МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Острозька академія»

Економічний факультет

Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістра

на тему: **«СТВОРЕННЯ МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ МОНІТОРИНГУ ВІДВІДУВАНОСТІ СТУДЕНТІВ ТА РОЗКЛАДУ»**

Виконав: студент 2 курсу, групи МУП-21
другого (магістерського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Управління проєктами»
Антонюк Євгеній Михайлович

Керівник: Front-end Developer "DOODLE", LLC,
Місай Володимир Віталійович

Рецензент: кандидат технічних наук, доцент кафедри прикладної математики та кібербезпеки Донецького національного університету імені Василя Стуса,
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О. Р.)

Протокол № ___ від «___» _____ 2024 р.

Острог, 2024

Міністерство освіти і науки України

Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: *122 Комп'ютерні науки*

Освітньо-професійна програма: *Управління проєктами*

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Ольга КРИВИЦЬКА

«___» _____ 20___ р.

ЗАВДАННЯ

на кваліфікаційну роботу студента

Антонюка Євгенія Михайловича

(прізвище, ім'я, по батькові)

1. Тема роботи: Створення мобільного застосунку для моніторингу відвідуваності студентів та розкладу

керівник роботи: Місай Володимир Віталійович, Front-end Developer "DOODLE", LLC,

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

*Затверджено наказом ректора НаУОА від "*03*"* листопада *2023 року №* 98

2. Термін здачі студентом закінченої роботи: 05.12.2024

3. Вихідні дані до роботи: React Native Expo, Firebase Firestore, Android Studio Emulator, React Navigation, QR-code , календар, Figma, Adobe Illustrator, Adobe After Effects

4. Перелік завдань, які належить виконати: налаштування середовища розробки, створення бази даних у Firebase, розробка авторизації та ролей користувачів, реалізація головної сторінки, розробка сторінки розкладу, реалізація функції чату, функціонал відмітки присутності, розробка сторінки профілю, дизайн інтерфейсу, тестування додатку

5. Перелік графічного матеріалу: рисунки, таблиці

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Місай В.В	01.12.23	01.12.23
2	Місай В.В	01.12.23	01.12.23
3	Місай В.В	01.12.23	01.12.23

7. Дата видачі завдання: 01.12.23

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми роботи/проєкту.	жовтень 2023	+
2	Планування та дослідження	листопад 2023	+
3	Проектування інтерфейсу користувача (UI/UX)	грудень 2023	+
4	Проектування структури бази даних Firebase Firestore	січень 2024	+
5	Налаштування проекту в React Native Expo	лютий 2024	+
6	Розробка функціоналу авторизації та входу	березень 2024	+
7	Розробка головної сторінки та функціоналу новин/подій	квітень 2024	+
8	Розробка екрану розкладу та чату	травень-червень 2024	+
9	Розробка функціоналу відмітки присутності (QR-коди)	липень-серпень 2024	+
10	Тестування та виправлення багів	вересень-грудень 2024	+
11	Документування і створення кваліфікаційної роботи	грудень 2024	+
12	Створення презентації і попередній захист кваліфікаційної роботи	грудень 2024	+
13	Публічний захист кваліфікаційної роботи перед екзаменаційною комісією	грудень 2024	+

Студент: _____ Антонюк Є. М.
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи: _____ Місай В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

кваліфікаційної роботи

на здобуття освітнього ступеня магістра

Тема: Створення мобільного застосунку для моніторингу відвідуваності студентів та розкладу

Автор: Антонюк Євгеній Михайлович

Науковий керівник: Місай Володимир Віталійович

Захищена «___» _____ 20___ року.

Пояснювальна записка до кваліфікаційної роботи: 87 *(кількість сторінок роботи) с.,* 16 *(кількість рисунків) рис.,* 10 *(кількість таблиць) табл.,* 0 *(кількість додатків) додатків,* 11 *(кількість джерел) джерел.*

Ключові слова: React Native, Firebase, QR-code, розклад занять, чат, відмітка присутності, мобільний додаток для навчання.

Короткий зміст праці: Кваліфікаційна робота на тему: «Створення мобільного застосунку для моніторингу відвідуваності студентів та розкладу» присвячена розробці мобільного додатку для студентів та викладачів на платформі Android.

Додаток забезпечує швидкий доступ до новин академії, розкладу занять, чату для студентів та викладачів, а також функції відмітки присутності за допомогою QR-кодів. Він адаптований для різних ролей користувачів, що забезпечує персоналізований досвід для студентів і викладачів.

Основні функції додатку включають:

1. **Головна сторінка:** Відображення новин академії та актуальних подій для студентів та викладачів.
2. **Перегляд розкладу:** Студенти можуть переглядати свій персональний розклад занять, а викладачі — тільки ті предмети, які вони викладають.
3. **Чат:** Можливість спілкування між студентами та викладачами.

4. **Відмітка присутності:** Викладачі можуть генерувати QR-коди для кожної пари, а студенти сканують їх для реєстрації присутності. Викладачі мають доступ до списку присутніх.
5. **Профіль:** Доступ до особистих даних користувача, перегляд розкладу дзвінків, корисні посилання та можливість виходу з акаунту.

Для розробки додатку були використані сучасні технології: мова програмування JavaScript, середовище розробки Microsoft Visual Code, а також графічні редактори Figma, Adobe Illustrator та Adobe After Effects. База даних реалізована у Firebase Firestore для ефективного зберігання та обробки даних. Основою для створення додатку є React Native Expo, що дозволяє ефективно розробляти, тестувати та публікувати додатки для iOS та Android без потреби глибоких знань у нативному коді.

Результатом роботи є багатофункціональний додаток, що полегшує організацію навчального процесу та покращує взаємодію між студентами та викладачами.

(підпис автора)

ABSTRACT

of the qualification work

for obtaining a master's degree

Topic: Creating a mobile application for monitoring student attendance and schedule

Author: Antoniuk Yevhen Mykhailovych

Supervisor: Misai Volodymyr Vitaliiovych

Defended «___» _____ 20___ year.

Explanatory note to the qualification work: 87 *(number of pages) p.*, 16 *(number of figures) figures,* 10 *(number of tables) tables,* 0 *(number of appendices) appendices,* 11 *(number of sources) sources.*

Keywords: React Native, Firebase, QR-code, class schedule, chat, presence marker, mobile app for learning.

Summary of work: The qualification work on the topic: “Creating a mobile application for monitoring student attendance and schedules” is devoted to the development of a mobile application for students and teachers on the Android platform.

The application provides quick access to the academy's news, class schedules, chat for students and teachers, as well as the function of marking presence using QR codes. It is adapted for different user roles, providing a personalized experience for students and teachers.

The main features of the application include:

1. **Home page:** Displays Academy news and current events for students and teachers.
2. **Viewing the schedule:** Students can view their personalized class schedule, and teachers can view only the subjects they teach.
3. **Chat:** Possibility of communication between students and teachers.
4. **Attendance mark:** Instructors can generate QR codes for each pair and students scan them to register presence. Teachers have access to the list of attendees.

5. **Profile:** Access to the user's personal data, view the call schedule, useful links, and the ability to log out of the account.

Modern technologies were used to develop the application: JavaScript programming language, Microsoft Visual Code development environment, as well as Figma, Adobe Illustrator, and Adobe After Effects graphic editors. The database is implemented in Firebase Firestore for efficient data storage and processing. The basis for creating the application is React Native Expo, which allows you to efficiently develop, test, and publish applications for iOS and Android without the need for deep knowledge of native code.

The result is a multifunctional application that facilitates the organization of the educational process and improves interaction between students and teachers.

(author's signature)

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API - Application Programming Interface (Інтерфейс програмування додатків): Набір функцій і процедур, які дозволяють програмам взаємодіяти між собою. API визначає, як програмні компоненти повинні взаємодіяти та обмінюватися даними.

JSON - JavaScript Object Notation (Формат обміну даними): Легкий формат для зберігання та передачі даних. Використовується для обміну інформацією між сервером та клієнтом в інтернет-додатках.

UI - User Interface (Користувацький інтерфейс): Взаємодія між користувачем і програмою або системою. UI включає в себе дизайн елементів інтерфейсу, як-от кнопки, поля введення, меню тощо.

UX - User Experience (Досвід користувача): Враження користувача від використання продукту, зокрема мобільного додатка чи вебсайту. Включає в себе зручність інтерфейсу, доступність функцій і загальний комфорт користування.

QR-код - Quick Response Code (Швидкий код для зчитування): Двовимірний штрихкод, що містить інформацію, яка може бути зчитана мобільним пристроєм. Використовується для швидкого доступу до вебсайтів, документів або даних.

npm - Node Package Manager (Управління пакетами для Node.js): Система для встановлення та управління бібліотеками і залежностями в проекті на JavaScript.

ПЗ - (Програмне забезпечення): Це загальний термін, який використовується для опису комп'ютерних програм, які призначені для виконання конкретних завдань або функцій на комп'ютері чи інших електронних пристроях. Програмне забезпечення може бути розроблено для різних цілей, включаючи бізнес, науку, освіту, розваги та інші сфери діяльності. Відомі приклади програмного забезпечення включають операційні системи, текстові редактори, антивірусні програми, графічні редактори, браузери та багато іншого.

ВСТУП

Актуальність - дослідження полягає в розв'язанні проблеми ефективного управління навчальним процесом та спрощення доступу до важливої інформації для студентів і викладачів. У сучасних умовах, коли все більше навчальних закладів переходять на онлайн чи змішане навчання, важливим аспектом стає оптимізація процесів, пов’язаних із розкладом занять, відміткою присутності студентів та оперативним обміном інформацією між учасниками освітнього процесу.

Розробка мобільного додатку, що дозволяє автоматизувати ці процеси, є актуальним і необхідним кроком у сучасному освітньому середовищі. Використання такого додатку допомагає не лише покращити комунікацію між студентами та викладачами, але й зменшити навантаження на адміністрацію, забезпечуючи швидкий доступ до розкладу, новин академії, а також простоту в організації процесу відмітки присутності. Враховуючи тенденцію до все більшого використання мобільних пристроїв у повсякденному житті, поширення мобільних додатків для освіти буде лише зростати. Смартфони стають основними інструментами для доступу до інформації, і дедалі більше користувачів віддають перевагу мобільним додаткам, замість традиційного використання комп'ютерів. Це робить мобільні додатки важливим елементом у процесі навчання, дозволяючи студентам та викладачам завжди мати під рукою потрібну інформацію та підтримувати постійний зв'язок.

Метою розробки мобільного додатку є реалізація інтуїтивно зрозумілого та ефективного інструменту для моніторингу відвідуваності студентів, перегляду розкладу занять, а також організації комунікації між студентами та викладачами. Додаток має забезпечити простоту використання, інтеграцію з існуючими системами навчальних закладів і сприяти зручному доступу до всіх необхідних функцій для ефективного навчання та управління процесом відвідуваності. Таким чином, розробка такого додатку є відповідає вимогам часу та забезпечує студентам і викладачам зручні інструменти для організації навчання, моніторингу відвідуваності та взаємодії в реальному часі.

Мета дослідження - полягає в розробці мобільного додатку для моніторингу відвідуваності студентів та управління розкладом занять, який забезпечує зручний доступ до важливої інформації для студентів і викладачів на платформі Android. Розробка додатку має на меті вирішення проблеми неефективного управління навчальним процесом, зокрема в частині моніторингу відвідуваності студентів, перегляду розкладу занять та комунікації між студентами і викладачами. Метою є створення інтуїтивно зрозумілого інтерфейсу, що дозволяє студентам та викладачам ефективно взаємодіяти, переглядати актуальний розклад, отримувати новини академії, а також здійснювати відмітку присутності за допомогою QR-кодів. Додаток має сприяти оптимізації навчального процесу та підвищенню ефективності роботи як студентів, так і викладачів.

Додаток буде містити низку ключових функцій, таких як перегляд розкладу занять, де студенти зможуть бачити свій індивідуальний розклад, а викладачі — лише свої заняття. Крім того, додаток включатиме чат для комунікації між студентами та викладачами, можливість отримувати новини та події академії, а також систему відмітки присутності через сканування QR-кодів, що спрощує реєстрацію відвідуваності.

Ці функції дозволяють не тільки оптимізувати організацію навчання, але й покращити взаємодію між учасниками навчального процесу.

Задачі дослідження:

- Аналіз предметної області;
- Проектування інформаційної системи;
- Вибір інструментарію, методів реалізації та тестування ПЗ.

Об'єктом дослідження - є мобільний додаток для моніторингу відвідуваності студентів та управління розкладом занять, який використовується студентами та викладачами для організації навчального процесу. Додаток передбачає автоматизацію процесів відмітки присутності, перегляду розкладу занять, а також

комунікації між учасниками освітнього процесу через інтегровані функції чату та новин академії.

Предметом дослідження - є процес розробки мобільного додатку для моніторингу відвідуваності студентів, перегляду розкладу занять та організації комунікації між студентами і викладачами. Дослідження охоплює технології, інструменти та методи розробки, такі як React Native Expo, Firebase, а також проектування користувацького інтерфейсу, впровадження функціональності для відмітки присутності через QR-коди та реалізацію інших ключових можливостей додатку для покращення взаємодії в навчальному процесі.

Для досягнення поставленої мети необхідно розв'язати такі задачі:

- проаналізувати існуючі методи, підходи та інструменти для оцінки ефективності використання мобільних додатків в освітньому процесі;
- розробити модель оцінки ефективності використання додатку, а також визначити критерії покращення процесу розробки та вихідного коду додатку;
- розробити програмний модуль для взаємодії з QR-кодами, що забезпечить реєстрацію відвідуваності, а також модулі для авторизації, розкладу, чату та новин академії;
- реалізувати інтеграцію між усіма функціональними модулями додатку для забезпечення їх взаємодії та зручного користування;
- перевірити розроблені теоретичні положення, методи, алгоритми та інструменти на практиці через тестування додатку, вивчення відгуків користувачів та оцінку його ефективності в реальних умовах навчального процесу.

Методи дослідження:

- Аналіз літератури та існуючих рішень - для вивчення сучасних підходів до розробки мобільних додатків для освітнього процесу, а також методів моніторингу відвідуваності та управління розкладом. Аналіз існуючих додатків

допоможе визначити найкращі практики та адаптувати їх для конкретного проекту.

- Метод моделювання - для створення теоретичних моделей взаємодії між користувачами додатку (студентами, викладачами) та основними функціями (розклад, чат, відмітка присутності, новини). Моделювання дозволить ефективно спланувати структуру додатку і визначити його функціональні вимоги.
- Метод розробки програмного забезпечення - використання сучасних інструментів і технологій для розробки мобільного додатку, таких як React Native, Firebase, для створення функціональних модулів і забезпечення інтеграції всіх компонентів.
- Метод тестування - проведення тестування розробленого додатку для перевірки його працездатності, ефективності та зручності використання. Це включає тестування функцій, таких як реєстрація відвідуваності через QR-коди, перегляд розкладу та новин, а також комунікація через чат.
- Метод порівняльного аналізу - порівняння ефективності розробленого додатку з іншими існуючими рішеннями для управління навчальним процесом і відмітки присутності в освітніх установах.

РОЗДІЛ 1. ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Опис предметного середовища (функціональної моделі, процесу діяльності)

Для реалізації мобільного застосунку «OA Engage», який призначений для моніторингу відвідуваності студентів та управління розкладом занять, було обрано низку технологій та інструментів, які дозволяють забезпечити високу ефективність, зручність та надійність роботи додатку. Ці технології включають:

React Native - це фреймворк для розробки мобільних додатків, який дозволяє створювати застосунки для iOS та Android, використовуючи JavaScript і React. Він дозволяє писати більшість коду одним разом, використовуючи веб-технології (JavaScript, JSX), а потім компілювати його в нативний код для кожної платформи.

Переваги React Native:

- Крос-платформеність: Ви можете писати один код для двох платформ (iOS і Android), що значно скорочує час і витрати на розробку.
- Швидкий розвиток: Завдяки гарячій перезагрузці (Hot Reloading) зміни можна відразу бачити без необхідності перезавантажувати додаток. Це пришвидшує процес розробки.
- Спільнота та бібліотеки: Велика і активна спільнота розробників та величезна кількість сторонніх бібліотек допомагають вирішити більшість завдань.
- Нативна продуктивність: React Native дозволяє виконувати нативний код для важких задач, таких як обробка зображень або відео, що забезпечує високу продуктивність.
- Можливість інтеграції з нативним кодом: Ви можете використовувати нативні модулі для додавання специфічних функцій, які не підтримуються за замовчуванням.
- Зручний для веб-розробників: React Native побудований на основі React, тому веб-розробники, знайомі з React, можуть швидко адаптуватися до мобільної розробки.

Недоліки React Native:

- Обмеження в продуктивності: Для дуже складних або ресурсоємних додатків, таких як ігри або додатки з високими вимогами до графіки, продуктивність React Native може бути нижчою за нативні рішення.
- Невідповідність між платформами: Іноді деякі елементи інтерфейсу або функціональність можуть поводитися по-різному на Android і iOS, що вимагає додаткових налаштувань для кожної платформи.
- Залежність від нативних модулів: Для використання специфічних функцій інколи потрібно створювати або інтегрувати нативні модулі, що додає складності в розробку.
- Проблеми з оновленнями: Оскільки React Native активно оновлюється, інколи оновлення можуть викликати сумісність з іншими бібліотеками або новими функціями, що потребує додаткових зусиль.
- Більша складність для новачків: Знання JavaScript і React - це лише основа. Для повноцінного використання React Native потрібно розуміти мобільні платформи (iOS та Android), що може бути складним для новачків у мобільній розробці.

Загалом, React Native підходить для більшості додатків, але може мати обмеження для специфічних або високопродуктивних задач.

Expo - це фреймворк і платформа для розробки мобільних додатків на основі React Native, яка спрощує процес створення, налагодження та розгортання додатків. Expo забезпечує інструменти, бібліотеки та сервіси, що дозволяють уникнути роботи з нативним кодом, а також пропонує готове середовище для розробки.

Переваги Expo:

- Expo надає готове середовище розробки "з коробки". Немає необхідності встановлювати Android Studio чи Xcode для початку роботи.
- Можна миттєво протестувати додаток на фізичних пристроях за допомогою застосунку Expo Go без складання окремих білдів.

- Підтримка Hot Reloading і Fast Refresh дозволяє швидко бачити результати змін у коді.
- Expo містить багато корисних API "з коробки", наприклад, для роботи з камерою, геолокацією, push-сповіщеннями, медіафайлами та іншими функціями.
- Додатки, створені з Expo, працюють на iOS і Android без потреби адаптувати код під кожну платформу.

Недоліки Expo:

- У стандартній конфігурації Expo не дозволяє додавати нативний код або використовувати сторонні бібліотеки, які потребують модифікації нативного коду. Для цього доведеться "вийти" з Expo через Eject, що збільшує складність розробки.
- Expo додає в додаток всі свої залежності, навіть якщо ви використовуєте лише частину функцій. Це збільшує розмір білда.
- Якщо ваш додаток потребує унікальних нативних функцій або інтеграцій, які не підтримуються Expo, вам доведеться перейти на bare workflow або використовувати нативний React Native.
- Ваш додаток залежить від бібліотек і сервісів Expo, і якщо якась функція не підтримується (або виникають проблеми з оновленнями), це може обмежити можливості додатка.
- Хоча продуктивність додатків на Expo зазвичай хороша, деякі функції можуть працювати повільніше порівняно з повністю нативними рішеннями.

