КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ОРГАНІЗАЦІЇ СПІЛЬНОЇ РОБОТИ

Виконав студент 4-го курсу

Михайло ГОЛОВАЦЬКИЙ

_____
(підпис)

Науковий керівник
ассистент, кандидат фізико-математичних наук
Андрій КРИВОЛАП

_____
(підпис)

Засвідчую, що в цій курсові роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування

«____» _____ 202_ р.,

протокол № ___
Завідувач кафедри
Микола НІКІТЧЕНКО _____
(підпис)

Київ - 2023

РЕФЕРАТ

Кваліфікаційна робота: 45 с., 33 рис., 15 джерел

МОБІЛЬНИЙ ЗАСТОСУНОК, KOTLIN, FIREBASE.

Об'єктом роботи є процес управління та організації спільної роботои в команді.

Предметом роботи є мобільний застосунок для організації спільної роботи.

Мета роботи - створення функціонального застосунку, який допоможе полегшити робочі процеси та підвищити продуктивність роботи команди.

Методи та інструменти розробки: середовище розробки - Android Studio, мова програмування - Kotlin, платформа для збереження даних – Firebase.

Результатом роботи є функціональний мобільний застосунок, що забезпечує роботу з дошками, сприяє збору статистики та надає користувачам контроль та керованість над завданнями.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDE - Integrated Development Environment (інтегроване середовище розробки)

JVM - Java Virtual Machine (віртуальна машина Java)

XML - Extensible Markup Language (розшиирювана моова розмітки)

FCM - Firebase Cloud Messaging (хмарний обмін повідомленнями Firebase)

HTTP - HyperText Transfer Protocol (протокол передачі гіпертекстових документів)

TCP - Transmission Control Protocol (протокол керування передаванням)

IP - Internet Protocol (міжмережевий протокол)

ВСТУП

Оцінка сучасного стану об’єкта розробки. З урахуванням швидкого темпу технологічного розвитку та зростаючих вимог організації спільної роботи, потреба у зручних та ефективних мобільних застосунках для планування та контролю виконання завдань командами стає все більш актуальною.

Актуальність роботи та підстави для її виконання. Сьогоднішні організації все більше прагнуть до ефективного планування, координації та контролю робочих завдань, особливо коли мова йде про команди робітників, що працюють в розподіленому середовищі. В той самий момент мобільні пристрої стають все більш поширеними і потужними. Використання мобільних застосунків для організації спільної роботи стає зручним та доступним рішенням для бізнесу та інших галузей. Саме цьому, тема є дуже актуальною на сьогодні.

Головні підстави для виконання даної роботи включають:

- недоліки існуючих мобільних застосунків для організації спільної роботи;
- популярність мобільних пристроїв;
- розвиток технологій;
- потреба у покращенні ефективності організації командної роботи.

Мета й завдання роботи. Метою даної роботи є розробка мобільного застосунку, який надасть централізований інструмент для спільної роботи команди або групи користувачів.

Для виконання зазначеної вище мети було поставлено наступні завдання:

- аналіз існуючих мобільних застосунків для організації спільної роботи з метою виявлення їх переваг та недоліків;

- визначення вимог до застосугку, який забезпечуватиме зручне планування та організацію завдань, спільну роботу команди та контроль за виконанням завдань;
- вибір технологій та інструментів розробки, які найкраще відповідають поставленим вимогам та дозволять ефективно реалізувати функціональність застосунку;
- розробка архітектури та функціональності застосугку, включаючи створення зручного інтерфейсу користувача, механізмів планування та призначення завдань, моніторингу їх стану та виконання;
- тестування та оптимізація застосунку для забезпечення стабільної та ефективної роботи;
- висновки щодо виконання поставлених завдань, оцінка результатів дослідження та пропозиції щодо подальшого розвитку застосугку.

