НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра телекомунікацій

До захисту допущено:

Завідувач кафедри

_____ Сергій КРАВЧУК

«___»_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія та програмування інфокомунікацій»

спеціальності 172 «Телекомунікації та радіотехніка»

на тему: «Розробка мобільного застосунку для оптимізації пошуку та публікації заходів з використанням Firebase»

Виконав:
студент IV курсу, групи ТЗ-11
Хрищенюк Максим Юрійович _____

Керівник:
Доцент кафедри ТК НН ІТС, с.н.с., к.т.н.
Міночкін Дмитро Анатолійович _____

Рецензент:
Доцент кафедри ІТТ НН ІТС, к.т.н., доцент
Суліма Світлана Валеріївна

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут телекомунікаційних систем
Кафедра телекомунікацій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інженерія та програмування інфокомунікацій»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_______ Сергій КРАВЧУК
«___»_____2025 р.

ЗАВДАННЯ

на дипломну роботу студенту

Хрищенюку Максиму Юрійовичу

1. Тема роботи «Розробка мобільного застосунку для оптимізації пошуку та публікації заходів з використанням Firebase», керівник роботи Міночкін Дмитро Анатолійович, с.н.с., к.т.н., затверджені наказом по університету від «26» травня 2025 р. № 1755-с.

2. Термін подання студентом роботи 9 червня 2025 р.

3. Вихідні дані до роботи: Бази даних Firebase; запити та взаємодія з базами даних; мова C#; середовище розробки Unity; середовище створення користувацького інтерфейсу Figma; механіки взаємодій елементів в застосунку.

4. Зміст роботи: Проаналізувати стан сучасного розвитку мобільної розробки та оглянути основні мобільні операційні системи, для визначення платформи розробки. Проаналізувати існуючі рішення з пошуку та публікації заходів та виявити їх недоліки та переваги. Описати власне рішення проблеми. Обрати технології розробки, в залежності від потреб та обмежень при створені дипломного проєкту. Реалізація окресленого рішення проблеми у вигляді мобільного застосунку.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

Слайд №1 Назва роботи

Слайд №2 План презентації

Слайд №3 Актуальність роботи

Слайд №4 Мета

Слайд №5 Поставлені задачі та хід виконання: Аналіз аналогів

Слайд №6 Поставлені задачі та хід виконання: Архітектура

Слайд №7 Поставлені задачі та хід виконання: Дизайн

Слайд №8 Поставлені задачі та хід виконання: Розробка

Слайд №9 Поставлені задачі та хід виконання: Тестування та оптимізація

Слайд №10 Результати роботи

Слайд №11 Висновок

6. Дата видачі завдання 22.10.2024р.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Підготовка матеріалів для вивчення теми та розробки мобільного застосунку	22.10.2024 - 28.10.2024	Виконано
2	Аналіз тенденцій мобільної розробки та актуальність платформ	29.10.2024 - 05.11.2024	Виконано
3	Вивчення проблеми пошуку та публікації заходів	06.11.2024 - 22.11.2024	Виконано
4	Аналіз та визначення технологій для розробки застосунку	23.11.2024 - 08.12.2024	Виконано
5	Проєктування архітектури застосунку	09.12.2024 - 04.01.2025	Виконано
6	Реалізація клієнтської та серверної частин застосунку	05.01.2025 - 13.05.2025	Виконано
7	Розробка інтерфейсу користувача	16.01.2025 - 18.04.2025	Виконано

8	Тестування та оптимізація застосунку	05.01.2025 - 19.05.2025	Виконано
9	Підготовка та оформлення матеріалів дипломної роботи	20.05.2025 - 09.06.2025	Виконано

Студент Максим ХРИЩЕНЮК

Керівник Дмитро МІНОЧКІН

РЕФЕРАТ

Дипломна робота викладена на 75 сторінках та включає 25 ілюстрацій, 29 джерел та 1 додаток.

Мета роботи: створення мобільного застосунку з інтеграцією баз даних, що забезпечить зручний та ефективний спосіб пошуку, фільтрації та публікації заходів з орієнтиром на українську молодь та локальних авторів.

Застосунок розроблений для мобільних пристроїв під операційну систему Android, тому рекомендується до публікації на основну платформу для поширення та просування застосунків та ігор Play Market, що допоможе у створенні та розширенні аудиторії, зацікавленої у сфері пошуку, публікації та просування заходів.

За результатами розробки, реалізоване рішення показало, як можна вирішити проблему незручного пошуку та публікації подій, за допомогою використання методів розробки та інтегрованих хмарних функцій.

Ключові слова: застосунок, пошук, публікація, створення, фільтрація, авторизація, запит, Firebase, Unity, Figma, Firestore, Realtime, Storage, Authentication.

ABSTRACT

The thesis is presented on 75 pages and includes 25 illustrations, 29 sources and 1 appendix.

Purpose of the work: creation of a mobile application with database integration, which will provide a convenient and effective way to search, filter and publish events with a focus on Ukrainian youth and local authors.

The application is developed for mobile devices under the Android operating system, therefore it is recommended for publication on the main platform for distributing and promoting applications and games Play Market, which will help in creating and expanding the audience interested in the field of search, publication and promotion of events.

According to the development results, the implemented solution showed how to solve the problem of inconvenient search and publication of events using development methods and integrated cloud functions.

Keywords: application, search, publication, creation, filtering, authorization, query, Firebase, Unity, Figma, Firestore, Realtime, Storage, Authentication.

ЗМІСТ

ВСТУП 11
РОЗДІЛ 1 13
СУЧАСНИЙ СТАН МОБІЛЬНОЇ РОЗРОБКИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 13
1.1 Аналіз сучасного стану розвитку мобільної розробки 13
1.1.1 Роль мобільних застосунків у повсякденному житті 13
1.1.2 Тенденції розвитку мобільних технологій 15
1.2 Огляд основних мобільних платформ 18
1.2.1 Android: особливості та популярність 18
1.2.2 iOS: переваги та обмеження 19
1.2.3 Інші платформи 21
1.3 Огляд застосунків-альтернатив для пошуку та публікації подій 22
1.3.1 Meetup 22
1.3.2 Eventbrite 23
1.3.3 Facebook Events 24
1.3.4 AllEvents 25
1.4 Виявлені обмеження альтернатив 26
1.4.1 Фрагментація інформації про події 26
1.4.2 Недостатня гнучкість або відсутність фільтрації 27
1.4.3 Обмеження для створення подій 27
1.4.4 Відсутність локалізації для української аудиторії 28
1.5 Пропоноване рішення 29
1.5.1 Централізована платформа для пошуку та публікації 29
1.5.2 Система фільтрації подій 29
1.5.3 Спрощений механізм створення подій 30
1.5.4 Фокус на українську аудиторію 31
Висновок до розділу 31
РОЗДІЛ 2 33
ПОРІВНЯННЯ ТА ВИБІР ТЕХНОЛОГІЙ 33

2.1 Критерії вибору технологій 33
2.1.1 Простота використання 33
2.1.2 Масштабованість 34
2.1.3 Економічна ефективність 34
2.1.4 Інтеграція між технологіями 35
2.2 Використані технології 36
2.2.1 Unity як середовище розробки 36
2.2.2 C# для логіки застосунку 38
2.2.3 Firebase для хмарних сервісів 39
2.2.4 Figma для дизайну інтерфейсу 40
2.2.5 GitHub Desktop для контролю версій 43
2.3 Альтернативні технології та обґрунтування вибору технологій 43
2.3.1 Альтернативи Unity 43
2.3.2 Альтернативи Firebase 44
2.3.3 Альтернативи Figma 45
2.3.4 Альтернативи GitHub Desktop 46
Висновок до розділу 47
РОЗДІЛ 3 49
ПРАКТИЧНА РЕАЛІЗАЦІЯ 49
3.1 Архітектура мобільного застосунку 49
3.1.1 Загальна структура 49
3.1.2 Модель: обробка даних із Firebase 51
3.1.3 Вигляд: інтерфейс користувача в Unity 55
3.1.4 Логіка взаємодії компонентів в Unity 57
3.2 Основні механіки застосунку 58
3.2.1 Пошук подій: перегляд і фільтрація 58
3.2.2 Створення подій: введення даних і завантаження медіа 60
3.2.3 Аутентифікація 63
3.2.4 Видалення застарілих подій 65
3.3 Оптимізація використання бази даних 66

3.3.1 Сортування за ID для ефективного пошуку 66
3.3.2 Зменшення кількості запитів до Firestore 67
3.3.3 Ефективне зберігання медіа в Storage 68
Висновок до розділу 69
ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ 71
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72
ДОДАТКИ 75
Додаток А 75

ПЕРЕЛІК СКОРОЧЕНЬ

ШІ	Штучний інтелект
IoT	Internet of Things (Інтернет речей)
AWS	Amazon Web Services
API	Application Programming Interface (Інтерфейс програмування застосунків)
UI	User Interface (Користувацький інтерфейс)
SDK	Software Development Kit (Набір засобів для розробки програмного забезпечення)
SVG	Scalable Vector Graphics (Масштабована векторна графіка)
YMDHMS	Year, Month, Day, Hour, Minute, Second (Рік, Місяць, День, Година, Хвилина, Секунда)
C#	Мова програмування (вимовляється "Сі-шарп")

ВСТУП

Цифрове середовище складно уявити без мобільних застосунків, які безумовно відіграють важливу роль у спрощенні повсякденних завдань., зокрема це стосується сфер пошуку, організації та управління заходами. Сучасні технології дозволяють користувачам швидко знаходити інформацію про події, реєструватися на них та взаємодіяти з організаторами, але існуючі рішення на ринку можуть не задовольняти потреби користувачів, особливо якщо йде мова про українську частку, де не так просто знайти щось цікаве в своєму місті.

Meetup, Eventbrite, Facebook Events, AllEvents – проаналізовані платформи, результат яких показав, що всі вони мають певні недоліки, тим самим ускладнюючи взаємодію з сервісом. Розрізненість платформ змушує користувачів шукати заходи у різних джерелах, починаючи від подібних платформ, для пошуку заходів, закінчуючи сторінками організаторів конкретних подій. Такий процес ускладнює швидкий доступ до актуальної інформації. Не всі сервіси мають гнучку систему фільтрації, що не дозволяє ефективно шукати події за категоріями, датою або іншими параметрами. Багато платформ обмежують можливість публікації заходів, вимагаючи узгодження публікації з адміністрацією платформи або передбачаючи платний доступ до розширених функцій. Найкритичніший знайдений недолік, являє тобою те, що серед платформ не було знайдено тих, котрі у вигляді мобільного застосунку мають цільову аудиторію українців / українську молодь.

Запропонований мобільний застосунок з пошуку і публікації подій має на меті усунути ці проблеми, об’єднавши в одному сервісі всі необхідні інструменти для пошуку, фільтрації та створення заходів, котрі будуть доступні для кожного. Реалізація на основі Firebase забезпечує швидку обробку даних, надійне збереження інформації у базах даних від Google, зручну систему авторизації.

При використанні сучасних технологій, можливо:

- Створити єдину платформу для пошуку та публікації заходів.
- Реалізувати систему пошуку та фільтрації за категоріями, місцем проведення, датою та іншими параметрами.
- Запровадити простий механізм публікації подій, не вимагаючи складних налаштувань для користувача або погоджень з адміністрацією платформи.
- Забезпечити безпеку даних, при використанні хмарних збережень на серверах Google.
- Оптимізувати взаємодію між відвідувачами та організаторами заходів, тим самим надаючи можливість легко створювати заходи та переглядати інформацію в зручному форматі.

Описаний застосунок вирішує ключові проблеми, з якими стикаються відвідувачі і організатори заходів, створюючи зручний, гнучкий та ефективний інструмент для взаємодії між користувачами.

Підсумовуючи, метою розробки мобільного застосунку, в основі котрого лежить взаємодія з базами даних, є створення платформи, котра ціле направлена на пошук, фільтрацію та створення подій в зручному форматі для користувачів – хто шукає події і хто їх створює.

РОЗДІЛ 1

СУЧАСНИЙ СТАН МОБІЛЬНОЇ РОЗРОБКИ ТА ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Аналіз сучасного стану розвитку мобільної розробки

1.1.1 Роль мобільних застосунків у повсякденному житті

Завдяки широкому використанню застосунків в різних сферах, включаючи комунікації, освіту, фінанси, розваги та планування заходів, вони стали для смартфонів стали ключовим аспектом цифрового світу.

Застосунки дозволяють людям залишатися залученими до місцевої громади, ефективно розпоряджатися своїм часом та брати участь у культурних та освітніх заходах. Вони допомагають активізувати громадське життя в Україні, де смартфони є основним інструментом, що використовується для більшості повсякденних завдань.

Sensor Tower пише, що хоч у 2024 році у світі було завантажено 136 мільярдів мобільних застосунків – що в свою чергу на 1% менше, ніж у 2023 році, але попри це, витрати користувачів зросли на 12,5% аж до рекордних 150 мільярдів доларів. Час використання застосунків досяг 4,2 трильйона годин [3].

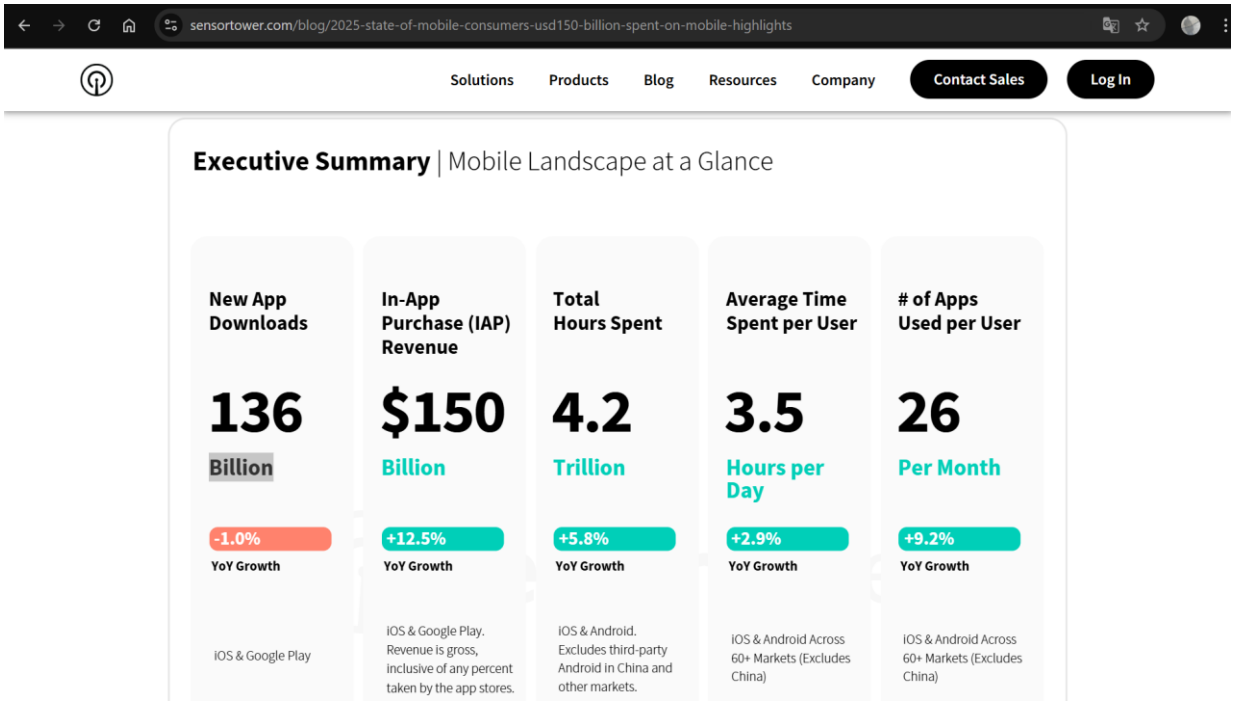

Рис. 1.1 Глобальні дані по мобільним застосункам.

За даними Statista, у 2024 році 36,2 мільйона людей в Україні користуватимуться смартфонами, а до 2029 року очікується, що ця кількість зросте до 36,8 мільйона [5]. Серед молоді, яка активно використовує смартфони для виконання щоденних завдань, це свідчить про зростаючу залежність від мобільних технологій та застосунків у повсякденному житті.

Мобільні застосунки, такі як Meetup, Eventbrite та Facebook Events, можуть значно спростити пошук, реєстрацію та координацію для планування заходів. Це стосується як студентських конференцій, лекцій, так й культурних фестивалів чи спортивних змагань.

Eventbrite стверджує, що особливо серед організаторів заходів 78% вважають мобільні застосунки важливими для залучення своєї аудиторії; 52% учасників отримують доступ до інформації про події за допомогою смартфонів [4]. Статистика вказує на мобільні застосунки як основний інструмент для планування місцевих заходів, особливо для студентів та молоді.

1.1.2 Тенденції розвитку мобільних технологій

Більшість сучасних мобільних застосунків зараз створюються на хмарних платформах, включаючи Amazon Web Services (AWS), Google Firebase, Microsoft Azure та Google Cloud Platform. Таке рішення на основі хмарних сервісів пропонує масштабованість, гнучкість та високу продуктивність. Вони дозволяють розробникам використовувати найновіші технології, швидко адаптувати застосунки до зростаючих потреб користувачів при цьому зменшуючи витрати на інфраструктуру.

Високопродуктивні мережі та 5G: впровадження мереж 5G забезпечує наднизьку затримку та високу пропускну здатність, покращуючи можливості хмарних сервісів у мобільних застосунках. Такі мережі дозволяють обробляти обсяги даних у режимі реального часу, що є досить суттєвим для сучасних мобільних застосунків. Зокрема, це критично важливо для мобільних застосунків, що потребують доповненої реальності, оскільки їм потрібна швидка та точна інтеграція віртуальних об'єктів з реальним середовищем; для навігаційних застосунків, які обробляють дані з датчиків у режимі реального часу; для потокових платформ, де якість перегляду та задоволення користувачів залежать від стабільної передачі відео та аудіо без затримки.

