Він дозволяє організаторам подій просувати свої заходи та залучати більше учасників, а спортсменам - знайти цікаві події та брати в них участь.

Загальний аналіз предметної області включає розгляд таких аспектів:

– Спортивна спільнота. Наявність спортивної спільноти позитивно впливає на мотивацію людей до зайняття спортом. Формування спільноти спортсменів та ентузіастів сприяє обміну досвідом, підтримці та взаємодії між учасниками.

– Мотивація до зайняття спортом. Важливим фактором успішного залучення людей до спортивних активностей є стимулювання їхньої мотивації. Це може бути досягнення конкретних цілей, соціальна взаємодія, покращення фізичного стану тощо. Розуміння мотиваційних факторів допоможе розробити ефективні інструменти для залучення користувачів до застосунку.

– Мобільні застосунки для спортивних заходів. В сучасному світі мобільні застосунки широко використовуються в спортивних сферах. Існують застосунки, які допомагають користувачам знаходити та реєструватися на спортивні події, спілкуватися з іншими спортсменами, відстежувати свої досягнення та виконувати тренувальні програми. Аналіз цих застосунків дозволяє визначити їхні переваги, недоліки та інноваційні можливості для розробки власного застосунку.

– Пошук партнерів для спортивних занять. Один з головних аспектів мобільного застосунку - це створення зручного механізму пошуку та вибору партнерів для спортивних занять.

1.3 Аналіз існуючих технологій та успішних ІТ-проєктів

Провівши аналіз, було виявлено наступні існуючі на сьогодні мобільні застосунки для організації спільних спортивних подій.

1.3.1 RacketPal

RacketPal - це мобільний застосунок, що дозволяє користувачам знаходити ближніх гравців та бронювати корти для гри в бадмінтон, теніс та настільний теніс. Він дозволяє користувачам з'єднуватися з іншими гравцями і графіком гри у зручний час та місце. На рисунку 1.1 зображено основні екрани застосунку RacketPal.

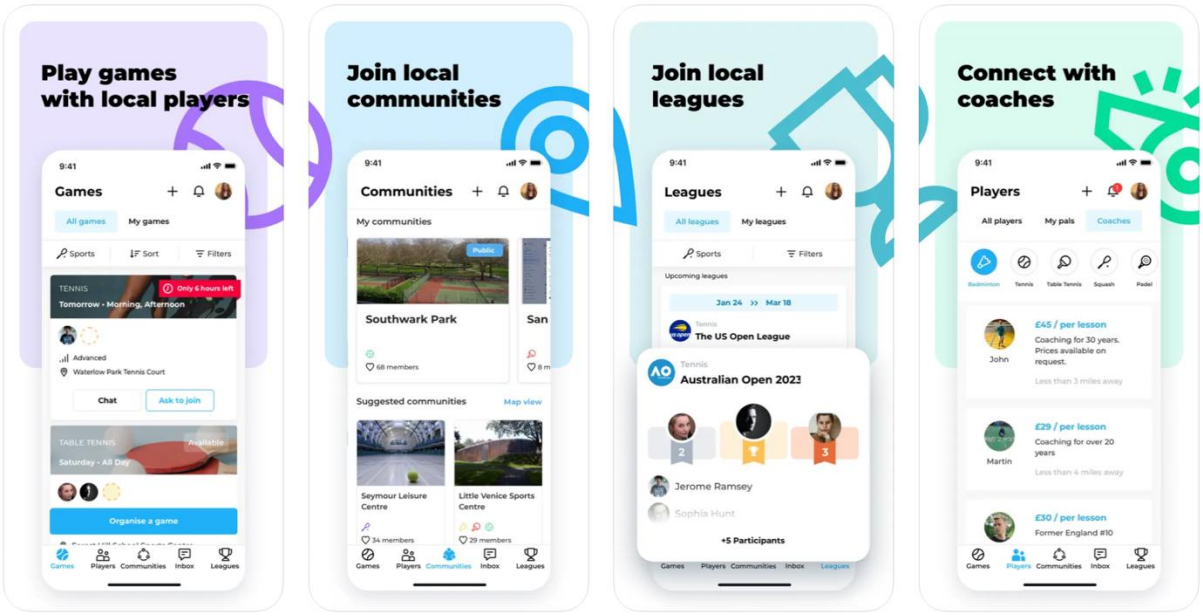


Рисунок 1.1 – Мобільний застосунок RacketPal

Переваги:

– RacketPal надає дуже зручний інтерфейс.

– Застосунок дозволяє користувачам відстежувати та керувати своїми гральними графіками та бронюваннями.

– RacketPal пропонує соціальну платформу для гравців, де вони можуть створювати та приєднуватися до спортивних заходів, що полегшує знаходження партнерів для гри.

Недоліки:

– RacketPal обмежений лише трьома видами спорту, що може не влаштовувати всіх користувачів.

– Застосунок може мати обмежену аудиторію, оскільки доступний лише в кількох країнах.

Функціональність:

– Знаходження ближніх гравців та планування ігор

– Бронювання кортів для гри в бадмінтон, теніс та настільний теніс

– Створення та приєднання до спортивних заходів

1.3.2 Playo

Playo - це спортивний застосунок, що дозволяє користувачам знаходити та бронювати спортивні майданчики, приєднуватися до спортивних заходів та зв'язуватися з гравцями у своєму районі. Застосунок має функцію ведення статистики і фіксування результатів, що полегшує гравцям відстежувати свій прогрес. Головні екрани застосунку Playo представлено на рисунку 1.2.

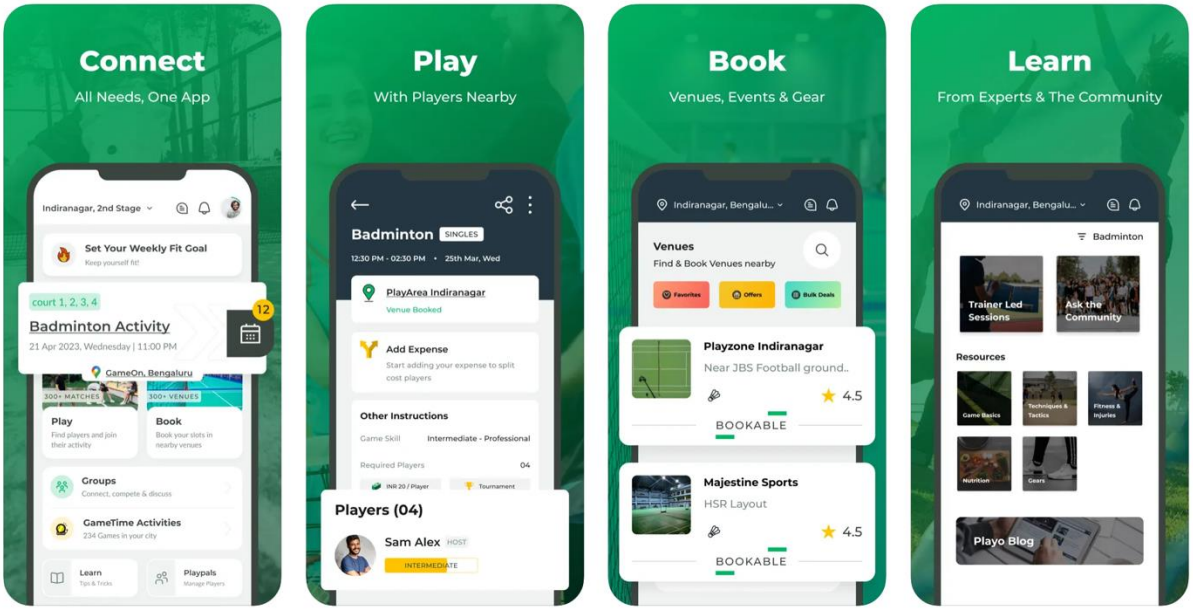


Рисунок 1.2 – Мобільний застосунок Playo

Переваги:

– Playo дозволяє користувачам знаходити та приєднуватися до спортивних заходів у своєму районі.

– Застосунок має функцію ведення статистики, що дозволяє користувачам відстежувати свій прогрес.

– Playo надає соціальну платформу для зв'язку гравців, де вони можуть знайти напарників та суперників.

Недоліки:

– Застосунок Playo націлений головним чином на індійських любителів спорту, що робить його важко доступним для міжнародних користувачів.

– Інтерфейс застосунку може бути заплутаним для деяких користувачів.

– Час завантаження застосунку може бути досить повільним, що під час очікування може дратувати користувачів.

Функціональність:

– Знаходження та бронювання спортивних майданчиків

– Ведення статистики і фіксування результатів

– Приєднання до спортивних заходів та зв'язок з гравцями.

1.3.3 Sportido

Sportido - це мобільний застосунок, що надає користувачам послуги бронювання спортивних майданчиків, сповіщення про спортивні заходи та можливість придбати квитки на спортивні події онлайн. На рисунку 1.3 наведено екрани застосунку Sportido.

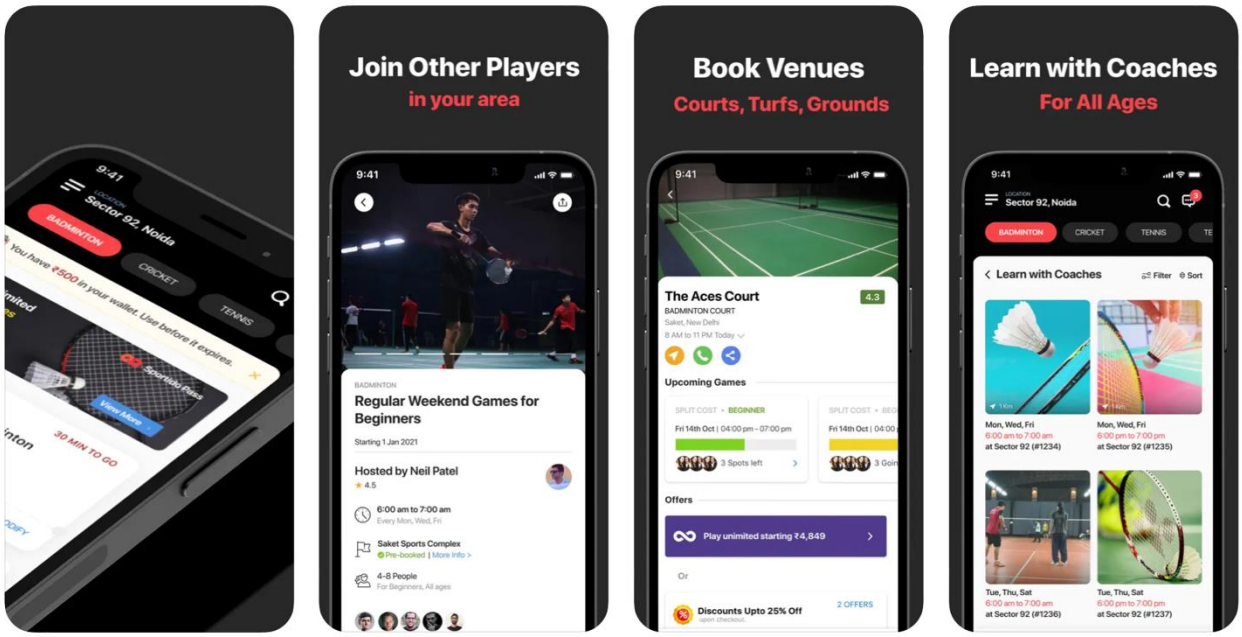


Рисунок 1.3 – Мобільний застосунок Sportido

Переваги:

– Sportido надає реальний час бронювання спортивних майданчиків.

– Застосунок пропонує користувачам можливість отримувати сповіщення про майбутні спортивні заходи.

– Sportido дозволяє користувачам придбати квитки на спортивні події онлайн, що полегшує процес.

Недоліки:

– Sportido має обмежену кількість доступних видів спорту.

– Інтерфейс застосунку потребує поліпшення, оскільки не є досить інтуїтивним.

В таблиці 1.1 було наведено порівняння між існуючими застосунками. Можна зробити висновок, що застосунок Sports Meeting, який планується розробити, надає більш широкі можливості щодо різноманітності видів спорту і доступності по всьому світу, а також відображення подій на мапі.

Таблиця 1.1 – Порівняння застосунків

Функціонал	RacketPal	Playo	Sportido	Sports Meeting
Знаходження користувачів поблизу	+	+	+	+
Різноманітність видів спорту	-	-	-	+
Доступність по всьому світу	+	-	-	+
Інтуїтивний дизайн	+	-	-	+
Відображення подій на мапі	+	-	-	+

1.4 Аналіз вимог до програмного забезпечення

Визначимо загальні категорії користувачів, які будуть використовувати розроблений мобільний застосунок для пошуку спільних спортивних подій.

1. Користувачі, що не мають облікового запису в системі.
2. Користувачі з власним обліковим записом в системі.

Система повинна мати наступні функціональні вимоги:

– створення облікового запису;
– авторизація та деавторизація користувача;
– створення спортивних подій;
– перегляд списку спортивних подій;
– фільтрація подій за типом спорту та місцезнаходженням;
– пошук спортивних подій на мапі;
– видалення події в разі неактуальності;
– перегляд детальної інформації події;
– перехід до чату з іншим користувачем.

Водночас, система повинна бути дружелюбною та інтуїтивно зрозумілою для користувачів, щоб вони легко користувались нею та мотивувалися до зайняття спортом.

1.4.1 Розроблення функціональних вимог

Визначивши функції, які має виконувати мобільний застосунок для зручного пошуку спортивних подій, були ідентифіковані основні сценарії використання застосунку, які представлені в таблицях 1.2 – 1.10.

Таблиця 1.2 – Варіант використання UC001

Назва	Створення облікового запису
Опис	Реєстрація користувача для подальшого входу у застосунок.
Учасники	Користувач
Передумови	Користувач немає облікового запису
Постумови	Створений обліковий запис користувача

Продовження таблиці 1.2

Основний сценарій	1. Користувач переходить з екрану входу на екран реєстрації. 2. Користувач вводить персональні дані, обов’язково пароль та email. 3. Система перевіряє валідність даних. 4. Система додає користувача в БД та відкриває головний екран.

Таблиця 1.3 – Варіант використання UC002

Назва	Вихід з облікового запису
Опис	Вихід користувача з облікового запису
Учасники	Користувач
Передумови	Користувач авторизований у застосунку
Постумови	Користувач вийшов з облікового запису
Основний сценарій	1. Користувач натискає на кнопку "Вийти". 2. Система закриває сеанс користувача та переходить на екран входу.

Таблиця 1.4 – Варіант використання UC003

Назва	Створення події
Опис	Можливість користувача створити нову спортивну подію.
Учасники	Користувач
Передумови	Користувач авторизований у застосунку
Постумови	Створена нова спортивна подія в системі

Продовження таблиці 1.4

Основний сценарій	1. Користувач переходить на екран "Додати подію". 2. Користувач вводить необхідну інформацію про подію, таку як назва, дата, час, місце, тип спорту тощо. 3. Система перевіряє правильність введених даних та зберігає нову спортивну подію в БД. 4. Система оновлює список спортивних подій на головному екрані.

Таблиця 1.5 – Варіант використання UC004

Назва	Перегляд події в списку
Опис	Можливість користувача переглянути список доступних спортивних подій.
Учасники	Користувач
Передумови	Користувач авторизований у застосунку
Постумови	Користувач переглянув список спортивних подій
Основний сценарій	1. Користувач відкриває головний екран застосунку. 2. Система відображає список доступних спортивних подій, включаючи їх назву та тип подій. 3. Користувач переглядає список подій та може вибрати подію для отримання більш детальної інформації.

Таблиця 1.6 – Варіант використання UC005

Назва	Перегляд події на мапі
Опис	Можливість користувача переглянути місця проведення спортивних подій на мапі.
Учасники	Користувач

Продовження таблиці 1.6

Передумови	Користувач авторизований у застосунку
Постумови	Користувач переглянув місця проведення спортивних подій на мапі
Основний сценарій	1. Користувач переходить на екран мапи. 2. Система відкриває карту, на якій позначені місця проведення спортивних подій. 3. Користувач може збільшувати, зменшувати та пересувати карту, щоб переглянути різні області. 4. Користувач може натиснути на мітку події на карті, щоб отримати додаткову інформацію.

Таблиця 1.7 – Варіант використання UC006

Назва	Перегляд детальної інформації події
Опис	Можливість користувача переглянути детальну інформацію про обрану спортивну подію.
Учасники	Користувач
Передумови	Користувач авторизований у застосунку та відкрито головний екран або мапу
Постумови	Користувач переглянув детальну інформацію про спортивну подію
Основний сценарій	1. Користувач вибирає спортивну подію зі списку або з мапи. 2. Система відображає детальну інформацію про обрану подію, таку як назва, дата, час, місце, тип спорту, опис тощо. 3. Користувач може переглядати всю детальну інформацію та перейти до чату.

Таблиця 1.8 – Варіант використання UC007

Назва	Перехід до чату з користувачем обраної події
Опис	Можливість користувача перейти до чату з іншим користувачем для спілкування та узгодження деталей спортивної події.
Учасники	Користувач
Передумови	Користувач авторизований у застосунку та має спортивну подію для спілкування
Постумови	Користувач перейшов до чату з іншим користувачем
Основний сценарій	1. Користувач відкриває детальну інформацію про спортивну подію. 2. Користувач натискає на кнопку "Відкрити чат" 3. Система відкриває чат, обраний користувачем

Таблиця 1.9 – Варіант використання UC008

Назва	Фільтрація подій за типом спорту та місцезнаходженням
Опис	Користувач може використовувати фільтри для відображення спортивних подій за певним типом спорту та/або у певному місцезнаходженні.
Учасники	Користувач
Передумови	Користувач авторизований у застосунку та має спортивну подію для спілкування
Постумови	Користувач переглянув список спортивних подій

Продовження таблиці 1.9

Основний сценарій	1. Користувач відкриває список спортивних подій. 2. Користувач натискає на опцію фільтрації подій. 3. Користувач обирає тип спорту та/або місцезнаходження для фільтрації. 4. Застосунок відображає список спортивних подій, що відповідають обраним критеріям фільтрації. 5. Користувач може переглянути докладну інформацію про вибрану подію або здійснити інші дії зі спортивними подіями, які задовольняють фільтрам.

Таблиця 1.10 – Варіант використання UC009

Назва	Видалення події в разі неактуальності
Опис	Користувач може видалити спортивну подію, яка стала неактуальною або більше не відбудеться.
Учасники	Користувач
Передумови	Користувач авторизований у застосунку
Постумови	Спортивна подія видалена з системи та більше не відображається для користувачів
Основний сценарій	1. Користувач відкриває свої спортивні події або сторінку з деталізованою інформацією про певну подію. 2. Користувач натискає на опцію видалення події. 3. Застосунок підтверджує намір користувача видалити подію. 4. Спортивна подія видаляється з системи та більше не відображається для користувачів.

Функціональні вимоги застосунку наведено у таблицях 1.11 – 1.18:

Таблиця 1.11 – Опис функціональної вимоги REQ001

Номер	REQ001
Назва	Створення облікового запису
Опис	Користувач має можливість створити новий обліковий запис у застосунку.

Таблиця 1.12 – Опис функціональної вимоги REQ002

Номер	REQ002
Назва	Створення спортивної події
Опис	Користувач може створювати нові спортивні події, вказуючи назву, дату, час, місцезнаходження та іншу інформацію про подію.

Таблиця 1.13 – Опис функціональної вимоги REQ003

Номер	REQ003
Назва	Перегляд списку спортивних подій
Опис	Користувач може переглядати список всіх доступних спортивних подій, які були створені іншими користувачами.

Таблиця 1.14 – Опис функціональної вимоги REQ004

Номер	REQ004
Назва	Пошук спортивних подій на мапі
Опис	Користувач може переглядати події, натискаючи на позначки на мапі та отримувати детальну інформацію про кожну подію.

Таблиця 1.15 – Опис функціональної вимоги REQ005

Номер	REQ005
Назва	Перегляд детальної інформації події
Опис	Користувач може передивитись детальну інформацію про подію.

Таблиця 1.16 – Опис функціональної вимоги REQ006

Номер	REQ006
Назва	Перехід до чату з іншим користувачем
Опис	Користувач може перейти до чату з іншим користувачем для спілкування та домовленостей щодо спортивних подій.

Таблиця 1.17 – Опис функціональної вимоги REQ007

Номер	REQ007
Назва	Фільтрація подій за типом спорту та місцезнаходженням
Опис	Користувач може застосовувати фільтри для пошуку спортивних подій за типом спорту та місцезнаходженням.

Таблиця 1.18 – Опис функціональної вимоги REQ008

Номер	REQ008
Назва	Видалення події в разі неактуальності
Опис	Користувач може видалити спортивну подію, якщо вона стала неактуальною.

У таблиці 1.19 представлено модель вимог, яка дозволяє відсортувати їх за пріоритетом, визначивши, які з них є найбільш важливими для виконання

проєкту. Це допомагає сконцентрувати зусилля на найважливіших елементах та забезпечити вчасну їх реалізацію. Модель вимог може включати оцінку ризиків для кожної вимоги. Це дозволяє ідентифікувати потенційні ризики, пов'язані з вимогами проекту, та прийняти відповідні заходи для їх управління.

Таблиця 1.19 – Модель вимог у загальному вигляді

TitleID	Full Description	Code	Priority	Risk	Status
1	Створення облікового запису	REQ001	3	Низький	Draft
2	Створення спортивних подій	REQ002	1	Середній	Draft
3	Перегляд списку спортивних подій	REQ003	2	Середній	Draft
4	Пошук спортивних подій на мапі	REQ004	2	Високий	Draft
5	Перегляд детальної інформації події	REQ005	3	Низький	Draft
6	Перехід до чату з іншим користувачем	REQ006	2	Середній	Draft
7	Фільтрація подій за типом спорту та місцезнаходженням	REQ007	2	Середній	Draft
8	Видалення події в разі неактуальності	REQ008	2	Низький	Draft

Для відстеження залежності та взаємозв’язків між вимогами та функціональністю було створено матрицю трасування, яка зображена на рисунку 1.4.

	REQ001(Створення облікового запису)	REQ002 (Створення спортивної події)	REQ003 (Перегляд списку спортивних подій)	REQ004 (Пошук спортивних подій на мапі)	REQ005 (Перегляд детальної інформації події)	REQ006 (Перехід до чату з іншим користувачем)	REQ007 (Фільтрація подій)	REQ008 (Видалення події в разі неактуальності)
UC001 (Створення облікового запису)	■							
UC002 (Вихід з облікового запису)	■							
UC003 (Створення події)		■						
UC004 (Перегляд події в списку)			■					
UC005 (Перегляд події на мапі)				■				
UC006 (Перегляд детальної інформації події)					■			
UC007 (Перехід до чату)						■		
UC008 (Фільтрація подій)							■	
UC009 (Видалення події в разі неактуальності)								■

Рисунок 1.4 – Матриця трасування

Для ідентифікації та візуалізації основних функцій, які можуть бути виконані користувачами, а також для того, щоб показати, як ці функції взаємодіють між собою та з системою, було створено діаграму варіантів використання (Use Case Diagram), яка представлена на рисунку 1.5. Діаграма варіантів використання допомагає уявити загальну картину функціональності системи та її взаємодію з користувачами, що дозволяє краще розуміти та аналізувати вимоги до системи та спрямовувати подальший процес розробки та тестування.