Expo-camera - це бібліотека в екосистемі Expo, яка надає функціонал для інтеграції камери в мобільні додатки. Вона дозволяє розробникам легко працювати з камерою пристрою, робити фото, записувати відео, сканувати QR-коди, штрих-коди, розпізнавати обличчя та виконувати інші задачі, пов’язані з камерою.

Переваги expo-camera:

- Надає зрозумілий API для інтеграції функцій камери без потреби писати нативний код.
- Працює на iOS і Android з єдиним кодом, що спрощує розробку.
- Підтримує базові та розширені функції, такі як фото, відео, зум, автоспалах, фокусування.
- Легко поєднується з бібліотеками для обробки зображень, як-от expo-image-manipulator.
- Має вбудовану підтримку розпізнавання штрих-кодів, QR-кодів та облич.
- Дозволяє налаштовувати спалах, тип камери (фронтальна чи основна), баланс білого, співвідношення сторін тощо.
- Можна швидко інтегрувати за допомогою команди npm і expo-camera.

Недоліки expo-camera:

- Працює найкраще в середовищі Expo. У разі використання Bare Workflow можуть виникнути додаткові налаштування.
- Не підтримує специфічні функції камери, які можуть бути доступні через нативні SDK. Наприклад, деякі унікальні функції камер пристроїв Samsung або iPhone.
- Не підходить для проєктів, які потребують роботи з нативними API камер на високому рівні (наприклад, для професійної обробки зображень чи відео).
- У великих додатках або на старих пристроях може працювати повільніше, ніж нативні рішення.

Firebase - це платформа для розробки веб- та мобільних додатків, створена Google. Вона надає набір інструментів та сервісів для бекенд-розробки, аналітики, розгортання, моніторингу та інтеграції з користувачами. Firebase допомагає розробникам швидко створювати високоякісні застосунки, не витрачаючи час на складну серверну інфраструктуру.

Переваги Firebase:

- Надає готовий бекенд, що значно зменшує час розробки.

- Підтримує Android, iOS, веб-додатки, Unity та Flutter.:
- Firebase автоматично масштабуються залежно від кількості користувачів.
- Зрозумілі SDK та детальна документація полегшують інтеграцію.
- Інструменти для налаштування політик доступу до даних, аутентифікації користувачів і захисту даних.
- Немає необхідності підтримувати фізичні сервери, все працює у хмарі.
- Firebase пропонує безкоштовний план, який підходить для невеликих додатків або прототипів.

Недоліки Firebase:

- Ви повністю залежите від сервісів Google, і якщо виникнуть перебої, ваш застосунок також може постраждати.
- Безкоштовний тарифний план обмежений за обсягом запитів, зберіганням даних і кількістю користувачів.
- На етапі масштабування вартість може суттєво зрости, особливо якщо додаток використовує багато функцій Firebase.
- Firestore та Realtime Database мають обмеження на складність запитів, що може викликати труднощі з деякими задачами.
- Firebase пропонує багато готових рішень, але в них є обмеження. Наприклад, якщо потрібні специфічні налаштування, можливості можуть бути недостатніми.

AsyncStorage - це бібліотека для зберігання даних на локальному пристрої у додатках на React Native. Вона забезпечує простий асинхронний інтерфейс для зберігання та отримання ключ-значення даних, що дозволяє зберігати інформацію навіть після перезавантаження додатка. Це зручний інструмент для реалізації функціоналу, такого як кешування, токени автентифікації або локальні налаштування користувача.

Переваги AsyncStorage:

- Інтуїтивний API для роботи з локальними даними.

- Не потребує складної конфігурації.
- Забезпечує збереження даних навіть після закриття програми.
- Однакова функціональність на Android та iOS.
- Не перевантажує додаток, використовуючи внутрішнє сховище пристрою.
- Підтримується великим ком'юніті, що робить її стабільною та надійною.

Недоліки AsyncStorage:

- Дані зберігаються у внутрішньому сховищі пристрою, яке має обмеження (близько 6MB для iOS і близько 50MB для Android).
- Не призначена для зберігання великих обсягів даних, таких як зображення чи файли.
- Час читання/запису може збільшуватися при роботі з великими обсягами ключів.
- Підходить лише для ключ-значення. Для складних запитів або реляційних даних краще використовувати бази даних, наприклад, SQLite.
- Зберігання локальних даних не підходить для додатків, які потребують синхронізації між пристроями або з сервером.
- Не забезпечує високого рівня захисту даних. Чутливі дані краще зберігати у спеціалізованих сховищах, таких як SecureStore або Keychain.

react-native-reanimated - це потужна бібліотека для створення анімованих інтерфейсів у React Native. Вона надає API, що дозволяє створювати високопродуктивні, складні та плавні анімації, які виконуються на стороні UI (JavaScript Bridge не використовується), забезпечуючи відмінну продуктивність.

Переваги react-native-reanimated:

- Завдяки виконанню анімацій на стороні UI thread забезпечується плавність навіть у складних сценаріях.
- Підтримується як на Android, так і на iOS.
- Легко поєднується з react-native-gesture-handler для створення взаємодій із користувачем.

- Можна створювати як прості анімації (fade, move), так і складні сценарії з декількома параметрами.
- Відкриває доступ до нативних API для створення ефектів із точністю до фрейму.
- Зручне використання у функціональних компонентах через хуки (useAnimatedStyle, useSharedValue).

Недоліки react-native-reanimated:

- Для початківців API може здатися складним, особливо через концепції worklets, shared values та стилів.
- У порівнянні з іншими бібліотеками, наприклад, Animated, потребує більше конфігурацій у Bare Workflow (зокрема, використання Hermes і Babel).
- Через виконання анімацій у UI thread, відстежувати помилки чи налагоджувати може бути складніше.
- У порівнянні з іншими анімаційними бібліотеками, react-native-reanimated є досить великою за розміром.
- Можуть виникати проблеми сумісності з іншими бібліотеками, якщо вони працюють через JavaScript Bridge.

react-native-qrcode-svg - це бібліотека для створення QR-кодів у додатках на React Native. Вона генерує QR-коди у вигляді SVG-графіки, що робить їх чіткими та масштабованими без втрати якості. Бібліотека легка у використанні та дозволяє кастомізувати зовнішній вигляд QR-кодів.

Переваги react-native-qrcode-svg:

- Просте встановлення та швидке використання без додаткових залежностей.
- Масштабовані QR-коди, які залишаються чіткими на будь-якому пристрої та розмірі екрану.
- Можливість змінювати кольори фону та точок QR-коду, а також додавати зображення (наприклад, логотип) у центр.
- Однаково працює на iOS та Android.

- Мінімалістична бібліотека, яка не додає значного навантаження на розмір додатка.
- QR-коди генеруються з високою точністю, що забезпечує їх легке сканування.

Недоліки react-native-qrcode-svg:

- Підходить лише для створення QR-кодів, а для сканування потрібна окрема бібліотека (наприклад, react-native-camera або expo-barcode-scanner).
- Використовує бібліотеку react-native-svg, що може додати зайву складність у проєкти, які її не використовують.
- Не підтримує складніші функції, як-от валідація даних або вбудоване кодування інформації.
- Іноді виникають проблеми сумісності зі старими версіями React Native або react-native-svg.
- Може знадобитися попереднє кодування даних перед створенням QR-коду.

Figma - це хмарний інструмент для створення дизайнів, прототипів і спільної роботи над проєктами. Завдяки хмарному підходу, він дозволяє дизайнерам, розробникам і менеджерам працювати в реальному часі, обмінюючись правками, коментарями та ідеями. Figma підходить для UI/UX-дизайну, створення прототипів мобільних додатків, вебсайтів, ілюстрацій, інтерактивних елементів і навіть розробки дизайнів для друку.

Переваги Figma:

- Доступ із будь-якого місця та пристрою без потреби встановлення програм.
- Одночасне редагування проєктів кількома користувачами з можливістю коментування та відстеження змін.
- Працює на Windows, macOS, Linux і навіть у браузері.
- Надається безкоштовний план із базовими можливостями, достатніми для малих команд і фрилансерів.
- Інтуїтивно зрозумілий інтерфейс, який дозволяє швидко розпочати роботу навіть новачкам.

Недоліки Figma:

- Робота в хмарі означає, що без підключення до Інтернету доступ обмежений (хоча є офлайн-режим для десктопного додатка, але з обмеженим функціоналом).
- На великих проєктах із численними кадрами (frames) можуть виникати затримки.
- Офлайн-режим підтримується тільки в десктопному додатку, але з мінімальною функціональністю.
- Безкоштовний план підходить для малих команд, але для великих команд потрібен платний тариф.
- Дані зберігаються в хмарі, що може бути проблемою для конфіденційних проєктів.

Функціональна модель додатку: мобільний додаток «OA Engage» складається з кількох основних функціональних блоків, кожен з яких відповідає за виконання конкретних завдань у процесі навчальної діяльності:

1. **Розклад занять:** Студенти мають доступ до перегляду свого індивідуального розкладу, що дозволяє їм планувати своє навчання. Викладачі, в свою чергу, можуть переглядати тільки розклад тих занять, які вони ведуть. Дані про розклад зберігаються в базі даних Firebase, що дозволяє легко синхронізувати і оновлювати інформацію на всіх пристроях.
2. **Відмітка присутності:** Викладач генерує QR-код для кожного заняття, який студенти повинні сканувати під час входу на пару. Це дозволяє автоматично реєструвати присутність студентів на занятті. Система обробки QR-кодів інтегрована в додаток і працює на основі камери мобільного пристрою.
3. **Чат для студентів та викладачів:** Для зручної комунікації між студентами та викладачами реалізована система чату. Це дозволяє обмінюватися повідомленнями, задавати питання по заняттях, обговорювати важливі теми в рамках навчального процесу. Чат також інтегровано з Firebase, що дозволяє в реальному часі отримувати нові повідомлення.

4. **Новини та події академії:** У додатку є окремий розділ, в якому публікуються новини та важливі події академії. Це може включати інформацію про майбутні заходи, зміни в розкладі, конкурси та інші ініціативи.
5. **Профіль користувача:** Кожен користувач (студент або викладач) має персональний профіль, в якому зберігаються його дані (ім'я, прізвище, електронна пошта, роль, тощо). У профілі також є доступ до функцій виходу з додатку, налаштувань та інших допоміжних функцій.

Процес діяльності користувачів:

1. **Студент:**
    - Після входу в додаток студент може переглядати свій розклад занять.
    - За допомогою камери смартфона студент сканує QR-код, щоб відмітити свою присутність на парі.
    - Студент має доступ до чату для спілкування з викладачами та одногрупниками.
    - Може переглядати новини та події академії.
2. **Викладач:**
    - Викладач має доступ лише до розкладу тих занять, які він веде.
    - Генерує QR-коди для відмітки присутності студентів на своїх парах.
    - Переглядає список відвідуваності студентів.
    - Спілкується з студентами та колегами через чат.
    - Може переглядати новини та події академії.

1.2. Огляд аналогів успішних сценаріїв

Для кращого розуміння потреб користувачів та оцінки можливих функціональних рішень було проведено детальний огляд існуючих аналогічних програмних продуктів, що надають можливість автоматизувати та оптимізувати процеси управління навчальним процесом. Основними аспектами для аналізу стали

функції, що використовуються для моніторингу відвідуваності студентів, управління розкладом, комунікації та організації освітніх процесів.

1. Google Classroom - це популярна платформа для організації навчального процесу, яку використовують у багатьох навчальних закладах. Вона надає викладачам зручний інструмент для створення курсів, роздачі завдань, взаємодії з учнями та оцінювання їхніх робіт (Рис. 1.1).

- **Функції:**
  - Створення курсів та управління навчальним контентом.
  - Видача та оцінка завдань із можливістю надання коментарів та зворотного зв’язку.
  - Інтеграція з іншими сервісами Google, такими як Google Docs, Google Drive та Google Meet.
  - Організація спільного доступу до документів для обміну інформацією між викладачами та студентами.
- **Переваги:**
  - Простий у використанні інтерфейс та інтеграція з іншими продуктами Google.
  - Ефективне управління навчальними матеріалами та завданнями.
  - Реалізація функціоналу отримання результатів у режимі реального часу.
  - Можливість масштабування системи для підтримки великої кількості учасників.
- **Недоліки:**
  - Відсутність функцій моніторингу фізичної присутності студентів.
  - Обмежений функціонал для управління розкладом занять та відсутність підтримки QR-кодів для відмітки присутності.
  - Недостатня інтеграція з іншими освітніми процесами, такими як управління освітнім контентом або організація студентських подій.

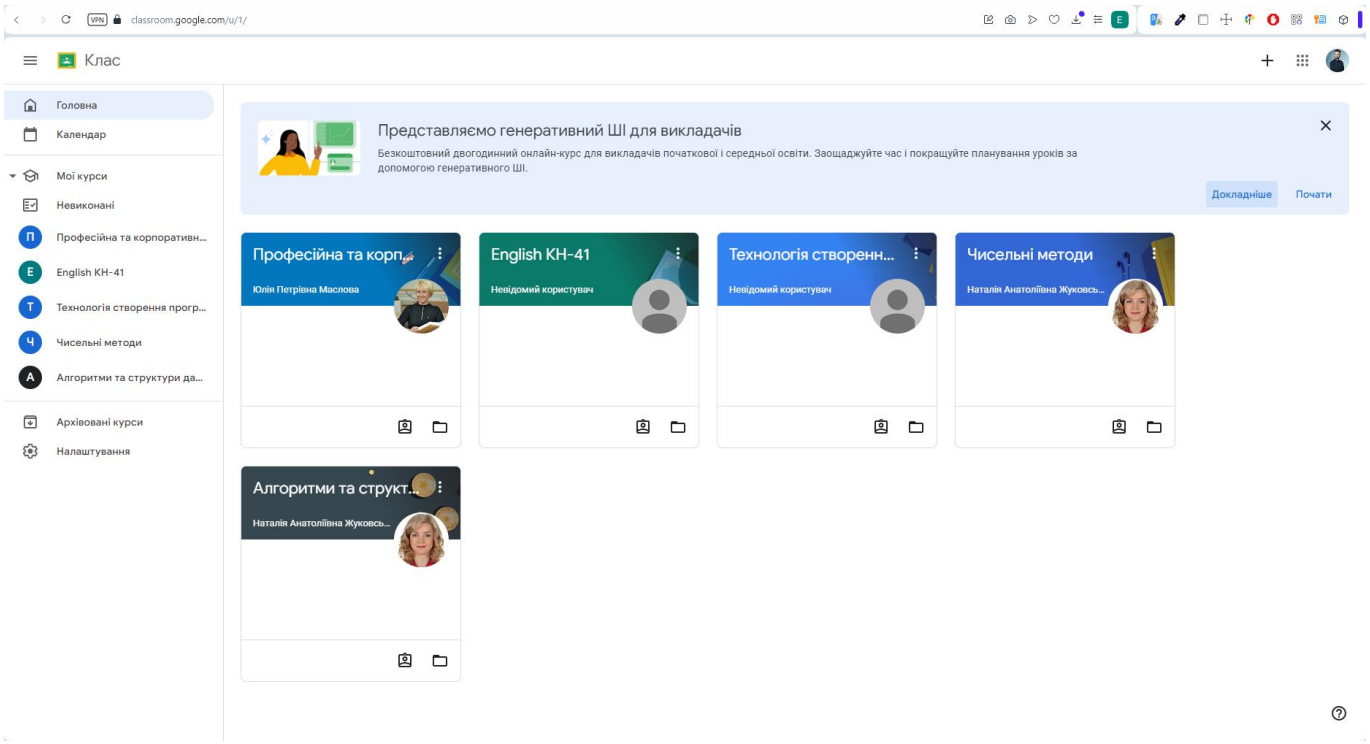

Рис. 1.1. Зовнішній вигляд додатку Google Classroom

Джерело: https://sites.google.com/view/classroom-workspace/

2. Moodle - це відкрите програмне забезпечення для управління навчанням (LMS), яке дозволяє викладачам створювати курси, роздавати завдання, проводити тести та відстежувати прогрес студентів (Рис. 1.2).

- **Функції:**
  - Створення та управління курсами та тестами.
  - Інструменти для оцінювання та зворотного зв'язку.
  - Можливість моніторингу прогресу студентів, завдань та результатів.
- **Переваги:**
  - Можливість масштабування на великі установи.
  - Широкий набір функцій для управління навчальними процесами.
- **Недоліки**:
  - Інтерфейс складний для нових користувачів, що може утруднити роботу зі студентами та викладачами, особливо для тих, хто не має досвіду в використанні LMS.

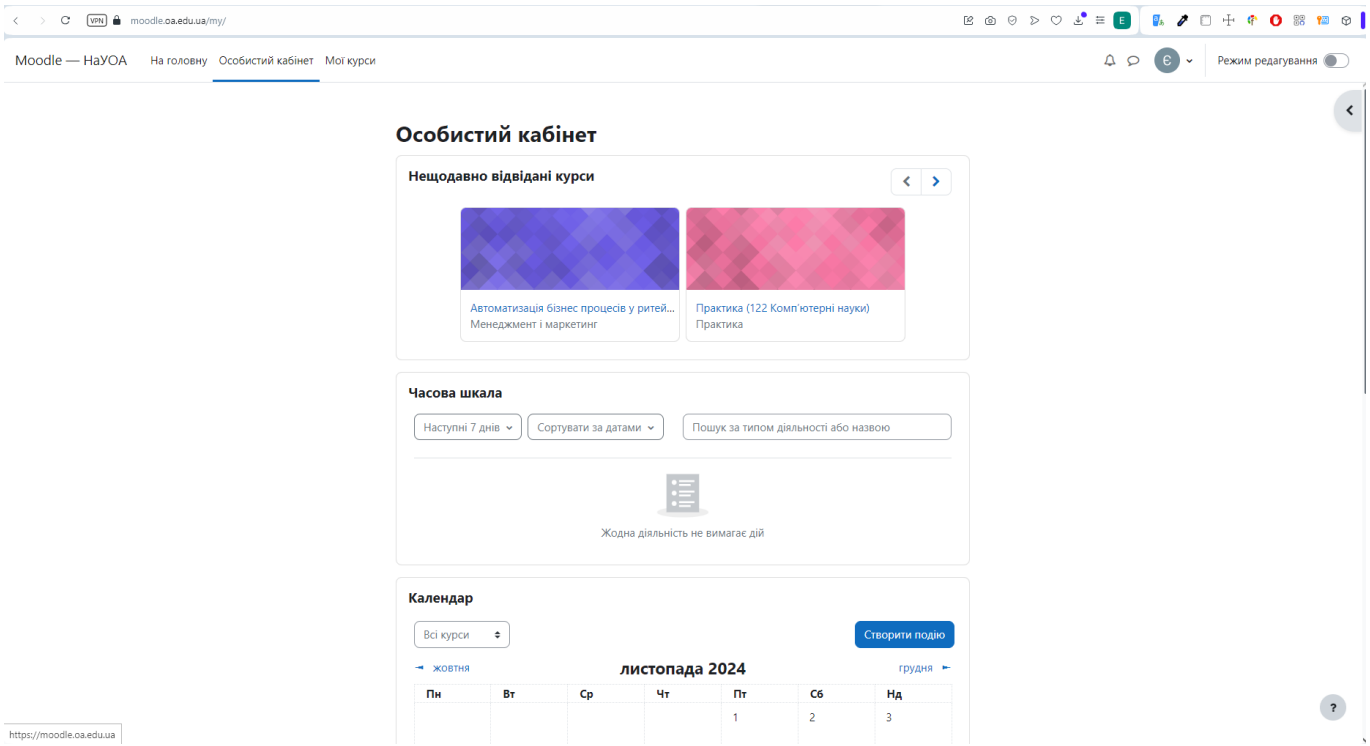

Рис. 1.2. Зовнішній вигляд додатку Moodle

Джерело: https://moodle.oa.edu.ua

3. Schoology - це платформа для навчання та управління контентом, яка дозволяє викладачам створювати курси, керувати завданнями, оцінками і взаємодіяти зі студентами через повідомлення та форуми (Рис. 1.3).

