БАКАЛАВРСЬКА РОБОТА

БР. ІП - 36.00.00.000 ПЗ

Група ІП-21-2

Тріщук Ніколау-Аугушто

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Тріщук Ніколау-Аугушто Миколайович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Побудова програмного сервісу адміністрування подій користувача

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього рівня Тріщук Н.-А.М.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Храбатин Роман Ігорович, к.ф.-м.н., доцент

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

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Тріщуку Ніколау-Аугушто Миколайовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Побудова програмного сервісу адміністрування подій користувача ”

керівник проекту (роботи) Храбатин Роман Ігорович

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області розробки програмних сервісів організації та адміністрування подій
2. Специфікація вимог до системи та програмні залежності проекту
3. Проектування архітектури програмного сервісу адміністрування подій користувача
4. Розробка діаграм послідовності
5. Програмна реалізація програмного сервісу адміністрування подій користувача

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Залежності проекту (рис. 1.1)
2. Конфігурація FireBase (рис. 1.2)
3. Залежності розробки проекту (рис. 1.3)
4. Інтерфейс платформи Bizzabo (рис. 1.4)
5. Віртуальна конференція засобами платформи Hopin (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області розробки програмних сервісів організації та адміністрування подій	01.05.2025	виконано
2	Специфікація вимог до системи та програмні залежності проекту	10.05.2025	виконано
3	Проектування архітектури програмного сервісу адміністрування подій користувача	20.05.2025	виконано
4	Розробка діаграм послідовності	29.05.2025	виконано
5	Програмна реалізація програмного сервісу адміністрування подій користувача	03.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 76 сторінок, 33 рисунки, список використаних джерел із 33 найменуваннями, 1 додаток.

Мета роботи: розробці програмного сервісу для адміністрування подій користувача, який дозволить забезпечити зручне управління структурою події.

Об'єкт дослідження: процес цифрового адміністрування та організації подій

Предмет дослідження: методи та засоби розробки програмних застосунків для управління подіями з урахуванням вимог користувачів, структури даних і сценаріїв взаємодії

Результати дослідження: **з**апропоновано структуровану архітектуру програмного сервісу для адміністрування подій з урахуванням інтеграції функцій керування подіями.

В першому розділі проаналізовано передумови, вимоги та наявні системи для адміністрування подій, що дозволило сформувати основу функціонального бачення розробки

В другому розділі спроєктовано архітектуру системи, визначено її логіку, ключові компоненти інтерфейсу та взаємодії.

В третьому розділі реалізовано інтерфейс, функціонал, збереження та обробку даних, а також проведено повноцінне тестування створеного сервісу.

Висновок: побудовано архітектурну модель, яка включає модулі керування подіями, реєстрації учасників, генерації QR-кодів та управління базою даних

КЛЮЧОВІ СЛОВА: АДМІНІСТРУВАННЯ ПОДІЙ, ПРОГРАМНИЙ СЕРВІС, ОРГАНІЗАЦІЯ ЗАХОДІВ, ВЕБДОДАТОК, QR-КОДИ, БАЗА ДАНИХ, УПРАВЛІННЯ КОРИСТУВАЧАМИ, ІНФОРМАЦІЙНА СИСТЕМА

ANNOTATION

The Bachelor's thesis contains 76 pages, 33 figures, a list of 33 references, and 1 appendix.

Purpose of the work: Development of a software service for administering user events, which will allow for convenient management of the event structure.

Object of research: The process of digital administration and organization of events.

Subject of research: Methods and tools for developing software applications for event management, considering user requirements, data structure, and interaction scenarios.

Research results: A structured architecture for a software service for event administration has been proposed, taking into account the integration of event management functions.

The first chapter analyzed the prerequisites, requirements, and existing systems for event administration, which allowed for the formation of the basis of the functional vision for the development.

The second chapter designed the system's architecture, defined its logic, key interface components, and interactions.

The third chapter implemented the interface, functionality, data storage and processing, and conducted comprehensive testing of the created service.

Conclusion: An architectural model has been built that includes modules for event management, participant registration, QR code generation, and database management.

KEYWORDS: EVENT ADMINISTRATION, SOFTWARE SERVICE, EVENT ORGANIZATION, WEB APPLICATION, QR CODES, DATABASE, USER MANAGEMENT, INFORMATION SYSTEM

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

BaaS – Бекенд як сервіс (Backend as a Service)

DB – База даних (Database)

DFD – Діаграма потоку даних (Data Flow Diagram)

ID – Ідентифікатор (Identifier)

NFC – Комунікація ближнього поля (Near Field Communication)

QA – Забезпечення якості (Quality Assurance)

QR – Швидкий відгук (Quick Response)

UAT – Тестування прийняття користувачем (User Acceptance Testing)

ВСТУП

У сучасному цифровому суспільстві організація та адміністрування подій набуває нового значення в контексті автоматизації, мобільності та інтеграції з іншими цифровими сервісами. Зростання кількості онлайн і офлайн заходів, які потребують ефективного планування, керування реєстраціями, комунікації з учасниками та збору статистичних даних, висуває нові вимоги до інструментів, які підтримують ці процеси. Існуючі програмні платформи, як правило, мають закриту архітектуру, високий поріг входу або обмежену можливість адаптації під специфічні потреби організаторів. Це створює попит на розробку спеціалізованих рішень, орієнтованих на гнучкість, простоту використання та розширюваність функціоналу. Тому тема побудови програмного сервісу адміністрування подій користувача є актуальною та відповідає сучасним тенденціям у галузі розробки інформаційних систем.

Актуальність теми

Актуальність обраної теми визначається не лише потребами ринку, але й наявністю простору для наукового пошуку у сфері проєктування програмних сервісів з урахуванням вимог зручності, масштабованості та безпеки. Результати роботи можуть бути використані як основа для впровадження в реальні сценарії організації подій або подальшого розвитку функціоналу відповідно до нових потреб користувачів.

Сучасні події, незалежно від масштабу та форми проведення, потребують високого рівня організації та технічного супроводу. У зв’язку з активною цифровізацією різних сфер життя зростає потреба в ефективних програмних інструментах, які дозволяють автоматизувати процеси адміністрування заходів — від реєстрації користувачів до збору статистичних даних та обробки зворотного зв’язку.

Мета цієї дипломної роботи полягає в розробці програмного сервісу для адміністрування подій користувача, який дозволить забезпечити зручне управління структурою події, взаємодію з учасниками, генерацію та обробку QR-кодів, ведення бази даних подій, а також забезпечення багаторівневого доступу до інформації.

Завдання дослідження

- Проаналізувати наявні системи організації подій і виділити їх ключові функціональні можливості.

- Визначити вимоги до розроблюваного програмного забезпечення.

- Розробити архітектуру програмного сервісу на основі сценаріїв використання.

- Реалізувати інтерфейс та функціонал управління подіями.

- Провести тестування системи за різними типами тестів.

Об’єкт дослідження - процес цифрового адміністрування та організації подій.

Предмет дослідження - методи та засоби розробки програмних застосунків для управління подіями з урахуванням вимог користувачів, структури даних і сценаріїв взаємодії.

Методи дослідження

Аналіз і порівняння систем, структурне й об'єктно-орієнтоване проектування, моделювання архітектури, розробка за методологією Agile, юзабіліті-тестування та функціональне тестування системи.

Наукова новизна

Запропоновано структуровану архітектуру програмного сервісу для адміністрування подій з урахуванням інтеграції функцій керування подіями, візуалізації за допомогою QR-кодів і підтримки багаторівневої структури доступу до даних.

Практичне застосування

Розроблений застосунок може бути використаний організаторами подій, освітніми закладами, компаніями та адміністративними структурами для автоматизації управління заходами.

У роботі було проведено аналіз предметної області та існуючих рішень, визначено технічні вимоги до системи, спроєктовано архітектуру сервісу та реалізовано його функціональні модулі. Також було проведено повноцінне тестування програмного продукту з метою перевірки відповідності функціоналу очікуванням користувачів.

Бакалаврська робота містить 76 сторінок, 33 рисунки, 3 розділи список використаних джерел із 33 найменуваннями, 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ РОЗРОБКИ ПРОГРАМНИХ СЕРВІСІВ ОРГАНІЗАЦІЇ ТА АДМІНІСТРУВАННЯ ПОДІЙ

1.1. Передумови розробки програмного додатку управління заходами та подіями

У великих організаціях щоденно проводиться значна кількість заходів. Традиційні методи інформування про події, такі як сторінки подій на веб-сайтах, друковані матеріали (флаєри, буклети) та усне спілкування, часто є неефективними. Процеси реєстрації та фіксації відвідуваності також характеризуються надмірним введенням даних. Зокрема, користувачі вимушені багаторазово вводити особисту інформацію при реєстрації на нові заходи, а також повторно вводити дані на вході до заходу, використовуючи паперові списки або електронні пристрої. Ці процедури є тривалими та обтяжливими, що знижує ефективність управління подіями. Цей проєкт спрямований на розробку системи, що усуває зазначені надмірності та оптимізує процеси оголошення, реєстрації та відвідування заходів.

1.1.1. Методологія розробки

З огляду на домінування мобільних платформ Android та iOS на світовому ринку смартфонів, розробка крос-платформного мобільного додатку є критично важливою для забезпечення широкого охоплення користувачів. Традиційна розробка нативних додатків для кожної платформи окремо є ресурсоємною та неефективною. З метою мінімізації складності та прискорення процесу розробки, було обрано React Native як фреймворк для крос-платформної розробки. React Native дозволяє розробляти додатки для обох платформ, використовуючи єдину кодову базу, написану на JSX та React.js, що значно підвищує ефективність розробки.

1.1.2. Мета проєкту

Основна мета цього проєкту полягає у створенні мобільного додатку, що істотно прискорить та спростить процеси оголошення, реєстрації та відвідування будь-яких заходів, що пропонуються організаціями. Це включає оптимізацію взаємодії користувача з системою управління подіями, мінімізацію ручного введення даних та покращення загального досвіду участі у заходах.

1.1.3. Аналіз існуючої системи

Для демонстрації функціональності системи було розглянуто типовий сценарій управління подіями, що часто зустрічається в освітніх та інших великих організаціях. Наприклад, події публікуються на веб-сайтах відповідних відділів, а централізований веб-сайт агрегує цю інформацію, надаючи хронологічно організований перелік подій. Процес реєстрації на певну подію або захід передбачає перехід потенційного користувача на сторінку відділу для введення особистих даних. У день заходу, для фіксації відвідуваності, учасники зазвичай пред'являють ідентифікаційну картку та або вписують дані у паперовий журнал, або вводять їх у цифрову форму на планшеті.

Основними недоліками існуючих системи є:

- Надмірне введення даних - користувачі змушені неодноразово вводити свої дані для реєстрації та відвідування різних заходів.

- Низька швидкість обробки - ручне введення даних під час реєстрації та на вході до заходу значно уповільнює процес, створюючи черги та "вузькі місця".

Ці недоліки підкреслюють необхідність розробки нової системи, яка автоматизує та оптимізує зазначені процеси, забезпечуючи більш ефективне управління подіями.

1.2. Специфікація вимог до системи та програмні залежності проекту

Цей документ деталізує вимоги до апаратного та програмного забезпечення, необхідні для розробки та функціонування системи.

1.2.1. Апаратні вимоги

Для розробки під Android необхідно використовувати персональний комп'ютер (ПК) з мінімально 8 ГБ оперативної пам'яті, 20 ГБ вільного дискового простору та процесором з тактовою частотою 2 ГГц. Розробка під Android та iOS вимагає комп'ютера Mac з аналогічними характеристиками: мінімум 8 ГБ оперативної пам'яті, 20 ГБ вільного дискового простору та процесором 2 ГГц. Для тестування мобільних додатків необхідний пристрій Android з мінімум 1 ГБ оперативної пам'яті або iPhone 5 чи новішої моделі.

1.2.2. Програмні вимоги

Використовується Firebase як сервіс бекенду, що забезпечує функціональність автентифікації користувачів та базу даних реального часу. Це рішення значно прискорює процес розробки мобільних додатків, оскільки Firebase бере на себе управління інфраструктурою.

Розробка здійснюється з використанням фреймворку React Native.

Для розробки використовується інтегроване середовище розробки (IDE) VS Code, а також Node.js та його менеджер пакетів npm. Для розробки під Android необхідний Android SDK. Для тестування використовується Jest.

1.2.3. Програмні залежності для функціоналу

Залежності – це набір фреймворків та бібліотек, необхідних для коректного функціонування додатку.

Наступні залежності необхідні для забезпечення роботи виробничої збірки додатку:

```
{
    "dependencies": {
        "jsc-android": "224109.x.x",
        "mobs": "^5.0.3",
        "mobs-react": "^5.2.3",
        "moment": "^2.22.2",
        "native-base": "^2.7.2",
        "react": "16.0.0-beta.5",
        "react-native": "0.49.3",
        "react-native-camera": "^1.2.0",
        "react-native-firebase": "^2.2.3",
        "react-native-htmlview": "^0.13.0",
        "react-native-loading-spinner-overlay": "^0.5.2",
        "react-native-qrcode": "0.2.7",
        "react-navigation": "^3.12.0",
        "recompose": "^0.30.0",
        "tcomb-form-native": "^0.6.15"
    }
}
```

Рисунок 1.1 - Залежності проекту

React (React.js) є високоефективною бібліотекою JavaScript, розробленою Facebook, призначеною для побудови односторінкових веб-додатків. Вона оптимізує маніпуляції з DOM (Document Object Model), мінімізуючи операції запису за рахунок використання віртуального DOM, що дозволяє застосовувати лише мінімально необхідні зміни. Ця оптимізація досягається функцією рендерингу, яка обчислює мінімальні зміни.

React Native — це кросплатформний фреймворк для розробки мобільних додатків. На відміну від гібридних фреймворків (наприклад, Ionic, Cordova), які використовують вбудовану функціональність браузера, React Native рендерить додатки за допомогою нативних компонентів, що забезпечує підвищену продуктивність.

JSC Android.

Фреймворк React Native на платформі Android використовує застарілу версію рушія JavaScript JSC (JavaScript Core). Для інтеграції актуальних

оновлень у фреймворк React Native використовується ця бібліотека як залежність.

MobX.

Зі зростанням кодової бази додатку React Native виникає потреба в рішенні для управління станом додатку протягом його життєвого циклу. MobX є спрощеною альтернативою для управління станом у React.

React Navigation.

Ця бібліотека використовується для навігації між сторінками в додатку React Native. Вона є повністю настроюваною та розширюваною. У поточному додатку ця бібліотека керує навігацією між різними сторінками.

React Native Camera.

Ця універсальна нативна бібліотека камери для React Native інтегрована в додаток для сканування QR-кодів під час відвідування подій.

React Native QR Code.

Бібліотека генератора QR-кодів, що дозволяє створювати скановані QR-коди з заданого рядка. Ці бібліотеки надають компоненти, які приймають рядок даних (наприклад, URL, текст, ідентифікатор події) і перетворюють його на візуальне зображення QR-коду. У поточному додатку ця бібліотека застосовується для генерації QR-коду події на основі її ідентифікатора.

Firebase.

Firebase є платформою для розробки мобільних додатків, що надає функціонал для аутентифікації та базу даних у реальному часі. Він бере на себе управління інфраструктурою, що прискорює процес розробки мобільних додатків.

React Native Firebase.

Бібліотека React Native Firebase надає API для взаємодії з backend Firebase, що значно спрощує написання коду, який вимагає комунікації з сервером.

```
import RNFirebase from 'react-native-firebase' // Імпорт бібліотеки RNFirebase

const firebase = RNFirebase.initializeApp({ // Ініціалізація Firebase додатку
    apiKey: "YOUR_API_KEY", // Ключ API для доступу до сервісів Firebase. (Зацензурова
    authDomain: "csusbevents.firebaseapp.com", // Домен для автентифікації Firebase.
    databaseURL: "https://csusbevents.firebaseio.com", // URL бази даних Firebase.
    projectId: "csusbevents", // Ідентифікатор проекту Firebase.
    storageBucket: "csusbevents.appspot.com", // Корзина для зберігання файлів Fireba
    messagingSenderId: "YOUR_MESSAGING_SENDER_ID" // Ідентифікатор відправника для Fi
});

export default firebase; // Експорт ініціалізованого об'єкта Firebase
```

Рисунок 1.2 – Конфігурація FireBase

Наступні залежності необхідні для забезпечення роботи розробницької збірки додатку.

```
{
    "devDependencies": {
        "babel-jest": "23.4.2",
        "babel-plugin-transform-decorators-legacy": "^1.3.5",
        "babel-preset-react-native": "3.0.0",
        "cary": "^0.0.1",
        "jest": "23.5.0",
        "react-native-config": "^0.11.5",
        "react-test-renderer": "16.0.0-beta.5"
    }
}
```

Рисунок 1.3 – Залежності розробки проекту

Babel Preset React Native.

Розробники часто використовують сучасний синтаксис JavaScript, проте якщо фреймворк базується на старішій версії мови, це може призвести до помилок. Babel автоматично трансформує код, написаний у будь-якій версії JavaScript, у синтаксис, зрозумілий більшості JavaScript-рушіїв, усуваючи необхідність ручного перетворення.

Babel Plugin Transform Decorators Legacy.

Декоратори розширюють функціональність об'єктів і розміщуються безпосередньо перед оголошенням класів, функцій або змінних, зазвичай починаючись із символу '@'. Оскільки ES7 декоратори за замовчуванням не

підтримуються в React Native, цей плагін необхідний для їхньої коректної трансформації. Зокрема, MobX активно використовує декоратори.

React Test Renderer.

Ця бібліотека трансформує компоненти React у чисті об'єкти JavaScript, що є особливо корисним для створення знімків компонента, які використовуються в процесі модульного тестування додатків.

Jest.

Jest є високопродуктивним фреймворком для модульного тестування JavaScript-додатків. Його функція тестування знімків дозволяє верифікувати рендеринг компонента шляхом створення знімка та JSON-дерева безпосередньо з коду компонента.

Babel Jest.

Це плагін, який забезпечує підтримку синтаксису ES6 та ES7 при виконанні тестів за допомогою Jest.

1.3. Аналіз існуючих систем організації та адміністрування подій і заходів

Аналіз існуючих програмних додатків для організації та адміністрування подій і заходів виявляє широкий спектр рішень, які відрізняються за функціоналом, цільовою аудиторією, масштабом і ціновою політикою. Ці системи призначені для автоматизації та спрощення всіх етапів життєвого циклу події – від початкового планування до післяподієвої аналітики.

1.3.1. Платформа Bizzabo

Bizzabo позиціонується як "Операційна система для подій" і пропонує інтегровану платформу для управління подіями, з акцентом на досвід учасників та аналітику. Особливо сильний у гібридних та віртуальних подіях.

Основний функціонал:

- Потужні інструменти для трансляцій, віртуальних кімнат, інтерактивних сесій, Q&A, опитувань.

- Гнучкі форми реєстрації, можливості для сегментації учасників.