Інтеграція IoT дозволяє розробляти контент-залежні мобільні застосунки, які взаємодіють з підключеними пристроями для покращення взаємодії з користувачем. Такі застосунки, наприклад, можуть надсилати розумні сповіщення під час подій або автоматично сканувати квитки за допомогою сумісних пристроїв.

Штучний інтелект та персоналізація: Технології ШІ зараз фундаментально змінюють взаємодію з користувачем, надаючи відповіді на буденні завдання, запитання за лічені секунди. TekRevol стверджує, що у 2024 році кількість застосувань штучного інтелекту, таких як алгоритми рекомендацій або чат-боти,

зросла на 35% [6]. У застосунках для проведення заходів штучний інтелект може вивчати, аналізувати вподобання та звички користувачів та на основі цих даних рекомендувати відповідні події.

Використовуючи єдину кодову базу, можна створювати застосунки для iOS та Android за допомогою таких інструментів, як Unity. Згідно з даними Biz4Solutions, у 2024 році 87% розробників створювали мобільні застосунки саме за допомогою крос платформних фреймворків [7]. Головною причиною на те стала потреба в підтримці кількох пристроїв та швидшому виході на ринок. Зокрема, Unity дозволяє швидко переносити застосунки з однієї платформи на іншу та створювати адаптивні інтерфейси за короткий проміжок часу.

Монетизація та утримання користувачів: За даними ASOMobile, ринок мобільних застосунків зараз рухається в напрямку утримання аудиторії [8]. Подібно до Spotify Premium, який забезпечує покращений досвід порівняно з безкоштовною версією, застосунки частіше використовували підписки та преміум-функції у 2024 році, і є підстави вважати, що ця тенденція тільки зросте у 2025. Приклад різниці функціоналу Spotify Free та Spotify Premium можна побачити на рисунку 1.2.

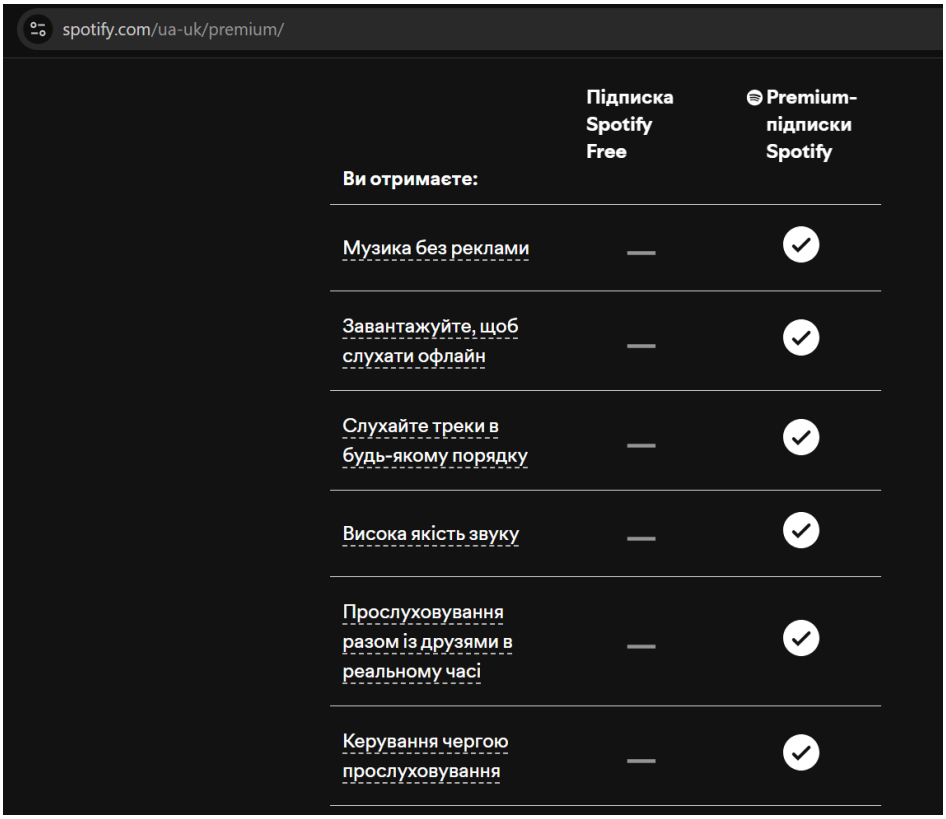

Рис. 1.2 Різниця між безкоштовною версією Spotify та Spotify Premium.

Такий метод монетизації вже перевірений часом і люди готові платити за додаткові зручності при використанні сервісів. Цей тип монетизації також може стосуватися платформ для проведення заходів та включати покращений пошук для користувачів, наприклад використання того ж ШІ, просування подій організаторам та надання розширених даних аналітики.

Безпека даних: Зі зростанням кіберзагроз безпека стає пріоритетною. За даними Cybersecurity Ventures, глобальні збитки від кіберзлочинності становили 9,5 трильйона доларів у 2024 році та, як очікується, досягнуть 10,5 трильйона доларів у 2025 році [28]. Оскільки мобільні застосунки використовуються так широко, вони є однією з основних цілей атак. Надійні способи зберігання даних на своїх серверах забезпечуються хмарними сервісами, такими як Firebase.

1.2 Огляд основних мобільних платформ

1.2.1 Android: особливості та популярність

Всім відомий Android, розроблений Google, є найпоширенішою мобільною операційною системою у світі. За даними TekRevol, Google Play Store у 2025 році налічує 2,87 мільйона застосунків, що робить його найбільшим ринком застосунків [6]. Частка Android на глобальному ринку становить приблизно 71%, завдяки широкому спектру пристроїв від різних виробників (Samsung, Xiaomi, Oppo) і доступності в різних цінових категоріях [29].

Android можна описати наступним чином, як зображно на рисунку 1.3:

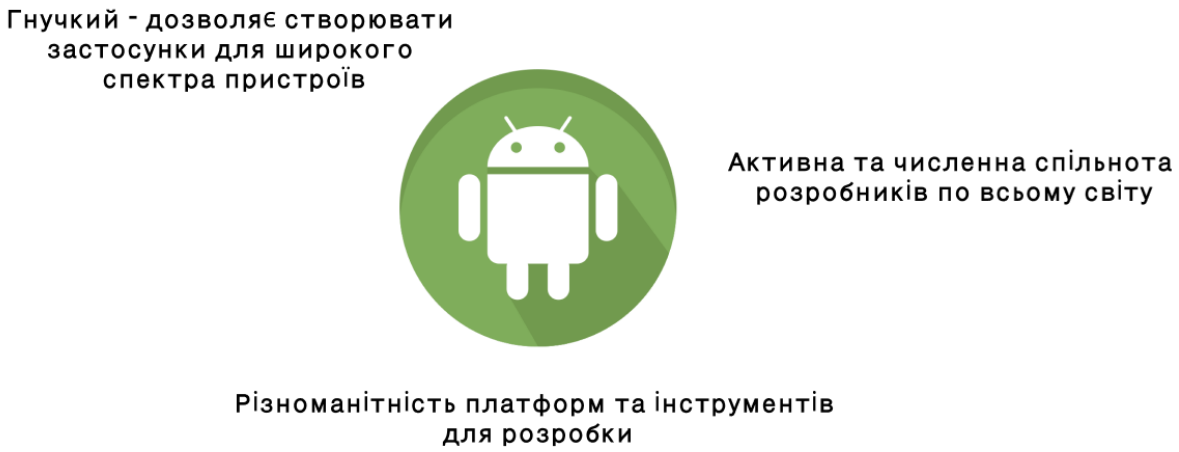

Рис. 1.3 Короткий опис Android.

Android відзначається своєю високою гнучкістю, що дозволяє створювати застосунки для широкого спектра пристроїв — від смартфонів, планшетів до телевізорів, розумних годинників і навіть автомобільних систем. Android також добре інтегрується з хмарними сервісами, зокрема такими як Firebase, надаючи розробникам доступ до потужних інструментів аналітики, баз даних у реальному часі, аутентифікації користувачів, хмарного зберігання та push-сповіщень.

Платформа також підтримує різноманітні середовища для розробки, серед яких і Unity — потужне середовище для створення не лише ігор, а й

інтерактивних та візуально насичених мобільних застосунків, що потребують 2D або 3D графіки. Крім того, активна спільнота розробників, велика кількість документації та навчальних матеріалів сприяють швидкому освоєнню технологій та ефективному розв’язанню технічних задач.

Одна з найбільших переваг Android — це активна та численна спільнота розробників по всьому світу. Завдяки цьому програмісти мають легкий доступ до великої кількості навчальних матеріалів, відео уроків, відкритого коду та технічної документації. Форуми, блоги та платформи дозволяють швидко знаходити рішення для типових проблем, обмінюватися досвідом та отримувати підтримку, таким прикладом може слугувати форум на кшталт Stack Overflow.

Попри гнучкість і широкі можливості, Android має низку недоліків, які можуть ускладнювати розробку та використання застосунків. Однією з основних проблем є фрагментація. Існує велика кількість пристроїв із різними версіями операційної системи, а також з різними поверхневими оболонками, на кшталт MIUI для пристроїв Xiaomi. Пристрої відповідно різняться розмірами екранів та технічними характеристиками. Тестування та оптимізація застосунків в таких умовах сильно ускладняється. Власні модифікації системи також можуть впливати на стабільність роботи програм. До недоліків також можна записати й безпеку — Android більш вразливий до шкідливого програмного забезпечення, особливо на пристроях без офіційної підтримки або з root-доступом.

1.2.2 iOS: переваги та обмеження

З часткою ринку ~28% операційна система iOS від Apple є другою за величиною мобільною платформою у світі [29]. За даними TekRevol, очікується, що до 2025 року в Apple App Store буде 2,06 мільйона програм, що на 500 тисяч більше, ніж у 2024 році, що свідчить про постійне розширення екосистеми [6]. Висока стабільність, надійна система безпеки та широка оптимізація апаратного забезпечення – це характеристики iOS, що гарантують швидку роботу програм

та приємний користувацький досвід. Короткий опис операційної системи можна побачити на рисунку 1.4.

Рис. 1.4 Короткий опис iOS.

Висока продуктивність є однією з головних переваг iOS. Тут зіграла роль тісної інтеграції потужного апаратного та програмного забезпечення, що дозволяє навіть менш потужним пристроям добре функціонувати. Ще однією сильною стороною є безпека: ізольована система виконання та шифрування значно знижує ймовірність витоку даних. Крім того, невелика кількість моделей пристроїв Apple спрощує тестування та розробку, що допомагає створювати високоякісні застосунки. Але є труднощі з розробкою для iOS. Вартість може бути стримуючим фактором для невеликих команд, оскільки це вимагає використання обладнання Apple та вимагає платної ліцензії App Store. Суворі стандарти якості та дизайну застосунків ускладнюють публікацію, а деякі проекти можуть бути затримані через відсутність можливостей налаштування порівняно з Android.

Розроблений застосунок наразі орієнтований на Android, але в майбутньому планується його адаптація для iOS. Такий крок є стратегічно виправданим, адже розширення на платформу Apple дозволить охопити додаткову аудиторію, зокрема користувачів із високою купівельною спроможністю. Адаптація для iOS, попри певні складнощі, відкриє нові можливості для підвищення конкурентоспроможності продукту на ринку.

1.2.3 Інші платформи

Окрім Android та iOS, існують альтернативні платформи, такі як HarmonyOS від Huawei. За даними Huawei, у 2024 році HarmonyOS встановлена на 900 мільйонах пристроїв [10], але її глобальна частка залишається дуже не високою і не враховується у глобальних рейтингах, ймовірно через санкції та орієнтир на локальний ринок Китаю. На рисунку 1.5 можна побачити статистику популярності платформ.

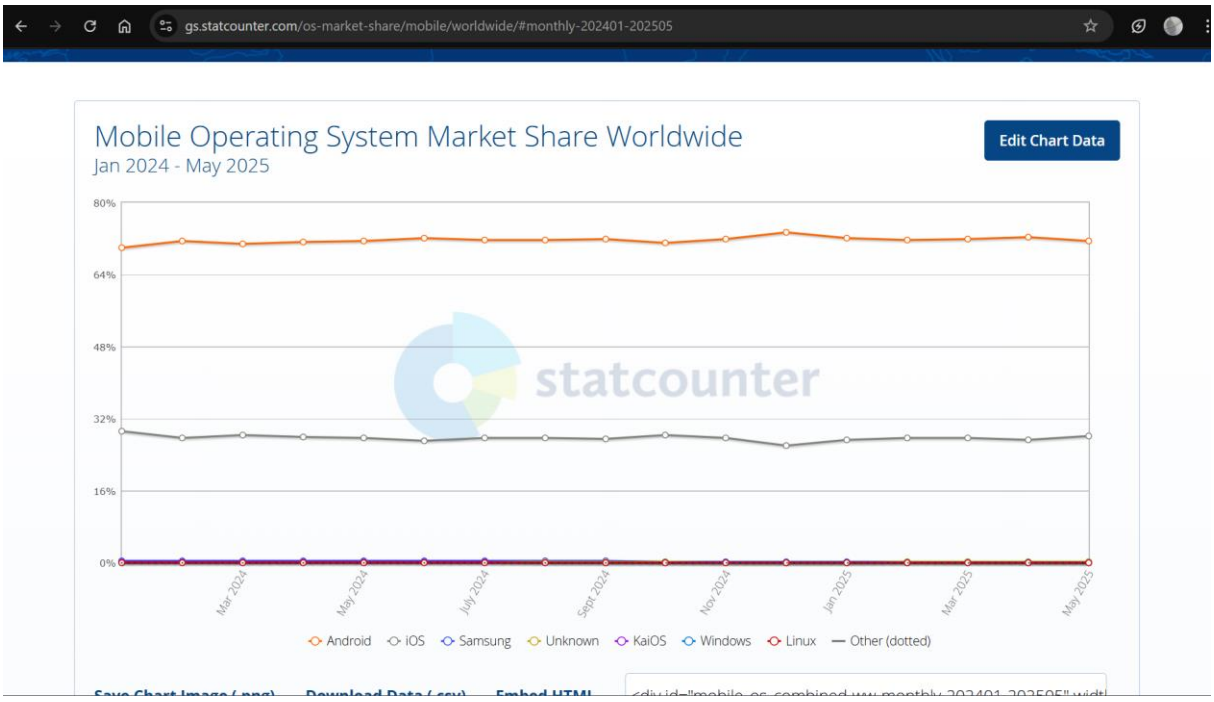

Рис. 1.5 Дані про популярність платформ в період з січня 2024 та по травень 2025 (без врахування HarmonyOS).

Інші платформи, як-от Windows Mobile, втратили актуальність через слабку екосистему та припинення підтримки у 2019 році, коли її частка впала з піку 3,7% у 2013 році до нуля. KaiOS, котра встановлена на 150 мільйонах пристроїв, підходить лише для базових функцій.

Android та iOS залишаються основними платформами для розробки завдяки розвиненим екосистемам і широкій підтримці.

1.3 Огляд застосунків-альтернатив для пошуку та публікації подій

1.3.1 Meetup

Meetup – мобільний застосунок і сайт, який орієнтований на створення спільнот за інтересами, дозволяючи користувачам приєднуватися до груп і отримувати сповіщення про локальні заходи. Платформа зараз налічує 60 мільйонів користувачів у 190 країнах і підтримує як онлайн, так і офлайн-події [11].

Переваги:

- Фокус на локальних спільнотах.
- Можливість регулярних заходів.
- Інтеграція з соціальними мережами.

Недоліки:

- Платний доступ для організаторів
- Обмежена підтримка локальних авторів
- Відсутність орієнтування на український ринок

За своєю суттю дуже схожий на розроблене рішення в ході дипломної роботи, але в цей же час головними недоліками вважається відсутність

орієнтування на український ринок та платне створення подій, котрі обмежують створення подій для організаторів. Українському користувачеві достатньо складно на даній платформі знайти щось зі свого міста, особливо якщо це стосується офлайн заходів. Українськи і іноземним організаторам складно освітлювати події, оскільки вони мають сплачувати платну підписку розміром $14.99-$30/міс. Не всі готові платити такі кошти, особливо якщо не так часто публікуються заходи. Також серед проблем можна виділити обмежену підтримку локальних авторів, оскільки перевага надається більш відомим організаторам, а локальні події менш видимі.

1.3.2 Eventbrite

Eventbrite спеціалізується на професійних заходах, таких як конференції та фестивалі. За даними Eventbrite, у 2023 році платформа організувала 5 мільйонів подій у 180 країнах, а в 2024 році 62% відвідувачів віддають перевагу гібридним подіям (змішення офлайн та онлайн подій) [4, 12].

Переваги:

- Потужна система продажу квитків.
- Аналітика для організаторів.
- Підтримка великих заходів.

Недоліки:

- Складність для локальних подій.
- Платні функції.
- Відсутність орієнтування на український ринок

Платформа дуже схожа на українську Karabas, але з наявністю безкоштовних заходів і не повністю прив’язаною системою до купівлі квитків. Але по при це, все ще варто враховувати, що платформа більш орієнтова на

платні події з продажами квитків. Сама ж платформа хоч і надає можливість локальним авторам створювати події, але такі події мають меншу видимість, якщо порівнювати з подіями від відомих авторів.

