

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 ЖАРИКОВ
(підпис) (ім'я прізвище)
« » _____ 2025 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 персоналізованих рекомендацій
культурних подій

Виконав студент IV курсу, групи ІП-15
(шифр групи)
Кондрацька Соня Леонідівна
(прізвище, ім'я, по батькові) _____ (підпис)

Керівник ст.викладач, Ковтунець О. В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) _____ (підпис)

Консультант асистент, Шулькевич Т. В.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) _____ (підпис)

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

Засвідчую, що у цьому дипломному проєкті
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____
(підпис)

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 ЖАРИКОВ
(підпис) (ім'я прізвище)

“ ” _____ 2025 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Кондрацькій Соні Леонідівні

(прізвище, ім'я, по батькові)

1. Тема проєкту Мобільний застосунок для персоналізованих рекомендацій культурних подій

керівник проєкту Ковтунець Олесь Володимирович, ст.викладач

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «23» травня 2025 р. № 1705-с

2. Термін подання студентом проєкту «16» червня 2025 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки

1) Предпроєктне обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, опис бізнес-процесів, постановка задачі.

2) Розроблення вимог до програмного забезпечення: варіанти використання, аналіз системних вимог, розроблення функціональних вимог, розроблення нефункціональних вимог.

3) Конструювання та розроблення програмного забезпечення: Архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення.

4) Аналіз якості та тестування програмного забезпечення: аналіз якості ПЗ, опис процесів тестування, опис контрольного прикладу.

5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна компонентів програмного забезпечення

3) Схема бази даних

6. Консультанти розділів проєкту

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

7. Дата видачі завдання «15» березня 2025 року

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	15.03.2025	
2	Аналіз існуючих методів розв'язання задачі	21.03.2025	
3	Постановка та формалізація задачі	30.03.2025	
4	Розробка інформаційного забезпечення	09.04.2025	
5	Алгоритмізація задачі	18.04.2025	
6	Обґрунтування вибору використаних технічних засобів	29.04.2025	
7	Розробка програмного забезпечення	07.05.2025	
8	Налагодження програми	18.05.2025	
9	Виконання графічних документів	23.05.2025	
10	Оформлення пояснювальної записки	28.05.2025	
11	Подання ДП на попередній захист	04.06.2025	
12	Подання ДП рецензенту	10.06.2025	
13	Подання ДП на основний захист	16.06.2025	

Студент

(підпис)

Соня КОНДРАЦЬКА

(ініціали, прізвище)

Керівник

(підпис)

Олесь КОВТУНЕЦЬ

(ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п'яти розділів, містить 36 таблиць, 28 рисунків та 27 джерел – загалом 77 сторінок.

Дипломний проєкт присвячений розробці мобільного застосунку Festigo, який надає користувачам персоналізовані рекомендації культурних подій з урахуванням їхніх інтересів, геолокації та зручного часу відвідування.

Метою роботи є підвищення рівня поінформованості про культурне життя в Україні та спрощення доступу до актуальних подій за допомогою інтелектуальної рекомендаційної системи.

У розділі “Передпроєктне обстеження предметної області” проаналізовано предметну область, проаналізовано відомі продукти та технічні рішення, описано бізнес-процеси та сформульовано постановку задачі.

Розділ “Розроблення вимог до програмного забезпечення” присвячений визначенню варіантів використання програмного забезпечення, аналізу системних вимог, розробленню функціональних та нефункціональних вимог.

У розділі “Конструювання та розроблення програмного забезпечення” визначено архітектуру програмного забезпечення, обґрунтовано вибір засобів розробки, проведене конструювання програмного забезпечення, а також проаналізовано безпеку даних.

У розділі “Аналіз якості та тестування програмного забезпечення” проведено аналіз якості програмного забезпечення, описано процеси тестування та наведено контрольний приклад.

Розділ “Розгортання та супровід програмного забезпечення” присвячений процесам встановлення, налаштування та підтримки програмного забезпечення після його розробки.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, КУЛЬТУРНІ ПОДІЇ, ПЕРСОНАЛІЗОВАНІ РЕКОМЕНДАЦІЇ, ПОШУК, SWIFT, PYTHON, GOOGLE FIREBASE, БАЗА ДАНИХ, XCODE.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 36 tables, 28 figures and 27 sources – in total 77 pages.

The diploma project is dedicated to developing the Festigo mobile application, which provides users with personalized recommendations for cultural events, taking into account their interests, geolocation, and convenient time of visit.

The work aims to increase awareness of cultural life in Ukraine and simplify access to current events using an intelligent recommendation system.

The “Pre-project survey of the subject area” section analyzes the subject area, analyzes known products and technical solutions, describes business processes, and formulates the problem statement.

The “Development of software requirements” section is devoted to determining software usage options, analyzing system requirements, and developing functional and non-functional requirements.

The section “Design and development of software” defines the software architecture, justifies the choice of development tools, carries out software design, and analyzes data security.

The “Software quality analysis and testing” section analyzes software quality, describes testing processes, and provides a control example.

The “Software Deployment and Maintenance” section is dedicated to the processes of installing, configuring, and maintaining software after it is developed.

KEYWORDS: MOBILE APPLICATION, CULTURAL EVENTS, PERSONALIZED RECOMMENDATIONS, SEARCH, SWIFT, PYTHON, GOOGLE FIREBASE, DATABASE, XCODE

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

**МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПЕРСОНАЛІЗОВАНИХ
РЕКОМЕНДАЦІЙ КУЛЬТУРНИХ ПОДІЙ**

Технічне завдання

КПІ.ІІ-1510.045490.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олесь КОВТУНЕЦЬ

Нормоконтроль:

_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:

_____ Соня КОНДРАЦЬКА

Київ – 2025

ЗМІСТ

1	НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ	4
2	ПІДСТАВА ДЛЯ РОЗРОБКИ	5
3	ПРИЗНАЧЕННЯ РОЗРОБКИ	6
4	ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	7
4.1	Вимоги до функціональних характеристик	7
4.1.1	Користувацького інтерфейсу	7
4.1.2	Облік користувачів:	10
4.1.2.1	Для гостя:	10
4.1.2.2	Для користувача:	11
4.1.2.3	Для адміністратора системи:	11
4.1.2.4	Додаткові вимоги:	11
4.2	Вимоги до надійності	11
4.3	Умови експлуатації	11
4.3.1	Вид обслуговування	11
4.3.2	Обслуговуючий персонал	11
4.4	Вимоги до складу і параметрів технічних засобів	12
4.5	Вимоги до інформаційної та програмної сумісності	12
4.5.1	Вимоги до вхідних даних	12
4.5.2	Вимоги до вихідних даних	12
4.5.3	Вимоги до мови розробки	12
4.5.4	Вимоги до середовища розробки	12
4.5.5	Вимоги до представленню вихідних кодів	12
4.6	Вимоги до маркування та пакування	13
4.7	Вимоги до транспортування та зберігання	13
4.8	Спеціальні вимоги	13
5	ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ	14

	3
5.1 Попередній склад програмної документації	14
5.2 Спеціальні вимоги до програмної документації	14
6 СТАДІЇ І ЕТАПИ РОЗРОБКИ	15
7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ	16

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Мобільний застосунок для персоналізованих рекомендацій культурних подій.

Галузь застосування:

Наведене технічне завдання поширюється на розробку мобільного застосунку під платформу iOS – “Festigo” [045490], котрий використовується для який використовується для пошуку, перегляду та отримання персоналізованих рекомендацій культурних заходів. Додаток орієнтований на широке коло користувачів, зацікавлених в активному культурному житті, зокрема відвідувачів концертів, театральних вистав, фестивалів, виставок та інших подій.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки мобільного застосунку для персоналізованих рекомендацій культурних подій є завдання на дипломне проектування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для пошуку, перегляду та отримання рекомендацій культурних заходів відповідно до інтересів і локації користувача, із можливістю збереження улюблених заходів та отримання сповіщень про актуальні події.

Метою розробки є автоматизація пошуку культурних подій та підвищення їх релевантності завдяки персоналізованим рекомендаціям.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функцій:

4.1.1 Користувацького інтерфейсу

- Сторінки для реєстрації та авторизації (рисунки 4.1 та 4.2).

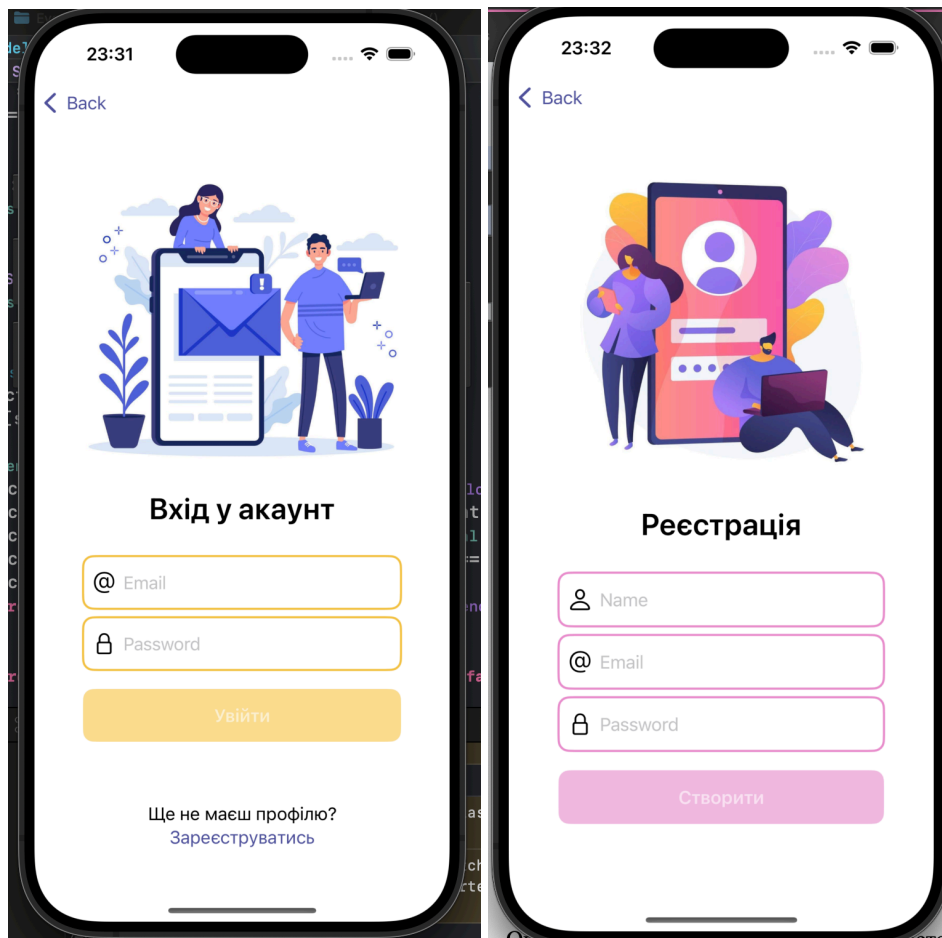


Рисунок 4.1 і 4.2 – Форми реєстрації та авторизації

- Опитування під час першого входу у застосунок для первинного налаштування профілю (рис. 4.3).

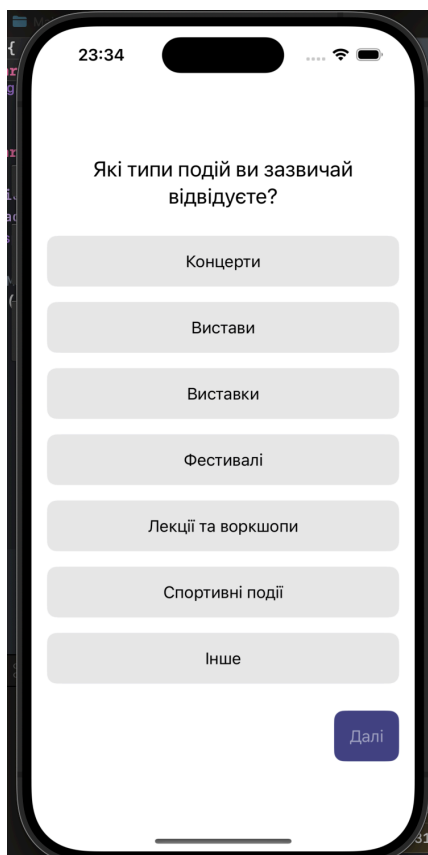


Рисунок 4.3 – Сторінка з опитуванням

– Головна сторінка, що відображує усі події з можливістю фільтрації/пошуку, та рекомендації (рис. 4.4).

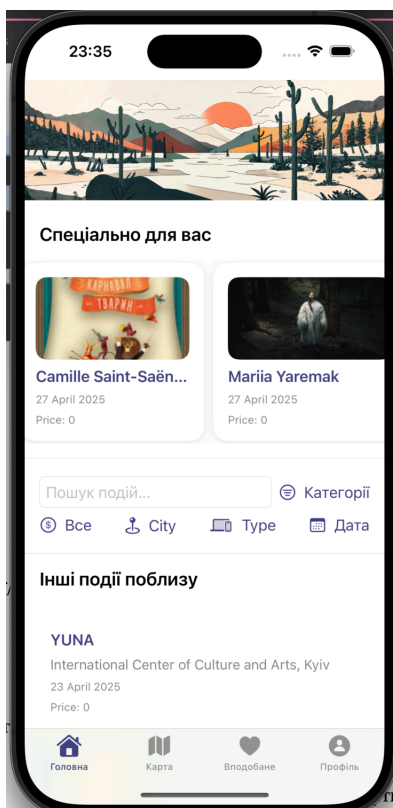


Рисунок 4.4 – Головна сторінка

- Сторінка для відображення подій на інтерактивній карті з можливістю натискання на окремі події для перегляду деталей (рис. 4.5).

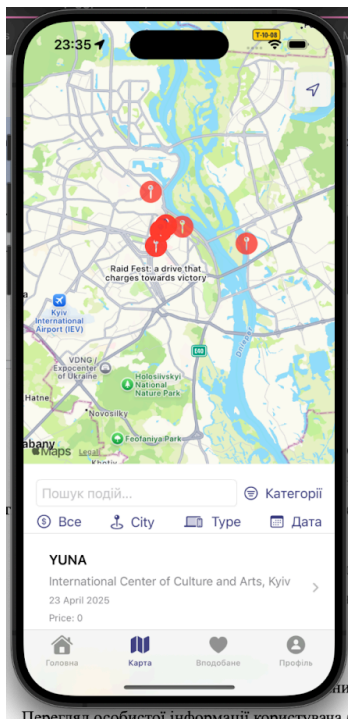


Рисунок 4.5 – Сторінка карти подій

- Сторінка з переліком заходів, які користувач додав до улюбленого (рис. 4.6).

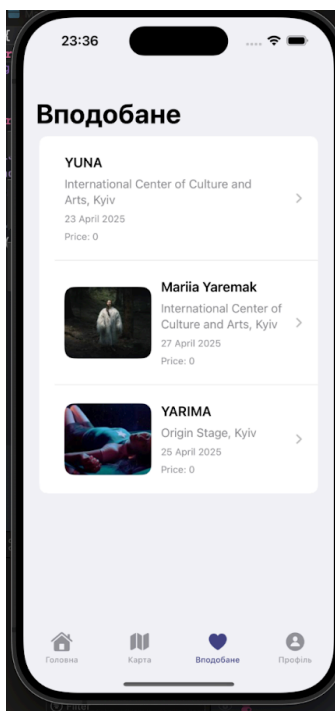


Рисунок 4.6 – Сторінка обраних подій

- Перегляд особистої інформації користувача (рис. 4.7).

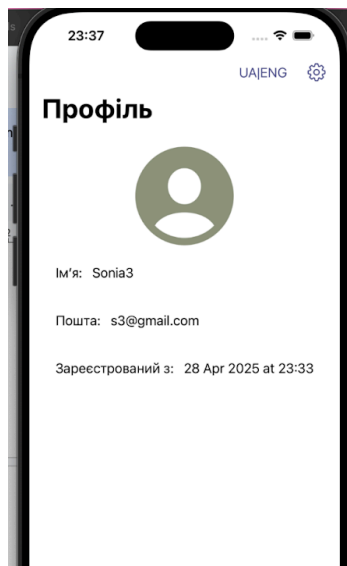


Рисунок 4.7 – Сторінка профілю користувача

- Сторінка з детальною інформацією про захід, додаванням до обраного або переходу на сторінку покупки квитків (рис. 4.8).

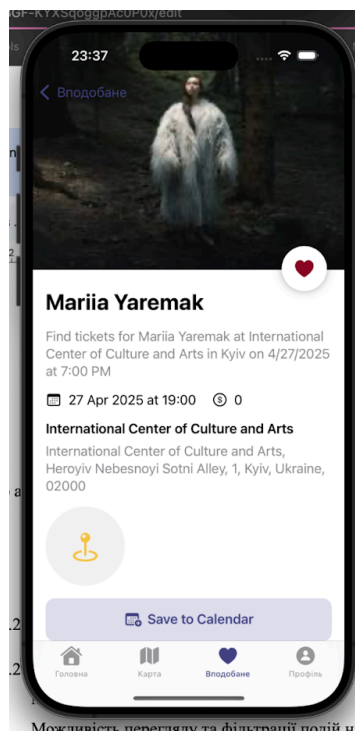


Рисунок 4.8 – Сторінка перегляду події

4.1.2 Облік користувачів:

4.1.2.1 Для гостя:

- Можливість реєстрації/авторизації.
- Можливість перегляду та фільтрації подій на інтерактивній мапі.

- Перегляд детальної інформації про подію.

4.1.2.2 Для користувача:

- Проходження опитування для налаштування профілю інтересів.
- Перегляд персоналізованих рекомендацій подій.
- Перегляд усіх доступних подій з можливістю фільтрації.
- Можливість перегляду та фільтрації подій на інтерактивній мапі.
- Перегляд детальної інформації про подію.
- Додавання заходів до обраного.
- Додавання заходів до Google Calendar.
- Налаштування отримання сповіщень.

4.1.2.3 Для адміністратора системи:

Не передбачений.

4.1.2.4 Додаткові вимоги:

- Підтримка інтеграції з обліковим записом Google для реєстрації користувачів у системі.
- Інтеграція з системою сповіщень iOS.
- Використання геолокації для пошуку заходів.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

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

Програмне забезпечення повинно функціонувати на мобільних пристроях, що підтримують операційну систему IOS.

Мінімальна конфігурація технічних засобів:

- тип процесору: Apple A15 Bionic.
- об'єм ОЗП: 4 Гб;
- підключення до мережі Інтернет зі швидкістю від 20 мегабіт;
- вільний об'єм пам'яті: 200 мб;

Рекомендована конфігурація технічних засобів:

- тип процесору: Apple A15 Bionic і вище.
- об'єм ОЗП: 4 Гб;
- підключення до мережі Інтернет зі швидкістю від 30 мегабіт;
- вільний об'єм пам'яті: 128 гб;

4.5 Вимоги до інформаційної та програмної сумісності

Підтримка пристроїв на базі iOS 17 і вище.

4.5.1 Вимоги до вхідних даних

Вимоги до вхідних даних не висуваються.

4.5.2 Вимоги до вихідних даних

Вимоги до вихідних даних не висуваються.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Swift із використанням фреймворку SwiftUI. Для серверної частини розробки використовувати мову Python.

4.5.4 Вимоги до середовища розробки

Розробку виконати на за допомогою середовища XCode.

4.5.5 Вимоги до представлення вихідних кодів

Вихідний код програми має бути представлений у вигляді файлів із розширенням .swift та .py.

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

Спеціальні вимоги до програмного продукту не висуваються.

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 структурна варіантів використання;
- схема структурна компонентів програмного забезпечення;
- схема таблиць бази даних;
- 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 проєкту	21.02	
2.	Розробка технічного завдання	15.03	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 проєкту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проєкту	20.05	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	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 персоналізованих рекомендацій
культурних подій**

КП.ІП-1510.045490.02.81

Київ – 2025

ЗМІСТ

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ	6
1.1 Постановка завдання дипломного проєктування	6
1.2 Аналіз предметної області	6
1.3 Аналіз існуючих рішень	8
1.3.1 Аналіз відомих програмних продуктів	8
1.3.2 Аналіз відомих алгоритмічних та технічних рішень	12
1.4 Аналіз та моделювання бізнес-процесів	16
Висновки до розділу	19
2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	21
2.1 Варіанти використання програмного забезпечення	21
2.2 Розроблення функціональних вимог	28
2.3 Розроблення нефункціональних вимог	31
2.4 Аналіз системних вимог	33
2.5 Аналіз економічних показників програмного забезпечення	34
2.6 Постановка завдання на розробку програмного забезпечення	37
Висновки до розділу	38
3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	39
3.1 Архітектура програмного забезпечення	39
3.2 Архітектурні рішення та обґрунтування вибору засобів розробки	44
3.3 Конструювання програмного забезпечення	46
3.3.1 Розробка рекомендаційного алгоритму	46
3.3.2 Опис структури бази даних	49
3.4 Аналіз безпеки даних	52
Висновки до розділу	53
4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	54
4.1 Аналіз якості ПЗ	54
4.2 Опис процесів тестування	56

	3
4.3 Опис контрольного прикладу	62
Висновки до розділу	66
5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	68
5.1 Розгортання програмного забезпечення	68
5.1.1 Розгортання серверної частини	68
5.1.2 Розгортання мобільного застосунку	68
5.2 Супровід програмного забезпечення	69
Висновки до розділу	70
ВИСНОВКИ	72
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	74
ДОДАТОК А ЗВІТ ПОДІБНОСТІ	77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	– Integrated Development Environment – інтегроване середовище розробки.
API	– Application programming interface, прикладний програмний Інтерфейс.
SDK	– Software Development Kit – комплект засобів розробки програмного забезпечення.
JSON	– JavaScript Object Notation – формат обміну даними, заснований на JavaScript.
ER	– Entity-Relationship Diagram – діаграма "сутність-зв'язок".
OC	– Операційна система.
БД	– База даних.
ПЗ	– Програмне забезпечення.
REST	– Representational State Transfer – архітектурний стиль веб сервісів.
XSS	– Cross-Site Scripting.
BPMN	– Business Process Model and Notation – модель та нотація бізнес-процесів.
SVM	– Support Vector Machine.
TF-IDF	– Term Frequency-Inverse Document Frequency.

ВСТУП

Багатьом знайома ситуація, коли у вільний час виникає бажання відвідати культурний захід, однак знайти актуальну та релевантну інформацію виявляється складно. Зазвичай ця проблема виникає через відсутність централізованих платформ для зручного пошуку або, навпаки, через перевантаженість інформаційних джерел. Через це багато заходів залишаються поза увагою, а люди втрачають можливість цікаво провести час і долучитися до культурного життя.

Актуальність розробки обумовлена важливістю культурної сфери для розвитку суспільства, а також браком зручних інструментів інформування, що враховують інтереси користувачів. Відсутність таких рішень ускладнює доступ до інформації та знижує залученість до культурних подій, що створює потребу в ефективних технологічних рішеннях. Зокрема, мобільні застосунки з персоналізованими рекомендаціями можуть суттєво покращити ситуацію. Світовий досвід демонструє високу ефективність рекомендаційних алгоритмів у таких сферах, як електронна комерція, потокове відео та музика, проте культурна сфера досі залишається недостатньо охопленою подібними підходами.

З цією метою пропонується розробка мобільного застосунку, який автоматизує та спрощує пошук культурних подій шляхом надання персоналізованих рекомендацій з урахуванням інтересів користувача, його місцезнаходження та часових вподобань.

Призначенням цієї розробки є створення зручного, адаптивного інструменту для залучення широкого кола користувачів до культурного життя та підвищення їх поінформованості про локальні події.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання дипломного проєктування

Дипломне проєктування передбачає виконання наступних завдань:

- аналіз предметної області та опис її ключових бізнес-процесів, визначення загального завдання розробки у рамках ДП;
- аналіз існуючих рішень (як програмних продуктів, так і алгоритмічних чи технічних рішень) обраного завдання розробки у рамках ДП;
- аналіз та моделювання бізнес-процесів;
- розроблення функціональних, нефункціональних та системних вимог до програмного забезпечення;
- аналіз економічних показників програмного забезпечення ДП;
- постановка завдання на розробку програмного забезпечення ДП;
- розроблення архітектури програмного забезпечення;
- розроблення архітектурних рішень та обґрунтування вибору засобів розробки програмного забезпечення;
- конструювання та розроблення програмного забезпечення;
- аналіз безпеки даних програмного забезпечення;
- аналіз якості та тестування програмного забезпечення;
- опис розгортання та супроводу програмного забезпечення;
- створення супроводжувальної документації до розробленого програмного забезпечення.

1.2 Аналіз предметної області

У сучасному інформаційному суспільстві людина щоденно піддається впливу великої кількості подій, новин та інформаційних потоків. Незважаючи на різноманіття культурних заходів, вибір найбільш релевантних перетворюється на складну задачу, особливо для туристів або мешканців великих міст, через обмежений вільний час, нестачу актуальної інформації

або відсутність зручних інструментів пошуку. Традиційні методи, зокрема фільтрація за категоріями або ключовими словами, не завжди забезпечують необхідний рівень персоналізації [1]. Як наслідок, потенційні відвідувачі можуть пропустити цікаві для них події, що обмежує їхню участь у культурному житті та знижує ефективність промоції культурного продукту.