- **Функції:**
  - Управління курсами, завданнями, оцінками та календарем.
  - Інструменти для співпраці та спілкування: чати, форуми, повідомлення.
  - Створення та подання матеріалів для студентів (лекції, тести, документи).
- **Переваги:**
  - Можливість організації та управління навчальним контентом у форматі курсів.
  - Широка підтримка інтерактивних функцій, таких як форуми та спільні проекти.
- **Недоліки:**
  - Обмежені можливості для моніторингу фізичної присутності або використання QR-кодів для автоматизації процесу.

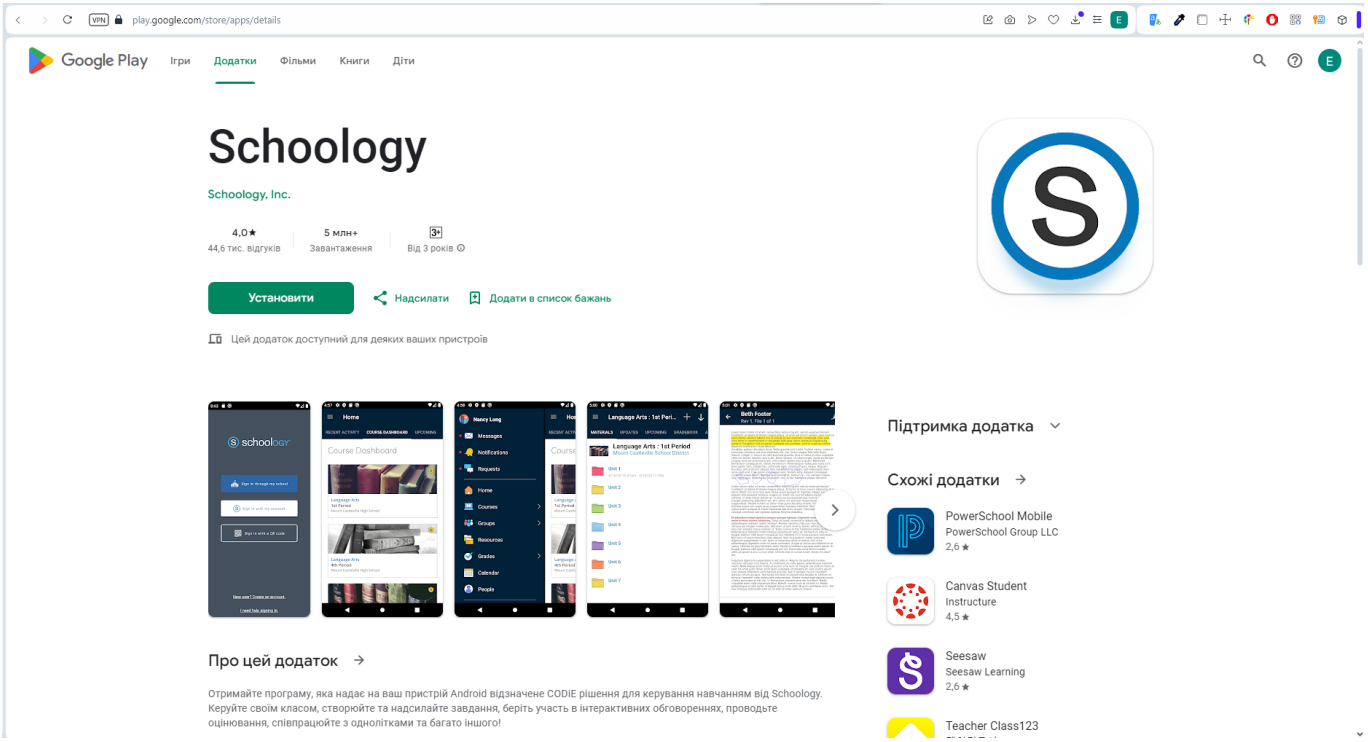

Рис. 1.3. Зовнішній вигляд додатку Schoology

Джерело: https://play.google.com/store/apps/details?id=com.schoology.app

4. Alora - Attendance Tracker - це мобільний додаток, який спеціалізується на моніторингу відвідуваності студентів. Він дозволяє викладачам відзначати присутність студентів за допомогою QR-кодів, а також надає різні функції для ефективного управління навчальним процесом (Рис. 1.4).

- **Функції:**
  - Моніторинг відвідуваності.
  - Звітність та аналіз.
- **Переваги:**
  - Автоматизація відвідуваності.
  - Зручний аналіз та звітність: Викладачі можуть швидко отримувати звіти про відвідуваність студентів.
- **Недоліки:**
  - Alora фокусується лише на моніторингу відвідуваності і не має функцій для обговорень, чатів або управління завданнями, що може бути важливим для комплексного освітнього процесу.

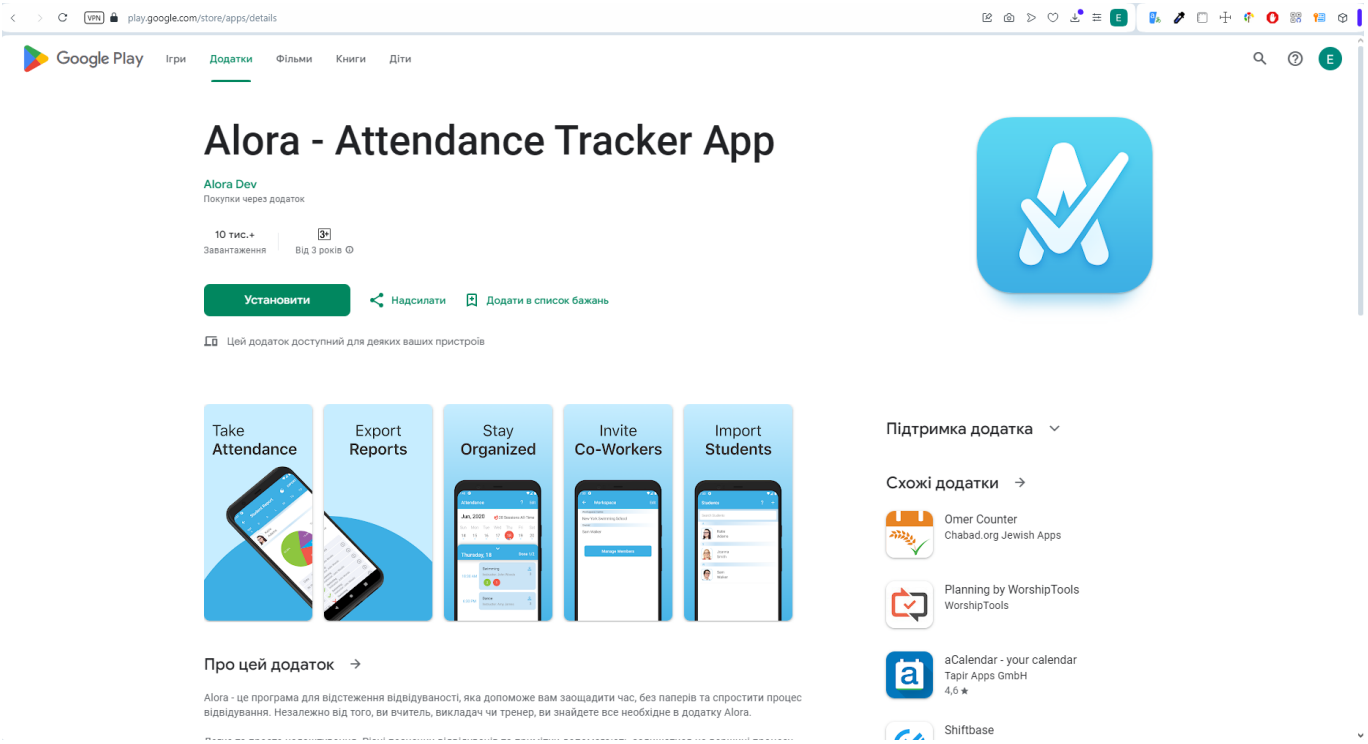

Рис. 1.4. Зовнішній вигляд додатку Alora – Atendance Tracker

Джерело: https://play.google.com/store/apps/details?id=com.smartlogicinc.attendancemanager

1.3. Постановка задачі

Даний мобільний додаток буде корисний як студентам, так і викладачам. Студенти зможуть зручно переглядати свій розклад, відмічати присутність на парах за допомогою QR-кодів, а також взаємодіяти з викладачами та іншими студентами через чат. Вони зможуть оперативно дізнаватися про новини академії та стежити за основними подіями Національного університету «Острозька академія». Викладачі, в свою чергу, зможуть генерувати QR-коди для відмітки присутності, переглядати список відвідуваності студентів, переглядати розклад своїх пар з детальною інформацією, а також взаємодіяти зі студентами та колегами через чат.

Основною метою додатку є створення зручної платформи для моніторингу відвідуваності студентів, організації розкладу та полегшення комунікації між студентами та викладачами. Додаток має на меті забезпечити ефективне управління

навчальним процесом, скорочення часу на організаційні питання, покращення доступу до важливої інформації та можливість оперативного реагування на зміни.

Мобільний додаток повинен мати наступний функціонал:

1. Сторінка авторизації

- **Вхід через домен:** Користувач може увійти в додаток тільки через електронну пошту з доменом @oa.edu.ua.
- **Валідація помилок:** Перевірка на правильність введених даних (електронної пошти та пароля), а також обробка помилок авторизації.
- **Рольовий вхід:** Після успішного входу, система визначає роль користувача (студент чи викладач) та надає відповідний доступ до функцій.

2. Головна сторінка

- **Перегляд новин та подій:** На головній сторінці користувачі (студенти та викладачі) можуть переглядати актуальні новини академії, а також важливі події, що відбуваються у навчальному закладі.
- **Інтерфейс:** Простий та зручний дизайн для легкого доступу до новин та подій.

3. Перегляд розкладу з фільтрацією по групах

- **Фільтрація по групах:** Розклад для студентів фільтрується відповідно до їхньої групи, що визначається за допомогою даних користувача. Тільки студенти своєї групи можуть переглядати свій розклад.
    - **Група студента:** Кожен студент має прив'язку до певної групи і на основі цієї інформації додаток фільтрує розклад.
    - **Інформація для студента:** Студент бачить розклад лише для тієї групи, до якої він належить. Розклад включає:
        - Тип заняття (лекція чи практична).
        - Формат заняття (онлайн або офлайн).
        - Якщо онлайн — посилання на Google Meet.
        - Якщо офлайн — номер кабінету.

        - Назва заняття та викладач, що веде його.
- **Розклад для викладачів:** Викладачі бачать розклад тільки для тих груп, з якими вони працюють.
    - **Інформація для викладача:** Викладач може переглядати розклад своїх занять, що включає:
        - Назву предмета.
        - Групи студентів, з якими проводить заняття.
        - Тип заняття (лекція або практична).
        - Формат заняття (онлайн чи офлайн).
        - Якщо онлайн — посилання на Google Meet.
        - Якщо офлайн — номер кабінету.
    - **Перегляд списку присутніх:** Викладач може переглядати список студентів, що відмітили свою присутність на парі через сканування QR-коду натиснувши на потрібний предмет.

4. Чат

- **Комунікація:** Студенти та викладачі можуть обмінюватися повідомленнями в чаті для ефективної комунікації щодо навчальних питань, запитів та обговорень.
- **Відокремлення чатів**: Використання компоненту Tab для організації чатів:
    - Чати з викладачами: Студенти та викладачі можуть мати окрему вкладку для чату з викладачами, що дає змогу вести обговорення з кожним викладачем.
    - Чати з студентами: Студенти та викладачі також можуть мати окрему вкладку для чату з студентами.
    - Інтерфейс: Вкладки дозволяють швидко перемикатись між чатами, забезпечуючи зручну навігацію.

5. Відмітка присутності

- **QR-код для викладача:** Викладач генерує QR-код на свої заняття, який студенти сканують для підтвердження присутності на парі.
- **Сканування студентами:** Студенти можуть сканувати QR-код за допомогою вбудованої камери телефона для підтвердження своєї присутності на занятті.

6. Сторінка профілю

- **Вихід з додатку:** Користувачі можуть вийти з додатку та повернутись на сторінку авторизації.
- **Дані дзвінків:** Перегляд інформації про тривалість дисциплін та перерв.
- **Корисні посилання:** Перегляд корисних ресурсів, як наприклад, посилання на навчальні матеріали або послуги академії.

7. Bottom Navigation для перемикання між сторінками

- **Навігація:** Знизу екрана розташована панель для зручного перемикання між основними розділами додатку:
    - **Головна сторінка** (новини та події).
    - **Розклад** (перегляд розкладу для студента або викладача).
    - **Відвідуваність** (генерація QR-коду для викладача та сканування для студентів)
    - **Чат** (переписка з іншими користувачами).
    - **Профіль** (налаштування профілю та вихід з додатку).

Мобільний додаток повинен бути інтуїтивно зрозумілим, простим у використанні, щоб забезпечити максимально комфортну взаємодію. Інтерфейс має бути структурованим, з чітким поділом на функціональні розділи, що дозволить користувачам швидко знайти необхідні функції, навіть якщо вони вперше використовують програму. Завдяки цьому додатку користувачі зможуть значно зекономити час, організувати процес навчання більш ефективно та підвищити продуктивність. Додаток спрямований на оптимізацію щоденних завдань, таких як відстеження розкладу, комунікація чи оцінка присутності, завдяки чому користувачі отримають сучасний і зручний інструмент для навчання та співпраці.

Висновок до розділу 1

У першому розділі було детально описано завдання та вимоги до кінцевого продукту, які мають бути враховані при розробці мобільного додатку. Зокрема, акцентовано на необхідності реалізації функцій, що відповідають потребам студентів і викладачів, таких як авторизація, перегляд розкладу, чат, система відмітки присутності та персональний профіль. Усі ці елементи повинні забезпечувати інтуїтивно зрозумілий інтерфейс та зручну навігацію. Також були розглянуті переваги і недоліки вибраних середовищ розробки, таких як React Native з Expo, що дозволяє створювати кросплатформенні додатки, та інструментів для емуляції і тестування, таких як Android Studio. Це дозволить вибрати оптимальне середовище для реалізації проекту з урахуванням необхідних функцій і стабільності роботи додатку.

У наступному розділі буде проаналізовано вибір інструментів для розробки додатку. Будуть розглянуті різні програмні мови та технології, які можуть бути використані для створення мобільного додатку, з огляду на вимоги до функціональності, особливості роботи на платформі Android, а також можливості обраного середовища розробки. Оскільки додаток містить важливі функції, такі як авторизація через конкретний домен, відмітка присутності через QR-коди та чат, особлива увага буде приділена безпеці даних користувачів, а також швидкості і стабільності роботи додатку. Вибір інструментів та технологій має бути зосереджений на забезпеченні не лише високої функціональності, але й зручності, швидкості роботи та безпеки даних. Оцінка програмних мов і інструментів дозволить створити додаток, який буде зручним, надійним та високопродуктивним.

Отже, ретельний аналіз доступних варіантів технологій дозволить вибрати оптимальні рішення для розробки мобільного додатку, який буде ефективним, безпечним і зручним для користувачів.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МЕТОДОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз предметної області (вхідні та вихідні дані)

Вхідні дані - це інформація, яку мобільний додаток OA Engage отримує від користувачів або з інших джерел, зокрема з бази даних Firebase Firestore. Firebase Firestore забезпечує централізоване зберігання та швидкий доступ до цих даних.

- **Дані про користувачів**
  - email (тип: string): Електронна адреса користувача, що використовується для авторизації в додатку. Додаток перевіряє цей email на відповідність домену @oa.edu.ua.
  - full_name (тип: string): Повне ім’я користувача.
  - photo_url (тип: string): URL до фото профілю користувача.
  - group_id (тип: reference): Посилання на документ з колекції groups, що вказує на групу до якої належатиме студент.
  - google_meet_link (тип: string): Коротке ім’я, яке використовуватиметься для формування посилання на Google Meet викладача.
  - role_id (тип: reference): Посилання на роль користувача в колекції roles (наприклад, "Студент" або "Викладач").
- **Дані про курси**
  - course_name (тип: string): Назва дисципліни, що викладається.
  - teacher_id (тип: reference): Посилання на документ викладача в колекції users, який веде цю дисципліну.
- **Дані про розклад**
  - course_id (тип: reference): Посилання на документ дисципліни з колекції courses.
  - group_id (тип: reference): Посилання на групу з колекції groups, до якої відноситься пара.
  - session_date (тип: timestamp): Дата та час проведення пари.
  - session_type (тип: string): Тип пари (лекція чи практична).

- format (тип: string): Формат заняття (онлайн чи офлайн).
- room_number (тип: string): Номер кабінету для офлайн-заняття.

- **Дані про чат**
  - roomId (тип: string): Ідентифікатор кімнати чату, що визначає, до якої групи користувачів належить чат.
  - text (тип: string): Текст повідомлення, що надсилається в чат.
  - senderName (тип: string): Ім’я відправника повідомлення.
  - profileUrl (тип: string): URL до фото профілю відправника.
- **Дані про присутність**
  - attendance_status (тип: string): Статус присутності (присутній/відсутній).
  - email (тип: string): Електронна адреса студента, щоб можна було відзначити його присутність на занятті.
  - full_name (тип: string): Повне ім’я студента.
  - photo_url (тип: string): Фото профілю студента.
- **Дані про події та новини**
  - title (тип: string): Назва події чи новини.
  - description (тип: string): Опис події чи новини.
  - dateTime (тип: timestamp): Дата та час проведення події або публікації новини.
  - url_adress (тип: string): Посилання на додаткові ресурси чи сайт події.

Вихідні дані - це інформація, яку мобільний додаток OA Engage надає користувачам після обробки вхідних даних або виконання певних дій у додатку. Вони зазвичай надходять з Firestore або генеруються на основі запитів до цієї бази даних.

- **Результати авторизації**
  - **Успішна авторизація:** Якщо email користувача перевірений, пароль правильний, і користувач має відповідну роль (студент або викладач), відображається персоналізований інтерфейс.