1.3.3 Facebook Events

Facebook Events – інтегрована платформа в соціальну мережу Facebook з 3 мільярдами активних користувачів, яка надає користувачам можливість створювати, знаходити, рекламувати та брати участь у подіях різного масштабу та формату, починаючи від локальних зустрічей і до великих фестивалів чи онлайн-конференцій [14]. Від моменту запуску платформи, вона стала популярним інструментом для організації заходів завдяки широкій аудиторії Facebook та інтеграції з його іншими функціями, такими як групи, сторінки та сам месенджер.

Переваги:

- Простота створення подій.
- Широка аудиторія.
- Користувачі самі створюють контент

Недоліки:

- Обмежений напрямок застосування
- Не популярний серед молоді
- Facebook – це в першу чергу про спілкування

Данна платформа є прикладом гарної альтернативи, яка має все необхідне для свого розширення, але в цей же час, залежність від месенджера Facebook робить платформу менш орієнтованою і менш зручною для користувачів, а також, на даний момент Facebook вже не є настільки актуальною платформою для молоді: молодь 18-24 років в квітні 2024 року в фейсбуку займала лише ~23%

від загальної кількості користувачів, а той же Instagram аж 32% [18, 19]. Може здатися це невелика різниця, але варто пам’ятати, що ці 9 відсотків – це не один десяток мільйонів користувачів. Через це багато організаторів молодіжних подій орієнтуються на інші, більш молодіжні платформи. Також варто пам’ятати, що Facebook – це в першу чергу про спілкування, а потім вже все інше, тому ця платформа не має основної цілі в розвитку інструменту поширення заходів як повноцінної, незалежної.

1.3.4 AllEvents

AllEvents надає користувачам доступ до обширної бази подій, що охоплює тисячі міст по всьому світу (понад 30 000 міст, за даними платформи). На платформі користувачі можуть знаходити різноманітні заходи, починаючи від концертів, вечірок і спортивних подій до бізнес-семінарів, локальних зустрічей і культурних подій [13]. База подій дозволяє організаторам публікувати інформацію про свої заходи, а учасникам знаходити актуальні події у своєму місті чи регіоні.

Переваги:

- Широка база подій.
- Базові можливості фільтрації пошуку.

Недоліки:

- Обмежена локалізація для України.
- Орієнтація на великі заходи

Загалом платформа виступає непоганою альтернативою, але все ще варто враховувати основний мінус – відсутність спрямованості контенту на український ринок. До мінусів також додається обмежена підтримка для локальних заходів, оскільки платформа більше схильна до просування

популярних подій, в той час як розроблене рішення публікує у стрічку події в не залежності від популярності авторів.

1.4 Виявлені обмеження альтернатив

1.4.1 Фрагментація інформації про події

Фрагментація даних про події відбувається, коли вони розподілені між різними платформами, включаючи локальні платформи, веб-сайти організаторів, соціальні мережі (такі як Facebook та Instagram), месенджери (такі як канали Telegram) і навіть офлайн-рекламу. Оскільки не існує єдиної платформи, яка б зібрала всю необхідну інформацію для користувачів, вони змушені витрачати багато часу на пошук відповідних подій.

Можна коротко окреслити декілька наслідків фрагментації:

Втрата часу: Користувачі повинні шукати на кількох платформах, щоб знайти потрібну подію, що знижує ефективність пошуку.

Неточна інформація: Деякі події, особливо місцеві або некомерційні, можуть не бути перелічені на відомих платформах.

Проблема для організаторів: Щоб охопити ширшу аудиторію, організатори повинні публікувати інформацію на кількох платформах, що збільшує їхнє робоче навантаження та можливо, фінансові витрати.

Відсутність спільного стандарту: Інформація подається по-різному на різних платформах, що ускладнює порівняння подій або швидке вивчення специфіки.

1.4.2 Недостатня гнучкість або відсутність фільтрації

Одним із головних недоліків альтернативних платформ пошуку подій, таких як Meetup, Eventbrite, Facebook Events та AllEvents, є їхня відсутність гнучкості або будь-якої фільтрації. Ці сервіси часто пропонують лише найпростіші варіанти фільтрації, і не дозволяють поєднувати більше одного критерію, наприклад, пошук безкоштовних освітніх заходів у певному місці в певний день. Хоча Meetup обмежує фільтрацію подіями певних груп, а Eventbrite зосереджується на комерційних заходах, виключаючи місцеві ініціативи, результати пошуку Facebook Events захаращені нерелевантними подіями та рекламою. Це явно знижує зручність використання та ефективність, змушуючи користувачів витрачати час на перегляд безглуздого контенту або використання інших джерел.

1.4.3 Обмеження для створення подій

Альтернативні платформи для створення та публікації подій часто стикаються з певними обмеженнями, які впливають на їхню функціональність і доступність для користувачів. Наприклад, створення подій може бути доступним лише за умови оплати, що робить цю функцію недоступною для тих, хто не готовий або не має можливості інвестувати кошти. У деяких випадках можливість самостійного створення подій взагалі відсутня, і користувачам доводиться звертатися до адміністрації платформи, щоб отримати дозвіл або технічну підтримку для розміщення їхнього заходу. Такий підхід значно ускладнює процес і може відлякувати потенційних організаторів.

Більше того, на багатьох платформах переважають події, створені відомими авторами, організаторами чи брендами, тоді як маловідомі або локальні автори рідко представлені. Це відбувається з кількох причин. По-перше, локальні автори можуть не бачити сенсу в розміщенні своїх подій на великих платформах, оскільки це може вимагати фінансових вкладень, які вони не в змозі

собі дозволити. По-друге, навіть якщо кошти не є проблемою, платформи можуть не надавати необхідного функціоналу для самостійної публікації подій, що обмежує можливості таких авторів. Крім того, деякі платформи встановлюють надто високі критерії для розміщення контенту, що робить процес звернення до адміністрації складним і недоступним для невеликих організаторів. У результаті локальні або менш відомі автори часто залишаються поза увагою, а їхні події не отримують належної видимості на таких платформах.

1.4.4 Відсутність локалізації для української аудиторії

Жодна з проаналізованих платформ не демонструє чіткої орієнтації на потреби українського ринку чи спеціалізації на подіях, що відбуватимуться в Україні. Серед альтернативи, яка не була згадана як основні раніше, можна виділити лише сервіс Karabas, який спеціально розроблений для просування подій, що відбуваються на території України. Однак його ключовою особливістю є фокус виключно на платних заходах, для яких передбачено продаж квитків. По суті, Karabas функціонує як платформа для реалізації квитків, а не як інструмент для широкого просування подій чи інформування про них.

Інші платформи, які частково охоплюють український ринок, зазвичай також зосереджені на продажі квитків, або пропонують настільки обмежену кількість актуальних подій, що вони не відповідають потребам пересічного українського користувача. Через це пошук подій на таких сервісах часто перетворюється на марнування часу: користувачі витрачають значні зусилля, переглядаючи пропозиції, але в більшості випадків не знаходять жодної корисної чи релевантної інформації про заходи, які могли б їх зацікавити. Відсутність достатньої кількості подій, адаптованих до інтересів української аудиторії, робить ці платформи малопридатними для ефективного використання в контексті пошуку місцевих заходів.

1.5 Пропоноване рішення

1.5.1 Централізована платформа для пошуку та публікації

Сам застосунок має на меті спростити доступ до пошуку та створення подій, в не залежності від фінансових можливостей, популярності чи формату подій для пошуку та / або публікації. Запропоноване рішення зможе надавати користувачам (учасникам та організаторам подій) єдину безкоштовну платформу для пошуку та публікації подій. Хоч застосунок повністю ніколи не зможе вирішити проблему фрагментації публікації подій, але він точно дасть можливість мати зразковий приклад платформи з подіями, де користувачі будуть самі генерувати контент, і одночасно зможуть доєднатись до подій створених іншими користувачами.

1.5.2 Система фільтрації подій

Користувачі можуть швидко знаходити потрібний контент на основі своїх інтересів завдяки зручній та зручній системі фільтрації подій застосунку за обраними категоріями. Функціональність буде значно розширена в майбутньому, зокрема, буде додано можливість пошуку подій за датою та місцем проведення. Технічна основа для реалізації цих функцій уже частково підготовлена завдяки збереженню відповідних даних у базі застосунку, що забезпечує надійну платформу для подальшого розвитку.

Кожен користувач матиме власний персоналізований профіль, який наразі використовується виключно для авторизації та базового доступу до системи. Однак у перспективі профілі мають значний потенціал для трансформації у повноцінні соціальні сторінки, подібні до профілів в Instagram. Це дозволить користувачам не лише створювати та публікувати власні події, але й ділитися ними з підписниками. Таким чином, кожен зможе переглядати добірку подій, організованих чи рекомендованих конкретним автором, що сприятиме

формуванню активної спільноти та поглибленню соціальної взаємодії в межах платформи.

1.5.3 Спрощений механізм створення подій

Маючи аккаунт, кожний користувач зможе створювати події самостійно, без затвердження з адміністрацією платформи. Завдяки простій і зрозумілій формі заповнення інформації про подію зображеній на рис. 1.6, створення події займає лічені хвилини, а публікація відбувається майже миттєво.

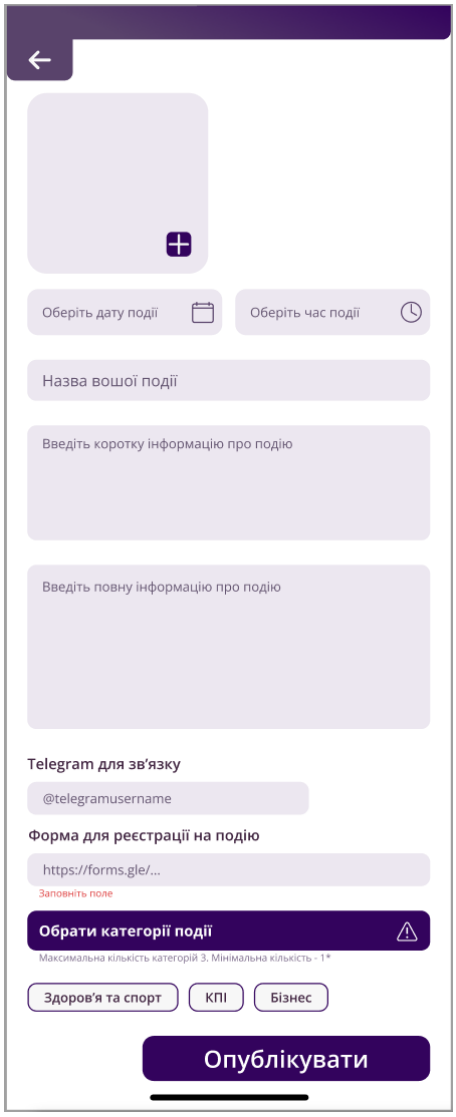

Рис.1.6 Зображення форми для створення подій.

Такий простий механізм надає можливість створення умовно необмеженої кількості подій в короткий проміжок часу.

1.5.4 Фокус на українську аудиторію

Початково застосунок планується бути орієнтований на молодіжну аудиторію Київського політехнічному інституту, а надалі розширюватись за його межі на український ринок. Такий напрямок дасть українським користувачам обирати події з тих, котрі знаходяться в їх місті, країні. Завдяки організаторам локального рівня, кількість публікованих подій на території України буде рости, що в свою чергу підвищить кількість користувачів-відвідувачів, котрі матимуть бажання бути присутніми на таких заходах. Відповідно і популярність застосунку також має зростати. Також важливим фактором відіграє локалізація застосунку, оскільки аналоги AllEvents, Meetup, Eventbrite не мають української мови взагалі.

Висновок

Проведений аналіз, який оцінюється у 139 мільярдів завантажень до 2025 року [6], продемонстрував важливість мобільних застосунків у повсякденному житті. Майбутнє розвитку формується такими тенденціями, як хмарні сховища, штучний інтелект та 5G. На ринку зараз домінують Android та iOS, і мало інтересу до альтернатив.

Відсутність локалізації для України, фрагментація, відсутність фокусу на локальних подіях – основні з недоліків проаналізованих платформ (Meetup,

Eventbrite, Facebook Events та AllEvents). Розроблений застосунок, має стати централізованою платформою з гнучкою фільтрацією, оптимізованим створенням подій та акцентом на українську аудиторію має вирішити ці проблеми.

РОЗДІЛ 2

ПОРІВНЯННЯ ТА ВИБІР ТЕХНОЛОГІЙ

2.1 Критерії вибору технологій

2.1.1 Простота використання

Простота використання технологій має суттєвий вплив на те, як швидко будуть розроблятися рішення, наскільки легко їх буде масштабувати та скільки це буде коштувати. В контексті розробки мобільного застосунку для дипломної роботи, де студенти сильно обмежені по часу та доводиться вивчати нові технології на ходу – простота чи не найважливіший чинник, від котрого залежить, який функціонал вдасться реалізувати. Прості технології дозволяють більше часу виділяти на саму розробку, а не на вивчення всіх аспектів технології.

У контексті масштабування, простота теж має значення. Технології, які мають зрозумілу структуру та функціонал, зменшують кількість потенційних помилок, в не залежності від рівня знань розробника, котрий буде використовувати цю технологію; полегшується адаптація, до нової кількості користувачів, при зростанні популярності застосунку.

Також серед переваг простих технологій можна виділити їх частіше використання серед розробників. Тут можна провести аналогію з мовою програмування Python: завдяки простоті використання, мова набувала та набуває популярності не тільки серед початківців, а й серед знавців сфери програмування. Відповідно, на такі технології набагато легше буде знайти розробника, котрий зможе підтримувати чи вносити новації в застосунок.

Останньою, але не менш важливою перевагою є економічна вигідність таких технологій. Коли є велика кількість розробників на просту технологію, відповідно технологію набагато легше інтегрувати та підтримувати, то й використання такої технології є більш економічно вигідною.

2.1.2 Масштабованість

Масштабованість також є однією з основних критеріїв вибору технологій, коли планується подальший розвиток будь-якого рішення. Коли проєкт початково розраховує тільки на невелику кількість користувачів, то через певний час зростання проєкту прийдеться не тільки інтегрувати нову технологію, щоб підтримувати більшу кількість користувачів, а й буде потрібно переписувати весь функціонал, котрий був оснований на певній технології. В такому випадку, зміна технології призведе до суттєвих витрат часу та фінансів. Важливо обирати технології, виходячи з розрахунком кількості потенційних користувачів, навіть крізь роки, якщо планується довгострокова розробка проєкту.

Масштабованість сприяє легкому розширенню функціоналу і впровадження не використаних елементів технологій. Технології з модульною структурою та гнучкими API дозволяють додавати нові функції без значних змін у коді чи архітектурі, що буде економити час і ресурси під час розробки. Це також полегшує інтеграцію з додатковими сервісами в майбутньому, наприклад, для аналітики чи маркетингових інструментів. Масштабовані рішення також спрощують перехід до платних планів у разі потреби, дозволяючи поступово нарощувати ресурси без необхідності переписувати код.

2.1.3 Економічна ефективність

Економічна ефективність важлива для рішень, котрі в майбутньому планують вихід на відкриті ринки, де при успіху, аудиторія буде зростати, відповідно й ресурсів на розробку потрібно буде більше. Деякі технології можуть бути зручними і вигідними на початковому етапі, але не підходити для більшої аудиторії, а деякі будуть здаватися дорогими на початку, але при росту застосунку, будуть розкривати свої економічні привілеї.

Кажучи за проєкт дипломної роботи, котрий надалі планується розвивати та розширювати, важливо було використовувати технології, котрі на початку мають безкоштовні версії, а надалі матимуть вигідні тарифні плани. Також в розрахунок додавалася можливість пошуку людей у команду розробки. В залежності від технології, ціни на послуги будуть абсолютно різними. Крім того, економічно ефективні технології оптимізують використання серверних ресурсів, що критично важливе для роботи з безкоштовними версіями хмарних сервісів, які мають обмеження на кількість запитів, обсяг даних і пропускну здатність. Технології, що дозволяють ефективно розподіляти навантаження та мінімізувати кількість запитів до бази даних, зменшують ризик перевищення лімітів, що могло б призвести до додаткових витрат на платні плани. Оптимальне розподілення даних між різними сервісами забезпечує економію ресурсів, дозволяючи підтримувати стабільну роботу застосунку навіть при зростанні кількості користувачів.

2.1.4 Інтеграція між технологіями

Інтеграція між технологіями – це ключовий критерій при розробці мобільного застосунку для пошуку та публікації заходів, оскільки взаємна інтеграція між технологіями забезпечує безперебійну взаємодію між візуальною, клієнтською та серверною частинами. Для такого комплексного проєкту, як мобільний застосунок, де потрібна швидка обробка даних і зручний користувацький досвід, якісна інтеграція створює цілісну систему, яка буде стабільно працювати та підтримувати масштабування. Навіть добре масштабовані, оптимізовані і економічно вигідні технології втрачають цінність без ефективної взаємодії з іншими технологіями.

Ефективна інтеграція забезпечує ефективний обмін даними між компонентами, наприклад, між клієнтською частиною на Unity та хмарними

сервісами Firebase. Сумісні API та готові бібліотеки, такі як Firebase SDK, дозволяють розробнику швидко налагодити зв'язок між базою даних та платформою розробки, зменшуючи час розробки та кількість помилок, що виникнуть в процесі. Такі поєднання технологій, забезпечують синхронізацію подій, що підвищує зручність для користувачів, і полегшує подальшу підтримку системи завдяки чітким каналам передачі даних.

Інтеграція сприяє модульності системи, дозволяючи кожній технології виконувати свою функцію. В розробленому застосунку, Unity відповідає за створення архітектури, поєднання візуальних та функціональних елементів, написання коду, Firebase — за збереження даних, Figma — за створення інтерфейсу для застосунку. Подібний розподіл технологій у менш значних ділянках застосунку, наприклад технологію вибору фото з галереї, дає свої переваги: не основні технології можна замінити без значних змін у коді, що спрощує адаптацію до нових вимог, наприклад, додавання персоналізованих рекомендацій по пошуку подій чи інтеграції з іншими платформами, підтримуючи розширення на ширшу аудиторію.