Рішенням цієї проблеми є використання рекомендаційних систем – інструментів, які, аналізуючи інтереси конкретного користувача, його поведінку та інші характеристики, пропонують індивідуально підібраний контент [2]. Вони активно використовуються у сфері електронної комерції та медіа (наприклад, Netflix, Spotify, Amazon, тощо) [3]. У культурній сфері застосування таких систем лише починає набувати популярності, хоча потенціал їхнього використання є дуже високим. Особливістю подій як об'єктів рекомендації є їхня обмеженість у часі та просторі, що вимагає адаптації традиційних алгоритмів до специфіки цієї галузі.

На сьогодні вже існують деякі програмні реалізації, включаючи, веб-платформи та мобільні застосунки, які частково вирішують розглянуту проблему. Вони дозволяють користувачам переглядати події за категоріями, зберігати обране або ділитися заходами з друзями. Водночас більшість таких систем мають вигляд каталогів і не реалізують механізми персоналізації, тобто не враховують поведінкову аналітику користувача, історію його уподобань, поточне місцезнаходження чи індивідуальні захоплення.

Бізнес-процеси в цій сфері, як правило, передбачають: реєстрацію користувача у систему, агрегацію інформації про події (з використанням відповідного API чи напряду з сайтів організаторів, афіш, соціальних мереж), модерацію контенту, візуалізацію подій для користувача, їх перегляд та взаємодія, взаємодію з системами бронювання або купівлі квитків, а також просування заходів через інструменти цифрового маркетингу.

У рамках даного дипломного проєкту було обрано комплексний підхід, який передбачає створення мобільного застосунку, що поєднує ці функції агрегації культурних подій, персоналізованої системи рекомендацій, інтерактивної карти та геолокаційних сповіщень. Запропоноване рішення дозволяє не лише полегшити процес пошуку подій, але й зробити його інтуїтивним, адаптивним та корисним у повсякденному житті користувача.

1.3 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації мобільного застосунку для персоналізованих рекомендацій культурних подій – Festigo. Далі будуть розглянуті готові програмні рішення, допоміжні програмні засоби та засоби розробки.

1.3.1 Аналіз відомих програмних продуктів

У цьому підрозділі проводиться аналіз наявних програмних рішень у сфері пошуку культурних заходів та подій. Особлива увага приділяється їх функціональності, зручності використання та релевантності рекомендацій. Метою аналізу є виявлення сильних та слабких сторін існуючих продуктів для подальшого врахування їх у власній розробці.

Для проведення такого аналізу було обрано наступні сервіси: Eventbrite, Ticketmaster, Fever та Facebook Events. Їх вибір обумовлений широким визнанням, різноманітним функціоналом та популярністю серед користувачів у сфері організації заходів.

Eventbrite – є одним із найпопулярніших сервісів для організації та пошуку подій. Користувачі можуть шукати заходи за місцем розташування та різноманітними тематиками. На платформі є спеціальна сторінка, де користувач отримує рекомендації щодо подій, що можуть його зацікавити (рис. 1.1).

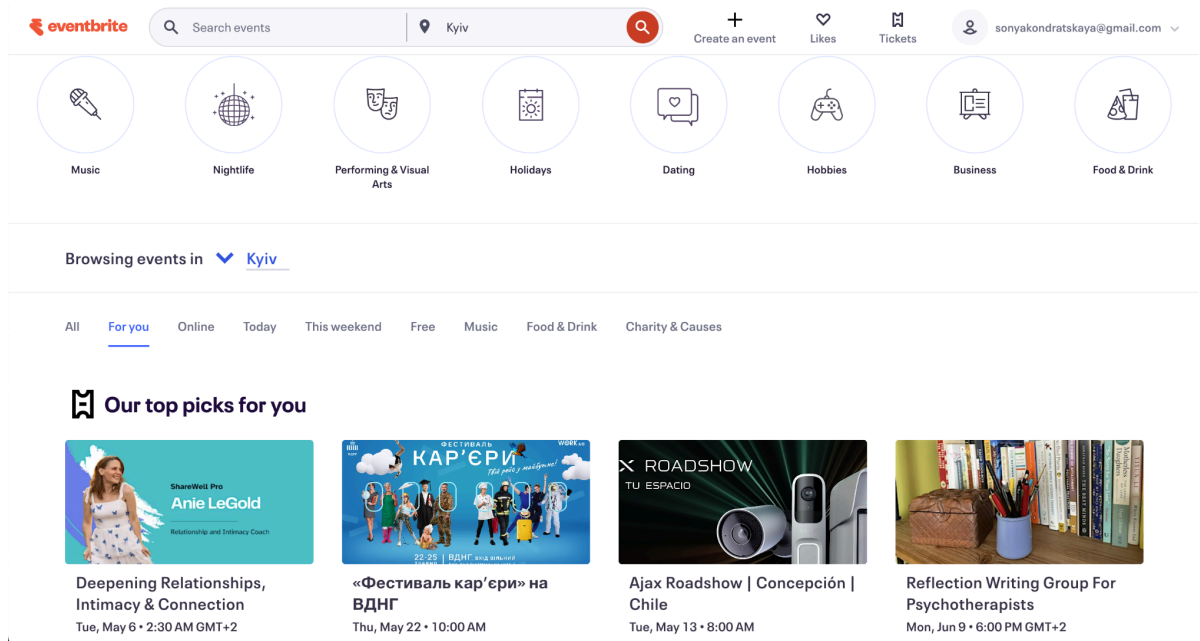


Рисунок 1.1 – Вигляд головної сторінки сайту Eventbrite.com [4]

Платформа Ticketmaster – в основному орієнтована на концерти, театральні вистави та спортивні події, з акцентом на продаж квитків. Однак система рекомендацій є досить базовою – переважно орієнтується на популярність подій у регіоні, без глибокого врахування індивідуальних уподобань користувача (рис. 1.2).

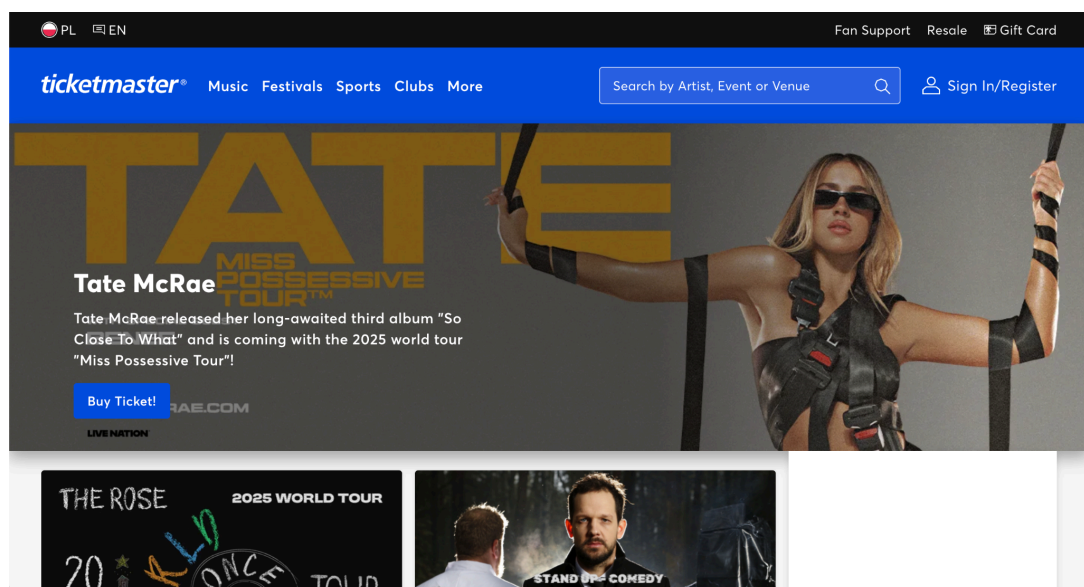


Рисунок 1.2 – Вигляд головної сторінки сайту Ticketmaster [5]

Fever – надає доступ до широкого переліку заходів із можливістю фільтрації за категоріями, локацією та часом. Персоналізованої системи рекомендацій на основі уподобань користувача немає, але на сторінці окремих подій пропонуються схожі заходи та пов'язані категорії. Хоча платформа охоплює велике різноманіття заходів, їх доступність залежить від конкретного міста або країни (рис. 1.3).

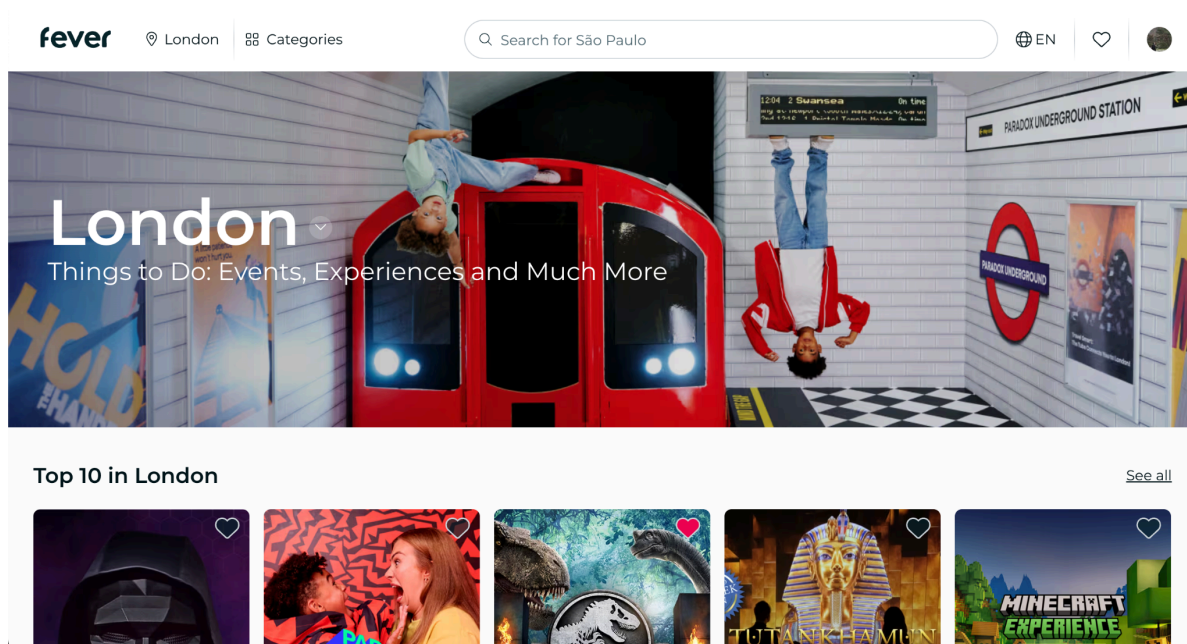


Рисунок 1.3 – Вигляд головної сторінки сайту Feverup.com [6]

Facebook Events – це окремий функціонал платформи Facebook, що дозволяє знаходити події, організовані іншими користувачами або організаціями. Користувачі можуть підписуватись на сторінки організаторів, ділитися цікавими подіями зі своїми друзями та бачити, які заходи відвідують або планують відвідати їхні знайомі, орієнтуючись на це при виборі (рис. 1.4).

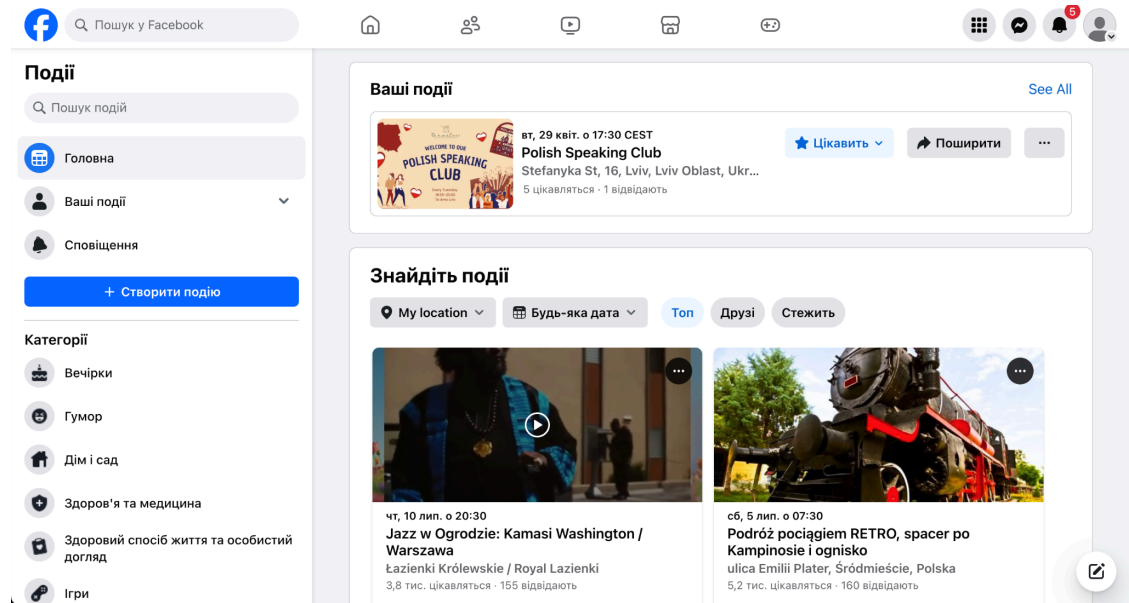


Рисунок 1.4 – Вигляд головної сторінки сайту Facebook Events [7]

Для порівняння дипломного проєкту з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Функціонал/ Критерій	Festi go	Event brite	Ticket master	Fever	Facebook Events	Пояснення
Персоналізовані рекомендації за інтересами	+	+ _	+	-	+ -	Відсутні у більшості аналогів
Наявність української локалізації	+	-	-	-	+ -	Більшість аналогів не підтримує українську мову
Покриття подіями в Україні	+	Тільк и Київ	-	-	+	Більшість аналогів не мають подій на території України
Урахування конкретної відстані до події	+	-	-	-	-	Більш точні рекомендації за локацією
Гнучкість налаштувань профілю	+	-	+	-	-	Можливість детального налаштування уподобань

1.3.2 Аналіз відомих алгоритмічних та технічних рішень

Рекомендаційний алгоритм

Ключовим елементом у розробці застосунку для рекомендації культурних заходів є вибір ефективного алгоритму персоналізації. У сфері рекомендаційних систем виділяють три основних підходи:

- фільтрація на основі контенту (Content-based Filtering) – зосереджується на аналізі характеристик об'єктів та їх зіставленні з відомими вподобаннями користувача. Для кожного об'єкта формується профіль, що базується на його описових ознаках. На основі позитивних оцінок, наданих користувачем, система будує персональний профіль уподобань, який використовується для пошуку та рекомендації схожих об'єктів. Однією з головних переваг цього підходу є незалежність від поведінки інших користувачів, що забезпечує високу персоналізацію рекомендацій. Крім того, метод добре функціонує навіть за умов обмеженої кількості даних [8]. Серед недоліків варто відзначити необхідність детального моделювання та структуризації характеристик об'єктів, що у випадку масштабних систем може потребувати значних ресурсів. Також фільтрація на основі контенту здебільшого обмежує користувача у відкритті нових елементів, адже пропонує переважно ті об'єкти, які є подібними до вже вподобаних. Але водночас, така система здатна швидко адаптуватися до змін у смаках користувача [9].
- колаборативна фільтрація (Collaborative Filtering) – рекомендації формуються на основі поведінки інших користувачів зі схожими інтересами. Наприклад, коли новий користувач реєструється, система співставляє його профіль із існуючими даними для знаходження "сусідів" – користувачів зі схожими вподобаннями. Надалі рекомендації генеруються, враховуючи оцінки або дії цих сусідів. Основними перевагами даного методу є те, що він не потребує жодних знань про характеристики об'єктів, а також здатен відкривати користувачам нові

інтереси. Проте його ефективність суттєво залежить від обсягу наявних даних, так як при нестачі інформації виникає проблема "холодного старту", коли для нових користувачів або нових об'єктів неможливо надати якісні рекомендації [9].

- гібридні методи (Hybrid Recommendation Systems) – поєднання перших двох підходів для досягнення кращої якості персоналізації. Існують різні способи інтеграції: змішування результатів, каскадне використання, підсилення характеристик або створення мета-рівневих моделей. Гібридні системи зазвичай демонструють підвищену точність і стійкість, проте їхня розробка є більш складною і потребує більших обчислювальних ресурсів [10].

У межах даної розробки в основі побудови системи рекомендацій буде використано саме фільтрацію на основі контенту. Проте класичний алгоритм потребує певних модифікацій, оскільки для підвищення релевантності рекомендацій необхідно враховувати додаткові динамічні параметри, характерні для культурних подій, зокрема час проведення заходу, відстань до місця події та інші індивідуальні характеристики, зазначені користувачем.

Архітектура

Архітектура програмного забезпечення – це набір структур, які визначають організацію системи. Вона включає компоненти програмного забезпечення, відношення та взаємодії між ними, а також властивості як самих компонентів, так і зв'язків між ними [19].

Існує декілька поширених архітектурних підходів, що використовуються в розробці сучасних інформаційних систем:

- монолітна архітектура – передбачає, що вся система розробляється як єдине ціле, що містить усі компоненти – це включає як інтерфейс користувача, так і обробку даних та бізнес-логіки. Цей підхід зручний на початкових етапах розробки, оскільки всі компоненти інтегровані в єдину програму. Проте з часом, коли система розростається, можуть

виникати труднощі з її масштабуванням, тестуванням та підтримкою. Оскільки всі частини додатка зв'язані в один моноліт, будь-яка зміна в одній частині може спричинити непередбачувані наслідки в інших [20].

- мікросервісна архітектура – передбачає розбиття програми на набір автономних сервісів, кожен з яких відповідає за виконання певної бізнес-логіки та взаємодіє з іншими через API. Кожен мікросервіс може мати свою власну базу даних і незалежно розвиватися, що дозволяє знижувати ризики, пов'язані з масштабуванням, та полегшує розподіл роботи між командами [20]. Проте для такого підходу необхідно розробляти інфраструктуру для забезпечення взаємодії між сервісами, а також зростає складність в управлінні та моніторингу великої кількості сервісів.
- клієнт-серверна архітектура – підхід, за якого система поділяється на дві частини: клієнт (користувацький інтерфейс) та сервер (обробка логіки, зберігання даних). Клієнт запитує ресурси або послуги у сервера через мережу. Сервер обробляє запити, зберігає дані в базі та відповідає клієнту. Клієнт-серверна модель легко масштабується як вертикально (тобто додавання потужностей сервера), так і горизонтально (додавання нових серверів), що дозволяє ефективно обробляти велику кількість запитів. Однак цей підхід має велику залежність від стабільної роботи мережі і сервера, а також виклики з безпекою, оскільки всі дані зберігаються централізовано на сервері [21].

Крім загальної архітектури застосунку, важливо також визначити архітектуру рівня інтерфейсу користувача, оскільки вона безпосередньо відповідає за взаємодію користувача із системою. Найбільш поширеними є такі архітектурні патерни:

- MVC (Model-View-Controller) – класична архітектура, яка розділяє модель даних, інтерфейс користувача і логіку керування. Хоча вона є

простою у впровадженні, з часом у складних застосунках контролери можуть ставати надмірно великими, що ускладнює підтримку.

- MVP (Model-View-Presenter) – покращена версія MVC з винесенням логіки у Presenter. Це підвищує тестованість та організованість коду, проте збільшує кількість допоміжних інтерфейсів [13].
- MVVM (Model-View-ViewModel) – архітектурний підхід, який забезпечує чітке розділення обов'язків між моделлю, представленням та логікою стану, що може значно підвищити ефективність роботи [14]. Особливо добре інтегрується з декларативним стилем SwiftUI завдяки механізмам зв'язування даних та спостереження за змінами.
- VIPER – складна архітектура, яка розділяє застосунок на п'ять компонентів (View, Interactor, Presenter, Entity, Router), орієнтована на великі, багатофункціональні застосунки. Її використання виправдане для корпоративних проєктів, але є надмірним для невеликих і середніх застосунків [15].

З огляду на характер і масштаб дипломного проєкту, для розробки було обрано архітектуру MVVM, яка природно поєднується із фреймворком SwiftUI, сприяє кращій модульності, спрощує тестування окремих компонентів і покращує підтримку коду. Це дозволяє ефективно управляти станами інтерфейсу, організувати чисту бізнес-логіку і спрощувати обробку взаємодії користувача із застосунком.

А оскільки застосунок працює з динамічними даними, передбачається використання клієнт-серверної архітектури для обміну інформацією із зовнішнім сервером, написаним мовою Python, через API. Це дозволить своєчасно оновлювати контент у застосунку та реалізувати персоналізовані алгоритми рекомендацій.

Щодо збереження даних, у даному проєкті було обрано Firebase Cloud Firestore як основну базу даних. Це рішення забезпечує ефективне зберігання та оновлення великої кількості даних про культурні події. Firestore надає

можливість синхронізації даних у реальному часі, що дозволяє користувачам отримувати актуальну інформацію без затримок. Крім того, цей сервіс автоматично масштабується відповідно до навантаження, що забезпечує стабільну роботу додатку незалежно від обсягу даних або кількості користувачів [16].

1.4 Аналіз та моделювання бізнес-процесів

При аналізі предметної області даного дипломного проєкту було виділено основні бізнес-процеси, що є критичними для функціонування додатку, а саме:

- реєстрація та аутентифікація
- початкове налаштування профілю та вподобань
- пошук та фільтрація усіх доступних подій
- перегляд рекомендацій
- взаємодія з подіями

Для моделювання бізнес-процесів було обрано нотацію BPMN як найбільш відповідну. Діаграми бізнес-процесів наведені на рисунках 1.5 - 1.8.

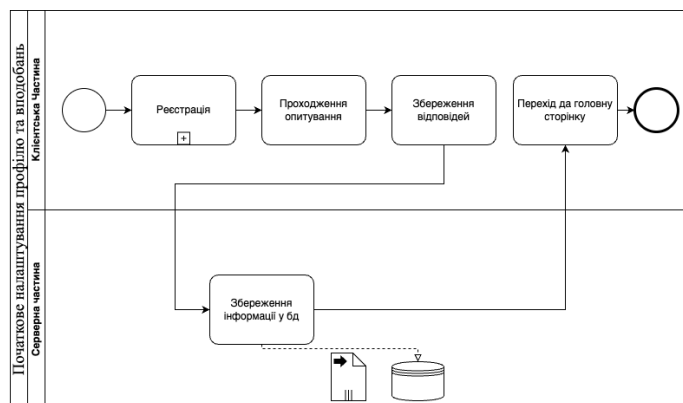


Рисунок 1.5 – Модель бізнес-процесу початкового налаштування профілю та вподобань

Опис моделі бізнес-процесу початкового налаштування профілю та вподобань:

- користувач заходить у застосунок та переходить на сторінку реєстрації;
- користувач проходить процес реєстрації.
- при успішній реєстрації – створюється обліковий запис у базі даних;
- користувач переходить до вибору інтересів;
- після вибору зберігаються уподобання до профілю користувача;
- користувач потрапляє на головну сторінку додатку.

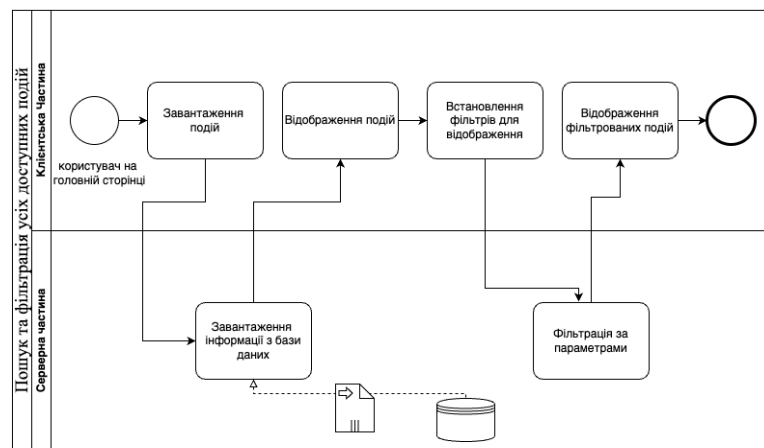


Рисунок 1.6 – Модель бізнес-процесу пошуку та фільтрації подій

Опис моделі бізнес-процесу пошуку та фільтрації подій:

- користувач знаходиться на головній сторінці;
- інформація про події завантажується з бд та відображається у застосунку;
- користувач обирає параметри фільтрації (категорія, локація, дата);
- на основі запиту формується фільтр та фільтруються події;
- користувач переглядає список знайдених подій;

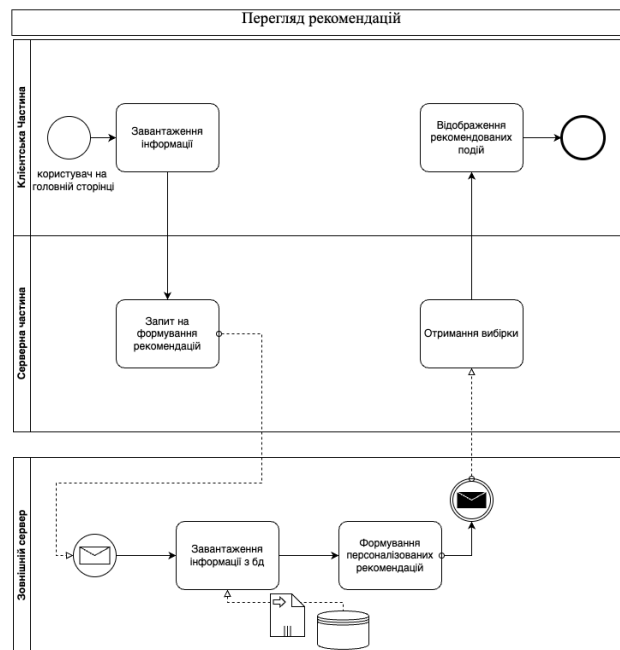


Рисунок 1.7 – Модель бізнес-процесу перегляду рекомендацій

Опис моделі бізнес-процесу перегляду рекомендацій:

- користувач знаходиться на головній сторінці;
- система ідентифікує користувача та робить запит на формування рекомендацій;
- зовнішній сервер отримує інформацію про інтереси користувача та наявні у бд події;
- формується колекція подій, що відповідають інтересам та повертається результат;
- користувач переглядає персоналізований список рекомендованих подій.