- Фокус на нетворкінгу, персоналізованому контенті, інтерактивності.

- Управління даними учасників та потенційних клієнтів.

- Email-маркетинг

- Детальні звіти про залученість, джерела реєстрацій, ефективність кампаній.

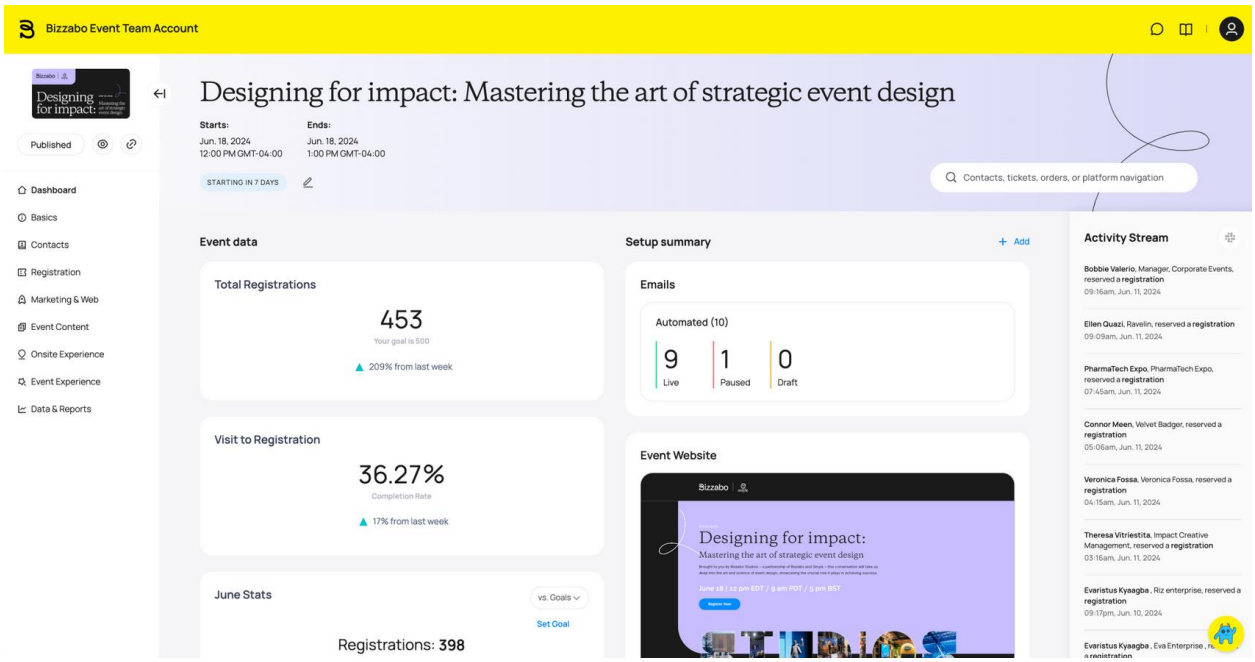


Рисунок 1.4 – Інтерфейс платформи Bizzabo

Цільовою аудиторією є компанії, які проводять корпоративні заходи, маркетингові відділи, організатори конференцій та виставок, особливо ті, хто активно використовує гібридний формат.

Потужний інструментарій для віртуальних та гібридних подій, акцент на досвіді учасників, сильна аналітика, інтуїтивно зрозумілий інтерфейс.

1.3.2. Платформа адміністрування подій Hopin

Hopin швидко здобув популярність під час пандемії як провідна платформа для віртуальних подій. Хоча зараз він розширив свої пропозиції, включаючи гібридні та офлайн-можливості, його сильні сторони все ще лежать у віртуальному просторі. Наразі Hopin трансформується і розширює свій портфель продуктів.

Основний функціонал наступний:

- Віртуальні зали - різні типи віртуальних приміщень (сцена, сесії, нетворкінг, експо).

- Відео- та аудіо трансляції з високою якістю потокового мовлення.

- Нетворкінг - функції для випадкових зустрічей, персоналізований нетворкінг.

- Чати, Q&A, опитування, білі дошки.

- Віртуальні виставкові стенди.

- Базова реєстрація та аналітика.

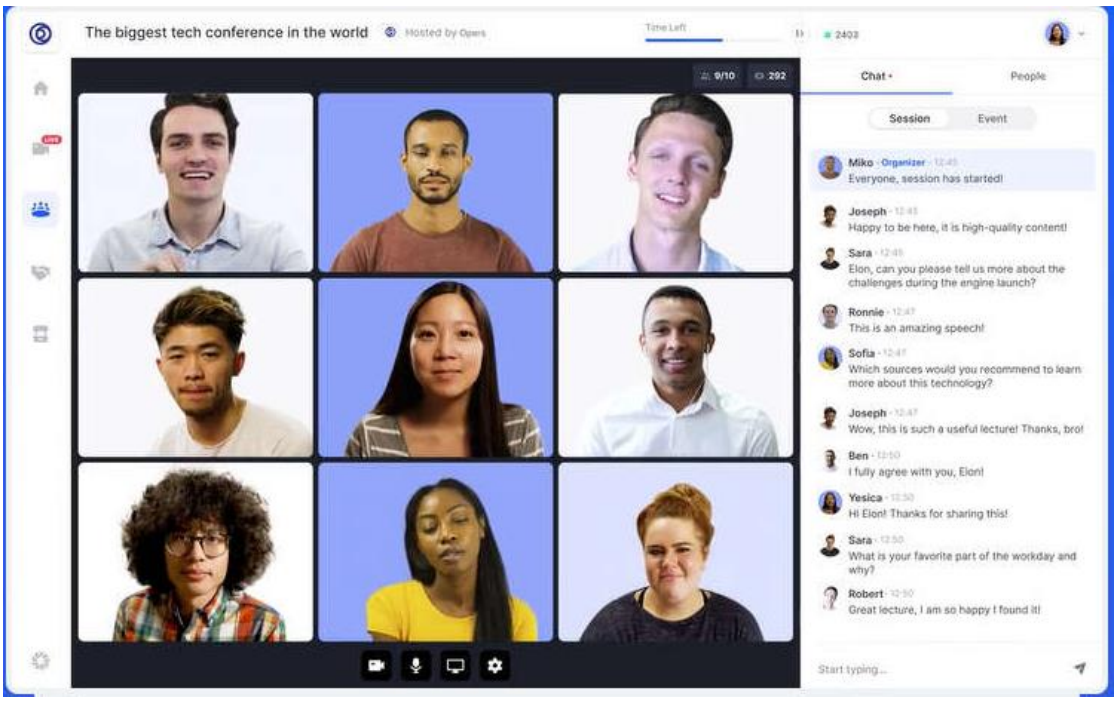


Рисунок 1.5 – Віртуальна конференція засобами платформи Hopin

Цільовою аудиторією є організатори, що проводять віртуальні та гібридні конференції, вебінари, онлайн-фестивалі.

Основною перевагою є дуже сильний у віртуальному форматі, інтуїтивно зрозумілий для учасників, велика кількість інтерактивних функцій.

1.3.3. Додаток для організації подій EventMobi

EventMobi є провідною платформою для подій, яка надає мобільні додатки для учасників та інструменти для організаторів. Відома своєю гнучкістю та налаштовуваністю.

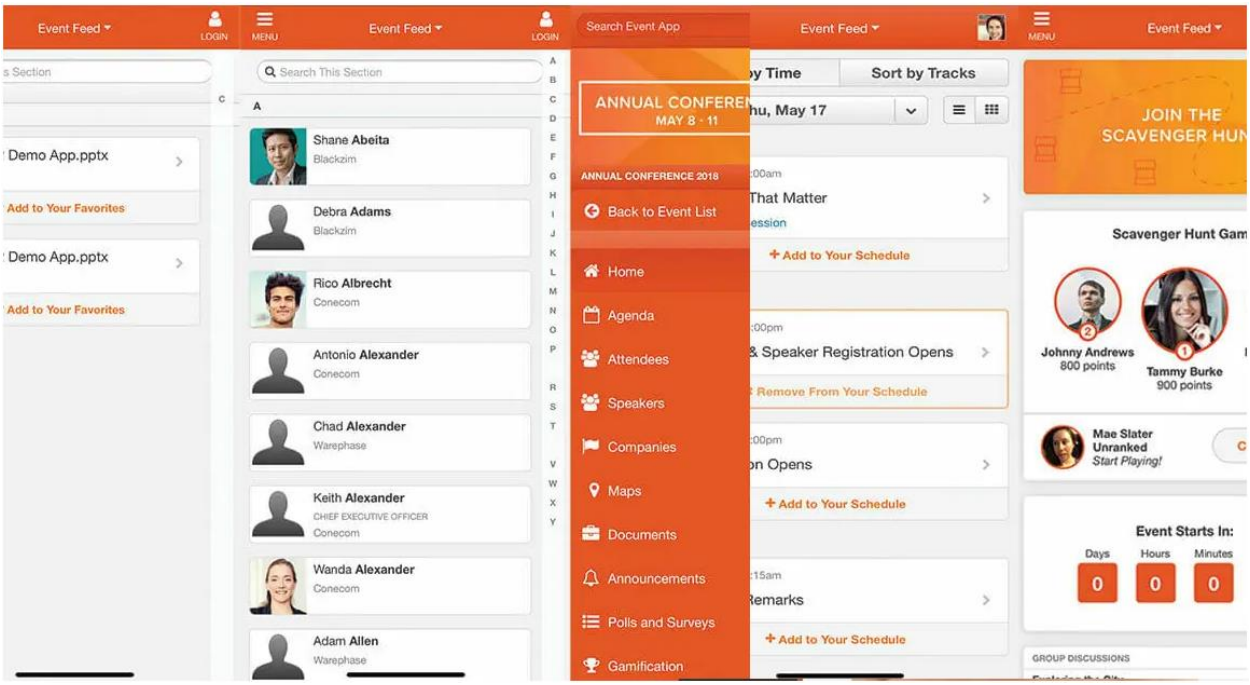


Рисунок 1.6 – Вигляд додатку EventMobi

Ключові функції (для учасників):

1. Повний цикл взаємодії від реєстрації до післяподієвих опитувань.

2. Розширений нетворкінг, функції для пошуку та зв'язку з іншими учасниками, приватні чати, групові обговорення.

3. Персоналізовані рекомендації на основі інтересів учасника.

4. Ігрові елементи для заохочення активності (наприклад, збір балів за відвідування сесій або взаємодію).

5. Доступ до презентацій, відео, матеріалів події.

Ключові функції (для організаторів):

1. Управління вмістом, легке оновлення розкладу, спікерів, спонсорів.

2. Розсилка повідомлень, сегментовані пуш-сповіщення.

3. Детальна статистика використання додатку та поведінки учасників.

4. Інструменти check-in та реєстрація на місці.

EventMobi надає організаторам високий рівень контролю над досвідом учасників та забезпечує потужні інструменти для залучення.

1.3.4. Платформа управління подіями Whova

Платформа Whova є однією з провідних сучасних систем для адміністрування та організації подій, яка широко використовується в академічному, професійному та корпоративному середовищі. Вона пропонує широкий спектр функцій, що охоплюють усі етапи підготовки та реалізації заходів — від планування програми до аналітики участі.

Whova вирізняється інтуїтивно зрозумілим інтерфейсом та високим рівнем інтерактивності. Однією з ключових переваг платформи є можливість створення детальної програми заходу, у якій кожна сесія супроводжується описом, зазначенням спікерів, додатковими матеріалами, посиланнями на трансляції або відеозаписи. Учасники мають змогу самостійно формувати власний розклад, що значно підвищує зручність користування та дозволяє персоналізувати досвід участі.

Особливу увагу в Whova приділено функціональності взаємодії між учасниками. Нетворкінг підтримується через можливості перегляду профілів, особистого спілкування в чаті, організації віртуальних або фізичних зустрічей. Окремим елементом є система гейміфікації — учасники

отримують бали за активність, відповіді на опитування, участь у дискусіях тощо, що стимулює залучення до події.

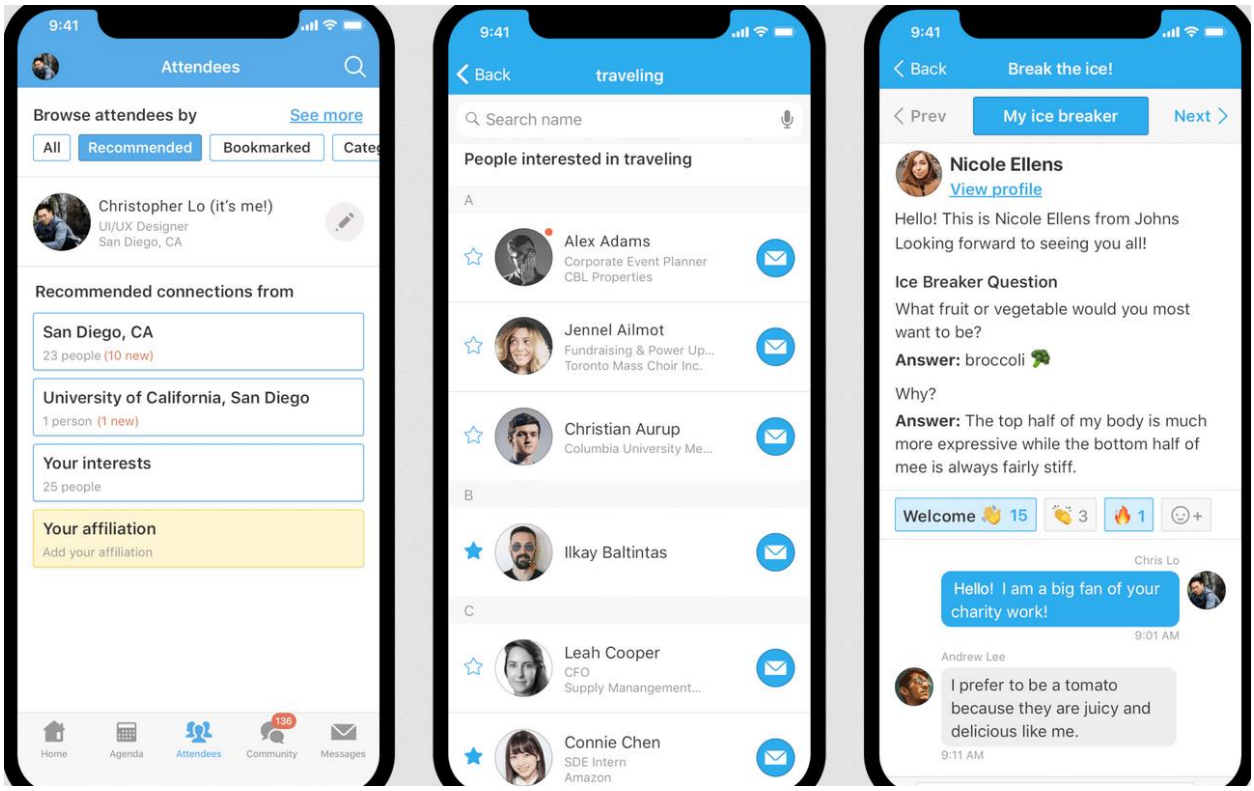


Рисунок 1.7 – Інтерфейс платформи Whova

У контексті віртуальних і гібридних заходів Whova пропонує інтеграцію з відеоплатформами (Zoom, YouTube, Vimeo), а також функціонал для створення віртуальних стендів, що може бути корисним для виставкових заходів або наукових постерних сесій. Крім того, організатори можуть проводити опитування, сесії запитань-відповідей та голосування безпосередньо під час доповідей, що значно підвищує динаміку спілкування та зворотний зв'язок.

Ще одним суттєвим плюсом платформи є розширені аналітичні інструменти. Організатори мають доступ до даних про активність учасників, відвідування сесій, завантаження матеріалів, що дозволяє оцінити ефективність події та зробити висновки для майбутніх заходів.

Whova активно використовується для проведення конференцій, семінарів, наукових симпозіумів, а також внутрішніх корпоративних подій. Універсальність платформи, її масштабованість та гнучкість дозволяють адаптувати функціонал до потреб як невеликих освітніх подій, так і великих міжнародних конференцій.

1.3.5. Платформа Airmeet

Airmeet — це спеціалізована платформа для організації віртуальних та гібридних подій, яка поєднує елементи відеоконференцій, інструменти для нетворкінгу та розширені можливості управління подіями. Вона орієнтована насамперед на великі професійні заходи: онлайн-конференції, вебінари, виставки, панельні дискусії, воркшопи, хакатони, а також корпоративні зустрічі. Особливу популярність Airmeet здобула серед організаторів, які прагнуть забезпечити глибоку взаємодію між учасниками у віртуальному середовищі.

Однією з ключових особливостей платформи є формат віртуальної сцени (stage), яка імітує досвід фізичної присутності. Доповідачі виступають у форматі "живої сцени", тоді як слухачі можуть долучатися до обговорення в режимі запитань та відповідей або заздалегідь запланованих інтерактивних сесій. Також доступні формати "файрсайд чатів", круглих столів і воркшопів з обмеженою кількістю учасників, що дозволяє створювати гнучку програму заходу.

Airmeet має розвинуту систему віртуального нетворкінгу. Користувачі можуть приєднуватися до "соціальних лаунжів", брати участь у "швидких зустрічах" (speed networking) та використовувати тематичні столи, де відбувається обговорення в реальному часі. Така модель спілкування імітує неформальні розмови під час офлайн-заходів, що є цінним для професійного знайомства та побудови зв’язків.