Нарешті, інтеграція оптимізує ресурси та полегшує масштабування. Використання сумісних інструментів зменшує потребу в розробці власних рішень, економлячи час і знижуючи витрати. Добре інтегрована технологія також спрощує діагностику помилок і дозволяє легко додавати нові функції, що створює надійну основу.

2.2 Використані технології

2.2.1 Unity як середовище розробки

Unity – потужний, простий й дуже популярний інструмент для створення ігор та мобільних застосунків. Хоч Unity в першу чергу створений для розробки

ігор, але завдяки відносно простому інтерфейсу, активній підтримці зі сторони розробників, масштабованості та кількості створених технологій під цей інструмент було обрано саме його для розробки мобільного застосунку. Unity дуже простий в освоєні та повсякденній розробці, особливо якщо порівнювати з такими альтернативами як Unreal Engine.

Для створення застосунку було початково обрано версію Unity 2021.3 , але дуже скоро її було замінено на більш новішу версію 2022.3 з довгостроковою підтримкою. В цьому є плюс Unity: коли оновлюєш версію, на котрій розробляється проєкт до більш нової, то не потрібно все вручну переносити, достатньо у відповідному вікні Unity Hub, при виборі проєкту, обрати новішу версію, якщо це потрібно, рис. 2.1.

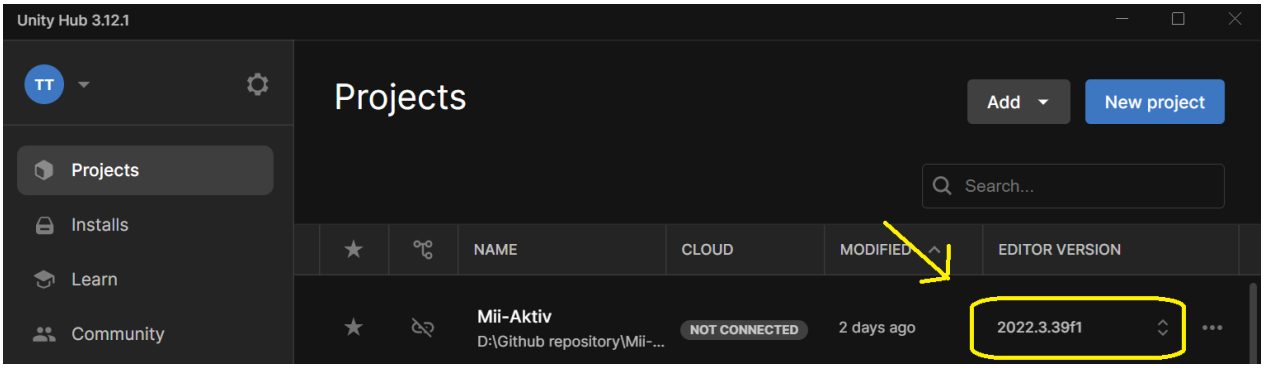

Рис. 2.1 Поле для зміни версії проєкту.

Важливо уточнити, що версії Unity для проєкту можна тільки оновлювати на більш нові, але не повертати на більш старі.

Unity Hub – інструмент для керування проєктами, версіями Unity, контенту в Unity Asset Store. Основна його задача, це можливість створення та запуску проєктів. Unity Asset Store – це платформа, де кожен розробник може поділитися або продати свої готові рішення / інструменти з іншим розробникам.

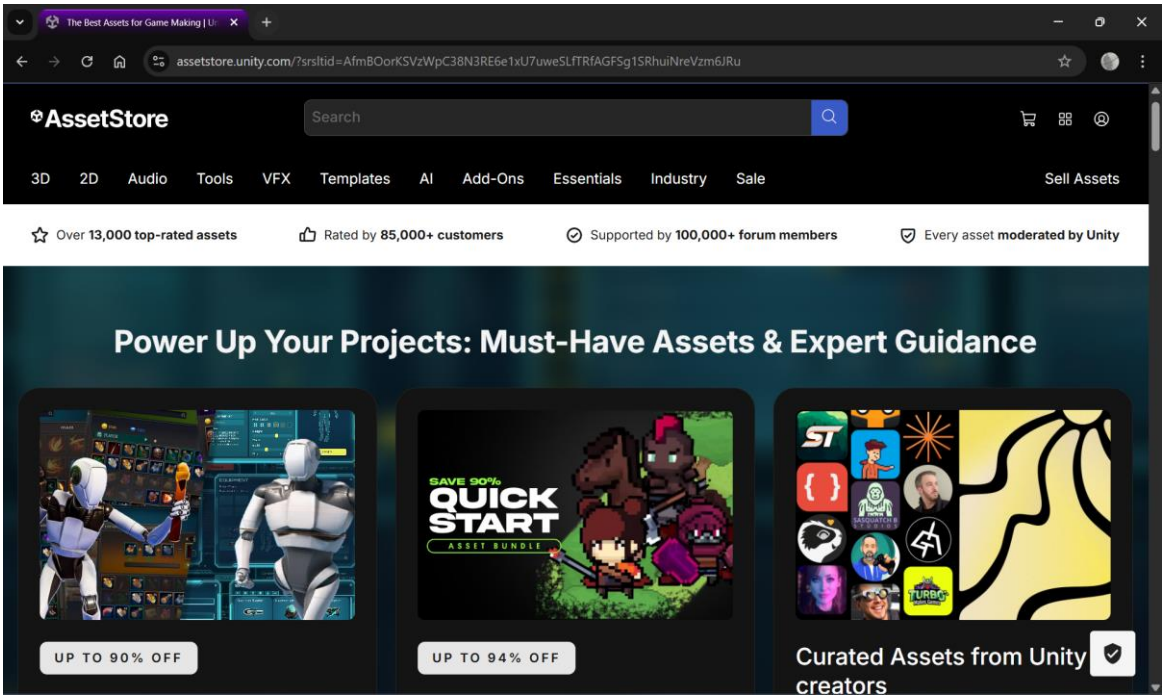

Рис. 2.2 Сайт Unity Asset Store.

Як ми можемо побачити на рисунку, на даній платформі розробник може взяти або купити готові рішення різних сфер: код (технології), готові зображення, звукові ефекти, доповнення до Unity, 3D моделі. Це дуже зручний інструмент, особливо коли потрібно в одиночку розробляти застосунок або гру, до якої потрібні звуки та текстури.

Також Unity має дуже зручну і зрозумілу документацію, що до його вбудованих можливостей. Під час розробки, можна прямо з компілятора перейти до документації та ознайомитись з логікою взаємодії скриптів, елементів.

2.2.2 C# для логіки застосунку

Оскільки було обрано Unity, як одну з основних технологій, то обирати мову програмування не довелося. Основною мовою розробки в Unity є C# (Charp).

C# - це сучасна, об'єктно-орієнтована мова програмування, розроблена Microsoft у 2000. Вона призначена для створення надійних, масштабованих і безпечних програм, які працюють на платформі .NET. Мова поєднує в собі простоту синтаксису, котрий подібний до мов C та C++, із сучасними можливостями, що робить C# універсальним для розробки різноманітних застосунків. Крайня версія C# 14 вийшла у листопаді 2024 року. Кожна нова версія додає зручний синтаксис, спрощуючи розробку.

Офіційним компілятором для мови є Visual Studio (не плутати з Visual Studio Code) від Microsoft, котрий слугує зручним інструментом, для написання коду на мові C#. Компілятор успішно інтегрується з Unity, завдяки чому не потрібно обирати альтернативу і можна використовувати всі можливості розробки на мові. Сама мова C# багато функціональна, що дозволяє на ній писати мобільні застосунки, веб-застосунки, Back-end для сайтів, нейромережі та багато іншого.

2.2.3 Firebase для хмарних сервісів

Firebase - це платформа хмарних сховищ від Google, для розробки мобільних і веб-застосунків. Проста, масштабована, та при правильному використанні економічно ефективна платформа, котра також активно підтримує інтеграцію з Unity. Firebase надає набір інструментів і сервісів, які допомагають розробникам створювати, тестувати та масштабувати застосунки без необхідності керувати власною серверною інфраструктурою. Серед основних інструментів Firebase, котрі були використані є: Firestore, Realtime Database, Storage та Authentication.

Якщо проводити аналогію, з розробленим рішенням, то кожен з елементів використовувався наступним чином:

В Firestore, в зберігається інформація про подію, котра не використовується для пошуку подій, за для економії кількості запитів до Firestore, котра є обмеженою.

Realtime зберігає інформацію, за котрою в нас відбувається, або потенційно може відбуватися пошук події. Також, тут же зберігається мета дані застосунку: підтримані версії, посилання на оновлення застосунку, статус самої бази даних. Таке рішення зумовлено відсутністю ліміту, на кількість запитів до цієї бази даних.

Storage – ідеально підходить для збереження файлів. В застосунку в цій базі даних зберігаються зображення та мета дані до них.

І в останньому елементі Authentication в нас зберігається інформація про користувачів, акаунти яких можна через консоль в Firebase блокувати, видаляти та скидати паролі від них, надсилаючи про це відповідне повідомлення на пошту користувача, щоб він змінив пароль.

2.2.4 Figma для дизайну інтерфейсу

Figma один з популярних інструментів створення інтерфейсів для застосунків під будь-яку платформу. Тут можна створювати інтерфейс не тільки за допомогою векторного малювання, а й використовуючи SVG код, котрий описує векторну графіку. При розробці застосунку, використовувалось обидва методи, доповнюючи один одного.

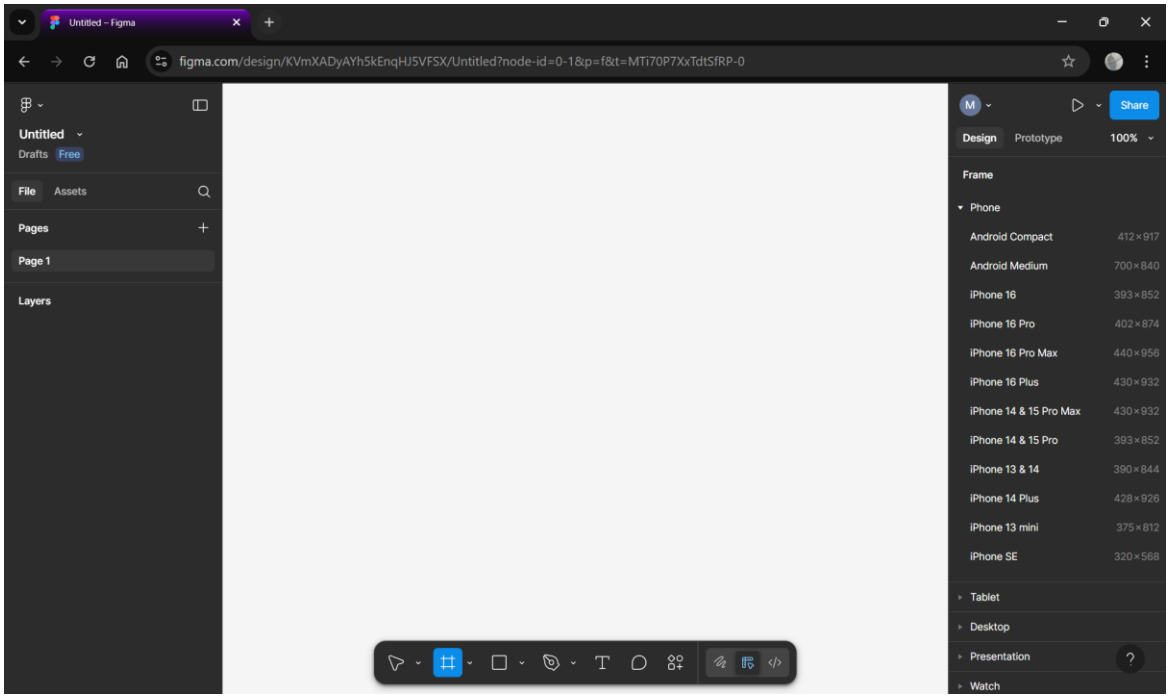

Рис. 2.3 Інтерфейс Figma.

Як можна побачити на рисунку 2.3, користувацький інтерфейс Figma дуже простий і мінімалістичний, на якому зображено три основних елементи для створення UI: вкладка з вибором об’єкта (або об’єктів) для редагування, вкладка з інструментами малювання та вкладка з параметрами для редагування об’єктів. Такий зовнішній вигляд Figma значно спрощує вивчення технології та взаємодію з нею. Серед інструментів малювання, користувачеві надається можливість використовувати та створювати прості фігури, малювати прямі та криві лінії, з можливістю викривлення та заповнення внутрішньої частини, додавати текст та створювати рамки, для певного екрану інтерфейсу. Також присутні параметри редагування об’єктів, тексту, такі як зміна розміру, шрифту, заокруглення, додавання текстури, зміна кольору, маски, прозорість об’єкту та інші. Приклад параметрів для редагування можна побачити на рисунку 2.4.

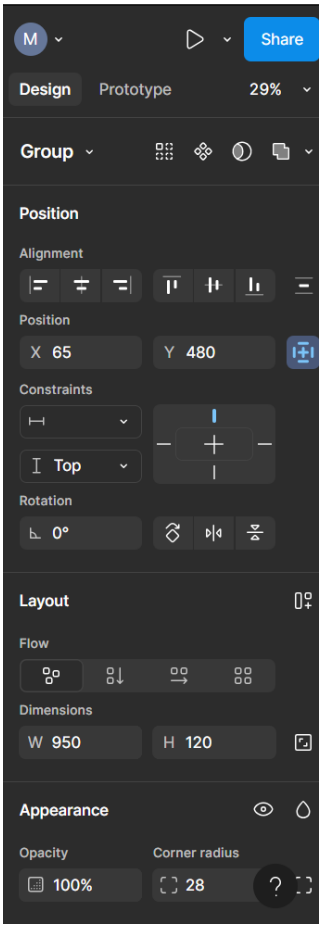

Рис. 2.4 Параметри редагування об’єкту.

Загалом Figma є простим в освоєні інструментом, що дуже важливо, при розробці дипломного проєкту. Експорт файлів із Figma може відбуватися в різних форматах, в залежності від потреб, серед яких .png, .jpg, .svg та .pdf. Після цього, ці файли зберігаються у відповідних папках проєкту Unity. Одним з зручних інструментів Figma є прототипування. Це означає, що розробник може протестувати, як мають відбуватися переходи з одних екранів на інші, при цьому не експортуючи компоненти, та не маючи функціоналу до них. Дуже корисний інструмент, коли потрібно протестувати, як будуть використовувати переходи між сторінками, не витрачаючи час на розробку самого застосунку з метою отримання оптимального користувацького досвіду.

2.2.5 GitHub Desktop для контролю версій

GitHub — це платформа для хостингу коду, контролю версій і спільної розробки, а GitHub Desktop зручний застосунок для Windows, Mac та Linux, котрий надає доступ до описаних можливостей у зручному вигляді для користувача платформи. GitHub Desktop базується на системі контролю версій Git. Вона дозволяє створювати репозиторії для зберігання коду, відстежувати історію змін, працювати з гілками та об’єднувати їх, що сильно спрощує командну розробку. GitHub також підтримує проєкти з відкритим вихідним кодом, надаючи інструменти для управління задачами, вікі, трекери помилок і автоматизацію через GitHub Actions.

2.3 Альтернативні технології та обґрунтування вибору технологій

2.3.1 Альтернативи Unity

Для розробки мобільного застосунку для пошуку та публікації заходів, Unity є чудовим рішенням, але існують альтернативи, такі як Unreal Engine, Godot, Cocos2d-x та фреймворки на кшталт Flutter чи React Native.

Unreal Engine, розроблений Epic Games. Він відомий своєю потужною 3D-графікою завдяки технологіям Nanite і Lumen, що робить його ідеальним для візуально вражаючих проєктів, але надмірно складним і ресурсоємним для простих 2D-застосунків. Також варто враховувати, що в Unreal Engine використовується мова програмування C++, котра сильно видозмінена під потреби рушія, що сильно ускладнює освоєння технології.

Godot - безкоштовний рушій із відкритим кодом, пропонує легкість і гнучкість для 2D-розробки, також підтримує C# і деякі інші мови програмування, але має набагато меншу екосистему порівняно з Unity. Хоч багато розробників

мають думку, що Godot є простішим, економічно вигідним, більш гнучким в порівняні з Unity, але великі компанії при розробці проєктів, більш схильні до розробки саме на Unity, завдяки його сформованості на ринку та великій команді підтримки.

Cocos2d-x, який також орієнтований на 2D і мобільні платформи. Він забезпечує високу продуктивність в розроблених рішеннях, проте значно поступається Unity у зручності створення UI.

Фреймворки, такі як Flutter і React Native, спеціалізуються на UI-орієнтованих застосунках і дозволяють швидко створювати кросплатформні інтерфейси, але їм бракує інструментів для інтерактивних або гейміфікованих елементів.

Unity обрано завдяки його оптимальному балансу між простотою, гнучкістю, масштабованістю та потужними інструментами для створення 2D-інтерфейсів і анімацій. В Unity присутній зручний компонент Canvas для створення адаптивних UI, що ідеально підходить для списків подій, фільтрів та загалом всього UI, а також підтримує інтерактивні елементи, такі як мапи чи календарі, завдяки вбудованим 2D-інструментам: Sprite Renderer, Tilemap.

2.3.2 Альтернативи Firebase

Firebase є популярним вибором завдяки інтеграції з Unity, підтримці real-time баз даних, автентифікації та push-повідомлень. Однак, часто іншим розробникам доводиться шукати альтернативи через певні обмеження Firebase. В розробленому рішенні, обмеження не є суттєвими, але при розробці інших застосунків, такі проблеми можуть виникати, особливо якщо база даних використовується як сервер для обміну даних.