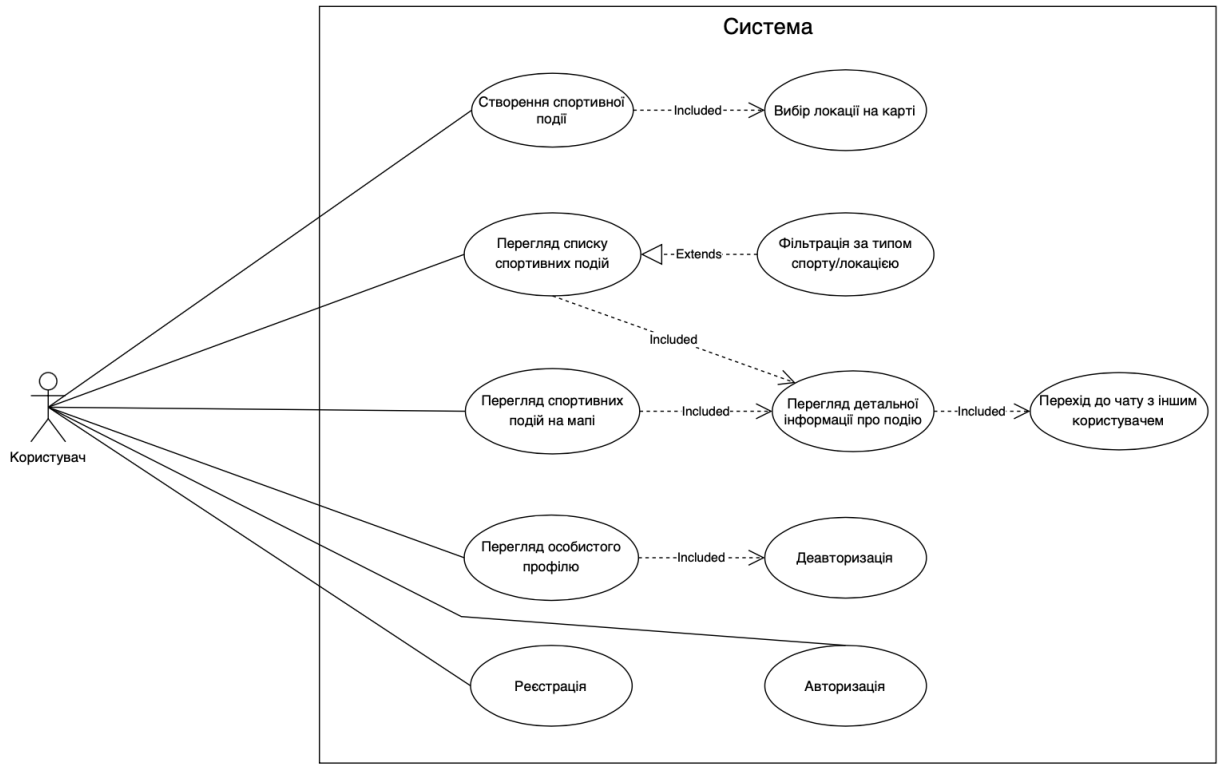


Рисунок 1.5 – Діаграма варіантів використання

1.4.2 Розроблення нефункціональних вимог

Основні нефункціональні вимоги мають бути наступними:

– Мобільний застосунок повинен підтримувати англійську мову як основну мову інтерфейсу для зручної інтерактивної взаємодії з користувачами.

– Мобільний застосунок повинен бути розроблений тільки для операційної системи iOS.

– Мобільний застосунок має бути спроєктований таким чином, щоб було можливо його використання тільки в портретній орієнтації.

– Мобільний застосунок має забезпечувати приватність і безпеку користувачів, щоб інформація про них не потрапляла в руки сторонніх осіб.

– Мобільний застосунок повинен працювати швидко та без затримок, щоб користувачі не зазнали незручностей під час пошуку та приєднання до спортивних подій.

– Мобільний застосунок повинен забезпечувати підтримку різних розмірів екрану мобільних пристроїв, щоб забезпечити належний комфорт використання користувачів на будь-яких пристроях.

1.5 Постановка задачі

Розробити мобільний застосунок, який допоможе користувачам знайти та приєднатися до спільних спортивних подій у їхньому окрузі, сприятиме збільшенню мотивації до занять спортом та зміцненню спортивної спільноти.

1.5.1 Призначення розробки

Створення зручної та функціональної платформи для швидкого та ефективного пошуку та приєднання до спільних спортивних заходів, що відбуваються в різних округах міста/регіону/країни. Застосунок буде допомагати знайти партнерів для спортивних занять, збільшувати мотивацію до занять спортом та сприяти розвитку спортивної спільноти.

1.5.2 Цілі розробки

Для розробки застосунку були поставлені наступні цілі:

– створення мобільного застосунку зі зручним та користувацьким інтерфейсом;

– розробка пошуку спортивних подій на мапі або у списку подій;

– розробка створення та додавання нових подій;

– тестування застосунку на різних пристроях;

– забезпечення безпеки користувачів.

Висновки до розділу

Отже, актуальність спорту та здорового способу життя стає все важливішою. Однак, існують проблеми, які перешкоджають людям брати участь в спортивних заходах, такі як важкість знаходження спортивних партнерів та обмеженість доступу до інформації про спортивні події. Усі ці

проблеми можна вирішити шляхом розробки мобільного застосунку для пошуку та приєднання до спортивних подій.

Аналіз предметної області показав, що на успішність залучення людей до зайняття спортом впливає безліч мотиваційних факторів. Тому, при розробці застосунку, важливо враховувати всі можливі мотиваційні фактори та застосовувати їх в підходах до користувачів.

Для того, щоб розроблений застосунок був ефективним, важливо також дослідити існуючі технології та розробки в сфері мобільних застосунків для спортивних заходів. Це дозволить забезпечити відповідний функціонал застосунку, який відповідатиме вимогам користувачів, а також поширити його серед цільової аудиторії.

Важливість цього проєкту полягає в можливості створення активної спортивної спільноти, яка буде взаємодіяти через мобільний застосунок. Це може сприяти поліпшенню здоров'я та життєвого стилю користувачів, а також популяризації спорту.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

2.1.1 Мова програмування Swift

Для розробки застосунку було обрано мову програмування Swift. Ця мова була розроблена компанією Apple спеціально для розробки програмного забезпечення для їхніх пристроїв, таких як iPhone, iPad та інших. Swift є сучасною мовою програмування, яка має високу продуктивність та надійність, що дозволяє розробникам створювати високоякісні застосунки.

Однією з ключових переваг Swift є його безпека, яка забезпечується за рахунок використання високої якості систем типів. Це дозволяє виявляти помилки на етапі компіляції, що унеможливлює їх появу під час роботи програми, що може призвести до некоректної роботи. Крім того, Swift має зручний синтаксис, який дозволяє розробникам створювати програми швидко та ефективно.

Ще однією важливою перевагою Swift є його інтеграція з платформою iOS. Swift використовує фреймворк UIKit, який є стандартом для розробки застосунків для iOS. Це дозволяє розробникам створювати застосунки, які максимально оптимізовані для роботи на пристроях Apple, з використанням усіх можливостей, що надає платформа iOS.

Також Swift має велику підтримку спільноти, що розробляє програмне забезпечення для iOS, тому для нього існує багато додаткових бібліотек та інструментів.

2.1.2 Фреймворк UIKit

UIKit - це фреймворк для розробки графічного інтерфейсу користувача на платформі iOS. UIKit забезпечує ряд класів, які дозволяють розробляти

інтерактивні застосунки зі складним інтерфейсом, включаючи таблиці, колекції, кнопки, текстові поля, переходи між екранами і багато іншого.

UIKit ідеально підходить для застосунку, оскільки він забезпечує широкі можливості для розробки зручного інтерфейсу, який допоможе користувачам легко зорієнтуватися у застосунку та здійснювати всі необхідні операції.

Наприклад, UITableView дозволяє показувати список спортивних подій, а UICollectionView дозволяє показувати їх у вигляді сітки. Для навігації між екранами можна використовувати UINavigationController або UITabBarController. Кнопки, текстові поля, сегментовані контроли та інші елементи інтерфейсу також присутні в UIKit і можуть бути використані для створення інтерактивного досвіду користувача.

Крім того, UIKit дозволяє створювати власні кастомні елементи інтерфейсу, які відповідають потребам конкретного проєкту. У комбінації з іншими технологіями, такими як Auto Layout, UIKit дозволяє розробляти інтерфейс, який працює на різних розмірах екранів та орієнтаціях.

2.1.3 IDE XCode

Xcode є офіційним інтегрованим середовищем розробки (IDE) для розробки програмного забезпечення під операційні системи macOS та iOS. Це безкоштовна програма, розроблена компанією Apple, яка надає розробникам можливість створювати програми для Mac, iPhone, iPad, Apple Watch та Apple TV.

Xcode має багато корисних функцій, які допомагають швидко та легко розробляти програми, такі як інструменти для створення інтерфейсу користувача, налагодження та профілювання коду, інструменти для побудови та збирання проєкту, підтримка систем контролю версій, автоматична перевірка помилок та багато іншого.

Оскільки Xcode є офіційним середовищем розробки для платформи iOS, воно містить всі необхідні інструменти для розробки застосунків на цій

платформі, включаючи підтримку мови програмування Swift та підтримку інтерфейсу користувача за допомогою UIKit.

2.1.4 Структура проєкту

При запуску застосунку користувач може зареєструватися або авторизуватися. Далі відображається Main Tab Screen із наступною схемою навігації, яка зображена на рисунку 2.1.

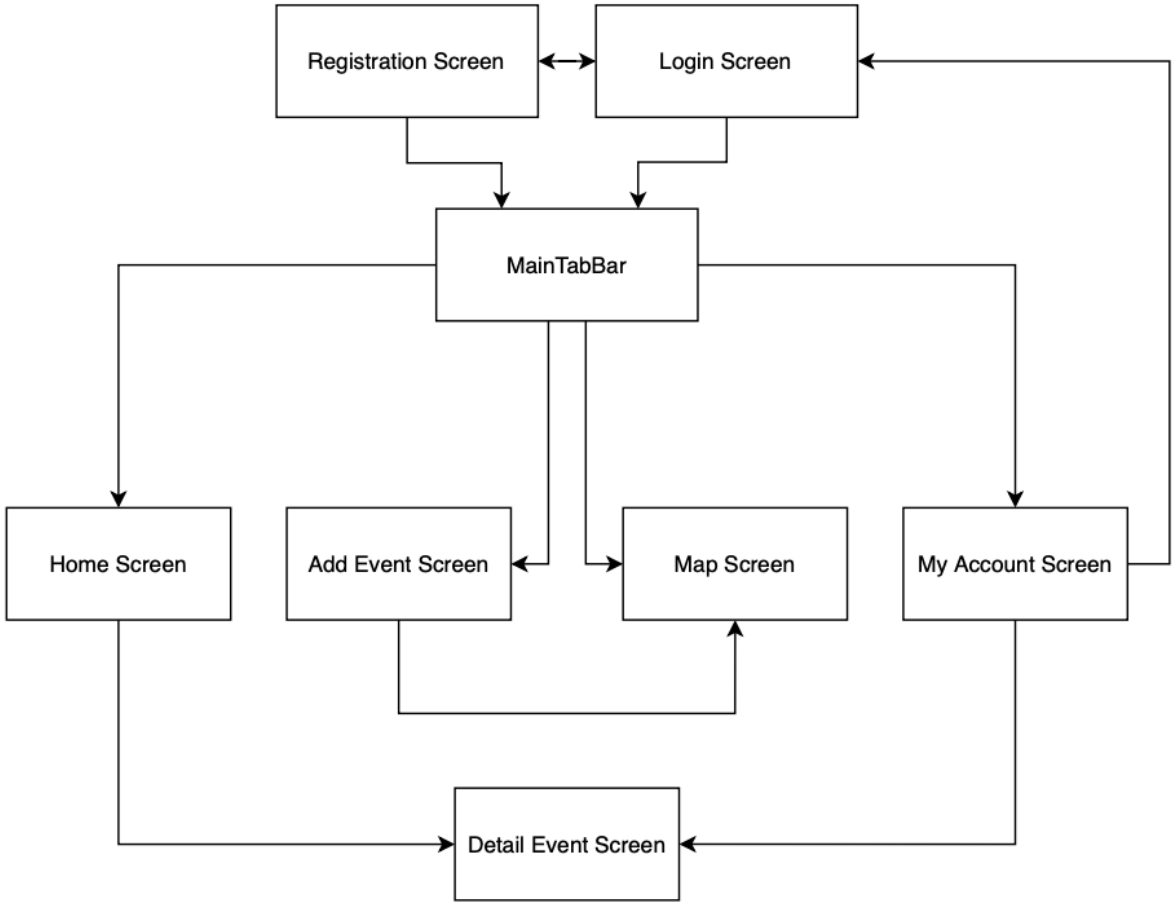


Рисунок 2.1 – Схема навігації екранів

Основні екрани застосунку:

– Main Tab Screen – екран, який містить в собі 4 основні екрани та слугує для зручної навігації між ними.

– Home Screen – екран, на якому відображається список усіх доданих подій.

– Add Event Screen – екран, на якому користувач може додати нову подію.

– Map Screen – екран, на якому відображається мапа із усіма подіями.

– MyAccount Screen – екран користувача із особистими даними та списком подій, які створив особисто він.

– Detail Event Screen – екран детальної інформації про подію.

– Login Screen – екран, на якому користувач може авторизуватися.

– Registration Screen – екран, на якому користувач може створити новий обліковий запис.

На рисунках 2.2 – 2.4 зображено бізнес процеси реєстрації, додавання та перегляд подій.

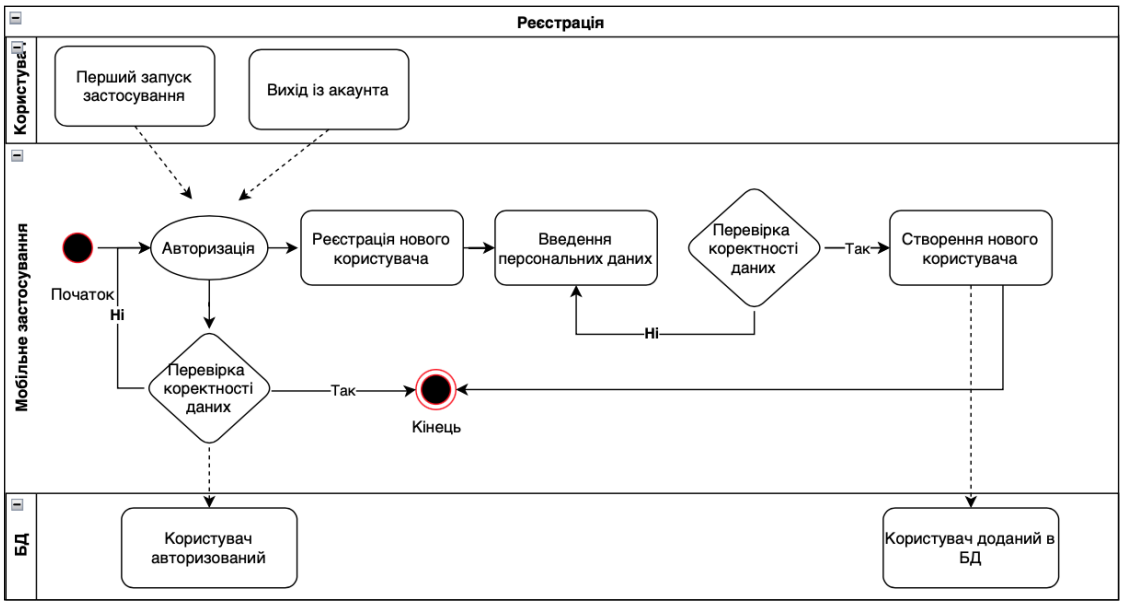


Рисунок 2.2 – Бізнес-процес реєстрації користувача

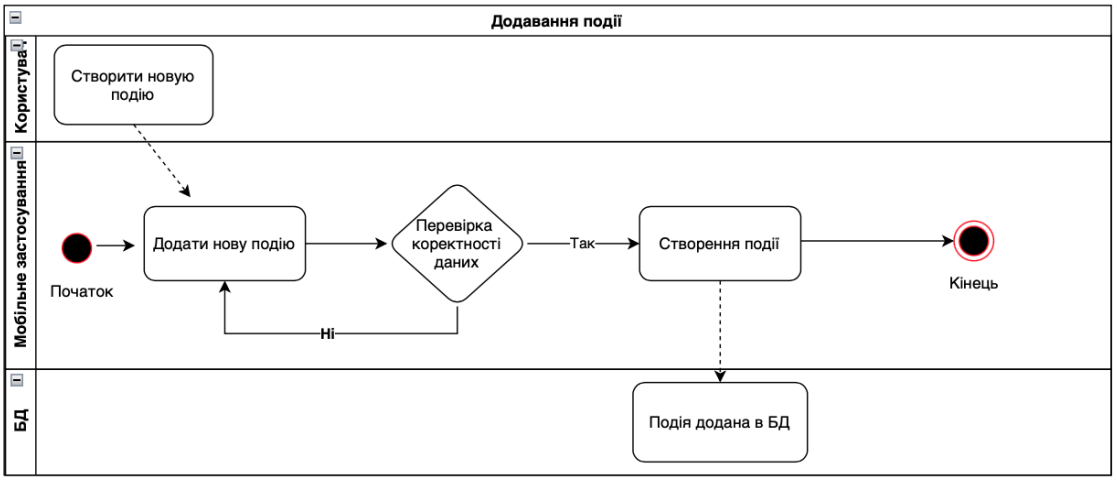


Рисунок 2.3 – Бізнес-процес додавання події

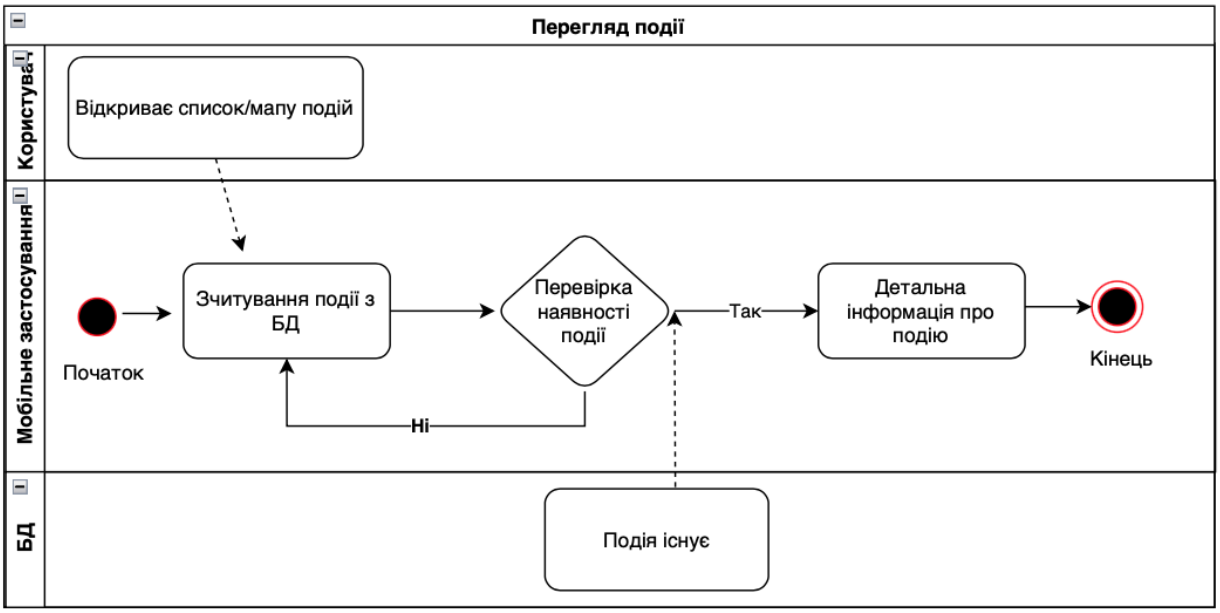


Рисунок 2.4 – Бізнес-процес перегляду подій

2.1.5 Діаграма класів

Кожен клас є важливим компонентом системи і має свою функціональність та роль. На рисунках 2.5 та 2.6 наведено діаграми кожного класу контролера окремо, розкриваючи його основні методи та зв'язки з іншими класами. У таблицях 2.1 – 2.7 описано детально головні класи-контролери застосунку.

Опис класів допоможе нам отримати глибше розуміння структури проєкту та розподілу функціональності між різними класами. Ми детально розглянемо кожен метод класу, його призначення і спосіб використання, що дозволить нам зрозуміти, як саме працює кожен клас і які завдання можуть бути виконані за допомогою цього класу.

Рисунок 2.5 – Діаграма класів контролерів Login, Registration, Details, AddEvent, Map

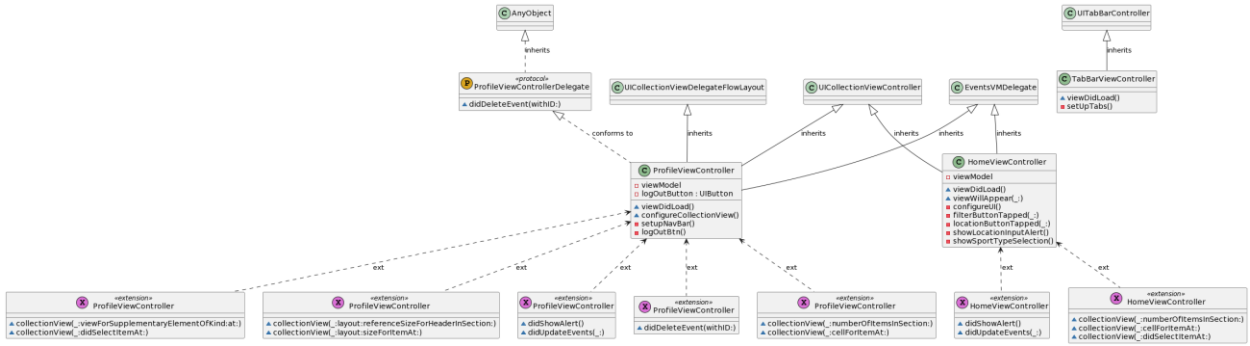


Рисунок 2.6 – Діаграма класів контролерів Home та Profile

Таблиця 2.1 – Опис класу RegistrationViewController

Назва класу	RegistrationViewController	
Опис класу	Клас, що представляє екран реєстрації нового користувача.	
Методи	**Назва методу**	**Опис методу**
	addProfilePhotoBtn()	Метод, який відображає вибір фото для профілю за допомогою imagePicker.

Продовження таблиці 2.1

	loginButtonTapped()	Метод, який закриває поточний екран реєстрації. Перехід до екрану авторизації.
	registrationButtonTapped()	Метод, який перевіряє наявність вибраного зображення профілю, а також введених даних. Залежно від результату реєстрації, відображає екран TabBarViewController або показує повідомлення про помилку.

Таблиця 2.2 – Опис класу LoginViewController

Назва класу	LoginViewController	
Опис класу	Клас, що представляє екран входу в систему користувача.	
Методи	**Назва методу**	**Опис методу**
	registrationButtonTapped()	Метод, який викликається при переході на екран реєстрації.
	loginButtonTapped()	Метод, який перевіряє, чи введені дані не є порожніми. Залежно від результату авторизації, відображає екран TabBarViewController або показує повідомлення про помилку.

Таблиця 2.3 – Опис класу DetailsEventViewController

Назва класу	DetailsEventViewController	
Опис класу	Клас контролера для перегляду деталей події.	
Методи	**Назва методу**	**Опис методу**
	openChatAction()	Метод, який відкриває вікно з вибором застосунку для відкриття чату з користувачем.
	deleteEventAction()	Метод, який виконує видалення події з бази даних.

Таблиця 2.4 – Опис класу ProfileViewController

Назва класу	ProfileViewController	
Опис класу	Клас, що відповідає за відображення профілю користувача у колекційному представленні.	
Методи	**Назва методу**	**Опис методу**
	logOutBtn()	У цьому методі відбувається вихід з облікового запису користувача.

Таблиця 2.5 – Опис класу MapViewController

Назва класу	MapViewController	
Опис класу	Клас MapViewController відповідає за керування екраном карти.	
Методи	**Назва методу**	**Опис методу**
	doneButtonPresed()	Метод, який викликається при натисканні кнопки "Готово". Повідомляє делегата про вибраний адресу і закриває контролер.
	centerViewInUserLocation()	Метод, який відображає місцезнаходження користувача на карті.
	closeButtonPresed()	Метод, який закриває контролер.
	addAnnotations()	Приватний метод, який створює анотації для кожної події і додає їх на карту.
	fetchEvents()	Приватний метод, який отримує список подій з бази даних і зберігає їх в масив events.