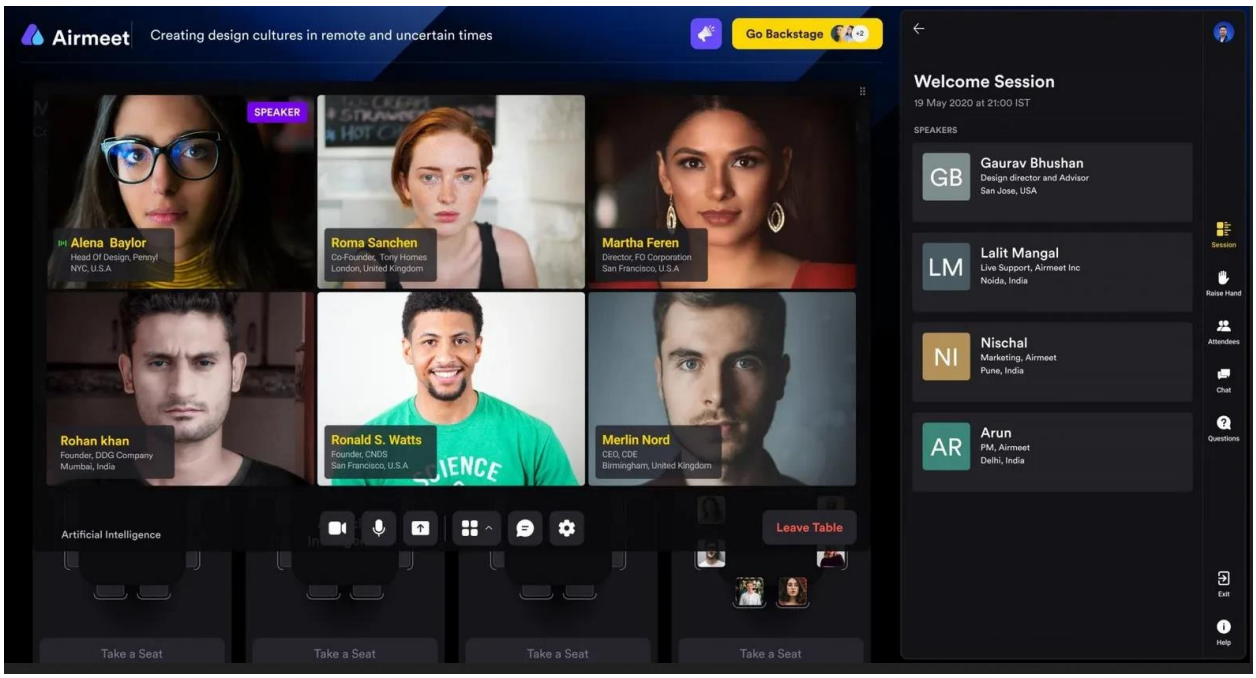


Рисунок 1.8 – Платформа організації подій Airmeet

Участь у події на Airmeet не потребує встановлення додатків — платформа повністю функціонує у веббраузері, що забезпечує легкий доступ для широкої аудиторії. Водночас вона підтримує мобільну версію для базових функцій участі.

Платформа також надає інструменти монетизації — можливість продажу квитків, підключення спонсорів, створення брендованих віртуальних стендів. Експоненти можуть демонструвати свою продукцію, взаємодіяти з відвідувачами в реальному часі та отримувати дані аналітики про інтерес до своїх стендів.

З технічного боку Airmeet забезпечує стабільну потокову трансляцію, підтримку інтерактивних елементів (опитування, голосування, реакції) і розширене модераторське управління. Організатори можуть керувати правами учасників, редагувати програму в реальному часі, а також отримувати детальні аналітичні звіти щодо активності, взаємодій і часу перебування кожного учасника на події.

Airmeet особливо цінується за високий рівень кастомізації — кожен елемент інтерфейсу може бути адаптований до стилістики заходу, включно з логотипами, кольорами, назвами локацій і залів.

Загалом, Airmeet — це повноцінне рішення для створення інтерактивного онлайн-середовища, яке максимально наближене до реального досвіду відвідування подій. Завдяки поєднанню професійного стримінгу, інструментів нетворкінгу та гнучкого адміністрування, ця платформа є потужним інструментом для організації сучасних цифрових подій, орієнтованих на високу залученість учасників та ефективну комунікацію.

1.3.6. Система Accelevents

Accelevents — це сучасна платформа для організації та адміністрування подій, яка поєднує функціональність управління онлайн, гібридними та офлайн-заходами. Завдяки гнучкості, широкому спектру інструментів та простоті використання, платформа набула популярності серед організаторів бізнес-конференцій, галузевих форумів, навчальних сесій, благодійних і маркетингових подій.

Однією з ключових особливостей Accelevents є інтегроване середовище для створення повного циклу події — від моменту планування до підбиття підсумків. Платформа надає зручний конструктор програми заходу, інструменти для керування реєстрацією учасників, продажу квитків, сповіщень, опитувань, а також системи для збирання зворотного зв’язку. Це дозволяє значно спростити роботу організаторам і мінімізувати використання зовнішніх ресурсів.

Важливою перевагою Accelevents є її потужний стримінговий модуль, який забезпечує високу якість трансляцій та дозволяє проводити інтерактивні сесії — з підтримкою чатів, опитувань, реакцій, сесій запитань-відповідей тощо. Учасники можуть легко перемикатися між доповідями, сесіями або

віртуальними кімнатами для спілкування. Для гібридних заходів передбачено синхронізацію досвіду як онлайн-, так і офлайн-учасників.

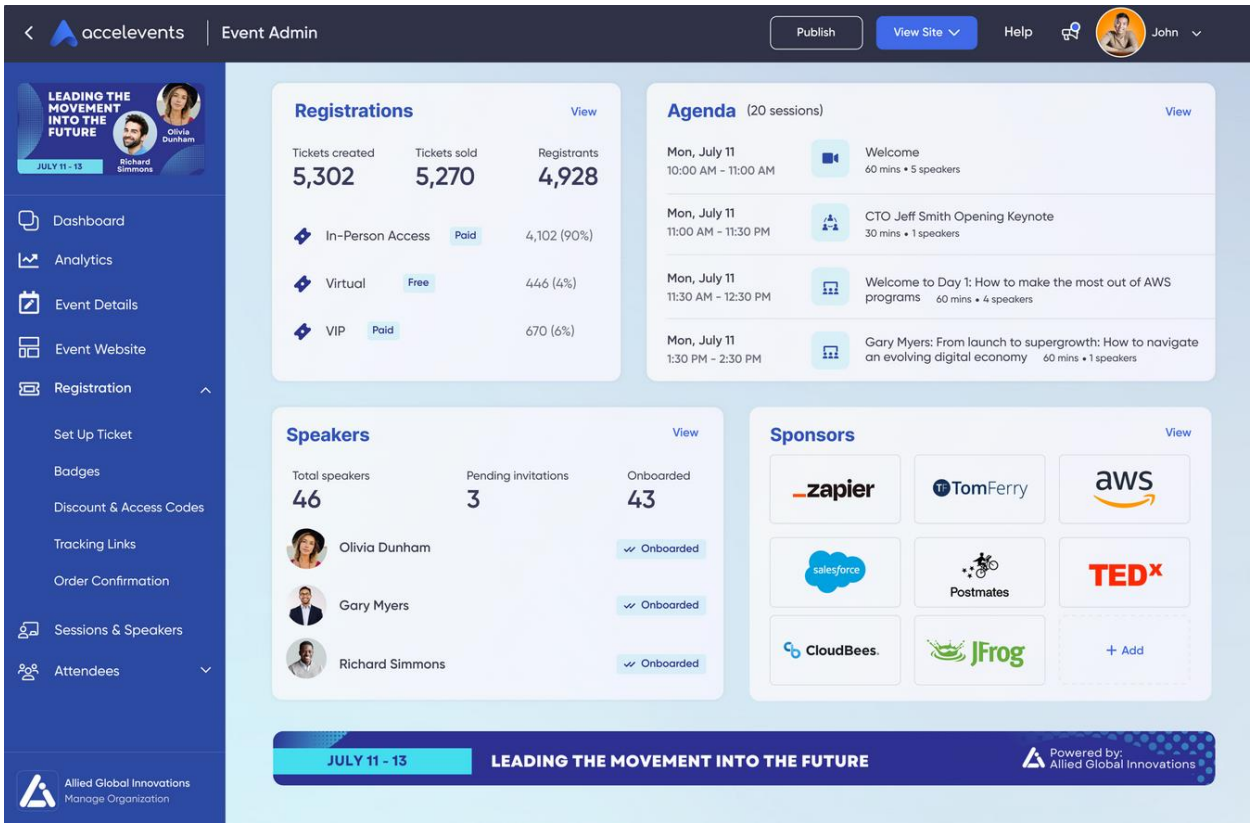


Рисунок 1.9 - Інтерфейс платформи для організації та адміністрування подій Accelevents

Особливу увагу в Accelevents приділено спонсорству та монетизації подій. Організатори можуть створювати брендовані віртуальні стенди для партнерів, розміщувати банери та відео, проводити спонсорські сесії, збирати ліди та аналітику щодо залучення відвідувачів. Це робить платформу привабливою для бізнесу, який прагне підвищити видимість на подіях.

Платформа також пропонує гнучку аналітику — з можливістю відстеження активності учасників, відвідуваності окремих сесій, часу перегляду, рівня взаємодії, кількості встановлених контактів. Ці дані подаються у зрозумілому форматі та можуть бути використані для оцінки ефективності події й обґрунтування рішень у майбутньому.

Крім того, Accelevents підтримує інтеграцію з популярними CRM- та маркетинговими системами, що дозволяє автоматизувати роботу з аудиторією до, під час і після події. Це особливо цінно для компаній, які розглядають події як частину ширшої маркетингової стратегії.

Загалом, Accelevents — це комплексне, масштабоване рішення, орієнтоване на ефективну взаємодію з аудиторією, високу якість подій та простоту в організації. Воно підходить як для великих міжнародних конференцій, так і для невеликих внутрішніх заходів, надаючи організаторам усі необхідні інструменти в єдиному цифровому середовищі.

Більшість провідних платформ для управління подіями пропонують власні мобільні додатки, які тісно інтегровані з їхньою основною системою. Це забезпечує безшовний потік даних між реєстрацією, розкладом, комунікаціями та аналітикою. Деякі платформи дозволяють брендувати ці додатки під подію або організацію, створюючи унікальний користувацький досвід.

РОЗДІЛ 2. ПРОЕКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОГО СЕРВІСУ АДМІНІСТРУВАННЯ ПОДІЙ КОРИСТУВАЧА

2.1. Опис основних сценарій пропонованого сервісу

Систематичний підхід, обраний перед тим, як фактична система буде розроблена в процесі розробки програмного забезпечення, називається проектуванням системи. Компоненти, модулі, архітектура та дані, що проходять через цю систему, визначаються на цьому етапі.

Розглянемо основні сценарії сервісу.

Вхід

1. Користувач запускає додаток Events.
2. Додаток відображає сторінку входу для користувача.
3. Користувач вводить ID та пароль і натискає кнопку входу.
4. Додаток надсилає інформацію для входу на сервер для аутентифікації.
5. Сервер аутентифікує користувача і надсилає відповідь додатку.
6. Додаток показує головну сторінку аутентифікованого користувача.

Перегляд усіх подій

1. Користувач успішно аутентифікований.
2. Додаток показує головну сторінку користувача.
3. Користувач натискає на кнопку "Усі події".
4. Додаток відображає всі події користувачу.

Перегляд моїх подій

1. Користувач успішно аутентифікований.
2. Додаток показує головну сторінку користувача.

3. Користувач натискає на кнопку "Мої події".

4. Додаток фільтрує події, в яких бере участь користувач, і відображає відфільтровані події користувачу. Участь означає, що користувач є або творцем події, або зареєстрований на подію.

Реєстрація/Відміна реєстрації

1. Користувач натискає на подію.

2. Додаток показує деталі події та кнопку для реєстрації/відміни реєстрації.

3. Користувач натискає на кнопку реєстрації/відміни реєстрації.

4. Додаток оновлює дані події, додаючи/видаляючи користувача зі списку учасників.

Відвідування події

1. Користувач натискає на подію.

2. Додаток відображає деталі події та кнопку "Відвідати", якщо користувач зареєстрований на подію.

3. Користувач натискає на кнопку "Відвідати".

4. Додаток відкриває сканер штрих-коду для сканування QR-коду події.

5. Користувач сканує QR-код події, відображений на події.

6. Додаток перевіряє сканований код з кодом події.

7. Додаток оновлює список відвідувань події на сервері з ідентифікатором користувача та відображає повідомлення про успішне сканування.

Додавання події

1. Модератор натискає на кнопку "Додати подію".

2. Додаток відображає форму для введення деталей події.

3. Модератор вводить всі деталі в форму.

4. Модератор натискає на кнопку "Створити подію".

5. Додаток оновлює сервер новою подією.

Отримання учасників

1. Модератор натискає на подію для отримання учасників.

2. Додаток генерує QR-код з коду події.

3. Додаток відображає згенерований QR-код для сканування учасником.

Вихід

1. Користувач натискає на кнопку "Вихід".

2. Додаток надсилає сигнал виходу на сервер і чекає відповіді.

3. Сервер завершує сеанс користувача і повертає статус додатку.

4. Додаток очищає дані користувача і показує сторінку входу.

2.2. Проектування діаграми потоків даних

На рисунку 2.1 подана діаграма потоків даних (DFD) нульового рівня, яка показує, як дані рухаються в системі управління подіями та які основні процеси в ній відбуваються.

Основні компоненти діаграми:

1. Зовнішні сутності:

- User - звичайний відвідувач системи, який може переглядати події, реєструватися на них, відвідувати тощо.

- Moderator - користувач з розширеними правами, який може створювати та керувати подіями.

- Server - зовнішня система або компонент, який, ймовірно, відповідає за зберігання сесій користувачів та синхронізацію даних.

2. Процеси:

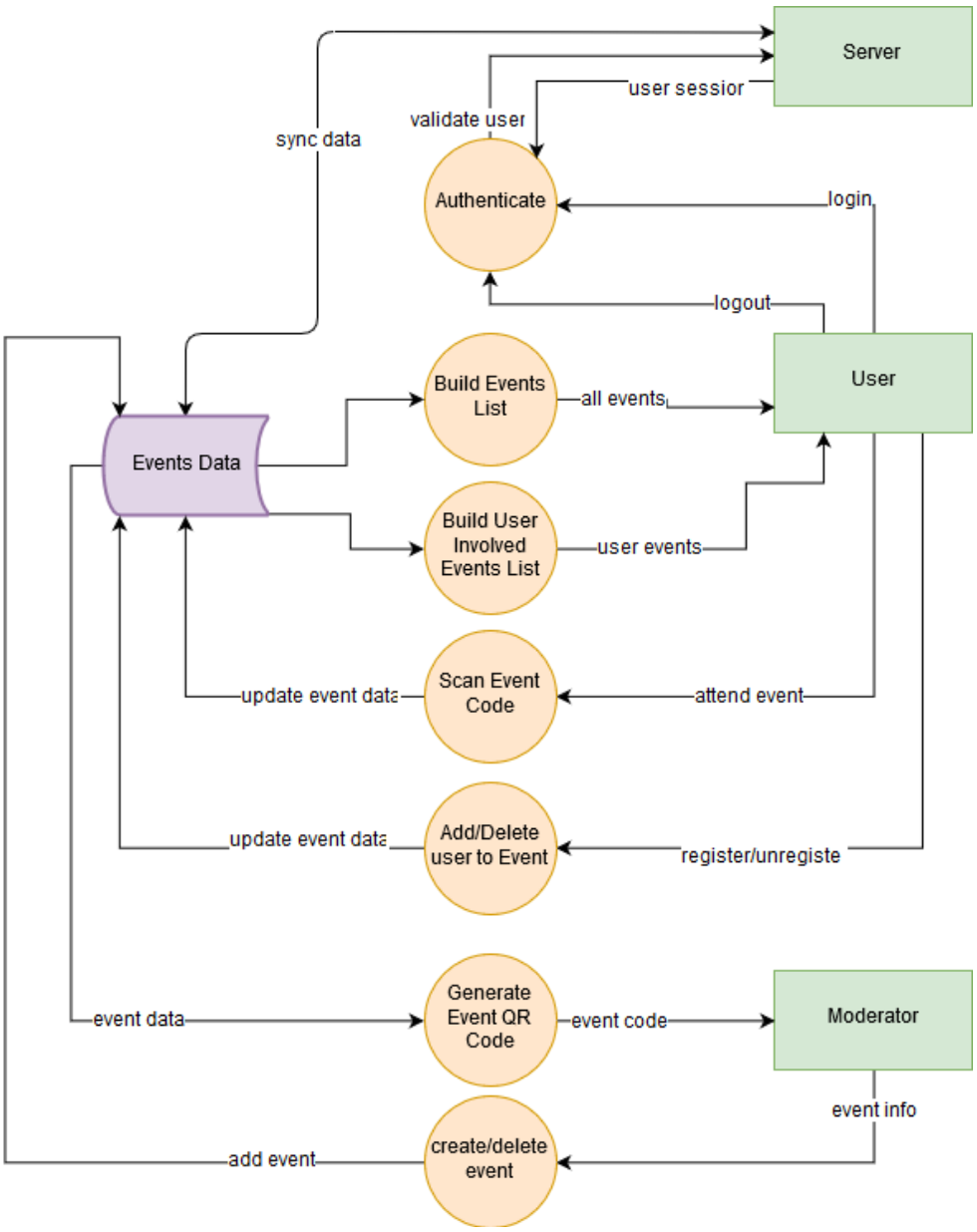


Рисунок 2.1 – Спроектована діаграма потоків даних

- Authenticate - процес, що відповідає за перевірку облікових даних користувача (логін/пароль) та створення сесії.

- Build Events List - процес, що отримує дані про всі доступні події та надає їх користувачеві.

- Build User Involved Events List - процес, що фільтрує події, до яких користувач вже зареєстрований або виявив інтерес.

- Scan Event Code - процес, що обробляє сканування QR-коду події (наприклад, для відмітки відвідуваності).

- Add/Delete user to Event - процес, що керує реєстрацією та скасуванням реєстрації користувача на подію.

- Generate Event QR Code - процес, що створює унікальний QR-код для певної події.

- Create/Delete Event - процес, що дозволяє модератору додавати або видаляти події з системи.

3. Сховище даних:

- Events Data - місце, де зберігається вся інформація про події (назва, дата, час, місце, опис, список учасників тощо).

4. Потоки даних - кожна стрілка представляє потік даних і позначена назвою даних, що передаються.

Наведемо опис потоків даних та взаємодії:

1. Взаємодія Користувача (User):

- Логін (login) - користувач ініціює вхід у систему, надсилаючи облікові дані до процесу "Authenticate".

- Сесія користувача (user session) - після успішної аутентифікації, "Authenticate" передає дані сесії на "Server".

- Вихід (logout) - користувач може вийти з системи/завершення сесії.

- Всі події (all events) - процес "Build Events List" надає користувачеві повний перелік подій.

- Події користувача (user events) - процес "Build User Involved Events List" надає користувачеві список подій, у яких він бере участь.

- Відвідати подію (attend event) - користувач ініціює процес "Scan Event Code", щоб відзначити свою присутність.

- Зареєструватися/скасувати реєстрацію (register/unregister) - користувач ініціює процес "Add/Delete user to Event" для управління своєю участю.

2. Взаємодія Модератора (Moderator):

- Інформація про подію (event info) - модератор надає дані для процесу "Create/Delete Event" для створення або видалення подій.

- Код події (event code) - модератор може запросити генерацію QR-коду для певної події через процес "Generate Event QR Code".

3. Взаємодія з Сервером (Server):

- Перевірити користувача (validate user) - процес "Authenticate" звертається до "Server" для перевірки облікових даних користувача.

- Синхронізація даних (sync data) - "Server" синхронізує дані з "Events Data" (хоча напрямок стрілки може свідчити про те, що дані з Events Data синхронізуються на Server, або Server ініціює синхронізацію).

4. Взаємодія з сховищем "Events Data":

- Дані події (event data) - процес "Generate Event QR Code" отримує інформацію про подію зі сховища "Events Data" для генерації коду.

- Додати подію (add event) - процес "Create/Delete Event" записує нову інформацію про подію до "Events Data".