Supabase це open-source платформа, яка пропонує real-time базу даних на основі PostgreSQL, автентифікацію, зберігання файлів і API (REST та GraphQL). Вона добре інтегрується з Unity через REST API, що дозволило б створювати списки подій, фільтри та форми для публікації при виборі цієї бази даних. Supabase може підходити завдяки підтримці SQL, що спрощує складні запити, і гнучкості розгортання на власних серверах.

Parse — ще одна open-source альтернатива, яка забезпечує NoSQL базу даних, автентифікацію, push-повідомлення та зберігання файлів. Вона має SDK для Unity і дозволяє self-hosting, забезпечуючи контроль над даними.

AWS Amplify має інструменти для автентифікації, зберігання, API (AppSync), але потребує складнішого налаштування для Unity порівняно з Firebase. Хоча серед його основних переваг можна виділити інтеграцію з екосистемою AWS для масштабування.

Firebase обрано серед всіх альтернатив, завдяки офіційному Unity SDK, який суттєво спростив інтеграцію з базою даних, завдяки вигідному економічному плану, як на початкових стадіях розвитку застосунку, так й на більш пізніх. Важливим нюансом є, що всі запити до Firebase пишеться на мові C#, що спрощує взаємодію з базою даних і не потрібно вивчати нові технології, достатньо буде подивитися, який функціонал надає Firebase через Unity SDK.

2.3.3 Альтернативи Figma

Figma представляє собою безкоштовний, популярний інструмент для векторного малювання, основною задачею якого є створення користувацького інтерфейсу та прототипування для десктопних та мобільних застосунків. Серед альтернатив, можна виділити декілька варіантів.

Adobe XD, пропонує велику різноманітність інструментів, для створення користувацького інтерфейсу, також має відповідні інструменти для прототипування, але в цей же час платна підписка на місяць за ~43$, що є серйозною перешкодою для використання для більшості людей, особливо студентів.

Sketch – ще один потужний інструмент при створенні векторного дизайну. Він пропонує розширену бібліотеку плагінів, працює офлайн, що може бути перевагою для стабільності, підтримує створення інтерактивних прототипів, хоча й з меншою гнучкістю порівняно з Figma. Sketch має платну підписку (близько $120 на рік) і прив’язаний до macOS, що робить його набагато менш доступним порівняно з Figma, яка доступна на різних платформах та має безкоштовний план.

Розробники, які шукають безкоштовне рішення, вважають Penpot привабливим, бо це інструмент з відкритим кодом, який працює в браузері, як і Figma, та спрощує онлайн-співпрацю. Хоча його інтерфейс користувача менш зручний, а екосистема плагінів і шаблонів менш розвинена, він дозволяє створювати прототипи та експортувати ресурси в SVG або CSS, які сумісні з Unity.

Figma в свою чергу, зуміла поєднати все необхідне для розробки інтерфейсу застосунку: безкоштовне використання, безліч плагінів, зручний інтерфейс використання, простий експорт та імпорт, використання SVG коду.

2.3.4 Альтернативи GitHub Desktop

SmartGit дає розширені можливості керування репозиторіями Git, такі як підтримка Bitbucket, GitLab та GitHub, а також інструменти пошуку завантажених оновлень проєкту та вирішення конфліктів, котрі корисні для складних проектів з великою кількістю ресурсів, таких як розроблений

застосунок. GitKraken дозволяє працювати з величезними файлами за допомогою Git LFS та пропонує простий у використанні інтерфейс з візуалізацією гілок; проте безкоштовна версія підходить лише для бізнес-застосунків. Sourcetree від Atlassian пропонує безпечний графічний інтерфейс користувача (GUI) для Git та Mercurial, хоча він може бути менш візуально привабливим та складнішим у налаштуванні.

GitHub Desktop було обрано, бо він поєднує все необхідне: простоту використання, безкоштовний функціонал та тісний зв'язок з GitHub. Для розробників, що працюють над розробкою застосунків, графічний інтерфейс GitHub Desktop спрощує виконання рутинних завдань, таких як оновлення версій проєкту, надсилання змін та розгалуження, і все без необхідності використання командного рядка. GitHub Desktop вимагає менше налаштувань, ніж Sourcetree, і простіший для розуміння, ніж SmartGit або GitKraken.

Висновок

Основа для будь-якої розробки – правильно обрані технології. Від правильно підібраних технологій, може залежати швидкість та якість розробки, кількість витрачених ресурсів, чи то фінанси й об’єм часу на розробку, чи кількість залучених людей до розробки.

В основі вибору технологій для розробки застосунку, було враховано кількість часу на розробку та освоєння технологій, можливості безкоштовного доступу до інструментів розробки та можливості впровадження в різні системи. Для цього було виділено декілька основних пунктів, за якими було обрано технології: простота використання, масштабованість, економічна ефективність та інтеграція між технологіями.

За описаними параметрами, було обрано декілька основних технологій для дизайну, проєктування, розробки й налаштування проєкту, хмарного зберігання

даних та контролем версій проєкту: Figma, Unity в зв’язці з C#, Firebase та GitHub Desktop відповідно. Кожна з технологій має альтернативи, з якими вона була порівняна й обрана, як найкращий варіант з існуючих для реалізованого рішення.

РОЗДІЛ 3

ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Архітектура мобільного застосунку

3.1.1 Загальна структура

Мобільний застосунок побудований на основі клієнт-серверної архітектури, де інтерфейс намальований у Figma, клієнтська частина працює в середовищі Unity, а серверна частина використовує Firebase для збереження та обробки даних. Загальну структуру застосунку можна зобразити, як показано на рис. 3.1.

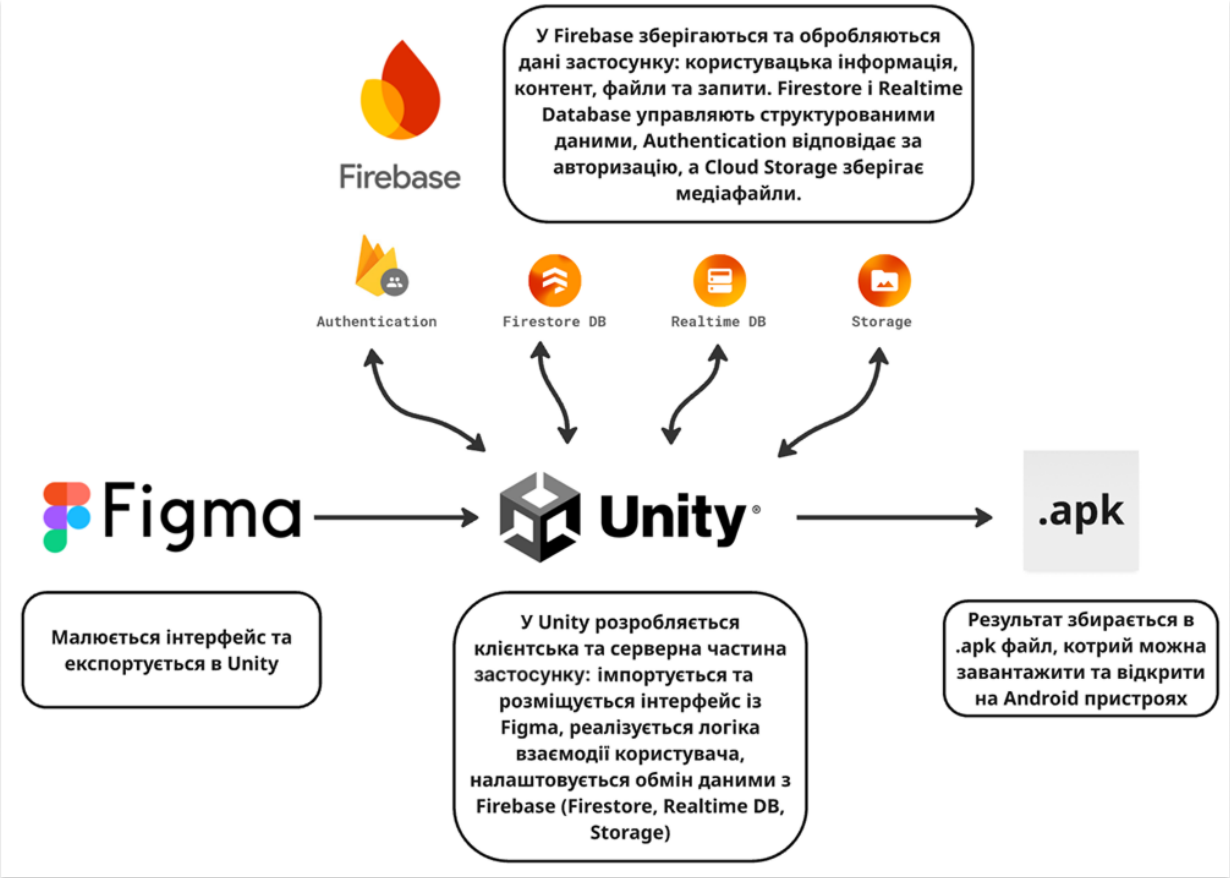

Рис. 3.1 Зображення загальної структури застосунку.

Логіка роботи організована таким чином, щоб взаємодія між інтерфейсом (UI), обробкою даних та бізнес-логікою була чітко розмежованою на рівні коду. Такий метод дозволяє легше адаптувати застосунок під нові вимоги при впроваджені нового функціоналу.

Інтерфейс, містить усі необхідні елементи керування, такі як кнопки, поля введення, списки подій тощо. Вигляд додатку окреслюється чотирма основними вкладками: основна (сторінка з зображеними подіями), вибір категорій для фільтрації, створення подій та профіль. Більш загальний вигляд з усіма екранами застосунку можна побачити на рис. 3.2.

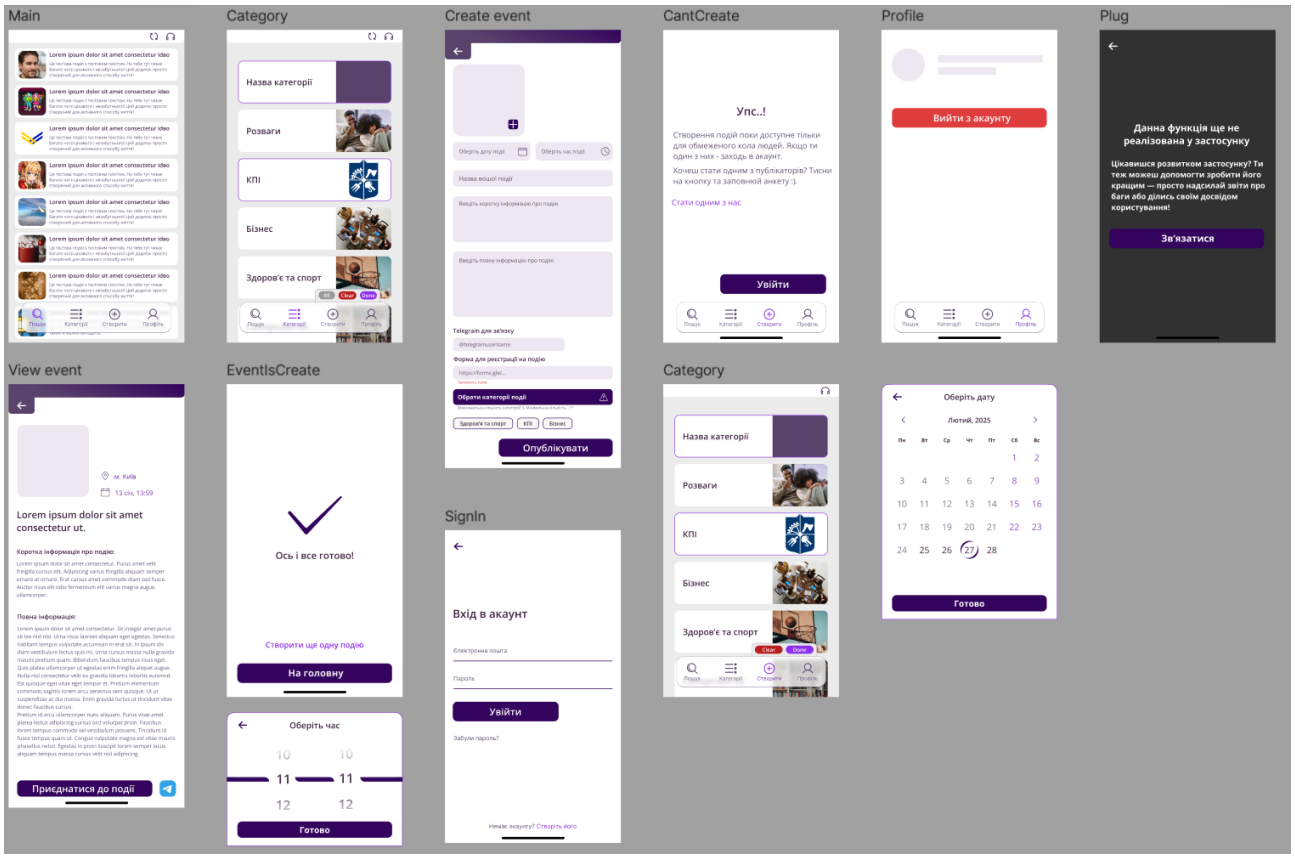

Рис. 3.2 Загальний вигляд інтерфейсу мобільного застосунку.

Основне завдання UI – надавати зручний, простий і візуально приємний доступ до функцій програми. Сам інтерфейс був намальований з використанням

SVG коду, хоч й більшу частину було намальовано стандартними інструментами малювання в Figma.

Клієнтська частина реалізована для платформи Android та відповідає за інтерфейс, обробку введених даних та взаємодію з сервером. Вона містить систему навігації між екранами, бізнес-логіку, управління введенням користувачів та інтеграцію з Firebase.

Серверна частина побудована на базі хмарного сервісу Firebase, що забезпечує зберігання інформації та медіа файлів, автентифікацію для користувачів, а також синхронізацію даних. Firestore та Realtime Database використовуються, щоб зберігати та обробляти інформацію про заходи, користувачів і взаємодії. Firebase Authentication відповідає за безпечний вхід, а Cloud Storage – за збереження файлів. В нашому випадку збереження зображень та мета дані до них.

Взаємодія між клієнтом та сервером відбувається через відправку асинхронних запитів до бази даних, отримання відповідей у реальному часі та оновлення інформації в застосунку та / або на стороні серверного сховища. Це означає, що застосунок не зупиняє роботу під час очікування відповіді від сервера, а продовжує виконувати інші завдання. На практиці це означає, що користувач може авторизуватися, переглядати список подій, створювати власні заходи та завантажувати медіа контент без візуальних зависань через обробку даних з бази даних.

3.1.2 Модель: обробка даних із Firebase

В моделі взаємодії з Firebase фігурують чотири основні елементи: Firestore Database (Firestore), Realtime Database, Firebase Cloud Storage (Storage) та

Firebase Authentication (Authentication). Кожен з елементів відповідає за певну частину даних, в обробці котрих є ліпшим рішенням за інші.

Firestore використовується для зберігання структурованих даних, тобто така інформація як про події, а саме опис, короткий опис, посилання на зображення у Storage, посилання на телеграм організатора та посилання на реєстраційну форму. Тут важливо зазначити, що в даній базі даних зберігається інформація, котра не використовується і не планується до використання для пошуку подій. Це обумовлено тим, що в Firestore кожне зчитування, запис та видалення є ліміти. Після використання ліміту, потрібно буде платити за додаткову кількість дій. Загалом, структура зберігання інформації про подію має наступний вигляд, що зображено на рис. 3.3.

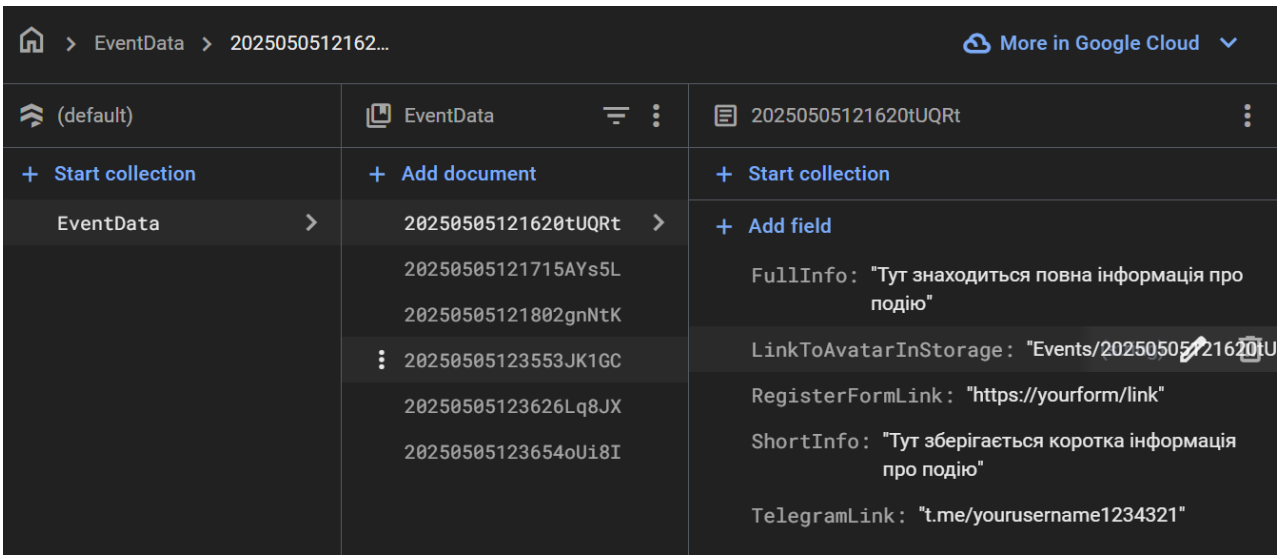

Рис. 3.3 Зображення структури збережених даних у Firestore.