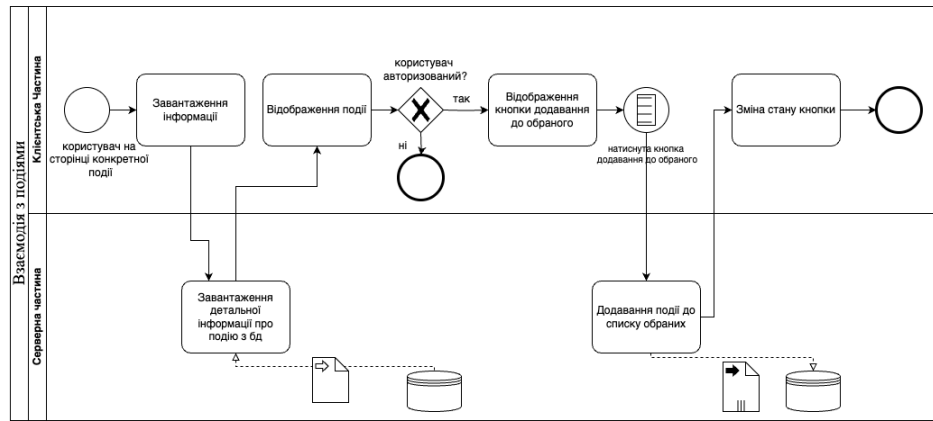


Рисунок 1.8 – Модель бізнес-процесу взаємодії з подіями

Опис моделі бізнес-процесу взаємодії з подіями:

- користувач обирає подію зі списку (рекомендованого або загального);
- відкривається сторінка події з детальним описом;
- якщо користувач авторизований, показати кнопку додавання до обраного;
- при натисканні на відповідну кнопку, подія зберігається у бд як “вподобана”.

Висновки до розділу

У першому розділі дипломного проєкту було здійснено передпроектне обстеження предметної області, яке дозволило чітко сформулювати проблематику, обґрунтувати актуальність проєкту та визначити основні вимоги до майбутнього програмного забезпечення. Було встановлено, що існуючі рішення для пошуку культурних заходів слабо враховують індивідуальні інтереси користувача.

Проведене порівняння аналогів показало, що більшість з них або обмежені у функціоналі, або не адаптовані до локального контексту (зокрема українського ринку). У результаті було запропоновано створення мобільного

застосунку Festigo, який поєднує персоналізовані рекомендації, підтримку української мови та актуальність подій для користувачів в Україні.

Було визначено доцільність використання алгоритму фільтрації на основі контенту з урахуванням специфіки культурних заходів, що дозволяє формувати рекомендації відповідно до інтересів користувача. З огляду на особливості цільової платформи – iOS – було обґрунтовано вибір архітектурного підходу MVVM, який забезпечує чітке розділення логіки представлення, бізнес-логіки та обробки даних. Інтеграція цієї архітектури з фреймворком SwiftUI дає змогу ефективно реалізувати динамічний та адаптивний інтерфейс користувача. Також було заплановано підключення до зовнішнього сервера, який відповідає за отримання, зберігання та оновлення даних про заходи і генерацію рекомендацій. Це дозволяє забезпечити централізоване управління даними, розширює можливості застосунку та підвищує його продуктивність. Крім того, було описано основні бізнес-процеси предметної області у нотації BPMN, що дозволило сформувати структуроване бачення функціональних вимог і сценаріїв взаємодії користувача із системою.

Таким чином, результати здійсненого аналізу формують надійну основу для розробки ефективного програмного забезпечення, яке відповідає актуальним потребам користувачів у галузі культурного дозвілля.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Діаграма варіантів використання розроблюваного мобільного застосунку з ключовими акторами та основними функціями наведена у графічному матеріалі, креслення 1.

В таблицях 2.1-2.13 наведено опис варіантів використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-01

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Гість (неzareєстрований користувач)
Trigger	Користувач бажає зареєструватися
Pre-conditions	-
Flow of Events	Користувач обирає спосіб реєстрації з двох варіантів: 1. Переходить на сторінку звичайної реєстрації. В поля вводяться відповідні дані: ім'я, email та пароль. Приймає умови користування і натискає кнопку "Зареєструватися". 2. Або користувач може зареєструватися в системі через обліковий запис Google. Після натискання кнопки "Google", користувач перенаправляється на сторінку авторизації Google, де йому пропонується увійти в свій обліковий запис.
Extension	У випадку введення некоректних даних, кнопка реєстрації стає неактивною
Post-Condition	Створення сторінки користувача, перехід на головну сторінку застосунку

Таблиця 2.2 – Варіант використання UC-02

Use case name	Авторизація користувача
Use case ID	UC-02
Goals	Отримання доступу до функціоналу зареєстрованого користувача
Actors	Гість
Trigger	Користувач бажає увійти до системи
Pre-conditions	Користувач попередньо зареєстрований у системі
Flow of Events	Користувач обирає спосіб авторизації з двох варіантів: 1. Переходить на сторінку звичайної авторизації. В поля вводяться відповідні дані: email та пароль. Натискає кнопку входу та відбувається перехід на головну сторінку застосунку. 2. Або користувач може увійти в систему через обліковий запис Google. Після натискання кнопки "Google", користувач перенаправляється на сторінку авторизації Google, де йому пропонується увійти в свій обліковий запис.
Extension	У випадку введення некоректних email чи паролю виводиться текст помилки автентифікації
Post-Condition	Користувач авторизований

Таблиця 2.3 – Варіант використання UC-03

Use case name	Налаштування профілю інтересів
Use case ID	UC-03
Goals	Надати інформацію про інтереси користувача
Actors	Авторизований користувач
Trigger	Користувач зареєструвався
Pre-conditions	Користувач зайшов у систему вперше

Продовження таблиці 2.3

Flow of Events	Користувач відповідає на серію питань щодо його уподобань та локації(може знайти своє місто вручну або дозволити автоматичне визначення геолокації), відповіді до яких зберігаються до його профілю
Extension	-
Post-Condition	Збережено інтереси користувача

Таблиця 2.4 – Варіант використання UC-04

Use case name	Перегляд подій на мапі
Use case ID	UC-04
Goals	Перегляд розташування подій на інтерактивній мапі
Actors	Авторизований користувач
Trigger	Користувач відкриває вкладку з мапою
Pre-conditions	Дозвіл на використання геолокації або наявність міста за замовчуванням у бд
Flow of Events	Користувач відкрив вкладку з мапою де відображається карта з пінгами подій та список цих подій. Користувач натискає на подію, переглядає деталі
Extension	Можливість фільтрації подій на мапі
Post-Condition	Переглянуто події на мапі

Таблиця 2.5 – Варіант використання UC-05

Use case name	Перегляд списку подій
Use case ID	UC-05
Goals	Перегляд подій у вигляді списку

Продовження таблиці 2.5

Actors	Авторизований користувач
Trigger	Користувач зайшов на головну сторінку
Pre-conditions	Авторизація. Події завантажені
Flow of Events	Користувач зайшов у застосунок. Відображається список подій. Користувач скролить та оглядає доступні події
Extension	Можливість фільтрації
Post-Condition	Події переглянуті

Таблиця 2.6 – Варіант використання UC-06

Use case name	Отримання рекомендацій
Use case ID	UC-06
Goals	Отримання списку подій на основі інтересів
Actors	Авторизований користувач
Trigger	Користувач зайшов на головну сторінку
Pre-conditions	Авторизація. Профіль інтересів заповнено
Flow of Events	Користувач зайшов у застосунок. Система формує та відображає список культурних подій на основі інтересів, історії та локації користувача. Користувач переглядає рекомендовані події.
Extension	-
Post-Condition	Отримано персоналізовані рекомендації

Таблиця 2.7 – Варіант використання UC-07

Use case name	Керування подіями
---------------	-------------------

Продовження таблиці 2.7

Use case ID	UC-07
Goals	Взаємодія з подією
Actors	Авторизований користувач
Trigger	Користувач зайшов на сторінку події
Pre-conditions	Авторизація
Flow of Events	Користувач знаходиться на сторінці конкретної події. Користувач може переглядати інформацію про подію, а також виконувати наступні дії: додавати/видаляти з обраного, ділитись з іншими, додавати у власний календар.
Extension	Користувач може перейти на сторінку-джерело по посиланню, або подивитись розташування на Google/Apple картах
Post-Condition	Переглянуто інформацію конкретної події

Таблиця 2.8 – Варіант використання UC-08

Use case name	Налаштування отримання сповіщень
Use case ID	UC-08
Goals	Користувач вмикає/вимикає push-сповіщення
Actors	Авторизований користувач
Trigger	Користувач відкриває налаштування
Pre-conditions	Авторизація
Flow of Events	Користувач переходить у налаштування зі сторінки профілю. Обирає, які повідомлення отримувати
Extension	Можливість вимкнути всі повідомлення
Post-Condition	Налаштування збережено

Таблиця 2.9 – Варіант використання UC-09

Use case name	Фільтрація подій
Use case ID	UC-09
Goals	Отримати перелік подій за певними критеріями
Actors	Авторизований користувач, гість
Trigger	Користувач обирає категорії/тип/дату/місто
Pre-conditions	-
Flow of Events	Користувач знаходиться на головній сторінці або на сторінці з мапою та обирає певні фільтри. Перелік подій оновлюється, щоб відповідав обраним параметрам
Extension	-
Post-Condition	Відображено відфільтровані події

Таблиця 2.10 – Варіант використання UC-10

Use case name	Редагування списку обраних подій
Use case ID	UC-10
Goals	Додати чи видалити подію зі списку обраних
Actors	Авторизований користувач
Trigger	Користувач натискає на відповідну кнопку
Pre-conditions	Авторизація
Flow of Events	Користувач знаходиться на сторінці конкретної події. Натискає на відповідну кнопку у вигляді серця.
Extension	-
Post-Condition	Подія додається/видаляється зі списку обраних в залежності від попереднього значення

Таблиця 2.11 – Варіант використання UC-11

Use case name	Поділитися подією
Use case ID	UC-11
Goals	Надіслати посилання на подію іншим користувачам
Actors	Авторизований користувач
Trigger	Користувач натискає на кнопку "Поділитися"
Pre-conditions	Авторизація
Flow of Events	Користувач знаходиться на сторінці конкретної події. Натискає на відповідну кнопку та обирає канал: месенджер, соцмережа, email. Надсилає посилання
Extension	-
Post-Condition	Посилання на подію надіслано

Таблиця 2.12 – Варіант використання UC-12

Use case name	Додавання події в календар
Use case ID	UC-12
Goals	Зберегти інформацію про подію у власному календарі
Actors	Авторизований користувач
Trigger	Користувач обирає опцію "Додати в календар"
Pre-conditions	Авторизація. Наявний доступ до календаря пристрою
Flow of Events	Користувач знаходиться на сторінці конкретної події. Натискає на відповідну кнопку. Подія додається з часом, назвою та описом
Extension	При не наданому доступі – виводиться запит
Post-Condition	Подія збережена в календарі пристрою

Таблиця 2.13 – Варіант використання UC-13

Use case name	Перегляд деталей події
Use case ID	UC-13
Goals	Отримання детальної інформації про подію
Actors	Авторизований користувач, гість
Trigger	Користувач зайшов на сторінку події
Pre-conditions	-
Flow of Events	Користувач клікнув на подію. Система відображає сторінку конкретної події, користувач переглядає інформацію
Extension	Користувач може перейти на сторінку-джерело по посиланню, або подивитись розташування на Google/Apple картах
Post-Condition	Переглянуто інформацію про подію

2.2 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. В таблиці 2.14 наведено загальну модель вимог, а в таблиці 2.15 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.1.

Таблиця 2.14 – Загальна модель вимог

№	Назва	ID Вимоги	Пріоритети	Ризики
1	Головна сторінка з подіями	FR-1	Високий	Високий
2	Система авторизації користувача.	FR-2	Високий	Високий
2.1	Реєстрація користувача.	FR-3	Високий	Високий
2.2	Авторизація користувача.	FR-4	Високий	Високий
2.3	Вихід із акаунту.	FR-5	Середній	Низький
3	Налаштування профілю інтересів	FR-6	Високий	Середній
4	Перегляд подій на мапі	FR-7	Середній	Високий
5	Отримання персоналізованих рекомендацій	FR-8	Високий	Високий

Продовження таблиці 2.14

6	Перегляд деталей події	FR-9	Високий	Середній
7	Керування подією	FR-10	Високий	Середній
7.1	Додавання події в обране	FR-11	Високий	Низький
7.2	Видалення події з обраного	FR-12	Середній	Низький
7.3	Поділитися подією	FR-13	Низький	Низький
7.4	Додати подію в календар	FR-14	Низький	Середній
8	Фільтрація подій	FR-15	Середній	Середній
9	Push-сповіщення.	FR-16	Середній	Середній

Таблиця 2.15 – Перелік функціональних вимог

Назва	Опис
FR-1	Головна сторінка з подіями. Система повинна відображати список актуальних подій зі зручним інтерфейсом для авторизованого користувача
FR-2	Система авторизації користувача. Система повинна забезпечувати можливість автентифікації користувача за допомогою email і пароля або через обліковий запис Google, а також дозволяти неавторизованим користувачам (гостям) переглядати обмежений контент.
FR-3	Реєстрація користувача. Система повинна надавати можливість новим користувачам здійснити реєстрацію за допомогою реєстраційної форми або через обліковий запис Google.
FR-4	Авторизація користувача. Система повинна дозволяти користувачам входити за допомогою введення електронної пошти та пароля або через обліковий запис Google.
FR-5	Вихід із акаунту. Користувач повинен мати змогу вийти зі свого профілю
FR-6	Налаштування профілю інтересів. Під час першого входу до системи користувач повинен пройти опитування, у якому вказує свої інтереси та бажану локацію для відстеження. Отримані дані зберігаються в профілі користувача для подальшого використання системи.
FR-7	Перегляд подій на мапі. Користувач повинен мати можливість переходити на сторінку з інтерактивною картою, на якій відображаються події відповідно до заданої геолокації, або поточного місця розташування.

Продовження таблиці 2.15

FR-8	Отримання персоналізованих рекомендацій. Система повинна сформувати актуальний список подій на основі інтересів, історії уподобань та локації користувача, що будуть відображатись на головній сторінці.
FR-9	Перегляд деталей події При виборі події користувач може переглядати опис, місце, дату, та іншу інформацію про неї.
FR-10	Керування подіями При виборі події авторизованим користувачем, система надає йому можливість виконувати наступні дії: додавати/видаляти з обраного, додавати у власний календар та ділитись з іншими користувачами.
FR-11	Додавання події в обране. Для авторизованого користувача система повинна відображати кнопку додавання події до списку вподобаних. При натисканні на кнопку подія додається до списку, якщо ще не була там присутня. Усі обрані події відображаються на сторінці “Вподобане”.
FR-12	Видалення події з обраного. Для авторизованого користувача система повинна відображати кнопку для видалення події з обраного. При натисканні на кнопку подія видаляється зі списку, якщо вже була в ньому. Ця подія більше не відображається на сторінці “Вподобане”.
FR-13	Поділитися подією. Для авторизованого користувача система повинна відображати кнопку для поширення події. Користувач має можливість поділитися посиланням на подію через різні канали.
FR-14	Додати подію в календар. Для авторизованого користувача система повинна відображати кнопку для додавання події до календаря. Після натискання система повинна інтегрувати подію до календаря користувача на його пристрої (наприклад, Google Calendar).
FR-15	Фільтрація подій. Система повинна надавати користувачам можливість фільтрувати події за різними параметрами, зокрема: категорією, діапазоном дат, містом проведення, а також типом події (онлайн чи офлайн).
FR-16	Push-сповіщення. Користувач повинен мати можливість обирати, чи отримувати повідомлення та нагадування про рекомендовані події через застосунок.

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13
FR-1					+								
FR-2	+	+											
FR-3	+												
FR-4		+											
FR-5		+											
FR-6			+										
FR-7				+									
FR-8						+							
FR-9													+
FR-10							+			+	+	+	
FR-11										+			
FR-12										+			
FR-13											+		
FR-14												+	
FR-15				+	+				+				
FR-16								+					

Рисунок 2.1 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

У межах даного застосунку нефункціональні вимоги класифікуються наступним чином:

1. Продуктивність

- час відгуку інтерфейсу не повинен перевищувати 200 мс на основні дії користувача (такі як, перехід між екранами, натискання кнопок).
- час завантаження застосунку – не більше 3 секунд на рекомендованих пристроях.

2. Масштабованість

- архітектура застосунку повинна забезпечувати можливість розширення функціоналу без потреби повного переписування базового коду.

3. Безпека

- користувачі мають різні рівні доступу до функціоналу застосунку залежно від ролі (авторизований користувач, гість).
- аутентифікація користувачів реалізована через Firebase Authentication із підтримкою: авторизації за електронною поштою та паролем; входу через Google-аккаунт (OAuth 2.0).

- дані у Firestore: шифруються як при передачі, так і під час зберігання (AES-256, автоматично); захищені правилами доступу (Firestore Security Rules), які дозволяють читання/запис лише авторизованим користувачам із відповідним uid.

4. Зручність використання

- інтерфейс має бути інтуїтивно зрозумілим, не вимагати навчання користувача.
- забезпечено інтуїтивну навігацію між основними розділами: максимальна глибина в ієрархії екранів – 3 рівні, що дозволяє швидко повертатися до основного меню;

5. Портативність

- підтримка всіх моделей iPhone, що працюють на iOS 17 та новіше.
- забезпечення коректної роботи інтерфейсу на різних розмірах екранів.

6. Вимоги до часу зберігання даних

- для забезпечення актуальності інформації про культурні події, дані автоматично очищаються після завершення або втрати актуальності.
- стандартний термін зберігання історії подій – не більше 1 дня після дати закінчення події.
- перевірка та видалення старих даних відбувається раз на 24 години автоматизовано на зовнішньому бекенд-сервері.
- персональні дані, пов'язані з подіями, також очищаються відповідно до політики конфіденційності після завершення терміну зберігання.

7. Конфігурованість:

- застосунок підтримує динамічне оновлення даних без потреби ручного оновлення з боку користувача (використовується Firestore + зовнішній сервер).

- зміни у структурі подій або доступних категоріях можуть вноситися адміністративно без оновлення клієнтської частини застосунку.
- при потребі нових функцій чи виправлення помилок, оновлення публікується через App Store, користувачеві надходить сповіщення згідно з налаштуваннями ОС.

2.4 Аналіз системних вимог

Програмне забезпечення повинно функціонувати на мобільних пристроях, що підтримують операційну систему IOS 17 та вище.

Визначення мінімальних системних вимог проводилось за допомогою моніторингу ресурсів у середовищі Xcode під час тестування роботи застосунку. Це дозволило точно встановити, які апаратні ресурси необхідні для забезпечення працездатності базового функціоналу.

Мінімальні системні вимоги:

- тип процесору: Apple A15 Bionic.
- об'єм ОЗУ: 4 Гб;
- підключення до мережі Інтернет зі швидкістю від 20 мегабіт;
- вільний об'єм пам'яті: 200 мб;

Рекомендовані параметри забезпечують комфортне використання застосунку з максимально плавним інтерфейсом, швидким завантаженням подій та стабільною роботою всіх інтегрованих сервісів (карта, push-сповіщення, додавання подій до календаря тощо).

Рекомендовані системні вимоги:

- тип процесору: Apple A15 Bionic і вище.
- об'єм ОЗУ: 4 Гб;
- підключення до мережі Інтернет зі швидкістю від 30 мегабіт;
- вільний об'єм пам'яті: 128 гб;

2.5 Аналіз економічних показників програмного забезпечення

Для початку буде проведено аналіз функціональних точок з метою оцінки орієнтовного обсягу даної розробки. На основі отриманих результатів буде визначено ключові економічні показники – вартість, зусилля та тривалість реалізації пз.

2.5.1 Аналіз функціональних точок

Внутрішні логічні файли (ILF) – дані, які зберігаються в системі та повністю контролюються нею. У системі передбачено:

- користувачі – 10 FP
- події – 10 FP
- список обраних подій – 7 FP

Разом ILF: $10 \cdot 2 + 7 = 27$ FP

Зовнішні інтерфейсні файли (EIF) – джерела даних, які використовуються системою, але не контролюються нею:

- служби автентифікації (Firebase Auth) – 5 FP
- Real-Time Events API (для оновлення інформації про події) – 5 FP

Разом EIF: $2 \cdot 5 = 10$ FP

Зовнішні введення (EI) – функції введення даних користувачем у систему:

- реєстрація користувача – 3 FP
- вхід у систему – 3 FP
- налаштування профілю інтересів – 3 FP
- додавання/видалення події з обраного – 3 FP
- керування push-сповіщеннями – 3 FP

Разом EI: $5 \cdot 3 = 15$ FP

Зовнішні виводи (EO) – функції, що ініціюють вихід інформації з системи

- надсилення push-сповіщень про події – 4 FP
- поділитися подією (генерація посилання) – 4 FP

Разом EO: $2 \cdot 4 = 8$ FP

Запити (EQ) – функції запитів і отримання інформації:

- пошук подій за критеріями (категорія, місто, дата, тип) – 3 FP
- отримання персоналізованих рекомендацій – 4 FP
- перегляд деталей події – 3 FP

Разом EQ: $2 \cdot 3 + 4 = 10$ FP

Загалом: $27 + 10 + 15 + 8 + 10 = 70$ FP

2.5.2 Коефіцієнт коригування вартості

Відповідно до нефункціональних вимог системи, оцінено 14 глобальних характеристик (GSC). Результати оцінювання представлені у таблиці 2.16.

Таблиця 2.16 – Оцінки основних характеристик системи

	Характеристика	Оцінка	Коментар
1	Обмін даними з іншими системами	3	Інтеграція з Firebase, API подій та зовнішнім сервером
2	Розподілена обробка	2	Дані розподіляються між клієнтом, Firebase та зовнішнім сервером
3	Продуктивність	2	Швидке реагування на дії користувача
4	Обмеження апаратних ресурсів	1	Описані мінімальні вимоги

Продовження таблиці 2.16

5	Транзакційне навантаження	2	Часті запити, push-нотифікації
6	Інтенсивність взаємодії з користувачем	3	Мобільна адаптація, інтерактивність
7	Ергономіка (UI/UX)	3	Фокус на зручності для користувача
8	Інтенсивність зміни даних	2	Події оновлюються регулярно, але не постійно
9	Складність обробки	3	Алгоритм рекомендацій
10	Повторне використання	1	Компоненти можна перевикористовувати
11	Зручність інсталяції	1	Просте встановлення та запуск на мобільних пристроях
12	Зручність адміністрування	0	Не вимагається
13	Портативність	1	Мобільний застосунок для iOS 17+
14	Гнучкість	0	Не вимагається

Сумуємо оцінки для знаходження значення TDI = 24

Коефіцієнт VAF = $0,65 + (TDI / 100) = 0,89$

Нормована кількість функціональних точок (AFPC) = $FP \cdot VAF = 70 \cdot 0,89 = 62,3$

2.5.3 Розрахунок обсягу коду (SLOC)

$SLOC = 62,3 \cdot 53$ (середнє значення множника для мови Swift) = 3 301,9 рядків коду

2.5.4 Оцінка зусиль, графіку та вартості

Розрахунок здійснюється за відповідними формулами моделі COCOMO [17], при цьому коефіцієнти визначаються залежно від розміру проєкту – у даному випадку для органічного типу.