Таблиця 2.6 – Опис класу AddEventViewController

Назва класу	AddEventViewController	
Опис класу	Клас AddEventViewController відповідає за керування екраном додавання події.	
Методи	**Назва методу**	**Опис методу**
	locationButtonTapped()	Приватний метод, який викликається при натисканні кнопки місцезнаходження. Показує контролер карт (MapViewController) для вибору місця.
	saveButtonTapped()	Приватний метод, який викликається при натисканні кнопки "Готово". Перевіряє заповненість полів "Назва події" і "Місцезнаходження" і зберігає нову подію в базу даних.

Таблиця 2.7 – Опис класу HomeViewController

Назва класу	HomeViewController	
Опис класу	Клас, що управляє головним екраном застосунку, який відображає колекцію подій.	
Методи	**Назва методу**	**Опис методу**
	filterButtonTapped()	Приватний метод, який відображає спливаюче вікно для введення міста.

Продовження таблиці 2.7

	showSportTypeSelection()	Приватний метод, який відображає спливаюче вікно для вибору типу спорту.
	locationButtonTapped()	Приватний метод, який відображає спливаюче вікно для введення міста.

2.2 Архітектура програмного забезпечення

Архітектура застосунку базується на патерні MVVM. Цей патерн дозволяє розділити логіку застосунку на три компоненти: модель (Model), представлення (View) та в'юмодель (ViewModel).

– Model – це компонент, який відповідає за зберігання та обробку даних. Він містить бізнес-логіку програми та забезпечує зв'язок між даними та іншими компонентами.

– View – це компонент, який відповідає за відображення даних та інтерфейс користувача. Він взаємодіє з користувачем та відображає інформацію, яку надає ViewModel.

– ViewModel – це компонент, який відповідає за обробку даних та взаємодію з View та Model. Він реалізує бізнес-логіку програми та забезпечує комунікацію між View та Model.

При використанні патерну MVVM ми отримуємо декілька переваг, зокрема:

– Розділення логіки застосунку на компоненти зробить код більш організованим та зрозумілим.

– Компоненти можуть бути протестовані окремо, що забезпечує більшу стабільність застосунку.

– Для зміни відображення даних не потрібно змінювати логіку застосунку, що знижує ризик появи помилок.

Нижче на рисунку 2.7 наведено схему архітектури програмного забезпечення на основі патерну MVVM:

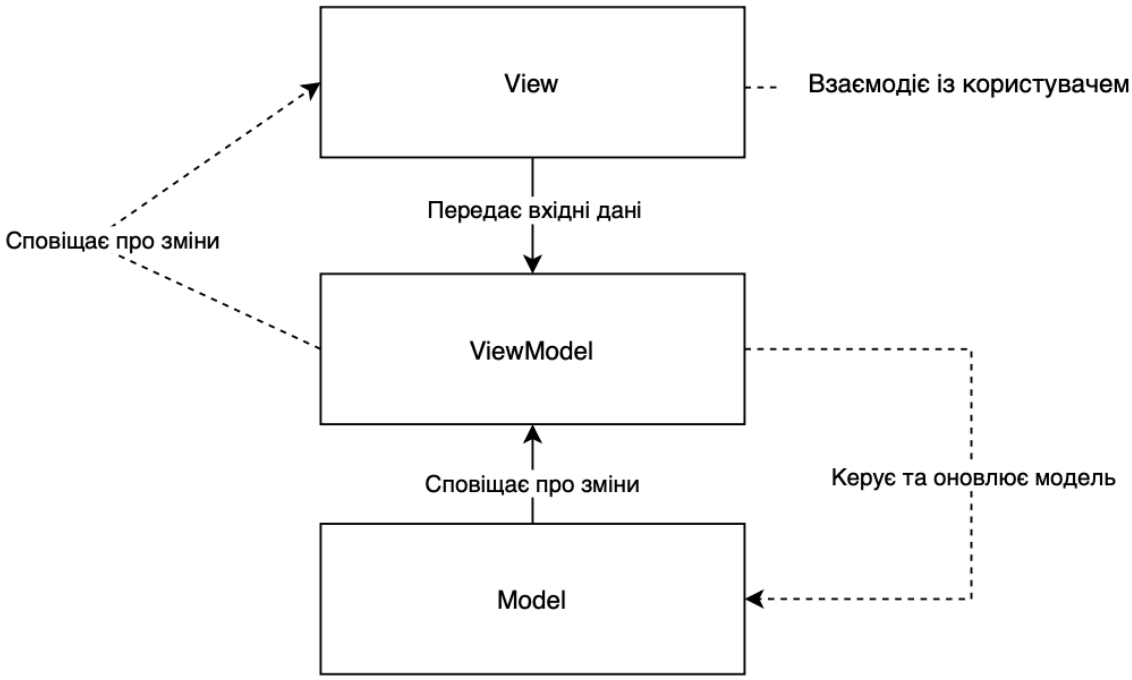


Рисунок 2.7 – Схема архітектури MVVM

Основною перевагою використання MVVM є те, що він дозволяє забезпечити чистоту коду та зменшити залежність між компонентами. Це дозволяє збільшити стабільність програми та зробити її більш масштабованою. Також цей підхід сприяє швидкому виконанню роботи, оскільки компоненти рівноправні та мають визначену роль в архітектурі.

Файлова структура проєкту (рисунок 2.8) була організована з використанням директорій для кращої структури та організації коду. Кожна директорія має свою функціональну роль і містить відповідні файли та компоненти проєкту.

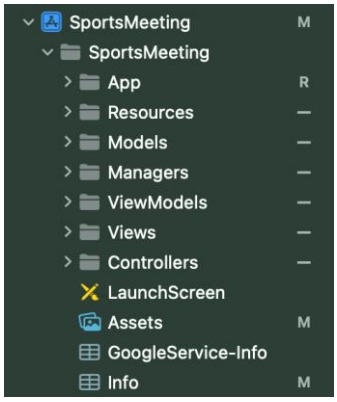


Рисунок 2.8 – Файлова структура

Перелік директорій файлової структури наведено нижче:

– App:

Директорія містить два файлу AppDelegate.swift та SceneDelegate.swift, які відповідають за налаштування та управління життєвим циклом застосунку відповідно. AppDelegate.swift – це точка входу у застосунок, де виконуються дії, необхідні для ініціалізації застосунку при запуску, а SceneDelegate.swift - відповідає за створення та конфігурацію сцен, які відображають застосунок на екрані.

– Controllers:

Директорія містить різні контролери, які відповідають за відображення та обробку даних на сторінках застосунку.

– Managers:

У цій директорії містяться менеджери, які відповідають за роботу з різними частинами застосунку, а саме робота з авторизацією, базою даних та картою. Вона містить три менеджери: AuthManager.swift, DatabaseManager.swift, MapManager.swift.

– Models:

Директорія зберігає моделі даних, які використовуються у застосунку.

– Resources:

Директорія містить корисні ресурси, такі як розширення та утиліти.

– ViewModels:

У цій директорії зберігаються ViewModel, які відповідають за логіку та обробку даних на сторінках застосунку.

– Views:

Директорія містить файли з програмним кодом сторінки застосунку, які відповідають за елементи інтерфейсу.

2.3 Конструювання програмного забезпечення

Щоб забезпечити збереження даних і автентифікації користувачів було використано Firebase. Firebase – це мобільна і веб-платформа розробки

застосунків, яка надає інструменти для створення якісних застосунків, забезпечуючи хмарне зберігання даних, аналітику, автентифікацію і багато іншого. В проєкті використовується три наступні інструменти: Firebase Authentication, Firebase Storage, Firebase Realtime Database.

Firebase Authentication дозволяє нам легко і безпечно автентифікувати користувачів за допомогою різних методів, таких як електронна пошта і пароль, Google, Facebook та інші. Це зменшує ризик порушення безпеки та спрощує процес входу у застосунок.

Firebase Storage надає безпечне зберігання файлів, таких як фотографії користувачів або медіа-файли, забезпечуючи доступ до них з будь-якої точки світу за допомогою URL-адреси. Це дозволяє зберігати великі обсяги даних без необхідності власного сервера.

Firebase Realtime Database - це хмарна база даних, яка забезпечує можливість зберігання та синхронізації даних в режимі реального часу між декількома користувачами. Для зберігання даних у Firebase Realtime Database використовується деревоподібна структура. Кожен вузол в дереві має свій унікальний ключ, а значення може бути будь-яким типом даних, таким як рядок, число, об'єкт або масив. На рисунку 2.9 можемо побачити, як виглядає архітектура із використанням вище перерахованими сервісами Firebase.

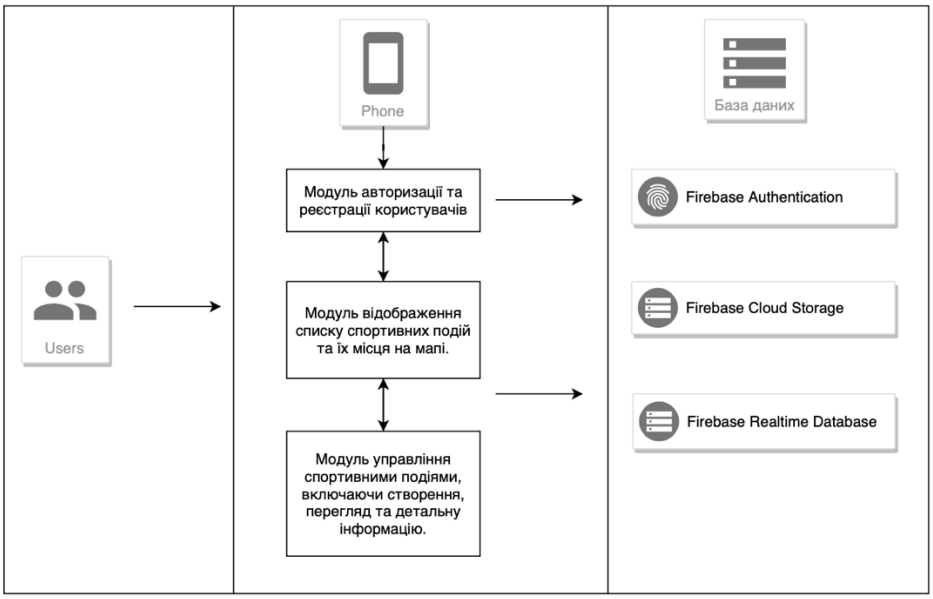


Рисунок 2.9 – Архітектура застосунку

Однією з головних переваг Firebase Realtime Database є можливість синхронізації даних в режимі реального часу. Це означає, що якщо дані змінюються на одному пристрої, вони автоматично оновляться на всіх інших пристроях, які мають доступ до цих даних. Це дуже корисна функція для мобільних застосунків, де користувачі можуть взаємодіяти з застосунком з різних пристроїв. Дані зберігаються в розподіленій системі з резервним копіюванням, що дозволяє забезпечувати високий рівень доступності та надійності.

Firebase Realtime Database має ряд API, які дозволяють отримувати, додавати, оновлювати та видаляти дані. Основні методи API:

– setValue: дозволяє зберегти дані у вузол дерева

– getValue: дозволяє отримати дані з вузла дерева

– updateChildValues: дозволяє оновити значення в одному або декількох вузлах дерева

– removeValue: дозволяє видалити вузол дерева

Firebase Realtime Database дозволяє створювати реляційні зв'язки між даними. Для цього використовується підхід, який називається денормалізацією даних. Денормалізація - це процес дублювання даних для забезпечення швидкого доступу до них.

Наприклад, якщо в базі даних є дві колекції: "users" та "events", можна створити зв'язок між ними, додавши до кожного елемента колекції "events" поле "userId", в якому зберігатиметься ідентифікатор користувача. Таким чином, можна легко знайти всі події, пов'язані з певним користувачем. На рисунку 2.10 зображено схему таблиць Firebase Realtime Database, а також зв’язки між ними.

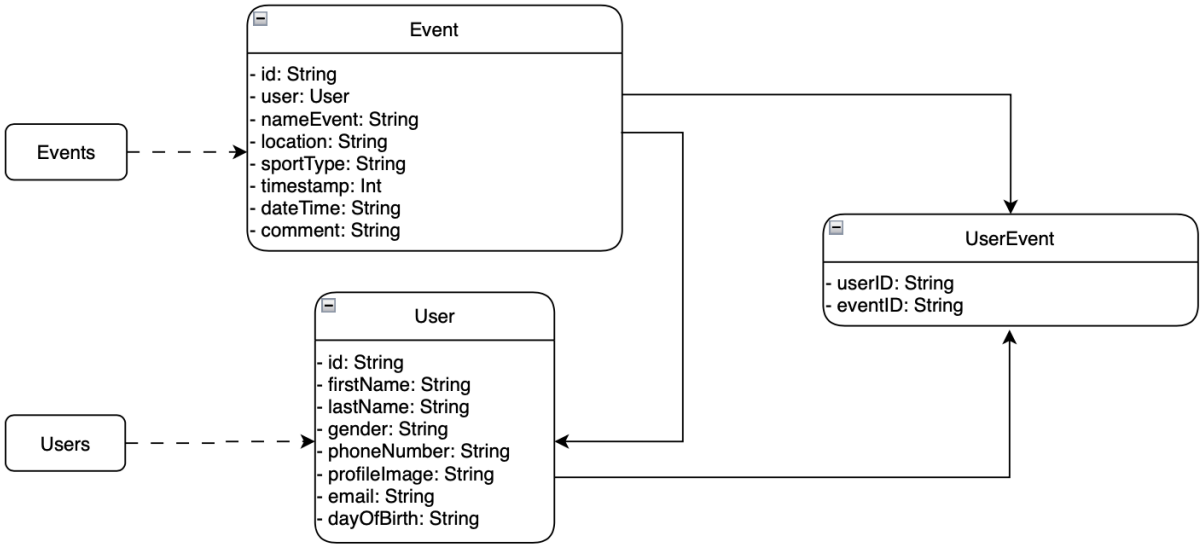


Рисунок 2.10 – Схема БД

Опис відношень User, Event та UserEvent представлено у таблицях 2.8, 2.9 та 2.10 відповідно.

Таблиця 2.8 – Опис таблиці User

Таблиця	Назва поля	Тип даних	Опис
User	id	string	унікальний ідентифікатор користувача
	firstName	string	ім'я користувача
	lastName	string	прізвище користувача
	gender	string	стать користувача
	phoneNumber	string	номер телефону користувача
	profileImage	string	посилання на профільне зображення користувача
	email	string	електронна пошта користувача
	dayOfBirth	string	дата народження користувача

Таблиця 2.9 – Опис таблиці Event

Таблиця	Назва поля	Тип даних	Опис
Event	id	string	унікальний ідентифікатор події
	nameEvent	string	назва події
	location	string	місце проведення події
	sporType	string	тип спорту або активності, пов'язаної з подією
	timestamp	int	дата та час публікації події
	dateTime	string	час проведення події
	comment	string	додатковий коментар, що стосується події

Таблиця 2.10 – Опис таблиці User

Таблиця	Назва поля	Тип даних	Опис
UserEvent	userID	string	унікальний ідентифікатор користувача
	eventID	string	унікальний ідентифікатор події

Таблиця UserEvent містить зв'язок між користувачем та подією, яку він створив. Таким чином, зв'язок між ними один до багатьох. Тобто один користувач може створити багато події, які будуть мати ідентифікатор цього користувача.

2.4 Аналіз безпеки даних

Безпека даних є однією з найважливіших складових проєкту, який має забезпечити надійність та захист інформації про користувачів. В даному застосунку Firebase Auth забезпечує безпеку авторизації та автентифікації користувачів, а Firebase Database Realtime забезпечує зберігання даних у реальному часі. Обидва ці інструменти забезпечують високий рівень захисту даних, використовуючи шифрування персональних даних.

Крім того, Swift має вбудовані функції захисту даних, що робить його надійним та безпечним інструментом для розробки застосунків.

Висновки до розділу

У розділі було розглянуто вибір мови програмування для розробки застосунку на платформі iOS. Також було детально описано архітектуру на основі патерну MVVM, що дозволяє розділити логіку на компоненти та забезпечити більшу організованість та зрозумілість коду.

Додатково, для збереження даних та автентифікації користувачів було використано Firebase, що забезпечує хмарне зберігання даних. Окрім цього, було проведено аналіз безпеки даних та прийнято заходи для забезпечення їхнього належного зберігання.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Тестування є важливим етапом у розробці застосунку, оскільки це дозволяє оцінити ефективність та надійність програми з точки зору її функціональності, продуктивності, безпеки, масштабованості та інших аспектів.

Основні метрики, які були протестовані:

– Функціональність:

В рамках аналізу функціональності було проведено перевірку виконання всіх основних функцій застосунку, включаючи створення профілю користувача, створення подій, пошук їх в списку та на мапі, а також перехід до чату. Результати перевірки підтвердили, що функціональність застосунку відповідає вимогам та користувачам надається повний функціонал для зручного використання.

– Ефективність:

Проведено вимірювання швидкодії та продуктивності застосунку, включаючи час відгуку, завантаження ресурсів та час виконання операцій. Результати показали, що застосунок працює ефективно навіть при значному обсязі даних та великому потоку користувачів.

– Безпека:

Були ідентифіковані потенційні вразливості застосунку та проведена перевірка використання безпечних протоколів та шифрування даних. Застосовані заходи забезпечують конфіденційність та цілісність інформації користувачів.

– Масштабованість:

Застосунок протестовано на масштабованість, щоб переконатися, що він здатний ефективно працювати при збільшенні обсягу оброблюваних даних та

кількості користувачів. Результати показали, що застосунок легко масштабується та може витримувати значні навантаження.

– Підтримка та управління:

Було оцінено рівень підтримки та управління застосунком, включаючи зручність розгортання, налаштування та оновлення, а також доступність документації та підтримки користувачів. Також для проведення тестування застосунку були використані різні моделі та розміри смартфонів iPhone. Тестування проводилося з метою перевірки сумісності застосунку з різними пристроями.

1. iPhone SE (3nd generation):

Розмір екрану: 4.7 дюйма

Результат тестування можна побачити на рисунку 3.1. Застосунок успішно пройшов тестування на iPhone SE (3nd generation) і коректно працює на даному пристрої. Відображення інтерфейсу, функціонал та загальна продуктивність застосунку відповідають очікуванням.

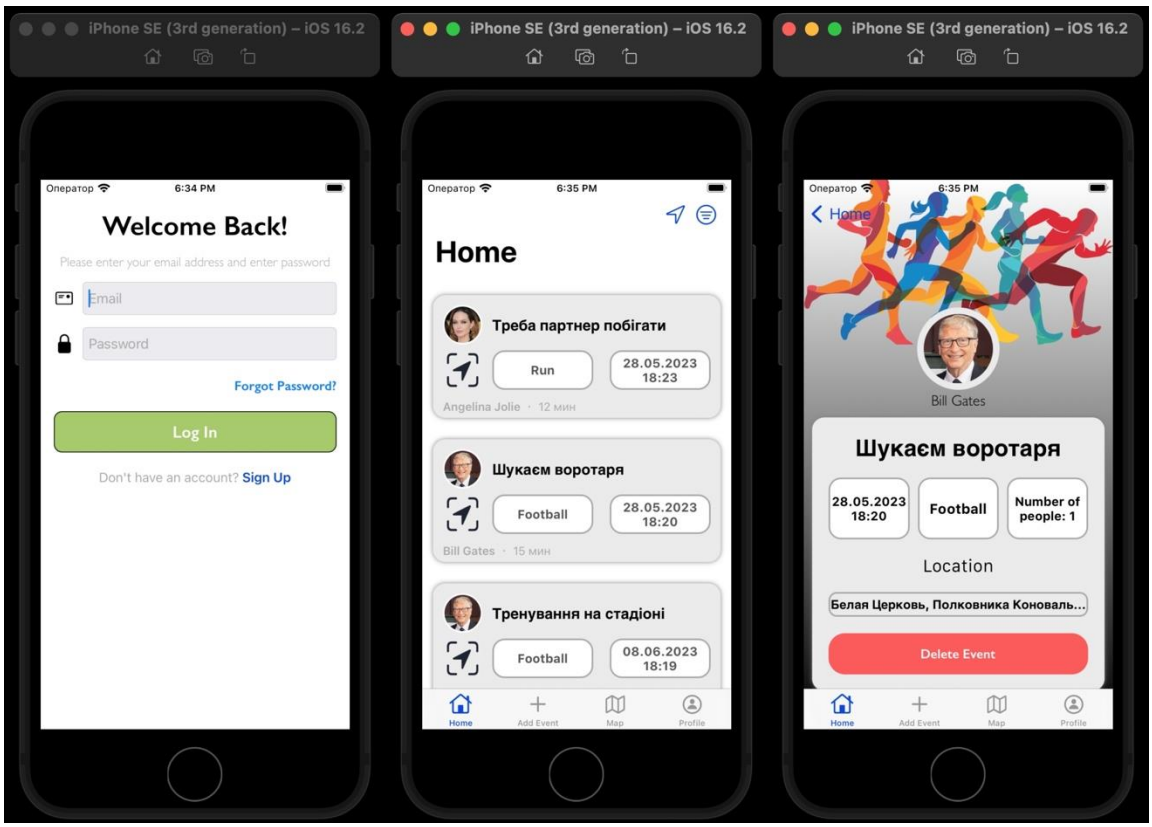


Рисунок 3.1 – Результат тестування на пристрої iPhone SE

2. iPhone XR:

Розмір екрану: 6.1 дюйма

Результат тестування можна побачити на рисунку 3.2. Застосунок був успішно протестований на iPhone XR. Його робота була перевірена з точки зору відображення інтерфейсу, функціоналу та загальної продуктивності. Всі важливі функції застосунку працюють належним чином на цьому пристрої.

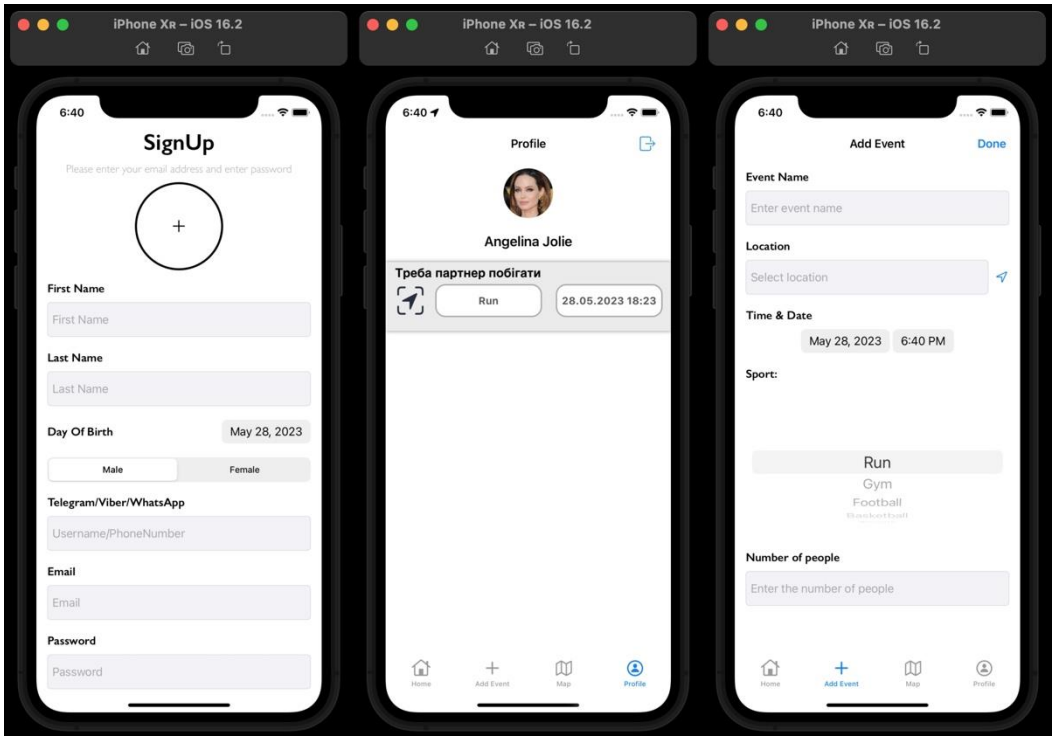


Рисунок 3.2 – Результат тестування на пристрої iPhone XR

3. iPhone 14 Pro Max:

Розмір екрану: 6.7 дюйма

Результат тестування можна побачити на рисунку 3.3. Застосунок був протестований на iPhone 14 Pro Max, найбільшому з доступних моделей. Всі аспекти застосунку, включаючи його інтерфейс, функціонал та продуктивність, були перевірені. Застосунок працює без проблем на цьому пристрої, і його можна комфортно використовувати на великому екрані iPhone 14 Pro Max.