- Оновити дані події (update event data) - процеси "Scan Event Code" та "Add/Delete user to Event" оновлюють дані про події (наприклад, змінюють статус відвідуваності або списки учасників) у "Events Data".

- Дані події (event data) - процеси "Build Events List" та "Build User Involved Events List" читають дані про події зі сховища.

Ця DFD візуалізує основні потоки інформації в системі управління подіями. Вона показує, хто (User, Moderator, Server) взаємодіє з системою, які ключові функції виконуються (Authenticate, Build Lists, Scan, Add/Delete, Generate QR, Create/Delete Event), і де зберігаються центральні дані (Events Data). Це дає загальне уявлення про архітектуру системи та її взаємозв'язки без заглиблення в деталі реалізації кожного процесу.

2.3. Проектування діаграми випадків використання

Елементи в системі взаємодіють для виконання завдання в системі. Ці взаємодії можна зобразити за допомогою діаграми випадків використання.

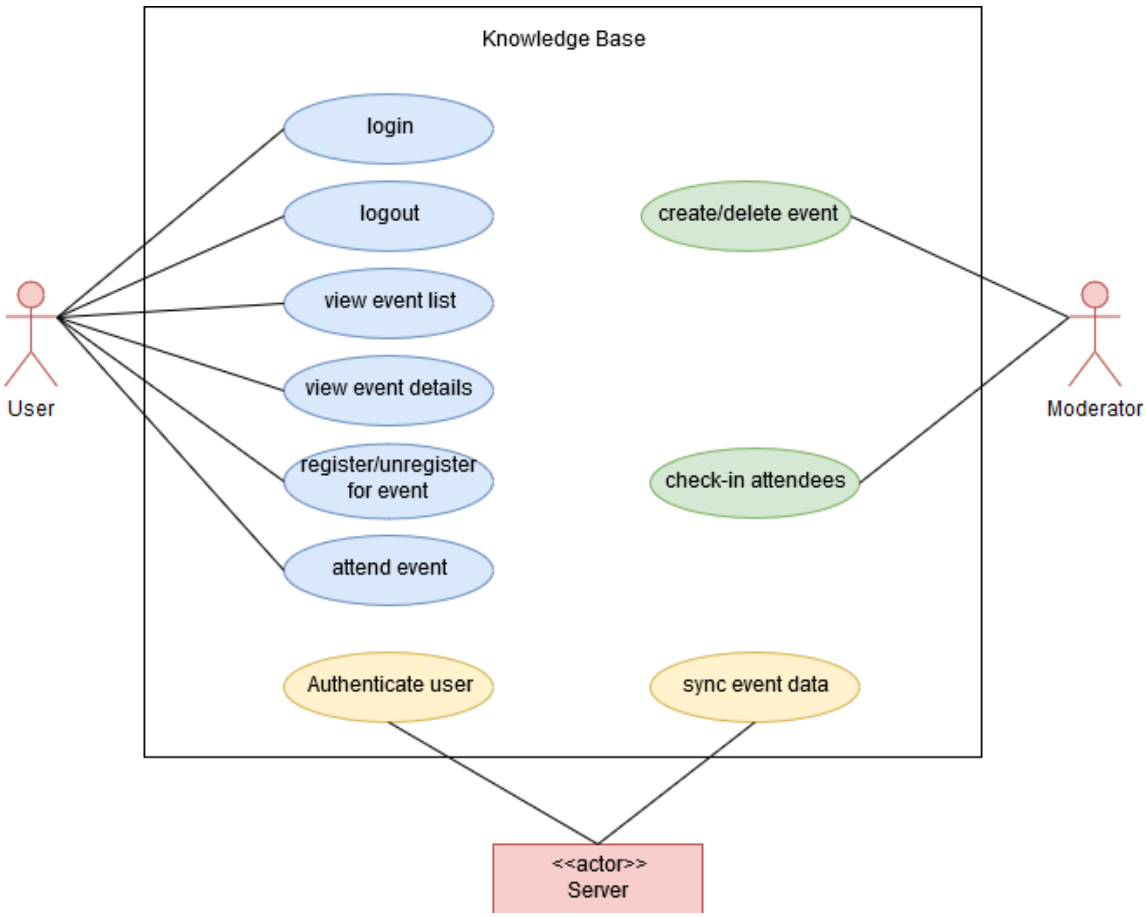


Рисунок 2.2 - Діаграма випадків використання

На діаграмі ідентифіковано три основні суб'єкти (акторів): User (Користувач), Moderator (Модератор) та Server (Сервер). Кожен з них має власну сукупність функціональних сценаріїв, які реалізуються у межах системи.

Користувач взаємодіє з системою шляхом виконання наступних дій:

- login (вхід у систему),
- logout (вихід із системи),
- view event list (перегляд списку подій),
- view event details (перегляд деталей події),
- register/unregister for event (реєстрація/скасування участі в події),
- attend event (відвідування події).

Ці дії забезпечують базовий функціонал користувача щодо ознайомлення з подіями та участі в них.

Модератор має розширений набір повноважень, що включає:

- create/delete event (створення/видалення подій),

- check-in attendees (відмітка учасників, які відвідали подію).

Це підкреслює роль модератора як адміністративного користувача, відповідального за управління подіями та контролем відвідуваності.

Сервер визначено як зовнішній технічний актор (<<actor>>), який виконує дві ключові системні функції:

- Authenticate user (автентифікація користувача) — забезпечує перевірку облікових даних при вході в систему;

- sync event data (синхронізація даних про події) — відповідає за оновлення інформації між клієнтською частиною та базою знань.

Діаграма ілюструє централізовану модель доступу до функціоналу системи на основі ролей, що сприяє підвищенню безпеки та організованості доступу до подій.

2.4. Розробка діаграм послідовності

Діаграми послідовності зображають взаємодії між різними об'єктами в системі в часовій послідовності.

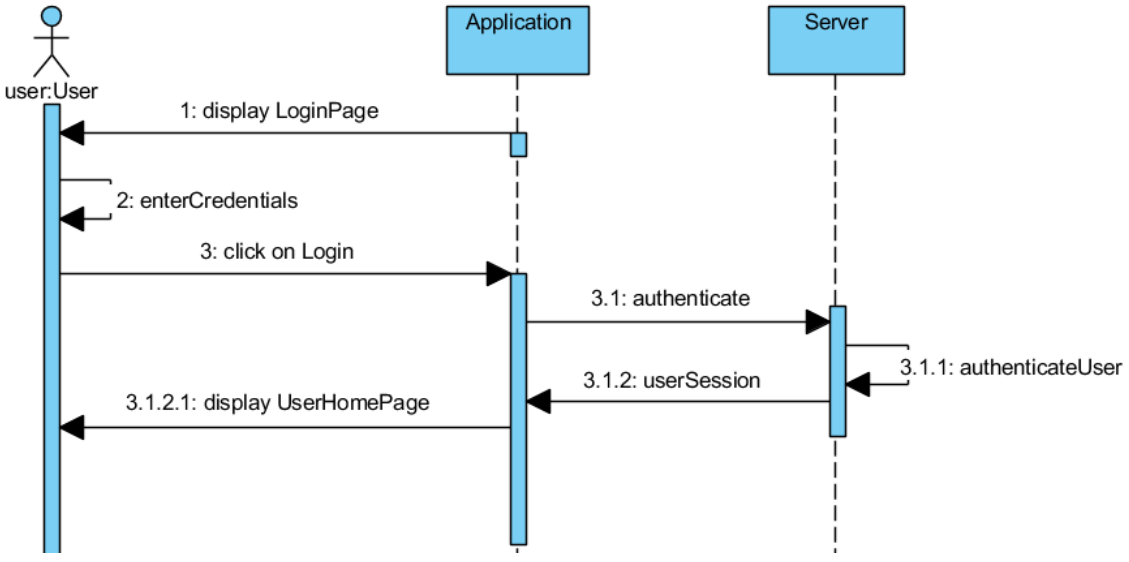


Рисунок 2.3 - Діаграма послідовності – Вхід

Представлена діаграма послідовності (рис. 2.3) моделює процес автентифікації користувача у програмному сервісі з використанням взаємодії між трьома основними об’єктами: користувачем (User), додатком (Application) та сервером (Server). Ця UML-діаграма ілюструє часову послідовність повідомлень, що передаються між учасниками системи під час виконання сценарію входу в систему.

На початку взаємодії (крок 1) система відображає сторінку входу (LoginPage) користувачу. Це ініціалізує взаємодію між користувачем і додатком. Далі (крок 2) користувач вводить свої облікові дані (логін та пароль), після чого виконує дію натискання на кнопку входу (крок 3: click on Login).

Ця дія активує програмну логіку на стороні клієнтського додатку, яка ініціює запит на автентифікацію до сервера (крок 3.1: authenticate). Сервер, у свою чергу, викликає внутрішній механізм перевірки користувача (крок 3.1.1: authenticateUser), який обробляє введені облікові дані, порівнюючи їх із тими, що зберігаються у базі даних чи іншій системі зберігання облікової інформації.

Після успішної перевірки автентичності сервер повертає результат у вигляді сесії користувача (крок 3.1.2: userSession) до додатку. На основі цієї інформації додаток формує відповідну відповідь, у якій відображає домашню сторінку користувача (UserHomePage) (крок 3.1.2.1) як підтвердження успішного входу.

Таким чином, діаграма послідовності демонструє узгоджену та структуровану логіку аутентифікації користувача в системі організації подій, відображаючи ключові етапи обробки введених даних, взаємодію між клієнтом та сервером, а також завершення сценарію через запуск користувацької сесії та зміну інтерфейсу.

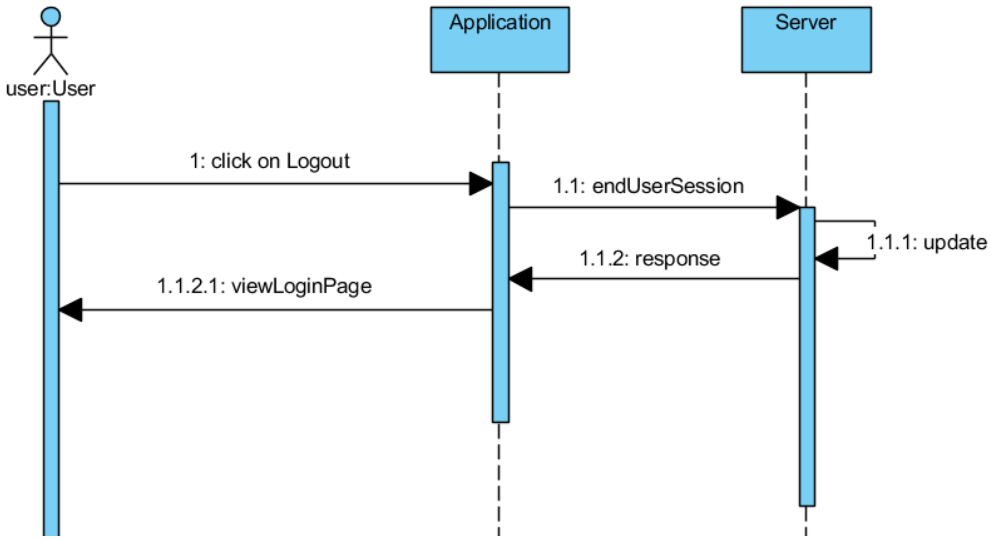


Рисунок 2.4 - Діаграма послідовності – Вихід

Представлена діаграма послідовності (рис. 2.4) моделює процес виходу користувача із системи (logout) у програмному сервісі адміністрування подій. Діаграма відображає часову послідовність взаємодії між трьома суб'єктами: користувачем (User), додатком (Application) та сервером (Server). Вона демонструє логіку завершення користувацької сесії та повернення до сторінки входу в систему.

На першому етапі (крок 1) користувач ініціює процес виходу, натискаючи кнопку Logout. Ця дія передається клієнтському застосунку, який запускає процес завершення сесії (крок 1.1: endUserSession), надсилаючи відповідний запит на сервер.

Сервер обробляє запит додатку шляхом оновлення стану користувача або сеансу у внутрішній системі управління сесіями (крок 1.1.1: update), що включає очищення збережених маркерів автентифікації або записів активності. Після цього сервер надсилає відповідь додатку (крок 1.1.2: response), яка підтверджує завершення операції.

У фінальному кроці (1.1.2.1: viewLoginPage) додаток оновлює інтерфейс, відображаючи користувачу сторінку входу, що означає успішний вихід із системи.

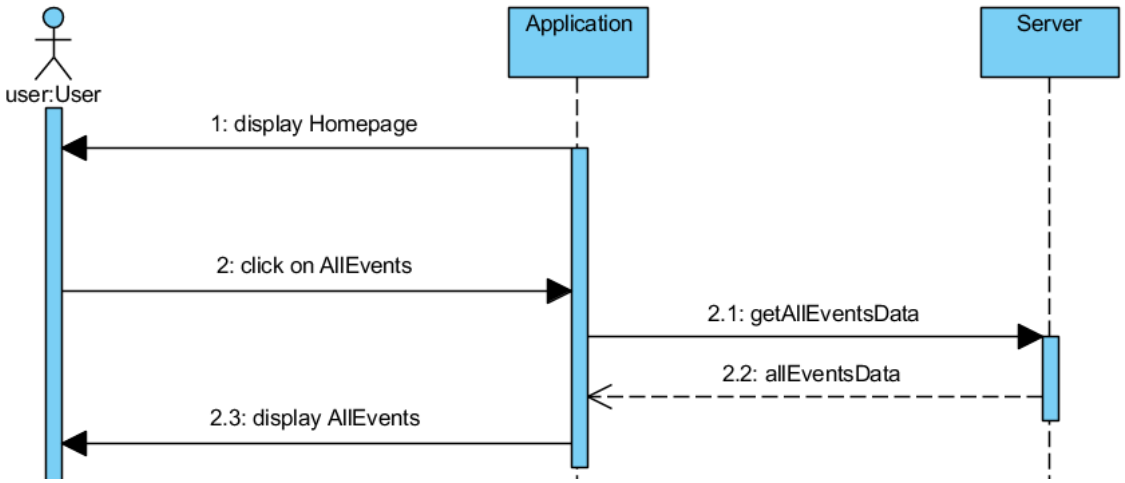


Рисунок 2.5 - Діаграма послідовності - Перегляд усіх подій

Представлена діаграма (рис. 2.5) відображає взаємодію між користувачем, Application та сервером під час доступу до сторінки з усіма подіями (AllEvents).

У даному випадку описується наступний сценарій:

1. Ініціація сеансу користувача. Користувач (actor user:User) відкриває головну сторінку застосунку. Ця дія позначена повідомленням "1: display Homepage", яке надходить від прикладної програми до користувача як реакція на завантаження інтерфейсу.

2. Запит на перегляд усіх подій. Користувач натискає на елемент інтерфейсу "AllEvents", що генерує повідомлення "2: click on AllEvents" до прикладної програми.

3. У відповідь на дію користувача прикладна програма надсилає запит до сервера — повідомлення "2.1: getAllEventsData", яке ініціює отримання даних про всі події.

4. Сервер обробляє запит і повертає відповідь у вигляді повідомлення "2.2: allEventsData", що містить релевантні дані.

5. Отримані дані обробляються прикладною програмою і надсилаються назад до користувача у вигляді повідомлення "2.3: display AllEvents", що призводить до візуалізації інформації про всі події на інтерфейсі

користувача.

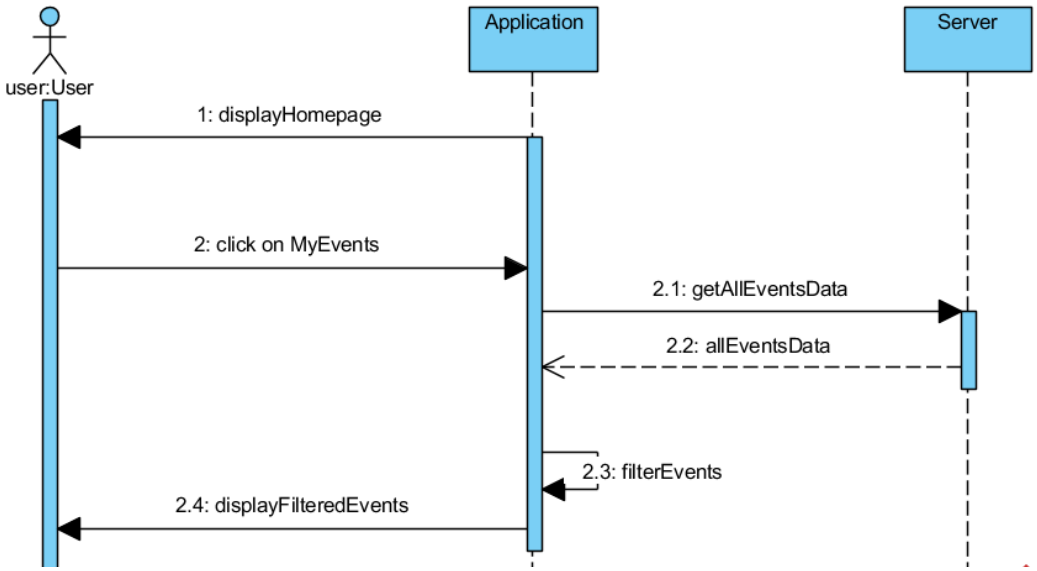


Рисунок 2.6 - Діаграма послідовності - Перегляд моїх подій

Наведена діаграма (рис. 2.6) моделює сценарій взаємодії між користувачем, прикладною програмою та сервером у процесі відображення персоналізованих подій користувача ("MyEvents").

Опис послідовності взаємодій:

1. Початковий стан – відображення головної сторінки.

Користувач (User) ініціює запуск застосунку, в результаті чого прикладна програма формує інтерфейс головної сторінки. Цей етап представлений повідомленням "1: displayHomepage".

2. Запит на перегляд власних подій.

Користувач натискає на елемент інтерфейсу "MyEvents", що активує повідомлення "2: click on MyEvents", адресоване прикладній програмі. Таким чином ініціюється процес обробки запиту на отримання даних про події.

3. Комунікація із сервером:

Додаток надсилає до сервера запит "2.1: getAllEventsData" з метою отримання повного переліку подій. У відповідь сервер повертає повідомлення "2.2: allEventsData", що містить дані про всі події в системі.

4. Фільтрація подій за користувачем.

Після отримання даних прикладна програма виконує внутрішній метод "2.3: filterEvents", що реалізує логіку відбору лише тих подій, які пов’язані з поточним користувачем (наприклад, за ID користувача або іншими критеріями).

5. Візуалізація результату.

На завершальному етапі прикладна програма надсилає користувачу результат у вигляді повідомлення "2.4: displayFilteredEvents", що призводить до відображення лише релевантних подій.