Можливі сфери застосування. Застосунок може знайти застосування в різних сферах діяльності, де важлива організація робочих процесів та спільна робота команди. Сферами застосування можуть бути:

- командна робота: застосунок може сприяти спільній роботі команд, де кожен учасник може бачити статус завдань, обмінюватися ідеями, виконувати завдання та спілкуватися з колегами;
- персональне планування: застосунок може бути використаний індивідуально для планування та керування особистими завданнями. Він допоможе користувачеві створити структурований розклад, встановити пріоритети та забезпечити ефективне **в**иконання завдань;

- навчання та освіта: застосунок може бути корисним для студентів, учнів, викладачів та інших осіб, що займаються навчанням або навчальними проєктами. Він допоможе організувати навчальний процес, планувати домашні завдання, контролювати їх виконання;
- індустрія розваг: застосунок може бути використаний в сфері розваг та організації подій, де потрібно планувати та виконувати різні завдання. Він допоможе координувати роботу персоналу, встановлювати терміни виконання та контролювати прогрес у підготовці подій.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ ЗАСТОСУНКУ

Перед тим як приступити до розробки власного мобільного застосунку для організації спільної роботи, важливо проаналізувати існуючі рішення на ринку. В даному розділі ми розглянемо деякі переваги та недоліки існуючих застосунків у цій сфері.

1.1 Застосунок Todoist

Todoist – мобільний застосунок для керування завданнями та організнації робочого потоку. Він дозволяє структурувати завдання за проєктами, папками та мітками. Серед його можливостей особливо виділяються: створення списків завдань, використання міток та нагадування, спільна робота над проєктами з іншими користувачами.

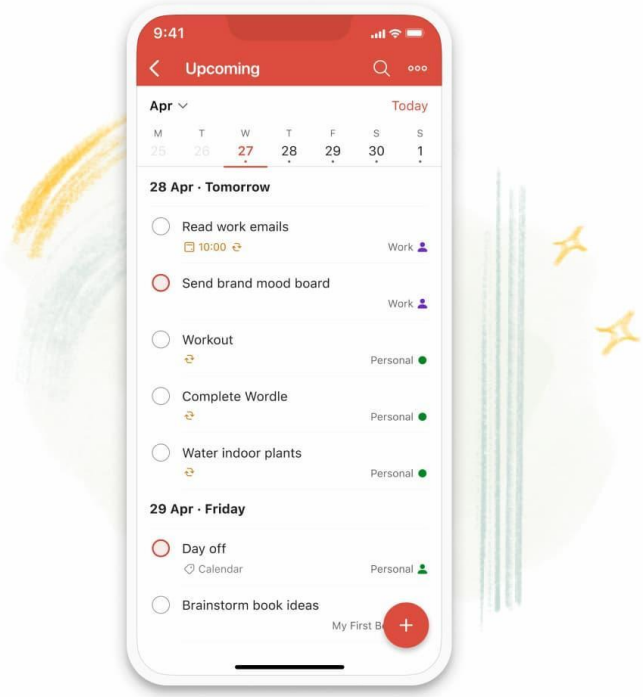


Рисунок 1.1 – Інтерфейс застосунку Todoist

У Todoist є підтримка спільної роботи над завданнями. За необхідності можна додавати користувачів до проєктів та розподіляти завдання.
Переваги застосунку Todoist:

- інтуїтивний інтерфейс. Простий і зрозумілий інтерфейс, який дозволяє швидко освоїти платформу і почати працювати;
- гнучкість налаштування. Є можливість встановлювати терміни виконання завдань, мітки та пріоритети;
- спільна робота. Todoist дозволяє ділитися завданнями з іншими користувачами, а також призначати виконавця;
- використання шаблонів. Є можливість створення шаблонів, які можна використовувати для повторюваних або регулярних завдань.

Недоліки:

- завдання можна призначати не більше ніж одному користувачу одночасно. Якщо потрібно розподілити виконання завдання між кількома учасниками – такого функціоналу не надається;
- обмеженість статусів завдань. Не має можливості відстежувати проміжні стани або прогрес виконання завдання;
- відсутня можливість управління правами учасників. Відсутність цієї функціональності може ускладнити керування та контроль над проєктом;
- відсутність україномовного інтерфейсу.