Зусилля = $aa \cdot (LOC/1000)^{bb} = 2,4 \cdot (3,3019)^{1,05} = 8,4$ людино-місяців

$$\text{Графік} = cb \cdot \text{Зусилля}^{db} = 2,5 \cdot 8,4^{0,38} = 5,61 \text{ місяців}$$

$$\text{Вартість} = \text{Зусилля} \cdot \text{Зарплата} = 8,4 \cdot 35000 \text{ грн (середня зарплата iOS розробника в Україні)} \approx 294\,000 \text{ грн}$$

2.6 Постановка завдання на розробку програмного забезпечення

Метою розробки є створення мобільного застосунку Festigo, який забезпечує персоналізовані рекомендації культурних заходів на основі інтересів, геолокації та поведінки користувача. Розробка спрямована на вирішення проблеми складності пошуку релевантних подій серед великого обсягу інформації. Основна ідея проєкту полягає у поєднанні можливостей агрегації подій з відкритих джерел та реалізації механізмів персоналізації рекомендацій, що ґрунтуються на аналізі індивідуальних уподобань користувача, його попередніх дій у додатку, та характеристиках подій (місце, час проведення, тематика тощо).

Для досягнення поставленої мети передбачено вирішення таких задач:

- реалізація функціоналу реєстрації та авторизації користувача, зокрема за допомогою Google-аккаунта.
- створення системи опитувань для налаштування профілю та визначення інтересів користувача.
- розробка головної сторінки застосунку з можливістю перегляду всіх доступних подій і персоналізованих рекомендацій.
- побудова рекомендаційної системи за допомогою алгоритму фільтрації на основі контенту.
- реалізація системи фільтрації подій за категоріями, датою, місцем проведення та типом заходу.
- створення інтерактивної карти з відображенням подій поблизу та можливістю перегляду їхніх деталей.

- реалізація детального перегляду подій з можливістю збереження в обране, поділитися з іншими користувачами, додавання до календаря та переходу на сторонні сервіси для купівлі квитків.

Висновки до розділу

У даному розділі було сформульовано вимоги до програмного забезпечення мобільного застосунку. В результаті аналізу потреб користувачів створено діаграму варіантів використання, яка відображає основних користувачів системи (авторизований користувач та гість) та їхню взаємодію з функціоналом додатка. Для кожного сценарію використання визначено передумови, послідовність дій, альтернативні шляхи розвитку подій та очікувані результати, що дає змогу краще зрозуміти логіку роботи застосунку з погляду кінцевого користувача.

Також було визначено перелік функціональних вимог, які охоплюють ключові підсистеми: реєстрацію та авторизацію користувачів, налаштування облікового запису, надання персоналізованих рекомендацій, перегляд подій у різних форматах (на мапі та в списку), взаємодію з подіями та систему сповіщень. Кожну вимогу оцінено за пріоритетом та потенційними ризиками, що дозволяє впорядкувати подальший процес реалізації та перевірки системи.

Окрім функціональних, у розділі також описані системні вимоги та нефункціональні вимоги, які стосуються таких аспектів, як швидкодія, безпека, доступність для різних категорій користувачів і зручність інтерфейсу.

Було проведено аналіз і розрахунок економічних показників, таких як зусилля, вартість та час розробки, а також визначено основну задачу проєкту.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Розроблюване програмне забезпечення реалізоване на основі клієнт-серверної архітектури, що передбачає чітке розділення на клієнтську та серверну частини. Клієнтом тут виступає сам мобільний застосунок, розроблений для iOS з використанням мови Swift та фреймворку SwiftUI, із застосуванням архітектурного патерну MVVM (Model–View–ViewModel). Загальний приклад патерну наведено на рисунку 3.1.

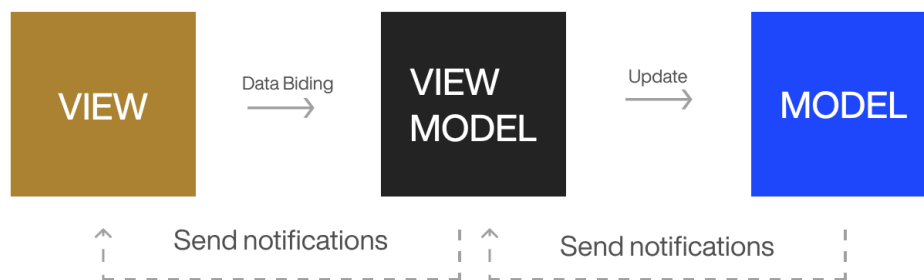


Рисунок 3.1 – Шаблон патерну MVVM [12].

Серверна частина реалізована у вигляді окремого бекенду, написаного на Python з використанням фреймворку FastAPI. Архітектура сервера побудована за багатомодульним принципом, тобто функціонал розділений по окремим файлам-модулям забезпечуючи читабельність коду і дозволяючи розширювати програму без впливу на основну логіку. Загалом він виконує декілька ключових функцій, таких як періодичний збір даних про події із зовнішнього API (через планувальник – scheduler), їх попередню обробку та категоризацію, збереження та оновлення інформації про події у базі даних Firestore, а також формування персоналізованих рекомендацій для користувачів за допомогою спеціального модуля рекомендаційної системи. Структура серверу наведена на рисунку 3.2 та описана у таблиці 3.1.

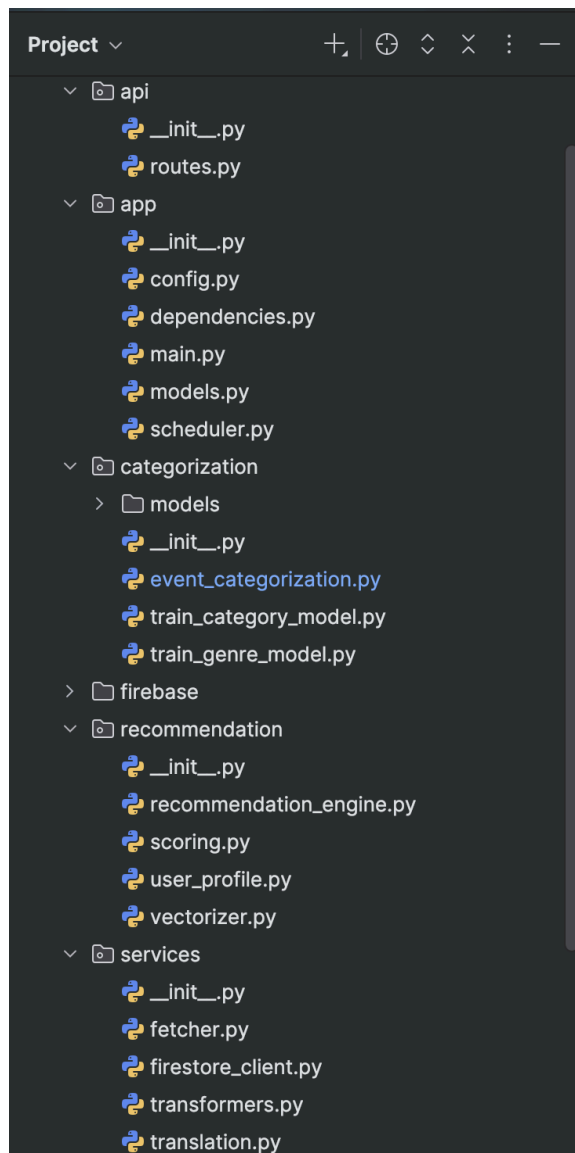


Рисунок 3.2 – Структура бекенд серверу

Таблиця 3.1 – Опис основних модулів та файлів серверу

app/	Містить основні компоненти бекенду: точку входу FastAPI, конфігурацію, моделі, планувальник та загальні залежності.
app/scheduler.py	Планувальник задач, що періодично викликає фонові функції: оновлення та обробка подій
api/routes.py	Визначає HTTP маршрути для обробки запитів від мобільного клієнта (отримання персоналізованих подій), а також маршрути для тестування запитів від Postman.

Продовження таблиці 3.1

categorization/	Містить логіку автоматичної категоризації подій використовуючи натреновані моделі.
services/	Включає логіку отримання, обробки та перетворення подій. Розбито на файли за відповідальністю (завантаження з API, переклад тексту, робота з Firestore).
recommendation/	Модуль рекомендаційної системи. Включає логіку формування списку рекомендованих подій на основі векторної подібності, скорингу тощо.
firebase/	Містить облікові дані сервісного аккаунта Firebase (додано до .gitignore для безпеки).
.env	Конфігураційні змінні середовища (ключі, URL, таймінги).

Обмін даними між клієнтом і сервером відбувається через REST API з використанням формату JSON та захищеного протоколу HTTPS, що забезпечує безпечну передачу інформації. Особливістю архітектури є те, що у більшості випадків мобільний застосунок функціонує автономно, без потреби постійного зв'язку з бекендом: він напряду звертається до бд для читання подій і виконання автентифікації через Firebase Auth. Запити до сервера здійснюються лише в тих випадках, коли необхідно отримати рекомендовані події, згенеровані на основі користувацьких даних.

Перед тим як надіслати запит до серверної частини для отримання рекомендацій, мобільний застосунок спочатку ініціює запит на видачу спеціального токена доступу. Сервер формує цей токен на основі автентифікаційної інформації користувача та власного ключа і повертає його клієнту. Надалі мобільний застосунок включає цей токен у заголовок кожного запиту до серверу, що дозволяє серверній частині автентифікувати ці запити.

Для візуального представлення архітектури доцільно використовувати діаграми перших трьох рівнів моделі C4 [22]:

Контекстна діаграма (рис. 3.3) – відображає ключових учасників взаємодії з системою Festigo. До них належать:

- користувач або кінцевий споживач застосунку
- Festigo System – цілісна програмна система, що поєднує мобільний застосунок та серверну частину
- Firebase Platform – зовнішній сервіс для автентифікації користувачів і зберігання даних
- Real-Time Event API – сторонній API, який надає актуальні дані про події.

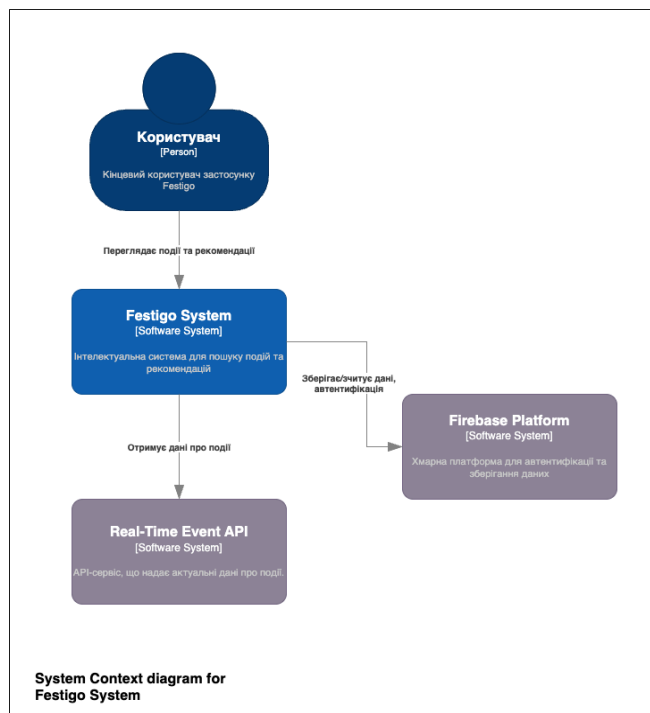


Рисунок 3.3 – Контекстна діаграма

Діаграма контейнерів (рис. 3.4) – деталізує внутрішню структуру системи Festigo. Вона включає:

- мобільний застосунок, розроблений на Swift з використанням архітектури MVVM, який забезпечує перегляд подій, автентифікацію та взаємодію з користувачем;

- серверну частину, реалізовану на Python з використанням FastAPI, яка відповідає за збір подій із зовнішнього API, їх обробку, категоризацію та формування рекомендацій;
- взаємодію з Firebase Authentication (для управління сесіями користувачів) та Firebase Firestore (для зберігання інформації про події, локації, профілі);
- обмін даними через захищені канали (HTTPS/REST API) та використання планувальника для періодичного оновлення подій.

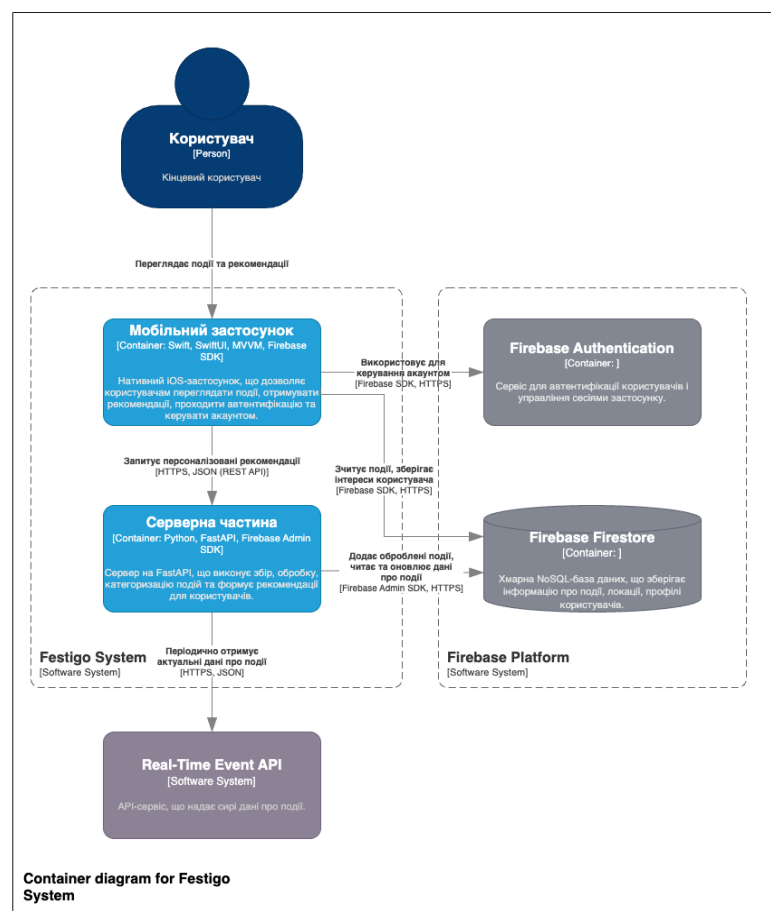


Рисунок 3.4 – Діаграма контейнерів

Діаграма компонентів, наведена у графічному матеріалі, креслення 2, деталізує внутрішню структуру серверної частини системи Festigo та демонструє розподіл функціональності на окремі логічні компоненти, які взаємодіють між собою для забезпечення збору, обробки, зберігання та аналізу подій. Включає:

- обробник API запитів – компонент на базі FastAPI, який обробляє запити від мобільного застосунку та повертає сформовані рекомендації користувачам;
- планувальник – виконує періодичний запуск фонових задач: отримання подій із зовнішнього API, обробка та збереження. Працює автоматично за заданим розкладом;
- модуль збору подій – отримує актуальні події зі стороннього Real-Time Event API, обробляє їх та зберігає у Firebase Firestore;
- модуль перекладу подій – забезпечує машинний переклад текстових описів подій за допомогою бібліотеки googletrans для подальшої локалізації застосунку;
- модуль категоризації подій – класифікує події за жанрами та темами. Використовує алгоритми машинного навчання (TfidfVectorizer, OneVsRestClassifier, LinearSVC) для автоматичної багатоміткової категоризації;
- модуль формування рекомендацій – векторизує дані про події та профіль користувача, обчислює схожість за допомогою cosine similarity та формує персоналізовані рекомендації.

3.2 Архітектурні рішення та обґрунтування вибору засобів розробки

У проєкті реалізовано управління користувачами за допомогою Firebase Authentication. Цей сервіс забезпечує централізоване зберігання облікових записів користувачів у хмарі та автоматизує процес аутентифікації. Застосунок підтримує вхід до системи за допомогою електронної пошти та пароля, а також інтегрований зі стороннім провайдером автентифікації Google.

Основна функціональність мобільного застосунку полягає в отриманні актуальної інформації про культурні події різних категорій. Для цього використовується API Real-Time Event Search, розміщене на платформі

RapidAPI.com. Даний сервіс надає єдиний інтерфейс для керування сторонніми API та спрощує процес інтеграції із зовнішніми джерелами даних. Запити до API виконуються за стандартним протоколом HTTPS із використанням ключа доступу. У відкритому доступі можна знайти й інші альтернативи, такі як Data Thistle API Explorer, Ticketmaster Discovery API, PredictHQ Events API тощо. Проте їх використання в межах проєкту є обмеженим через наступні причини: недостатню кількість різноманітних категорій подій, відсутність безкоштовних тарифних планів, а також відсутність інформації про події на території України.

Для зберігання даних у проєкті використовується хмарна NoSQL бд – Firestore, що забезпечує зберігання структурованих даних у форматі колекцій та документів. Кожна колекція містить документи, які мають унікальний ідентифікатор та складаються з полів різного типу. Такий підхід забезпечує масштабованість і гнучкість при моделюванні даних, та просту реалізацію фільтрації й пошуку [18].

Розробка дипломного проєкту здійснюється у вигляді мобільного застосунку для платформи iOS, який створюється з використанням мови програмування Swift та фреймворку SwiftUI – сучасного інструменту Apple для побудови інтерфейсів. Цей вибір обумовлений високою продуктивністю, підтримкою останніх версій операційної системи, інтеграцією з системними сервісами та активною підтримкою з боку спільноти [11]. Для розробки використовується Xcode – офіційне середовище від Apple, яке надає все необхідне для розробки, тестування та налагодження застосунку.

Серверна частина реалізована на мові Python із використанням FastAPI – це легкий і швидкий фреймворк, який підтримує асинхронну обробку запитів і дозволяє просто описувати REST API. Альтернативами могли б бути Flask або Django, однак FastAPI краще підходить для цього проєкту завдяки продуктивності та простоті в роботі з JSON-запитами [27].

Python було обрано як мову для бекенду завдяки її зрозумілому синтаксису, великій кількості готових бібліотек, а також через зручність реалізації рекомендаційного алгоритму, який є важливою частиною функціоналу застосунку. Розробка серверної логіки здійснюється у PyCharm, спеціально створеному для роботи з Python-проєктами IDE.

3.3 Конструювання програмного забезпечення

3.3.1 Розробка рекомендаційного алгоритму

Для реалізації алгоритму персоналізованих рекомендацій подій використано модифіковану контентну фільтрацію, яка поєднує класичний підхід із векторизацією текстових полів та додатковими динамічними фільтрами. Порядок роботи класичного підходу зображено на рисунку 3.5. Основна ідея полягає у представленні як подій, так і користувачів у вигляді векторів, після чого виконується обчислення косинусної подібності для визначення рівня релевантності.

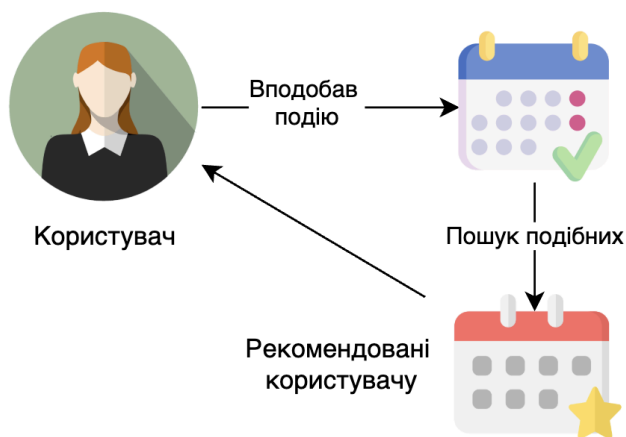


Рисунок 3.5 – Загальний приклад роботи класичної контентної фільтрації

Важливим підготовчим етапом є категоризація подій — процес автоматичного визначення основних тематик кожної події [25]. Для цього використовуються машинні моделі (SVM з TF-IDF) і словникові правила, які аналізують текст назви, опису, місця проведення тощо. Результатом є список релевантних категорій із ваговими коефіцієнтами, які зберігаються в полі

main_categories. За аналогічним принципом визначаються жанри для музичних подій(концерти), фільмів та театральних вистав.

Щоб уникнути проблеми "холодного старту" — коли про нового користувача ще немає історії взаємодій — система формує його початковий профіль на основі відповідей у опитуванні при реєстрації [26]. Там користувач обирає вподобані категорії подій, жанри музики, кіно тощо. Ці дані і утворюють початковий профіль.

У подальшому, коли користувач починає взаємодіяти з подіями (додавати до списку обраних), система може використовувати не лише вектор його профілю з опитування, а й вектори вже вподобаних подій. Це дозволяє адаптувати рекомендації до реальної поведінки користувача і покращити точність персоналізації.

Кожна подія описується набором атрибутів (назва, опис, категорії, жанри, місце тощо), які перетворюються у текст і векторизуються з використанням методу TF-IDF – показник, який оцінює важливість слова в документі (конкретній події) відносно колекції документів.

Тобто для кожного поля F_i з переліку атрибутів формується текстове представлення. Далі використовується техніка `TfidfVectorizer` (з виключенням стоп-слів) з бібліотеки `sklearn` для побудови вектору [23]. У результаті кожна подія описується набором векторів, що зберігаються у бд в полі `component_vectors`.

За схожим принципом будується профіль користувача – його векторні представлення також створюються для кожного поля і зберігаються в `component_profile_vectors`.

Для знаходження схожих подій використовується косинусна подібність між вподобаними подіями користувача(або його профілем) і усіма подіями. Це метрика, яка вимірює кутову відстань між двома векторами, незалежно від їх довжини. Чим менший кут між векторами – тим вищою є подібність. У межах від 0 до 1, де 1 означає повну ідентичність (вектори напрямлені в один

бiк), 0 – повну вiдсутнiсть схожостi (кут 90°) [24]. Приклад близьких векторiв зображено на рисунку 3.6. Формула обчислення подiбностi:

$$\text{cosine_similarity}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \quad (3.1)$$

Обчислюємо cosine similarity для кожного атрибуту, пiсля чого беремо середнє значення всiх отриманих коефiцiєнтiв, яке i буде "базовою" оцiнкою подiї.

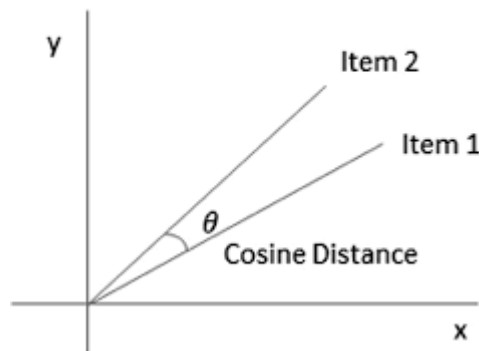


Рисунок 3.6 – Косинусна подiбнiсть [24]

Щоб зробити результат бiльш персоналiзованим, базове значення схожостi додатково коригується на основi зовнiшнiх факторiв:

- географiчна вiдстань мiж координатами подiї та мiсцезнаходженням користувача, яка рахується за допомогою бiблiотеки geору (метод geodesic). Якщо подiя знаходиться далi, нiж вказаний користувачем максимум – оцiнка подiї множиться на 0.5, тобто подiя стає менш релевантною
- час проведення подiї. Зiставляється час подiї iз зручними для користувача часовими промiжками (ранок, день, вечiр, вихiднi). Якщо не збiгається – оцiнка подiї множиться на 0.7

Таким чином, формула фiнальної оцiнки виглядає так:

$$\text{final_score} = \text{base_similarity} \cdot \text{time_mod} \cdot \text{distance_mod} \quad (3.2)$$

Останнім етапом є сортування подій за спаданням значення `final_score`. Топ-N будуть відображатись у мобільному застосунку, як персональні рекомендації.

3.3.2 Опис структури бази даних

Як систему керування базами даних для розробки було обрано Firebase Firestore. База даних використовується для зберігання культурних подій, користувачів, а також даних про їх уподобання. Опис таблиць бази даних наведено у таблицях 3.2-3.6. Модель бази даних наведена наведено у графічному матеріалі, креслення 3.

Таблиця 3.2 – Опис таблиці user

Назва поля	Тип даних	Опис
id	String	Унікальний ідентифікатор користувача
email	String	Електронна адреса користувача
name	String	Ім'я користувача
joined	Timestamp	Час створення акаунта
did_complete_onboarding	Boolean	Ознака проходження первинного налаштування

Таблиця 3.3 – Опис таблиці event

Назва поля	Тип даних	Опис
id	String	Унікальний ідентифікатор події
name	String	Назва події
description	String?	Опис події
start_time	Timestamp	Дата й час початку події

Продовження таблиці 3.3

end_time	Timestamp	Дата й час завершення події
city	String	Місто проведення події
country	String	Країна проведення події
image_url	String	Посилання на зображення
is_virtual	Boolean	Чи є подія віртуальною
link	String?	Посилання на зовнішнє джерело
main_categories	[String]	Основні категорії події
genres	[String]	Жанри події
venue_id	String	Зовнішній ключ, посилання на id місця події з таблиці venue

Таблиця 3.4 – Опис таблиці venue

Назва поля	Тип даних	Опис
id	String	Унікальний ідентифікатор місця проведення
name	String	Назва локації
address	String	Повна адреса місця проведення
latitude	Number	Географічна широта
longitude	Number	Географічна довгота
subtypes	[String]	Підтип об'єкта

Таблиця 3.5 – Опис таблиці onboarding_responses

Назва поля	Тип даних	Опис
id	String	Унікальний ідентифікатор запису
user_id	String	Ідентифікатор користувача, до якого належать відповіді
answers	[[String]]	Електронна адреса користувача
timestamp	Timestamp	Час проходження

Таблиця 3.6 – Опис таблиці favourite_event

Назва поля	Тип даних	Опис
event_id	String	Зовнішній ключ, посилання на id події
user_id	String	Зовнішній ключ, посилання на id користувача
date_created	Timestamp	Дата додавання події до обраного

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.7.