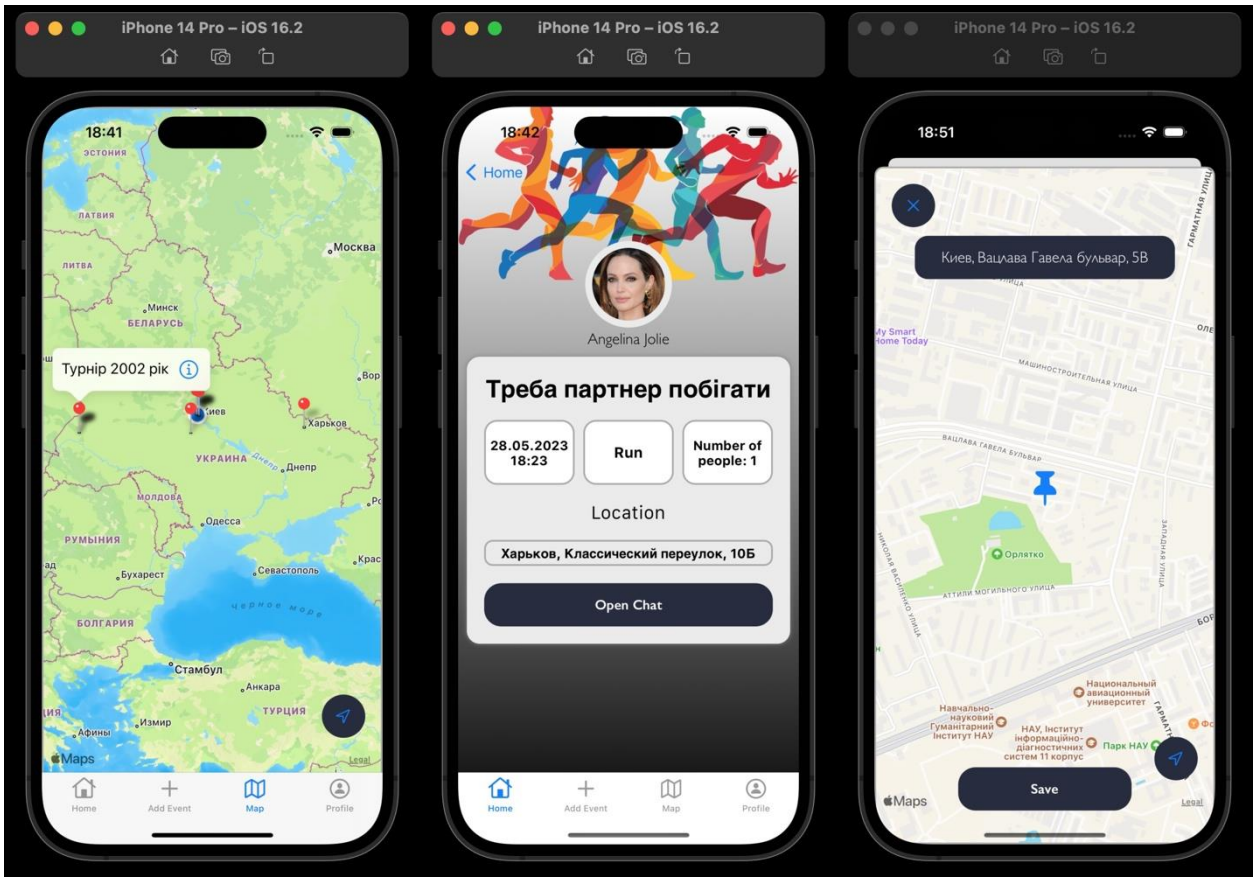


Рисунок 3.3 – Результат тестування на пристрої iPhone 14 Pro Max

3.2 Опис процесів тестування

Розглянемо основні сценарії тестування, які включають в себе набір тест-кейсів. Детальні описи тест-кейсів можна знайти в таблицях 3.1 – 3.5.

Таблиця 3.1 – Тест 1.1

Тест	Створення облікового запису
Модуль	Автентифікація та керування користувачами
Номер тесту	1.1
Початковий стан системи	Користувач відсутній
Вхідні данні	Персональні дані нового користувача (ім'я, прізвище, телефон, стать, зображення, електронна пошта, пароль)

Продовження таблиці 3.1

Опис проведення тесту	1. Користувач відкриває застосунок і переходить на екран реєстрації. 2. Користувач вводить обов'язкові поля: ім'я, електронну пошту та пароль. 3. Користувач натискає кнопку "Зареєструватися". 4. Система перевіряє валідність введених даних та наявність користувача з такою ж електронною поштою.
Очікуваний результат	Після успішного створення облікового запису, користувач отримує підтвердження про реєстрацію та може увійти до застосунку зі своїми обліковими даними.
Фактичний результат	Обліковий запис користувача успішно створений, і користувач може використовувати його для входу у застосунок.

Таблиця 3.2 – Тест 1.2

Тест	Створення спортивної події
Модуль	Управління спортивними подіями
Номер тесту	1.2
Початковий стан системи	Користувач авторизований
Вхідні данні	Дані про нову спортивну подію (назва, дата, час, місце)
Опис проведення тесту	1. Користувач переходить на екран створення спортивної події. 2. Користувач вводить дані про спортивну подію (назва, дата, час, місце). 3. Користувач натискає кнопку "Створити" для збереження спортивної події.

Продовження таблиці 3.2

Очікуваний результат	Після успішного створення спортивної події, вона додається до списку доступних подій і стає видимою для інших користувачів.
Фактичний результат	Спортивна подія успішно створена і відображається у списку доступних подій для інших користувачів.

Таблиця 3.3 – Тест 1.3

Тест	Перегляд списку спортивних подій
Модуль	Управління спортивними подіями
Номер тесту	1.3
Початковий стан системи	Користувач авторизований
Вхідні данні	Відсутні
Опис проведення тесту	1. Користувач відкриває екран списку спортивних подій. 2. Користувач переглядає список доступних спортивних подій або вибирає опцію перегляду на мапі.
Очікуваний результат	Користувач бачить список спортивних подій або відображення подій на мапі з відповідною інформацією про кожну подію.
Фактичний результат	Користувач успішно переглядає список спортивних подій або відображення на мапі з правильною інформацією про кожну подію.

Таблиця 3.4 – Тест 1.4

Тест	Перегляд детальної інформації події
Модуль	Користувач на екрані мапи або у списку спортивних подій
Номер тесту	1.4

Продовження таблиці 3.4

Початковий стан системи	Користувач авторизований
Вхідні данні	ID обраної спортивної події
Опис проведення тесту	1. Користувач вибирає спортивну подію, для якої він бажає переглянути детальну інформацію. 2. Користувач переходить до екрану деталей спортивної події.
Очікуваний результат	Користувач може переглянути повну інформацію про спортивну подію, таку як назва, опис, дата, місце проведення, тощо.
Фактичний результат	Користувач успішно переходить до екрану деталей спортивної події, де може переглянути всю необхідну інформацію про подію.

Таблиця 3.5 – Тест 1.5

Тест	Перехід до чату з іншим користувачем
Модуль	Управління спортивними подіями
Номер тесту	1.5
Початковий стан системи	Користувач авторизований
Вхідні данні	ID чату або інформація про іншого користувача
Опис проведення тесту	1. Користувач вибирає спортивну подію або користувача, з яким він бажає спілкуватись. 2. Користувач виконує перехід до чату з вибраним користувачем або спортивною подією.
Очікуваний результат	Користувач успішно переходить до чату з іншим користувачем або спортивною подією.

Продовження таблиці 3.5

Фактичний результат	Користувач успішно переходить до чату з вибраним користувачем або спортивною подією, де він може спілкуватися та обмінюватися повідомленнями.

Таблиця 3.6 – Тест 1.6

Тест	Видалення події в разі неактуальності
Модуль	Управління спортивними подіями
Номер тесту	1.6
Початковий стан системи	Користувач авторизований
Вхідні данні	Подія, яка вимагає видалення через неактуальність
Опис проведення тесту	1. Користувач обирає відповідну подію для видалення. 2. Користувач ініціює процес видалення.
Очікуваний результат	Подія успішно видаляється з системи.
Фактичний результат	Подія видалена.

Висновки до розділу

В результаті аналізу якості ПЗ було встановлено, що розроблений мобільний застосунок забезпечує зручний та надійний інструмент для користувачів, що дозволяє їм знаходити та приєднуватися до спортивних подій. Після проведення тестування застосунку на різних моделях та розмірах iPhone, можна зробити висновок, що застосунок є сумісним з цими пристроями та коректно працює на них. Він відображається на екранах різного розміру, забезпечуючи зручне користування та збереження функціональності. Таким чином, користувачі iPhone з різними моделями та розмірами екрану зможуть насолоджуватися застосунком без проблем.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Розгортання мобільного застосунку, розробленого на Swift у середовищі Xcode, включає наступні кроки:

– Налаштування середовища розробки:

1. Встановлення Xcode:

Завантажуємо та встановлюємо Xcode на MacBook (ПК із операційною системою Windows не підтримує Xcode). Xcode є основним інтегрованим середовищем розробки (IDE) для створення мобільних додатків для iOS.

2. Налаштування проєкту:

Імпортуємо проєкт із GitHub або, створивши новий проєкт в Xcode, додаємо код із застосунку.

– Тестування на симуляторі або реальному пристрої:

Використовуємо вбудовані можливості Xcode для тестування застосунку на симуляторі iOS або підключаємо реальний iPhone до комп'ютера та запускаємо застосунок на пристрої.

– Розгортання в App Store (необов’язково):

Для того, щоб розгорнути застосунок в App Store, необхідно підписатися на розробницьку програму Apple Developer Program (100 доларів на рік).

1. Підготовка застосунку:

Переконаємось, що ваш застосунок готовий до публікації, включаючи правильну версію, опис, знімки екрану тощо.

2. Завантаження застосунку:

Виконаємо вимоги Apple для публікації застосунку в App Store, включаючи створення ідентифікатора, налаштування App Store Connect, встановлення ціни (якщо необхідно), надання метаданих та знімків екрану.

3. Перевірка і схвалення:

Очікуємо перевірки і рецензії вашого застосунку Apple. Після схвалення застосунок стає доступним для завантаження і встановлення користувачами з App Store.

4.2 Підтримка програмного забезпечення

Після успішного розгортання застосунку та запуску його користувачами, необхідно забезпечити надійну та ефективну підтримку для забезпечення безперебійного функціонування та задоволення потреб користувачів.

Підтримка програмного забезпечення включає такі аспекти:

– Виявлення та усунення помилок: систематичний аналіз помилок та швидке реагування на них, виправляючи проблеми та надаючи оновлення з виправленнями.

– Оновлення та поліпшення: Крім виправлення помилок, підтримка програмного забезпечення також включає випуск оновлень, які містять новий функціонал, покращення та оптимізації.

– Користувацька підтримка: будуть створені канали комунікації, через які користувачі зможуть звертатись зі своїми питаннями, проблемами або пропозиціями. Також буде забезпечена швидка та якісна відповідь на запити користувачів, що допоможе вирішити їхні проблеми та забезпечити задоволення від використання застосунку.

– Технічна підтримка: буде забезпечено розгляд і вирішення технічних питань, пов'язаних з застосунком.

Загальна мета підтримки програмного забезпечення полягає у тому, щоб користувачі мали позитивний досвід використання застосунку, відчували його надійність та отримували своєчасну допомогу у разі потреби. Команда розробників буде старанно працювати над підтримкою застосунку, щоб забезпечити задоволення потреб користувачів та досягти успіху у розвитку спортивної спільноти.

Висновки до розділу

Розгортання та підтримка програмного забезпечення є важливим етапом, який дозволяє розробнику представити свій застосунок користувачам і забезпечити його доступність.

Застосунок може бути розгорнутий в App Store для загального доступу користувачів. Це забезпечує зручний спосіб поширення застосунку та дозволяє його знайти та встановити користувачам. Застосунок може легко оновлюватись через App Store, що дозволяє вносити поліпшення та новий функціонал для задоволення потреб користувачів. Це дозволить ефективно використовувати можливості мобільного застосунку та забезпечує доступність його функцій для широкого кола користувачів.

ВИСНОВКИ

В рамках даного дипломного проєкту був розроблений мобільний застосунок для організації та планування спільних спортивних подій. Предметна область включала аналіз спортивної спільноти, мотивації до зайняття спортом, використання мобільних застосунків для спортивних заходів та пошуку партнерів для спортивних занять.

Оцінюючи результати дослідження, можна зазначити, що розроблений мобільний застосунок успішно відповідає сучасному рівню наукових і технічних знань. Він надає користувачам зручні інструменти для створення та управління спортивними подіями, фільтрації та пошуку за інтересами, а також сприяє взаємодії та спілкуванню між спортсменами.

Ступінь впровадження розробленого застосунку може бути значним, оскільки мобільні застосунки для спортивних заходів є популярними серед широкої аудиторії. Результати роботи можуть бути використані в спортивних спільнотах, клубах, фітнес-центрах та інших організаціях, що пропонують спортивні події та послуги. Впровадження даного застосунку сприятиме поліпшенню організації спортивних заходів, залученню нових користувачів та підвищенню мотивації до зайняття спортом.

В результаті виконання дипломного проєкту всі поставлені задачі були вирішені. Мобільний застосунок був успішно розроблений та пройшов випробування, його функціональність була реалізована згідно з вимогами та цілями проєкту.

Застосування середовища розробки XCode, мови програмування Swift та Firebase бази даних було обґрунтовано зручним інтерфейсом та потужними можливостями для розробки мобільних додатків. Використання архітектурного шаблону MVVM дозволило забезпечити чітке розділення бізнес-логіки та відображення даних у застосунку.

Перевірка функціональності підтвердила, що застосунок відповідає всім основним функціям, включаючи створення профілю користувача, створення

подій, пошук їх в списку та на мапі. Це означає, що користувачам надається повний функціонал для зручного використання застосунку.

Оцінка рівня підтримки та управління застосунком включала аналіз зручності розгортання, налаштування та оновлення, а також доступність документації та підтримки користувачів. Використання різних моделей та розмірів iPhone під час тестування допомогло перевірити сумісність застосунку з різними пристроями, забезпечуючи його коректне відображення на екранах різних розмірів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Swift [Електронний ресурс]. – Режим доступу: https://developer.apple.com/swift/
2. UIKit [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/documentation/uikit
3. MapKit [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/documentation/mapkit
4. Xcode [Електронний ресурс] – Режим доступу: https://developer.apple.com/documentation/xcode
5. iOS Storyboards [Електронний ресурс] – Режим доступу до ресурсу: https://www.kodeco.com/5055364-ios-storyboards-getting-started
6. iOS MVVM Tutorial [Електронний ресурс] – Режим доступу: https://www.kodeco.com/6733535-ios-mvvm-tutorial-refactoring-from-mvc
7. Firebase Real-Time Database Tutorial for iOS [Електронний ресурс] – Режим доступу:https://www.kodeco.com/29394678-firebase-real-time-database-tutorial-for-ios
8. Get started with Cloud Firestore [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/firestore/quickstart
9. Authenticate Using Apple [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/auth/ios/apple
10. Firebase Security Rules [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/rules

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1015369542

Дата перевірки:
01.06.2023 15:08:59 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2023 15:52:17 EEST

ID користувача:
76913

Назва документа: **IT-94_Хоменко_ПЗ**

Кількість сторінок: **54** Кількість слів: **7288** Кількість символів: **61268** Розмір файлу: **4.33 MB** ID файлу: **1015035661**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

5.32%
Схожість

Найбільша схожість: **1.63%** з джерелом з Бібліотеки (ID файлу: **1000044221**)

4.06% Джерела з Інтернету	75	Сторінка 56
5.09% Джерела з Бібліотеки	197	Сторінка 58

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	2
Підозріле форматування	10 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ТА ПЛАНУВАННЯ СПІЛЬНИХ СПОРТИВНИХ ПОДІЙ

Текст програми

КПІ.ІТ-9428.045490.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Олександр ХОМЕНКО

Київ – 2023

Файл SceneDelegate.swift

```
import UIKit
import FirebaseAuth

class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions:
UIScene.ConnectionOptions) {
        guard let windowScene = (scene as? UIWindowScene) else { return }

        // Check if the user has been saved to Firebase
        if Auth.auth().currentUser != nil {
            // If yes, then open tabBarViewController
            let vc = TabBarViewController()
            let window = UIWindow(windowScene: windowScene)
            window.rootViewController = vc
            window.makeKeyAndVisible()
            self.window = window
        } else {
            // If not, then open the login screen
            let vc = LoginViewController()
            let window = UIWindow(windowScene: windowScene)
            window.rootViewController = vc
            window.makeKeyAndVisible()
            self.window = window
        }
    }
}
```

Файл UIButton.swift

```
import UIKit

extension UIButton {
    func sizeSymbol(name: String, size: CGFloat, weight: UIImage.SymbolWeight, scale:
UIImage.SymbolScale){
        let symbolConfiguration = UIImage.SymbolConfiguration(pointSize: size, weight: weight, scale:
scale)
        let buttonImage = UIImage(systemName: name, withConfiguration: symbolConfiguration)
        self.setImage(buttonImage, for: .normal)
    }
}
```

Файл UIColor.swift

```
import UIKit

extension UIColor {
    static let customDarkBlue = UIColor(named: "darkBlue")
    static let customRed = UIColor(named: "lightRed")
}
```

Файл UIImage.swift

```
import Foundation
import UIKit

extension UIImage {
    func darkened() -> UIImage? {
        UIGraphicsBeginImageContextWithOptions(size, false, 0)
        defer { UIGraphicsEndImageContext() }
```

```
        guard let ctx = UIGraphicsGetCurrentContext(), let cgImage = cgImage else {
            return nil
        }

        // flip the image, or result appears flipped
        ctx.scaleBy(x: 1.0, y: -1.0)
        ctx.translateBy(x: 0, y: -size.height)

        let rect = CGRect(origin: .zero, size: size)
        ctx.draw(cgImage, in: rect)
        UIColor(white: 0, alpha: 0.5).setFill()
        ctx.fill(rect)

        return UIGraphicsGetImageFromCurrentImageContext()
    }
}
```

Файл UIView.swift

```
import UIKit

extension UIView {
    func setDimensions(width: CGFloat, height: CGFloat) {
        translatesAutoresizingMaskIntoConstraints = false
        widthAnchor.constraint(equalToConstant: width).isActive = true
        heightAnchor.constraint(equalToConstant: height).isActive = true
    }

    // For insert layer in Foreground
    func addBlackGradientLayerInForeground(frame: CGRect, colors:[UIColor]){
        let gradient = CAGradientLayer()
        gradient.frame = frame
        gradient.colors = colors.map{$0.cgColor}
        self.layer.addSublayer(gradient)
    }
    // For insert layer in background
    func addBlackGradientLayerInBackground(frame: CGRect, colors:[UIColor]){
        let gradient = CAGradientLayer()
        gradient.frame = frame
        gradient.colors = colors.map{$0.cgColor}
        self.layer.insertSublayer(gradient, at: 0)
    }

    func shadowSetup(color: CGColor, opacity: Float, radius: Double) {
        layer.shadowColor = color
        layer.shadowOpacity = opacity
        layer.shadowOffset = CGSize.zero
        layer.shadowRadius = radius
    }
}
```

Файл Utilities.swift

```
import UIKit

final class Utilities {

    func containerView(withlabel text: String, view: UIView) -> UIStackView {
        let stack = UIStackView()
        stack.addArrangedSubview(titleLabel(with: text))
        stack.addArrangedSubview(view)
        stack.axis = .vertical
        stack.spacing = 10
```

```
        return stack
    }

    func imageContainerView(withImage image: UIView, textField: UIView) -> UIStackView {
        let stack = UIStackView()
        stack.addArrangedSubview(image)
        stack.addArrangedSubview(textField)
        stack.axis = .horizontal
        stack.spacing = 10
        return stack
    }

    private func titleLabel(with text: String) -> UILabel {
        let label = UILabel()
        label.textAlignment = .left
        label.text = text
        label.numberOfLines = 2
        label.font = UIFont(name: "Gill Sans SemiBold", size: 17)
        return label
    }

    func textField(withPlaceholder placeholder: String) -> UITextField {
        let textField = UITextField()
        textField.attributedPlaceholder = NSAttributedString(string: placeholder, attributes:
[NSAttributedString.Key.foregroundColor: UIColor.lightGray])
        textField.borderStyle = .roundedRect
        textField.autocorrectionType = .no
        textField.backgroundColor = .secondarySystemBackground
        return textField
    }

    func attributedButton(_ firstPart: String, _ secondPart: String) -> UIButton {
        let button = UIButton(type: .system)

        let attributedTitle = NSMutableAttributedString(string: firstPart, attributes:
[NSAttributedString.Key.font: UIFont.systemFont(ofSize: 16), NSAttributedString.Key.foregroundColor:
UIColor.lightGray])

        attributedTitle.append(NSAttributedString(string: secondPart, attributes:
[NSAttributedString.Key.font: UIFont.boldSystemFont(ofSize: 16),
NSAttributedString.Key.foregroundColor: UIColor.systemBlue]))

        button.setAttributedTitle(attributedTitle, for: .normal)

        return button
    }
}
```

Файл User.swift

```
import Foundation

struct User {
    let uid: String
    var firstName: String
    var lastName: String
    var dayOfBirth: String
    var gender: String
    var phoneNumber:String
    let email: String
    var profileImageUrl: URL?
}
```

```
extension User {
    var fullName: String {
        firstName + " " + lastName
    }
}
```

Файл Event.swift

```
import Foundation

struct Event {
    let nameEvent: String
    let eventID: String
    var location: String
    var dateTime: String
    let sportType: String
    var timestamp: Date!
    let user: User
    var comment: String

    init(user: User, eventID: String, dictionary: [String: Any]) {
        self.eventID = eventID
        self.user = user

        self.nameEvent = dictionary["nameEvent"] as? String ?? ""
        self.location = dictionary["location"] as? String ?? ""
        self.sportType = dictionary["sportType"] as? String ?? ""
        self.comment = dictionary["comment"] as? String ?? ""
        self.dateTime = dictionary["dateTime"] as? String ?? ""

        if let timestamp = dictionary["timestamp"] as? Double {
            self.timestamp = Date(timeIntervalSince1970: timestamp)
        }
    }
}
```

Файл DatabaseManager.swift

```
import Firebase

struct DatabaseManager {

    static let shared = DatabaseManager()


    func fetchEvents(completion: @escaping([Event]) -> Void) {
        var events = [Event]()

        database.child("events").observe(.childAdded) { snapshot in
            let eventID = snapshot.key

            self.fetchEvent(withEventID: eventID) { event in
                events.append(event)
                completion(events.sorted(by: { $0.timestamp > $1.timestamp }))
            }
        }
    }

    func fetchEvents(forUser user: User, completion: @escaping([Event]) -> Void) {
        var events = [Event]()
        database.child("user-events").child(user.uid).observe(.childAdded) { snapshot in
            let eventID = snapshot.key

            fetchEvent(withEventID: eventID) { event in
                events.append(event)
```

```
                completion(events.sorted(by: { $0.timestamp > $1.timestamp }))
            }
        }
    }

    func fetchEvent(withEventID eventID: String, completion: @escaping(Event) -> Void) {
        database.child("events").child(eventID).observeSingleEvent(of: .value) { snapshot in
            guard let dictionary = snapshot.value as? [String: Any] else { return }
            guard let uid = dictionary["uid"] as? String else { return }

            fetchUser(uid: uid) { user in
                let event = Event(user: user, eventID: eventID, dictionary: dictionary)
                completion(event)
            }
        }
    }

    public func deleteEvent(eventID: String, completion: @escaping (Bool) -> Void) {
        // Remove the event from the "events" node
        database.child("events").child(eventID).removeValue { error, _ in
            if error == nil {
                // Remove the event reference from the "user-events" node for all users
                database.child("user-events").observeSingleEvent(of: .value) { snapshot in
                    guard let userEventsDict = snapshot.value as? [String: AnyObject] else {
                        completion(false)
                        return
                    }

                    for (userID, userEvents) in userEventsDict {
                        if let userEventDict = userEvents as? [String: Int], userEventDict[eventID] != nil {
                            database.child("user-events").child(userID).child(eventID).removeValue()
                        }
                    }

                    completion(true)
                }
            } else {
                completion(false)
            }
        }
    }
}
```