1.2 Застосунок Microsoft To Do

Microsoft To Do – безкоштовний застосунок для планування та управління завданнями, розроблений компанією Microsoft. Основні функції включають в себе створення завдань, списків завдань, встановлення строків, додавання нагадувань та інші. Завдання можна розподіляти за категоріями та налаштувати сповіщення для важливих завдань.

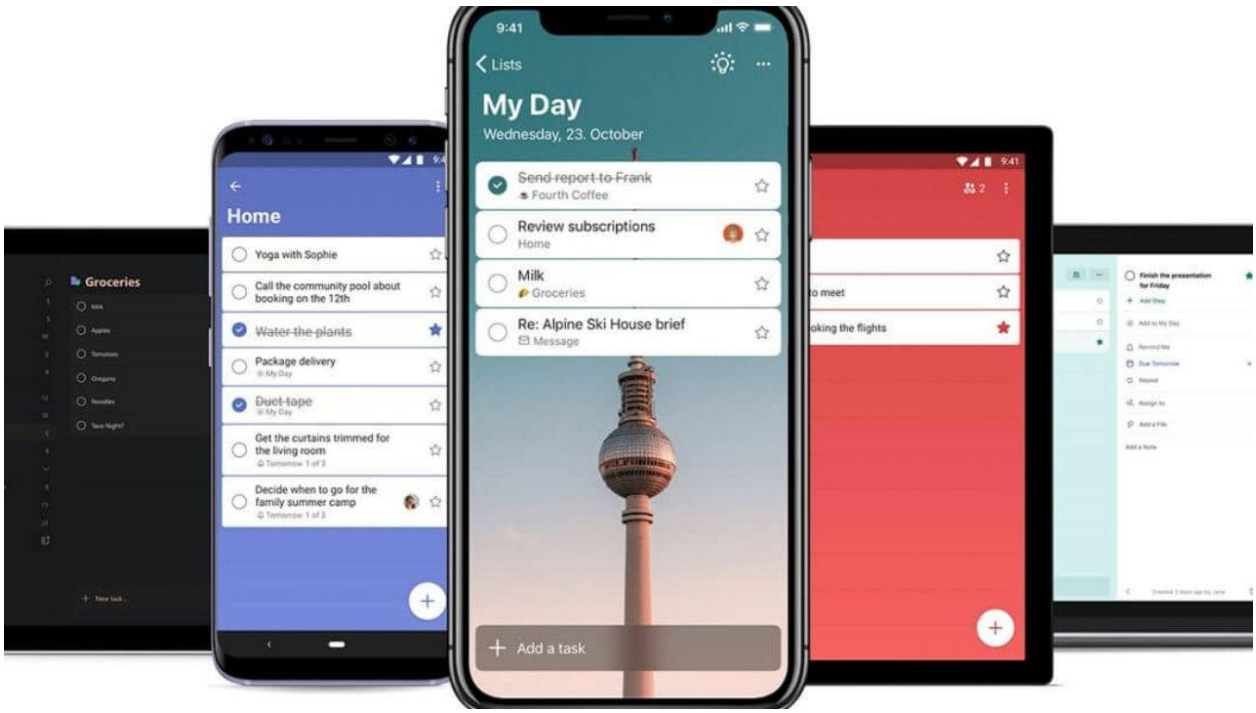


Рисунок 1.2 – Інтерфейс застосунку Microsoft To Do

Microsoft To Do також підтримує спільну роботу – застосунок дозволяє запрошувати користувачів до своїх списків завдань.

Переваги застосунку Microsoft To Do:

- простота використання. Інтерфейс простий та інтуїтивно зрозумілий;
- інтеграція з екосистемою Microsoft. Це дозволяє зручно інтегрувати свої завдання з іншими сервісами Microsoft;
- спільна робота. Microsoft To Do дозволяє ділитися завданнями з іншими користувачами, а також призначати виконавця;
- гнучкість налаштування. Є можливість встановлювати терміни виконання завдань, мітки та пріоритети;

Недоліки:

- завдання можна призначати не більше ніж одному користувачу одночасно. Якщо потрібно розподілити виконання завдання між кількома учасниками – такого функціоналу не надається;
- обмеженість статусів завдань. Не має можливості відстежувати проміжні стани або прогрес виконання завдання;
- відсутня можливість управління правами учасників. Відсутність цієї функціональності може ускладнити керування та контроль над проєктом;

1.3 Застосунок Tick-Tick

Tick-Tick – крос-платформений застосунок-планувальник, що дозволяє користувачам створювати, організовувати та відстежувати завдання та розклади. Він дозволяє групувати завдання в різні списки або категорії.