Структура спрощує роботу з інформацією завдяки організації даних у вигляді колекцій і документів.

Realtime Database використовується для даних, котрі будуть використані для фільтрації, або котрі потенційно можуть бути використанні для фільтрації подій. Тобто, зараз в застосунку відсутній пошук по назві, але потенційно ця

функція може бути реалізована, тому такі дані зберігаються саме тут. Як це загалом працює: коли користувач додає новий захід з певними заданими категоріями, він з’являється в списку подій у всіх. Як тільки інший користувач оновить список заходів, або відбере заходи по відповідній категорії, з котрою була створена нова подія, у результатах буде видано новостворену подію. Хоч початково може здаватися, що в Realtime Database буде вигідно зберігати абсолютно всю інформацію про події, щоб в майбутньому можна було по будь-якому елементу реалізувати пошук, але на відміну від Firestore, який має обмеження що до кількості зчитувань, записів та видалення даних, Realtime Database має ліміти по кількості збережених даних у цій базі даних (1Гб), а також по кількості завантаженої інформації користувачами (360Мб на день). Тому, тут не варто зберігати інформацію, котра буде займати забагато місця в сховищі. Структуру збереження динамічної інформації у Realtime Database можна побачити на рис. 3.4.

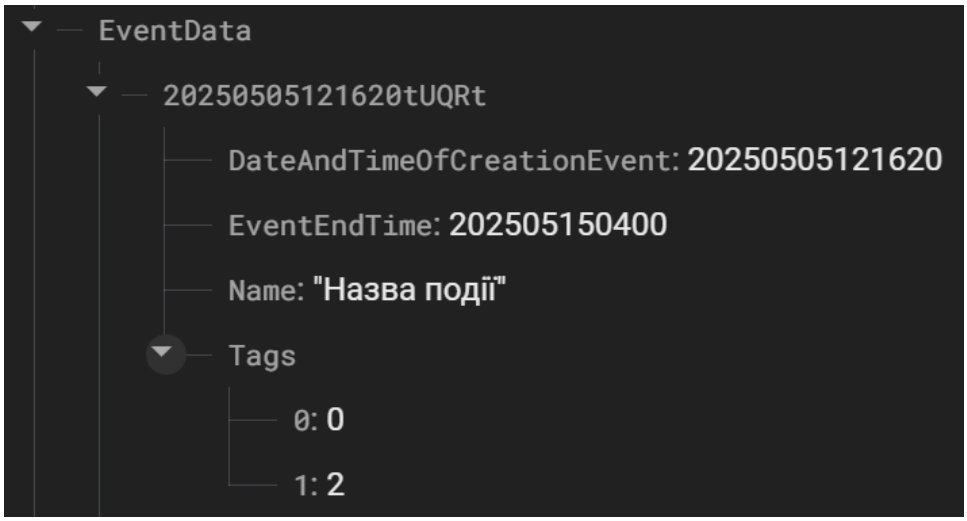

Рис. 3.4 Приклад структури збережених даних у Realtime Database.

Для збереження файлів, а саме зображень та їх мета даних використовується Storage. Цей елемент відрізняється від попередніх можливістю безкоштовного зберігання до 5Гб даних, 30Гб на місяць завантаженої та

вивантаженої інформації, 210тис. операцій завантаження та аж 1 890тис. операцій скачування. Тобто цей елемент найкраще підходить для збереження об’ємної кількості інформації. Коли користувач створив подію з фото для свого заходу, то фото зберігається у Storage, а в Firestore зберігається лише посилання на цей файл. Структура зберігання медіа даних зображена на рис. 3.5.

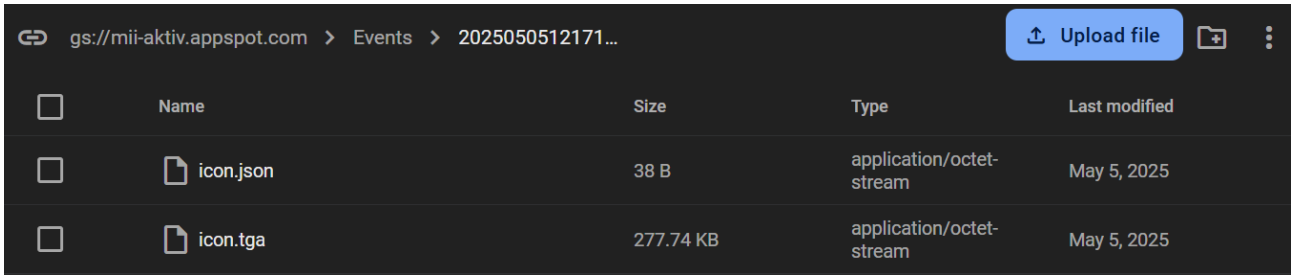

Рис. 3.5 Зображення структури зберігання даних у Storage.

Authentication призначений для авторизації користувачів. Це може відбуватися як за допомогою простого використання пошти та паролю, як це зараз реалізовано в додатку, так і через номер телефону, Google аккаут, соціальні мережі (наприклад Facebook), Apple. Firebase автоматично зберігає сесію користувача, навіть після перезапуску програми, а адміністратори бази даних мають можливості через інтерфейс Firebase, або розроблений ними інтерфейс, створювати, блокувати та видаляти акаунти користувачів. Панель керування данними має вигляд рисунку 3.6.

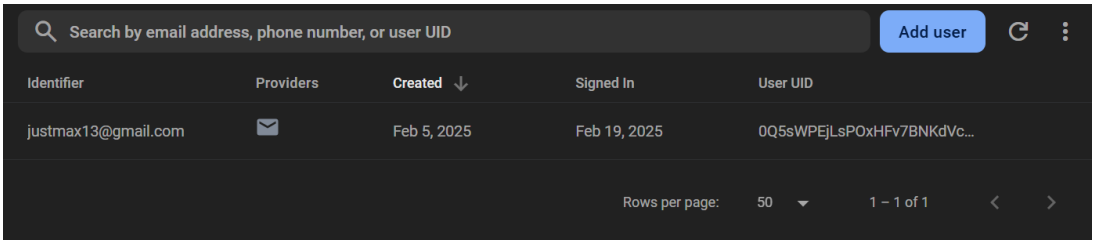

Рис. 3.6 Панель керування аккаунтами користувачів.

Описаний вище розподіл даних по чотирьом елементам Firebase дає можливість використовувати кожен елемент за своїм цільовим призначення, і замість того, щоб один елемент бази даних виконував багато роботи, навантаження фрагментується між кількома.

3.1.3 Вигляд: інтерфейс користувача в Unity

Щоб інтерфейс попав в Unity, початково його треба намалювати або згенерувати за допомогою коду в Figma. Після цього, інтерфейс експортується в Unity у вигляді png зображень у заданих пропорціях. Дані зображення називаються спрайтами (sprites). Надалі, спрайти будуть встановлені на раніше заготовлений шаблон інтерфейсу в Unity, котрий має мати аналогічні екрани, розміщенні елементи, їх розміри. В залежності від послідовності розробки, даний інтерфейс в Unity буде або ж вже мати написаний кодом функціонал, але зображення початково залишиться лише візуальним, а потім до нього реалізують функціонал. В Unity для прив’язки інтерфейсу до функціоналу використовується інспектор (Inspector), рис. 3.7.

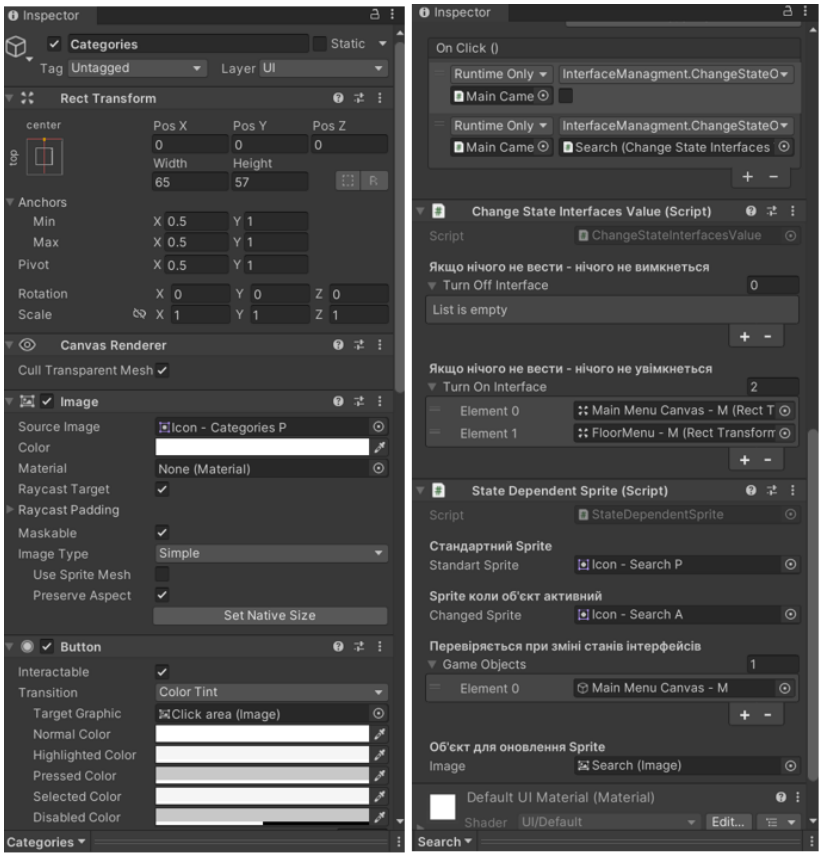

Рис. 3.7 Приклади вигляду інспектора в Unity.

Інспектор – зручний інструмент, завдяки котрому ми можемо бачити та змінювати інформацію про об’єкт, його компоненти, характеристики та іншу інформацію. Коли ж нам потрібно додати якийсь скрипт (функціонал) до об’єкту, ми для початку обираємо сам об’єкт на сцені, а після натискаємо кнопку “Add Component” внизу інспектора. Після цього, в залежності від написаного скрипту, ми маємо або не маємо можливості змінювати данні. Наприклад, скрипт “StateDependentSprite” з рис. 3.7 має такі параметри для передачі як Standart Sprite, Changed Sprite, Game Objects, Image.

3.1.4 Логіка взаємодії компонентів в Unity

Вся взаємодія логіки застосунку відбувається через вбудовані можливості та компоненти Unity, та написані розробником скрипти на C#. Такі складові як візуальне розташування елементів на сцені, базові компоненти об’єктів (наприклад Transform), визначення співвідношення сторін екрану – це все частина вбудованої логіки взаємодій, яка надається розробникам в Unity як зручний фундамент для базових дій над об’єктами. В свою чергу розробник може або скористатися вже готовими рішеннями базових задач, або навіть написати даний функціонал самостійно. Зрозуміло, що тільки базових функцій буде не достатньо, щоб створити застосунок, тому розробник має написати власну, більш складну логіку взаємодії, в залежності від поставлених задач.

Компоненти (вони ж скрипти) створені розробником, щоб мати можливість додаватися на об’єкти, мають успадковуватися від базового класу MonoBehaviour. Цей клас надає багато можливостей компоненту, основними серед яких можна назвати виконання дій покадрово (методи Update(), LateUpdate()), виконання дій з вказаним проміжком часу оновлення фізики (FixedUpdate()), виконання певних дій перед запуском основної логіки застосунку (Awake(), Start()) та можливість доступу до батьківського об’єкту.

Скрипти можуть взаємодіяти між собою посилаючись на інші: створюючи змінні, описуючи логіку в методах та устатковуючись. На рис. 3.8 продемонстровано, як ми можемо створити змінну класу, котрий є базовим функціоналом – TextMeshProUGUI з назвою змінної _tmp; також одночасно можемо устатковуватися від іншого класу, що написаний розробником

```
public class TextDateValue : FieldIsFilled
{
    [SerializeField] private TextMeshProUGUI _tmp;
public abstract class FieldIsFilled : MonoBehaviour
```

Рис. 3.8 Приклад взаємодії класів.

Можна побачити, що хоч клас TextDateValue не устаткований від MonoBehaviour, батьківський клас FieldIsFilled устаткований від нього, що надає можливість класу TextDateValue повністю використовувати функціонал MonoBehaviour.

3.2 Основні механіки застосунку

3.2.1 Пошук подій: перегляд і фільтрація

Пошук відбувається методом перегляду стрічки з найновішими подіями, а фільтрація за допомогою фільтрів. Такий простий метод одночасно робить пошук простим та зручним для використання. Розберемо для початку головну стрічку подій.

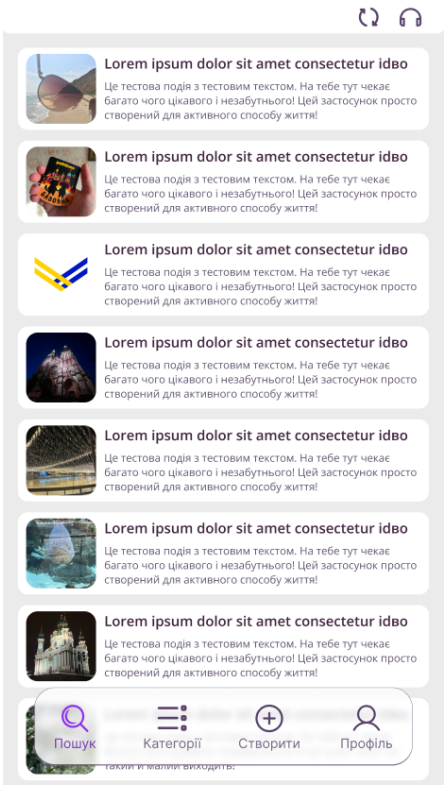

Рис. 3.9 Зображення головного екрану зі стрічкою подій.

На рис. 3.9 ми бачимо сам головний екран, на якому окрім самих подій також зображено верхню панель та нижнє навігаційне меню між екранами. На верхній панелі розміщено дві кнопки: оновлення подій на головному екрані та посилання на технічну підтримку. Нижнє навігаційне меню дозволяє переходити між різними вкладками, такими як пошук, категорії, створення подій і профіль користувача.

Сама область з зображеними подіями, візуально зараз відображає 8 подій при співвідношенні сторін 16:9, але функціонально завантажує 15 таких карток з інформацією про події і у користувача є можливість прокрутити даний екран вниз, а після закінчення списку буде відображена кнопка завантаження ще 10 подій. Функціонал реалізований таким чином, що можна задати будь-яку кількість подій, що завантажиться в стрічку, але було обрано саме 15 для оптимізації кількості запитів у базу даних.

На рис. 3.10 ми бачимо вкладку вибору категорій, на котрій так само є верхня панель та нижнє навігаційне меню.

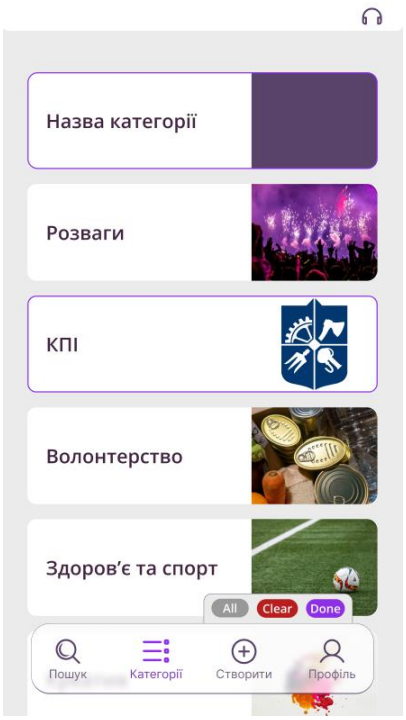

Рис. 3.10 Зображення екрану вибору категорій для фільтрації.

Також, на цій вкладці було додано панель з трьома кнопками: обрати всі категорії, очистити вибір категорій та підтвердити вибрані категорії. Візуально, обрана категорія виділяється рамкою навколо.

Якщо казати про саму реалізацію завантаження та фільтрування подій, вона відбувається у декілька етапів. Спочатку ми отримуємо список подій з Firestore, котрий вже відсортований самою базою даних по ID, а в нашому випадку ID події містить в назві дату та час створення події. Тепер, ми перевіряємо, чи обирав користувач категорії, за котрими потрібно відібрати події: якщо не обрав, ми просто повертаємо список певної заданої кількості подій. У випадку, коли присутні категорії для фільтрації, ми в Realtime Database беремо почергово списки події, котрі зафіксовані за певними категоріями, а потім уніфікуємо в один єдиний список, котрий буде відсортований по ID (тобто по даті створення. Ми не зчитуємо одразу всі події з бази даних, бо це б забрало велику кількість операцій зчитування. Після отриманого списку, для кожного елементу маємо завантажити з Storage зображення, за умови що воно є. Цей етап, як і саме завантаження списку подій відбувається асинхронно, відповідно інтерфейс застосунку у цей час не блокується. Далі потрібно створити об’єкти, котрі поступово будуть завантажуватися у інтерфейс головного екрану (рис. 3.9). Для цього в нас має бути вже заготовлений шаблон такого об’єкту, в котрий ми просто вставимо отриманні дані з бази даних. Після цього йде перевірка, чи є в нас хоч ще одна подія у базі даних. Якщо в нас така подія присутня, то ми додаємо в кінці списку кнопку “Завантажити ще”.

3.2.2 Створення подій: введення даних і завантаження медіа

Створення подій в застосунку можлива тільки за умови, що користувач знаходиться в аккаунті. Форма для створення події зображена на рис. 3.11.