Файл EventViewModel.swift

```
import UIKit

struct EventViewModel {
    // MARK: - Properties

    let event: Event
    let user: User

    var profileImageUrl: URL? {
        return event.user.profileImageUrl
    }

    var timestamp: String {
        let formatter = DateComponentsFormatter()
        formatter.allowedUnits = [.second, .minute, .hour, .day, .weekOfMonth]
        formatter.maximumUnitCount = 1
        formatter.unitsStyle = .abbreviated
        let now = Date()
```

```
        return formatter.string(from: event.timestamp, to: now) ?? "2m"
    }

    var userInfoText: NSAttributedString {
        let title = NSMutableAttributedString(string: user.fullName,
                                              attributes: [.font: UIFont.boldSystemFont(ofSize: 14)])
        title.append(NSAttributedString(string: " · \(timestamp)",
                                        attributes: [.font: UIFont.systemFont(ofSize: 14),
                                                     .foregroundColor: UIColor.lightGray]))
        return title
    }

    init(event: Event) {
        self.event = event
        self.user = event.user
    }
}
```

Файл HomeViewModel.swift

```
import ProgressHUD

protocol EventsVMDelegate: AnyObject {
    func didUpdateEvents(_ events: [Event])
    func didShowAlert()
}

class HomeViewModel {
    weak var delegate: EventsVMDelegate?

    var allEvents = [Event]()

    var events: [Event] = [] {
        didSet {
            DispatchQueue.main.async {
                self.delegate?.didUpdateEvents(self.events)
                if self.events.isEmpty {
                    self.events = self.allEvents
                    self.delegate?.didShowAlert()
                }
            }
        }
    }

    func fetchEvents() {
        DatabaseManager.shared.fetchEvents { events in
            self.events = events.sorted(by: { $0.timestamp > $1.timestamp })
            self.allEvents = events
        }
    }
}

extension HomeViewModel {
    func filterEventsBySportType(_ sportType: String?) {
        if let type = sportType, type != "All" {

            events = allEvents.filter { $0.sportType == type }
        } else {
            fetchEvents()
        }
    }

    func filterEventsByLocation(_ location: String?) {
        if let city = location?.trimmingCharacters(in: .whitespacesAndNewlines), !city.isEmpty {
```

```
            let filteredEvents = allEvents.filter { $0.location.range(of: city, options: .caseInsensitive) != nil }
            self.events = filteredEvents.isEmpty ? [] : filteredEvents
        } else {
            fetchEvents()
        }
    }
}
```

Файл ProfileViewModel.swift

```
import FirebaseAuth

class ProfileViewModel {
    weak var delegate: EventsVMDelegate?

    var user: User? {
        didSet {
            self.fetchEvents()
        }
    }

    var events: [Event] = [] {
        didSet {
            DispatchQueue.main.async {
                self.delegate?.didUpdateEvents(self.events)
            }
        }
    }

    func deleteEvent(withID eventID: String) {
        guard let index = events.firstIndex(where: { $0.eventID == eventID }) else {
            return
        }
        events.remove(at: index)
    }

    func fetchEvents() {
        guard let user = user else { return }
        DatabaseManager.shared.fetchEvents(forUser: user) { events in
            self.events = events
        }
    }

    func loadData() {
        guard let uid = Auth.auth().currentUser?.uid else {
            print("User not logged in")
            return
        }
        DatabaseManager.shared.fetchUser(uid: uid) { user in
            self.user = user

            if self.events.isEmpty {
                self.delegate?.didUpdateEvents(self.events)
            }
        }
    }
}
```

Файл LoginView.swift

```
import UIKit

class LoginView: UIView {

    // MARK: Outlets
```

```
// Labels

private let welcomeLabel: UILabel = {
    let label = UILabel()
    label.text = "Welcome Back!"
    label.font = UIFont(name: "Gill Sans SemiBold", size: 33)
    label.numberOfLines = 0
    label.textAlignment = .center
    return label
}()

private let adviceLabel: UILabel = {
    let label = UILabel()
    label.text = "Please enter your email address and enter password"
    label.font = UIFont(name: "Gill Sans Light", size: 16)
    label.numberOfLines = 2
    label.textAlignment = .center
    label.textColor = .lightGray
    return label
}()

// Buttons
private lazy var forgotPassButton: UIButton = {
    let button = UIButton()
    button.setTitle("Forgot Password?", for: .normal)
    //button.addTarget(self, action: #selector(), for: .touchUpInside)
    button.setTitleColor(.link, for: .normal)
    button.titleLabel?.font = UIFont(name: "Gill Sans SemiBold", size: 16)
    button.contentHorizontalAlignment = .right

    return button
}()

private(set) var loginButton: UIButton = {
    let button = UIButton()
    button.setTitle("Log In", for: .normal)
    button.backgroundColor = UIColor(named: "lightGreen")
    button.titleLabel?.font = UIFont(name: "Gill Sans SemiBold", size: 20)
    button.layer.cornerRadius = 10
    button.layer.borderWidth = 1
    button.layer.borderColor = UIColor.black.cgColor
    return button
}()

private(set) var registrationButton: UIButton = {
    let button = Utilities().attributedButton("Don't have an account?", " Sign Up")
    return button
}()

// Containers
private lazy var emailContainer: UIStackView = {
    let imageView = UIImageView()
    imageView.tintColor = .black
    imageView.contentMode = .scaleAspectFit
    imageView.image = UIImage(systemName: "mail")
    imageView.setDimensions(width: 25, height: 40)
    let stack = Utilities().imageContainerView(withImage: imageView, textField: emailTextField)
    return stack
}()

private lazy var passwordContainer: UIStackView = {
    let imageView = UIImageView()
    imageView.tintColor = .black
```

```
        imageView.contentMode = .scaleAspectFit
        imageView.image = UIImage(systemName: "lock.fill")
        imageView.setDimensions(width: 25, height: 40)
        let stack = Utilities().imageContainerView(withImage: imageView, textField: passwordTextField)
        return stack
    }()

    // Text Field
    private(set) var emailTextField: UITextField = {
        let textField = Utilities().textField(withPlaceholder: "Email")
        textField.autocapitalizationType = .none
        textField.returnKeyType = .continue
        return textField
    }()

    private(set) var passwordTextField: UITextField = {
        let textField = Utilities().textField(withPlaceholder: "Password")
        textField.isSecureTextEntry = true
        textField.autocapitalizationType = .none
        textField.returnKeyType = .done
        return textField
    }()

    // MARK: - Initializers
    override init(frame: CGRect) {
        super.init(frame: frame)
        setupUI()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // MARK: Setup UI
    private func setupUI() {
        let stackMain = UIStackView(arrangedSubviews: [welcomeLabel, adviceLabel, emailContainer,
passwordContainer, forgotPassButton, loginButton,  registrationButton])
        stackMain.translatesAutoresizingMaskIntoConstraints = false
        stackMain.axis = .vertical
        stackMain.spacing = 15

        addSubview(stackMain)
        NSLayoutConstraint.activate([
            stackMain.leadingAnchor.constraint(equalTo: safeAreaLayoutGuide.leadingAnchor, constant: 16),
            stackMain.trailingAnchor.constraint(equalTo: safeAreaLayoutGuide.trailingAnchor, constant: -16),
            stackMain.topAnchor.constraint(equalTo: safeAreaLayoutGuide.topAnchor, constant: 16)
        ])

        NSLayoutConstraint.activate([
            emailTextField.heightAnchor.constraint(equalToConstant: 40),
            passwordTextField.heightAnchor.constraint(equalToConstant: 40),
            loginButton.heightAnchor.constraint(equalToConstant: 50)
        ])
    }
}
```

Файл RegistrationView.swift

```
import UIKit

class RegistrationView: UIView {

    // MARK: Outlets
```

```
//ScrollView
private let scrollView = UIScrollView()

//Image
private(set) var imagePicker = UIImagePickerController()
var profileImage: UIImage?

//Labels
private let registrationLabel: UILabel = {
    let label = UILabel()
    label.text = "SignUp"
    label.font = UIFont(name: "Gill Sans SemiBold", size: 33)
    label.numberOfLines = 0
    label.textAlignment = .center
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

private let adviceLabel: UILabel = {
    let label = UILabel()
    label.text = "Please enter your email address and enter password"
    label.font = UIFont(name: "Gill Sans Light", size: 16)
    label.numberOfLines = 2
    label.textAlignment = .center
    label.textColor = .lightGray
    label.translatesAutoresizingMaskIntoConstraints = false
    return label
}()

// Date Picker
private(set) var datePicker: UIDatePicker = {
    let datePicker = UIDatePicker()
    datePicker.datePickerMode = .date
    datePicker.contentHorizontalAlignment = .right
    datePicker.contentVerticalAlignment = .center
    return datePicker
}()

// Segmented Control
private(set) var segmentedControl: UISegmentedControl = {
    let items = ["Male", "Female"]
    let segmentedControl = UISegmentedControl(items: items)
    segmentedControl.selectedSegmentIndex = 0
    return segmentedControl
}()

// Containers
private lazy var firstNameContainer: UIStackView = {
    let stack = Utilities().containerView(withlabel: "First Name", view: nameTextField)
    return stack
}()

private lazy var lastNameContainer: UIStackView = {
    let stack = Utilities().containerView(withlabel: "Last Name", view: lastNameTextField)
    return stack
}()

private lazy var dobContainer: UIStackView = {
    let stack = Utilities().containerView(withlabel: "Day Of Birth", view: datePicker)
    stack.axis = .horizontal
    return stack
}()
```

```
private lazy var genderContainer: UIStackView = {
    let stack = Utilities().containerView(withlabel: "Gender", view: segmentedControl)
    return stack
}()

private lazy var phoneNumberContainer: UIStackView = {
    let stack = Utilities().containerView(withlabel: "Telegram/Viber/WhatsApp", view: phoneTextField)
    return stack
}()

private lazy var emailContainer: UIStackView = {
    let stack = Utilities().containerView(withlabel: "Email", view: emailTextField)
    return stack
}()

private lazy var passwordContainer: UIStackView = {
    let stack = Utilities().containerView(withlabel: "Password", view: passwordTextField)
    return stack
}()

private lazy var confirmPassContainer: UIStackView = {
    let stack = Utilities().containerView(withlabel: "Confirm Password", view: confirmPassTextField)
    return stack
}()

// Text Fields
private(set) var nameTextField: UITextField = {
    let textField = Utilities().textField(withPlaceholder: "First Name")
    textField.returnKeyType = .continue
    return textField
}()

private(set) var lastNameTextField: UITextField = {
    let textField = Utilities().textField(withPlaceholder: "Last Name")
    textField.returnKeyType = .continue
    return textField
}()

private(set) var phoneTextField: UITextField = {
    let textField = Utilities().textField(withPlaceholder: "Username/PhoneNumber")
    textField.returnKeyType = .continue
    return textField
}()

private(set) var emailTextField: UITextField = {
    let textField = Utilities().textField(withPlaceholder: "Email")
    textField.returnKeyType = .continue
    textField.autocapitalizationType = .none
    return textField
}()

private(set) var passwordTextField: UITextField = {
    let textField = Utilities().textField(withPlaceholder: "Password")
    textField.returnKeyType = .continue
    textField.autocapitalizationType = .none
    textField.isSecureTextEntry = true
    return textField
}()

private(set) var confirmPassTextField: UITextField = {
    let textField = Utilities().textField(withPlaceholder: "Confirm Password")
    textField.returnKeyType = .done
```

```
        textField.autocapitalizationType = .none
        textField.isSecureTextEntry = true
        return textField
    }()

    // Buttons
    private(set) var registrationButton: UIButton = {
        let button = UIButton()
        button.setTitle("Register", for: .normal)
        button.backgroundColor = UIColor(named: "lightRed")
        button.titleLabel?.font = UIFont(name: "Gill Sans SemiBold", size: 20)
        button.layer.cornerRadius = 10
        button.layer.borderWidth = 1
        button.layer.borderColor = UIColor.black.cgColor
        return button
    }()

    private(set) var loginButton: UIButton = {
        let button = Utilities().attributedButton("Already have an account?", " Log In")
        return button
    }()

    private(set) var plusPhotoButton: UIButton = {
        let button = UIButton(type: .system)
        button.setImage(UIImage(systemName: "plus"), for: .normal)
        button.tintColor = .black
        button.setDimensions(width: 128, height: 128)
        button.layer.cornerRadius = 128 / 2
        button.layer.borderColor = UIColor.black.cgColor
        button.layer.borderWidth = 3

        return button
    }()

    // MARK: - Initializers
    override init(frame: CGRect) {
        super.init(frame: frame)
        setupUI()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // MARK: Setup UI
    private func setupUI(){
        let stackMain = UIStackView(arrangedSubviews: [firstNameContainer, lastNameContainer,
dobContainer, segmentedControl, phoneNumberContainer, emailContainer, passwordContainer,
confirmPassContainer, registrationButton, loginButton])
        stackMain.translatesAutoresizingMaskIntoConstraints = false
        stackMain.axis = .vertical
        stackMain.spacing = 20

        scrollView.keyboardDismissMode = .onDrag
        scrollView.translatesAutoresizingMaskIntoConstraints = false

        addSubview(scrollView)
        scrollView.addSubview(registrationLabel)
        scrollView.addSubview(adviceLabel)
        scrollView.addSubview(plusPhotoButton)
        scrollView.addSubview(stackMain)

        NSLayoutConstraint.activate([
```

```
            scrollView.topAnchor.constraint(equalTo: safeAreaLayoutGuide.topAnchor),
            scrollView.bottomAnchor.constraint(equalTo: safeAreaLayoutGuide.bottomAnchor),
            scrollView.leadingAnchor.constraint(equalTo: safeAreaLayoutGuide.leadingAnchor),
            scrollView.trailingAnchor.constraint(equalTo: safeAreaLayoutGuide.trailingAnchor)
        ])

        NSLayoutConstraint.activate([
            registrationLabel.topAnchor.constraint(equalTo: scrollView.topAnchor),
            registrationLabel.centerXAnchor.constraint(equalTo: scrollView.centerXAnchor),
            adviceLabel.topAnchor.constraint(equalTo: registrationLabel.bottomAnchor, constant: 10),
            adviceLabel.centerXAnchor.constraint(equalTo: scrollView.centerXAnchor),
            plusPhotoButton.topAnchor.constraint(equalTo: adviceLabel.bottomAnchor, constant: 10),
            plusPhotoButton.centerXAnchor.constraint(equalTo: scrollView.centerXAnchor)
        ])

        NSLayoutConstraint.activate([
            stackMain.leadingAnchor.constraint(equalTo: scrollView.leadingAnchor, constant: 16),
            stackMain.trailingAnchor.constraint(equalTo: scrollView.safeAreaLayoutGuide.trailingAnchor,
constant: -16),
            stackMain.topAnchor.constraint(equalTo: plusPhotoButton.bottomAnchor, constant: 16),
            stackMain.bottomAnchor.constraint(equalTo: scrollView.bottomAnchor, constant: -16)
        ])

        NSLayoutConstraint.activate([
            nameTextField.heightAnchor.constraint(equalToConstant: 50),
            lastNameTextField.heightAnchor.constraint(equalToConstant: 50),
            segmentedControl.heightAnchor.constraint(equalToConstant: 35),
            phoneTextField.heightAnchor.constraint(equalToConstant: 50),
            emailTextField.heightAnchor.constraint(equalToConstant: 50),
            passwordTextField.heightAnchor.constraint(equalToConstant: 50),
            confirmPassTextField.heightAnchor.constraint(equalToConstant: 50),
            registrationButton.heightAnchor.constraint(equalToConstant: 50)
        ])
    }
}
```

Файл EventCell.swift

```
import UIKit

class EventCell: UICollectionViewCell {

    static let identifier = "EventCell"

    // MARK: - Properties
    var event: Event? {
        didSet { configure() }
    }

    // MARK: Outlets
    private lazy var container: UIView = {
        let view = UIView()
        view.backgroundColor = UIColor(named: "lightGray")
        view.layer.cornerRadius = 15
        view.translatesAutoresizingMaskIntoConstraints = false

        return view
    }()

    private lazy var profileImageView: UIImageView = {
        let iv = UIImageView()
        iv.contentMode = .scaleAspectFill
        iv.backgroundColor = .black
```

```
        iv.clipsToBounds = true
        iv.setDimensions(width: 48, height: 48)
        iv.layer.cornerRadius = 48 / 2
        iv.translatesAutoresizingMaskIntoConstraints = false
        iv.layer.borderWidth = 2
        iv.layer.borderColor = UIColor.white.cgColor
        iv.isUserInteractionEnabled = true
        return iv
    }()

    private let titleLabel: UILabel = {
        let label = UILabel()
        label.translatesAutoresizingMaskIntoConstraints = false
        label.font = UIFont(name: "Galvji Bold", size: 18)
        label.textAlignment = .left
        return label
    }()

    private lazy var descriptionContainer: UIStackView = {
        let stack = UIStackView()
        stack.translatesAutoresizingMaskIntoConstraints = false
        stack.alignment = .center
        stack.spacing = 20
        stack.alignment = .fill
        stack.distribution = .fillEqually
        return stack
    }()

    private let sportTypeLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont(name: "Galvji Bold", size: 15)
        label.textColor = .darkGray
        label.textAlignment = .center
        label.layer.cornerRadius = 15
        label.backgroundColor = .white
        label.clipsToBounds = true
        label.layer.borderWidth = 2
        label.layer.borderColor = UIColor.lightGray.cgColor
        return label
    }()

    private let dateLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont(name: "Galvji Bold", size: 15)
        label.textAlignment = .center
        label.textColor = .darkGray
        label.numberOfLines = 3
        label.layer.cornerRadius = 15
        label.backgroundColor = .white
        label.clipsToBounds = true
        label.layer.borderWidth = 2
        label.layer.borderColor = UIColor.lightGray.cgColor
        return label
    }()

    private let userPublishLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont(name: "Gill Sans Light", size: 15)
        label.textColor = .lightGray
        label.textAlignment = .center
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()
```

```
    private lazy var locationButton: UIButton = {
        let button = UIButton()
        button.tintColor = UIColor.customDarkBlue
        button.setDimensions(width: 48, height: 48)
        button.translatesAutoresizingMaskIntoConstraints = false
        button.sizeSymbol(name: "location.fill.viewfinder", size: 48, weight: .light, scale: .medium)
        button.addTarget(self, action: #selector(locationBtn), for: .touchUpInside)
        return button
    }()

    @objc func locationBtn() {

    }

    override init(frame: CGRect) {
        super.init(frame: frame)
        shadowSetup(color: UIColor.gray.cgColor, opacity: 1, radius: 3)
        backgroundColor = .clear
        addSubview(container)

        NSLayoutConstraint.activate([
            container.topAnchor.constraint(equalTo: topAnchor, constant: 8),
            container.leadingAnchor.constraint(equalTo: leadingAnchor, constant: 8),
            container.trailingAnchor.constraint(equalTo: trailingAnchor, constant: -8),
            container.bottomAnchor.constraint(equalTo: bottomAnchor, constant: -8),
        ])

        container.addSubview(profileImageView)
        container.addSubview(locationButton)
        container.addSubview(titleLabel)
        container.addSubview(descriptionContainer)
        descriptionContainer.addArrangedSubview(sportTypeLabel)
        descriptionContainer.addArrangedSubview(dateLabel)
        container.addSubview(userPublishLabel)

        NSLayoutConstraint.activate([
            profileImageView.leadingAnchor.constraint(equalTo: container.leadingAnchor, constant: 12),
            profileImageView.topAnchor.constraint(equalTo: container.topAnchor, constant: 12),

            titleLabel.topAnchor.constraint(equalTo: container.topAnchor, constant: 12),
            titleLabel.leadingAnchor.constraint(equalTo: profileImageView.trailingAnchor, constant: 12),
            titleLabel.heightAnchor.constraint(equalToConstant: 48),

            locationButton.topAnchor.constraint(equalTo: profileImageView.bottomAnchor, constant: 7),
            locationButton.leadingAnchor.constraint(equalTo: container.leadingAnchor, constant: 12),

            descriptionContainer.topAnchor.constraint(equalTo: titleLabel.bottomAnchor, constant: 7),
            descriptionContainer.leadingAnchor.constraint(equalTo: locationButton.trailingAnchor, constant:
12),
            descriptionContainer.trailingAnchor.constraint(equalTo: container.trailingAnchor, constant: -12),

            descriptionContainer.heightAnchor.constraint(equalToConstant: 48),

            userPublishLabel.leadingAnchor.constraint(equalTo: container.leadingAnchor, constant: 12),
            userPublishLabel.bottomAnchor.constraint(equalTo: container.bottomAnchor, constant: -5),
        ])
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
```

```
    private func configure() {
        guard let event = event else { return }
        let viewModel = EventViewModel(event: event)

        profileImageView.sd_setImage(with: viewModel.profileImageUrl)
        titleLabel.text = event.nameEvent
        dateLabel.text = event.dateTime
        sportTypeLabel.text = event.sportType
        userPublishLabel.attributedText = viewModel.userInfoText
    }
}
```

Файл ProfileEventCell.swift

```
import UIKit

class ProfileEventCell: UICollectionViewCell {

    static let identifier = "ProfileEventCell"

    // MARK: - Properties
    var event: Event? {
        didSet { configure() }
    }

    // MARK: Outlets
    private let titleLabel: UILabel = {
        let label = UILabel()
        label.translatesAutoresizingMaskIntoConstraints = false
        label.font = UIFont(name: "Galvji Bold", size: 18)
        label.textAlignment = .center
        return label
    }()

    private lazy var descriptionContainer: UIStackView = {
        let stack = UIStackView()
        stack.translatesAutoresizingMaskIntoConstraints = false
        stack.alignment = .center
        stack.spacing = 20
        stack.alignment = .fill
        stack.distribution = .fillEqually
        return stack
    }()

    private let sportTypeLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont(name: "Galvji Bold", size: 15)
        label.textColor = .darkGray
        label.textAlignment = .center
        label.layer.cornerRadius = 15
        label.backgroundColor = .white
        label.clipsToBounds = true
        label.layer.borderWidth = 2
        label.layer.borderColor = UIColor.lightGray.cgColor
        return label
    }()

    private let dateLabel: UILabel = {
        let label = UILabel()
        label.font = UIFont(name: "Galvji Bold", size: 15)
        label.textAlignment = .center
        label.textColor = .darkGray
        label.numberOfLines = 3
```

```
        label.layer.cornerRadius = 15
        label.backgroundColor = .white
        label.clipsToBounds = true
        label.layer.borderWidth = 2
        label.layer.borderColor = UIColor.lightGray.cgColor
        return label
    }()

    private lazy var locationButton: UIButton = {
        let button = UIButton()
        button.tintColor = UIColor.customDarkBlue
        button.setDimensions(width: 48, height: 48)
        button.translatesAutoresizingMaskIntoConstraints = false
        button.sizeSymbol(name: "location.fill.viewfinder", size: 48, weight: .light, scale: .medium)
        button.addTarget(self, action: #selector(locationBtn), for: .touchUpInside)
        return button
    }()

    // MARK: - Initializers
    override init(frame: CGRect) {
        super.init(frame: frame)
        setupUI()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // MARK: Actions
    @objc func locationBtn() {
    }

    // MARK: Setup UI
    private func configure() {
        guard let event = event else { return }
        titleLabel.text = event.nameEvent
        dateLabel.text = event.dateTime
        sportTypeLabel.text = event.sportType
    }

    private func setupUI() {
        shadowSetup(color: UIColor.gray.cgColor, opacity: 1, radius: 3)
        backgroundColor = UIColor(named: "lightGray")
        addSubview(locationButton)
        addSubview(titleLabel)
        addSubview(descriptionContainer)
        descriptionContainer.addArrangedSubview(sportTypeLabel)
        descriptionContainer.addArrangedSubview(dateLabel)

        NSLayoutConstraint.activate([
            titleLabel.topAnchor.constraint(equalTo: safeAreaLayoutGuide.topAnchor, constant: 5),
            titleLabel.leadingAnchor.constraint(equalTo: safeAreaLayoutGuide.leadingAnchor, constant: 12),
            titleLabel.heightAnchor.constraint(equalToConstant: 20),
        ])

        NSLayoutConstraint.activate([
            locationButton.topAnchor.constraint(equalTo: titleLabel.bottomAnchor, constant: 7),
            locationButton.leadingAnchor.constraint(equalTo: safeAreaLayoutGuide.leadingAnchor, constant:
12),
        ])

        NSLayoutConstraint.activate([
            descriptionContainer.topAnchor.constraint(equalTo: titleLabel.bottomAnchor, constant: 7),
```

```
            descriptionContainer.leadingAnchor.constraint(equalTo: locationButton.trailingAnchor, constant:
12),
            descriptionContainer.trailingAnchor.constraint(equalTo: safeAreaLayoutGuide.trailingAnchor,
constant: -12),
            descriptionContainer.heightAnchor.constraint(equalToConstant: 48)
        ])
    }
}
```