- **Помилка авторизації:** Якщо дані введені неправильно (наприклад, email або пароль), користувач отримає повідомлення про помилку.

- **Інтерфейс розкладу**
  - Для **студентів:** Додаток генерує розклад на основі даних про групу студента, що зберігаються в колекції *users*. Це дає змогу студену побачити лише свої заняття.
  - Для **викладачів:** Додаток виводить лише ті курси, які викладач веде, отримуючи їх через зв’язок між викладачами та курсами в колекції *courses*.
- **Чат**
  - Користувач може надсилати повідомлення у чат, який зберігається в колекції *rooms* (чат). Повідомлення зберігаються у підколекції *messages*, де кожне повідомлення має дату створення (*createdAt*), ім’я відправника, текст повідомлення та URL до фото профілю відправника.
  - Повідомлення можна відправляти тільки у чаті, де користувач є учасником, що визначається через *roomId*.
- **Інтерфейс присутності**
  - Для **викладачів**: Додаток дозволяє викладачеві генерувати QR-код, який студенти сканують для реєстрації присутності. Після цього, статус присутності студента зберігається в підколекції *attendance* в документі, що відноситься до конкретного заняття в колекції *schedule*.
  - Для **студентів**: Студенти сканують QR-код, і їхня присутність реєструється в Firestore.
- **Інтерфейс новин та подій**
  - Всі новини та події зберігаються в колекції *maiv_Events* (для подій) та *news_Academic* (для новин). Користувачі можуть переглядати деталі подій та новин, а також отримувати посилання на ресурси чи додаткову інформацію.
- **Профіль користувача**

- Вихід з додатку передбачає очищення сесії і перенаправлення на сторінку входу.

Firebase Firestore забезпечує централізоване зберігання даних, високошвидкісну обробку запитів, реальний час оновлення та синхронізацію між користувачами додатку. Це дозволяє створювати інтерактивний, динамічний досвід для користувачів.

2.2. Створення плану розробки системи (план розробки/впровадження обов'язкових складових системи для етапу мінімального життєздатного продукту)

Для етапу мінімального життєздатного продукту (MVP) мобільного додатку **OA Engage**, план розробки має включати основні функції, які забезпечать базову, але повноцінну роботу додатку для викладачів та студентів. Цей етап повинен покривати ключові функціональні вимоги, а також забезпечити основні механізми взаємодії між користувачами і системою.

1. **Етапи розробки MVP**

1.1.Проектування інтерфейсу користувача (UI/UX)

- **Тривалість:** 1 тиждень.
- **Завдання:**
  - Створити прості макети основних екранів у **Figma**.
  - Запланувати структуру навігації (екрани для авторизації, розкладу, чату, профілю).

1.2.Проектування структури бази даних Firebase Firestore

- **Тривалість:** 2-3 дні.
- **Завдання:**
  - **Структурування даних:**
    - Ролі користувачів (*roles*).

        - Дані про студентів, викладачів, групи, курси, розклад (*users*, *groups*, *courses*, *schedule*).
        - Дані для новин, подій, чату, відмітки присутності (*news_Academic*, *maiv_Events*, *rooms*, *attendance*).
    - **Зв'язки між колекціями:**
        - Встановити референції для логічного зв'язку (наприклад, *role_id*, *group_id*, *teacher_id*).
        - Забезпечити зручність фільтрації та пошуку даних.
    - **Початкове наповнення:**
        - Створити тестові дані для всіх колекцій (один студент, викладач, група, курс, розклад).
    - **Налаштування правил доступу:**
        - Визначити рівні доступу:
            - Студент може читати лише дані, що стосуються його групи.
            - Викладач має доступ до даних своїх курсів і груп.

1.3.Налаштування проекту

- **Тривалість:** 1-2 дні.
- **Завдання:**
    - Ініціалізувати проект у **React Native Expo**.
    - Підключити Firebase до проекту (Firestore).
    - Інсталювати необхідні бібліотеки: для QR-кодів, роботи з Firestore, календаря, чату.

1.4.Авторизація через Firestore

- **Тривалість:** 2-3 дні.
- **Завдання:**
    - Використовувати колекцію *users* для входу:
        - Перевірка email і пароля.
        - Ідентифікація ролі користувача через *role_id*.
    - Реалізувати редирект на відповідний інтерфейс залежно від ролі.

1.5.Головна сторінка (новини та події)

- **Тривалість:** 1 тиждень.
- **Завдання:**
    - Отримувати дані новин з Firebase (*news_Academic*).
    - Відображати події академії з колекції *maiv_Events*.
    - Зробити базовий дизайн (список новин і подій).

1.6.Екран розкладу

- **Тривалість:** 2 тижні.
- **Завдання:**
    - Реалізувати календар із відображенням розкладу занять.
    - Підключити Firebase Firestore для отримання розкладу з колекції *schedule*.
    - Зробити фільтрацію розкладу за роллю користувача:
        - Студент бачить тільки свої заняття.
        - Викладач бачить тільки ті дисциплыни, які він викладає.
    - Відображати деталі пари (час, аудиторія, тип, формат).

1.7.Функціонал чату

- **Тривалість:** 2-3 тижні.
- **Завдання:**
    - Створити простий екран чату з інтеграцією Firestore (*rooms* і *messages*).
    - Додати можливість відправляти/отримувати повідомлення.
    - Реалізувати оновлення чату в реальному часі.

1.8.Відмітка присутності (QR-код)

- **Тривалість:** 2-3 тижні.
- **Завдання:**
    - Викладач: реалізувати генерацію QR-коду, прив’язаного до конкретного заняття.
    - Студент: створити функцію сканування QR-коду та реєстрацію присутності.
    - Зберігати статус присутності в підколекції *attendance* в Firestore.

1.9.Профіль користувача

- **Тривалість:** 1 тиждень.
- **Завдання:**
    - Реалізувати простий екран профілю з можливістю перегляду даних користувача.
    - Додати функцію виходу з акаунту.
    - Додати корисні посилання або інформацію про дзвінки.

1.10. Тестування

- **Тривалість:** 1-2 тижні.
- **Завдання:**
    - Перевірити кожну функцію на працездатність.
    - Використати емулятор Android Studio для тестування додатку.
    - Знайти та виправити баги.

2. **Оцінка тривалості розробки**
    - Загальна тривалість: 10-12 тижнів (3 місяці) з урахуванням того, що я самотужки розробляв проєкт.

2.3. Методології розробки

Для створення мобільного додатку **OA Engage** була застосована методологія гнучкої (Agile) розробки, що дозволила швидко реагувати на змінні вимоги та потреби користувачів. Зокрема, використано підхід Scrum, що допомагає організувати робочі процеси в команді та забезпечити регулярні поставки функціональних етапів продукту. Розробка проводилась за такими етапами:

1. **Аналіз вимог:**
    - Визначено важливі фічі, такі як авторизація через домен @oa.edu.ua, перегляд розкладу, чат та відмітка присутності.
2. **Планування спринтів:**

- Розробка була організована на основі спринтів, кожен з яких тривав 1-2 тижні. Кожен спринт містив чітке планування задач, тестування та реалізацію нових фіч.

3. **Ітеративна розробка:**
   - В процесі кожного спринту були розроблені окремі частини функціоналу додатку, зокрема:
     - Інтерфейс авторизації, що забезпечує вхід тільки через академічний домен.
     - Основні функції: перегляд новин, розкладу, чату, відмітки присутності та профілю.
     - Модулі для завантаження даних з Firebase: роль користувача, група, розклад, чати та відмітки присутності.
4. **Інтеграція та тестування:**
   - Регулярно тестувались окремі компоненти додатку, такі як компоненти авторизації, відмітки присутності, взаємодія з Firebase для отримання та запису даних.
   - Для тестування використовувалося ручне тестування для перевірки на помилки користувацького інтерфейсу.
5. **Робота з Firebase:**
   - Під час розробки додатку було створено структуру бази даних у Firebase, що включає колекції, як-от roles, groups, users, courses, schedule, attendance, rooms, messages, maiv_Events, news_Academic. Кожен етап роботи з базою даних було сплановано таким чином, щоб забезпечити надійне зберігання даних і швидкий доступ до них.

Таким чином, завдяки використанню методології Agile з інкрементальним підходом до розробки, регулярним спринтам та постійним зворотним зв'язком від зацікавлених сторін, вдалося забезпечити швидку, адаптивну та ефективну розробку. Це дозволило впровадити функціонал, який не лише відповідає технічним вимогам, а й повністю задовольняє потреби та очікування кінцевих користувачів. Кожен етап

розробки супроводжувався ретельним тестуванням, що сприяло підвищенню якості продукту та його надійності. Такий підхід гарантував, що додаток залишатиметься актуальним, інтуїтивно зрозумілим і корисним для користувачів, створюючи максимально комфортний досвід взаємодії.

Висновки до розділу 2

У другому розділі було розглянуто основні аспекти інформаційного та методологічного забезпечення мобільного додатку OA Engage. Описано структуру вхідних і вихідних даних, що обробляються додатком, зокрема дані про користувачів, курси, розклад, чат, відмітку присутності та новини. Вхідні дані надходять з Firebase Firestore і включають електронні адреси, імена, роль користувачів, дані про курси та події. Вихідні дані - це результати авторизації, персоналізований інтерфейс, відображення розкладу, повідомлення чату, інформація про присутність та новини. Також було розглянуто план розробки для створення мінімального життєздатного продукту (MVP) додатку. Цей план передбачає етапи проектування, розробки, налаштування Firebase, авторизації, інтеграції функцій (новини, розклад, чат, відмітка присутності), а також тестування. Оцінено загальну тривалість розробки на 10-12 тижнів. Застосована методологія гнучкої розробки Agile з використанням підходу Scrum дозволила ефективно організувати процес і забезпечити регулярну поставку функціональних етапів продукту.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Для створення мобільного додатку OA Engage були використані сучасні технології та інструменти, які забезпечили ефективну та швидку розробку, тестування та інтеграцію з базою даних. Ось основні засоби, які застосовувались у процесі розробки:

React Native і Expo стали основою для розробки, забезпечуючи створення кросплатформених додатків для iOS та Android за допомогою єдиної кодової бази. Це значно скоротило час розробки та спростило інтеграцію з функціями завдяки готовим інструментам, які надає Expo.

- React Native - це потужний фреймворк для створення кросплатформених мобільних додатків за допомогою JavaScript та React. Він дозволяє розробляти додатки для iOS та Android, використовуючи єдину кодову базу, що знижує витрати часу та ресурсів на розробку.
- Expo є фреймворком на основі React Native, який забезпечує набір готових інструментів для швидкої розробки, таких як доступ до камери, GPS, сповіщень і багатьох інших функцій. Використання Expo значно полегшує налаштування середовища розробки та інтеграцію з різними API.

Firestore (Firebase) використовувався як хмарна база даних для зберігання інформації в реальному часі. Він забезпечує зручне збереження та обробку даних користувачів, розкладу, курсів і чату.

- Firestore використовується для зберігання даних додатку в реальному часі. Це хмарна база даних, яка дозволяє безпосередньо зберігати та обробляти дані користувачів, розкладу, курсів, чату та інших елементів додатку.
- У Firestore зберігаються документи, що містять інформацію про користувачів (включаючи їх email, повне ім’я, роль і інші дані), курси, групи, а також розклад занять.

- Для авторизації в додатку використовується Firestore, де в колекції *users* зберігається інформація про користувачів, включаючи їх email та пароль (зашифрований). Вхід здійснюється за допомогою перевірки введеного email на відповідність домену **@oa.edu.ua** та перевірки даних для авторизації.

Visual Studio Code став основним інструментом для написання коду. Його функції автодоповнення, підсвітки синтаксису, відлагодження та інтеграції з Git значно спрощують розробку.

- Visual Studio Code (VS Code) - це основний текстовий редактор, який використовується для написання коду. VS Code має вбудовані функції автодоповнення, підсвітки синтаксису, відлагодження та інтеграцію з Git, що значно прискорює процес розробки.
- Для React Native в VS Code доступні плагіни, що полегшують роботу з JavaScript та React, зокрема плагіни для React, ESLint, Prettier, що допомагають підтримувати високий стандарт коду та уникати помилок.

Android Studio забезпечував емуляцію пристроїв та тестування додатку, що дозволяло виявляти проблеми з продуктивністю та адаптувати додаток під різні розміри екранів.

- Android Studio використовується для емуляції мобільних пристроїв та тестування додатка на Android. Цей інструмент дозволяє налаштовувати емулятори різних пристроїв, що дає можливість перевірити, як додаток виглядатиме на різних екранах і з різними характеристиками.
- З Android Studio також можна проводити тестування та профілювання додатку для виявлення можливих проблем з продуктивністю.

Figma забезпечила проектування інтерфейсу користувача, дозволяючи створювати інтерактивні прототипи та макети екранів додатку.

- Figma використовується для проектування інтерфейсу користувача додатку. Це потужний інструмент для дизайну, який дозволяє створювати інтерактивні

прототипи та макети екранів додатку, що полегшує процес розробки інтерфейсу.

- Figma дає можливість команді дизайнерів і розробників працювати над дизайном у реальному часі, що значно пришвидшує процес і покращує співпрацю.

Adobe Illustrator і Adobe After Effects допомогли створити графіку та анімації, які зробили інтерфейс додатку більш привабливим та зручним для користувачів.

- Adobe Illustrator використовується для створення векторної графіки, такої як іконки, логотипи, а також елементи інтерфейсу, що потребують високої якості та масштабованості на різних роздільних здатностях екранів мобільних пристроїв.
- Adobe After Effects застосовується для створення анімацій, що додають візуальні ефекти до інтерфейсу додатку. Це дозволяє створювати ефекти переходів, анімації кнопок та інші графічні елементи, що покращують взаємодію користувача з додатком.

Git забезпечив контроль версій, дозволяючи організовано зберігати історію змін та ефективно співпрацювати в команді.

- Git є основною системою контролю версій, яка дозволяє зберігати історію змін у коді, відслідковувати версії та співпрацювати з іншими розробниками. Завдяки Git можна ефективно управляти гілками коду та зливати зміни, що робить розробку більш організованою та безпечною.

Для тестування додатків на реальних пристроях використовувався **Expo Go**, що дозволило швидко перевіряти роботу додатку без компіляції APK або IPA файлів.

- Expo Go - мобільний додаток для тестування додатків, створених за допомогою Expo. Це дозволяє безпосередньо запускати додатки на реальних пристроях без потреби компілювати APK або IPA файли, що значно пришвидшує тестування та виправлення помилок.

3.2. Вимоги до технічного та програмного забезпечення

У даному підрозділі визначаються основні вимоги до технічного та програмного забезпечення, необхідного для функціонування мобільного додатку OA Engage. Ці вимоги охоплюють як апаратне забезпечення для розробки та тестування, так і програмне забезпечення, яке використовується для створення, компіляції та виконання додатку.

1. **Технічні вимоги**

- **Апаратне забезпечення для розробки**
    - Процесор: мінімум Intel i5 або еквівалентний ARM процесор, з тактовою частотою від 2.5 GHz та вищою, для комфортної роботи з інструментами розробки.
    - Оперативна пам’ять (RAM): мінімум 8 ГБ для ефективної роботи з програмним забезпеченням для розробки та емуляцією мобільних пристроїв.
    - Місце на диску: мінімум 50 ГБ вільного місця на SSD для встановлення необхідного програмного забезпечення (Visual Studio Code, Android Studio, Expo, бібліотеки, емулятори тощо).
    - Екран: мінімум Full HD (1920x1080) для зручного перегляду коду та макетів дизайну.
- **Апаратне забезпечення для тестування**
    - Мобільні пристрої для тестування: для тестування на реальних пристроях використовуються Android-смартфони з версією Android 5.0 або вище. Такі пристрої забезпечують достовірну перевірку функціональності, продуктивності та сумісності додатку. У майбутньому планується додаткове тестування на пристроях з iOS, зокрема iPhone, для розширення охоплення користувачів.
    - Емулятори: Android Studio надає зручні емулятори, що дозволяють перевіряти додаток на різних версіях Android і конфігураціях пристроїв. Це значно спрощує процес виявлення помилок та оптимізації роботи

додатку без необхідності використання фізичних девайсів, забезпечуючи ефективне тестування навіть на ранніх етапах розробки.

2. **Програмні вимоги**

- **Операційна система**
    - Windows 10/11 або macOS: Операційні системи, що забезпечують підтримку розробки мобільних додатків з використанням React Native та Expo. Для тестування на реальних пристроях необхідні відповідні драйвери для Android (Windows) або Xcode для iOS (macOS).
- **Програмне забезпечення для розробки**
    - **Visual Studio Code (VS Code):**
        - Легкий та потужний текстовий редактор, який використовує безліч плагінів для підтримки мов програмування, таких як JavaScript, JSX, TypeScript, React Native.
        - Плагіни: React Native Tools, ESLint, Prettier, GitLens.
    - **Node.js (версія 16 або новіша):**
        - Необхідний для запуску npm (Node Package Manager) та управління бібліотеками, використаними у проекті. Node.js є основою для роботи з Expo та React Native.
    - **Expo CLI:**
        - Ідентифікація мобільного додатку в середовищі Expo для швидкої розробки та тестування. Expo також забезпечує інтеграцію з численними бібліотеками, що дозволяє зменшити час на налаштування.
    - **Android Studio:**
        - Для емуляції мобільних пристроїв та тестування додатку на Android. Також використовуєть для налагодження та аналізу продуктивності додатку.
        - Вимагає встановлення SDK для Android та налаштування емуляторів для тестування додатку на різних пристроях.

- **Xcode (для розробки під iOS на macOS):**
    - Для тестування та компіляції додатків на платформі iOS. Включає емулятори iOS-пристроїв та інструменти для розробки додатків.
- **Figma:**
    - Для проектування інтерфейсу користувача (UI/UX) та створення прототипів. Figma дозволяє розробникам швидко тестувати макети інтерфейсу та отримувати зворотний зв'язок від користувачів.
- **Git:**
    - Для контролю версій та співпраці в команді. Git дозволяє зберігати історію змін, керувати різними версіями проекту та інтегруватися з платформами, такими як GitHub або GitLab.
    - Для зберігання коду в Git репозиторіях та синхронізації змін з іншими розробниками.

- **Програмне забезпечення для бази даних**
    - **Firebase:**
        - Використовується для зберігання даних додатку (розклад, курси, чати, присутність). Включає Firestore для роботи з документами та колекціями, а також Firebase Authentication для обробки авторизації, хоча в даному випадку використовується Firestore для авторизації користувачів.
        - Використовуються також Firebase SDK для взаємодії додатку з Firestore, що дозволяє реалізувати швидкий доступ до даних у реальному часі.
- **Інші інструменти**
    - **Adobe Illustrator та Adobe After Effects:**
        - Використовуються для створення графічних елементів, які додають професійний вигляд інтерфейсу користувача мобільного додатку. Adobe After Effects дозволяє розробляти анімації та візуальні ефекти, надаючи йому сучасний та динамічний вигляд.