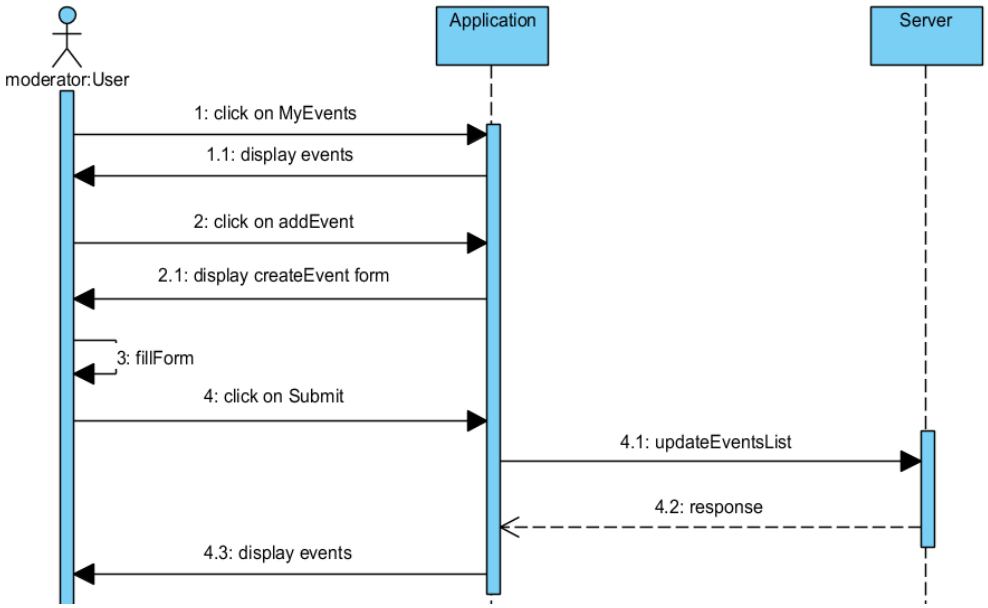


Рисунок 2.7 - Діаграма послідовності - Додавання події

Діаграма послідовності (рис. 2.7) моделює процес взаємодії між модератором, додатком та сервером у контексті створення нової події.

Опис сценарію взаємодії:

1. Ініціація перегляду подій.

Модератор (moderator:User) розпочинає взаємодію, натискаючи на опцію "MyEvents". У відповідь додаток відображає список подій, що належать модератору (повідомлення "1.1: display events").

2. Ініціація створення нової події.

Наступним кроком є натискання на опцію "addEvent", що викликає повідомлення "2: click on addEvent". У відповідь додаток завантажує та відображає форму створення події — повідомлення "2.1: display createEvent form".

3. Введення даних.

Модератор заповнює форму (подія "3: fillForm"), після чого натискає кнопку підтвердження — "4: click on Submit".

4. Передача даних на сервер.

Прикладна програма передає введену інформацію до сервера за допомогою повідомлення "4.1: updateEventsList", яке ініціює оновлення переліку подій на стороні сервера.

5. Обробка відповіді.

Після обробки запиту сервер повертає повідомлення "4.2: response", що підтверджує успішне оновлення або інформує про результат операції.

6. Відображення оновленого списку подій.

У завершальній фазі прикладна програма формує оновлений список подій і відображає його користувачу — повідомлення "4.3: display events".

Представлена діаграма послідовності ілюструє сценарій створення нової сутності в інформаційній системі організації і адміністрування подій із роллю модератора, що має розширені повноваження. Вона демонструє повну транзакцію створення об'єкта, що охоплює як ініціацію дії на клієнтському боці, так і її завершення на сервері з наступним оновленням інтерфейсу користувача.

Така модель є хорошим рішенням для сучасних веб-застосунків та мобільних додатків, де забезпечується чіткий поділ відповідальності між фронтендом і бекендом, а також забезпечується інтерактивна взаємодія з користувачем.

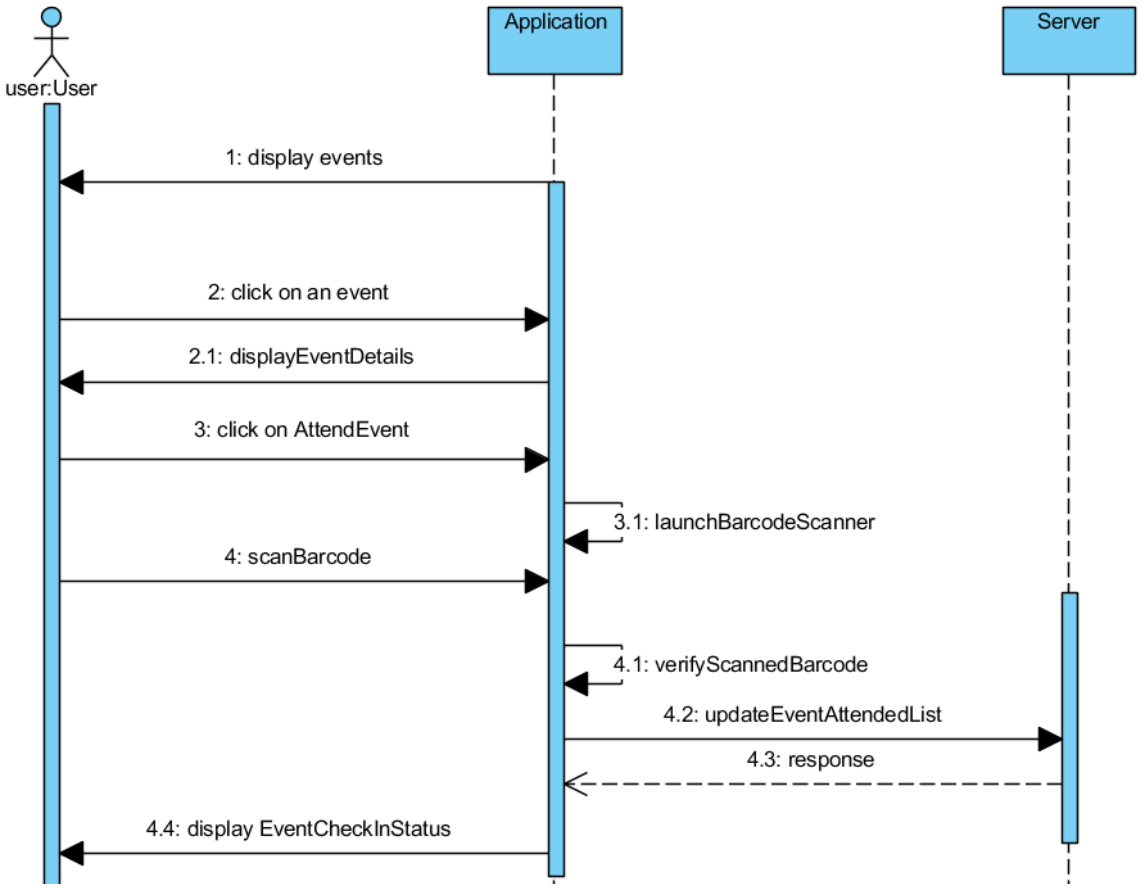


Рисунок 2.8 - Діаграма послідовності - Відвідування події

Діаграма послідовності (рис. 2.8) представляє взаємодію між користувачем, додатком та сервером у процесі обробки участі в заході з використанням сканування штрих-коду. Процес розпочинається з того, що додаток отримує від сервера список подій і відображає їх користувачу (крок 1: display events). Користувач обирає конкретну подію, натискаючи на неї (крок 2: click on an event), після чого додаток запитує та відображає деталі обраної події (крок 2.1: displayEventDetails).

Далі користувач вирішує взяти участь у події, натискаючи на опцію "AttendEvent" (крок 3: click on AttendEvent), що активує запуск сканера штрих-кодів додатком (крок 3.1: launchBarcodeScanner). Користувач сканує штрих-код (крок 4: scanBarcode), і отримані дані передаються додатком до сервера. Сервер верифікує відсканований штрих-код (крок 4.1: verifyScannedBarcode) та оновлює список учасників події (крок 4.2:

updateEventAttendedList), після чого надсилає відповідь додатку (крок 4.3: response). Нарешті, додаток відображає користувачу статус перевірки участі (крок 4.4: displayEventCheckInStatus), завершуючи процес.

Така послідовність ілюструє чіткий обмін повідомленнями між компонентами системи, забезпечуючи ефективну взаємодію для реалізації функціоналу реєстрації на подію.

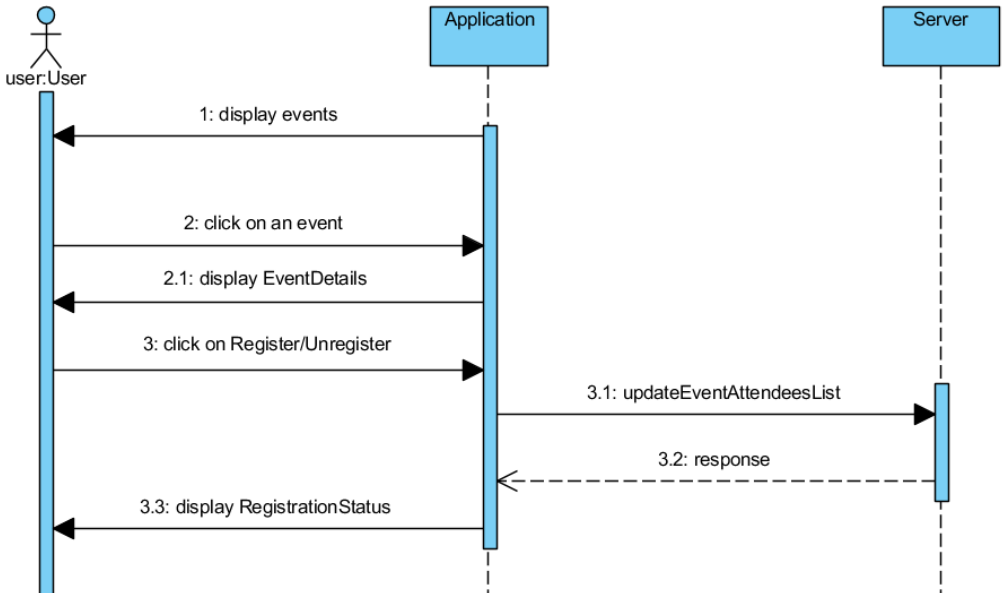


Рисунок 2.9 - Діаграма послідовності - Реєстрація/Відміна реєстрації

Діаграма послідовності (рис. 2.9) відображає взаємодію між користувачем, додатком та сервером у процесі реєстрації або скасування реєстрації на подію. Процес розпочинається з того, що додаток отримує від сервера список подій і відображає їх користувачу (крок 1: display events). Користувач обирає конкретну подію, натискаючи на неї (крок 2: click on an event), після чого додаток запитує та відображає деталі обраної події (крок 2.1: displayEventDetails).

Далі користувач вирішує зареєструватися або скасувати реєстрацію, натискаючи на опцію "Register/Unregister" (крок 3: click on Register/Unregister). Додаток передає запит до сервера, який оновлює список учасників події (крок 3.1: updateEventAttendeesList) та надсилає відповідь

додатку (крок 3.2: response). На завершальному етапі додаток відображає користувачу статус реєстрації (крок 3.3: displayRegistrationStatus), завершуючи процес.

Ця послідовність демонструє структурований обмін повідомленнями між компонентами системи, забезпечуючи ефективну реалізацію функціоналу управління участю в подіях.

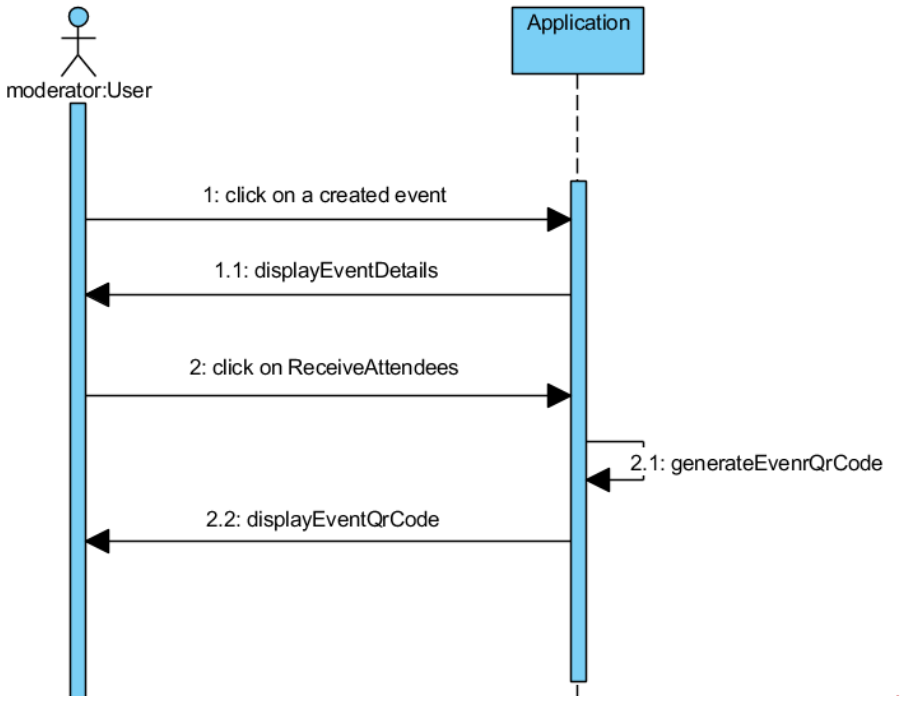


Рисунок 2.10 - Діаграма послідовності - Отримання учасників

Діаграма послідовності (рис. 2.10) ілюструє взаємодію між модератором і додатком у процесі управління подією та генерації QR-коду для неї. Процес розпочинається з того, що модератор обирає створену подію, натискаючи на неї (крок 1: click on a created event), після чого додаток відображає деталі цієї події (крок 1.1: displayEventDetails).

Далі модератор ініціює отримання списку учасників, натискаючи на опцію "ReceiveAttendees" (крок 2: click on ReceiveAttendees). У відповідь додаток генерує QR-код для події (крок 2.1: generateEventQRCode) та

відображає його модератору (крок 2.2: displayEventQRCode), завершуючи процес.

Ця послідовність демонструє чіткий обмін повідомленнями між модератором і додатком, забезпечуючи ефективне управління подією та доступ до необхідної інформації у вигляді QR-коду.

2.5. Визначені користувацькі компоненти системи

У рамках розробки даної програмної системи, з метою забезпечення узгодженості інтерфейсу та підвищення ефективності розробки, було ідентифіковано та визначено необхідність створення наступних користувацьких компонентів:

1. Компонент "Кнопка Додатку" (Application Button Component)

Призначення.

Створення уніфікованого графічного елемента взаємодії, що представляє собою кнопку з текстовою міткою.

```
1  import React, { Component } from 'react';
2  import { View, Text, Button } from 'react-native';
3
4  export default class AppButton extends Component{
5      render() {
6          return (
7              <View style={{ marginTop: 5, margin:10 }}>
8                  <Button
9                  title={this.props.title}
10                 onPress={this.props.onPress}
11                 />
12             </View>
13         );
14     }
15 }
```

Рисунок 2.11 - Компонент кнопки додатку

Обґрунтування.

Необхідність стандартизації візуального стилю та поведінки всіх текстових кнопок в інтерфейсі додатку. Використання єдиного компонента гарантує відповідність дизайнерським вимогам, спрощує внесення

глобальних змін у зовнішній вигляд кнопок та зменшує вірогідність розбіжностей у стилях, що забезпечує покращену користувацьку експірієнс та підтримку коду.

2. Компонент "Список Подій" (Event List Component)

Призначення.

Відображення переліку подій, що передаються компоненту як вхідні дані.

Обґрунтування.

Виявлено, що функціональна мета сторінок "Усі Події" (All Events) та "Мої Події" (My Events) полягає у представленні списків подій. Розробка окремого, параметризованого компонента для відображення списків подій дозволяє уникнути дублювання коду, централізувати логіку рендерингу елементів списку та забезпечити послідовність у представленні інформації про події на різних екранах додатку. Цей компонент буде приймати набір об'єктів подій та відповідатиме за їхнє структурне та стилістичне відображення.

```
import React, { Component } from 'react';
import { FlatList, StyleSheet, View, Text, TouchableOpacity } from 'react-native';
export default class ListEvents extends Component {
    render() {
        return (
            <View style={styles.container}>
                <FlatList
                    data={this.props.eventList.map((item, i) => Object.assign({key:i}, item))}
                    renderItem={({item}) =>
                        <TouchableOpacity style={styles.item}
                            onPress={() => {this.props.navigation.navigate('eventDetails', { ev
                            <Text style={styles.eventName}>{item.name.substring(0,32)}{item.nam
                            <Text style={styles.eventOrganizer}>{item.organizer}</Text>
                            <Text style={styles.eventDate}>{item.when.date}</Text>
                        </TouchableOpacity>
                    }
                />
            </View>
        );
    }
}
```

Рисунок 2.12 - Компонент списку подій

2.6. Компоненти навігації в архітектурі додатку

У архітектурі мобільних додатків ключове значення має реалізація механізмів навігації між різними екранами. Для вирішення цієї задачі було використано два основних типи навігаторів: перемикач навігації (Switch Navigator) та стек навігації (Stack Navigator), кожен з яких виконує специфічні функції.

1. Перемикач Навігації (Switch Navigator)

Призначення.

Забезпечення переходу між екранами без збереження історії навігації. Це означає, що при переході з одного екрану на інший за допомогою Switch Navigator, попередній екран видаляється зі стеку, і немає можливості повернутися до нього за допомогою стандартних навігаційних операцій (наприклад, кнопки "назад").

Функціональні особливості.

Відсутність збереження стану попередніх екранів є критично важливою для сценаріїв, де не потрібно дозволяти користувачу повертатися до попереднього стану (наприклад, після успішного входу в систему).

Застосування в системі.

У даній реалізації Switch Navigator використовується для управління потоком входу до додатку. Це забезпечує безпечний перехід від екранів завантаження та входу до основного функціоналу додатку без можливості "відкату" назад до екранів, що вимагають аутентифікації.

Конфігурація Switch Navigator включає наступні екрани/навігатори:

- LoadingScreen, що відображається під час отримання або перевірки даних користувача з сервера.

Login - екран для входу користувача в додаток.

RootStack (тип Stack Navigator) - основний стек навігації, який містить усі екрани після успішної аутентифікації.

Початковим екраном для Switch Navigator встановлено LoadingScreen.

```
import { createSwitchNavigator } from 'react-navigation'
// import the different screens
import LoadingScreen from './loadingScreen'
import Login from './login'
import RootStack from './appNavigator'
// create our app's navigation
export const RootSwitch = createSwitchNavigator(
    {
        LoadingScreen,
        Login,
        rootStack: RootStack,
    },
    {
        initialRouteName: 'LoadingScreen'
    }
);
```

Рисунок 2.13 - Перемикач навігації

2. Стек Навігації (Stack Navigator)

Призначення.

Забезпечення навігації між екранами за принципом стеку (LIFO - Last In, First Out), що дозволяє зберігати історію переходів. При додаванні нового екрану він "накладається" на попередній, і користувач може легко повернутися до попереднього екрану, використовуючи функцію "назад".