Файл AddEventView.swift

```
import UIKit

class AddEventView: UIView {

    // MARK: - Variables
    private(set) var activeTextField : UITextField? = nil
    private(set) var selectedSportType: String?
    let sports = ["Run", "Gym", "Football", "Basketball", "Tennis", "Volleyball", "Hockey"]

    // MARK: Outlets
    private let scrollView = UIScrollView()

    // Containers
    private lazy var eventContainer: UIStackView = {
        let stack = Utilities().containerView(withlabel: "Event Name", view: eventTextField)
        return stack
    }()

    private lazy var locationTitleContainer: UIStackView = {
        let stack = Utilities().containerView(withlabel: "Location", view: locationContainer)
        return stack
    }()

    private lazy var locationContainer: UIStackView = {
        let stack = Utilities().imageContainerView(withImage: locationTextField, textField: locationButton)
        stack.axis = .horizontal
        return stack
    }()

    private lazy var dateContainer: UIStackView = {
        let stack = Utilities().containerView(withlabel: "Time & Date", view: datePicker)
        return stack
    }()

    private lazy var sportTypeContainer: UIStackView = {
        let stack = Utilities().containerView(withlabel: "Sport:", view: sportTypePicker)
        return stack
    }()

    private lazy var peopleContainer: UIStackView = {
        let stack = Utilities().containerView(withlabel: "Number of people", view: peopleTextField)
        return stack
    }()

    // Text Fields
    private(set) var eventTextField: UITextField = {
        let textField = Utilities().textField(withPlaceholder: "Enter event name")
        textField.returnKeyType = .continue
        return textField
    }()

    private(set) var locationTextField: UITextField = {
```

```
        let textField = Utilities().textField(withPlaceholder: "Select location")
        textField.returnKeyType = .continue
        return textField
    }()

    private(set) lazy var peopleTextField: UITextField = {
        let textField = Utilities().textField(withPlaceholder: "Enter the number of people")
        textField.returnKeyType = .continue
        textField.keyboardType = .numberPad
        return textField
    }()

    private(set) var datePicker: UIDatePicker = {
        let datePicker = UIDatePicker()
        datePicker.datePickerMode = .dateAndTime
        datePicker.contentHorizontalAlignment = .center
        datePicker.contentVerticalAlignment = .center
        return datePicker
    }()

    private(set) var sportTypePicker = UIPickerView()

    // Buttons
    private(set) var locationButton: UIButton = {
        let button = UIButton()
        button.setImage(UIImage(systemName: "location"), for: .normal)
        return button
    }()

    // MARK: - Initializers
    override init(frame: CGRect) {
        super.init(frame: frame)
        setupUI()
        sportTypePicker.delegate = self
        peopleTextField.delegate = self
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // MARK: Setup UI
    private func setupUI() {
        let stackView = UIStackView(arrangedSubviews: [eventContainer, locationTitleContainer,
dateContainer, sportTypeContainer, peopleContainer])
        stackView.translatesAutoresizingMaskIntoConstraints = false
        stackView.axis = .vertical
        stackView.spacing = 20

        scrollView.keyboardDismissMode = .onDrag
        scrollView.translatesAutoresizingMaskIntoConstraints = false

        addSubview(scrollView)
        scrollView.addSubview(stackView)

        NSLayoutConstraint.activate([
            scrollView.topAnchor.constraint(equalTo: safeAreaLayoutGuide.topAnchor),
            scrollView.bottomAnchor.constraint(equalTo: safeAreaLayoutGuide.bottomAnchor),
            scrollView.leadingAnchor.constraint(equalTo: safeAreaLayoutGuide.leadingAnchor),
            scrollView.trailingAnchor.constraint(equalTo: safeAreaLayoutGuide.trailingAnchor)
        ])

        NSLayoutConstraint.activate([
```

```
            eventTextField.heightAnchor.constraint(equalToConstant: 50),
            locationTextField.heightAnchor.constraint(equalToConstant: 50),
            peopleTextField.heightAnchor.constraint(equalToConstant: 50),

            stackView.leadingAnchor.constraint(equalTo: scrollView.leadingAnchor, constant: 16),
            stackView.trailingAnchor.constraint(equalTo: scrollView.safeAreaLayoutGuide.trailingAnchor,
constant: -16),
            stackView.topAnchor.constraint(equalTo: scrollView.topAnchor, constant: 16),
            stackView.bottomAnchor.constraint(equalTo: scrollView.bottomAnchor, constant: -16),
        ])
    }
}

// MARK: - UIPickerViewDataSource, UIPickerViewDelegate
extension AddEventView: UIPickerViewDataSource, UIPickerViewDelegate {

    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return sports.count
    }

    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) ->
String? {
        return sports[row]
    }

    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
        selectedSportType = sports[row]
    }
}

extension AddEventView: UITextFieldDelegate {
    func textFieldDidBeginEditing(_ textField: UITextField) {
        self.activeTextField = textField
    }

    func textFieldDidEndEditing(_ textField: UITextField) {
        self.activeTextField = nil
    }
}
```

Файл MapView.swift

```
import UIKit
import MapKit

class MapView: UIView {

    // Images
    private(set) var mapPinImage: UIImageView = {
        let imageView = UIImageView()
        imageView.image = UIImage(systemName: "pin.fill")
        imageView.contentMode = .scaleAspectFill
        imageView.clipsToBounds = true
        imageView.setDimensions(width: 40, height: 40)
        return imageView
    }()

    // Labels
```

```
    private(set) var addressLabel: UILabel = {
        let label = UILabel()
        label.layer.cornerRadius = 15
        label.clipsToBounds = true
        label.backgroundColor = UIColor.customDarkBlue
        label.font = UIFont(name: "Gill Sans Light", size: 18)
        label.numberOfLines = 0
        label.textAlignment = .center
        label.textColor = .white
        label.setDimensions(width: 300, height: 50)
        return label
    }()

    // Map
    private(set) var mapView: MKMapView = {
        let map = MKMapView()
        map.overrideUserInterfaceStyle = .light
        map.translatesAutoresizingMaskIntoConstraints = false
        map.layer.cornerRadius = 10
        map.layer.borderWidth = 2
        map.layer.borderColor = UIColor.lightGray.cgColor
        return map
    }()

    // Buttons
    private(set) var doneButton: UIButton = {
        let button = UIButton()
        button.setTitle("Save", for: .normal)
        button.setTitleColor(.white, for: .normal)
        button.titleLabel?.font = UIFont(name: "Gill Sans SemiBold", size: 16)
        button.layer.cornerRadius = 20
        button.backgroundColor = UIColor.customDarkBlue
        button.setDimensions(width: 200, height: 50)
        return button
    }()

    private(set) var userLocationButton: UIButton = {
        let button = UIButton()
        button.layer.cornerRadius = 50 / 2
        button.backgroundColor = UIColor.customDarkBlue
        button.setImage(UIImage(systemName: "location"), for: .normal)
        button.setDimensions(width: 50, height: 50)
        return button
    }()

    private(set) var closeButton: UIButton = {
        let button = UIButton()
        button.layer.cornerRadius = 50 / 2
        button.backgroundColor = UIColor.customDarkBlue
        button.setImage(UIImage(systemName: "xmark"), for: .normal)
        button.setDimensions(width: 50, height: 50)
        return button
    }()

    // MARK: - Initializers
    override init(frame: CGRect) {
        super.init(frame: frame)
        setupUI()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
```

```
    // MARK: Setup UI
    private func setupUI(){
        addSubview(mapView)
        NSLayoutConstraint.activate([
            mapView.topAnchor.constraint(equalTo: topAnchor),
            mapView.bottomAnchor.constraint(equalTo: bottomAnchor),
            mapView.leadingAnchor.constraint(equalTo: leadingAnchor),
            mapView.trailingAnchor.constraint(equalTo: trailingAnchor)
        ])

        mapView.addSubview(mapPinImage)
        NSLayoutConstraint.activate([
            mapPinImage.centerXAnchor.constraint(equalTo: mapView.centerXAnchor),
            mapPinImage.centerYAnchor.constraint(equalTo: mapView.centerYAnchor, constant: -20)
        ])

        mapView.addSubview(closeButton)
        NSLayoutConstraint.activate([
            closeButton.leadingAnchor.constraint(equalTo: mapView.leadingAnchor, constant: 20),
            closeButton.topAnchor.constraint(equalTo: mapView.safeAreaLayoutGuide.topAnchor, constant:
20),
        ])

        mapView.addSubview(addressLabel)
        NSLayoutConstraint.activate([
            addressLabel.topAnchor.constraint(equalTo: mapView.safeAreaLayoutGuide.topAnchor,
constant: 80),
            addressLabel.centerXAnchor.constraint(equalTo: mapView.centerXAnchor)
        ])

        mapView.addSubview(userLocationButton)
        NSLayoutConstraint.activate([
            userLocationButton.trailingAnchor.constraint(equalTo: mapView.trailingAnchor, constant: -20),
            userLocationButton.bottomAnchor.constraint(equalTo:
mapView.safeAreaLayoutGuide.bottomAnchor, constant: -40),
        ])

        mapView.addSubview(doneButton)
        NSLayoutConstraint.activate([
            doneButton.centerXAnchor.constraint(equalTo: mapView.centerXAnchor),
            doneButton.bottomAnchor.constraint(equalTo: mapView.safeAreaLayoutGuide.bottomAnchor,
constant: -5),
        ])
    }
}
```

Файл ProfileHeaderView.swift

```
import UIKit

class ProfileHeaderView: UICollectionReusableView {

    static let identifier = "ProfileHeaderView"

    // MARK: - Properties
    var user: User? {
        didSet { configure() }
    }

    private let profileImageView: UIImageView = {
        let iv = UIImageView ()
        iv.contentMode = .scaleAspectFit
```

```
        iv.clipsToBounds = true
        iv.backgroundColor = .lightGray
        iv.layer.borderColor = UIColor.white.cgColor
        iv.layer.borderWidth = 4
        iv.translatesAutoresizingMaskIntoConstraints = false
        return iv
    }()

    private let fullnameLabel: UILabel = {
        let label = UILabel()
        label.textAlignment = .center
        label.font = UIFont.boldSystemFont(ofSize: 20)
        label.translatesAutoresizingMaskIntoConstraints = false
        return label
    }()

    // MARK: - Lifecycle
    override init(frame: CGRect) {
        super.init(frame: frame)

        addSubview(profileImageView)
        profileImageView.setDimensions(width: 80, height: 80)
        profileImageView.layer.cornerRadius = 80 / 2

        let userDetailsStack = UIStackView(arrangedSubviews: [fullnameLabel])
        userDetailsStack.axis = .vertical
        userDetailsStack.distribution = .fillProportionally
        userDetailsStack.spacing = 4
        userDetailsStack.translatesAutoresizingMaskIntoConstraints = false

        addSubview(userDetailsStack)
        NSLayoutConstraint.activate([

            profileImageView.safeAreaLayoutGuide.topAnchor.constraint(equalTo: topAnchor, constant: 10),
            profileImageView.safeAreaLayoutGuide.leadingAnchor.constraint(equalTo: leadingAnchor),
            profileImageView.centerXAnchor.constraint(equalTo: centerXAnchor),

            userDetailsStack.topAnchor.constraint(equalTo: profileImageView.bottomAnchor),
            userDetailsStack.leadingAnchor.constraint(equalTo: leadingAnchor),
            userDetailsStack.trailingAnchor.constraint(equalTo: trailingAnchor),
            userDetailsStack.bottomAnchor.constraint(equalTo: bottomAnchor)
        ])
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    // MARK: - Helpers

    func configure() {
        guard let user = user else { return }
        profileImageView.sd_setImage(with: user.profileImageUrl)
        fullnameLabel.text = user.fullName
    }
}
```

Файл LoginViewController.swift

```
import UIKit

class LoginViewController: UIViewController {
```

```
    // MARK: View
    private var views = LoginView()

    // MARK: Lifecycle
    override func loadView() {
        view = views
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .systemBackground
        hideKeyboard()
        setupEventHandlers()
    }

    // MARK: Actions
    private func setupEventHandlers() {
        views.loginButton.addTarget(self, action: #selector(self.loginButtonTapped), for: .touchUpInside)
        views.registrationButton.addTarget(self, action: #selector(self.registrationButtonTapped), for:
.touchUpInside)
    }

    @objc func registrationButtonTapped() {
        DispatchQueue.main.asyncAfter(deadline: .now() + 0.1) {
            let registrationVC = RegistrationViewController()
            registrationVC.modalPresentationStyle = .overFullScreen
            self.present(registrationVC, animated: true) {
            }
        }
    }

    @objc func loginButtonTapped() {
        guard let email = views.emailTextField.text,
              !email.isEmpty,
              let password = views.passwordTextField.text,
              !password.isEmpty,
              password.count >= 8 else {
            self.showAlert(withTitle: "Login Error", message: "Check the data is correct and try again!")
            return
        }

        AuthManager.shared.loginUser(email: email, password: password) { success in
            DispatchQueue.main.async {
                if success {
                    let vc = TabBarViewController()
                    vc.modalPresentationStyle = .overFullScreen
                    self.present(vc, animated: true)
                } else {
                    self.showAlert(withTitle: "Log In Error", message: "Check the data is correct and try again!")
                }
            }
        }
    }
}
```

Файл RegistrationViewController.swift

```
import UIKit

class RegistrationViewController: UIViewController {

    // MARK: - Variables
    var activeTextField : UITextField? = nil
```

```
    // MARK: View
    private var views = RegistrationView()

    // MARK: Lifecycle
    override func loadView() {
        view = views
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .systemBackground
        setupKeyboard()
        setupEventHandlers()
        setupDelegates()
    }

    // MARK: Actions
    private func setupEventHandlers() {
        views.registrationButton.addTarget(self, action: #selector(registrationButtonTapped), for:
.touchUpInside)
        views.loginButton.addTarget(self, action: #selector(loginButtonTapped), for: .touchUpInside)
        views.plusPhotoButton.addTarget(self, action: #selector(addProfilePhotoBtn), for: .touchUpInside)
    }

    private func setupDelegates() {
        views.passwordTextField.delegate = self
        views.confirmPassTextField.delegate = self
        views.imagePicker.delegate = self
        views.imagePicker.allowsEditing = true
    }

    @objc func addProfilePhotoBtn() {
        present(views.imagePicker, animated: true, completion: nil)
    }

    @objc func loginButtonTapped() {
        dismiss(animated: true, completion: nil)
    }

    @objc func registrationButtonTapped() {
        guard let profileImage = views.profileImage else {
            self.showAlert(withTitle: "Registration Error", message: "Please select a profile image.")
            return
        }

        guard let firstName = views.nameTextField.text, !firstName.isEmpty,
              let lastName = views.lastNameTextField.text, !lastName.isEmpty,
              let phoneNumber = views.phoneTextField.text, !phoneNumber.isEmpty,
              let email = views.emailTextField.text, !email.isEmpty,
              let password = views.passwordTextField.text, !password.isEmpty, password.count >= 8,
              let confirmPass = views.confirmPassTextField.text, !confirmPass.isEmpty,
              password == confirmPass else {
            showAlert(withTitle: "Registration Error", message: "Check the data is correct and try again")
            return
        }

        let genderIndex = views.segmentedControl.selectedSegmentIndex
        let gender = genderIndex == 0 ? "Male" : "Female"

        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "dd.MM.yyyy"

        let birthday = dateFormatter.string(from: views.datePicker.date)
```

```
        AuthManager.shared.registerNewUser(firstName: firstName,
                                           lastName: lastName,
                                           dayOfBirth: birthday,
                                           gender: gender,
                                           phoneNumber: phoneNumber,
                                           email: email,
                                           password: password,
                                           profileImage: profileImage){ registered in
            DispatchQueue.main.async {
                if registered {
                    let vc = TabBarViewController()
                    vc.modalPresentationStyle = .overFullScreen
                    self.present(vc, animated: true)
                } else {
                    self.showAlert(withTitle: "Registration Error", message: "Check the data is correct and try
again")
                }
            }
        }
    }

    // MARK: Setup Keyboard
    private func setupKeyboard() {
        hideKeyboard()
        NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillShow), name:
UIResponder.keyboardWillShowNotification, object: nil)
        NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillHide), name:
UIResponder.keyboardWillHideNotification, object: nil)
    }

    @objc func keyboardWillShow(notification: NSNotification) {
        guard let keyboardSize = (notification.userInfo?[UIResponder.keyboardFrameEndUserInfoKey] as?
NSValue)?.cgRectValue else { return }
        var shouldMoveViewUp = false

        if let activeTextField = activeTextField {
            let bottomOfTextField = activeTextField.convert(activeTextField.bounds, to: self.view).maxY;
            let topOfKeyboard = self.view.frame.height - keyboardSize.height

            if bottomOfTextField > topOfKeyboard {
                shouldMoveViewUp = true
            }
        }
        if(shouldMoveViewUp) {
            self.view.frame.origin.y = 0 - keyboardSize.height
        }
    }

    @objc func keyboardWillHide(notification: NSNotification) {
        self.view.frame.origin.y = 0
    }
}

extension RegistrationViewController: UITextFieldDelegate {
    func textFieldDidBeginEditing(_ textField: UITextField) {
        self.activeTextField = textField
    }

    func textFieldDidEndEditing(_ textField: UITextField) {
        self.activeTextField = nil
    }
}
```

```
extension RegistrationViewController: UIImagePickerControllerDelegate,
UINavigationControllerDelegate {
    func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any]) {
        guard let profileImage = info[.editedImage] as? UIImage else { return }
        self.views.profileImage = profileImage

        views.plusPhotoButton.layer.cornerRadius = 128 / 2
        views.plusPhotoButton.layer.masksToBounds = true
        views.plusPhotoButton.imageView?.contentMode = .scaleAspectFill
        views.plusPhotoButton.imageView?.clipsToBounds = true
        views.plusPhotoButton.layer.borderColor = UIColor.black.cgColor
        views.plusPhotoButton.layer.borderWidth = 3

        self.views.plusPhotoButton.setImage(profileImage.withRenderingMode(.alwaysOriginal), for:
.normal)

        dismiss(animated: true, completion: nil)
    }
}
```

Файл HomeViewController.swift

```
import UIKit

class HomeViewController: UICollectionViewController {

    // MARK: - Variables
    private var viewModel = HomeViewModel()

    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .systemBackground
        title = "Home"
        viewModel.delegate = self

        viewModel.fetchEvents()
        configureUI()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        viewModel.fetchEvents()
    }

    // MARK: - Helpers
    private func configureUI() {
        let locationButton = UIButton(type: .system)
            locationButton.setImage(UIImage(systemName: "location"), for: .normal)
            locationButton.addTarget(self, action: #selector(locationButtonTapped(_:)), for: .touchUpInside)
            let locationBarButtonItem = UIBarButtonItem(customView: locationButton)

            let filterButton = UIButton(type: .system)
            filterButton.setImage(UIImage(systemName: "line.horizontal.3.decrease.circle"), for: .normal)
            filterButton.addTarget(self, action: #selector(filterButtonTapped(_:)), for: .touchUpInside)
            let filterBarButtonItem = UIBarButtonItem(customView: filterButton)

            navigationItem.rightBarButtonItems = [filterBarButtonItem, locationBarButtonItem]

        let layout = UICollectionViewFlowLayout()
        layout.itemSize = CGSize(width: view.bounds.width - 10, height: 165)
        layout.scrollDirection = .vertical
```

```
        layout.sectionInset = UIEdgeInsets(top: 16, left: 16, bottom: 16, right: 16)
        collectionView = UICollectionView(frame: .zero, collectionViewLayout: layout)
        collectionView.register(EventCell.self, forCellWithReuseIdentifier: EventCell.identifier)
    }

    @objc private func filterButtonTapped(_ sender: UIButton) {
        showSportTypeSelection()
    }

    @objc private func locationButtonTapped(_ sender: UIButton) {
        showLocationInputAlert()
    }

    private func showLocationInputAlert() {
        let alertController = UIAlertController(title: "Enter Your City", message: nil, preferredStyle: .alert)
        alertController.addTextField { textField in
            textField.placeholder = "City"
        }

        let searchAction = UIAlertAction(title: "Search", style: .default) { [weak self] _ in
            if let city = alertController.textFields?.first?.text {
                self?.viewModel.filterEventsByLocation(city)
            }
        }
        alertController.addAction(searchAction)

        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
        alertController.addAction(cancelAction)

        present(alertController, animated: true, completion: nil)
    }

    private func showSportTypeSelection() {
        let alertController = UIAlertController(title: "Select Sport Type", message: nil, preferredStyle:
.actionSheet)
        alertController.popoverPresentationController?.sourceView =
navigationItem.rightBarButtonItem?.customView

        let sportTypes = ["All", "Run", "Gym", "Football", "Basketball", "Tennis", "Volleyball", "Hockey"]
        for sportType in sportTypes {
            let action = UIAlertAction(title: sportType, style: .default) { [weak self] _ in
                self?.viewModel.filterEventsBySportType(sportType)
            }
            alertController.addAction(action)
        }

        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
        alertController.addAction(cancelAction)

        present(alertController, animated: true, completion: nil)
    }
}

// MARK: - UICollectionViewDelegate/DataSource
extension HomeViewController {
    override func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int)
-> Int {
        return viewModel.events.count
    }

    override func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath)
-> UICollectionViewCell {
```

```
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: EventCell.identifier, for:
indexPath) as! EventCell
        cell.event = viewModel.events[indexPath.row]
        return cell
    }

    override func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath:
IndexPath) {
        let controller = DetailsEventViewController()
        controller.event = viewModel.events[indexPath.row]
        navigationController?.pushViewController(controller, animated: true)
    }
}

extension HomeViewController: EventsVMDelegate {
    func didShowAlert() {
        showAlert(withTitle: "Event not found", message: "Please try again")
    }

    func didUpdateEvents(_ events: [Event]) {
        collectionView.reloadData()
    }
}
```