3. Мережеві вимоги

- Для роботи з Firestore та Firebase необхідно стабільне інтернет-з'єднання. Всі операції з базою даних, а також синхронізація даних у реальному часі здійснюються через хмарні сервіси Firebase.
- Рекомендується використовувати швидке підключення до Інтернету для зручної роботи з емуляторами та швидкої перевірки API-запитів.

3.3. Опис програмної реалізації

3.3.1. Прототипування додатку в Figma

Прототипування є важливою частиною процесу розробки мобільного додатку OA Engage, оскільки дозволяє створити візуальну концепцію інтерфейсу користувача та взаємодії з додатком ще до його розробки. Для цього було використано програмне забезпечення Figma, яке забезпечує потужні інструменти для створення інтерактивних прототипів, дизайну інтерфейсу та колаборації в команді. На рисунку 3.1 зображено процес створення прототипу мобільного додатку, який згодом слугував основою для реалізації інтерфейсу.

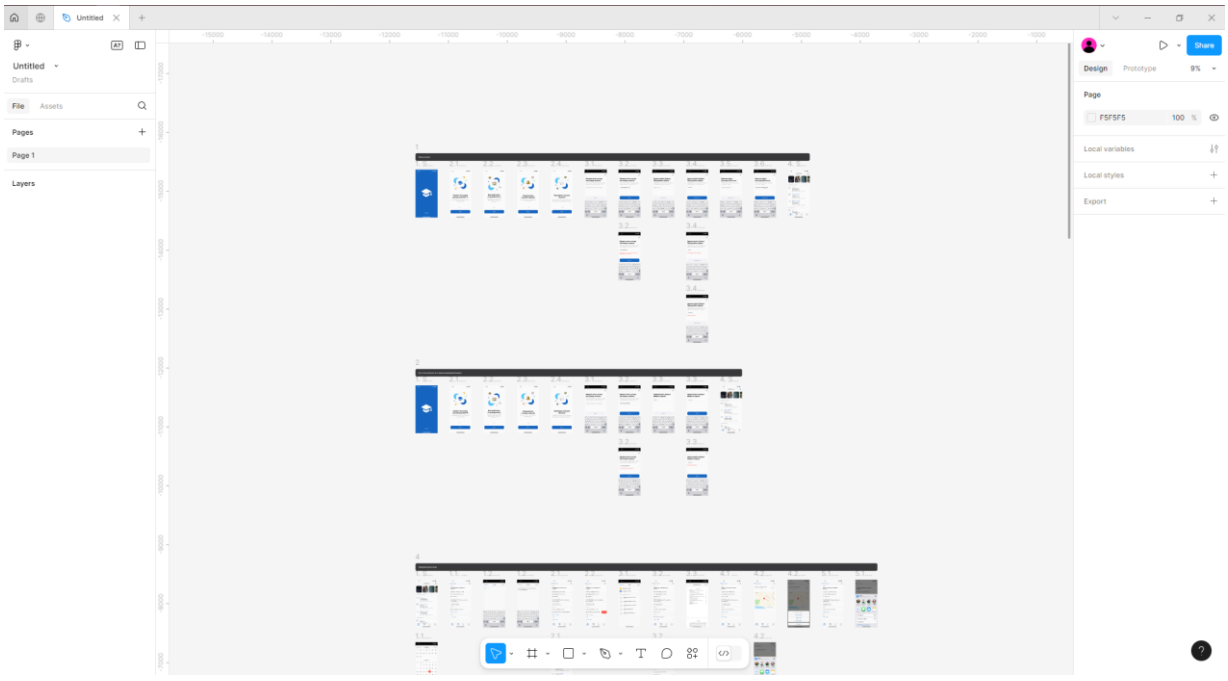

Рис. 3.1. Створення прототипу для майбутнього додатку

Джерело: Розроблено автором

3.3.2. Створення векторних ілюстрацій для майбутнього додатку

На цьому етапі я створював векторні ілюстрації в **Adobe Illustrator** для дизайну додатку. Я малював графічні елементи, такі як векторні зображення для початкового екрана, яке є основним елементом переходу до екрану входу. Також я створив ілюстрацію для екрана авторизації та інших, щоб додати візуальну привабливість та зручність для користувачів. Ці векторні зображення були оптимізовані для використання в мобільному додатку та забезпечили гарний вигляд на різних екранах На рисунку 3.2 наведено приклад створення векторних ілюстрацій, які використовувалися в дизайні додатку.

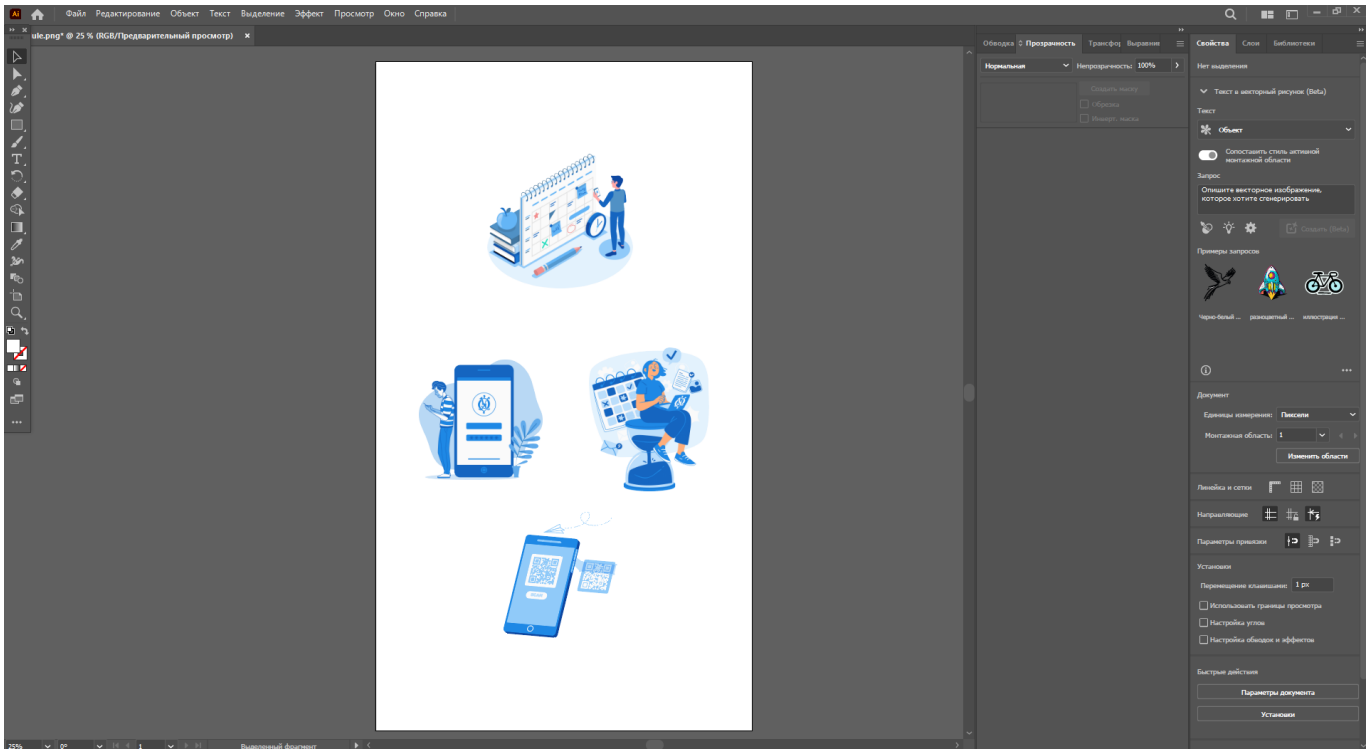

Рис. 3.2. Створення векторних ілюстрацій для мобільного додатку

Джерело: Розроблено автором

3.3.3. Моделювання бази даних у dbdiagram.io та створення БД у Firebase Firestore

На першому етапі розробки бази даних було проведено моделювання її структури за допомогою інструменту **dbdiagram.io**. Це дозволило візуалізувати всі

таблиці, зв'язки між ними та структуру даних для подальшого створення реальної бази даних у **Firebase Firestore**. Такий підхід забезпечив системний підхід до планування бази даних, уникнення помилок у зв'язках і спрощення подальшого кодування.

Моделювання включало визначення основних колекцій та полів, які мають бути представлені у базі даних. Основні колекції, що були розроблені:

- **roles:** колекція ролей користувачів.
- **groups:** колекція груп студентів.
- **users:** колекція користувачів, де зберігаються дані про кожного користувача.
- **courses:** колекція дисциплін, що містить інформацію про назву дисципліни та викладача.
- **schedule:** колекція розкладів, де зберігаються дані про кожну пару, тип і формат.
- **attendance:** підколекція розкладу, яка зберігає інформацію про присутність студентів на заняттях.
- **rooms:** колекція для чатів, яка містить кімнати для переписки між студентами та викладачами.
- **messages:** підколекція чату для зберігання повідомлень.

За допомогою **dbdiagram.io** була створена **ER-діаграма** (Entity-Relationship Diagram), що чітко відображає структуру бази даних та зв'язки між сутностями (рис. 3.3). Це стало основою для впровадження бази даних у Firebase Firestore, а також забезпечило розробку логічної та масштабованої архітектури для мобільного додатка.

Моделювання структури дозволило заздалегідь врахувати можливі сценарії використання системи та забезпечити ефективну обробку даних у додатку.

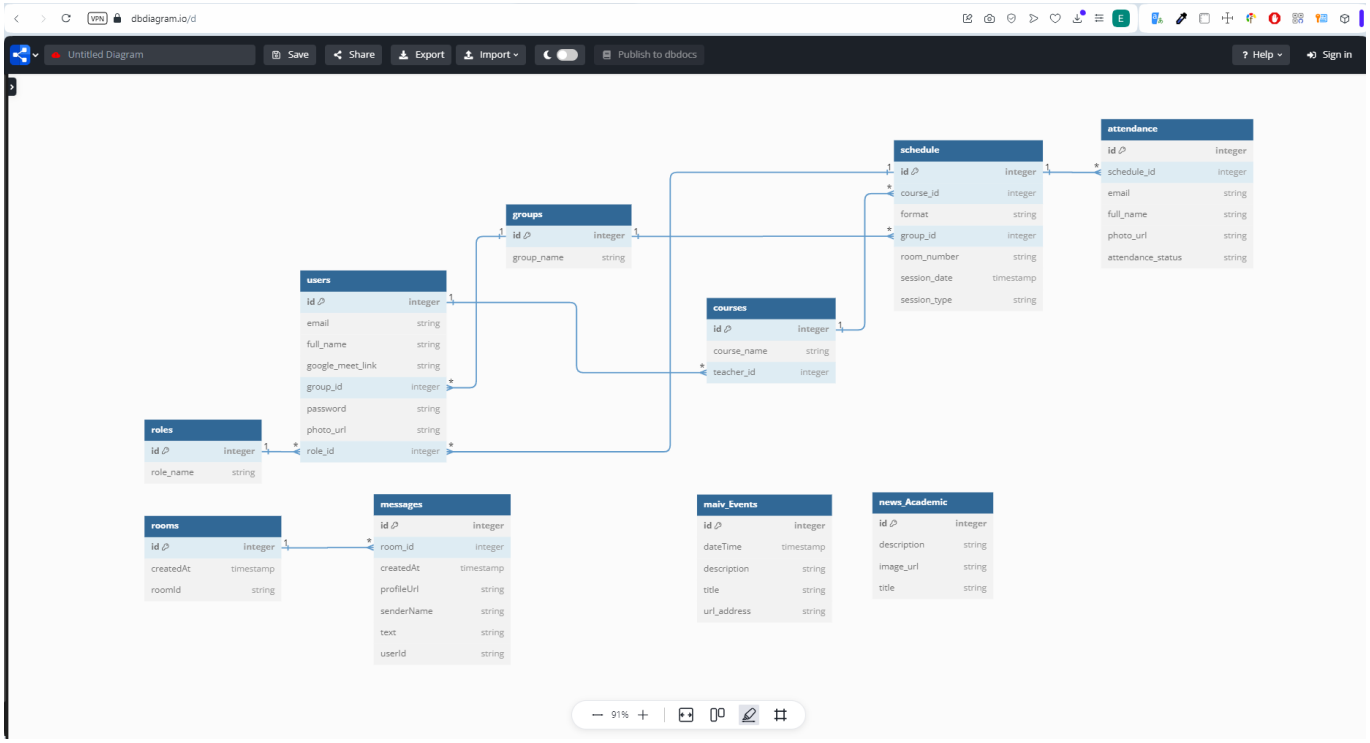

Рис. 3.3. Візуалізація структури БД для мобільного додатку

Джерело: Розроблено автором

Після того як структура бази даних була змодельована у **dbdiagram.io**, наступним кроком було створення бази даних у **Firebase Firestore**. Це дозволило реалізувати схему бази даних, використовуючи реальні колекції та документи.

Основні кроки створення бази даних у Firebase Firestore:

- **Створення проекту в Firebase:** Створено новий проект у Firebase, який був інтегрований з мобільним додатком на платформі React Native.
- **Створення колекцій:** У Firestore були створені колекції, що відповідають моделям, визначеним на етапі моделювання. Кожна колекція містить відповідні документи з полями, які відповідають вимогам додатку.
- **Налаштування правил безпеки:** Були налаштовані правила доступу до даних у Firestore для забезпечення безпеки та обмеження доступу до чутливої інформації. Наприклад, студенти можуть бачити лише свій розклад і відмічатися на своїх парах, а викладачі мають доступ до курсів та чату.

Процес моделювання бази даних у **dbdiagram.io** дозволив чітко спланувати структуру даних, після чого була реалізована база даних у **Firebase Firestore** (рис 3.4).

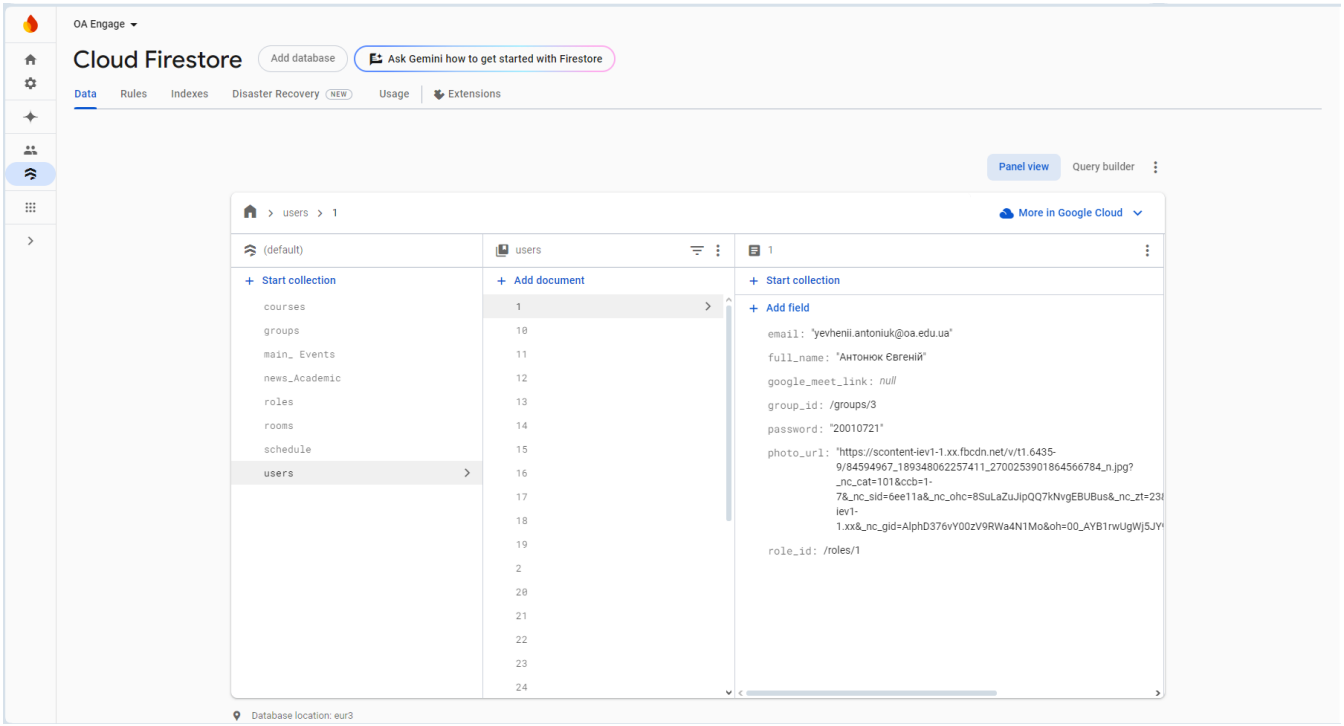

Рис. 3.4. Готова структура БД у Firebase Firestore

Джерело: Розроблено автором

3.3.4. Встановлення необхідних бібліотек для розробки додатку

Для розробки мобільного додатку OA Engage використовувалась технологія React Native з Expo, що дозволило забезпечити швидку розробку та зручне тестування на різних платформах. Під час створення додатку необхідно було встановити ряд бібліотек, які забезпечують функціональність, зручність розробки та інтеграцію з Firebase для збереження даних.

Нижче наведено основні бібліотеки, які були використані для роботи з додатком:

- **expo:** Основна бібліотека для роботи з React Native в середовищі Expo. Встановлення Expo дозволяє швидко запускати проєкти та тестувати їх на реальних пристроях.
- **firebase:** Основна бібліотека для інтеграції з Firebase. Використовується для доступу до Firestore, аутентифікації користувачів, відправки push-повідомлень і зберігання даних. Вона забезпечує зручну інтеграцію з Firebase у додатку.

- **react-native-qrcode-svg:** Бібліотека для створення QR-кодів. Вона використовується для генерації QR-кодів для викладачів, які можуть бути скановані студентами для відмітки присутності на заняттях.
- **react-native-async-storage/async-storage:** Бібліотека для зберігання даних локально на пристрої, таких як налаштування користувача, токени аутентифікації або інші дані, що зберігаються між сесіями. Вона дозволяє читати та записувати дані в місцеву пам'ять.
- **expo-camera:** Бібліотека для доступу до камери пристрою. Вона використовується для реалізації функції сканування QR-кодів, що дозволяє студентам відмічати свою присутність на заняттях.
- **expo-linear-gradient:** Бібліотека для створення градієнтів в інтерфейсі. Вона дозволяє застосовувати красиві градієнтні фони для екранів додатку або різних елементів інтерфейсу.
- **expo-router:** Бібліотека для спрощення навігації в додатку з використанням маршрутизації. Вона дозволяє зручно організувати переходи між екранами, базуючись на структурі файлів.
- **lottie-react-native:** Бібліотека для анімацій у додатку на основі файлів у форматі JSON (Lottie). Вона використовується для додавання анімацій, таких як завантаження або ефекти взаємодії з користувачем.
- **react-native-material-ripple:** Бібліотека для додавання ефекту рипл (хвиля) до елементів інтерфейсу, таких як кнопки та інші інтерактивні елементи. Це дозволяє створити більш динамічний та сучасний вигляд додатку.
- **react-native-reanimated:** Бібліотека для анімацій та обробки жестів у додатку. Вона дозволяє створювати плавні анімації та інтерактивні елементи, що робить додаток більш приємним для користувачів.
- **react-native-svg:** Бібліотека для роботи з SVG (Scalable Vector Graphics) в React Native. Вона дозволяє додавати масштабовані векторні графічні елементи в інтерфейс додатку.