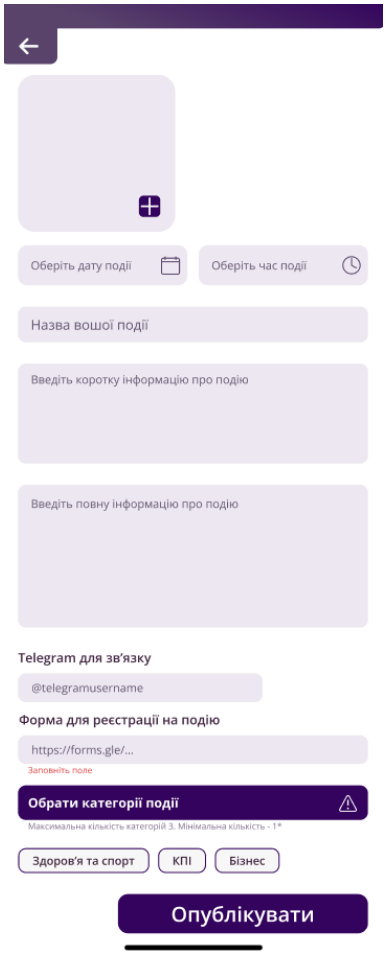

Рис. 3.11 Екран створення події.

Як можна побачити, для створення події користувачу надано всі необхідні поля для заповнення: вибір зображення з галереї, вибір дати та часу події, назва, коротка і повна інформація, телеграм для зв'язку, поле посилання на форму реєстрації та вибір категорій. Екран в більшості випадках цілком не влізе в екран смартфону, тому реалізовано прокручування. Є поля, котрі не обов'язково заповнювати, а саме це вибір зображення. Всі інші мають бути заповнені, в інакшому випадку після натискання з'являться попередження про незаповнені поля і подія не буде створена. Для вибору категорій події, було створено окремий інтерфейс, з обмеженням на вибір 3х категорій та виключеною кнопкою вибору всі подій. Все інше ідентичне.

Після натискання на кнопку створити, та за умови, що всі необхідні поля заповненні – починається збереження даних в базу даних. Для початку ми почнемо зберігати зображення для події, якщо його додали. Ми беремо зображення та конвертуємо його в 2 файли. Перший файл має формат .tga, який зберігає кожен піксель зображення напряму, як набір байтів, без стискань, подібних у форматах .jpeg, .png. Завдяки цьому, формам дуже легко програмно обробляється. Другий файл формату .json, в якому вже зберігаються мета дані зображення, наприклад, розмір збереженого зображення. Збереження 2х файлів асинхронне, тому поки вони зберігаються, виконуємо збереження інших даних події.

Данні з усіх текстових полів записуються в один об’єкт класу EventData, котрий створений для подальшої структурованої передачі даних про події. Далі генерується унікальний ID, котрий містить в собі дату та час до секунд, а також п’ять випадкових букв англійського алфавіту, з врахуванням регістру введення.

Для даних які будуть зберігатися в Firestore, створюється словник у вигляді { string, object }, де string це назва поля, за котрим будуть подальші звернення для отримання інформації, а object – сама інформація, котра буде зберігатися. Хоч object в C# і може зберігати будь який тип, але в базі даних може зберігатися тільки деякі стандартні типи, такі як string, int, bool і подібні. При створенні словника, ми його заповнюємо даними, як зображено на рис. 3.12.

```
// Запис до Firestore
var dataForFireStore = new Dictionary<string, object>()
{
    {"ShortInfo", eventData.ShortInfo},
    {"FullInfo", eventData.FullInfo},
    {"TelegramLink", eventData.TelegramLink},
    {"RegisterFormLink", eventData.RegisterFormLink},
    {"LinkToAvatarInStorage", eventData.LinkToAvatarInStorage},
};
```

Рис. 3.12 Збереження інформації в словник для Firestore.

Після цього, данні зберігаються у вигляді документу в колекції у Firestore. Результат збереження даних буде подібний до рис. 3.3.

Дані, котрі будуть збережені в Realtime, проходять схожий алгоритм. Створюємо словник у вигляду { string, object } і зберігаємо такі дані як назву події, дата та час публікації, дата та час кінця реєстрації на подію та обрані категорії, рис. 3.13. ID події такий же, як і при збережені даних в Firestore.

```
// Запис до Realtime
var dataForRealtime = new Dictionary<string, object>()
{
    {"Name", eventData.Name},
    {"DateAndTimeOfCreationEvent", eventData.DateAndTimeOfCreationEvent},
    {"EventEndTime", eventData.EventEndTime},
    {"Tags", eventData.Tags},
};
```

Рис. 3.13 Збереження інформації в словник для Realtime.

В Realtime також зберігаються цілі списки події, котрі поділені за категоріями. Тобто, кожна категорія має свій список з переліком записаних подій. Тому, після основного запису даних в Realtime, ми додаємо ID події у списки, до яких прив’язана подія.

3.2.3 Аутентифікація

Аутентифікація в застосунку відбувається через Firebase Authentication, методом використання пошти, як логіну, та паролю. Від того, чи користувач авторизований, залежать екрани, котрі будуть відображені користувачу: наприклад, якщо користувач не авторизований, для нього у вкладках створення події та профілю будуть екрани для входу в аккаунт, а якщо користувач в

аккаунті, то йому буде відображено екрани створення події та профіль відповідно.

Сам вхід в аккаунт проходить достатньо просто: користувач у відповідній вкладці вводить інформацію про аккаунт, рис. 3.14.

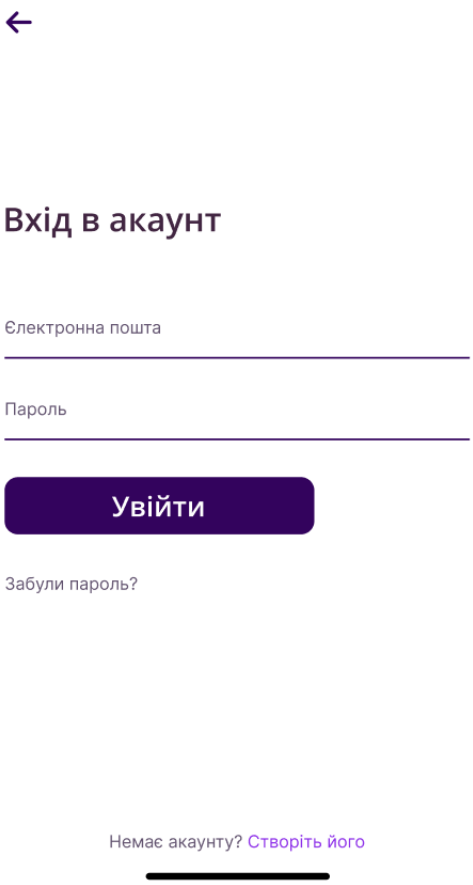

Рис. 3.14 Екран входу в акаунт для користувачів.

При введені інформації і натисканні на кнопку “Увійти”, якщо інформація збігається з тією, що збережена в базі даних, то користувач успішно авторизувався і для нього розблоковується вкладка з створенням подій та профіль. Коли ж користувач вводить не коректну інформацію, то він отримує текст помилки, що саме не так: не правильна пошта, не правильний пароль, багато спроб сходу за короткий час і подібні. У випадку, коли користувач

перезавантажує застосунок, то відбувається перевірка, чи користувач взагалі з аккаунтом, а якщо так, то чи не заблокований аккаунт адміністраторами.

Якщо казати про реєстрацію аккаунту, то вона поки можлива тільки через адміністратора, який має доступ до консолі Firebase. Це рішення було обрано по причині, що поки не було передбачено систему фільтрації спам контенту при створені події, тому, створення подій на даному етапі розробки будуть мати тільки довірені адміністрацією особи. Подібний запит можна надіслати через форму, посилання на котру надається у вкладці створення події, коли користувач не авторизований.

3.2.4 Видалення застарілих подій

Коли користувач входить в застосунок, відбувається ініціалізація бази даних, під час котрої відбувається й перевірка на застарілі події. Для цього, потрібно взяти певну кількість подій з кінця списку Realtime, котрі відсортовані за спаданням за кінцевою датою та часом реєстрації події. Тобто відбираються ті події, котрі мають найменше значення в форматі YMDHMS. Після цього, в циклі, відбуваються ітерації з перевіркою, чи менше значення YMDHMS за дату та час у теперішній момент. До прикладу, якщо в нас подія має дату та час події 2024 12 31 12 00 00, а у теперішній момент 2025 01 01 12 00 00, то перше значення менше другого, через що можна зробити висновок, що вже час на реєстрацію події пройшов. Важливо зазначити, що у прикладі одне число було візуально розділено пробілами, для кращого розуміння, а в коді це число є одним цілим.

Даний метод видалення застарілих подій, було обрано з розрахунком, що користувачів, котрі просто входять в застосунок в рази більше, ніж кількість створених подій за один період часу.

3.3 Оптимізація використання бази даних

3.3.1 Сортування за ID для ефективного пошуку

До оптимізації, ID для певної події генерувалось випадково, інструментами Firebase. Тоді, події сортувалися за значенням, котре бралося з окремого поля “ DateAndTimeOfCreationEvent”. Це спричиняло велику кількість зчитувань у Firebase, або ж велику кількість відправленого трафіку на клієнтську сторону з Realtime, якщо б реалізація поля була саме в цій частині. Відповідно, щоб знайти події, котрі були найновішими по даті створення, потрібно було б взяти та перебрати всю базу даних і відсортувати її. Це не створювало великих проблем, коли в базі даних 5, 10, або 30 подій, але коли подумати в масштабах 100, 200 або більше подій, то проблема набуває серйозний характер: ми мали б зробити що найменше по одному запиту на одну подію, не враховуючи інші технічні запити. Це б відбувалося кожен раз, коли користувач хотів отримати новий список подій.

Якщо уявити, що в базі даних 100 подій і 1 користувач за день оновить 10 раз список подій, не враховуючи навіть те, що він буде сортувати події за категоріями, то це вже 1000 запитів за день. Спираючись на те, що в тому ж Firestore кількість запитів на зчитування в безкоштовній версії 50 000, то в даному випадку застосунок міг би мати максимум активних 50 користувачів за день (1000 * 50 = 50 000).

Для того, щоб уникнути даної проблеми, було вирішено вписати дату та час створення події в саме ID. Новий формат ID, котрий використовується і на даний момент зображено на рис 3.15.

Рис. 3.15 ID нового формату.

Firebase зберігає події у порядку спадання значення ID, відповідно скориставшись цим, вписуємо дату та час події перед унікальними символами. Всього символів 5, і вони генеруються випадково, з врахування варіацій різного регістру. Може здатися що 5х символів мало, але керуючись даним методом реалізації, в одну секунду може бути створено аж 380 204 032 подій, чого більш ніж достатньо, навіть якщо мати цільову аудиторію, як у найпопулярнішіших застосунків на даний момент.

3.3.2 Зменшення кількості запитів до Firestore

Початково, використовувався всього один елемент з двох, а саме Firestore. Вибір був обумовлений тим, що розробники Firebase рекомендували використовувати саме цей елемент, через його більш просунутий функціонал та можливості, в порівнянні з Realtime. На початку розробки, використання Firestore дійсно набагато розширювало можливості, полегшувало розробку і давала вже реалізовані рішення стандартних завдань, при взаємодії з базою даних.

Розробка виключно на основі Firestore відбувалось рівно до того моменту, як не було створено з десяток подій через функціонал застосунку. Саме створення не створювало проблем, а от під час звичайного перегляду та фільтрування подій за категоріями, відбувалася величезна кількість запитів до бази даних, кількість котрих була обмежена. Ситуація б тільки погіршувалася,

зважаючи на те, що сортування планувалося не тільки за одними категоріями, а й за іншими параметрами, такими як міста та назви. Через зазначені причини, використання Rialtime, де немає обмеження по кількості зчитувань, записів та видаленню, мало в геометричній кількості зменшити кількість запитів до Firestore.

Для реалізації даного методу, дані, що використовувалися для пошуку події, чи потенційно могли бути використані, мали бути збережені в Realtime. Як результат, кількість зчитувань була зменшена в рази, що дозволило б за день збільшити кількість потенційної аудиторії для застосунку, не виходячи за рамки безкоштовних можливостей Firebase, або ж оптимізувати витрати, коли застосунок все ж таки буде потребувати набагато більше ресурсів для більшої аудиторії.

3.3.3 Ефективне зберігання медіа в Storage

Ще одною з основних проблем застосунку було завантаження зображень до подій. Користувач також міг завантажити фото будь-якого розміру та форми при створені події. Такі зображення потребували багато місця в сховищі, відображалися не коректно в виділеній формі для зображень, а саме розтягувалися, а також вони дуже довго завантажувались з бази даних, щоб бути відображеними у списку подій та при перегляді повної інформації про подію.

Для усунення проблеми, для початку потрібно було зробити зображення квадратного формату, для збереження правильних пропорції виділеної форми в інтерфейсі. Для цього було вирішено використати такий інструмент як Image Cropper, котрий є безкоштовним рішенням, щоб обрізати передане фото в Unity. Хоч й інструмент дозволяв обрізати фото в різні форми, як от коло, зірка, але в нашому випадку було достатньо квадратного рішення.

Другим кроком було потрібно зменшити зображення до потрібних розмірів. Таким розміром на даний момент виявилося 400x400 пікселів: зображення не є настільки дрібним, щоб при перегляді на великому екрані смартфону чи планшета було видно зернистість і це візуально заважало б чи дратувало, але в той же час, зображення такого розміру вже не займає стільки місця, скільки б могло забрати зображення роздільною здатністю, наприклад, 2k.

В такий спосіб, вдалося зменшити як об’єм інформації, що весь час завантажується та вивантажується з бази даних, та об’єм, що зберігається в базі даних, так і зберегти розміри зображень, для їх коректного відображення в інтерфейсі.

Висновок

При практичній реалізації мобільного застосунку було розроблено його архітектуру, основні механіки та було виконано оптимізацію по використанню бази даних.

Клієнтська частина застосунку, розроблена для Android на Unity, відповідає за функціональний інтерфейс, обробку даних і взаємодію з сервером через Firebase. Серверна частина поділялася на 4 елементи: Firestore, Realtime Database, Cloud Storage, Authentication. Кожен використаний елемент дозволив ефективно організувати як зберігання з обробку та синхронізацію даних, оптимізуючи роботу з подіями, так і медіа та автентифікацією користувачів. У застосунку використовується Firestore і Realtime Database для зберігання та обробки даних про події й користувачів, Firebase Authentication для безпечного входу та Cloud Storage для збереження медіа, а асинхронна взаємодія між клієнтом і сервером змогла забезпечити плавну роботу застосунку, дозволяючи користувачам авторизуватися, переглядати та створювати події, завантажувати медіа без візуально помітних затримок.

Інтерфейс, розроблений у Figma та експортований у Unity. Головною ідеєю інтерфейсу, було забезпечити зручну навігацію, простоту використання та візуальну привабливість для потенційних користувачів. Сам інтерфейс був намальований як і за допомогою стандартних інструментів малювання, так й за допомогою SVG коду.

Проведена оптимізація бази даних змогла сильно підвищити продуктивність і зменшити потенційні витрати в майбутньому, значно зменшивши обсяг даних, що зберігаються, передаються, та кількість запитів зчитування, запису та видалення. Зображення обрізаються до квадратної форми за допомогою Image Cropper, вони ж зменшуються у розмірах для заощадження місця, а події фільтруються при збереженні за допомогою ID, котрий містить дату та час створення.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

В ході виконання дипломної роботи був проведений широкий аналіз стану сучасного розвитку мобільної розробки, продемонструвавши важливість ролі мобільних застосунків та їх тенденції у повсякденному житті користувачів. Також був проведений аналіз переваг і недоліків існуючих операційних систем для розробки мобільних застосунків під них, котрий показав популярність та доступність різних платформ, їх прибутковість та тенденції платформ за останні роки.

Проведено аналіз існуючих альтернативних платформ для пошуку та публікації заходів, таких як Meetup, Eventbrite, Facebook Events та AllEvents, який показав масштаб платформ, їх орієнтування, переваги та недоліки кожної з. Було виділено основні недоліки кожної з платформ, для їх врахування і виправлення в розробленому застосунку.

Було проведено порівняння технологій для розробки мобільного застосунку на основі критеріїв вибору, таких як простота використання, масштабованість, економічна ефективність та інтеграція між технологіями. Серед технологій було обрано Unity в поєднанні з C#, як платформа та мова для розробки, Figma, як інструмент з створення користувацького інтерфейсу, Firebase, як набір баз даних, для інтеграції хмарних можливостей в застосунок, а GitHub Desktop, як інструмент контролю версій проєкту.