Файл AddEventViewController.swift

```
import UIKit

class AddEventViewController: UIViewController {

    // MARK: View
    private var views = AddEventView()

    // MARK: Lifecycle
    override func loadView() {
        view = views
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        title = "Add Event"
        view.backgroundColor = .systemBackground
        setupKeyboard()
        setupEventHandlers()
    }

    // MARK: Actions
    private func setupEventHandlers() {
        navigationItem.rightBarButtonItem = UIBarButtonItem(title: "Done",
                                                            style: .done,
                                                            target: self,
                                                            action: #selector(saveButtonTapped))
        views.locationButton.addTarget(self, action: #selector(locationButtonTapped), for: .touchUpInside)
    }

    @objc private func locationButtonTapped() {
        let mapVC = MapViewController(mapType: .AddEventMap)
        mapVC.mapViewControllerDelegate = self
        present(mapVC, animated: true, completion: nil)
    }

    @objc private func saveButtonTapped() {
        guard let nameEvent = views.eventTextField.text, !nameEvent.isEmpty,
```

```
            let location = views.locationTextField.text, !nameEvent.isEmpty else {
            showAlert(withTitle: "Event creation error",
                    message: "Make sure that the fields 'Event Name' and 'Location' have been filled in!")
            return
        }

        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "dd.MM.yyyy HH:mm"
        let dataTime = dateFormatter.string(from: views.datePicker.date)

        DatabaseManager.shared.insertNewEvent(nameEvent: nameEvent, location: location, dateTime:
dataTime, sportType: views.selectedSportType ?? views.sports[0], comment: views.peopleTextField.text
?? "") { inserted in
            if inserted {
                let vc = TabBarViewController()
                vc.modalPresentationStyle = .overFullScreen
                self.present(vc, animated: true)
            } else {
                self.showAlert(withTitle: "Event creation error",
                            message: "Make sure that the fields 'Event Name' and 'Location' have been filled in!")
            }
        }
    }

    // MARK: Setting Keyboard
    private func setupKeyboard() {
        hideKeyboard()
        NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillShow), name:
UIResponder.keyboardWillShowNotification, object: nil)
        NotificationCenter.default.addObserver(self, selector: #selector(keyboardWillHide), name:
UIResponder.keyboardWillHideNotification, object: nil)
    }

    @objc func keyboardWillShow(notification: NSNotification) {
        guard let keyboardSize = (notification.userInfo?[UIResponder.keyboardFrameEndUserInfoKey] as?
NSValue)?.cgRectValue else {
            return
        }
        var shouldMoveViewUp = false

        if let activeTextField = views.activeTextField {

            let bottomOfTextField = activeTextField.convert(activeTextField.bounds, to: self.view).maxY;
            let topOfKeyboard = self.view.frame.height - keyboardSize.height

            if bottomOfTextField > topOfKeyboard {
                shouldMoveViewUp = true
            }
        }
        if(shouldMoveViewUp) {
            self.view.frame.origin.y = 0 - keyboardSize.height
        }
    }

    @objc func keyboardWillHide(notification: NSNotification) {
        self.view.frame.origin.y = 0
    }
}

extension AddEventViewController: MapViewControllerDelegate {
    func getAddress(_ address: String?) {
        views.locationTextField.text = address
    }
```

```
}
```

Файл MapViewController.swift

```
import UIKit
import MapKit
import CoreLocation

enum MapType {
    case AllEvenstMap
    case AddEventMap
}

protocol MapViewControllerDelegate {
    func getAddress(_ address: String?)
}

class MapViewController: UIViewController {

    let mapManager = MapManager()
    var mapViewControllerDelegate: MapViewControllerDelegate?
    private var events: [Event] = []
    private var mapAnnotations: [MKPointAnnotation] = []

    var mapType: MapType

    // MARK: View
    private var views = MapView()

    // MARK: Lifecycle
    override func loadView() {
        view = views
    }

    init(mapType: MapType) {
        self.mapType = mapType
        super.init(nibName: nil, bundle: nil)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        setupEventHandlers()
        mapConfig()
    }

    // MARK: Actions
    private func setupEventHandlers() {
        views.closeButton.addTarget(self, action: #selector(closeButtonPresed), for: .touchUpInside)
        views.userLocationButton.addTarget(self, action: #selector(centerViewInUserLocation), for:
.touchUpInside)
        views.doneButton.addTarget(self, action: #selector(doneButtonPresed), for: .touchUpInside)
    }

    @objc func doneButtonPresed() {
        mapViewControllerDelegate?.getAddress(views.addressLabel.text)
        dismiss(animated: true)
    }
```

```
    @objc func centerViewInUserLocation() {
        mapManager.showUserLocation(mapView: views.mapView)
    }

    @objc func closeButtonPresed() {
        dismiss(animated: true)
    }

    private func setupMapView() {
        mapManager.checkLocationServices(mapView: views.mapView) {
            self.mapManager.locationManager.delegate = self
        }
        self.mapManager.showUserLocation(mapView: views.mapView)

    }
    private func fetchEvents() {
        DatabaseManager.shared.fetchEvents { events in
            self.events = events
            self.addAnnotations()
        }
    }


    private func addAnnotations() {
        for event in events {
            let annotation = MKPointAnnotation()
            let geocoder = CLGeocoder()

            geocoder.geocodeAddressString(event.location) { (placemarks, error) in
                guard let placemark = placemarks?.first,
                      let location = placemark.location?.coordinate else {
                    return
                }
                annotation.coordinate = location
                annotation.title = event.nameEvent
                self.views.mapView.addAnnotation(annotation)
                self.mapAnnotations.append(annotation)
            }
        }
    }

    func mapConfig() {
        views.mapView.delegate = self
        switch mapType {
        case .AllEvenstMap:
            views.doneButton.isHidden = true
            views.closeButton.isHidden = true
            views.mapPinImage.isHidden = true
            views.addressLabel.isHidden = true
            fetchEvents()
            mapManager.checkLocationServices(mapView: views.mapView) {
                self.mapManager.locationManager.delegate = self
            }
        case .AddEventMap:
            views.addressLabel.text = ""
            setupMapView()
        }
    }
}

extension MapViewController: MKMapViewDelegate {

    func mapView(_ mapView: MKMapView, regionDidChangeAnimated animated: Bool) {
```

```
        let center = mapManager.getCenterLocation(for: mapView)
        let geocoder = CLGeocoder()

        geocoder.cancelGeocode()
        geocoder.reverseGeocodeLocation(center) { (placemarks, error) in
            if let error = error {
                print(error)
                return
            }

            guard let placemarks = placemarks else { return }
            let placemark = placemarks.first
            let streetName = placemark?.thoroughfare
            let buildNumber = placemark?.subThoroughfare
            let city = placemark?.locality

            DispatchQueue.main.async {
                if streetName != nil && buildNumber != nil && city != nil {
                    self.views.addressLabel.text = "\(city!), \(streetName!), \(buildNumber!)"
                } else if streetName != nil && city != nil {
                    self.views.addressLabel.text = "\(city!), \(streetName!)"
                } else {
                    self.views.addressLabel.text = ""
                }
            }
        }
    }

    func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) -> MKAnnotationView? {
        guard annotation is MKPointAnnotation else {
            return nil
        }

        let identifier = "event"
        var annotationView = mapView.dequeueReusableAnnotationView(withIdentifier: identifier)

        if annotationView == nil {
            annotationView = MKPinAnnotationView(annotation: annotation, reuseIdentifier: identifier)
            annotationView?.canShowCallout = true
            annotationView?.rightCalloutAccessoryView = UIButton(type: .detailDisclosure)
        } else {
            annotationView?.annotation = annotation
        }

        return annotationView
    }

    func mapView(_ mapView: MKMapView, annotationView view: MKAnnotationView,
calloutAccessoryControlTapped control: UIControl) {
        if control == view.rightCalloutAccessoryView {
            if let annotation = view.annotation as? MKPointAnnotation {
                // Find the selected annotation in the mapAnnotations array
                if let selectedAnnotation = mapAnnotations.first(where: { $0.title == annotation.title }) {
                    // Get the corresponding event based on the annotation
                    let selectedEvent = events.first(where: { $0.nameEvent == selectedAnnotation.title })

                    // Create an instance of DetailViewController and configure it with the selected event
                    let detailViewController = DetailsEventViewController()
                    detailViewController.event = selectedEvent
                    present(detailViewController, animated: true)

                }
```

```
            }
        }
    }
}

extension MapViewController: CLLocationManagerDelegate {

    func locationManager(_ manager: CLLocationManager,
                         didChangeAuthorization status: CLAuthorizationStatus) {
        mapManager.checkLocationAuthorization(mapView: views.mapView)
    }
}
```

Файл ProfileViewController.swift

```
import UIKit
import SDWebImage
import FirebaseAuth

protocol ProfileViewControllerDelegate: AnyObject {
    func didDeleteEvent(withID eventID: String)
}

class ProfileViewController: UICollectionViewController {

    private var viewModel = ProfileViewModel()

        // Buttons
        private lazy var logOutButton: UIButton = {
            let button = UIButton()
            button.addTarget(self, action: #selector(logOutBtn), for: .touchUpInside)
            button.sizeSymbol(name: "rectangle.portrait.and.arrow.right", size: 20, weight: .light, scale:
.medium)
            return button
        }()

    // MARK: VC Lifecycle
    override func viewDidLoad() {
        super.viewDidLoad()
        title = "Profile"
        view.backgroundColor = UIColor(named: "lightGray")
        configureCollectionView()
        viewModel.delegate = self
        viewModel.loadData()
        setupNavBar()
    }

    func configureCollectionView() {
        collectionView.backgroundColor = .white
        collectionView.register(ProfileEventCell.self, forCellWithReuseIdentifier: ProfileEventCell.identifier)
        collectionView.register(ProfileHeaderView.self,
                                forSupplementaryViewOfKind: UICollectionView.elementKindSectionHeader,
                                withReuseIdentifier: ProfileHeaderView.identifier)
    }

    private func setupNavBar() {
        navigationItem.rightBarButtonItem = UIBarButtonItem(customView: logOutButton)
    }

    // MARK: Actions
    @objc private func logOutBtn() {
        showConfirmationAlert(withTitle: "Log Out", message: "Are you sure you want to log out?",
confirmationHandler: {
```

```
            AuthManager.shared.logOut(completion: { success in
                DispatchQueue.main.async {
                    if success {
                        // Present log in
                        let loginVC = LoginViewController()
                        loginVC.modalPresentationStyle = .fullScreen
                        self.present(loginVC, animated: true) {
                            self.navigationController?.popToRootViewController(animated: false)
                            self.tabBarController?.selectedIndex = 0
                        }
                    } else {
                        // Error occurred
                        fatalError("Could not log out user")
                    }
                }
            })
        })
    }
}

// MARK: - UICollectionViewDataSource
extension ProfileViewController {
    override func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int)
-> Int {
        return viewModel.events.count
    }

    override func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath)
-> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: ProfileEventCell.identifier, for:
indexPath) as! ProfileEventCell
        cell.event = viewModel.events[indexPath.row]
        return cell
    }
}

// MARK: - UICollectionViewDelegate
extension ProfileViewController {
    override func collectionView(_ collectionView: UICollectionView,
viewForSupplementaryElementOfKind kind: String, at indexPath: IndexPath) ->
UICollectionReusableView {
        let header = collectionView.dequeueReusableSupplementaryView(ofKind: kind, withReuseIdentifier:
ProfileHeaderView.identifier, for: indexPath) as! ProfileHeaderView
        header.user = viewModel.user
        return header
    }

    override func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath:
IndexPath) {
        let controller = DetailsEventViewController()
        controller.delegate = self
        controller.event = viewModel.events[indexPath.row]
        navigationController?.pushViewController(controller, animated: true)
    }
}

// MARK: - UICollectionViewDelegateFlowLayout
extension ProfileViewController: UICollectionViewDelegateFlowLayout {
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, referenceSizeForHeaderInSection section: Int) -> CGSize {
        return CGSize(width: view.frame.width, height: 150)
    }
```

```
    func collectionView(_ collectionView: UICollectionView, layout collectionViewLayout:
UICollectionViewLayout, sizeForItemAt indexPath: IndexPath) -> CGSize {
        return CGSize(width: view.frame.width, height: 100)
    }
}

extension ProfileViewController: EventsVMDelegate {
    func didShowAlert() {
        showAlert(withTitle: "Let's go to sports!", message: "Please add new events")
    }

    func didUpdateEvents(_ events: [Event]) {
        collectionView.reloadData()
    }
}

extension ProfileViewController: ProfileViewControllerDelegate {
    func didDeleteEvent(withID eventID: String) {
        viewModel.deleteEvent(withID: eventID)
        collectionView.reloadData()
    }
}
```