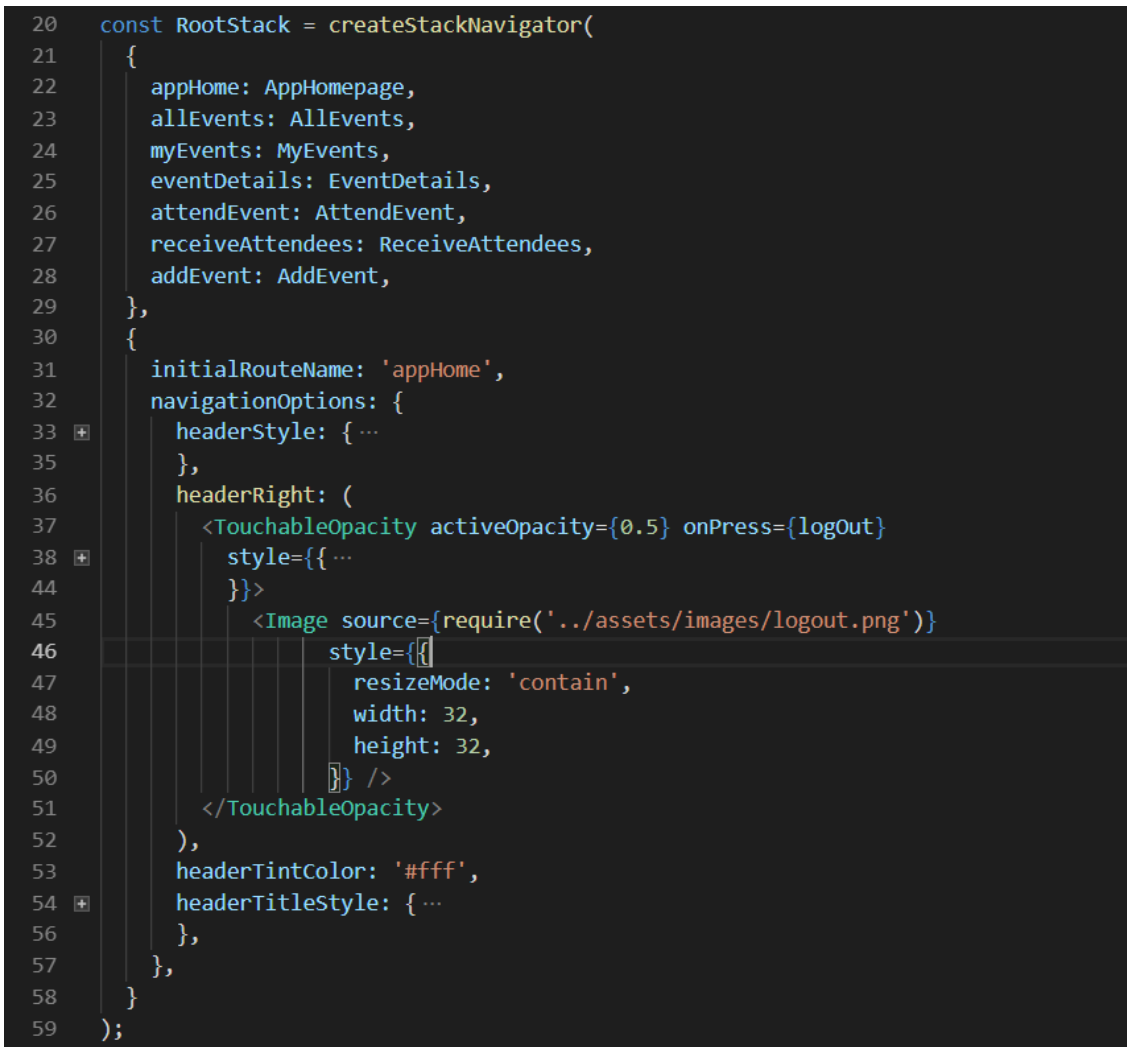
Функціональні особливості.

Цей тип навігатора ідеально підходить для управління послідовними екранами в рамках одного логічного потоку додатку, де важливим є збереження контексту та можливість повернення.

Застосування в системі.

Після успішного входу користувача в додаток, керування навігацією передається Stack Navigator. Це дозволяє користувачеві вільно переміщатися між різними розділами додатку (наприклад, списками подій, деталями подій, профілем тощо) зі збереженням історії переходів.

Stack Navigator інкапсулює основні екрани додатку, які стають доступними після аутентифікації.



```
const RootStack = createStackNavigator(
  {
    appHome: AppHomepage,
    allEvents: AllEvents,
    myEvents: MyEvents,
    eventDetails: EventDetails,
    attendEvent: AttendEvent,
    receiveAttendees: ReceiveAttendees,
    addEvent: AddEvent,
  },
  {
    initialRouteName: 'appHome',
    navigationOptions: {
      headerStyle: {···
      },
      headerRight: (
        <TouchableOpacity activeOpacity={0.5} onPress={logOut}
          style={{···
          }}>
            <Image source={require('../assets/images/logout.png')}
                   style={{
                     resizeMode: 'contain',
                     width: 32,
                     height: 32,
                   }} />
        </TouchableOpacity>
      ),
      headerTintColor: '#fff',
      headerTitleStyle: {···
      },
    },
  }
);
```

Рисунок 2.14 - Стек навігації

Таким чином, комбіноване використання Switch Navigator для управління потоком аутентифікації та Stack Navigator для організації основної навігації всередині додатку забезпечує ефективну, безпечну та інтуїтивно зрозумілу взаємодію користувача з системою.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО СЕРВІСУ АДМІНІСТРУВАННЯ ПОДІЙ КОРИСТУВАЧА

3.1. Розробка інтерфейсу основних сторінок додатку

Розробка користувацького інтерфейсу системи включає створення кількох ключових екранів, кожен з яких виконує специфічну функцію у взаємодії користувача з додатком.

1. Екран Завантаження (Loading Screen)

Призначення.

Інформування користувача про процес ініціалізації додатку та завантаження необхідних ресурсів.

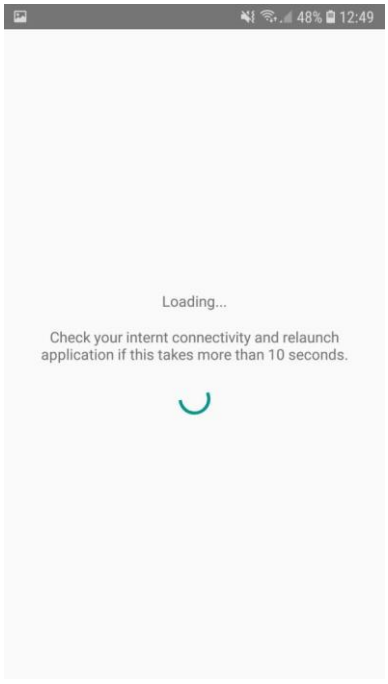


Рисунок 3.1 - Екран завантаження

Візуальне представлення.

Екран містить центрально розташований індикатор завантаження (спіннер), що візуалізує активність фонових процесів. Це забезпечує позитивний користувацький досвід, запобігаючи враженню "зависання"

додатку під час тривалих операцій (наприклад, отримання даних користувача або конфігурації з сервера).

2. Екран Входу (Login Screen)

Призначення.

Надання інтерфейсу для автентифікації користувача в системі.

Функціональність.

Екран містить поля для введення ідентифікатора студента (ID) та відповідного пароля. Після введення облікових даних користувач може ініціювати процес входу, що забезпечує доступ до функціоналу додатку.

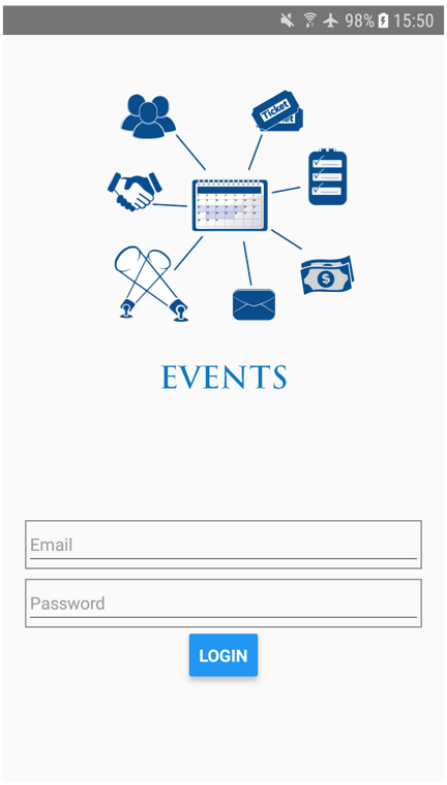


Рисунок 3.2 - Екран входу

3. Головна Сторінка (Home Screen)

Призначення.

Персоналізована стартова точка для користувача після успішної автентифікації.

Функціональність.

На екрані відображається вітальне повідомлення, що включає ім'я користувача, забезпечуючи індивідуалізований підхід. З цього екрану користувач має можливість перейти до двох ключових розділів системи: "Усі Події" (All Events) та "Мої Події" (My Events) за допомогою відповідних кнопок. Це забезпечує інтуїтивно зрозумілу навігацію до основних функцій додатку.

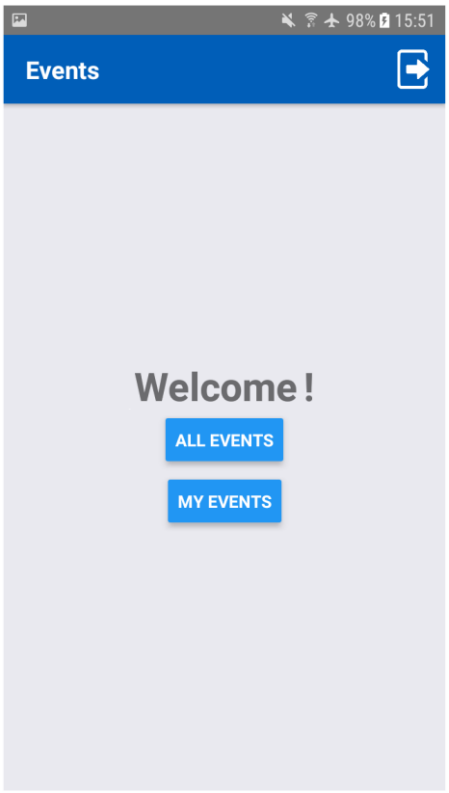


Рисунок 3.3 - Головна сторінка

Ці екрани формують базовий потік взаємодії користувача, починаючи від завантаження та входу, до доступу до основних функціональних можливостей системи.

3.2. Реалізація функціоналу управління подіями в системі

Інтерфейс системи включає низку екранів, призначених для ефективного відображення та управління інформацією про події, враховуючи роль користувача.

1. Екран "Усі Події" (All Events Screen)

Призначення.

Представлення консолідованого переліку всіх доступних подій, що містяться в базі даних системи.

Функціональність.

На цьому екрані користувачам надається стисла інформація про кожну подію, що дозволяє швидко оглянути поточні та майбутні заходи. Інформація зазвичай включає назву події, дату, час, місце проведення та, можливо, організатора.

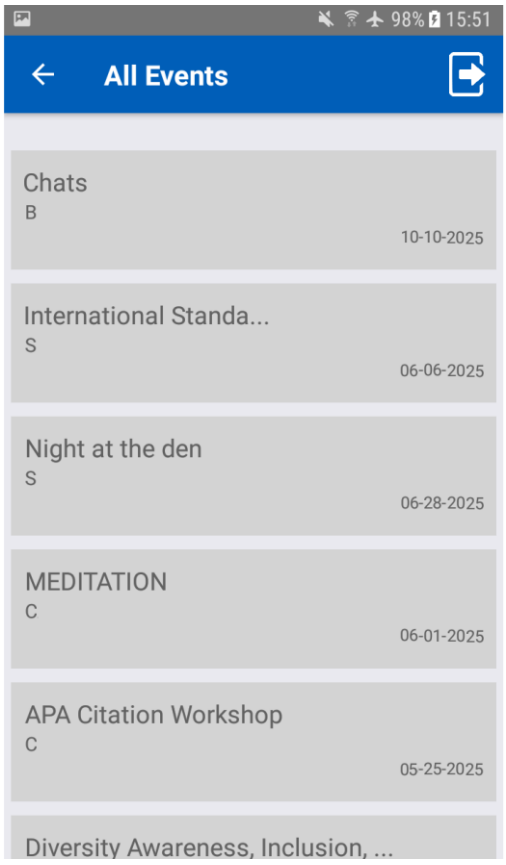


Рисунок 3.4 – Інтерфейс сторінки "Усі події"

2. Екран "Мої Події" (My Events Screen)

Призначення.

Персоналізоване відображення подій, у яких поточний користувач є учасником або організатором.

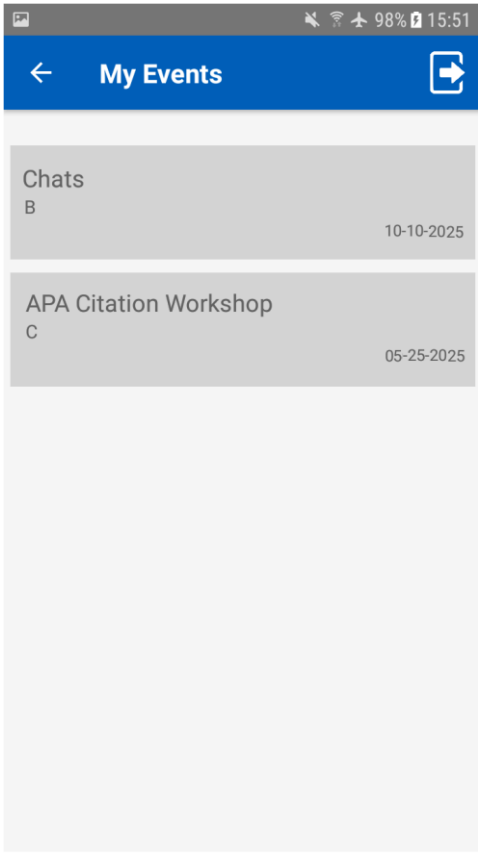


Рисунок 3.5 - Сторінка "Мої події" (для звичайного користувача)

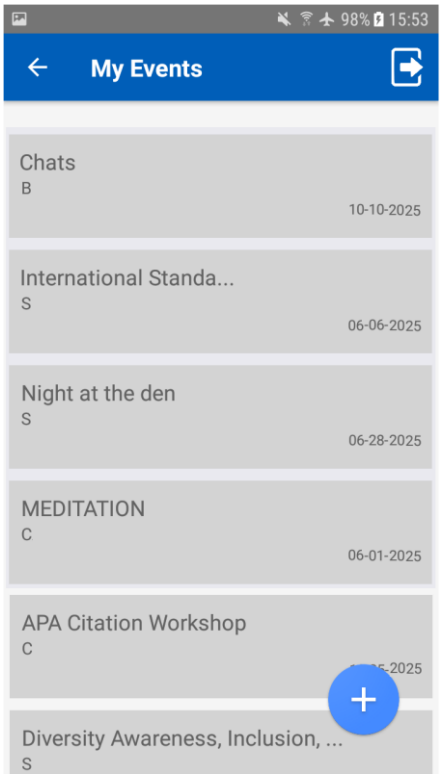


Рисунок 3.6 - Сторінка "Мої події" (для користувача з роллю модератора)

Функціональність.

Екран містить стислу інформацію лише про ті події, які релевантні для авторизованого користувача, тобто, на які він зареєстрований або які він створив.

Для звичайних користувачів відображається список подій, на які вони зареєстровані, надаючи швидкий доступ до відповідної інформації (рис. 3.5).

Для користувачів з роллю модератора (рис. 3.6) додатково до списку подій, у яких модератор бере участь, у правому нижньому куті екрану розміщено круглу плаваючу кнопку (Floating Action Button). Ця кнопка слугує для швидкого доступу до функції створення нової події.

3. Екран "Деталі Події" (Event Details Screen)

Призначення.

Надання вичерпної інформації про конкретну подію та можливості взаємодії з нею, залежно від статусу користувача.

Функціональність.

Цей екран активується після вибору певної події зі списку. Він містить повний набір даних про подію, включаючи:

- Назву події.
- Детальний опис.
- Ім'я організатора.
- Місце проведення.
- Дату та час події.
- Динамічні кнопки взаємодії (залежно від ролі та статусу користувача):

- для незареєстрованих учасників відображається кнопка для реєстрації на подію (рис. 3.7).

- для зареєстрованих учасників відображаються кнопки для скасування реєстрації та/або для відмітки відвідування події (сканування коду, якщо це передбачено) (рис. 3.8).

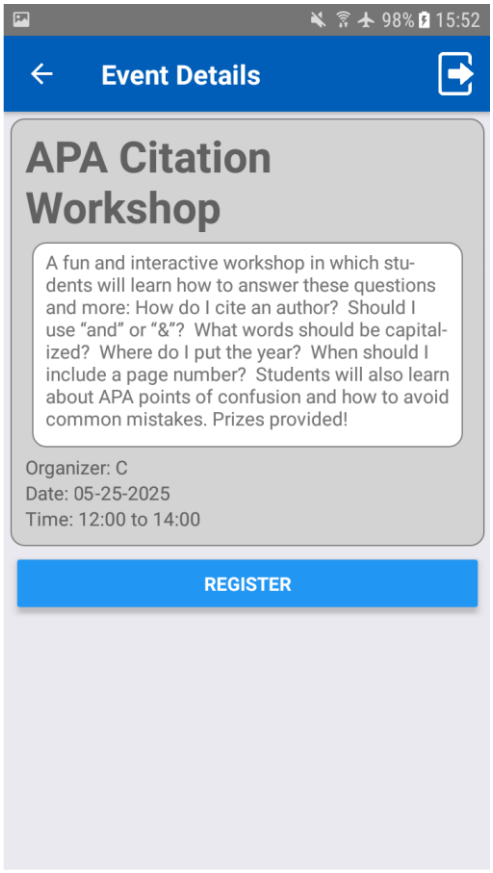


Рисунок 3.7 – Сторінка "Деталі події (незареєстрований користувач)"

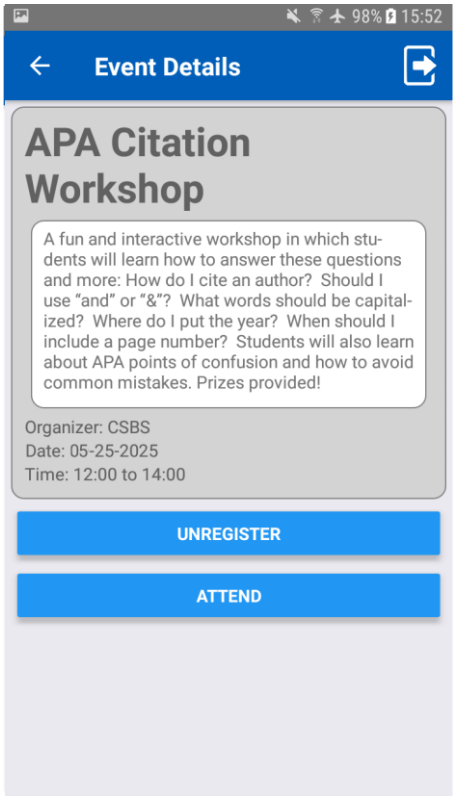


Рисунок 3.8 – Сторінка "Деталі події (зареєстрований користувач)"

- Для модераторів (творців події) замість кнопок реєстрації/скасування реєстрації відображаються кнопки для отримання списку учасників (можливо, для експорту або перегляду) та кнопка для видалення події (рис. 3.9).