Таблиця 3.7 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Xcode	Головне середовище розробки програмного забезпечення клієнтської частини дипломного проєкту.
2	PyCharm	Середовище розробки програмного забезпечення серверної частини.
3	Postman	Інструмент для тестування REST-запитів, який використовувався для перевірки викликів до зовнішнього сервера через API.

Продовження таблиці 3.7

	Cloud Firestore	Хмарна база даних, що використовується для зберігання та синхронізації даних подій у реальному часі між клієнтом і сервером.
	Git	Система керування версіями.
	Real-Time Event Search	API, що використовується для отримання актуальної інформації про культурні події в реальному часі. Дозволяє інтегрувати події з зовнішніх джерел у застосунок.
	Scikit-learn	Бібліотека мови Python, що використовується для машинного навчання у алгоритмах категоризації та рекомендацій.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

З урахуванням обробки персональних даних користувачів, в архітектурі застосунку приділено значну увагу питанням безпеки. Було реалізовано низку заходів, спрямованих на зниження ризиків та захист інформації на всіх рівнях системи.

У процесі розробки клієнтської частини використовуються вбудовані засоби безпеки Xcode, що попереджають про використання застарілих або потенційно небезпечних API.

Усі форми введення даних – зокрема при реєстрації, логіні, надсиланні геолокації – проходять попередню валідацію. Це дозволяє запобігти атакам типу SQL/NoSQL injection, XSS або надсиланню некоректних даних.

Користувач самостійно обирає, чи надавати дозвіл на доступ до своєї геолокації та інформації про свої інтереси, відповідно до політик конфіденційності. Такі дані користувача зберігаються у базі Firestore для подальшої персоналізації контенту. Доступ до них суворо обмежений за допомогою Firebase Security Rules (лише автентифікований користувач має доступ).

Клієнт і бекенд-сервер обмінюються даними через захищений протокол HTTPS у форматі JSON. Для взаємодії використовується унікальний токен доступу, який безпечно зберігається в середовищі застосунку і запобігає несанкціонованому доступу до серверних ресурсів. Аналогічно, файл конфігурації Firebase захищено та обмежено у доступі.

Сесії користувача захищені через Firebase Authentication. Користувач може виконати вихід, або ж самостійно видалити свій обліковий запис. У разі видалення, усі персональні дані, знищуються з бд, відповідно до політик обробки персональної інформації.

Висновки до розділу

У цьому розділі було детально розглянуто архітектуру та технічну реалізацію програмного забезпечення, яке ґрунтується на клієнт-серверній моделі з використанням сучасних технологій розробки. Клієнтська частина реалізована як iOS-застосунок на основі SwiftUI та паттерну MVVM, що забезпечує чітке розділення відповідальностей та зручність підтримки коду. Серверна частина написана на Python із використанням FastAPI, що дозволяє ефективно обробляти запити та легко масштабувати систему.

Особливу увагу приділено реалізації рекомендаційної системи, яка базується на модифікованому підході контентної фільтрації з урахуванням векторної подібності, геолокаційних і часових обмежень. Це забезпечує персоналізований підбір подій для кожного користувача.

Для зберігання даних використано хмарну базу даних Firestore. Також описано комплекс заходів безпеки для захисту персональних даних, зокрема використання протоколів HTTPS, автентифікації через Firebase Auth та зберігання чутливої інформації в захищеному середовищі.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

У даному підрозділі проведено аналіз якості розробленого програмного забезпечення на основі обраних метрик та зіставлення їх з нефункціональними вимогами, визначеними на етапі проєктування. Основною метою є обґрунтування відповідності застосунку вимогам до продуктивності, масштабованості, зручності використання, безпеки, портативності та конфігурованості.

Для оцінки якості ПЗ було використано такі метрики:

- цикломатична складність
- кількість рядків коду (LOC)
- наявність дублікацій
- індекси підтримуваності, надійності та безпеки
- Оцінка часу відгуку та швидкості запуску

Цикломатична складність визначає кількість незалежних шляхів у програмному коді, що дозволяє оцінити складність логіки, рівень розгалужень та потенційні труднощі з тестуванням і супроводом ПЗ. За результатами аналізу з використанням платформи SonarCloud, середнє значення цикломатичної складності на файл становить 3.88 для Swift та 3.77 для Python. Ці значення перебувають у межах рекомендованої норми (до 10) і свідчать про добре структурований код. Таким чином, дотримано нефункціональну вимогу масштабованості, що дозволяє розширювати функціонал без необхідності суттєвих змін у базовому коді.

Кількість логічних рядків коду дозволяє оцінити загальний обсяг реалізованого функціоналу. За результатами аналізу з використанням утиліти Сloc отримали 2840 рядків коду на мобільний застосунок та 1168 на сервер. Ці показники є типовими для застосунків такого класу і свідчать про адекватну складність рішення. Наявність чітко структурованого коду з

помірним обсягом підтверджує відповідність вимогам зручності супроводу та портативності, а також забезпечує достатню гнучкість для конфігурації.

Наявність повторюваних фрагментів коду також знижує підтримуваність та ускладнює внесення змін. За даними SonarCloud, рівень дублікації коду склав 1.0% та 3.9% дублікацій для клієнтської та серверної частин відповідно. Такі низькі значення вказують на добру модульність та повторне використання коду.

Оцінювання програмного забезпечення на платформі SonarCloud додатково дало такі результати:

- підтримуваність: A (100%)
- надійність: A
- безпека: A

Ці оцінки вказують на відсутність критичних помилок, вразливостей безпеки та збалансоване використання ресурсів. Таким чином, підтверджується виконання вимог щодо надійності, а також високий рівень безпеки, зокрема через використання Firebase Authentication, шифрування і Firestore Security Rules.

Також результати ручного тестування на симуляторі iPhone 16 (iOS 17) показали відповідність часу відгуку інтерфейсу та часу запуску застосунку відповідно до вимоги продуктивності.

Графічне представлення результатів наведено на рисунках 4.1, 4.2 та 4.3.

URL	https://github.com/kisellsn/festigo-server					
Branch	master					
Commit	3b8d7abe083e7424445ad87e9d7f721712eb75d0					
Language	Files	Lines	Blanks	Comments	Code	Complexity
Python	22	1400	168	64	1168	88
Markdown	1	1	0	0	1	0
PlainText	1	55	2	0	53	0
gitignore	1	69	17	16	36	0
Total	25	1525	187	80	1258	83

Рисунок 4.1 – Метрики платформи Clock для серверу

 URL	https://github.com/kisellsn/FestiGo					
 Branch	main					
 Commit	b4d0c54f563726d067388c8e0a54b0c8f2cab6c0					
Language	Files	Lines	Blanks	Comments	Code	Complexity
Swift	51	4464	649	975	2840	198
JSON	17	7621	1	0	7620	0
gitignore	1	83	18	25	40	0
Total	69	12168	668	1000	10500	198

Рисунок 4.2 – Метрики платформи Clock для клієнту

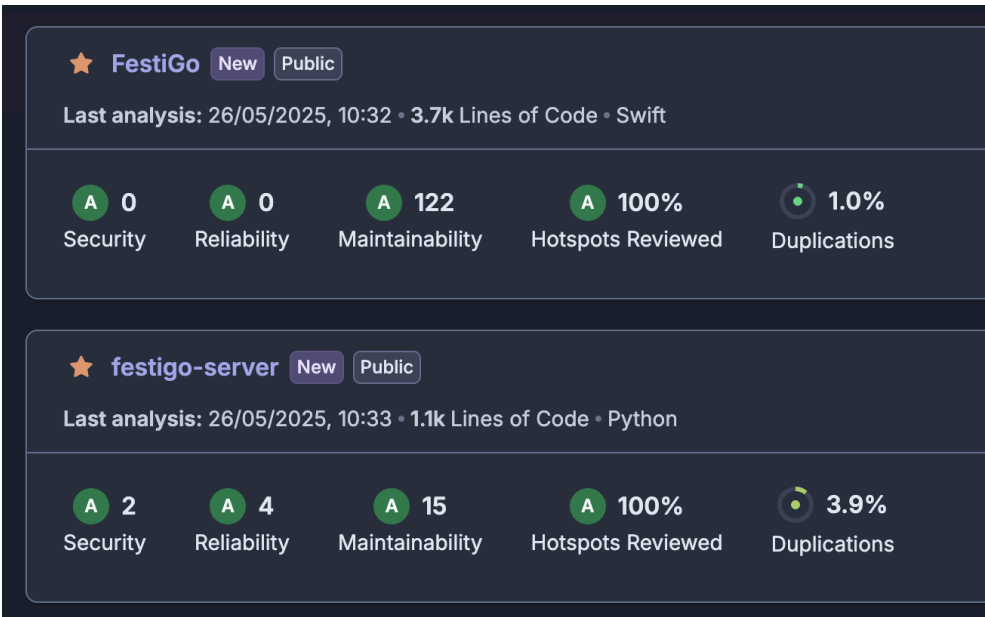


Рисунок 4.3 – Метрики платформи SonarCloud

4.2 Опис процесів тестування

Тестування виконується згідно документу «Програма та методика тестування».

Було виконане мануальне тестування програмного забезпечення, опис відповідних тестів наведено у таблицях 4.1 – 4.12.

Таблиця 4.1 – Тест 1.1 Реєстрація користувача

Початковий стан системи	Користувач знаходиться на сторінці реєстрації
Вхідні дані	Ім'я, електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: ім'я користувача, коректна електронна пошта, яка до цього не була зареєстрована в системі, пароль від 8 до 15 символів. Після цього натискається кнопка підтвердження реєстрації
Очікуваний результат	Реєстрація проходить успішно, користувач додається у систему і перенаправляється на сторінку з опитуванням
Фактичний результат	Збігається з очікуваним

Таблиця 4.2 – Тест 1.2 Авторизація користувача

Початковий стан системи	Користувач знаходиться на сторінці авторизації
Вхідні дані	Електронна пошта, пароль
Опис проведення тесту	У відповідні поля вводяться: коректна електронна пошта, яка до цього вже була зареєстрована в системі, та пароль від 8 до 15 символів, що співпадає з паролем у БД. Після цього натискається кнопка підтвердження авторизації
Очікуваний результат	Авторизація проходить успішно, користувач перенаправляється на головну сторінку
Фактичний результат	Збігається з очікуваним

Таблиця 4.3 – Тест 1.3 Вхід користувача через Google-акаунт

Початковий стан системи	Користувач знаходиться на початковій сторінці
Вхідні дані	Google-акаунт

Продовження таблиці 4.3

Опис проведення тесту	Натискається кнопка входу через Google та підтверджується доступ до акаунту, що вже був зареєстрованим цим способом
Очікуваний результат	Авторизація проходить успішно, користувач перенаправляється на головну сторінку
Фактичний результат	Збігається з очікуваним

Таблиця 4.4 – Тест 1.4 Налаштування профілю

Початковий стан системи	Користувач зареєструвався і знаходиться на сторінці з опитуванням
Вхідні дані	Обрані категорії інтересів, місцезнаходження
Опис проведення тесту	Активується поле згоди з умовами використання та політикою конфіденційності. Заповнюється профіль інтересів і натискається кнопка підтвердження відповідей.
Очікуваний результат	Інтереси збережені до профілю користувача
Фактичний результат	Збігається з очікуваним

Таблиця 4.5 – Тест 1.5 Перегляд подій на мапі

Початковий стан системи	Користувач авторизований у системі
Вхідні дані	Активовано геолокацію пристрою або встановлено локацію за замовчуванням у налаштуваннях профілю.
Опис проведення тесту	Перехід на вкладку "Мапа". Система завантажує карту, на якій здійснюється візуалізація подій, розташованих поблизу поточної або збереженої геолокації.

Продовження таблиці 4.5

Опис проведення тесту	Перевіряється коректність завантаження мапи, відповідність позначень подій їхнім реальним координатам, а також функціональність інтерфейсу (масштабування, переміщення, відображення деталей подій за натисканням).
Очікуваний результат	Карта завантажується коректно, відображаючи події відповідно до заданої локації. Усі позначки на мапі точно відповідають географічним координатам подій. Інтерфейс працює стабільно.
Фактичний результат	Збігається з очікуваним

Таблиця 4.6 – Тест 1.6 Фільтрація подій

Початковий стан системи	Користувач авторизований у системі, знаходиться на головній сторінці або на сторінці з картою
Вхідні дані	-
Опис проведення тесту	Застосовуються різні комбінації фільтрів для звуження вибірки подій. Зокрема, обирається кілька категорій, встановлюється місто за замовчуванням, визначається часовий проміжок проведення подій та обирається інший тип подій.
Очікуваний результат	Система успішно застосовує всі вибрані фільтри. Відображаються лише ті події, які відповідають принаймні одній з обраних категорій, розташовані в обраному місті, відбуваються у вказаний часовий проміжок та мають вказаний тип.

Продовження таблиці 4.6

Фактичний результат	Збігається з очікуваним
---------------------	-------------------------

Таблиця 4.7 – Тест 1.7 Персоналізовані рекомендації

Початковий стан системи	Користувач авторизований у системі, профіль інтересів заповнено
Вхідні дані	-
Опис проведення тесту	Переходимо на головну сторінку
Очікуваний результат	Рекомендовані події відображено
Фактичний результат	Збігається з очікуваним

Таблиця 4.8 – Тест 1.8 Перегляд деталей події

Початковий стан системи	Користувач авторизований у системі
Вхідні дані	-
Опис проведення тесту	Зі списку обирається подія, відкривається сторінка з її деталями
Очікуваний результат	Вся інформація про подію доступна. Надається можливість вподобати, поділитися, додати в календар та перейти на сторінку-джерело цієї події
Фактичний результат	Збігається з очікуваним

Таблиця 4.9 – Тест 1.9 Додавання та видалення події з обраного

Початковий стан системи	Користувач авторизований у системі
Вхідні дані	-

Продовження таблиці 4.9

Опис проведення тесту	Переходимо на сторінку з деталями події. Натискається відповідна кнопка у вигляді серця.
Очікуваний результат	Якщо кнопка, відображається у стані "невподобаної" (іконка порожнього серця), після натискання подія має бути додана до списку обраного. Якщо кнопка відображається у стані "вподобаної" (іконка заповненого серця), після натискання подія має бути видалена зі списку обраного. Зміна стану вподобання події має коректно синхронізуватись на всіх відповідних екранах інтерфейсу користувача.
Фактичний результат	Збігається з очікуваним

Таблиця 4.10 – Тест 1.10 Поділитися подією

Початковий стан системи	Користувач авторизований у системі
Вхідні дані	-
Опис проведення тесту	Переходимо на сторінку з деталями події. Натискається відповідна кнопка "Поділитися" та обирається месенджер
Очікуваний результат	Діалог надсилання відкрився та було надіслане посилання на подію в обраному месенджері
Фактичний результат	Збігається з очікуваним

Таблиця 4.11 – Тест 1.11 Додати в календар

Початковий стан системи	Користувач авторизований у системі
Вхідні дані	-

Продовження таблиці 4.11

Опис проведення тесту	Переходимо на сторінку з деталями події. Натискається відповідна кнопка "Додати в календар" і надається дозвіл на використання
Очікуваний результат	Відкрився власний календар на телефоні користувача з автоматично введеними даними події
Фактичний результат	Збігається з очікуванням

Таблиця 4.12 – Тест 1.12 Налаштування push-сповіщень

Початковий стан системи	Користувач авторизований у системі
Вхідні дані	-
Опис проведення тесту	Переходимо на сторінку з налаштуваннями застосунку. Вмикаємо функцію отримання повідомлень
Очікуваний результат	Користувач періодично отримує повідомлення про рекомендації чи нагадування
Фактичний результат	Збігається з очікуванням

4.3 Опис контрольного прикладу

У попередньому підрозділі було протестовано основні функції реалізованого мобільного застосунку. В описі контрольного прикладу будемо відтворювати цю взаємодію користувача із застосунком у реальних умовах.

Для початку зареєструємо нового користувача за допомогою облікового запису Google. Для цього обираємо варіант “Увійти з Google” на початковій сторінці та надаємо дозвіл на використання цього сервісу (рис. 4.4).

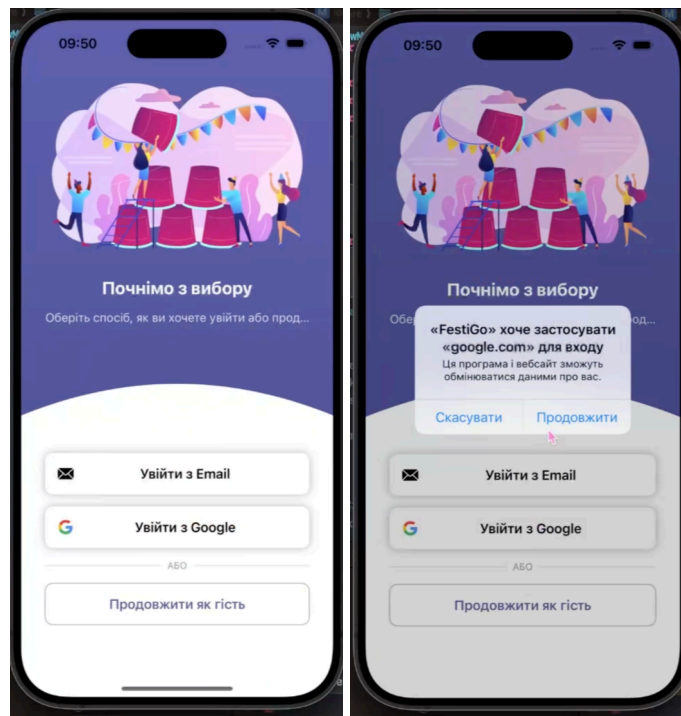


Рисунок 4.4 – Початкова сторінка застосунку

Так як це перший вхід у застосунок даним акаунтом, необхідно пройти опитування, що включає інформацію про інтереси, місцезнаходження та зручний час відвідування, для налаштування профілю. На рисунку 4.5 наведено декілька таких питань.

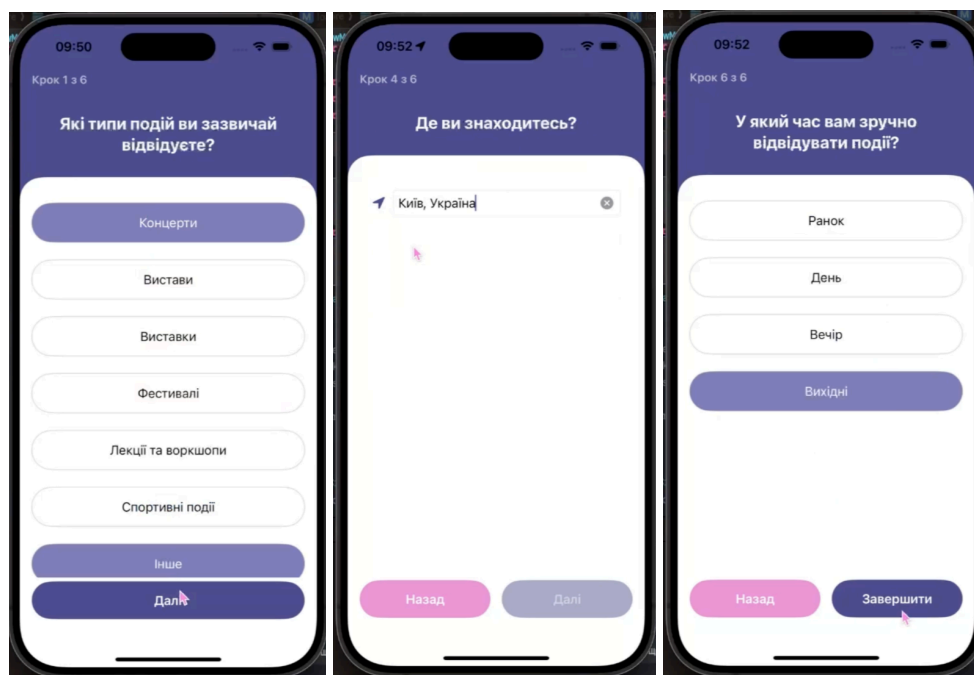


Рисунок 4.5 – Налаштування профілю

Після проходження, користувача переносить на головну сторінку (рис. 4.6), де відображаються, вже згенеровані системою, рекомендації на основі відповідей, а також усі інші події поруч. На тій же сторінці можна фільтрувати події, на основі різних категорій, міста, дати та типу.

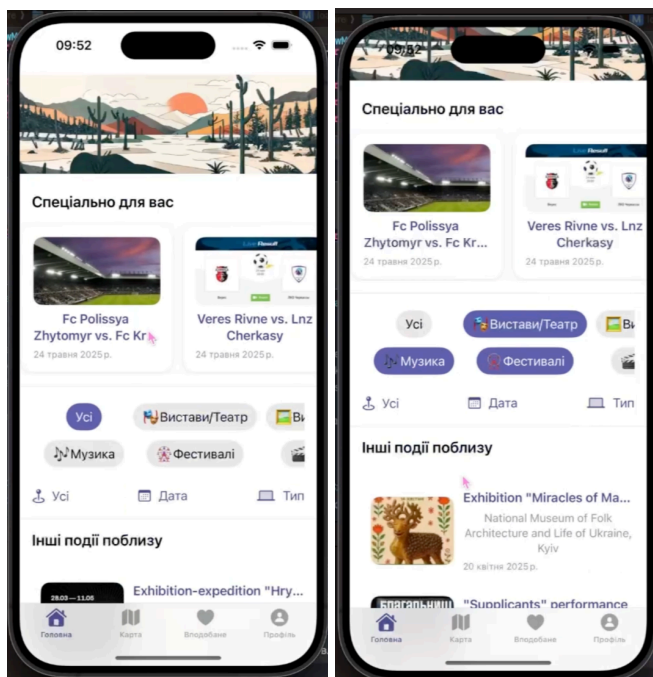


Рисунок 4.6 – Головна сторінка

Далі перейдемо на сторінку з картою подій, що відображає точну локацію кожної події (рис. 4.7).

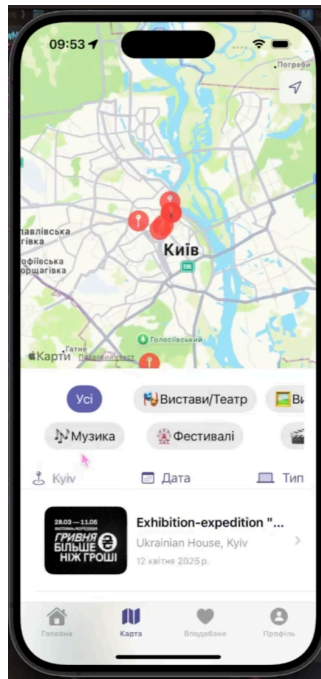


Рисунок 4.7 – Мапа подій

При виборі події, для авторизованого користувача надається можливість додати її до обраного, поділитися з друзями або ж додати у власний календар. Сторінка події зображена на рисунку 4.8.

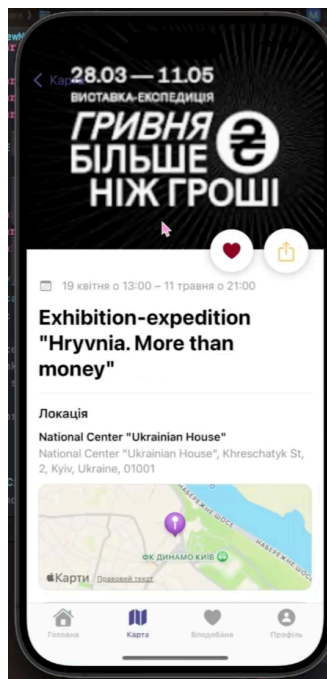


Рисунок 4.8 – Сторінка події

Так як ця подія була вподобана, вона буде відображатись на відповідній сторінці (рис. 4.9).

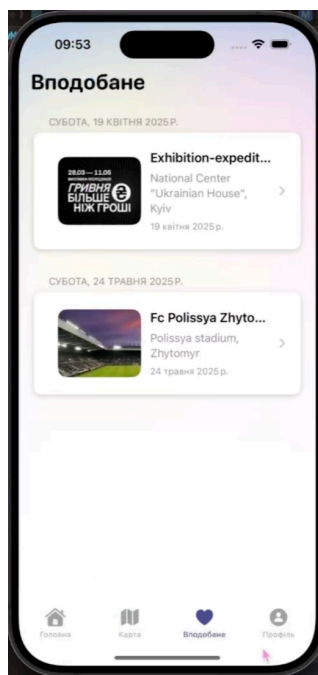


Рисунок 4.9 – Сторінка вподобаних подій

Висновки до розділу

У даному розділі було проведено оцінку якості та тестування розробленого мобільного застосунку. Аналіз якості програмного забезпечення здійснювався з використанням метрик, таких як цикломатична складність, кількість логічних рядків коду, рівень дублікації, індекси підтримуваності, надійності та безпеки, а також показники продуктивності.