Файл DetailsEventViewController.swift

```
import UIKit
import FirebaseAuth

class DetailsEventViewController: UIViewController {

    var event: Event!
    let scrollView = UIScrollView()
    weak var delegate: ProfileViewControllerDelegate?

    //MARK: Outlets

    // Images
    private lazy var headerImage: UIImageView = {
        let imageView = UIImageView()
        imageView.image = UIImage(named: "run")
        imageView.contentMode = .scaleAspectFill
        imageView.translatesAutoresizingMaskIntoConstraints = false
        imageView.clipsToBounds = true
        return imageView
    }()

    private lazy var userImage: UIImageView = {
        let imageView = UIImageView()
        imageView.layer.cornerRadius = 100 / 2
        imageView.setDimensions(width: 100, height: 100)
        imageView.clipsToBounds = true
        imageView.contentMode = .scaleAspectFill
        imageView.layer.borderWidth = 8
        imageView.layer.borderColor = UIColor(named: "lightGray")?.cgColor
        return imageView
    }()

    // Labels
    private lazy var authorLabel: UILabel = {
        let label = UILabel()
        label.translatesAutoresizingMaskIntoConstraints = false
        label.font = UIFont(name: "Gill Sans Light", size: 18)
        label.numberOfLines = 0
        label.textAlignment = .center
```

```
    return label
}()

private lazy var titleLabel: UILabel = {
    let label = UILabel()
    label.translatesAutoresizingMaskIntoConstraints = false
    label.font = UIFont(name: "Galvji Bold", size: 28)
    label.numberOfLines = 0
    label.textAlignment = .center
    return label
}()

private lazy var sportTypeLabel: UILabel = {
    let label = UILabel()
    label.font = UIFont(name: "Galvji Bold", size: 17)
    label.textAlignment = .center
    label.layer.cornerRadius = 15
    label.backgroundColor = .white
    label.clipsToBounds = true
    label.numberOfLines = 2
    label.layer.borderWidth = 2
    label.layer.borderColor = UIColor.lightGray.cgColor
    return label
}()

private lazy var peopleLabel: UILabel = {
    let label = UILabel()
    label.font = UIFont(name: "Galvji Bold", size: 15)
    label.textAlignment = .center
    label.layer.cornerRadius = 15
    label.backgroundColor = .white
    label.clipsToBounds = true
    label.numberOfLines = 3
    label.layer.borderWidth = 2
    label.layer.borderColor = UIColor.lightGray.cgColor
    return label
}()

private lazy var dateLabel: UILabel = {
    let label = UILabel()
    label.font = UIFont(name: "Galvji Bold", size: 15)
    label.textAlignment = .center
    label.layer.cornerRadius = 15
    label.backgroundColor = .white
    label.clipsToBounds = true
    label.numberOfLines = 3
    label.layer.borderWidth = 2
    label.layer.borderColor = UIColor.lightGray.cgColor
    return label
}()

private lazy var locationTitle: UILabel = {
    let label = UILabel()
    label.text = "Location"
    label.font = UIFont(name: "Galvji", size: 20)
    label.translatesAutoresizingMaskIntoConstraints = false
    label.textAlignment = .center
    return label
}()

private lazy var locationLabel: UILabel = {
    let label = UILabel()
    label.font = UIFont(name: "Galvji Bold", size: 15)
```

```
        label.textAlignment = .center
        label.layer.cornerRadius = 10
        label.numberOfLines = 1
        label.layer.borderWidth = 2
        label.layer.borderColor = UIColor.lightGray.cgColor
        return label
    }()

    // Stacks
    private lazy var detailsStack: UIStackView = {
        let stack = UIStackView()
        stack.translatesAutoresizingMaskIntoConstraints = false
        stack.alignment = .center
        stack.spacing = 10
        stack.alignment = .fill
        stack.distribution = .fillEqually
        return stack
    }()

    private lazy var stackMain: UIStackView = {
        let stackMain = UIStackView(arrangedSubviews: [titleLabel, detailsStack, locationTitle,
locationLabel, openChatButton, deleteButton])
        stackMain.translatesAutoresizingMaskIntoConstraints = false
        stackMain.axis = .vertical
        stackMain.spacing = 20
        stackMain.layer.cornerRadius = 15
        stackMain.backgroundColor = UIColor(named: "lightGray")
        stackMain.layoutMargins = UIEdgeInsets(top: 20, left: 20, bottom: 20, right: 20)
        stackMain.isLayoutMarginsRelativeArrangement = true
        //shadow
        stackMain.layer.masksToBounds = false
        stackMain.layer.shadowColor = UIColor.gray.cgColor
        stackMain.layer.shadowOpacity = 1
        stackMain.layer.shadowOffset = CGSize.zero
        stackMain.layer.shadowRadius = 3
        return stackMain
    }()

    // Buttons
    private lazy var openChatButton: UIButton = {
        let button = UIButton()
        button.setTitle("Open Chat", for: .normal)
        button.addTarget(self, action: #selector(openChatAction), for: .touchUpInside)
        button.setTitleColor(.white, for: .normal)
        button.titleLabel?.font = UIFont(name: "Gill Sans SemiBold", size: 16)
        button.layer.cornerRadius = 20
        button.backgroundColor = UIColor.customDarkBlue
        button.setDimensions(width: 300, height: 50)
        button.isHidden = false
        return button
    }()

    private lazy var deleteButton: UIButton = {
        let button = UIButton()
        button.setTitle("Delete Event", for: .normal)
        button.addTarget(self, action: #selector(deleteEventAction), for: .touchUpInside)
        button.setTitleColor(.white, for: .normal)
        button.titleLabel?.font = UIFont(name: "Gill Sans SemiBold", size: 16)
        button.layer.cornerRadius = 20
        button.backgroundColor = UIColor.customRed
        button.setDimensions(width: 300, height: 50)
        button.isHidden = true
        return button
```

```
    }()

    override func viewDidLoad() {
        super.viewDidLoad()
        configure()
        setupUI()
    }

    @objc func openChatAction() {
        let username = event.user.phoneNumber

        let alertController = UIAlertController(title: "Open Chat", message: "Select an app", preferredStyle:
.actionSheet)

        // Telegram
        if let telegramURL = URL(string: "tg://resolve?domain=\(username)") {
            print(telegramURL)
            let telegramAction = UIAlertAction(title: "Telegram", style: .default) { _ in
                UIApplication.shared.open(telegramURL, options: [:], completionHandler: nil)
            }
            alertController.addAction(telegramAction)
        }

        // Viber
        if let viberURL = URL(string: "viber://pa?chatURI=\(username)") {
            let viberAction = UIAlertAction(title: "Viber", style: .default) { _ in
                UIApplication.shared.open(viberURL, options: [:], completionHandler: nil)
            }
            alertController.addAction(viberAction)
        }

        // WhatsApp
        if let whatsappURL = URL(string: "whatsapp://send?phone=\(username)") {
            let whatsappAction = UIAlertAction(title: "WhatsApp", style: .default) { _ in
                UIApplication.shared.open(whatsappURL, options: [:], completionHandler: nil)
            }
            alertController.addAction(whatsappAction)
        }

        // Cancel action
        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
        alertController.addAction(cancelAction)

        // Present the action sheet
        if let popoverController = alertController.popoverPresentationController {
            popoverController.sourceView = self.view
            popoverController.sourceRect = self.view.bounds
        }
        present(alertController, animated: true, completion: nil)
    }

    private func configure() {
        userImage.sd_setImage(with: event.user.profileImageUrl)
        authorLabel.text = event.user.fullName
        titleLabel.text = event.nameEvent
        sportTypeLabel.text = event.sportType
        peopleLabel.text = "Number of people:" + " " + event.comment
        dateLabel.text = event.dateTime
        locationLabel.text = event.location

        if event.user.uid == Auth.auth().currentUser?.uid {
            deleteButton.isHidden = false
            openChatButton.isHidden = true
```

```
        }
    }
    @objc private func deleteEventAction() {
        let alertController = UIAlertController(title: "Delete Event", message: "Are you sure you want to
delete this event?", preferredStyle: .alert)
        let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
        alertController.addAction(cancelAction)

        let deleteAction = UIAlertAction(title: "Delete", style: .destructive) { _ in

            DatabaseManager.shared.deleteEvent(eventID: self.event.eventID) { success in
                if success {
                    if let navigationController = self.navigationController {
                        self.delegate?.didDeleteEvent(withID: self.event.eventID)
                        navigationController.popToRootViewController(animated: true)
                    } else {
                        self.delegate?.didDeleteEvent(withID: self.event.eventID)
                        self.dismiss(animated: true, completion: nil)
                    }

                } else {
                    DispatchQueue.main.async {
                        self.showAlert(withTitle: "Deletion error", message: "Please try again")
                    }
                }
            }
        }
        alertController.addAction(deleteAction)
        present(alertController, animated: true, completion: nil)
    }

    //MARK: Constraints
    private func setupUI(){
        view.backgroundColor = .systemBackground
        scrollView.translatesAutoresizingMaskIntoConstraints = false
        scrollView.contentInsetAdjustmentBehavior = .never
        scrollView.addBlackGradientLayerInForeground(frame: view.bounds, colors: [.clear, .black])

        view.addSubview(scrollView)
        NSLayoutConstraint.activate([
            scrollView.topAnchor.constraint(equalTo: view.topAnchor),
            scrollView.bottomAnchor.constraint(equalTo: view.safeAreaLayoutGuide.bottomAnchor),
            scrollView.leadingAnchor.constraint(equalTo: view.leadingAnchor),
            scrollView.trailingAnchor.constraint(equalTo: view.trailingAnchor)
        ])

        scrollView.addSubview(headerImage)
        NSLayoutConstraint.activate([
            headerImage.topAnchor.constraint(equalTo: scrollView.topAnchor),
            headerImage.leadingAnchor.constraint(equalTo: scrollView.leadingAnchor),
            headerImage.trailingAnchor.constraint(equalTo: scrollView.safeAreaLayoutGuide.trailingAnchor),
            headerImage.heightAnchor.constraint(equalToConstant: 200)
        ])

        scrollView.addSubview(userImage)
        NSLayoutConstraint.activate([
            userImage.topAnchor.constraint(equalTo: headerImage.bottomAnchor, constant: -45),
            userImage.centerXAnchor.constraint(equalTo: scrollView.centerXAnchor),
        ])

        scrollView.addSubview(authorLabel)
        NSLayoutConstraint.activate([
            authorLabel.topAnchor.constraint(equalTo: userImage.bottomAnchor, constant: 3),
```

```
            authorLabel.centerXAnchor.constraint(equalTo: scrollView.centerXAnchor)
        ])

        scrollView.addSubview(stackMain)
        detailsStack.addArrangedSubview(dateLabel)
        detailsStack.addArrangedSubview(sportTypeLabel)
        detailsStack.addArrangedSubview(peopleLabel)

        NSLayoutConstraint.activate([
            detailsStack.heightAnchor.constraint(equalToConstant: 75),
            locationLabel.heightAnchor.constraint(equalToConstant: 30),

            stackMain.topAnchor.constraint(equalTo: authorLabel.bottomAnchor, constant: 10),
            stackMain.leadingAnchor.constraint(equalTo: scrollView.leadingAnchor, constant: 10),
            stackMain.trailingAnchor.constraint(equalTo: scrollView.safeAreaLayoutGuide.trailingAnchor,
constant: -10),
            stackMain.bottomAnchor.constraint(equalTo: scrollView.bottomAnchor, constant: -15),
        ])
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ТА ПЛАНУВАННЯ СПІЛЬНИХ СПОРТИВНИХ ПОДІЙ

Програма та методика тестування

КПІ.ІТ-9428.045490.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Олександр ХОМЕНКО

Київ – 2023

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є мобільний застосунок для організації та планування спільних спортивних подій. Програмний продукт створений мовою Swift в середовищі розробки XCode. Застосунок має підтримувати версію iOS не менше 15.0.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– переконатися, що створення облікового запису для користувача працює правильно і безперебійно;

– перевірити, що авторизація та деавторизація користувача працюють коректно і забезпечують безпеку системи;

– перевірити, що створення спортивних подій відбувається згідно з вимогами, включаючи вказання назви, дати, часу, локації, типу спорту та кількості учасників;

– переконатися, що перегляд списку спортивних подій дозволяє користувачу швидко та ефективно знайти події за допомогою фільтрації за типом спорту або локацією;

– перевірити, що користувач може отримати докладну інформацію про конкретну спортивну подію;

– переконатися, що користувач може переглядати спортивні події на мапі для зручного визначення їх розташування;

– перевірити, що перехід до чату з іншим користувачем відбувається успішно, забезпечуючи зручну комунікацію між користувачами.

Загальна мета тестування полягає у виявленні будь-яких дефектів, помилок або некоректного функціонування системи, щоб їх можна було виправити та покращити перед впровадженням застосунку.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– статичне тестування – перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;

– функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

– системне тестування – перевіряється усе програмне забезпечення в цілому;

– тестування «білої скриньки» – об’єктом тестування є внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним;

– тестування інтерфейсу користувача на різних пристроях за допомогою симуляторів XCode.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Для перевірки якості та виявлення помилок та недоліків у функціональній частині програмного забезпечення та його зручності в користуванні застосовано ручне тестування. Планується виконати наступні види тестування, щоб перевірити працездатність та стійкість додатку:

– тестування на різних моделях та розмірах смартфонів iPhone;

– тестування зміни орієнтації екрану;

– тестування працездатності програми при відсутності з'єднання з мережею;

– тестування інтерфейсу користувача;

– тестування безпеки та надійності зберігання даних користувачів;

– тестування ефективності та швидкодії застосунку;

– тестування зручності використання.

Ці тести допоможуть нам впевнитися в належній працездатності та надійності нашого додатку, а також забезпечити зручне використання користувачами.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ТА ПЛАНУВАННЯ СПІЛЬНИХ СПОРТИВНИХ ПОДІЙ

Керівництво користувача

КПІ.ІТ-9428.045490.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Олександр ХОМЕНКО

Київ – 2023

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Мобільний застосунок для організації та планування спільних спортивних подій призначений для пошуку партнерів для спільного заняття спортом або гравців будь-якого рівня для подій пов’язаних із командними видами спорту.

За допомогою цього застосунку користувачі можуть створювати та переглядати спортивні події, вибирати місце проведення на мапі або вручну, вказувати тип спорту та кількість учасників. Для зручного користування було розроблено інтуїтивно зрозумілий та простий інтерфейс для користувачів будь-якого віку.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

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 застосунку на мобільному пристрої повинно бути не менше 80 МБ вільної пам’яті.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч, використовуючи IDE XCode. Для цього необхідно підключити мобільний пристрій до комп’ютера, дочекатись поки він відобразиться у панелі запуску симулятора та натиснути на кнопку «Run».

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку. Користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись початкова сторінка застосунку.

У разі, якщо дана іконка не з’явилась, то при завантаженні відбулася помилка. Необхідно уважно перевірити чи розблокований пристрій і чи добре підключений до комп’ютера.

3 ВИКОНАННЯ ПРОГРАМИ

При першому запуску застосунку після завантаження користувачу надається можливість створити новий обліковий запис або авторизуватися. Якщо користувач вже має обліковий запис, він може ввести свої облікові дані (email і пароль) і натиснути кнопку "Login". Він буде авторизований в системі. Якщо користувач ще не має облікового запису, він може натиснути на посилання "Registration" і ввести необхідну інформацію. Після цього він отримає новий обліковий запис. Після успішної авторизації або створення облікового запису, користувач потрапляє на головний екран застосунку. Екрани авторизації та реєстрації зображено на рисунку 3.1

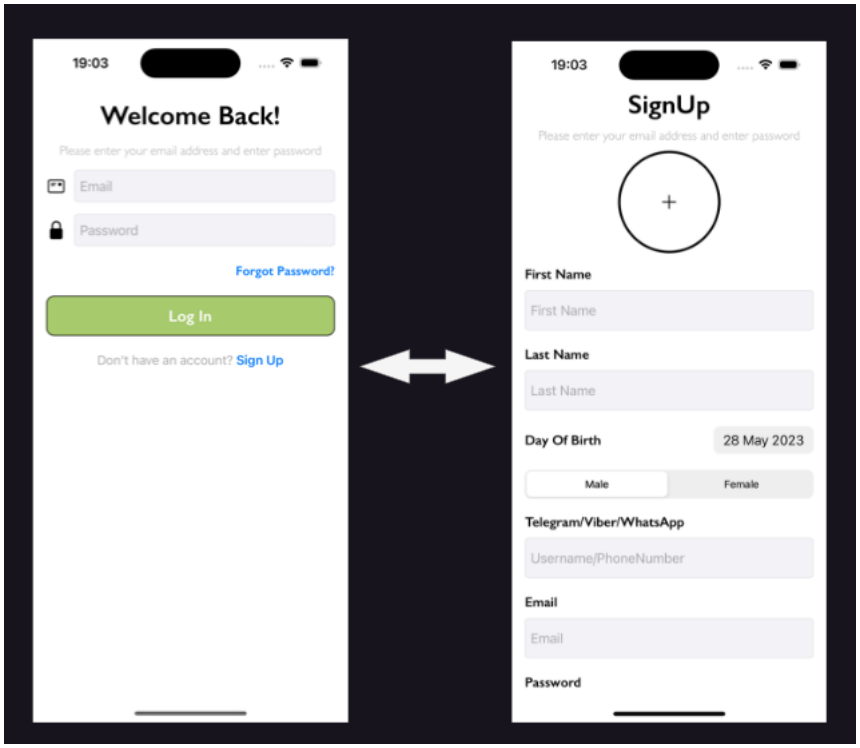


Рисунок 3.1 – Екрани авторизації та реєстрації

На головному екрані користувач може бачити список доступних спортивних подій, які відображаються відповідно до його фільтрів і налаштувань. Для перегляду детальної інформації про певну спортивну подію, користувач торкається відповідного елемента списку подій і відкривається сторінка з деталізованою інформацією про подію, включаючи назву, дату, час, місце проведення, тип спорту та кількість необхідних учасників. Екрани зі списком подій та детальною інформацією зображено на

рисунку 3.2. Щоб перейти до чату з іншим користувачем, необхідно відкрити сторінку деталізованої інформації про певну спортивну подію і натиснути "Open Chat". Після цього необхідно вибрати доступний месенджер та почати спілкування.

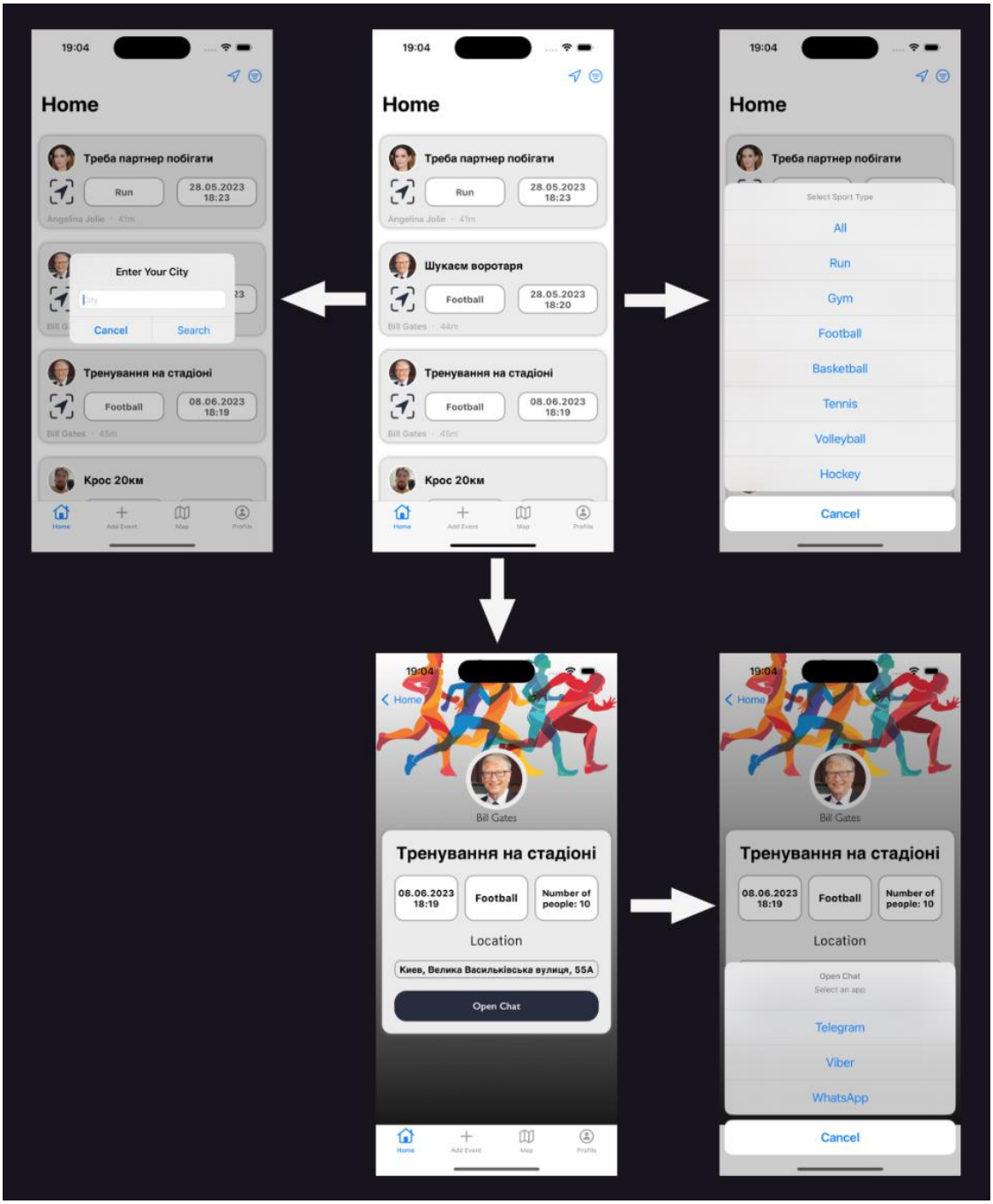


Рисунок 3.2 – Головна сторінка та екран детальної інформації

Для створення нової спортивної події, користувач натискає вкладку "Add Event" і заповнює необхідну інформацію, таку як назва, дата, час, місце проведення, тип спорту та кількість учасників. Також користувач може вибрати локацію на мапі, що видно на рисунку 3.3. Після введення всіх

даних, він натискає кнопку "Done" і подія додається до списку та відображається на мапі.

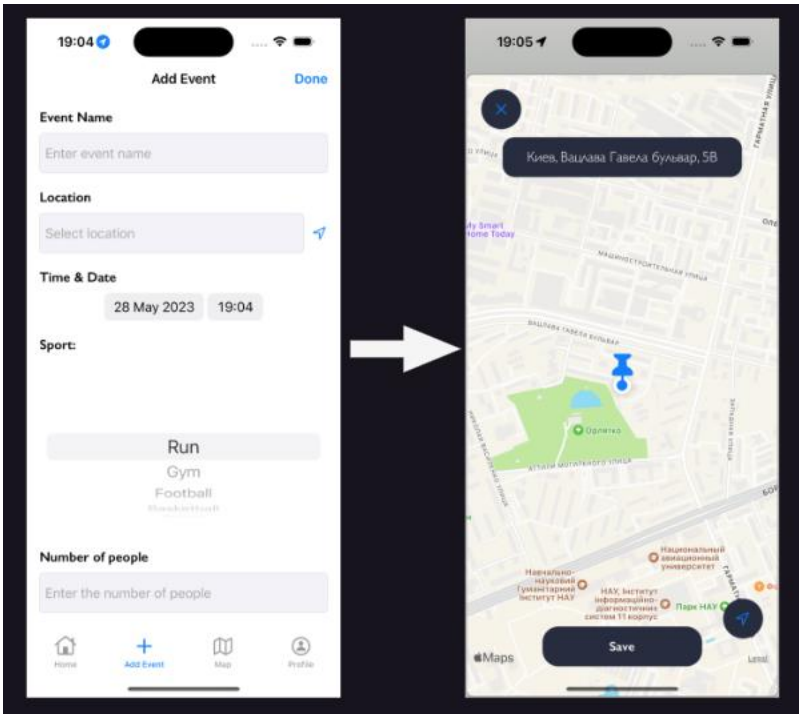


Рисунок 3.3 – Екран створення нової події

Користувач може переглянути спортивні події на мапі, натиснувши на відповідну вкладку "Map", як можна побачити на рисунку 3.4. Після цього на карті відображаються місця проведення подій.

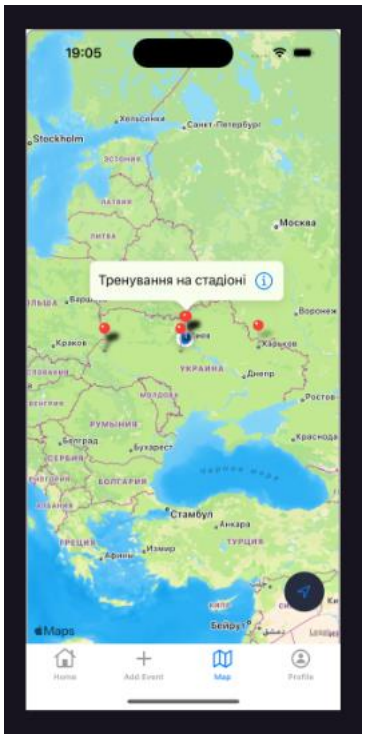


Рисунок 3.4 – Екран мапи із актуальними подіями

Користувач може переглянути власний профіль та особисті події, а також видалити їх, якщо вони вже неактуальні. Детально зображено на рисунку 3.5. Для того щоб деавторизуватися, треба натиснути на відповідну кнопку в правому верхньому кутку та перейти на екран авторизації.

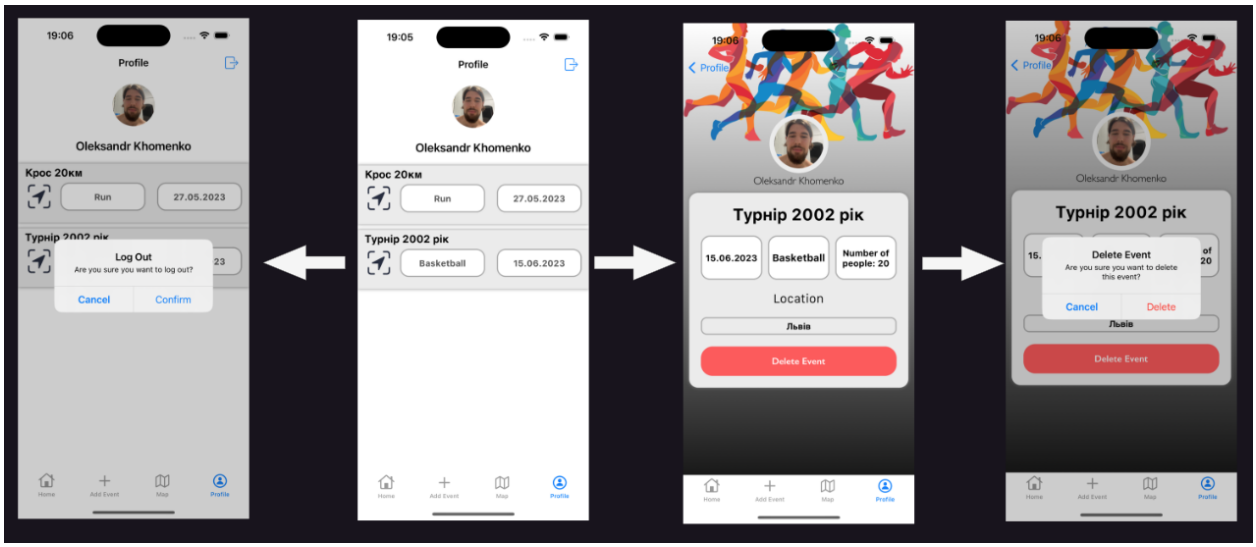


Рисунок 3.5 – Екран особистого профілю користувача

Ці кроки надають користувачу можливість орієнтуватися та використовувати основні функції застосунку для організації та планування спільних спортивних подій.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2023 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ ТА ПЛАНУВАННЯ СПІЛЬНИХ СПОРТИВНИХ ПОДІЙ

Графічний матеріал

КПІ.ІТ-9428.045490.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Юлія КРАМАР

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Олександр ХОМЕНКО

Київ – 2023

КПІ.ІТ-9428.045490.06.99.СБД

Зм.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-9428.045490.06.99.СБД			
Розроб.		Хоменко О.С.			Схема бази даних	Лит.	Арк.	Аркушів
Перев.		Крамар Ю.М.					1	
Т. Кон.						Аркуш	Аркушів	
Н. Кон.		Вітковська І.І.			Мобільний застосунок для організації та планування спільних спортивних подій	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-94		
Затв.		Крамар Ю.М.						

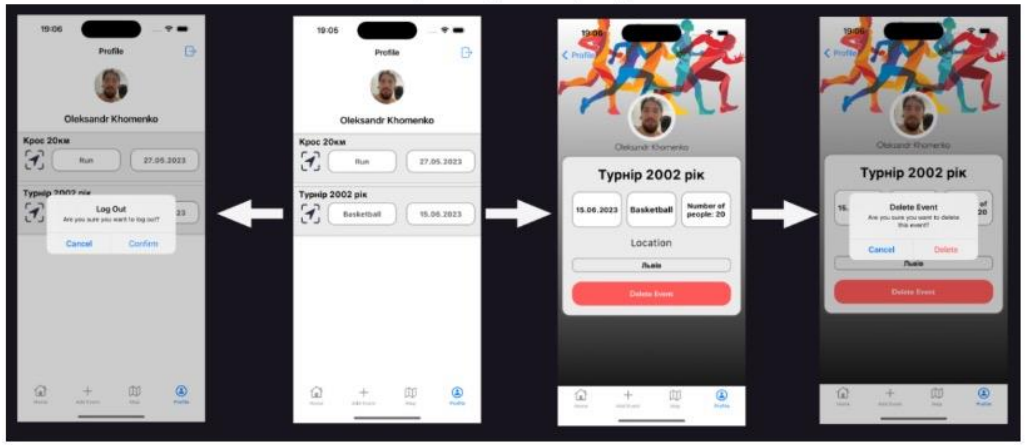
КПІ.ІТ-9428.045490.07.99.ССК

					КПІ.ІТ-9428.045490.07.99.ССК
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення
Розробив		Хоменко О.С.			
Перевірив		Крамар Ю.М.			
Т. кон.					
Н. кон.		Вітковська І.І.			Мобільний застосунок для організації та планування спільних спортивних подій
Затвердив		Крамар Ю.М.			

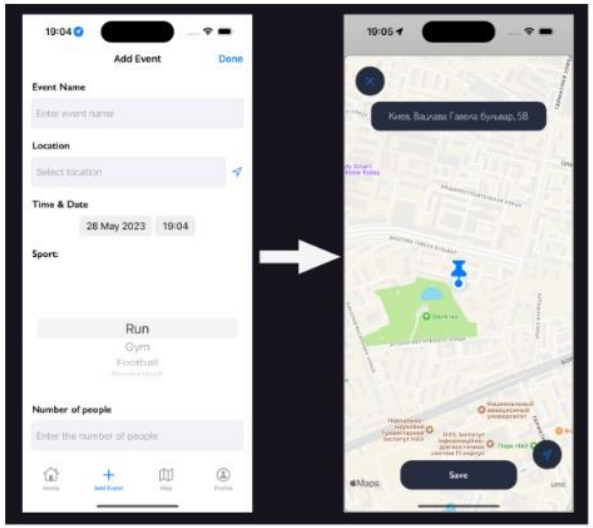
Літера	Маса	Масштаб
Аркуш	Аркушів	
КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-94		

КПІ.ІТ-9428.045490.08.99.КЕ

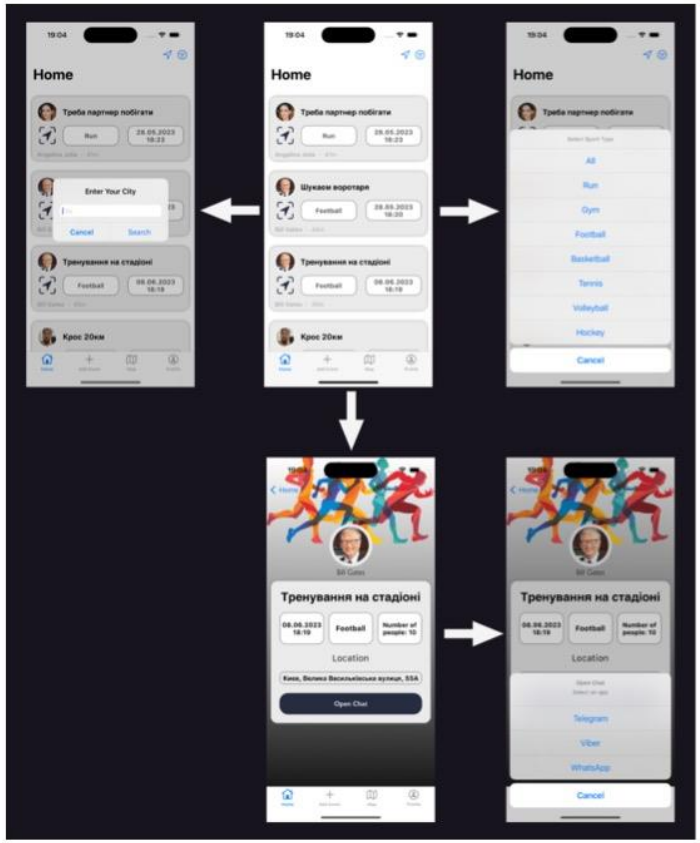
Екран сторінки користувача



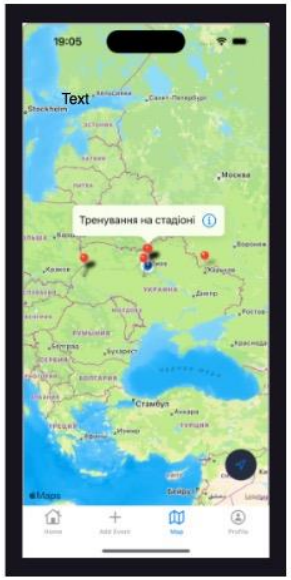
Екран створення нової події



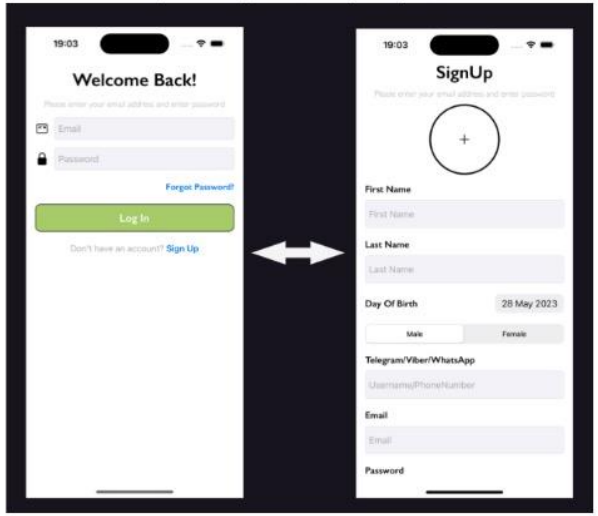
Головний екран із списком подій та детальною інформацією



Екран із подіями на мапі



Екран авторизації та реєстрації



					КПІ.ІТ-9428.045490.08.99.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата		Літера / Маса / Масштаб
Розробив		Хоменко О.С.			Креслення вигляду екранних форм	
Перевірив		Крамар Ю.М.				Аркуш / Аркушів
Т. кон.						
Н. кон.		Вітковська І.І.			Мобільний застосунок для організації та планування спільних спортивних подій	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-94
Затвердив		Крамар Ю.М.				