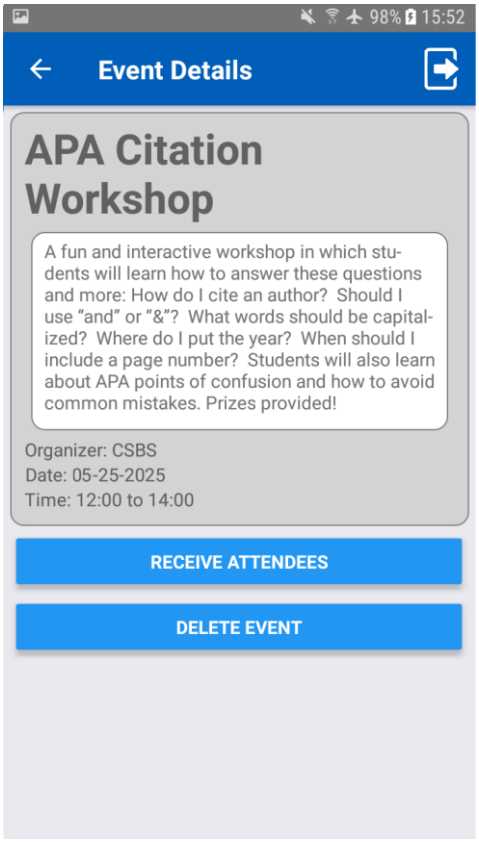


Рисунок 3.9 – Сторінка "Деталі події (модератор)"

Рисунки 3.7 – 3.9 ілюструють різні стани екрану "Деталі події" залежно від ролі та статусу користувача.

4. Екран "Створення Події" (Create Event Screen)

Призначення.

Надання форми для внесення даних модератором з метою реєстрації нової події в системі.

Функціональність.

Екран являє собою інтерактивну форму, що дозволяє модератору ввести всю необхідну інформацію про нову подію, включаючи:

- Назву події.
- Організатора.
- Місце проведення.
- Детальний опис.
- Дату та час події.

Після заповнення форми дані відправляються до системи для збереження.

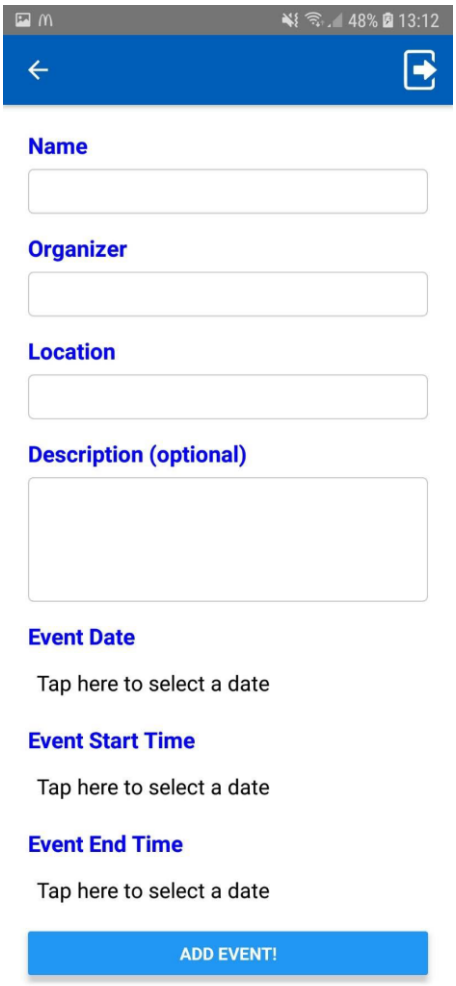


Рисунок 3.10 - Сторінка "Створення події"

Ці екрани забезпечують повний цикл взаємодії з подіями в системі: від огляду та вибору до детального перегляду, реєстрації/управління та створення нових заходів, з адаптацією функціоналу під роль кожного користувача.

3.3. Інтерфейсні екрани для взаємодії з QR-кодами подій

Система передбачає спеціалізовані екрани для взаємодії з QR-кодами, що забезпечують функції відвідування подій та отримання списків учасників.

1. Екран "Відвідування Події" (Event Attendance Screen)

Призначення.

Запуск функціоналу сканування QR-кодів для верифікації присутності учасника на події.

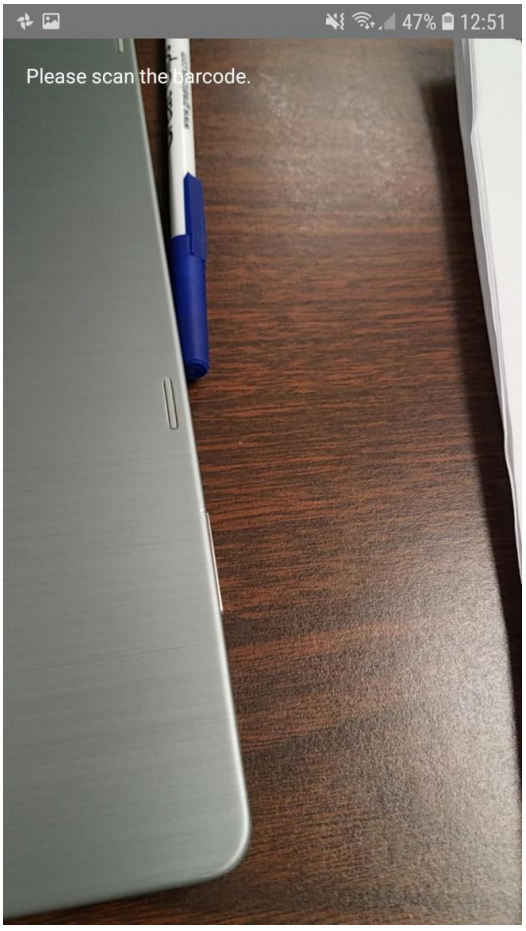


Рисунок 3.11 - Сторінка "Відвідування події"

Функціональність.

Після активації цього екрану ініціюється вбудований сканер QR-кодів (зазвичай за допомогою камери мобільного пристрою). Система аналізує відсканований код. У разі відповідності відсканованого коду унікальному

QR-коду певної події, користувачеві відображається сповіщення, що підтверджує успішне відзначення його присутності. Цей процес є критично важливим для автоматизації контролю доступу та ведення обліку відвідуваності.

2. Екран "Отримання Учасників" (Participant Retrieval Screen)

Призначення.

Відображення згенерованого QR-коду події, який призначений для сканування учасниками з метою підтвердження їхньої реєстрації або відвідування.

Функціональність.

На цьому екрані відображається унікальний QR-код, який був згенерований на основі ідентифікатора (коду) конкретної події. Цей QR-код є засобом для ідентифікації події при скануванні її учасниками. Екран слугує для демонстрації коду, який потім може бути відсканований іншими пристроями або додатками.

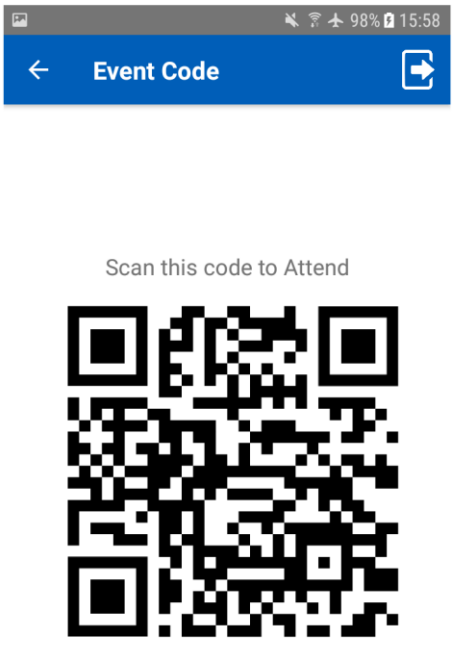


Рисунок 3.12 - Сторінка "Приймання учасників на подію за допомогою QR коду"

Ці екрани підкреслюють інтеграцію QR-кодів як ключового елемента для оптимізації процесів реєстрації, контролю доступу та збору даних про відвідуваність у системі управління подіями.

3.4. Стратегія зберігання даних та структура бази даних

Ефективне функціонування мобільного додатку "Events" критично залежить від доступності інформації про користувачів та події в реальному часі. Для реалізації цієї вимоги було обрано Firebase Realtime Database як основне сховище даних. Вибір Firebase обґрунтований його архітектурою, яка забезпечує синхронізацію даних у реальному часі, що є фундаментальною вимогою для динамічних систем управління подіями.

3.4.1. Ієрархічна структура бази даних

Оскільки Firebase Realtime Database оперує даними у форматі JSON (JavaScript Object Notation), структура бази даних була розроблена ієрархічно. Центральним об'єктом є "cevents", який інкапсулює два основні під-об'єкти верхнього рівня: "users" та "events". Ця структура забезпечує логічне розділення та ефективний доступ до категорій даних.

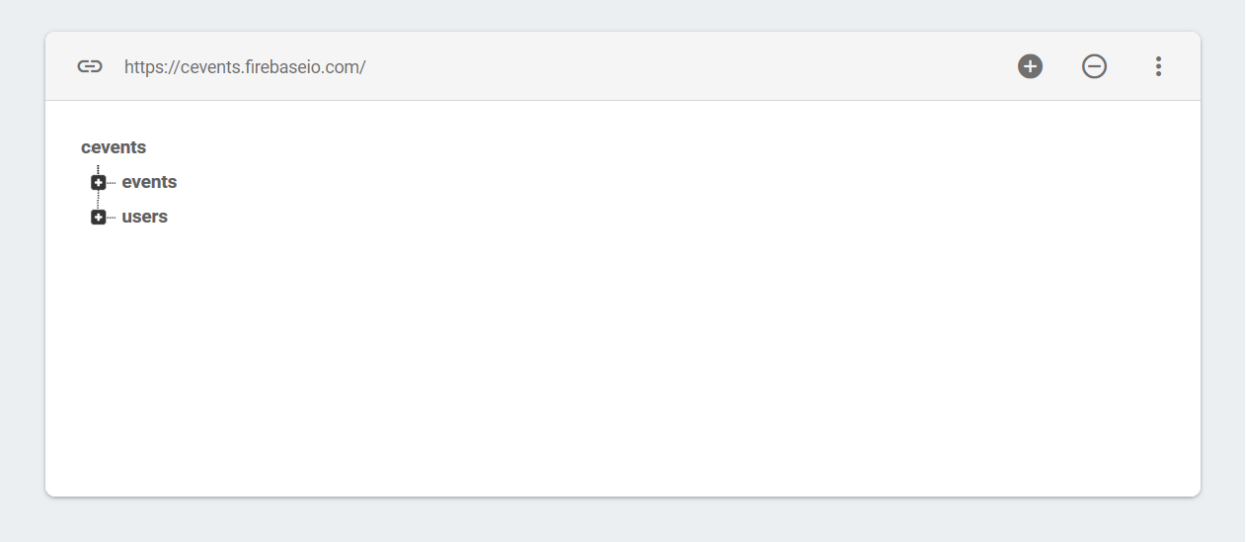


Рисунок 3.13 - Об'єкти бази даних

3.4.2. Атрибути об’єктів бази даних

Об'єкт "users" (Користувачі) призначений для зберігання всієї релевантної інформації про зареєстрованих користувачів системи. Кожен користувач унікально ідентифікується за допомогою свого ID, який функціонує як первинний ідентифікатор для доступу та маніпуляції даними користувача.

Атрибут	Тип даних	Опис
Name	String	Повне ім'я та прізвище користувача.
Dob	String (MM-DD-YYYY)	Дата народження користувача у форматі "місяць-день-рік".
Major	String	Академічна спеціальність користувача.
Moderator	Boolean	Бінарний прапорець, що визначає роль користувача. Значення `true` вказує на роль модератора, `false` — на звичайного користувача.

Рисунок 3.14 - Атрибутивний склад об'єкта users

Атрибут	Тип даних	Опис
Name	String	Назва події.
Code	String	Унікальний ідентифікаційний код події.
CreatorId	String	Ідентифікатор користувача, який створив дану подію.
Desc	String	Детальний опис події.
Organizer	String	Назва відділу або організації, відповідальної за проведення події.
Moderators	Object	Об'єкт, що містить перелік ідентифікаторів всіх модераторів, які мають право вносити зміни до даної події.
When	Object	Об'єкт, що містить інформацію про дату та час проведення події.
Where	String	Місце проведення події.
Attendees	Object	Список, що містить унікальні ідентифікатори всіх користувачів, які зареєструвалися на подію.
Attended	Object	Список, що містить унікальні ідентифікатори всіх користувачів, які фактично відвідали подію.

Рисунок 3.15 - Атрибутивний склад об'єкта events

Об'єкт "events" (Події) - містить повну інформацію про всі події, зареєстровані в системі. Кожній події присвоюється унікальний ключ, автоматично згенерований Firebase, що забезпечує її неповторну ідентифікацію в базі даних.

3.4.3. Компонент управління доступом до даних (Data Access Layer)

Для забезпечення структурованої та централізованої взаємодії з Firebase Realtime Database було реалізовано патерн Одинак (Singleton) у вигляді об'єкта dbData.

Призначення.

Інкапсуляція всієї логіки комунікації з сервером та виконання запитів до бази даних. Це дозволяє уніфікувати точки доступу до даних, зменшити дублювання коду та спростити підтримку системи.

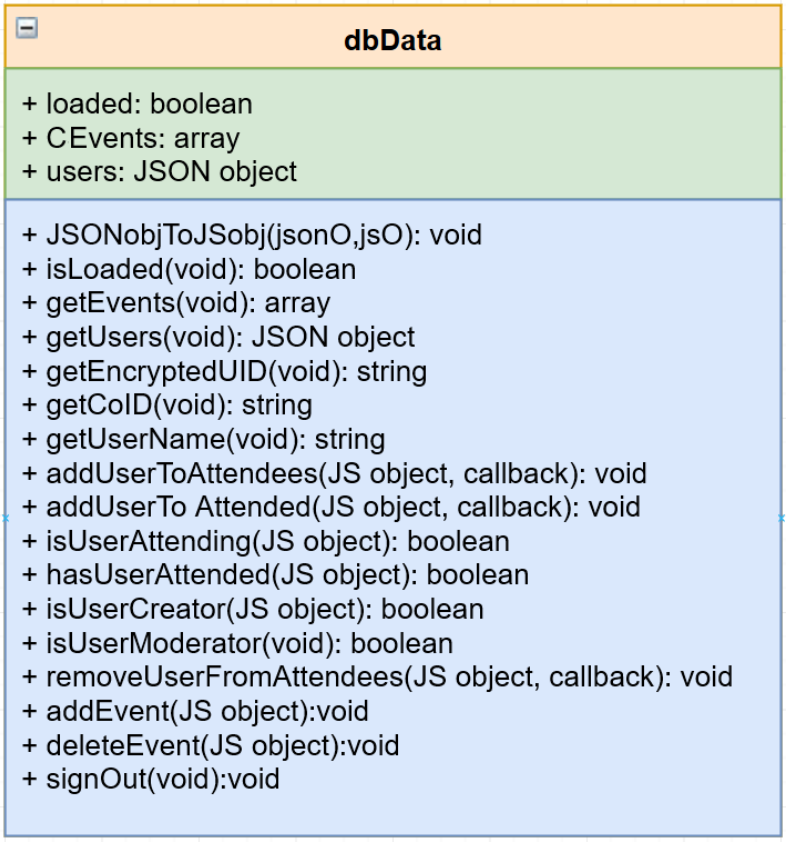


Рисунок 3.15 - Діаграма класу dbData

Функціональність.

Об'єкт dbData є єдиним екземпляром, через який будь-який компонент додатку може ініціювати операції читання, запису, оновлення або видалення даних у Firebase. Такий підхід гарантує цілісність даних та узгодженість взаємодії з базою даних.

Ця стратегія зберігання та доступу до даних забезпечує масштабованість, надійність та реактивність додатку "Events", що є ключовими аспектами для сучасних мобільних систем.

3.5. Тестування сервісу адміністрування та організації подій користувача

У процесі розробки програмного забезпечення, тестування є невід'ємним етапом, якому слід надавати пріоритет нарівні з безпосередньою розробкою. Метою тестування є забезпечення розробки надійного, якісного та безпомилкового програмного продукту, що в перспективі призводить до зниження експлуатаційних витрат. У рамках даного проекту були застосовані наступні методи тестування на всіх етапах життєвого циклу розробки:

3.5.1. Модульне Тестування (Unit Testing)

Модульне тестування являє собою процес верифікації найменших автономних компонентів або "одиниць" програмного коду. Основне завдання — визначити, чи коректно функціонує кожна одиниця, видаючи очікуваний результат при заданих вхідних даних.

У даному проекті модульне тестування проводилося як вручну, так і автоматизовано за допомогою фреймворку Jest для JavaScript. Jest використовує концепцію тестування знімків (snapshot testing), яка є особливо ефективною для тестування компонентів користувацького інтерфейсу.

При першому запуску тесту для певного компонента, Jest створює "знімок" його відрендереного стану (структури). При наступних запусках тесту, Jest повторно рендерить компонент і порівнює новий знімок з раніше збереженим. Будь-які відмінності свідчать про несподівані зміни в інтерфейсі користувача, що може вказувати на регресію або помилку. Цей підхід забезпечує незмінність візуального представлення компонентів.

Нижче представлено приклад виводу автоматизованих тестів Jest, що демонструє успішне створення та порівняння знімків для різних компонентів навігації та інтерфейсу.

Рисунок 3.16 - Результати модульного тестування за допомогою Jest

На рисунку 3.16 показано вивід консолі, що демонструє результати модульних тестів (unit tests) для додатку "Events". Зокрема, показано успішне виконання тестів для різних компонентів навігації (наприклад, attendEvent.js, eventDetails.js, myEvents.js, allEvents.js, addEvent.js, appHomepage.js, login.js) та компонентів інтерфейсу користувача (наприклад, appButton.js, listEvents.js). Також зазначено, що під час цих тестів було записано 8 знімків

(snapshots written) з 8 тестових наборів, а загалом 10 тестів пройшли успішно з 10.