3.3.5. Інтеграція Firebase та створення базових функцій

Для інтеграції Firebase у мобільний додаток на React Native потрібно налаштувати Firebase-конфігурацію, підключити Firestore для роботи з базою даних, а також створити утиліти та базові функції для спрощення доступу до даних.

Крок 1. Ініціалізація Firebase та підключення колекцій:

Я використав метод *initializeApp* з Firebase SDK для налаштування підключення до Firebase, передавши конфігурацію проекту. Також було налаштовано доступ до Firestore через функцію *getFirestore*. Після налаштування Firebase я створив референси до основних колекцій в Firestore, таких як *roles, users, schedule, news_Academic*, і *main_Events*, що використовуватимуться для зберігання та отримання даних.

Лістинг коду 3.1. firebaseConfig.jsx

```
import { initializeApp } from "firebase/app";
import { collection, getFirestore } from "firebase/firestore";
// Конфігурація Firebase
const firebaseConfig = {
  apiKey: "AIzaSyCSQUCBDDG9t88wXEcZ9ukhdTpi9nahNeM",
  authDomain: "oa-engage-a0d19.firebaseapp.com",
  projectId: "oa-engage-a0d19",
  storageBucket: "oa-engage-a0d19.firebasestorage.app",
  messagingSenderId: "1026244140316",
  appId: "1:1026244140316:web:faac18136c8386af35eed9"
};
// Ініціалізація Firebase
const app = initializeApp(firebaseConfig);
export const db = getFirestore(app);
// Визначаємо колекції для Firestore
export const rolesRef = collection(db, 'roles');
export const usersRef = collection(db, 'users');
export const newsRew = collection(db, 'news_Academic');
export const eventRew = collection(db, 'main_ Events');
export const scheduleRef = collection(db, 'schedule')
```

Примітка. Код реалізує ініціалізацію Firebase з додатком та роботу з колекціями

Крок 2 . Функції для роботи з сесією:

Для збереження даних користувача між сесіями використовую *AsyncStorage*, що дозволяє зберігати і отримувати інформацію про користувача.

Лістинг коду 3.2. firebaseConfig.jsx

```
import AsyncStorage from "@react-native-async-storage/async-storage";
// Збереження сесії користувача
export const setSession = async (user) => { try {
    await AsyncStorage.setItem('user', JSON.stringify(user));
  } catch (error) {
    console.error("Помилка при збереженні сесії:", error);
  }
};
// Отримання сесії користувача
export const getSession = async () => {
  try {
    const storedUser = await AsyncStorage.getItem('user');
    return storedUser ? JSON.parse(storedUser) : null;
  } catch (error) {
    console.error("Помилка при зчитуванні сесії:", error);
    return null;
  }
};
// Очищення сесії
export const clearSession = async () => {
  try { await AsyncStorage.removeItem('user');
  } catch (error) {
    console.error("Помилка при очищенні сесії:", error);
  }
```

Примітка. Код реалізує збереження та завантаження сесії

Крок 3. Отримання деталей користувача:

За допомогою функції *getUserDetails* я отримую дані про користувача, включаючи його роль та групу, шляхом виклику відповідних документів з колекцій *roles* і *groups*.

Лістинг коду 3.3. userService.jsx

```
import { getDoc } from 'firebase/firestore';

export const getUserDetails = async (userData) => {
  try {
    const roleRef = userData.role_id;
    const roleSnap = await getDoc(roleRef);
    const roleData = roleSnap.exists() ? roleSnap.data() : null;
    const groupRef = userData.group_id;
    const groupSnap = groupRef ? await getDoc(groupRef) : null;
    const groupData = groupSnap?.exists() ? groupSnap.data() : null;
    return {
      id: userData.id,
      name: userData.full_name,
      photo: userData.photo_url,
```

```
      role: roleData ? roleData.role_name : 'Невідомо',
      group: groupData ? groupData.group_name : 'Невідомо'
    };
  } catch (error) {
    console.error("Error fetching user details:", error);
    return null;
  }
};
```

Примітка. Код отримує дані про поточного користувача

Крок 4. Отримання та фільтрація розкладу:

Для отримання розкладу я реалізував функцію *getSchedule*, яка запитує дані з колекції *schedule*, включаючи деталі дисциплін, груп, викладачів та часу проведення занять. Враховуючи роль користувача, я фільтрую розклад, щоб студенти бачили лише свої пари, а викладачі — тільки ті, які вони ведуть.

Лістинг коду 3.4. scheduleService.jsx

```
import { getDocs, getDoc } from 'firebase/firestore';
import { scheduleRef } from '../config/firebaseConfig';
// Функція для форматування часу
const formatTime = (date) => {
  return date.toLocaleTimeString('uk-UA', { hour: '2-digit', minute: '2-digit' });
};
// Функція для отримання даних викладача
const getTeacherData = async (teacherId) => {
  if (!teacherId) return { name: 'Невідомо', meetLink: 'Невідомо', photo: 'Невідомо'
};
  try {
    const teacherSnap = await getDoc(teacherId);
    if (teacherSnap.exists()) {
      const teacherData = teacherSnap.data();
      return {
        name: teacherData.full_name || 'Невідомо',
        meetLink: teacherData.google_meet_link || 'Невідомо',
        photo: teacherData.photo_url || 'Невідомо',
      };
    }
  } catch (error) {
    console.error("Error fetching teacher data:", error);
  }
  return {
name: 'Невідомо', meetLink: 'Невідомо', photo: 'Невідомо'
 };
};
// Функція для отримання розкладу
```

```
export const getSchedule = async (user) => {
  try {
    const scheduleSnapshot = await getDocs(scheduleRef);
    const scheduleList = await Promise.all(scheduleSnapshot.docs.map(async (doc) => {
      const data = doc.data();
      const courseSnap = await getDoc(data.course_id);
      const groupSnap = await getDoc(data.group_id);
      const teacherId = courseSnap.exists() ? courseSnap.data().teacher_id : null;
      const teacherData = await getTeacherData(teacherId);
      const sessionDate = data.session_date.toDate();
      const startTime = formatTime(sessionDate);
      const endDate = new Date(sessionDate.getTime() + 80 * 60000);
      const endTime = formatTime(endDate);
      const currentTime = new Date();
      const tenMinutesBeforeStart = new Date(sessionDate.getTime() - 10 * 60000);
      const isCurrentClass = currentTime >=
tenMinutesBeforeStart && currentTime <= endDate;
      return {
        id: doc.id,
        ...data,
        course_name: courseSnap.exists() ? courseSnap.data().course_name : 'Невідомо',
        group_name: groupSnap.exists() ? groupSnap.data().group_name : 'Невідомо',
        session_date: sessionDate,
        teacher_name: teacherData.name,
        teacher_meet: teacherData.meetLink,
        teacher_photo: teacherData.photo,
        start_time: startTime,
        end_time: endTime,
        is_current_class: isCurrentClass,
      };
    }));
    // Фільтрація розкладу на основі ролі користувача
    const filteredScheduleList = scheduleList.filter(item => {
      if (user.role === 'Студент') {
        return item.group_name === user.group;
      } else if (user.role === 'Викладач') {
        return item.teacher_name === user.name;
      }
      return false;
    });
    // Сортування розкладу за датою
    const sortedScheduleList = filteredScheduleList.sort((a, b) => a.session_date -
b.session_date);
    return sortedScheduleList;
  } catch (error) {
    console.error("Error fetching schedule:", error);
    return [];
  }
};
```

Примітка. Код отримує дані про пари та відповідно фільтрує їх

Крок 5. Підколекція присутності:

Для відслідковування присутності студентів на заняттях я реалізував функції *createAttendanceSubCollection* та *updateStudentAttendance*. Перша функція створює підколекцію *attendance* для кожного заняття, де зберігається інформація про студентів та їхній статус присутності, а друга оновлює статус студента на "Присутній".

Лістинг коду 3.5. attendanceService.jsx

```
import { doc, collection, setDoc, getDocs, query, where } from 'firebase/firestore';
import { usersRef, scheduleRef } from '../config/firebaseConfig';
export const createAttendanceSubCollection = async (currentClass) => {
  try {
    const groupRef = currentClass.group_id;
    const studentsQuery = query(usersRef, where('group_id', '==', groupRef));
    const studentsSnapshot = await getDocs(studentsQuery);
    if (studentsSnapshot.empty) {
      console.warn('Група не має студентів.');
      return;
    }
    const students = studentsSnapshot.docs.map((doc) => ({
      id: doc.id,
      full_name: doc.data().full_name,
      email: doc.data().email,
      photo_url: doc.data().photo_url || 'Невідомо',
      attendance_status: 'Відсутній',
    }));
    const attendanceRef = collection(doc(scheduleRef, currentClass.id), 'attendance');
    await Promise.all(
      students.map((student) =>
        setDoc(doc(attendanceRef, student.id), {
          full_name: student.full_name,
          email: student.email,
          photo_url: student.photo_url,
          attendance_status: student.attendance_status,
        })
      )
    );
    console.log('Підколекцію attendance успішно створено.');
  } catch (error) {
    console.error('Помилка під час створення підколекції attendance:', error);
  }
};
export const updateStudentAttendance = async (currentClass, studentId) => {
  try {
    const attendanceRef = collection(doc(scheduleRef, currentClass.id), 'attendance');
    // Оновлюємо статус присутності для студента
```

```
    await setDoc(doc(attendanceRef, studentId), {
      attendance_status: 'Присутній',
    }, { merge: true });
    console.log('Статус присутності студента оновлено.');
  } catch (error) {
    console.error('Помилка при оновленні статусу присутності:', error);
  }
};
```

Примітка. Код надає можливість керувати присутністю студентів

Крок 6. Отримання новин та подій:

Для отримання новин і подій я створив функції *getNews* та *getEvents*, які запитують дані з відповідних колекцій *news_Academic і main_Events* та фільтрують майбутні події.

Лістинг коду 3.6. newsService.jsx та eventService.jsx

```
import { getDocs, collection, query, where } from 'firebase/firestore';
import { eventRew } from '../config/firebaseConfig';

export const getEvents = async () => {
    try {
        // Отримуємо сьогоднішню дату в форматі UTC для порівняння
        const today = new Date();
        const eventsCol = eventRew;
        // Створюємо запит на фільтрацію майбутніх подій без завантаження всіх даних
        const eventsQuery = query(
            eventsCol,
            where("dateTime", ">=", today) // Фільтруємо події, що відбудуться після
сьогодні
        );
        const eventsSnapshot = await getDocs(eventsQuery);
        const eventsList = eventsSnapshot.docs.map(doc => {
            const data = doc.data();
            const dateTime = data.dateTime.toDate();
            const date = dateTime.toLocaleDateString('uk-UA', {
                year: 'numeric',
                month: 'long',
                day: 'numeric',
            });
            const time = dateTime.toLocaleTimeString('uk-UA', {
                hour: '2-digit',
                minute: '2-digit',
            });
            return {
                id: doc.id, dateTime,
                date,
```

```
                time,
                description: data.description,
                title: data.title,
                url_adress: data.url_adress
            };
}).sort((a, b) => a.dateTime - b.dateTime);
        return eventsList;
    } catch (error) {
        console.error("Error fetching events:", error);
        return [];
    }};
import { getDocs } from 'firebase/firestore';
import { newsRew } from '../config/firebaseConfig';
export const getNews = async () => {
    try {const newsSnapshot = await getDocs(newsRew);
        const newsList = newsSnapshot.docs.map(doc => ({
            id: doc.id, ...doc.data()
        })
);
        return newsList;
    } catch (error) {
        console.error("Error fetching news:", error);
        return [];
    }
};
```

Примітка. Код отримує дані про новини та події академії

На цьому етапп я створив основну структуру для роботи з даними в Firebase, що дозволяє ефективно інтегрувати ці функціональності в мобільний додаток.

3.3.6. Базова конфігурація додатку та авторизація

Оскільки додаток потребує гнучкої навігації, було прийнято рішення використовувати підхід із табовою навігацією, яка дозволяє користувачам легко переміщатися між різними частинами додатка. Кожна вкладка відповідає за певну функціональність:

- Головна сторінка: показує новини та події академії.
- Розклад: дозволяє студентам переглядати розклад занять, а викладачам — тільки ті курси, які вони ведуть.

- Сканер присутності: дозволяє викладачам генерувати QR-коди для відмітки присутності студентів, а студентам сканувати QR-коди.
- Чат: забезпечує можливість для студентів та викладачів спілкуватися.
- Профіль користувача: дає можливість вийти з додатка.

Налаштування навігації: для реалізації навігації між екранами використано бібліотеку *expo-router*, яка дозволяє організувати маршрутизацію за допомогою декларативних компонентів. Структура файлів відповідає модульному підходу, де кожна сторінка представлена окремим компонентом. Для організації табової навігації використовувався компонент *Tabs* із зазначенням екранів та їх налаштувань. Табова навігація є дуже зручною для мобільних додатків, оскільки забезпечує доступ до основних функцій додатка без зайвих переходів (рис 3.5).

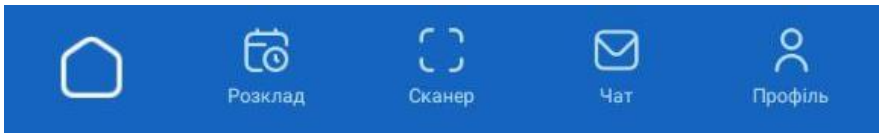

Рис. 3.5. Вигляд табової навігації

Джерело: Розроблено автором

Логіка авторизації та відображення сторінок: я додав обробку авторизації через контекст, що дозволяє перевіряти, чи авторизований користувач. Це відбувається у компоненті *MainLayout*, де на основі значення *isAuthenticated* ми перенаправляємо користувача на відповідну сторінку:

- Якщо користувач авторизований і знаходиться в додатку, він буде спрямований на головну сторінку.
- Якщо не авторизований — на сторінку привітання (welcome_Page).

Лістинг коду 3.7. MainLayout.jsx

```
import React, { useEffect } from 'react'
import { Slot, useRouter, useSegments } from 'expo-router'
import { AuthContextProvider, useAuth } from "../context/authContext"

const MainLayout = () => {
  const { isAuthenticated } = useAuth();
  const segments = useSegments();
  const router = useRouter();
```

```
  useEffect(() => {

    if (typeof isAuthenticated == 'undefined') return;
    const inApp = segments[0] == '(tab)';
    if (isAuthenticated && !inApp) {
      router.replace('home_Page');
    } else if (isAuthenticated == false) {
      router.replace('welcome_Page');
    }
  }, [isAuthenticated])

  return <Slot />
}

const RootLayout = () => {

  return (
    <AuthContextProvider>
      <MainLayout />
    </AuthContextProvider>
  );
};
export default RootLayout;
```

Примітка. Код обробляє стан авторизації та перенаправляє на відповідні сторінки

Створення контексту для авторизації: я створив контекст автентифікації для мого React-додатку, використовуючи *createContext* та *useState*. Цей контекст дозволяє зберігати інформацію про користувача та його автентифікаційний статус, а також надає функції для входу та виходу з системи.

У компоненті **AuthContextProvider** я реалізував:

- Відновлення сесії при завантаженні додатку через *useEffect* - перевіряється, чи є збережена сесія користувача, і на основі цього встановлюється його статус.
- Функцію **login**, яка перевіряє email і пароль користувача в базі даних Firestore. Якщо дані правильні, зберігається інформація про користувача в стані та сесії.
- Функцію **logout**, яка очищає сесію та скидає статус автентифікації на false.

За допомогою хука *useAuth*, я можу отримувати доступ до стану користувача та використовувати ці дані в будь-якому компоненті додатку. Це дозволяє

централізовано керувати автентифікацією та передавати інформацію про користувача між компонентами без необхідності її повторно отримувати.

Лістинг коду 3.8. authContext.jsx

```
import { createContext, useContext, useEffect, useState } from 'react';
import { setSession, getSession, clearSession, usersRef } from
'../config/firebaseConfig';
import { query, where, getDocs } from 'firebase/firestore';
import { getUserDetails } from '../services/userService'

export const AuthContext = createContext();

export const AuthContextProvider = ({ children }) => {
    const [user, setUser] = useState(null);
    const [isAuthenticated, setIsAuthenticated] = useState(undefined);

    useEffect(() => {
        const restoreSession = async () => {
            const storedUser = await getSession();
            setUser(storedUser);
            setIsAuthenticated(!!storedUser);
        };
        restoreSession();
    }, []);
    const login = async (email, password) => {
        try {
            const q = query(usersRef, where("email", "==", email), where("password",
"==", password));
            const snapshot = await getDocs(q);
            if (!snapshot.empty) {
                const doc = snapshot.docs[0];
                const userData = doc.data();
                const userDetails = await getUserDetails(userData);
                userDetails.id = doc.id;
                setUser(userDetails);
                setIsAuthenticated(true);
                await setSession(userDetails);
                return { success: true };
            }
            return { success: false, msg: 'Невірний email або пароль' };
        } catch (e) {
            return { success: false, msg: e.message };
        }
    };
    const logout = async () => {
        await clearSession();
        setUser(null);
        setIsAuthenticated(false);
```

```
        return { success: true };
    };
    return (
        <AuthContext.Provider value={{ user, isAuthenticated, login, logout }}>
            {children}
        </AuthContext.Provider>
    );
};
export const useAuth = () => {
    const context = useContext(AuthContext);
    if (!context) {
        throw new Error('useAuth must be used within an AuthContextProvider');
    }
    return context;
};
```

Примітка. Код дозволяє зберігати інформацію про користувача та його автентифікаційний статус

Реалізація початкової сторінки: на цій сторінці я додав інтерфейс з привабливим вітальним екраном і кнопкою для переходу до сторінки входу (рис 3.6).

Рис. 3.6. Демонстрація початкової сторінки

Джерело: Розроблено автором

Реалізація сторінки входу з перевіркою даних: компонент **LoginPage** є екраном для авторизації, де користувач вводить свою електронну пошту та пароль. Цей компонент включає валідацію полів, обробку помилок та інтерфейс з інтерактивними елементами, що дозволяє забезпечити зручність користування (рис 3.7).

У компоненті ми використовуємо кілька стейтів:

- **loading** - для відображення індикатора завантаження, коли ми очікуємо на результат логіну. Це дає користувачу зрозуміти, що процес авторизації триває.
- **emailError і passwordError** - для відображення помилок, якщо користувач введе неправильні дані.

Функція **handleLogin** виконує важливу роль - перевірку введених даних (email та пароль):

- Спочатку очищуємо попередні помилки, якщо такі були.
- Потім перевіряємо:
    - чи введено email (якщо ні, показуємо помилку);
    - чи email має правильний формат (@oa.edu.ua);
    - чи введено пароль (якщо ні, також показуємо помилку).
- Якщо всі перевірки успішні, викликаємо метод login із контексту авторизації, який спробує авторизувати користувача за допомогою введених даних.
- Якщо авторизація не вдалася, ми показуємо помилку для пароля.

Такий підхід дозволяє не тільки забезпечити безпеку при введенні даних, але й полегшує користувачам процес входу в систему, надаючи чіткі вказівки у разі помилок.