Результати аналізу засвідчили, що структура коду є оптимальною для подальшого розширення функціоналу (середня цикломатична складність менше 10), код добре підтримується, має мінімальний рівень дублікацій та високі показники безпеки. Кількість логічних рядків коду відповідає типовому рівню складності для подібних застосунків. Продуктивність також перебуває на належному рівні: час запуску і час відгуку інтерфейсу в межах очікувань.

Також було проведено мануальне тестування і описаний контрольний приклад, що охоплює основні сценарії користувача, включаючи реєстрацію та авторизацію, налаштування профілю, перегляд і фільтрацію подій, персоналізовані рекомендації, перегляд деталей, збереження вподобаних подій, додавання до календаря та керування push-сповіщеннями. За результатами тестування, усі функції працюють згідно з вимогами та очікуваним функціоналом, без критичних збоїв і помилок.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Програмне забезпечення дипломного проєкту складається з двох основних компонентів: мобільного застосунку для iOS та серверної частини на Python.

5.1.1 Розгортання серверної частини

Серверна частина розгортається на хостинговій платформі Render, що забезпечує автоматичне масштабування, моніторинг та зручний деплой через GitHub.

Першим етапом є налаштування змінних середовища та створення файлу requirements.txt із переліком необхідних бібліотек та їх версій. Після цього вихідний код розміщується в GitHub.

На платформі Render створюється новий сервіс із підключеним репозиторієм як джерелом. Вказується команда запуску сервера, додаються змінні середовища та JSON-файл з ключем для доступу до Firebase.

Після кожного push у головну гілку репозиторію Render автоматично виконує деплой (рис. 5.1).

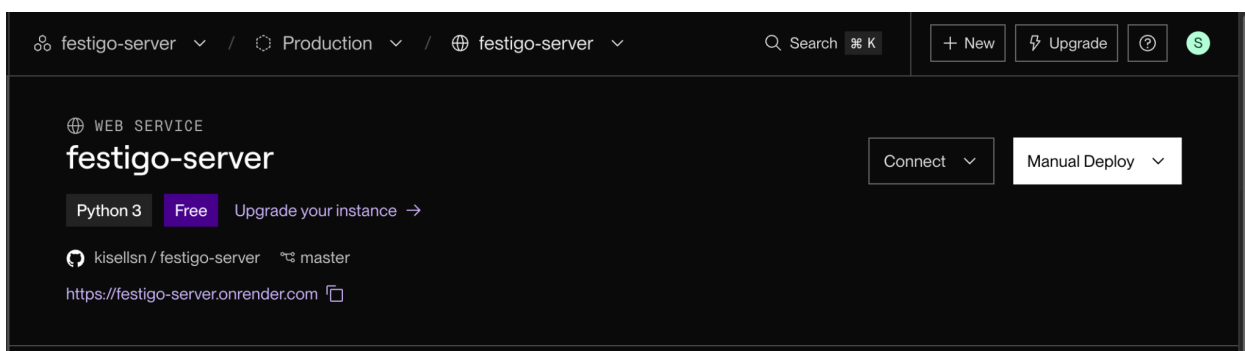


Рисунок 5.1 – Розгортання серверу на Render

5.1.2 Розгортання мобільного застосунку

Для початку потрібно зареєструвати обліковий запис Apple Developer, що надає доступ до App Store Connect – платформи для керування мобільними застосунками.

Після входу створюється новий застосунок у розділі My Apps і заповнюється основна інформація про нього (назва, платформа, мова, ідентифікатор проєкту в Xcode тощо). Наступним кроком є цифровий підпис застосунку, що підтверджує авторські права. Для цього використовуються сертифікат для підпису релізної версії, унікальний ідентифікатор (Bundle ID) та provisioning profile, який дозволяє запускати застосунок на фізичних пристроях і завантажувати його в App Store.

Після налаштування і підпису у середовищі Xcode створюється збірка застосунку, яка архівується та завантажується до App Store Connect. Далі застосунок проходить внутрішнє тестування за допомогою TestFlight, під час якого перевіряється його стабільність та відповідність вимогам.

Завершальним етапом є подання збірки на модерацію. У разі схвалення від Apple застосунок стає доступним для завантаження користувачами через App Store.

5.2 Супровід програмного забезпечення

Інструкція користувача наведена в окремому документі «Керівництво користувача».

Супровід програмного забезпечення передбачає забезпечення стабільної роботи мобільного застосунку та серверної частини, усунення виявлених помилок, оновлення функціональності відповідно до змін вимог користувачів, а також підтримку сумісності з новими версіями операційних систем та зовнішніх сервісів (наприклад, API подій чи Firebase).

Платформа Render надає інструменти для моніторингу працездатності серверної частини, а саме автоматичного відстеження помилок, логів та продуктивності сервісу. У разі збоїв або перевищення лімітів відправляються сповіщення. Для оновлення серверної частини достатньо викласти оновлений коду GitHub репозиторій. Оновлення серверної частини відбувається за таким алгоритмом:

- внесення змін до вихідного коду на локальному середовищі;

- тестування функціональності на тестовому середовищі;
- завантаження змін у головну гілку GitHub-репозиторію;
- Render автоматично виконує збірку та деплой нової версії.

Цей підхід дозволяє швидко впроваджувати нові функції або виправлення без зупинки сервісу.

Процес оновлення мобільного застосунку включає кілька етапів:

- внесення змін у вихідний код проєкту (нова функціональність, виправлення помилок, UI/UX покращення тощо);
- тестування оновлення локально, а потім через систему внутрішнього тестування TestFlight;
- створення нової збірки застосунку у Xcode, її підпис і завантаження в App Store Connect;
- заповнення опису версії та подання застосунку на перевірку;
- після проходження модерації Apple оновлення стає доступним у App Store.

Далі користувач бачить сповіщення про доступне оновлення у App Store та може вручну його встановити. Якщо ж в налаштуваннях його пристрою активовано автоматичне оновлення застосунків, нова версія буде завантажена автоматично.

Висновки до розділу

У даному розділі було розглянуто процеси розгортання та супроводу програмного забезпечення, що складається з двох основних компонентів: серверної частини, реалізованої на мові програмування Python, та мобільного застосунку для iOS.

Було описано розгортання серверу на хмарній платформі Render. Ця платформа забезпечує зручну інтеграцію з GitHub, автоматичний деплой, масштабування та моніторинг продуктивності з можливістю оперативного оновлення без зупинки сервісу.

Також розглянуто процес публікації мобільного застосунку через App Store Connect, який передбачає підготовку застосунку в Xcode, його підпис, завантаження до App Store, тестування через TestFlight та проходження модерації. Завдяки дотриманню цих кроків забезпечується надійна доставка оновлень кінцевим користувачам.

Таким чином, описані механізми дозволяють ефективно впроваджувати зміни, підтримувати стабільну роботу програмного забезпечення та швидко реагувати на зміну вимог або виявлені помилки.

ВИСНОВКИ

У ході виконання дипломного проєкту було розроблено мобільний застосунок Festigo, призначений для персоналізованих рекомендацій культурних подій. Метою проєкту, що полягає у підвищенні поінформованості про актуальні події та спрощення доступу до інформації про культурні заходи через персоналізовані рекомендації, було досягнуто.

У першому розділі було проведено передпроектне обстеження предметної області, проаналізовано проблеми користувачів у пошуку культурних подій, визначено обмеження існуючих рішень та сформульовано вимоги до застосунку. Проведено огляд аналогів, визначено ключові бізнес-процеси та змодельовані в BPMN.

У другому розділі були описані функціональні, нефункціональні та системні вимоги, сценарії використання та основні взаємодії користувача із системою. Проведено оцінку економічних показників і сформульовано технічне завдання.

У третьому розділі описано архітектуру ПЗ: мобільний застосунок реалізовано на MVVM із SwiftUI, серверна частина – на FastAPI (Python). Описано структуру бекенду, базу даних Firebase Firestore та модуль рекомендацій із контентною фільтрацією за інтересами, локацією і часом.

Четвертий розділ присвячено аналізу якості коду та проведенню тестування, завдяки якому було підтверджено відповідність розробленого програмного забезпечення основним нефункціональним вимогам.

У п'ятому розділі описано розгортання і супровід: сервер розміщено на Render, мобільний застосунок підготовлено до публікації в App Store. Надано рекомендації щодо оновлення контенту, технічної підтримки та масштабування системи.

Функціональні задачі застосунку, визначені на етапі проєктування, реалізовано в повному обсязі. А саме: реалізовано реєстрацію та авторизацію (у тому числі через Google), налаштування профілю інтересів, персоналізовану видачу подій, інтерактивну мапу, фільтрацію за параметрами, збереження подій в обране, додавання в календар і можливість поширення.

Особливу увагу приділено українському контексту: інтерфейс застосунку повністю локалізовано, а події охоплюють реальні локації українських міст, що відрізняє рішення від більшості аналогів, орієнтованих на закордонний ринок.

Таким чином, розроблений застосунок є конкурентоздатним рішенням, що має значний потенціал для масштабування та комерціалізації у сфері культурних сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) De Pessemier T., Coppens S., Geebelen K. Collaborative recommendations with content-based filters for cultural activities via a scalable event distribution platform. *Multimed Tools Appl.* 2012. Vol. 58. P. 167–213. DOI: <https://doi.org/10.1007/s11042-010-0715-8>
- 2) Мелешко Є.В., Міхав В.В. Рекомендаційні системи у складних комп'ютерних мережах: навч. посіб. Кропивницький: ЦНТУ, 2025. 155 с.
- 3) Чередніченко О.Ю., Янголенко О.В., Іващенко О.В., Матвеев О.М. Моделі формування рекомендацій у інтелектуальних системах електронної комерції. Системи обробки інформації. 2020. С. 32–39. DOI: 10.30748/soi.2020.160.04
- 4) Eventbrite [Електронний ресурс]. URL: <https://www.eventbrite.com/> (дата звернення: 17.04.2025).
- 5) Ticketmaster [Електронний ресурс]. URL: <https://www.ticketmaster.pl/> (дата звернення: 17.04.2025).
- 6) Fever [Електронний ресурс]. URL: <https://feverup.com/> (дата звернення: 17.04.2025).
- 7) Facebook Events [Електронний ресурс]. URL: <https://www.facebook.com/events/> (дата звернення: 17.04.2025).
- 8) Ko H., Lee S., Park Y., Choi A. A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields. *Electronics*. 2022. Vol. 11. Article 141. DOI: <https://doi.org/10.3390/electronics11010141>
- 9) Roy D., Dutta M. A systematic review and research perspective on recommender systems. *Journal of Big Data*. 2022. Vol. 9. Article 59. DOI: <https://doi.org/10.1186/s40537-022-00592-5>
- 10) Fayyaz Z., Ebrahimian M., Nawara D., Ibrahim A., Kashef R. Recommendation Systems: Algorithms, Challenges, Metrics, and Business Opportunities. *Applied Sciences*. 2020. Vol. 10. Article 7748. DOI: <https://doi.org/10.3390/app10217748>

- 11) Fojtik R. Swift a New Programming Language for Development and Education. 2020. DOI: 10.1007/978-3-030-37737-3_26
- 12) Vujevic T. The mobile app architecture guide. Decode Agency. 2023 [Електронний ресурс]. URL: <https://decode.agency/article/mobile-app-architecture/> (дата звернення: 28.04.2025).
- 13) Nunkesser R. Choosing a Global Architecture for Mobile Applications [Електронний ресурс]. TechRxiv. 2021. 19 березня. DOI: 10.36227/techrxiv.14212571.v1
- 14) Indrawan D., Kusumo D.S., Puspitasari S. Analysis of the Implementation of MVVM Architecture Pattern on Performance of iOS Mobile-Based Applications. Jurnal Ilmiah Penelitian dan Pembelajaran Informatika. 2023. T. 8, № 1. DOI: <https://doi.org/10.29100/jipi.v8i1.3293>
- 15) Ferrer García R. iOS Architecture Patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift. 1-е вид. Berkeley, CA: Apress, 2023. XVIII, 397 с. (Professional and Applied Computing, Apress Access Books). ISBN 978-1-4842-9069-9. DOI: <https://doi.org/10.1007/978-1-4842-9069-9>
- 16) Jacobsson P. Building scalable real time applications with Firestore [Електронний ресурс]. Google Cloud Blog. 2022. 21 жовтня. URL: <https://cloud.google.com/blog/products/databases/building-scalable-real-time-applications-with-firestore> (дата звернення: 21.04.2025).
- 17) Сидоров М.О., Родіонов П.Ю., Марченко О.І. Економіка ІТ-індустрії та підприємництво: практикум. Київ: КПІ ім. Ігоря Сікорського, 2024. 100 с.
- 18) Firebase documentation [Електронний ресурс]. URL: <https://firebase.google.com/docs/firestore> (дата звернення: 21.04.2025).
- 19) Bass L., Clements P., Kazman R. Software Architecture in Practice. 3rd ed. Addison-Wesley, 2012.

- 20) Blinowski G., Ojdowska A., Przybyłek A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. IEEE Access. 2022. Vol. 10. P. 20357–20374. DOI: 10.1109/ACCESS.2022.3152803
- 21) Mwamba Nyabuto G., Mony V., Mbugua S. Architectural review of client-server models. International Journal of Scientific Research & Engineering Trends. 2024. Vol. 10, № 1. C. 1–7. URL: https://d1wqtxts1xzle7.cloudfront.net/111170439/Architectural_Review_of_Client_Server_Models-libre.pdf (дата звернення: 01.05.2025)
- 22) C4 Model [Електронний ресурс]. URL: <https://c4model.com/> (дата звернення: 02.05.2025)
- 23) Scikit-learn TfidfVectorizer documentation [Електронний ресурс]. URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html (дата звернення: 06.05.2025).
- 24) Suman S., Riya, Chakravorty C. Content Based Recommender System Using Machine Learning. In: Pandit M., Gaur M.K., Rana P.S., Tiwari A. (eds) Artificial Intelligence and Sustainable Computing. Algorithms for Intelligent Systems. Singapore: Springer, 2022. DOI: https://doi.org/10.1007/978-981-19-1653-3_12
- 25) Munshi, A., Ameen, M. M., Gohil, D., & Gopalan, N. P. (2021). Enactment of TF-IDF and Word2Vec on Text Categorization. In: Proceedings of the 6th International Conference on Inventive Computation Technologies (ICICT 2021). Lecture Notes in Networks and Systems, vol. 228. Singapore: Springer, pp. 181–188. DOI: https://doi.org/10.1007/978-981-15-9712-1_17.
- 26) Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. Knowledge-Based Systems, 46, 109–132. DOI: <https://doi.org/10.1016/j.knosys.2013.03.012>.
- 27) FastAPI [Електронний ресурс]. URL: <https://fastapi.tiangolo.com/> (дата звернення: 28.05.2025)

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

Дата звіту
Дата редагування

5/30/2025
5/30/2025

Документ прийнятий

Звіт подібності

метадані

Назва організації

National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute

Заголовок

ІП-15_Кондрацька_ПЗ

Автор

Науковий керівник / Експерт

КондрацькаКовтунець О.В.

підрозділ

ФІОТ, К-а інформатики та програмної інженерії

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

14.51%

14.51%

КП 1

10

Довжина фрази для коефіцієнта подібності 2

10506

Кількість слів

82261

Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв

0

Інтервали

0

Мікропробіли

0

Білі знаки

0

Парафрази (SmartMarks)

79

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз

Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)	
1	ІП-13_Макарчук_ПЗ 5/27/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (ФІОТ, К-а інформатики та програмної інженерії)	118 1.12 %	
2	ІП-11_Трикош_ПЗ 5/28/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (ФІОТ, К-а інформатики та програмної інженерії)	80 0.76 %	

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

**МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПЕРСОНАЛІЗОВАНИХ
РЕКОМЕНДАЦІЙ КУЛЬТУРНИХ ПОДІЙ**

Текст програми

КПІ.ІП-1510.045490.03.12

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олесь КОВТУНЕЦЬ

Нормоконтроль:

_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:

_____ Соня КОНДРАЦЬКА

Київ – 2025

Посилання на репозиторій з повним текстом програмного коду

Мобільний застосунок – <https://github.com/kisellsn/FestiGo>

Сервер – <https://github.com/kisellsn/festigo-server>

Файл Core/Survey/OnboardingViewModel.swift

Реалізація опитування та збереження відповідей

```
class OnboardingViewModel: ObservableObject {
    @Published var currentStep = 0
    @Published var answers: [Int: [String]] = [:]
    @Published var questions: [OnboardingQuestion] = []
    @Published var selectedLocationCoordinates: (lat: Double, lon: Double)? = nil

    init() {
        loadQuestions()
    }

    func loadQuestions() {
        if let url = Bundle.main.url(forResource: "questions", withExtension: "json"),
            let data = try? Data(contentsOf: url),
            let loadedQuestions = try? JSONDecoder().decode([OnboardingQuestion].self, from:
data) {
            self.questions = loadedQuestions
        } else {
            print("Failed to load onboarding questions")
        }
    }

    func setLocationAnswer(for step: Int, title: String, coordinates: (Double, Double)?) {
        answers[step] = [title]
        if step == 3 {
            selectedLocationCoordinates = coordinates
        }
    }

    func finalAnswers() -> [String: Any] {
        var result: [String: Any] = [:]

        for question in questions {
            if let answer = answers[question.id] {
                result["\(question.id)"] = answer
            }
        }

        if let location = answers[3]?.first {
            var locationDict: [String: Any] = ["title": location]
            if let coords = selectedLocationCoordinates {
                locationDict["lat"] = coords.lat
                locationDict["lon"] = coords.lon
            }
            result["3"] = [locationDict]
        }
    }
}
```

```

    }

    return result
}

func saveCurrentUserAnswers(completion: @escaping (Bool) -> Void) {
    guard let userId = Auth.auth().currentUser?.uid else {
        print("User not logged in.")
        completion(false)
        return
    }

    saveAnswers(for: userId, answers: finalAnswers(), completion: completion)
}

func saveAnswers(for userId: String, answers: [String: Any], completion: @escaping (Bool)
-> Void) {
    let db = Firestore.firestore()
    let responseData: [String: Any] = [
        "userId": userId,
        "timestamp": Date().timeIntervalSince1970,
        "answers": answers
    ]

    db.collection("onboardingResponses")
        .document(userId)
        .setData(responseData) { error in
            if let error = error {
                print("Failed to save answers:", error.localizedDescription)
                completion(false)
            } else {
                print("Answers saved!")
                completion(true)
            }
        }

    TokenManager.shared.sendUserIdToBackend(userId)
    DispatchQueue.main.asyncAfter(deadline: .now() + 0.5) {
        UserProfileService.shared.initProfile()
    }
}

func isCurrentStepValid(with locationManager: LocationSearchManager? = nil) -> Bool {
    let question = questions[currentStep]

    switch question.inputType {
    case .multipleChoice, .singleChoice:
        return answers[currentStep]?.isEmpty == false
    case .textInput:
        let text = answers[currentStep]?.first ?? ""
        return !text.trimmingCharacters(in: .whitespaces).isEmpty
    case .location:
        guard let locationManager = locationManager else { return false }

```

```

        let query = locationManager.query.trimmingCharacters(in: .whitespaces)
        let savedAnswer = answers[currentStep]?.first ?? ""
        return !query.isEmpty && query == savedAnswer && query.isSafeInput
    }
}
}

```

Файл `recommendation/user_profile.py`

Реалізація створення та оновлення вектору профілю інтересів користувача

```

def build_profile_vector(user_id):
    """
    creates profile vector for new user based on interests from onboarding responses
    """
    onboarding_doc = db.collection("onboardingResponses").document(user_id).get()
    if not onboarding_doc.exists:
        return None

    onboarding = onboarding_doc.to_dict()
    answers = onboarding.get("answers", {})

    # translate and normalize
    selected_categories = [CATEGORY_TRANSLATION.get(x, "other") for x in answers.get("0",
    [])]
    selected_genres_music = [GENRE_TRANSLATION.get(x, "other") for x in answers.get("1",
    [])]
    selected_genres_theater = [GENRE_TRANSLATION.get(x, "other") for x in answers.get("2",
    [])]

    profile_texts = {
        "main_categories": selected_categories,
        "genres": selected_genres_music + selected_genres_theater,
    }

    profile_components = {}
    for field, selected_items in profile_texts.items():
        vector = genre_vector(selected_items) if field == "genres" else
        category_vector(selected_items)
        profile_components[field] = vector

    # print(profile_components)
    db.collection("users").document(user_id).update({
        "component_profile_vectors": profile_components
    })
    return profile_components

def update_profile_vector(user_id: str):
    """
    Updates the user's profile vector by averaging the onboarding profile with
    the one derived from liked events.
    """
    user_doc = db.collection("users").document(user_id).get()

```

```

if not user_doc.exists:
    return None

user_data = user_doc.to_dict()
existing_profile = user_data.get("component_profile_vectors")
if not existing_profile:
    build_profile_vector(user_id)

# Завантаження улюблених подій
favourite_events_ref =
db.collection("users").document(user_id).collection("favourite_events")
favourite_events = list(favourite_events_ref.stream())

if not favourite_events:
    return None

liked_categories = set()
liked_genres = set()

for fav_event in favourite_events:
    fav_data = fav_event.to_dict()
    event_id = fav_data.get("id")
    if not event_id:
        continue

    event_doc = db.collection("events").document(event_id).get()
    if not event_doc.exists:
        continue

    event_data = event_doc.to_dict()
    liked_categories.update(event_data.get("main_categories", []))
    liked_genres.update(event_data.get("genres", []))

# new vector
new_profile_components = {
    "main_categories": category_vector(list(liked_categories)),
    "genres": genre_vector(list(liked_genres))
}
# print(new_profile_components)
# avg з існуючим профілем
averaged_profile = {}
for key in new_profile_components:
    old_vector = existing_profile.get(key)
    new_vector = new_profile_components[key]
    if not old_vector or len(old_vector) != len(new_vector):
        averaged_profile[key] = new_vector # fallback
    else:
        averaged = [0.3 * old + 0.7 * new for old, new in zip(old_vector, new_vector)]
        averaged_profile[key] = averaged

# print(averaged_profile)
db.collection("users").document(user_id).update({
    "component_profile_vectors": averaged_profile
})

```

```
}}
```

```
return averaged_profile
```

Файл API/UserProfileService.swift

Реалізація запитів до серверу для створення/оновлення профілю та отримання рекомендацій

```
class UserProfileService {
    static let shared = UserProfileService()
    private init() {}
    func initProfile(completion: ((Bool) -> Void)? = nil) {
        guard let request = TokenManager.shared.authorizedRequest(to: "/user/init_profile",
method: "POST") else {
            print(" Failed to create request for init_profile")
            completion?(false)
            return
        }

        URLSession.shared.dataTask(with: request) { data, response, error in
            if let error = error {
                print(" Init profile error: \(error.localizedDescription)")
                completion?(false)
            } else if let httpResponse = response as? HTTPURLResponse {
                print("Init profile status: \(httpResponse.statusCode)")
                completion?(httpResponse.statusCode == 200)
            }
        }.resume()
    }

    func updateProfile(completion: ((Bool) -> Void)? = nil) {
        guard let request = TokenManager.shared.authorizedRequest(to: "/user/update_profile",
method: "POST") else {
            print(" Failed to create request for update_profile")
            completion?(false)
            return
        }
    }
}
```

```

URLSession.shared.dataTask(with: request) { data, response, error in
    if let error = error {
        print("Update profile error: \(error.localizedDescription)")
        completion?(false)
    } else if let httpResponse = response as? HTTPURLResponse {
        print(" Update profile status: \(httpResponse.statusCode)")
        completion?(httpResponse.statusCode == 200)
    }
}.resume()
}

```

```

func getRecommendations(completion: @escaping ([String]) -> Void) {
    guard let request = TokenManager.shared.authorizedRequest(to:
"/user/get_recommendations") else {
        print(" Failed to create request for get_recommendations")
        completion([])
        return
    }
}

```

```

URLSession.shared.dataTask(with: request) { data, response, error in
    if let error = error {
        print("Get recommendations error: \(error.localizedDescription)")
        completion([])
        return
    }
}

```

```

guard let data = data else {
    print(" No data received")
    completion([])
    return
}

```

```

do {
    let json = try JSONSerialization.jsonObject(with: data) as? [String: Any]
}

```



```

        let recommendations = json?["recommendations"] as? [String] ?? []
        print(" Recommendations received:", recommendations)
        completion(recommendations)
    } catch {
        print(" JSON parsing error: \(error.localizedDescription)")
        completion([])
    }
    }.resume()
}
}

```

Файл Core/Main/Event/EventListViewModel.swift

Реалізація головної сторінки з можливістю перегляду всіх доступних подій, персоналізованих рекомендацій та фільтрацією.