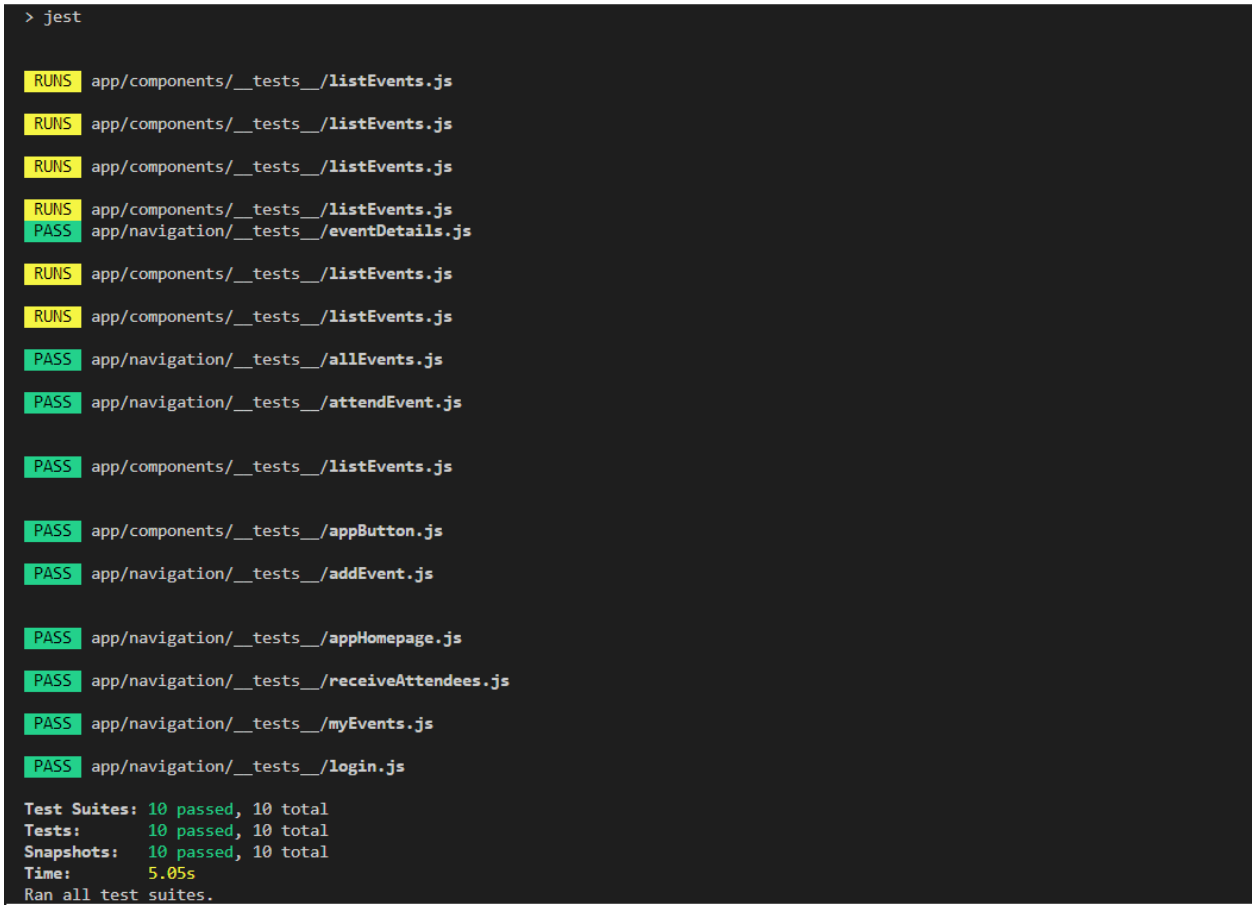


Рисунок 3.17 - Результати модульного тестування за допомогою Jest

На рисунку 3.17 відображено вивід консолі, який показує успішне виконання модульних тестів. Це підтверджується повідомленнями "PASS" для численних файлів, що відповідають компонентам додатку, таким як app/navigation/_tests_/attendEvent.js, app/components/_tests_/appButton.js, app/navigation/_tests_/eventDetails.js тощо.

Також у "Snapshot Summary" зазначено, що "8 snapshots written from 8 test suites" та "10 passed, 10 total" для Test Suites, Tests і Snapshots, що свідчить про успішне тестування функціональності та візуальної стабільності компонентів.

3.5.2 Інтеграційне тестування (Integration Testing)

Інтеграційне тестування зосереджене на перевірці взаємодії між окремими модулями, які пройшли модульне тестування. Метою є виявлення дефектів, що виникають внаслідок некоректної інтеграції компонентів, та забезпечення їх спільного функціонування відповідно до системних вимог.

Інтеграційні тести в даному проекті проводилися вручну. У ході тестування відстежувалася поведінка системи при взаємодії різних модулів. Виявлені аномалії або несподівана поведінка документувалися, виправлялися, після чого виконувалося повторне тестування для підтвердження усунення дефекту та відсутності регресійного ефекту (тобто, що виправлення однієї помилки не призвело до появи нових при взаємодії з іншими компонентами).

3.5.3. Тестування прийняття користувачем (User Acceptance Testing – UAT)

Тестування прийняття користувачем є фінальним етапом тестування, на якому кінцеві користувачі перевіряють систему на відповідність бізнес-вимогам та її придатність для використання в реальних умовах. Успішне проходження UAT є обов'язковою умовою для випуску продукту.

На цьому етапі було обрано групу випадкових користувачів, яким надавалася інформація про функціонал додатку. Альфа-версія додатку встановлювалася на різні мобільні пристрої для цих користувачів. Користувачі взаємодіяли з додатком у типових сценаріях використання та надавали зворотний зв'язок щодо будь-яких несподіваних поведінок, помилок або проблем, з якими вони зіткнулися.

Отримані відгуки ретельно збиралися, аналізувалися. Підтверджені дефекти та проблеми вирішувалися розробниками, після чого оновлена версія додатку повторно надсилалася на UAT. Цей ітераційний процес тривав

до тих пір, поки користувачі не підтвердили відсутність значних проблем та повну відповідність функціоналу їхнім очікуванням.

Таблиця 3.1 - Приклад зворотного зв'язку та виправлень під час User Acceptance Testing

ID користувача	Статус прийому	Роль	Коментар/відгук користувача	Примітки
2	Умовно прийнята	Користувач	Проблема: Сторінка опису події не прокручується вниз, коли опис події надто довгий.	Виправлено: Оновлено компонент EventDetails для використання компонента ScrollView замість компонента View.
5	Умовно прийнята	Модератор	Проблема: Додаток не йде далі екрану завантаження.	Виправлено: Додаток завис, тому що він втратив зв'язок з сервером, оскільки телефон користувача втратив підключення до Інтернету. Ця проблема була вирішена шляхом оновлення тексту на екрані завантаження

Тестування програмного забезпечення здійснюється з високим ступенем ретельності, що забезпечує ідентифікацію та усунення всіх виявлених дефектів і аномалій. Кінцевою метою цього процесу є гарантування того, що фінальна, готова до розгортання версія додатку функціонуватиме згідно з розробленими специфікаціями та очікуваною поведінкою.

Для подальшого підвищення юзабіліті та розширення функціональних можливостей додатку "Events" пропонується впровадження наступних аспектів.

1. Геолокаційні сервіси для визначення місця події.

Інтеграція картографічних сервісів і реалізація можливості для модераторів точно позначати місце проведення події на інтерактивній карті під час її створення. Це дозволить забезпечити точну геолокацію.

Навігація для користувачів, тобто користувачі, що зареєструвалися на подію, зможуть отримувати покрокові навігаційні інструкції від своєї поточної локації до місця проведення заходу. Ця функція особливо корисна для нових відвідувачів, полегшуючи їм орієнтацію та знаходження локацій.

2. Удосконалення методів реєстрації на заходах.

Впровадження Near Field Communication (NFC). Замість існуючої системи реєстрації через QR-коди, що вимагає точного позиціонування камери пристрою відносно коду, пропонується використання технології NFC.

Реєстрація за допомогою NFC здійснюється шляхом простого торкання мобільним пристроєм до зчитувача, незалежно від кута нахилу. Це значно прискорює процес реєстрації учасників, підвищуючи ефективність та зручність.

3. Система сповіщень та нагадувань.

Впровадження механізму push-сповіщень, які інформуватимуть користувачів про появу нових подій на кампусі в режимі реального часу. Це забезпечить актуальну обізнаність користувачів щодо всіх заходів.

Додавання функціоналу нагадувань, які надсилатимуться користувачам у день проведення події, на яку вони зареєстровані. Це допоможе запобігти пропуску подій та покращить планування часу користувачів.

4. Функція обміну подіями з друзями.

Розробка функції, що дозволяє користувачам легко ділитися інформацією про події, які вони планують відвідати, зі своїми контактами.

Зручний доступ до інформації. Завдяки цій функції, друзям не доведеться самостійно шукати подію в додатку; вони зможуть отримати прямий доступ до детальної інформації про захід за посиланням, що значно спростить процес ознайомлення та реєстрації.

Представлений проект фокусується на розробці програмного забезпечення, призначеного для оптимізації процесів управління подіями.

Ключовою перевагою розробленої системи є мінімізація ручного введення даних, що забезпечує значну економію часу як для учасників, так і для організаторів заходів. Система спрощує процедури реєстрації та обліку відвідуваності подій.

Розроблений додаток пройшов всебічне тестування, що підтвердило його стабільну роботу в різних сценаріях використання та відсутність значних програмних помилок.

ВИСНОВКИ

У процесі виконання дипломної роботи на тему «Побудова програмного сервісу адміністрування подій користувача» було здійснено дослідження предметної області, розроблено архітектурні рішення та реалізовано функціональний програмний сервіс, що забезпечує ефективну організацію та адміністрування подій.

На першому етапі роботи проведено аналіз сучасного стану розробки програмних засобів у сфері управління подіями. Визначено передумови створення нового сервісу, зокрема потребу в універсальному інструменті для адміністрування заходів з урахуванням вимог користувачів до зручності, функціональності та адаптивності інтерфейсу. Проаналізовано існуючі рішення на ринку, такі як Bizzabo, Hopin та EventMobi, виявлено їхні переваги й обмеження, що дало змогу обґрунтувати доцільність розробки власного сервісу.

У другому розділі було спроєктовано архітектуру майбутньої системи з використанням сучасних методів моделювання — діаграм потоків даних, випадків використання, послідовності, що забезпечило структуроване представлення логіки взаємодії компонентів. Також були визначені основні користувацькі ролі, навігаційні сценарії та функціональні модулі, які сформували основу інтерфейсу системи.

У третьому розділі здійснено програмну реалізацію сервісу. Розроблено інтерфейси для основних сторінок та екранів взаємодії з QR-кодами подій, реалізовано стратегію зберігання даних з урахуванням ієрархічної структури бази даних та впроваджено шар доступу до даних (Data Access Layer), що забезпечує гнучкість та безпечність у роботі з інформацією. Проведено комплексне тестування (модульне, інтеграційне, тестування прийняття), яке підтвердило функціональну повноту сервісу та його готовність до використання кінцевими користувачами.

Таким чином, у рамках дипломної роботи було успішно реалізовано всі етапи життєвого циклу програмного забезпечення — від аналізу вимог і проєктування до реалізації й тестування. Отриманий результат є повноцінним застосунком, що відповідає сучасним вимогам до систем управління подіями та може бути використаний як у корпоративному, так і в освітньому чи громадському середовищі.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Bizzabo Software Reviews, Demo & Pricing – 2025 - https://www.softwareadvice.com/hybrid-event/bizzabo-profile/
2. Virtual events company Hopin moves to .Com - Domain Name Wire | Domain Name News - https://domainnamewire.com/2020/11/12/virtual-events-company-hopin-moves-to-com/
3. Best Practices for Event App Design | EventMobi - https://www.eventmobi.com/blog/best-practices-for-event-app-design/
4. Pressman, R. S. (2019). Software Engineering: A Practitioner's Approach. McGraw-Hill Education.
5. Myers, G. J., Badgett, T., & Sandler, T. M. (2011). The Art of Software Testing. John Wiley & Sons.
6. Beck, K. (2000). Extreme Programming Explained: Embrace Change. Addison-Wesley.
7. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.
8. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. IEEE Computer Society. IEEE Software..
9. Norde, K. (2019). Fullstack React Native: Build iOS, Android, and Web Apps with JavaScript. Fullstack.io.
10. Hurlburt, J. (2019). Mobile App Development with Flutter: A Developer's Guide. Packt Publishing.
11. Bell, J., & McDonald, S. (2018). Firebase Cookbook: Develop faster and more robust applications with Firebase. Packt Publishing.
12. Sharma, R. (2019). Building Real-time Applications with Firebase. Packt Publishing.

13. Kaner, C., Falk, J., & Nguyen, H. Q. (1999). Testing Computer Software. John Wiley & Sons.
14. Dustin, E., Rashka, D., & Paul, J. (1999). Automated Software Testing: Introduction, Management, and Techniques. Addison-Wesley.
15. Pettichord, B. (2001). Lessons Learned in Software Testing: A Context-Driven Approach. John Wiley & Sons.
16. Osherove, R. (2009). The Art of Unit Testing: With Examples in .NET. Manning Publications.
17. Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code. Addison-Wesley.
18. Whittaker, J. A. (2002). How to Break Software: A Practical Guide to Testing. Addison-Wesley.
19. Jorgensen, A. K. (2013). The Psychology of Software Testing: A Guide for Testers and Developers. Auerbach Publications.
20. Crispin, L., & Gregory, J. (2009). Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley.
21. Gelperin, D., & Hetzel, W. (1988). The Growth of Software Testing. Communications of the ACM, 31(6), 687-695.
22. Lane, N. D., et al. (2010). A survey of mobile phone sensing. IEEE Communications Magazine, 48(9), 140-150.
23. Want, R. (2006). An introduction to NFC. IEEE Pervasive Computing, 5(1), 86-90.
24. Chen, J., et al. (2012). Location-Based Social Networks: An Introduction. IEEE Communications Magazine, 50(9), 118-125.
25. Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner's Approach (9th ed.). McGraw-Hill Education.
26. Fowler, M. (2002). Patterns of Enterprise Application Architecture. Addison-Wesley.

27. Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice (3rd ed.). Addison-Wesley.
28. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
29. Alahmadi, T., & Hussain, F. K. (2021). A comparative study of event management systems: Evaluation and requirements. Journal of Systems and Software, 179, 111027.
30. Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning for event-driven process prediction: A survey. Business & Information Systems Engineering, 63(6), 637–653.
31. Papageorgiou, A., & Papanikolaou, Y. (2018). Event-driven software architectures: A survey and research roadmap. Software: Practice and Experience, 48(6), 1147–1171.
32. Ashraf, J., & Hurson, A. R. (2020). Developing modular event management applications with microservices architecture. Journal of Cloud Computing, 9(1), 1–15.
33. Aljohani, A., & Qureshi, M. R. J. (2020). Web-based event management system: A survey of recent frameworks and tools. International Journal of Web Information Systems, 16(4), 347–362.

ДОДАТКИ

Додаток А

Програмні коди

index.js

```
import React from 'react';
import { AppRegistry } from 'react-native';
import App from './App';
export default AppRegistry.registerComponent('test', () => App);
```

App.js

```
import React, { Component } from 'react';
import { RootSwitch } from './app/navigation/rootSwitch';
import NavigationService from './app/navigation/NavigationService';
export default class App extends Component<{}> {
    render() {
        return (
            <RootSwitch ref={navigatorRef => {
                NavigationService.setTopLevelNavigator(navigatorRef);
            }}/>
        );
    }
}
```

rootSwitch.js

```
import React, { Component } from 'react';
import { RootSwitch } from './app/navigation/rootSwitch';
import NavigationService from './app/navigation/NavigationService';
export default class App extends Component<{}> {
    render() {
        return (
            <RootSwitch ref={navigatorRef => {
                NavigationService.setTopLevelNavigator(navigatorRef);
            }}/>
        );
    }
}
```

appNavigator.js

```
import { createStackNavigator } from 'react-navigation';
import AppHomepage from './appHomepage'
import AllEvents from './allEvents'
import MyEvents from './myEvents'
import EventDetails from './eventDetails'
import AttendEvent from './attendEvent'
import ReceiveAttendees from './receiveAttendees'
import AddEvent from './addEvent.js'
import React from 'react';
import { Image, TouchableOpacity } from 'react-native';
import dbData from '../store/dataStore'
import NavigationService from './NavigationService';
logOut = function()
{
    dbData.signOut();
    NavigationService.navigate('Login');
}
const RootStack = createStackNavigator(
    {
        appHome: AppHomepage,
        allEvents: AllEvents,
        myEvents: MyEvents,
        eventDetails: EventDetails,
        attendEvent: AttendEvent,
        receiveAttendees: ReceiveAttendees,
        addEvent: AddEvent,
    },
    {
        initialRouteName: 'appHome',
        navigationOptions: {
            headerStyle: {
            backgroundColor: '#005eb8',
        },
        headerRight: (
            <TouchableOpacity activeOpacity={0.5} onPress={logOut}
                style={{
                    marginRight: 10,
                    width: 32,
                    height: 32,
                    alignItems: 'center',
                    justifyContent: 'center',
            }}>
                <Image source={require('../assets/images/logout.png')}
                    style={{
                        resizeMode: 'contain',
                        width: 32,
                        height: 32,
                    }} />
                    </TouchableOpacity>
            ),
            headerTintColor: '#fff',
            headerTitleStyle: {
                fontWeight: 'bold',
            },
        },
    }
);
```

myEvents.js (частина)

```
import React, { Component } from 'react';
import { StyleSheet, View, Image, TouchableOpacity, Alert, Text } from 'react-native';
import ListEvents from '../components/listEvents';
import {observer} from 'mobx-react/native';
import firebase from '../firebase.config';
import dbData from '../store/dataStore'
@observer
class MyEvents extends Component {
    static navigationOptions = {
        title: 'My Events',
    };
    navigateTo=()=>{
        this.props.navigation.navigate('addEvent');
    }
    myEvent=(e)=>{
        if(dbData.isUserAttending(e)!==null || dbData.hasUserAttended(e) || dbData.isUserCreato
        return true;
        return false;
    }
    render() {
        const { navigation } = this.props;
        CEvents = navigation.getParam('CEvents', {});
        return (
            <View style={styles.MainContainer}>
                <ListEvents eventList = {CEvents.filter((event) => this.myEvent(event))}
            {dbData.isUserModerator() ?
                <TouchableOpacity
                    activeOpacity={0.5}
                    onPress={this.navigateTo}
                    style={styles.TouchableOpacityStyle}>
                    <Image
                        source={require('../assets/images/add.png')}
                        style={styles.FloatingButtonStyle}
                    />
                </TouchableOpacity>
                : null}
            </View>
        );
    }
}
const styles = StyleSheet.create({
    MainContainer: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
    },
    TouchableOpacityStyle: {
        position: 'absolute',
        width: 50,
        height: 50,
        alignItems: 'center',
        justifyContent: 'center',
        right: 30,
        bottom: 30,
    },
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Побудова програмного сервісу адміністрування подій користувача ”

Обсяг пояснювальної записки: 76 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____