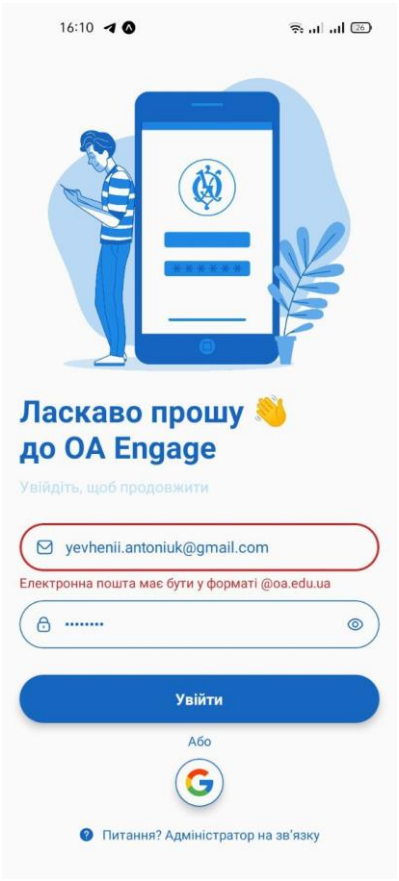

Рис. 3.7. Демонстрація сторінки входу

Джерело: Розроблено автором

3.3.7. Налаштування всіх функцій та сторінок додатку

Реалізація головної сторінки: на початку я створив головну сторінку для мобільного додатку, яка містить кілька важливих елементів для взаємодії з користувачем та відображення актуальної інформації. Ця сторінка має інтерфейс, що надає користувачам зручний доступ до основних функцій додатку, таких як перегляд новин, подій, та можливість здійснення пошуку.

Ось більш детальний опис того, як реалізовано кожен з компонентів:

1. **Інтерфейс користувача:**
   - З самого верху екрану розміщена панель з привітанням користувача та його ім'ям. Ця інформація отримується з контексту автентифікації за допомогою

хука *useAuth* (з контексту *authContext*). Якщо користувач має фото, воно відображається поруч з ім'ям.

- Крім того, є кнопка для сповіщень (іконка дзвінка), яку можна натискати. Вона стилізована через компонент *ButtonCustom* та використовує іконку *NotificationIcon*.
- Під панеллю користувача є інпут для пошуку, який дозволяє користувачу вводити запит, щоб шукати різні дані (за допомогою компонента *InputCustom*).

2. **Завантаження та відображення новин і подій:**
   - За допомогою функції *fetchData* та за допомогою асинхронних запитів отримуються новини (*news*) та події (*events*) через сервіси *getNews* і *getEvents*. Дані зберігаються у станах *news* та *events*.
   - Якщо є необхідність оновити дані, то реалізовано механізм оновлення через *RefreshControl*. Коли користувач протягує список для оновлення, викликається функція *onRefresh*, яка спочатку включає індикатор завантаження, потім оновлює дані, а після завершення оновлення знову вимикає індикатор.
3. **Відображення новин і подій:**
   - Новини відображаються за допомогою компонента *Sider*, який отримує дані з масиву *news*.
   - Події відображаються внизу екрану через компонент *Categories*, який отримує дані з масиву *events*.

Таким чином, ця сторінка виконує роль головної інформаційної панелі для користувача, де він може побачити своє ім'я та фото, отримати доступ до новин і подій, а також здійснити пошук за допомогою пошукової панелі (рис 3.8).

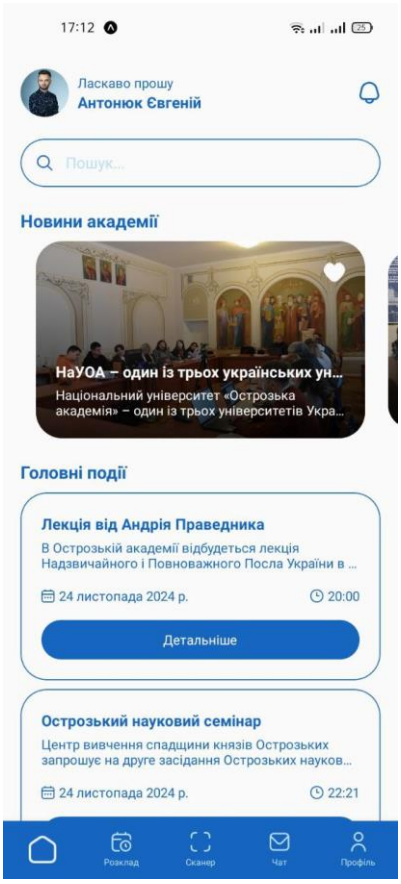

Рис. 3.8. Демонстрація головної сторінки

Джерело: Розроблено автором

Реалізація сторінки розкладу відповідно до ролі: у моєму додатку створення сторінки розкладу проходило через кілька ключових етапів. Головним завданням було забезпечити можливість відображення, оновлення та взаємодії з розкладом як для студентів, так і для викладачів. Ось як це було реалізовано:

1. **Ініціалізація компонента SchedulePage**

Спочатку я створив компонент *SchedulePage*, який є контейнером для всієї логіки роботи з розкладом:

- **Імпорт необхідних бібліотек і функцій:** використав *react-native, react-native-safe-area-context*, а також власні модулі: *authContext* (для доступу до даних користувача) і *scheduleService* (для отримання розкладу з Firebase).
- **Локальний стан:** за допомогою *useState* зберігаю:
    - schedule - дані розкладу.
    - refreshing - стан рефрешу для оновлення.

- **Функції:**
    - fetchSchedule: асинхронно завантажує розклад через сервіс *getSchedule*.
    - handleRefresh: оновлює дані розкладу.

2. **Інтеграція з Firebase**

Розклад отримується через сервіс *getSchedule*. Ця функція приймає поточного користувача (*user*), ідентифікує його роль (студент або викладач) та завантажує дані з колекції *schedule*, застосовуючи фільтрацію відповідно до:

- групи студента;
- курсів, які викладає викладач.

3. **Використання компонента ScheduleComponent**

Основну логіку та інтерфейс для роботи з розкладом виніс у дочірній компонент *ScheduleComponent*. Його функціональність включає:

- **Підтримка локалізації:** використав бібліотеку *react-native-calendars*, задавши українську локалізацію (*LocaleConfig*).
- **Перетворення даних:** у agendaItems згрупував розклад за датами для сумісності з *Agenda* (бібліотека react-native-calendars).
- **Рендеринг подій:**
    - Створив кастомні елементи для подій із відображенням:
        - часу (start_time, end_time);
        - типу заняття (session_type);
        - формату (онлайн або офлайн);
        - додаткової інформації (група, аудиторія, Google Meet).
- **Модальне вікно для відвідуваності:**
    - Додав можливість для викладача відкривати модальне вікно, яке показує дані про відвідуваність (підтягуються з підколекції *attendance*).
    - У модалі реалізована обробка стану присутності (наприклад, "Присутній"/"Відсутній").

4. **Логіка для різних ролей**

Інтерфейс адаптований під роль користувача:

- **Для студентів:**
  - Відображаються деталі заняття, ім’я викладача, фото та посилання на Google Meet (для онлайн-занять).
- **Для викладачів:**
  - Відображається назва групи, можливість перегляду відвідуваності студентів.

5. **Підтримка рефрешу та порожніх даних**
   - **Рефрешинг:** через властивість *refreshing* у Agenda та функцію *handleRefresh*.
   - **Порожні дані:** якщо на певну дату немає занять, відображається кастомний екран із повідомленням та ілюстрацією.

Сторінка розкладу розроблена як зручний та функціональний інструмент для студентів і викладачів. Вона дозволяє швидко отримувати необхідну інформацію про заняття, переглядати деталі та взаємодіяти з іншими модулями додатку. Інтуїтивний інтерфейс, мінімалістичний дизайн і добре структуровані дані забезпечують позитивний користувацький досвід (рис 3.9).

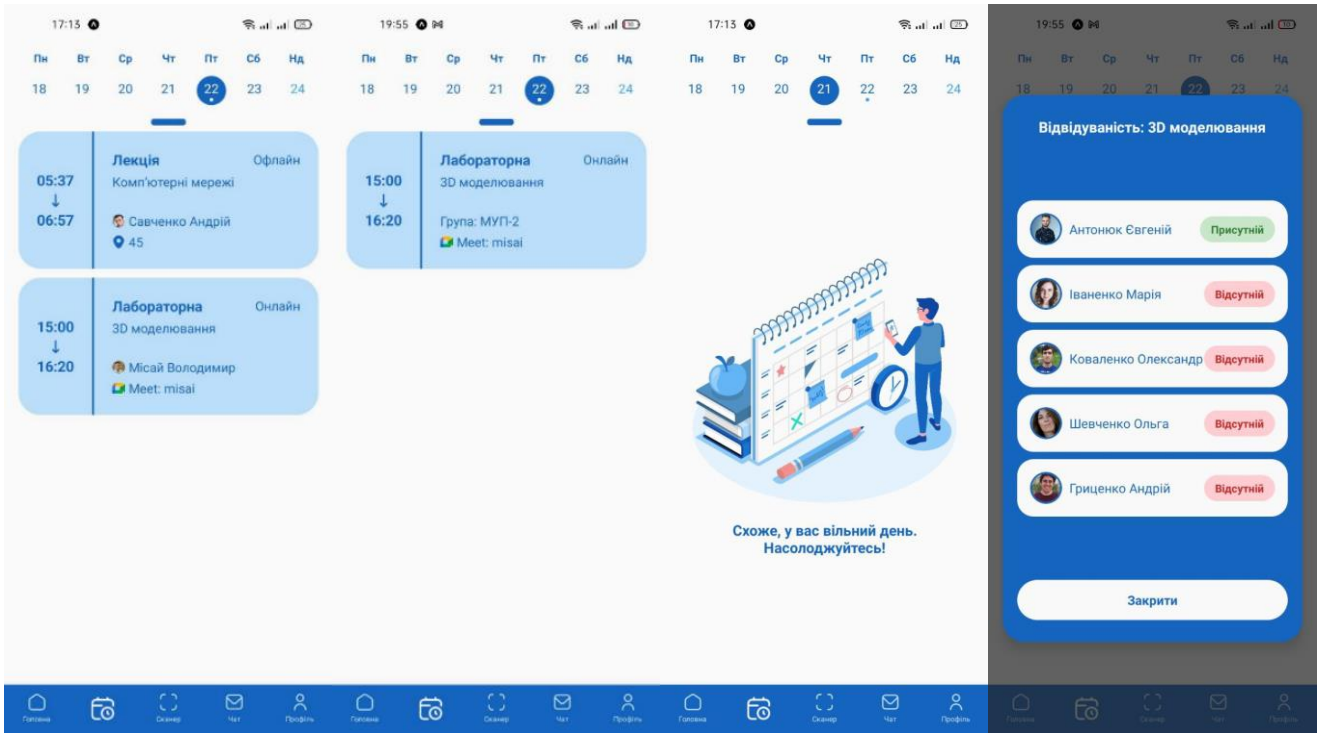

Рис. 3.9. Демонстрація сторінки розкладу для двох ролей

Джерело: Розроблено автором

Реалізація сторінки сканування та генерації QR-коду відповідно до ролі: сторінка відвідуваності розроблена для мобільного додатку та забезпечує функціональність сканування QR-кодів для студентів і генерації їх для викладачів. Ця функціональність є ключовою частиною процесу відмітки присутності на заняттях.

Для студентів сторінка містить модуль для сканування QR-коду, який швидко розпізнає інформацію про заняття, пов’язану з кодом, і передає ці дані у Firebase. Для викладачів реалізовано генерацію унікального QR-коду, який містить інформацію про заняття, зокрема назву курсу, дату та час. Викладачі можуть показати цей код студентам для сканування. Ось як вона створювалась.

1. **Визначення ролі користувача**

За допомогою контексту аутентифікації *useAuth* я отримав дані про поточного користувача (зокрема, його роль). Це дозволило:

- Відображати відповідний функціонал:
    - **Студент:** кнопка для сканування QR-коду.
    - **Викладач:** кнопка для генерації QR-коду.
- Обмежити доступ до функцій, що не відповідають ролі користувача.

2. **Організація станів**

Я додав кілька станів за допомогою хука *useState*:

- *isModalOpen*: контролює видимість модального вікна.
- *cameraActive*: активує або деактивує камеру для сканування.
- *qrValue*: зберігає унікальне значення QR-коду, яке використовується для перевірки.

Ці стани допомагають управляти поведінкою компонентів у реальному часі.

3. **Обробка взаємодії викладача**

Для викладача реалізовано функцію генерації QR-коду:

- **handleStartScanning:**
    - Отримує дані про поточний розклад через функцію *getSchedule(user)*.

- Шукає поточне заняття за допомогою фільтрації (перевіряється поле *is_current_class*).
- Якщо заняття знайдено:
    - Викликається функція *createAttendanceSubCollection*, яка створює підколекцію *attendance* для запису присутності.
    - Генерується значення QR-коду (ідентифікатор заняття) і передається у компонент *QRCode*.

Інтеграція з Firebase: підколекція *attendance* у структурі Firestore дозволяє викладачу переглядати відвідуваність студентів.

4. Обробка взаємодії студента

Для студента реалізовано функцію сканування QR-коду:

- **handleStudentAttendance:**
    - Отримує дані про поточний розклад.
    - Шукає поточне заняття за аналогічним принципом.
    - Оновлює статус присутності через функцію *updateStudentAttendance*.

Процес сканування

- Користувач натискає кнопку "**Відсканувати QR-code**", що активує камеру через *CameraView*.
- Камера сканує QR-код викладача.
- Якщо відскановане значення співпадає з *qrValue*, викликається *updateStudentAttendance* для запису присутності.

5. Модальне вікно

Для спрощення роботи із спливаючими вікнами я створив кастомний компонент *Modal*:

- Він обгортає стандартний компонент *RNModal* і додає стилізацію з затемненням фону.
- Вікно відображає різний контент залежно від ролі:

- Для студента: активна камера для сканування.
- Для викладача: згенерований QR-код.

Модальне вікно відкривається через *handleOpenModal* і закривається через *handleCloseModal*.

6. Інтеграція з Firebase

Для інтеграції з Firestore я додав наступні функції:

- **getSchedule(user):**
  - Отримує розклад для поточного користувача (фільтрується за групою студента або викладача).
  - Використовується для визначення поточного заняття.
- **createAttendanceSubCollection(currentClass):**
  - Створює підколекцію *attendance* для поточного заняття, якщо її ще немає.
- **updateStudentAttendance(currentClass, userId):**
  - Додає запис про присутність студента (поля: email, full_name, photo_url, attendance_status).

7. Динамічний інтерфейс

Кінцевий вигляд сторінки адаптується залежно від станів і ролей:

- Студентам і викладачам відображаються різні текстові інструкції та кнопки.
- QR-коди, модальні вікна та камера змінюють свою поведінку в залежності від взаємодії.

Таким чином, створена готова сторінка *AttendancePage*, яка забезпечує функціонал реєстрації відвідуваності занять. Вона враховує роль користувача (студент або викладач), динамічно адаптує інтерфейс і автоматизує процес зберігання даних у Firestore. Викладачі можуть генерувати QR-коди для занять, а студенти сканують їх, підтверджуючи свою присутність. Реалізація інтегрує сучасні технології, такі як камеру пристрою, бібліотеку QR-кодів і Firestore, що робить сторінку зручною, функціональною та готовою до використання (рис 3.10).

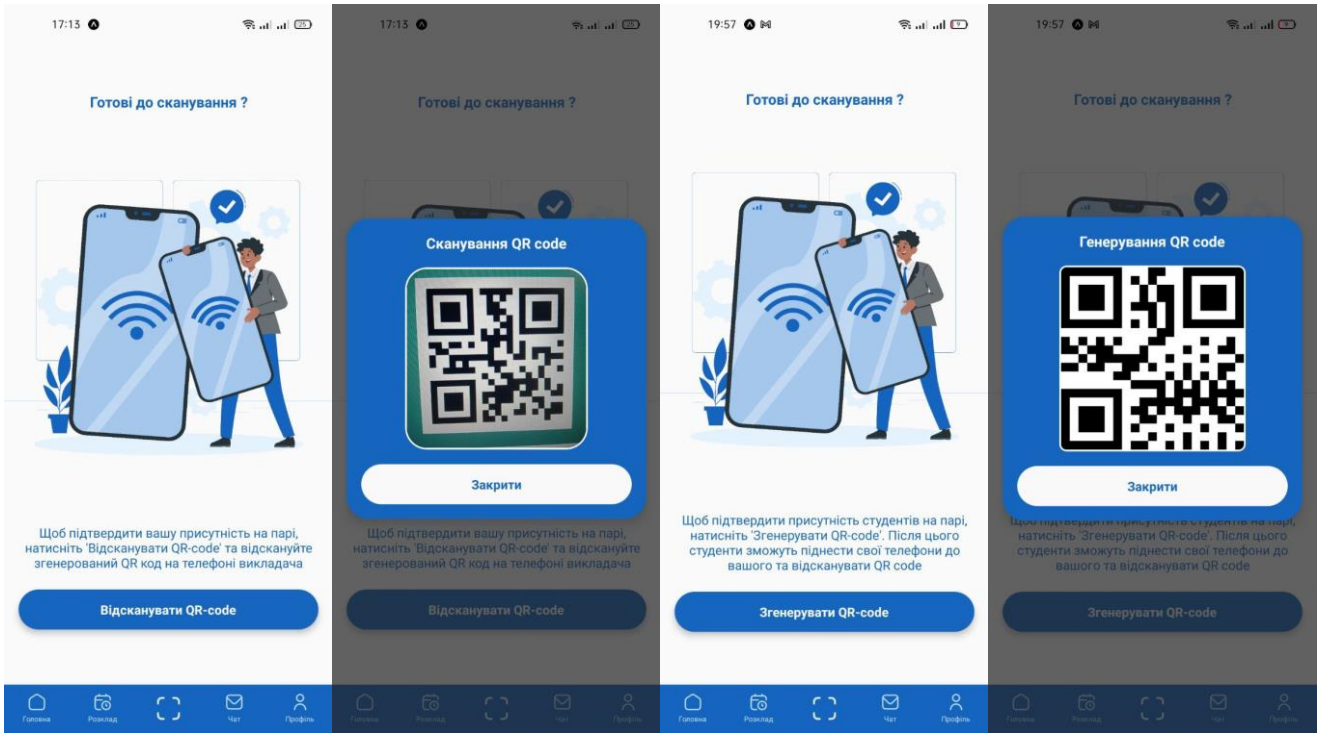

Рис. 3.10. Демонстрація сторінки відвідуваності для двох ролей

Джерело: Розроблено автором

Реалізація чату : при створенні сторінки чату в додатку, я реалізував кілька важливих компонентів і функціональностей для забезпечення інтерактивності, зручності користування та синхронізації з Firebase. Ось детальний опис процесу:

1. **Створення загальної сторінки чату (ChatPage)**

Це головна сторінка, де користувач бачить список доступних контактів для чату.