@MainActor

```

class EventListViewModel: ObservableObject {
    @Published private(set) var events: [Event] = []
    private var lastDocument: DocumentSnapshot? = nil
    @Published private(set) var didLoadRecommendations = false

    @Published var selectedCategories: [String]? = nil
    @Published var isOnline: TypeOption? = nil    // nil = all, true = only online, false = only
offline
    @Published var selectedCity: CityOption = .all
    @Published var allCitiesMap: [String: CLLocationCoordinate2D?] = [:]

    @Published var startDate: Date? = nil
    @Published var endDate: Date? = nil
    @Published var dateRange: ClosedRange<Date>? {
        didSet {
            getEvents(reset: true)
        }
    }

    @Published var recommendedEventIds: [String] = []
    @Published var recommendedEvents: [Event] = []

```

```

@Published var isLoadingEvents: Bool = false

private var firestoreListener: ListenerRegistration? = nil
private var cancellables = Set<AnyCancellable>()

init() {
    fetchAllCities()
    Task {
        await applyUserDefaultCity()
    }
}

func loadRecommendationsIfNeeded() {
    guard let user = Auth.auth().currentUser else {
        print("🔒 Користувач не авторизований — рекомендації не потрібні")
        return
    }
    Task {
        do {
            let isOnboardingCompleted = try await
                UserManager.shared.isOnboardingCompleted(userId: user.uid)

            guard isOnboardingCompleted else {
                print("📝 Користувач ще не пройшов опитування — рекомендації не потрібні")
                return
            }
            self.fetchRecommendations()
        } catch {
            print("Помилка при перевірці опитування: \(error)")
        }
    }
}

deinit {
    firestoreListener?.remove()
}

func updateDateRange() {

```

```

    if let start = startDate, let end = endDate {
        dateRange = start...end
    }
}

@Published var selectedEventOnly: Event? = nil

//MARK: - Category filter
enum CategoryOption: String, CaseIterable {
    case all
    case music
    case show
    case festival
    case exhibition
    case cinema
    case education
    case sport

    var label: LocalizedStringResource {
        switch self {
            case .all: return "Все"
            case .music: return "🎵Музика"
            case .show: return "🎭Вистави/Театр"
            case .festival: return "🎪Фестивалі"
            case .exhibition: return "🖼️Виставки"
            case .cinema: return "🎬Кіно"
            case .education: return "📖Освіта"
            case .sport: return "⚡️Спорт/Активний відпочинок"
        }
    }
}

func categorySelected(option: CategoryOption) async throws {
    if option == .all {
        selectedCategories = nil
    } else {

```

```

    if selectedCategories == nil {
        selectedCategories = [option.rawValue]
    } else if selectedCategories!.contains(option.rawValue) {
        selectedCategories!.removeAll { $0 == option.rawValue }
        if selectedCategories!.isEmpty {
            selectedCategories = nil
        }
    } else {
        selectedCategories!.append(option.rawValue)
    }
}

self.events = []
self.lastDocument = nil
self.getEvents(reset: true)
}

//MARK: - Type filter
enum TypeOption: String, CaseIterable {
    case noType = "Усі"
    case online = "Онлайн"
    case offline = "Офлайн"

    var typeKey: Bool? {
        switch self {
            case .noType: return nil
            case .online: return true
            case .offline: return false
        }
    }
}

func typeSelected(option: TypeOption) async throws {
    self.isOnline = option
    self.events = []
    self.lastDocument = nil

```

```

        self.getEvents(reset: true)

    }

//MARK: - City filter
enum CityOption: Equatable {
    case all
    case named(String)

    var value: String? {
        switch self {
            case .all: return nil
            case .named(let name): return name
        }
    }

    var label: String {
        switch self {
            case .all: return "Yci"
            case .named(let name): return name
        }
    }

    static func == (lhs: CityOption, rhs: CityOption) -> Bool {
        return lhs.value == rhs.value
    }
}

func fetchAllCities() {
    Task {
        do {
            self.allCitiesMap = try await EventsManager.shared.getAllCities()
        } catch {
            print("Error fetching cities: \(error)")
        }
    }
}

```

```

    }
}

func citySelected(option: CityOption) async throws {
    self.selectedCity = option
    self.events = []
    self.lastDocument = nil
    self.getEvents(reset: true)
}

func applyUserDefaultCity() async {
    guard let userId = Auth.auth().currentUser?.uid else { return }

    do {
        let (userCity, userRadius) = try await
UserManager.shared.getUserCityAndRadiusFromOnboardingResponses(userId: userId)

        guard let userCity = userCity else {
            try? await citySelected(option: .all)
            return
        }

        var userCoord: CLLocationCoordinate2D?

        if let coord = allCitiesMap[userCity] {
            userCoord = coord
        } else {
            do {
                userCoord = try await LocationSearchManager.shared.getCoordinates(for: userCity)
            } catch {
                print("Failed to geocode city: \(error.localizedDescription)")
                try? await citySelected(option: .all)
                return
            }
        }
    }
}

```

```

guard let finalUserCoord = userCoord else {
    try? await citySelected(option: .all)
    return
}

let nearbyCities = allCitiesMap.compactMap { city, optionalCoord -> (String,
CLLocationDistance)? in
    guard let coord = optionalCoord, let radius = userRadius else { return nil }
    let distance = calculateDistance(from: finalUserCoord, to: coord)
    return distance <= radius ? (city, distance) : nil
}

if let closestCity = nearbyCities.min(by: { $0.1 < $1.1 }) {
    try await citySelected(option: .named(closestCity.0))
} else {
    try await citySelected(option: .all)
}

} catch {
    print(" Failed to apply city filter: \(error)")
    try? await citySelected(option: .all)
}
}

func calculateDistance(from: CLLocationCoordinate2D, to: CLLocationCoordinate2D) ->
CLLocationDistance {
    let loc1 = CLLocation(latitude: from.latitude, longitude: from.longitude)
    let loc2 = CLLocation(latitude: to.latitude, longitude: to.longitude)
    return loc1.distance(from: loc2) / 1000.0
}

//MARK: - load
var filteredEvents: [Event] {
    if let selectedOnly = selectedEventOnly {

```

```

        return [selectedOnly]
    }

    return events
}

func loadMockData() {
    events = MockEvents.loadEvents()
}

func addRealtimeEventsListener() {
    firestoreListener?.remove()

    firestoreListener = EventsManager.shared.addEventsListener(
        selectedCategories: selectedCategories,
        city: selectedCity.value,
        isOnline: isOnline?.typeKey,
        startDate: startDate,
        endDate: endDate
    ) { [weak self] newEvents in
        DispatchQueue.main.async {
            self?.events = newEvents
        }
    }
}

func getEvents(reset: Bool = false) {
    Task {
        do {
            if reset {
                self.events = []
                self.lastDocument = nil
            }
            self.isLoadingEvents = true

```



```

let (newEvents, lastDocument) = try await EventsManager.shared.getAllEvents(
    selectedCategories: selectedCategories,
    city: selectedCity.value,
    isOnline: isOnline?.typeKey,
    startDate: startDate,
    endDate: endDate,
    count: 10,
    lastDocument: lastDocument
)
self.events.append(contentsOf: newEvents)
if let lastDocument {
    self.lastDocument = lastDocument
}
self.isLoadingEvents = false
} catch {
    print("Error fetching events: \(error)")
}
}
}

```

```

func fetchRecommendations() {
    guard !didLoadRecommendations else { return }

```

```

    UserProfileService.shared.getRecommendations { [weak self] ids in
        DispatchQueue.main.async {
            self?.recommendedEventIds = ids
            self?.fetchRecommendedEvents()
            self?.didLoadRecommendations = true

```

```

        UserProfileService.shared.updateProfile { success in
            if success {
                print(" Профіль оновлено після рекомендацій")
            } else {
                print(" Update profile не вдалося")
            }

```

```

    }
  }
}

func fetchRecommendedEvents() {
  Task {
    do {
      let events = try await EventsManager.shared.getEventsByIds(ids:
recommendedEventIds)

      let ordered = recommendedEventIds.compactMap { id in
        events.first(where: { $0.id == id })
      }

      DispatchQueue.main.async {
        self.recommendedEvents = ordered
      }
    } catch {
      print(" Не вдалося завантажити рекомендовані події: \(error)")
    }
  }
}

```

Файл categorization/event_categorization.py

Реалізація категоризації подій

```

def detect_genres(event: Dict, main_categories: List[str]) -> List[str]:
    applicable = {"music", "cinema", "show"}
    if not any(cat in applicable for cat in main_categories):
        return []

    text = preprocess_text(
        event.get("name", ""),
        event.get("description", ""),
        event.get("link", ""),
        event.get("venue", {}).get("name", ""),
        event.get("publisher", ""),
        ''.join(event.get("venue", {}).get("subtypes") or [])
    )

```

```

scores = genre_model.decision_function([text])[0]
genres = genre_mlb.classes_

detected = [genre for genre, score in zip(genres, scores) if score > 0]

return sorted(set(detected))

def categorize_event(event: Dict) -> List[Tuple[str, float]]:
    text = preprocess_text(
        event.get("name", ""),
        event.get("description", ""),
        event.get("link", ""),
        event.get("venue", {}).get("name", ""),
        event.get("publisher", ""),
        ''.join(event.get("venue", {}).get("subtypes") or [])
    )

    # --- Model scores ---
    model_scores_raw = category_model.decision_function([text])[0]
    model_categories = category_mlb.classes_
    model_scores = {cat: float(score) for cat, score in zip(model_categories, model_scores_raw) if
score > 0}

    # --- Keyword scores ---
    keyword_scores = {}
    for category, keywords in CATEGORY_KEYWORDS.items():
        score = 0.0
        for keyword, weight in keywords.items():
            if re.search(rf"\b{re.escape(keyword)}\b", text):
                score += weight
        if score > 0:
            keyword_scores[category] = round(score, 3)

    # --- Combined ---
    combined_scores = {}
    for category in set(model_scores) | set(keyword_scores):
        score_model = model_scores.get(category, 0)
        score_keyword = keyword_scores.get(category, 0)
        combined_scores[category] = round(0.3 * score_model + 0.7 * score_keyword, 3)

    return sorted(combined_scores.items(), key=lambda x: x[1], reverse=True)

def assign_categories_to_events(events: List[Dict]) -> List[Dict]:
    enriched_events = []

    for event in events:
        categories = categorize_event(event)
        categories_dicts = [{"category": cat, "score": score} for cat, score in categories]
        event["categories_scored"] = categories_dicts

```

```

main_categories = [c["category"] for c in categories_dicts if c["score"] >= 0.4]
event["main_categories"] = main_categories

event["genres"] = detect_genres(event, main_categories)
enriched_events.append(event)

return enriched_events

```

Файл `app/scheduler.py`

Реалізація планувальника подій

```

from apscheduler.schedulers.background import BackgroundScheduler
from services.fetcher import fetch_and_store_events
from app.config import SCHEDULE_INTERVAL_HOURS,
SCHEDULE_DELETE_INTERVAL_HOURS
from services.firestore_client import delete_expired_events

```

```

def start_scheduler():
    scheduler = BackgroundScheduler()
    scheduler.add_job(fetch_and_store_events, 'interval',
hours=SCHEDULE_INTERVAL_HOURS)
    scheduler.add_job(delete_expired_events, 'interval',
hours=SCHEDULE_DELETE_INTERVAL_HOURS)
    scheduler.start()

```

Файл `recommendation/vectorizer.py`

Реалізація вилучення векторів з подій

```

def category_vector(selected_categories):
    """ Створює вектор для вибраних категорій на основі фіксованого списку. """
    return [1 if category in selected_categories else 0 for category in ALL_CATEGORIES]

def genre_vector(selected_genres):
    """ Створює вектор для вибраних жанрів на основі фіксованого списку """
    return [1 if genre in selected_genres else 0 for genre in ALL_GENRES]

def vectorize_field(texts, stop_words="english"):
    vectorizer = TfidfVectorizer(stop_words=stop_words)
    matrix = vectorizer.fit_transform(texts)
    return matrix.toarray(), vectorizer

def build_event_components(event):
    fields = {
        "name": event.get("name", ""),

```

```

    "description": event.get("description", ""),
    "main_categories": " ".join(event.get("main_categories", [])),
    "genres": " ".join(event.get("genres", [])),
    "venue_name": event.get("venue", {}).get("name", ""),
    "venue_subtypes": " ".join(event.get("venue", {}).get("subtypes", [])),
    "isVirtual": "virtual" if event.get("isVirtual") else ""
}
return {k: v.lower() for k, v in fields.items()}

```

```
def extract_event_fields_for_vectorization(event):
```

```

    def get_text(val):
        if isinstance(val, list):
            return " ".join(val)
        elif isinstance(val, bool):
            return "online" if val else "offline"
        elif isinstance(val, dict):
            return " ".join(str(v) for v in val.values() if isinstance(v, str))
        return str(val) if val else ""

```

```

    return {
        "name": get_text(event.get("name", "")),
        "description": get_text(event.get("description", "")),
        "main_categories": get_text(event.get("main_categories", [])),
        "genres": get_text(event.get("genres", [])),
        "isVirtual": get_text(event.get("isVirtual", False)),
        "venue_name": get_text(event.get("venue", {}).get("name", "")),
        "venue_subtypes": get_text(event.get("venue", {}).get("subtypes", []))
    }

```

```
def generate_events_vectors(events: List[Event]):
```

```

    raw_events = [extract_event_fields_for_vectorization(e.model_dump()) for e in events]
    field_names = raw_events[0].keys()
    field_vectors = {}
    vectorizers = {}

```

```

for field in field_names:
    if field == "main_categories":
        field_vectors[field] = [category_vector(e[field]) for e in raw_events]
    elif field == "genres":
        field_vectors[field] = [genre_vector(e[field]) for e in raw_events]
    else:
        # TfidfVectorizer
        texts = [e[field].lower() for e in raw_events]
        vectors, vec = vectorize_field(texts)
        field_vectors[field] = vectors
        vectorizers[field] = vec

for i, event in enumerate(events):
    component_vectors = {
        field: field_vectors[field][i] if isinstance(field_vectors[field][i], list) else
        field_vectors[field][i].tolist()
        for field in field_names
    }
    event.component_vectors = component_vectors

return events

```

Файл recommendation/recommendation_engine.py

Реалізація системи рекомендацій

```

def is_event_time_suitable(event, preferences):
    start_time = event.get("startTime")
    if not isinstance(start_time, datetime):
        return False

    preferred_times = preferences.get("time", [])
    for bucket in preferred_times:
        mapped = TIME_LABELS_UA.get(bucket)
        if not mapped:
            continue

        if mapped == "weekend":
            if start_time.weekday() in TIME_BUCKETS[mapped]:
                return True
        else:
            low, high = TIME_BUCKETS[mapped]
            if low <= start_time.hour < high:
                return True

```

```

return False

def is_within_distance(user_loc, event_loc, max_km):
    if not user_loc or not event_loc or None in user_loc or None in event_loc:
        return False, None
    try:
        dist = geodesic(user_loc, event_loc).km
        return dist <= max_km, dist
    except Exception:
        return False, None

def recommend_events_for_user(user_id, top_n=20):
    """
    create recommendations for user

    :param user_id
    :param top_n: count of recommendations
    :return: list of recommended events
    """

    # ----- user info
    user_doc = db.collection("users").document(user_id).get()
    if not user_doc.exists:
        return []

    onboarding_doc = db.collection("onboardingResponses").document(user_id).get()
    if not onboarding_doc.exists:
        return []

    responses = onboarding_doc.to_dict()
    answers = responses.get("answers", {})

    # --- координати
    try:
        location_data = answers.get("3")[0]
        user_location_coords = (location_data["lat"], location_data["lon"])
    except (IndexError, KeyError, TypeError):
        user_location_coords = None

    # --- максимальна відстань
    try:
        distance_answer = answers.get("4", [""])[0]
        digits = re.findall(r"\d+", distance_answer)
        max_distance_km = int(digits[0]) if digits else DEFAULT_DISTANCE_KM
    except Exception:
        max_distance_km = DEFAULT_DISTANCE_KM

    preferred_times = answers.get("5", [])

    profile_components = user_doc.to_dict().get("component_profile_vectors")
    if not profile_components:
        profile_components = build_profile_vector(user_id)

```

```

    if not profile_components:
        return []

    # ----- fav events info
    liked_events_docs =
db.collection("users").document(user_id).collection("favourite_events").stream()
    liked_event_refs = [doc.to_dict().get("id") for doc in liked_events_docs]
    liked_events = []
    for event_id in liked_event_refs:
        event_doc = db.collection("events").document(event_id).get()
        if event_doc.exists:
            liked_events.append(event_doc.to_dict())

    aggregated_profile = {}
    weight_liked = 0.7
    weight_profile = 0.3

    if liked_events:
        liked_fields = [extract_event_fields_for_vectorization(event) for event in liked_events]
        field_names = liked_fields[0].keys()

        liked_field_vectors = {}

        for field in field_names:
            if field == "main_categories":
                liked_field_vectors[field] = [category_vector(e[field]) for e in liked_fields]
            elif field == "genres":
                liked_field_vectors[field] = [genre_vector(e[field]) for e in liked_fields]
            else:
                texts = [e[field].lower() for e in liked_fields]
                vectors, _ = vectorize_field(texts)
                liked_field_vectors[field] = vectors

        for field in field_names:
            liked_vecs = np.array(liked_field_vectors[field])
            liked_mean = np.mean(liked_vecs, axis=0)

            # combined = (liked_mean + profile_vec) / 2
            if profile_components and field in profile_components:
                profile_vec = np.array(profile_components[field])
                combined = liked_mean * weight_liked + profile_vec * weight_profile
                aggregated_profile[field] = combined
            else:
                aggregated_profile[field] = liked_mean

    else:
        # Якщо немає улюблених подій, беремо профіль з анкети
        aggregated_profile = {k: np.array(v) for k, v in profile_components.items()}
        field_names = aggregated_profile.keys()

    # ----- events info
    events_docs = db.collection("events").stream() #TODO: smth more effective than stream()

```



```

events = []
for doc in events_docs:
    event = doc.to_dict()
    if "component_vectors" not in event:
        continue
    if not isinstance(event.get("endTime"), datetime) or event["endTime"] <
datetime.now(timezone.utc):
        continue
    events.append(event)

scored = []
for event in events:
    event_components = event.get("component_vectors", {})
    avg_score = score_event_by_components(aggregated_profile, event_components,
field_names)

    # --- ВПЛИВ ВІДСТАНІ
    venue = event.get("venue", {})
    lat = venue.get("latitude")
    lon = venue.get("longitude")

    if lat is not None and lon is not None:
        event_coords = (lat, lon)
    else:
        event_coords = None

    within_dist, dist = is_within_distance(user_location_coords, event_coords,
max_distance_km)

    if dist is not None:
        if within_dist:
            distance_score_multiplier = 1.0
        else:
            distance_score_multiplier = max(0.1, 1.0 - (dist - max_distance_km) / 100)
        avg_score *= distance_score_multiplier
    elif location_data:
        event_city = get_city_from_address(venue.get("address", ""))
        try:
            user_city = get_city_from_address(location_data.get("title", ""))
        except Exception:
            user_city = location_data
        if user_city and event_city and user_city.lower() != event_city.lower():
            avg_score *= 0.5

    # --- ВПЛИВ ЧАСУ
    if not is_event_time_suitable(event, {"time": preferred_times}):
        avg_score *= 0.7

    scored.append((avg_score, event))

scored.sort(reverse=True, key=lambda x: x[0])
return [e.get("id") for _, e in scored[:top_n]]

```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

**МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПЕРСОНАЛІЗОВАНИХ
РЕКОМЕНДАЦІЙ КУЛЬТУРНИХ ПОДІЙ**

Програма та методика тестування

КПІ.ІІ-1510.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олесь КОВТУНЕЦЬ

Нормоконтроль:

_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:

_____ Соня КОНДРАЦЬКА

Київ – 2025

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАНЬ	3
2 МЕТА ТЕСТУВАННЯ	4
3 МЕТОДИ ТЕСТУВАННЯ	5
4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ	6

1 ОБ'ЄКТ ВИПРОБУВАНЬ

Об'єктом випробувань є мобільний застосунок Festigo, розроблений для операційної системи iOS 17 і вище. Застосунок призначений для персоналізованих рекомендацій культурних подій на території України. Програмне забезпечення реалізовано з використанням мови програмування Swift (клієнтська частина) та Python (серверна частина), із використанням хмарної платформи Firebase для зберігання даних і системи автентифікації.

Тестування проводилося на реальному пристрої та симуляторі в Xcode, що відповідає вимогам до цільової платформи. Також у межах випробувань перевірялась взаємодія клієнтської частини з сервером, розгорнутим на платформі Render, та перевірка відповідності функціоналу заявленим вимогам.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження даних;
- перевірка сумісності додатку на емуляторах різних розмірів;
- знаходження проблем, помилок і недоліків з метою їх усунення;
- перевірка зручності та інтуїтивності графічного інтерфейсу.

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;
- динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Під час проведення тестування будуть використовуватись наступні допоміжні засоби:

- Postman – для ручного тестування REST API серверної частини застосунку;
- Xcode Simulator – для тестування iOS-застосунку на віртуальних пристроях із різними версіями iOS та розмірами екранів;
- Firebase Console – для моніторингу запитів, аутентифікації користувачів і перевірки коректності запису/читання даних.

Тестування програмного забезпечення виконувалося за наступним порядком:

- проведення динамічного тестування застосунку згідно з функціональними вимогами, зазначеними в технічному завданні;
- перевірка коректності відображення інтерфейсу на мобільних пристроях із різними розмірами екрана (iPhone SE, iPhone 13, iPhone 16 Pro Max тощо);
- тестування зручності використання інтерфейсу, включаючи навігацію, доступність елементів і логіку взаємодії;
- тестування роботи API через Postman – зокрема, перевірка обробки запитів, коректність відповідей, обробка помилок;
- верифікація функціональності з боку безпеки (авторизація, захист доступу до даних);
- проведення мануального тестування у цілісному вигляді (клієнтська частина + сервер) з типовими сценаріями користувача.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

**МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПЕРСОНАЛІЗОВАНИХ
РЕКОМЕНДАЦІЙ КУЛЬТУРНИХ ПОДІЙ**

Керівництво користувача

КПІ.ІІ-1510.045490.05.34

“ПОГОДЖЕНО”

_____ Олесь КОВТУНЕЦЬ

Нормоконтроль:

_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:

_____ Соня КОНДРАЦЬКА

Київ – 2025

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ	3
2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ	4
2.1 Системні вимоги для коректної роботи	4
2.2 Завантаження застосунку	4
2.3 Перевірка коректної роботи	4
3 ВИКОНАННЯ ПРОГРАМИ	5

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Festigo» – це мобільний застосунок для iOS, призначений для персоналізованих рекомендацій культурних подій. Основна мета застосунку — спростити користувачам пошук релевантних заходів з урахуванням їхніх інтересів, місцезнаходження, формату, а також зручного часу для відвідування.

Кінцева версія програмного забезпечення являє собою мобільний застосунок для iOS з локалізованим інтерфейсом українською мовою та підтримкою пристроїв на базі iOS 17 і новіших версій. Застосунок постачається у вигляді інсталяційного пакету, що може бути завантажений та встановлений користувачем через App Store після проходження модерації.

Застосунок використовує зовнішні API для імпорту актуальних подій та обробку рекомендацій на сервері. Тому для коректної роботи програми необхідне постійне підключення до Інтернету

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 17 або вище;
- процесор: Apple A15 Bionic;
- наявність доступу до Інтернету;
- для встановлення додатку на мобільному пристрої повинно бути не менше 200 МБ вільної пам'яті.

2.2 Завантаження застосунку

На даний момент застосунок можна встановити власноруч через середовище Xcode, Репозиторій проєкту доступний за адресою <https://github.com/kisellsn/FestiGo>. Для встановлення необхідно завантажити проєкт, відкрити його в середовищі Xcode, підключити мобільний пристрій на базі iOS та виконати запуск програми на фізичному пристрої. Після проходження перевірки Apple застосунок буде доступний для завантаження через App Store.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку. У разі, якщо дана іконка не з'явилась, то встановлення відбулось не успішно. Інакше користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись початкова сторінка застосунку.

3 ВИКОНАННЯ ПРОГРАМИ

При запуску мобільного застосунку буде відображено початкову сторінку з коротким описом програми, на якій користувач також має можливість увійти в систему або зареєструватися (рис. 3.1).

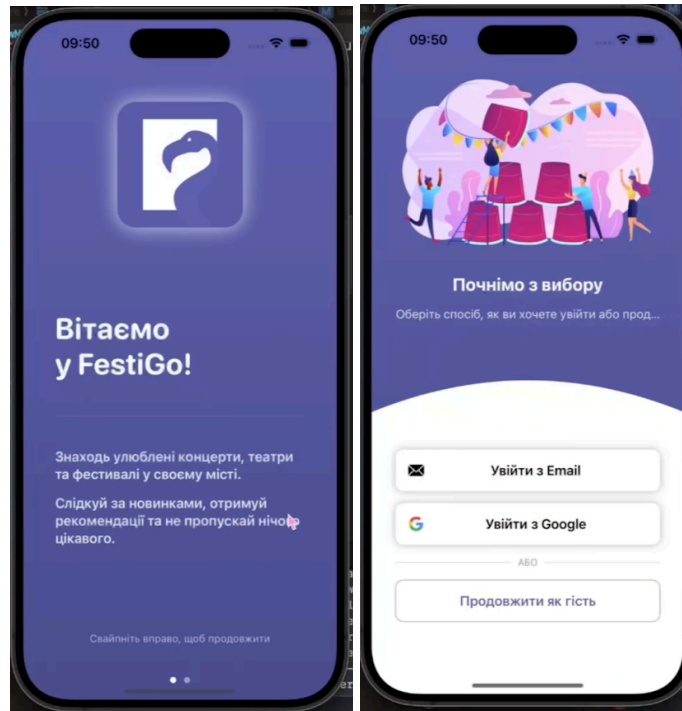


Рисунок 3.1 – Початкова сторінка застосунку

Для реєстрації передбачено два способи: введення електронної пошти з паролем (рис. 3.2) або вхід через Google-акаунт. Також є можливість зайти як гість, що надає обмежену функціональність, але підходить для швидкого пошуку подій поруч.