В ході практичної реалізації, на основі визначених технологій, було розроблено повноцінний мобільний застосунок, з фільтруванням за категоріями, датою та часом, котрий реалізований під Android пристрої. Застосунок може значно спрощувати пошук та публікації заходів, й відповідно, спрощувати взаємодію між організаторами та відвідувачами подій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ферроне Х. Learning C# by Developing Games with Unity – Seventh Edition. – Birmingham : Packt Publishing, 2022. – 466 с. – ISBN 978-1-83763-687-7.
2. Майо Дж. C# Cookbook: Modern Recipes for Professional Developers. – Sebastopol : O'Reilly Media, 2021. – 862 с. – ISBN 978-1-492-09859-1.
3. SensorTower. 2025 State of Mobile Consumers: USD150 Billion Spent on Mobile Highlights. [Електронний ресурс] – Режим доступу до ресурсу: https://sensortower.com/blog/2025-state-of-mobile-consumers-usd150-billion-spent-on-mobile-highlights
4. Eventbrite. Event Statistics. [Електронний ресурс] – Режим доступу до ресурсу: https://www.eventbrite.com/blog/event-statistics-ds00/
5. Statista. Predicted Number of Smartphone Users in Ukraine. [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/1134645/predicted-number-of-smartphone-users-in-ukraine/
6. Tekrevol. Mobile App Download Statistics. [Електронний ресурс] – Режим доступу до ресурсу: https://www.tekrevol.com/blogs/mobile-app-download-statistics/
7. Biz4Solutions. Cross-Platform Mobile Development Statistics You Need to Know. [Електронний ресурс] – Режим доступу до ресурсу: https://www.biz4solutions.com/blog/cross-platform-mobile-development-statistics-you-need-to-know/
8. AsoMobile. Mobile App Market 2024 Report. [Електронний ресурс] – Режим доступу до ресурсу: https://asomobile.net/en/blog/mobile-app-market-2024-report/

9. Wezom. Найновіші технології веб-розробки 2025: як створюються сучасні сайти. [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/naynovishi-tehnologiyi-veb-rozrobki-2025-yak-stvoryuyutsya-suchasni-sayti

10. News.cn. [Електронний ресурс] – Режим доступу до ресурсу: https://english.news.cn/20240621/66928cb9ac124e7fb167d0af22bea1eb/c.html

11. Meetup. [Електронний ресурс] – Режим доступу до ресурсу: https://www.meetup.com/

12. Eventbrite. [Електронний ресурс] – Режим доступу до ресурсу: https://www.eventbrite.com/

13. AllEvents. [Електронний ресурс] – Режим доступу до ресурсу: https://allevents.in/

14. Facebook. Events. [Електронний ресурс] – Режим доступу до ресурсу: https://www.facebook.com/events

15. Figma. [Електронний ресурс] – Режим доступу до ресурсу: https://www.figma.com/

16. GitHub. Mob-Sakai / SoftMaskForUGUI. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/mob-sakai/SoftMaskForUGUI

17. Unity Asset Store. Image Cropper. [Електронний ресурс] – Режим доступу до ресурсу: https://assetstore.unity.com/packages/tools/gui/image-cropper-116650

18. Statista. Instagram: Distribution of Global Audiences by Age Group. [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/325587/instagram-global-age-group/

19. Statista. Facebook: Distribution of Global Users by Age. [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/376128/facebook-global-user-age-distribution/

20. Microsoft Learn. .NET 10 Overview. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-10/overview

21. Godot Engine. [Електронний ресурс] – Режим доступу до ресурсу: https://godotengine.org/

22. Unreal Engine. [Електронний ресурс] – Режим доступу до ресурсу: https://www.unrealengine.com/en-US

23. Cocos. Cocos2d-x. [Електронний ресурс] – Режим доступу до ресурсу: https://www.cocos.com/en/cocos2d-x

24. Adobe. Creative Cloud Plans. [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/ua/creativecloud/plans.html

25. Syntevo. SmartGit. [Електронний ресурс] – Режим доступу до ресурсу: https://www.syntevo.com/smartgit/

26. GitKraken. [Електронний ресурс] – Режим доступу до ресурсу: https://www.gitkraken.com/

27. SourceTree. [Електронний ресурс] – Режим доступу до ресурсу: https://www.sourcetreeapp.com/

28. Cybersecurity Ventures. [Електронний ресурс] – Режим доступу до ресурсу: https://cybersecurityventures.com/cybercrime-damage-costs-10-trillion-by-2025/

29. Statcounter. [Електронний ресурс] – Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/worldwide

ДОДАТКИ

Додаток А

Лістинг коду основних скриптів, пов’язаний з взаємодіями з базами даних

DatabaseActions.cs

```
using Filters.FilteringValues.Category;
using Firebase.Database;
using Firebase.Firestore;
using Firebase.Storage;
using General.EventAction;
using General.Value;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
using EventData = General.Data.EventData;

namespace General.Database
{
```

```
public class DatabaseActions : MonoBehaviour
{
    [Header("Settings")]
    [SerializeField] private bool _devDB;

    private static bool _databaseInitializationComplete;

    private static string _pathToDbStateOnUpdates = "Db state on updates";
    private static string _pathToSupportedVersionsRT = "Supported versions";
    private static string _pathToUpdateAppLinkRT = "UpdateAppLink";
    private static string _firestorePathToEvent = "EventData";
    private static string _realtimePathToEvent = "EventData";
    private static string _realtimePathToTagsList = "TagsList";
    private static string _pathToEventImageFolder = "Events";
    private static string _pathInEventImageFolderToIcon = "icon/icon";

    private static DatabaseReference _refToRealtime;
    private static FirebaseFirestore _refToFirestore;
    private static FirebaseStorage _storage;
    private static StorageReference _refToStorage;

    private static DeleteEvent _deleteEvent;
```

```
        public static bool DatabaseInitializationComplete { get =>
_databaseInitializationComplete; set => _databaseInitializationComplete = value; }


        public static string PathToDbStateOnUpdates { get =>
_pathToDbStateOnUpdates; }
        public static string PathToSupportedVersionsRT { get =>
_pathToSupportedVersionsRT; }
        public static string PathToUpdateAppLinkRT { get =>
_pathToUpdateAppLinkRT; }
        public static string FirestorePathToEvent { get => _firestorePathToEvent; }
        public static string RealtimePathToEvent { get => _realtimePathToEvent; }
        public static string RealtimePathToTagsList { get => _realtimePathToTagsList;
}
        public static string PathToEventImageFolder { get =>
_pathToEventImageFolder; }
        public static string PathInEventImageFolderToIcon { get =>
_pathInEventImageFolderToIcon; }


        public static DatabaseReference RefToRealtime { get => _refToRealtime; }
        public static FirebaseFirestore RefToFirestore { get => _refToFirestore; }
        public static StorageReference RefToStorage { get => _refToStorage; }
        public static FirebaseStorage Storage { get => _storage; }
        private void Awake()
```

```
    {
        _refToRealtime = FirebaseDatabase.DefaultInstance.RootReference;
        _refToFirestore = FirebaseFirestore.DefaultInstance;
        _storage = FirebaseStorage.DefaultInstance;

        if (_devDB)
        {
            _refToStorage = _storage.GetReferenceFromUrl("gs://mii-aktive---
dev.appspot.com/");
            Debug.Log("Використовується DEV db");
        }
        else
        {
            _refToStorage = _storage.GetReferenceFromUrl("gs://mii-
aktiv.appspot.com");
            Debug.Log("Використовується USER db");
        }

        _deleteEvent = new DeleteEvent();
        var deleteTask = DeletePastEventsAsync(10);

        var allInitTask = new List<Task>()
        {
```

```
            deleteTask
        };
        var allTaskComplete = Task.WhenAll(allInitTask);

        allTaskComplete.ContinueWith((task) =>
        {
            _databaseInitializationComplete = true;
        });
    }

    public static async Task DeletePastEventsAsync(int countDeletions)
    {
        var snapshot = await _refToRealtime.Child(RealtimePathToEvent)
            .OrderByChild("EventEndTime")
            .LimitToLast(countDeletions)
            .GetValueAsync();

        long currentDateTime =
long.Parse(EventData.ToYMDHM(DateTime.Now));

        var eventForDelete = new List<string>();
        foreach (var someEvent in snapshot.Children)
        {
```

```
            long value = (long)someEvent.Child("EventEndTime").Value;
            if (value <= currentDateTime)
                eventForDelete.Add(someEvent.Key);
        }

        if (eventForDelete.Count > 0)
            await _deleteEvent.FullDeleteAsync(eventForDelete);
    }
    public static async Task<byte[]> GetBitesArray(string pathInStorage)
    {
        StorageReference imageRef = _refToStorage.Child(pathInStorage +
ActionWithTexture2D.TextureExpansion);

        byte[] imageBytes = null;
        try
        {
            imageBytes = await imageRef.GetBytesAsync(long.MaxValue);
        }
        catch
        {
            Debug.LogWarning("Картинка не завантажилася, не отримано потік
байтів");
            return null;
```

```
        }

        return imageBytes;
    }
    public static async Task<List<string>> GetEvents(int elementCount, string
startAfterEventWithId = "")
    {
        if (elementCount < 0)
            elementCount = 0;

        List<string> collection = new List<string>();

        if (elementCount != 0)
        {
            if (startAfterEventWithId == "")
            {
                var query = _refToRealtime.Child(_realtimePathToEvent)
                    .LimitToLast(elementCount);

                DataSnapshot snapshot = await query.GetValueAsync();

                foreach (var item in snapshot.Children)
                    collection.Add(item.Key);
```

```
        }
        else
        {
            var snapshot = await _refToRealtime.Child(_realtimePathToEvent)
              .OrderByKey()
              .EndAt(startAfterEventWithId)
              .LimitToLast(elementCount + 1)
              .GetValueAsync();

            foreach (var item in snapshot.Children)
            {
                if (item.Key == snapshot.Children.Last().Key)
                    break;

                collection.Add(item.Key);
            }
        }
    }

    if (collection == null || collection.Count == 0)
        Debug.LogWarning("Повернута колекція розміром у 0 елементів, або рівна null. Можливо некоректно вказано стартовий елемент.");
```

```
        return collection;
    }
    public static async Task<List<string>>
GetEventsWithTags(List<CategoryObject> tags, int elementCount, string
startAfterEventWithId = "")
    {
        if (elementCount < 0)
            elementCount = 0;

        List<string> collection = new List<string>();

        if (tags.Count == 0 || tags == null)
        {
            collection = await GetEvents(elementCount, startAfterEventWithId);
            Debug.LogWarning("Фільтри не були введені. Повернуто колекцію без
врахування фільтрів");
            return collection;
        }

        if (elementCount != 0)
        {
            var taskArray = new Task<DataSnapshot>[tags.Count];
```

```
if (startAfterEventWithId == "")
{
    for (int i = 0; i < tags.Count; i++)
        taskArray[i] = _refToRealtime.Child(_realtimePathToTagsList)
            .Child(tags[i].ID.ToString())
            .LimitToLast(elementCount)
            .GetValueAsync();
}
else
{
    for (int i = 0; i < tags.Count; i++)
        taskArray[i] = _refToRealtime.Child(_realtimePathToTagsList)
            .Child(tags[i].ID.ToString())
            .OrderByKey()
            .EndAt(startAfterEventWithId)
            .LimitToLast(elementCount + 1)
            .GetValueAsync();
}
await Task.WhenAll(taskArray);

var allEventsDataSnapshot = new List<DataSnapshot>();
foreach (var task in taskArray)
```

```
                allEventsDataSnapshot =
allEventsDataSnapshot.Concat(task.Result.Children).ToList();

            var allEventId = new List<string>();
            foreach (var child in allEventsDataSnapshot)
                allEventId.Add(child.Key);
            allEventId = allEventId.Distinct().ToList();
            allEventId.Sort();

            if (startAfterEventWithId != "")
                allEventId.Remove(startAfterEventWithId);

            int index = elementCount;
            if (index > allEventId.Count)
                index = 0;
            else
                index = allEventId.Count - elementCount;
            for (int i = 0; i < elementCount && i < allEventId.Count; i++, index++)
                collection.Add(allEventId.ElementAt(index));
        }

        if (collection == null)
        {
```

```
            Debug.LogWarning("Колекція рівна null. Повертаємо пусту колекцію. Можливо некоректно вказано стартовий елемент.");
            return new List<string>();
        }

        return collection;
    }
    public static async Task<ImageValues> GetImageValues(string pathInStorage)
    {
        StorageReference jsonRef = _refToStorage.Child(pathInStorage + ".json");

        ImageValues imageValues = null;
        try
        {
            Stream stream = await jsonRef.GetStreamAsync();

            await Task.Run(() =>
            {
                using (StreamReader reader = new StreamReader(stream))
                {
                    string jsonString = reader.ReadToEnd();
                    imageValues = JsonUtility.FromJson<ImageValues>(jsonString);
                }
```

```
        });
    }
    catch
    {
        Debug.LogWarning("Помилка читання файлу з Firebase Storage: ");
        return null;
    }

    return imageValues;
}
public static void SaveEvent(EventData eventData, string eventID)
{
    // Запис до Firestore
    var dataForFireStore = new Dictionary<string, object>()
    {
        {"ShortInfo", eventData.ShortInfo},
        {"FullInfo", eventData.FullInfo},
        {"TelegramLink", eventData.TelegramLink},
        {"RegisterFormLink", eventData.RegisterFormLink},
        {"LinkToAvatarInStorage", eventData.LinkToAvatarInStorage},
    };
```

```
        var document = _refToFirestore.Collection(FirestorePathToEvent)
            .Document(eventID);
        document.SetAsync(dataForFireStore);

        // Запис до Realtime
        var dataForRealtime = new Dictionary<string, object>()
        {
            {"Name", eventData.Name},
            {"DateAndTimeOfCreationEvent",
eventData.DateAndTimeOfCreationEvent},
            {"EventEndTime", eventData.EventEndTime},
            {"Tags", eventData.Tags},
        };

        for (int i = 0; i < eventData.Tags.Length; i++)
        {
            _refToRealtime.Child(_realtimePathToTagsList)

.Child(eventData.Tags[i].ToString()).Child(document.Id).SetValueAsync("");
        }
```

```
_refToRealtime.Child(_realtimePathToEvent).Child(document.Id).SetValueAsync(dataForRealtime);
    }
    public static async Task UploadImageAsync(Texture2D texture2D, string pathToSave)
    {
        StorageReference imageRef = _refToStorage.Child(pathToSave + ActionWithTexture2D.TextureExpansion);
        StorageReference jsonRef = _refToStorage.Child(pathToSave + ".json");

        // Save Image
        var imgTask = imageRef.PutBytesAsync(ActionWithTexture2D.CompressTextureToBytes(texture2D));
        await imgTask;
        if (imgTask.IsFaulted || imgTask.IsCanceled)
            Debug.LogError(imgTask.Exception.ToString());
        else
            Debug.Log("Img upload successful!");

        // Save JSON
        var image = new ImageValues(texture2D);
        string jsonString = JsonConvert.SerializeObject(image, Formatting.Indented,
```

```
            new JsonSerializerSettings()
            {
                ReferenceLoopHandling = ReferenceLoopHandling.Ignore
            });

            var stream = new MemoryStream(Encoding.UTF8.GetBytes(jsonString));

            var JsonTask = jsonRef.PutStreamAsync(stream);
            await JsonTask;

            if (JsonTask.IsFaulted || JsonTask.IsCanceled)
                Debug.LogError(JsonTask.Exception.ToString());
            else
                Debug.Log("JSON upload successful!");
        }
    }
}
```

DatabaseAuth.cs

```
using Firebase;
using Firebase.Auth;
using System;
using System.Threading.Tasks;
```

```
using UnityEngine;

namespace General.Database
{
    public class DatabaseAuth : MonoBehaviour
    {
        private FirebaseAuth _auth;

        public static FirebaseAuth Auth { get; private set; }

        public static Action SignInSuccessful;
        public static Action SignOutSuccessful;

        private void Awake()
        {
            _auth = FirebaseAuth.DefaultInstance;

            Auth = _auth;

            if (Auth.CurrentUser != null)
                _ = CheckCurrentUserOnDisable();
        }

        public static async Task<bool> CheckCurrentUserOnDisable()
        {
            try
            {
                await Auth.CurrentUser.ReloadAsync();
                return true;
            }
```

```
        catch (Exception)
        {
            Auth.SignOut();
            return false;
        }
    }

    private void HandleSignInError(FirebaseException ex)
    {
        switch (ex.ErrorCode)
        {
            case (int)AuthError.MissingEmail:
                AuthFields.LoginStatus.text = "Не вказано електронну пошту";
                break;

            case (int)AuthError.MissingPassword:
                AuthFields.LoginStatus.text = "Не вказано пароль";
                break;

            case (int)AuthError.InvalidEmail:
                AuthFields.LoginStatus.text = "Некоректна адреса електронної
пошти";
                break;

            case (int)AuthError.WrongPassword:
                AuthFields.LoginStatus.text = "Неправильний пароль";
                break;

            case (int)AuthError.UserNotFound:
```

```
                AuthFields.LoginStatus.text = "Ця електронна пошта не
зареєстрована";
                break;

            case (int)AuthError.NetworkRequestFailed:
                AuthFields.LoginStatus.text = "Проблема підключення до сервера.
Перевірте інтернет зв'язок та спробуйте ще раз";
                break;

            case (int)AuthError.TooManyRequests:
                AuthFields.LoginStatus.text = "Багато спроб входу за короткий час.
Спробуйте пізніше";
                break;

            case (int)AuthError.OperationNotAllowed:
                AuthFields.LoginStatus.text = "Данна операція поки що недоступна.
Спробуйте пізніше";
                break;

            case (int)AuthError.UserDisabled:
                AuthFields.LoginStatus.text = "Данний обліковий запис заблоковано
адміністратором";
                break;

            default:
                AuthFields.LoginStatus.text = "Непередбачена помилка: " +
ex.Message;
                break;
        }
    }
    public async void SignIn()
    {
```

```
            try
            {
                AuthResult authResult = await
_auth.SignInWithEmailAndPasswordAsync(AuthFields.Email.text,
AuthFields.Password.text);
                AuthFields.LoginStatus.color = Color.blue;
                AuthFields.LoginStatus.text = "Успішно ввійшли в акаунт: " +
authResult.User.Email;


                SignInSuccessful?.Invoke();
            }
            catch (Exception ex)
            {
                AuthFields.LoginStatus.color = Color.red;
                if (ex.InnerException is FirebaseException firebaseException)
                    HandleSignInError(firebaseException);
                else
                    AuthFields.LoginStatus.text = "Помилка: " + ex.Message;
            }


        }
        public void SignOut()
        {
            _auth.SignOut();
            AuthFields.LoginStatus.color = Color.black;
            AuthFields.LoginStatus.text = "Ви вийшли з акаунту";


            SignOutSuccessful?.Invoke();
        }
    }
}
```