- Отримання списку користувачів:
  - Використовую *useEffect*, щоб завантажити дані про користувачів через функцію *getAllUsers*.
  - Дані зберігаються у стані *users* за допомогою *useState*.
- Інтеграція компонента TabScreen:
  - Передаю список користувачів та поточного користувача як пропси.
  - *TabScreen* відповідає за відображення списку контактів.

2. **Реалізація кімнати чату (ChatRoom)**

Це екран, де користувач може спілкуватися з іншими.

- Створення кімнати в Firebase:
    - Використовую функцію *createRoomIfNotExists*, яка перевіряє існування кімнати чату за унікальним *roomId* (генерується функцією *getRoomId*).
    - Якщо кімнати не існує, створюю новий документ у колекції *rooms*.
- Отримання повідомлень:
    - Використовую *onSnapshot*, щоб слухати зміни в колекції *messages* для відповідної кімнати.
    - Повідомлення сортуються за часом створення (*orderBy('createdAt', 'asc')*) та зберігаються у стані *messages*.
- Відправлення повідомлень:
    - Вміст текстового поля зберігається у *textRef*, а функція *hanldeSendMessage* додає повідомлення до колекції *messages* в Firebase.
    - Після успішного відправлення текстове поле очищується.
- Адаптація до клавіатури:
    - При появі клавіатури список повідомлень автоматично прокручується до кінця за допомогою *scrollViewRef*.

3. Компоненти для чату

Для реалізації функціональності чату я розробив окремі компоненти:

- ChatRoomHeadr:
    - Відображає інформацію про співрозмовника (аватар та ім'я).
    - Містить кнопку "Назад" для повернення до списку чатів.
- MessageList:
    - Рендерить список повідомлень за допомогою компонента *ScrollView*.
    - Використовує *MessageItem* для відображення кожного повідомлення.
- MessageItem:
    - Відображає текст повідомлення.

- Повідомлення вирівнюються по різні боки залежно від того, хто є автором (поточний користувач чи співрозмовник).

4. **Відображення списку контактів (ChatItem)**

На головній сторінці чату кожен контакт представлений у вигляді картки:

- Оновлення останнього повідомлення:
  - Використовую *onSnapshot* для отримання останнього повідомлення в кімнаті.
  - Виводжу текст останнього повідомлення та його час.
- Перехід до чату:
  - При натисканні на контакт використовую *router.push*, щоб перейти до відповідної кімнати.

5. **Інтерактивний інтерфейс**

- Додано анімаційний компонент *TabButton*, що забезпечує інтерактивну зміну вкладок.

Ця реалізація забезпечує інтерактивний досвід користувача з функціоналом реального часу, мінімальною затримкою та зручним інтерфейсом для чату(рис 3.11).

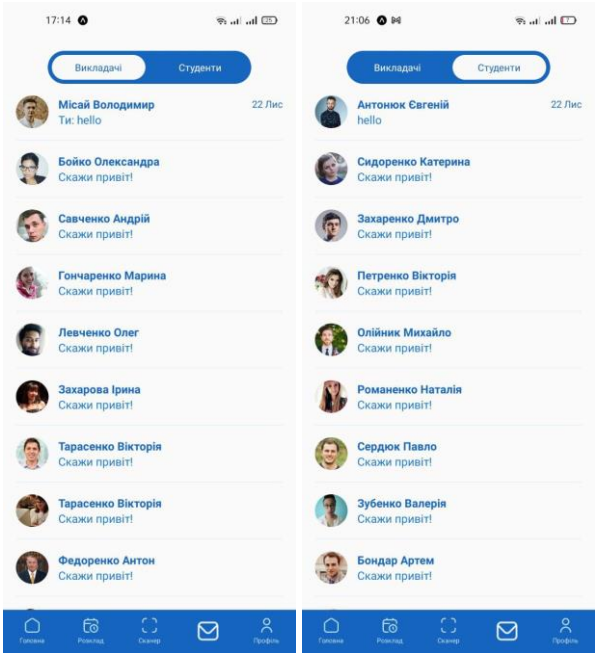

Рис. 3.11. Демонстрація сторінки чату

Джерело: Розроблено автором

Реалізація сторінки профілю: на сторінці профілю я реалізував кілька функціональних елементів, щоб забезпечити зручний інтерфейс для користувачів (студентів та викладачів). Ось як я розробив цю сторінку:

- **Отримання даних користувача**
  - Використовуючи контекст автентифікації (через *useAuth*), я отримав дані користувача, зокрема ім'я, роль, групу, фотографію профілю та інші характеристики. Це дозволяє персоналізувати відображення для кожного користувача.
  - Зокрема, відображаю: ім'я користувача, його роль (студент або викладач), та за наявності групу (для студента).
- **Кнопки для різних дій**
  - Я створив кілька кнопок для користувача, кожна з яких відповідає за певну дію:
    - **Залікова книжка** - це кнопка для студента, що, веде до функціоналу перегляду залікової книжки (майбутнє розширення).
    - **Розклад дзвінків** - кнопка для студентів та викладачів, яка відкриває розклад дзвінків (майбутнє розширення).
    - **Корисні посилання** - ще одна кнопка, яка може містити доступ до корисних ресурсів або зовнішніх веб-сторінок (майбутнє розширення).
    - **Вийти з додатку** - важлива кнопка для виходу користувача з додатку. Цю функцію я реалізував через виклик функції *logout()* з контексту автентифікації, що очищує сесію користувача і переносить його назад на екран авторизації.

Таким чином, на сторінці профілю користувач отримує всю необхідну інформацію про себе, а також має доступ до різноманітних функцій, таких як вихід з додатку, перегляд розкладу дзвінків та корисних посилань. Кожна кнопка та елемент інтерфейсу на сторінці профілю має чітке призначення та інтуїтивно зрозуміле розташування, що дозволяє користувачу без зусиль взаємодіяти з додатком.

Інтерфейс побудовано з урахуванням вимог зручності та простоти використання, а також адаптовано для мобільних пристроїв, що гарантує комфортне користування додатком навіть на малих екранах (рис 3.12).

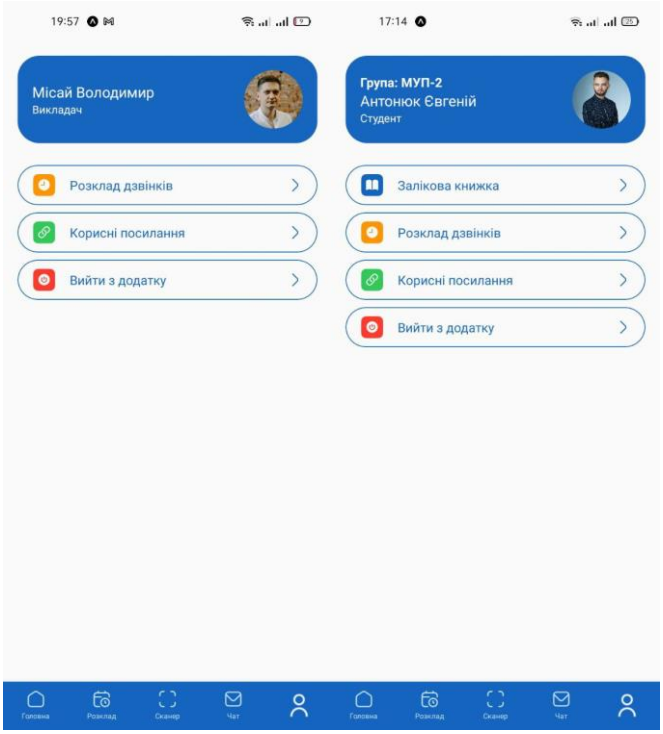

Рис. 3.12. Демонстрація сторінки профілю

Джерело: Розроблено автором

3.4. Керівництво користувача

Функціональне тестування є важливою частиною процесу тестування програмного забезпечення. Його основне завдання - перевірка відповідності функціональних характеристик програмного продукту встановленим вимогам. В ході функціонального тестування виконуються різноманітні тести, спрямовані на перевірку коректності роботи та функціональності програмного забезпечення.

За результатами тестування складається звіт, який дозволяє оцінити стан програмного продукту і виявити потенційні проблеми, які необхідно виправити перед релізом. Звіт може містити інформацію про кількість знайдених помилок, час, витрачений на тестування, а також показники ефективності та якості роботи

програмного продукту. Варто зазначити, що функціональне тестування є лише одним із етапів тестування програмного забезпечення і не охоплює інші види, такі як тестування продуктивності, безпеки та інші.

Таблиця 3.1

Результати функціонального тестування

№	Функція	Мета тестування	Кроки тестування	Очікуваний результат	Тестові дані
1	**Початкова сторінка**	Перевірка наявності вітального екрану та кнопки для переходу до сторінки входу	1. Відкрити додаток. 2. Перевірити наявність привітального екрану та кнопки для переходу до сторінки входу.	Кнопка для переходу є доступною, екран виглядає привабливо.	-
2	**Сторінка входу (LoginPage)**	Перевірка валідації даних та обробки помилок при введенні даних для входу	1. Ввести правильні дані (email: user@oa.edu.ua, password: correctPassword). 2. Ввести неправильний email або пароль.	Якщо email та пароль правильні, користувач авторизується. Якщо дані неправильні, з'являється помилка.	Правильні/неправильні email і пароль.
3	**Головна сторінка**	Перевірка доступу до основних функцій додатку: новини, події.	1. Перевірити наявність панелі з ім'ям користувача. 2. Перевірити відображення новин та подій після завантаження.	Панель з ім'ям користувача і фото працюють коректно. Новини та події завантажуються і відображаються.	-
4	**Сторінка розкладу (SchedulePage)**	Перевірка відображення розкладу відповідно до ролі користувача (студент/викладач)	1. Увійти як студент і перевірити, чи відображаються лише його предмети. 2. Увійти як викладач і перевірити, чи відображаються лише предмети, що він викладає. 3. Перевірити можливість рефрешу розкладу.	Відображення відповідних розкладів для кожної ролі, коректний рефреш.	Студент/викладач з конкретними предметами.
5	**Сканування QR-коду для студентів**	Перевірка сканування QR-коду для реєстрації присутності студента	1. Студент натискає кнопку для сканування QR-коду. 2. Сканує QR-код викладача.	Студент успішно сканує QR-код, присутність реєструється.	QR-код заняття.

			3. Перевірити, чи оновлюється статус присутності в Firestore.		
6	**Генерація QR-коду для викладачів**	Перевірка генерації QR-коду викладачем для реєстрації присутності студентів	1. Викладач натискає кнопку для генерації QR-коду. 2. Перевірити, чи генерується правильний QR-код для поточного заняття. 3. Перевірити, чи оновлюється підколекція attendance.	Викладач генерує QR-код для заняття, студент може його відсканувати для реєстрації присутності.	-
7	**Модальне вікно (Modal)**	Перевірка відображення модального вікна для різних ролей	1. Перевірити, чи відкривається модальне вікно для студента (для сканування QR-коду). 2. Перевірити, чи відкривається модальне вікно для викладача (для генерації QR-коду).	Модальне вікно відображається відповідно до ролі користувача, коректно закривається.	Студент/викладач.
8	**Інтеграція з Firebase (Attendance)**	Перевірка правильності інтеграції даних присутності зі структурою Firestore	1. Сканувати QR-код студента для запису присутності. 2. Перевірити, чи збережено правильні дані в підколекції attendance (email, name, статус).	Дані присутності коректно зберігаються у Firestore.	Студент з присутністю.

Usability тестування (тестування зручності використання) - це тип тестування, спрямований на оцінку зручності, ефективності та інтуїтивності інтерфейсу користувача мобільного додатку або вебсайту. Метою цього тестування є перевірка того, наскільки легко та швидко користувачі можуть виконувати завдання та досягати своїх цілей в межах додатку без труднощів, а також виявлення можливих проблем у взаємодії з інтерфейсом.

Завдяки тестуванню зручності використання можна виявити проблеми на ранніх етапах, таких як неінтуїтивно зрозуміле розташування кнопок, важкодосяжні функції або інші бар'єри, що можуть перешкоджати користувачам ефективно використовувати додаток. Таким чином, usability тестування допомагає покращити

загальне враження від додатку, зробити його більш доступним і комфортним для кінцевих користувачів.

Таблиця 3.2

Результати Usability тестування

№	Назва функціональності	Опис тесту	Очікуваний результат	Результат тесту
1	Взаємодія зі сторінкою відвідуваності	Оцінка інтуїтивності інтерфейсу сторінки відвідуваності для студента і викладача. Перевірка, чи легко користувачі можуть зрозуміти функціональність.	Користувач повинен без труднощів зрозуміти, як відзначати присутність та генерувати QR-код.	Пройдено
2	Чат	Оцінка зручності користування чат-функціоналом. Перевірка інтерфейсу для відправлення повідомлень, перегляду історії чату та наявності повідомлень.	Користувачі мають безперешкодно відправляти і отримувати повідомлення.	Пройдено
3	Загальний інтерфейс мобільного додатку	Оцінка загального користувацького досвіду. Перевірка навігації між сторінками, швидкість реагування елементів інтерфейсу та загальна зручність.	Інтерфейс має бути зручним, швидким і інтуїтивно зрозумілим для користувача.	Пройдено
4	Доступність для різних ролей користувачів	Перевірка доступності всіх функцій та елементів інтерфейсу для студентів та викладачів. Перевірка, чи коректно відображаються елементи в залежності від ролі.	Інтерфейс має відображати відповідні функції для кожної ролі.	Пройдено
5	Реакція на помилки та індикатори завантаження	Перевірка відображення індикаторів завантаження та повідомлень про помилки. Оцінка, наскільки швидко користувачі отримують необхідну інформацію.	Індикатори завантаження та помилки мають відображатись коректно, не викликаючи заплутаності.	Пройдено

3.5. Висновок до розділу 3

У процесі розробки кваліфікаційного проекту був визначений комплекс засобів розробки, що використовуватимуться для створення мобільного застосунку. Окремо були уточнені вимоги до технічного та програмного забезпечення, що забезпечують

стабільну та ефективну роботу програми. Детально розглянуто етапи програмної реалізації, включаючи використання інструментів для створення, тестування та налагодження продукту. Також описано процес оцінки якості програмного продукту та ефективності розробки, що сприяє високій якості кінцевого продукту.

Усі завдання були успішно виконані, тестування не виявило жодних проблем. Респонденти відзначили зручність користування, привабливий дизайн та адекватний рівень складності застосунку. Тестування зручності використання (usability test) пройшло успішно.

ВИСНОВКИ

У рамках розробки мобільного додатку OA Engage було створено ефективну систему для управління навчальним процесом, орієнтуючись на потреби студентів та викладачів. Оцінка результатів роботи базується на глибокому аналізі функціональних можливостей та особливостей розробленого продукту, а також на використанні сучасних технологій та інструментів для розробки мобільних додатків.

Додаток OA Engage має на меті оптимізувати організацію навчального процесу за допомогою інтеграції декількох ключових функцій: авторизація за допомогою домену @oa.edu.ua, перегляд новин академії та подій, доступ до розкладу занять з можливістю фільтрації для студентів і викладачів, чат для комунікації, а також функція відмітки присутності з використанням QR-кодів. Структура бази даних Firebase була ретельно спланована для підтримки цих функцій та забезпечення безперебійної роботи додатку. Аналіз наявних аналогів виявив низку слабких місць у мобільних додатках для навчальних закладів, зокрема відсутність інтегрованих функцій для студентів і викладачів в єдиному інтерфейсі. На основі цього було сформульовано завдання створити універсальний мобільний додаток з простим інтерфейсом, що відповідає потребам різних категорій користувачів і дає змогу зручно керувати навчальними процесами. Для реалізації проекту було обрано методологію гнучкої розробки, що дозволяє швидко адаптуватися до змін в процесі створення продукту. Розробка додатку проходила етапами: починаючи з прототипування в Figma та створення векторних ілюстрацій, до налаштування бази даних у Firebase та інтеграції з додатком. Застосування dbdiagram.io для моделювання бази даних дозволило візуалізувати структуру даних, що сприяло спрощенню процесу її реалізації у Firebase Firestore. Додаток розроблявся за допомогою технології React Native, що дозволяє створювати кросплатформенні додатки з високою продуктивністю. Expo було використано для тестування та емуляції додатку на різних пристроях. Firebase Firestore служить основною базою даних, забезпечуючи зберігання та доступ до даних, а також інтеграцію з іншими компонентами додатку, такими як авторизація, чат та відмітка присутності. У результаті розробки додатку

було створено зручний інтерфейс, який дозволяє користувачам швидко орієнтуватися в функціях додатку, відзначати присутність на заняттях, переглядати актуальні новини та події, а також користуватися всіма іншими доступними можливостями. Керівництво користувача надає детальну інформацію про роботу з кожною функцією додатку, що сприяє легкому освоєнню продукту.

Розробка додатку OA Engage стала успішним проектом, що відповідає всім основним вимогам до функціональності та зручності використання. Він охоплює ключові аспекти організації навчального процесу та взаємодії між студентами і викладачами. Використані технології, такі як React Native, Firebase, Figma та інші інструменти, дозволили реалізувати проект на високому технічному рівні. Подальша оптимізація та удосконалення додатку допоможуть забезпечити його стабільну роботу та додаткові функції для користувачів в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **Документація Firebase.** Офіційна документація Firebase для налаштування та роботи з Firestore, а також інтеграції з мобільними додатками. URL: https://firebase.google.com/docs/firestore
2. **Документація React Native**. Офіційний сайт React Native, що містить інформацію про розробку мобільних додатків для платформ iOS та Android. URL: https://reactnative.dev/docs/getting-started
3. **Документація Expo.** Офіційна документація фреймворку Expo, який використовується для створення React Native додатків. URL: https://docs.expo.dev/
4. **Документація Visual Studio Code.** Офіційна документація для IDE Visual Studio Code, що використовувалась для розробки мобільного додатку. URL: https://code.visualstudio.com/docs
5. **Документація Android Studio.** Офіційна документація для Android Studio, яка використовувалась для емуляції девайсу під час розробки. URL: https://developer.android.com/studio
6. **Документація Firebase Authentication.** Офіційна документація для використання Firebase Authentication для автентифікації користувачів через електронну пошту або інші методи. URL: https://firebase.google.com/docs/auth
7. **Документація Figma.** Офіційна документація для інструменту Figma, який використовувався для створення дизайну інтерфейсу користувача мобільного додатку. URL: https://www.figma.com/resources/learn-design/
8. **Документація Adobe Illustrator.** Офіційний ресурс для вивчення основ та можливостей програмного забезпечення Adobe Illustrator для графічного дизайну. URL: https://helpx.adobe.com/illustrator/tutorials.html
9. **Документація Adobe After Effects.** Офіційний сайт для навчання та використання Adobe After Effects для створення анімацій та відео-контенту. URL: https://helpx.adobe.com/after-effects/tutorials.html

10. **Ресурси по QR-кодах для мобільних додатків.** Посібники та інструменти для інтеграції технології QR-кодів у мобільні додатки. URL: https://www.npmjs.com/package/react-native-qrcode-svg
11. **Документація для роботи з Firestore у Node.js.** Офіційні приклади та інструкції для інтеграції Firestore з серверною частиною додатку. URL: https://firebase.google.com/docs/firestore/quickstart