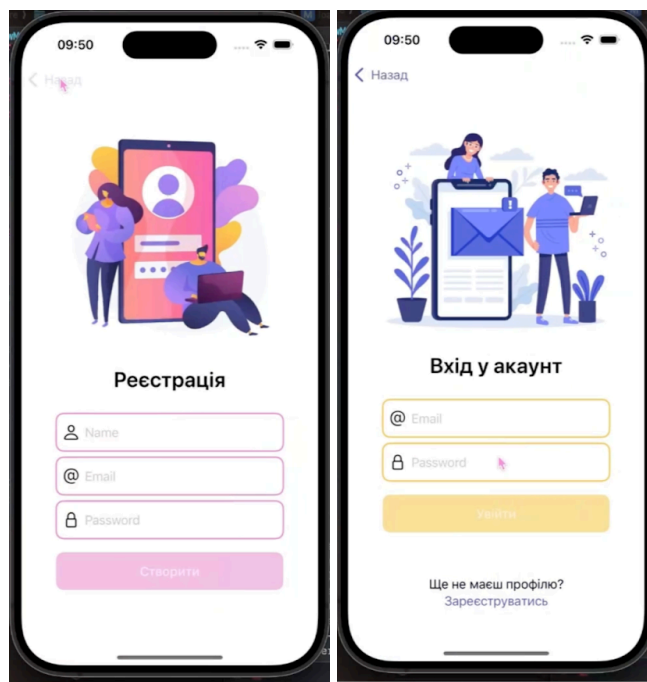


Рисунок 3.2 – Сторінки реєстрації та авторизації

Після авторизації, якщо це перший вхід, відкривається екран налаштування профілю інтересів. Тут користувач відповідає на кілька коротких запитань: обирає категорії культурних подій, які його цікавлять (наприклад, музика, театр, виставки), зазначає місто (автоматично визначається або обирається вручну) і задає зручний діапазон часу для відвідування заходів. Ці дані зберігаються у профілі користувача та використовуються для формування персоналізованих рекомендацій. На рисунку 3.3 наведено декілька таких питань.

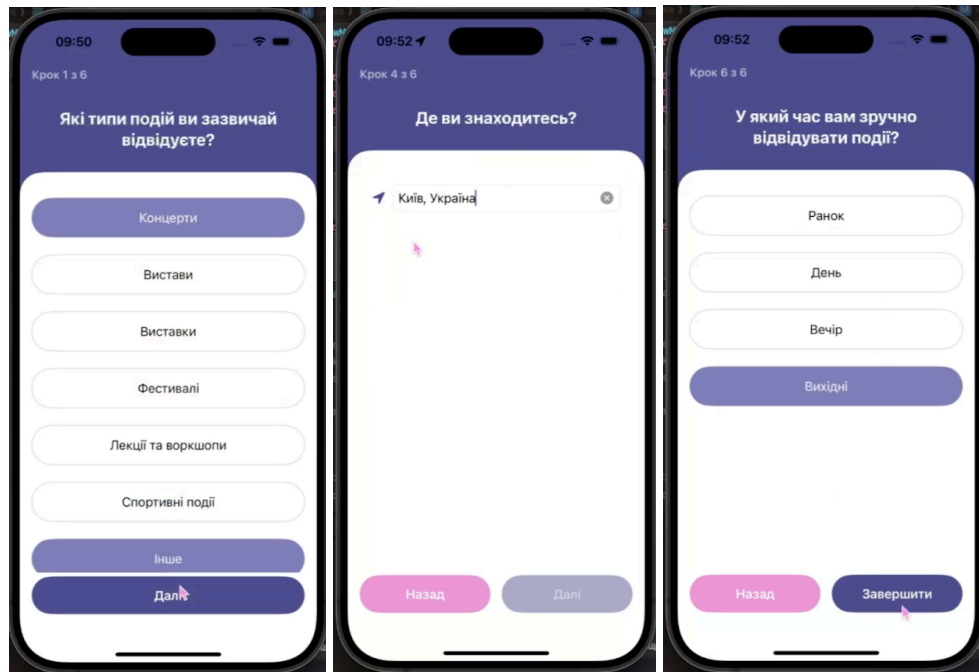


Рисунок 3.3 – Опитування для налаштування профілю інтересів

Далі користувач потрапляє на головну сторінку застосунку (рис. 3.4), яка містить два основні блоки: загальний список актуальних подій та персоналізовані рекомендації. Події можна фільтрувати за категоріями, містом, датою проведення та форматом (онлайн або офлайн).

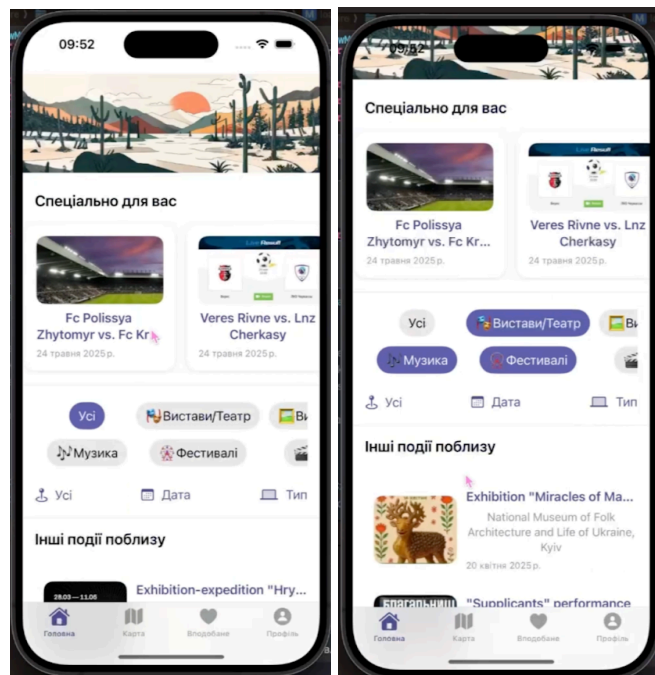


Рисунок 3.4 – Головна сторінка

Окремий розділ – це екран з інтерактивною мапою (рис. 3.5), на якій відображаються події з урахуванням геолокації користувача. Клік на позначку події дозволяє отримати її детальну інформацію.

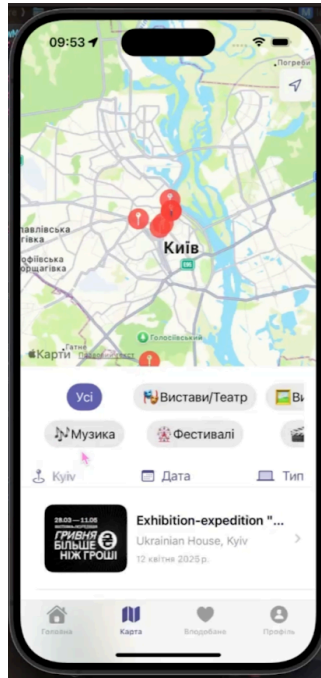


Рисунок 3.5 – Інтерактивна мапа з подіями поблизу

На екрані події (рис. 3.6) користувач бачить назву, опис, місце, дату, тип події та, якщо він авторизований, має змогу: додати подію в обране (натиснувши на іконку серця); поділитися подією через соцмережі чи месенджери; зберегти її у власному календарі пристрою; переглянути місце проведення на карті пристрою для навігації.

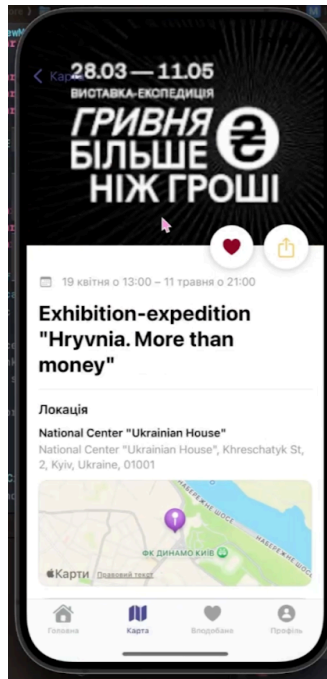


Рисунок 3.6 – Сторінка з деталями події

Користувач також може перейти до екрану вподобаних подій, де зберігаються всі додані в обране заходи, згруповані за датою проведення (рис. 3.7).

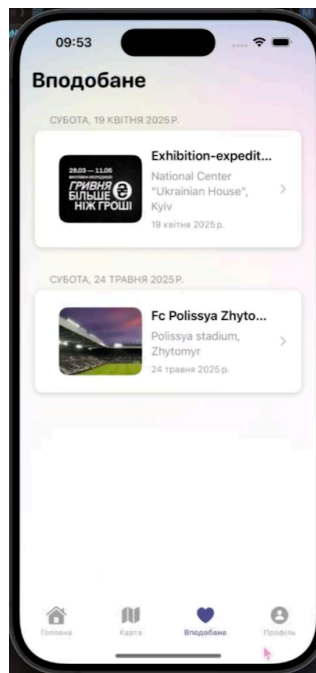


Рисунок 3.7 – Список вподобаних подій

Далі можна перейти на сторінку профілю користувача, де відображені його пошта та дата реєстрації (рис. 3.8). Тут є можливість виходу з акаунту, зміна мови додатку(українська та англійська) і перехід до налаштувань.

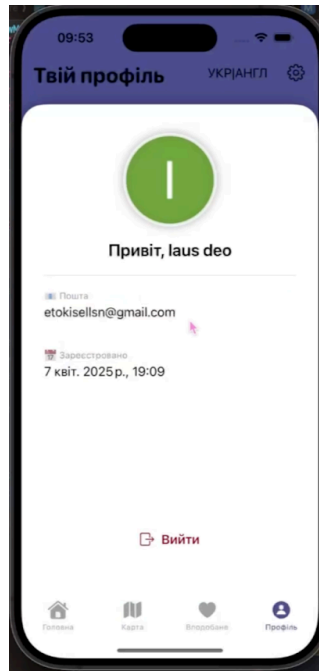


Рисунок 3.8 – Профіль користувача

У налаштуваннях профілю можна змінити локацію за щамовчуванням, бажану відстань та параметри сповіщень, а також змінити тему чи видалити свій акаунт назавжди (рис. 3.9).

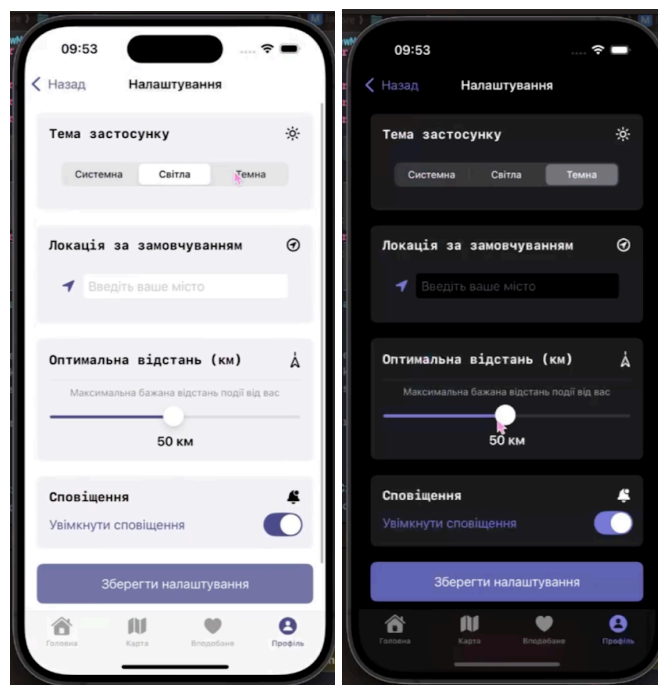


Рисунок 3.9 – Екран налаштувань

Завершення роботи із застосунком здійснюється через стандартну навігацію iOS – повернення на головний екран або закриття програми.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“ ____ ” _____ 2025 р.

**МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПЕРСОНАЛІЗОВАНИХ
РЕКОМЕНДАЦІЙ КУЛЬТУРНИХ ПОДІЙ**

Графічний матеріал

КПІ.ІІ-1510.045490.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олесь КОВТУНЕЦЬ

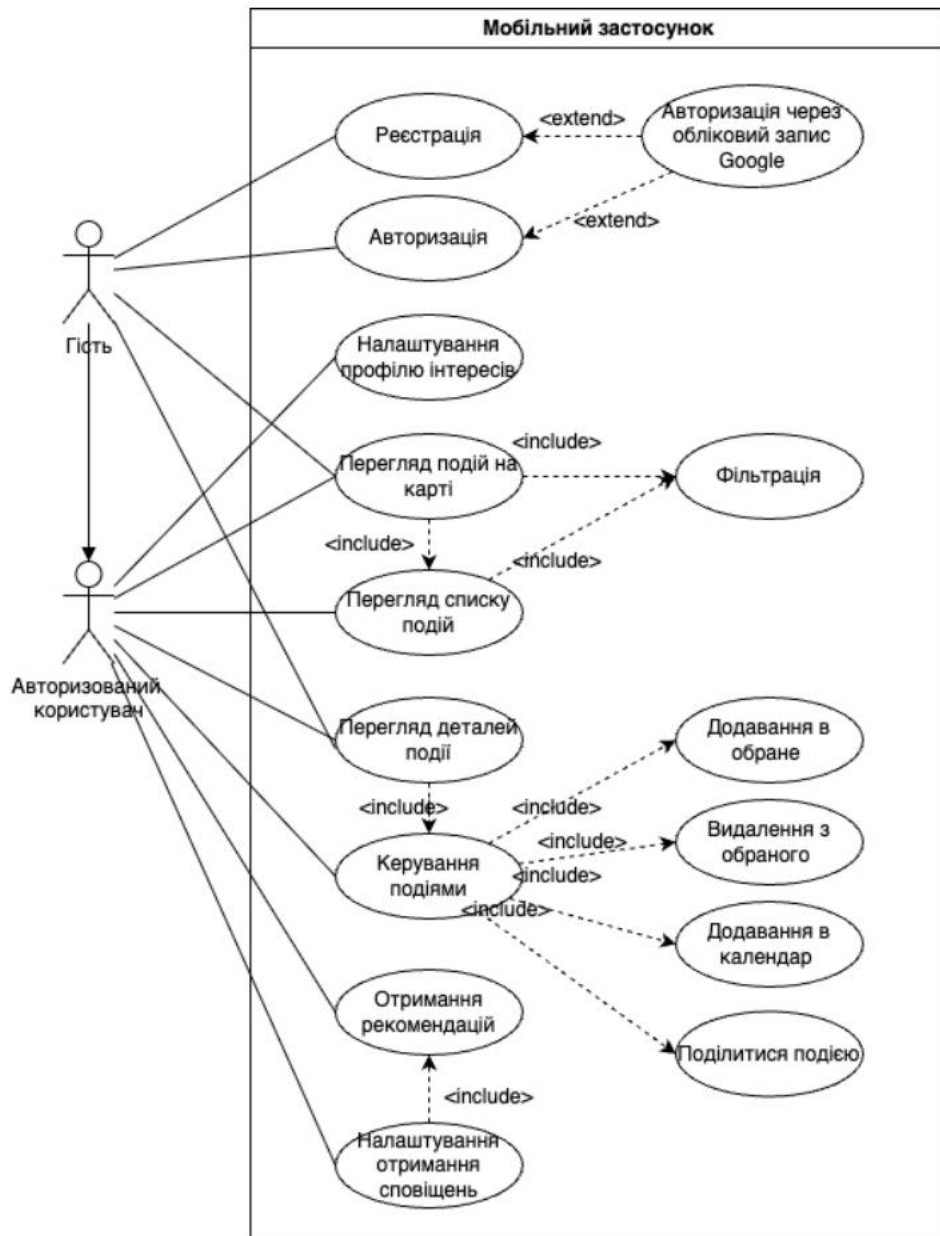
Нормоконтроль:

_____ Тетяна ШУЛЬКЕВИЧ

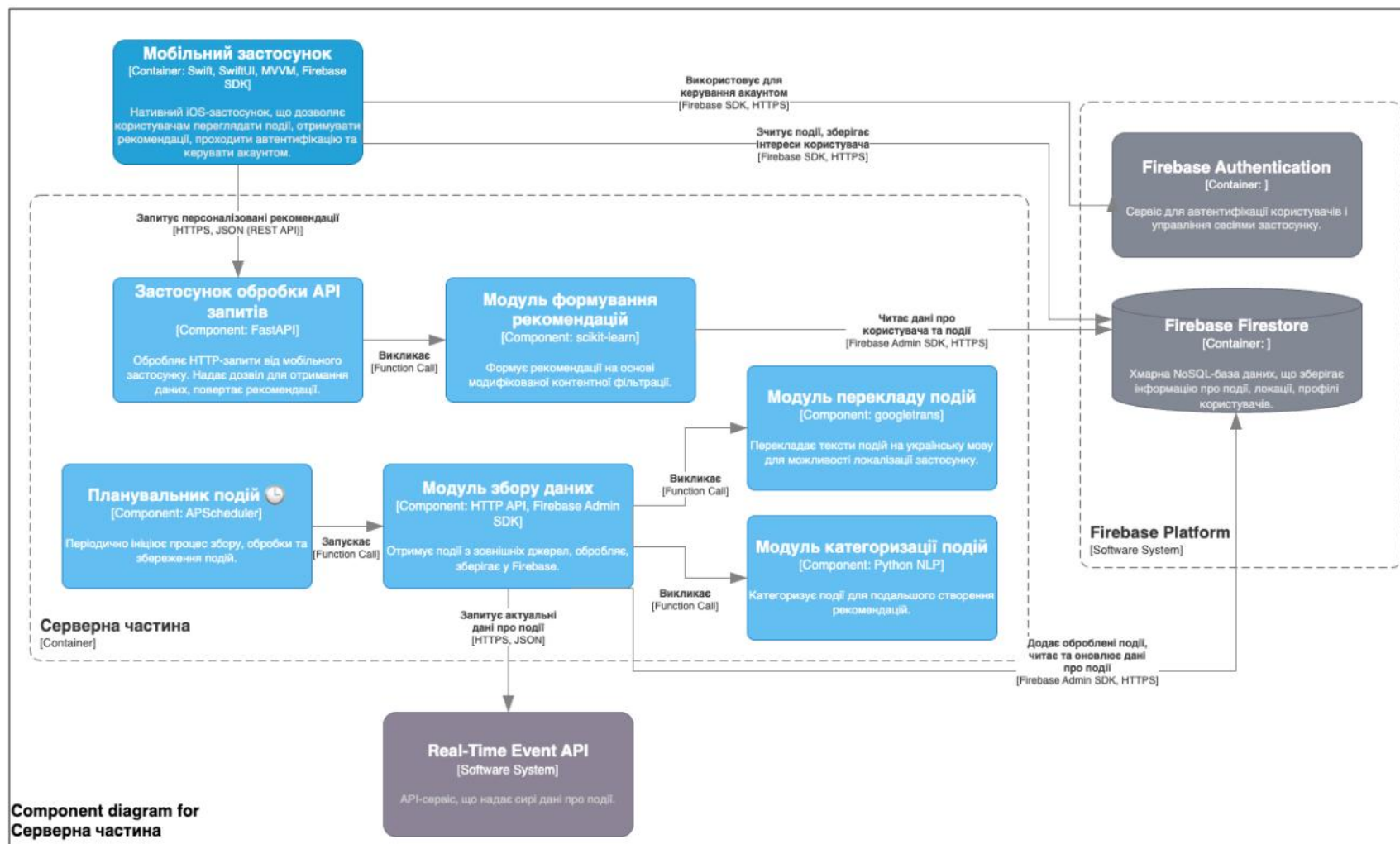
Виконавець:

_____ Соня КОНДРАЦЬКА

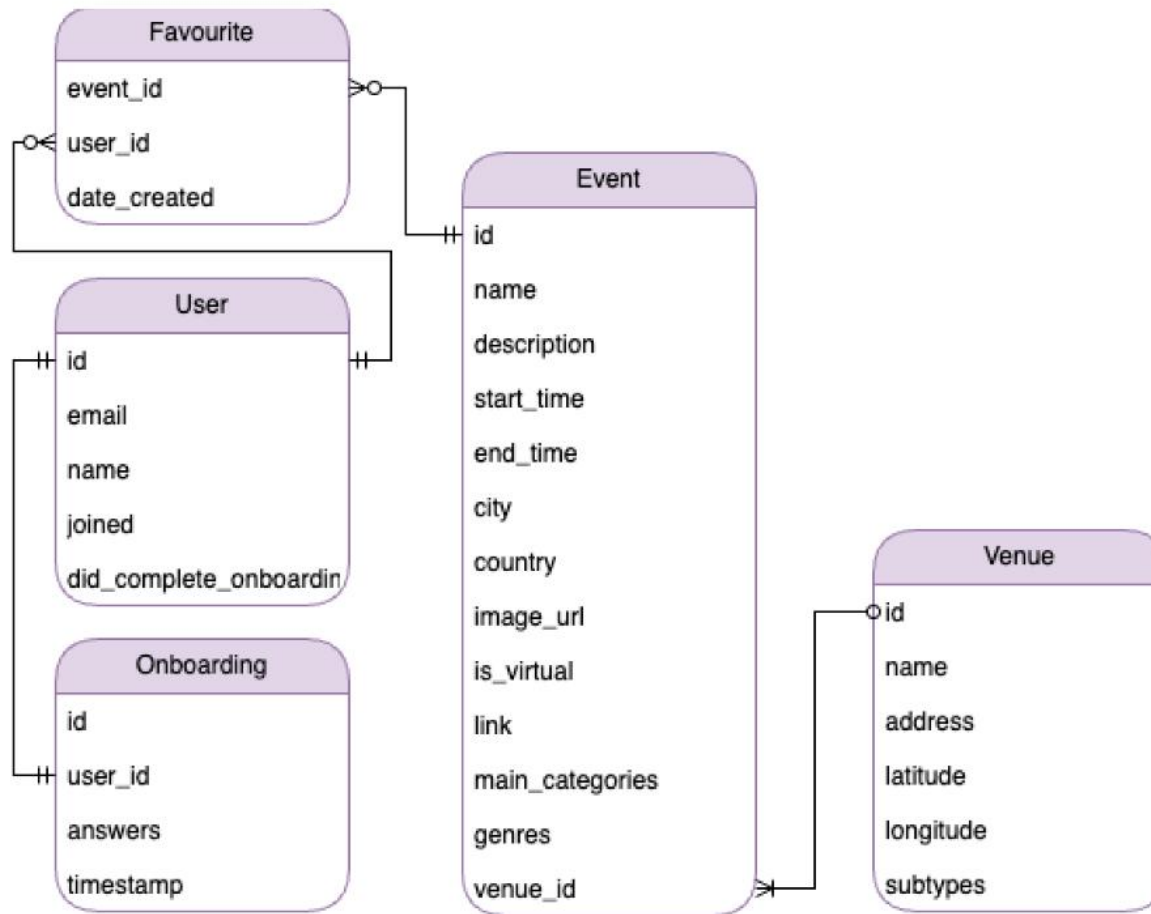
Київ – 2025



						КПІ.ІП-1510.045490.06.99.CCB					
						Схема структурна варіантів використань	Лит.		Маса	Масштаб	
Зм.	Арк.	№ докум.	Підп.	Дата							
Розробив	Кондрацька С.Л.										
Перевірів	Ковтунець О.В.										
Т. контр.											
							Аркуш 1		Аркушів 1		
Н. контр.	Шулькевич Т.В.					Мобільний застосунок для персоналізованих рекомендацій культурних подій	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-15				
Затвердив	Жаріков Е.В.										



						КПІ.ІП-1510.045490.06.99.ССМ			
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера		Маса	Масштаб
Розробив		Кондрацька С.Л.							
Перевірив		Ковтунець О.В.							
Т. контр.									
Н. контр.		Шулькевич Т.В.			Мобільний застосунок для персоналізованих рекомендацій культурних подій	Аркуш 1		Аркушів 1	
Затвердив		Жаріков Е.В.				КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-15			



						КПІ.ІП-1510.045490.06.99.СБД				
						Схема бази даних	Літера		Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата						
Розробив		Кондрацька С.Л.								
Перевірів		Ковтунець О.В.								
Т. контр.							Аркуш 1		Аркушів 1	
Н. контр.		Шулькевич Т.В.				Мобільний застосунок для персоналізованих рекомендацій культурних подій	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-15			
Затвердив		Жаріков Е.В.								