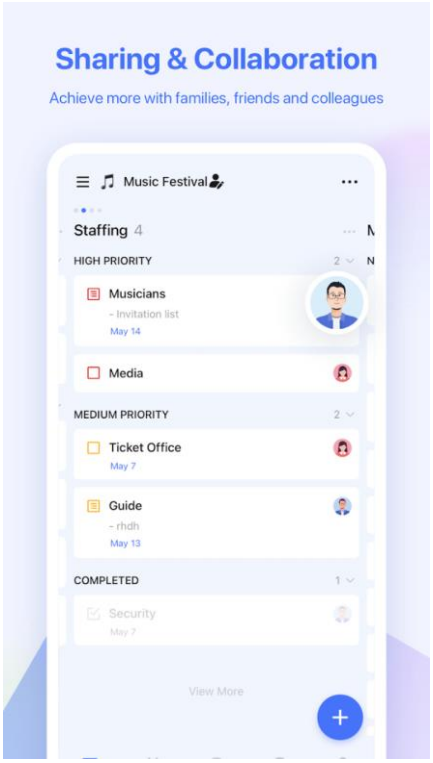


Рисунок 1.3 – Інтерфейс застосунку Tick-Tick

Застосунок дозволяє спільно працювати над завданнями, ділитися списками та спільно виконувати проєкти.

Переваги застосунку Tick-Tick:

- спільна робота. Tick-Tick дозволяє ділитися завданнями з іншими користувачами, а також призначати виконавця;
- персоналізація. Є можливість широкого налаштування вигляду та оформлення застосунку;
- гнучкість налаштування. Є можливість створення підзавдань, використовувати теги, мітки та використовувати пріорітети;
- офлайн-доступ. В умовах відсутності підключення до мережі, Tick-Tick дозволяє користувачам працювати зі своїми завданнями;
- управління правами учасників. Власник списку має можливість змінювати права користувачів, що доєдналися.

Недоліки:

- завдання можна призначати не більше ніж одному користувачу одночасно. Якщо потрібно розподілити виконання завдання між кількома учасниками – такого функціоналу не надається;
- обмеженість статусів завдань. Не має можливості відстежувати проміжні стани або прогрес виконання завдання;
- відсутність україномовного інтерфейсу.

1.4 Переваги розробленого рішення

Застосунок, що було розроблено в результаті цієї кваліфікаційної роботи має ряд переваг, а саме:

- кількість користувачів, яким можна призначати завдання необмежена;
- управління правами учасників. Власник має можливість призначати та звільняти адміністраторів з дошки;
- можливість відстеження прогресу виконання кожного завдання у відсотках;
- наявність чату для користувачів. Користувач може спілкуватися з іншими користувачами та адміністраторами за допомогою вбудованого чату, обмінюючись повідомленнями та інформацією;
- інтуїтивний інтерфейс. Застосунок пропонує зрозумілий та простий у використанні інтерфейс;
- україномовний інтерфейс. Це забезпечить зручність та зрозумілість використання застосунку для україномовних користувачів;
- використання сучасних технологій, таких як Kotlin, Android Studio, Gradle, Firebase Realtime Database, FCM та ін. Це забезпечує гнучкість високу продуктивність застосунку.

Ці переваги роблять розроблений застосунок гарним рішенням для організації спільної роботи в команді.

РОЗДІЛ 2. ОГЛЯД ІНСТРУМЕНТІВ РОЗРОБКИ

2.1 Платформа Android

Android – операційна система, що розроблена компанією Google. Вона базується на ядрі Linux, що забезпечує високу настроюваність та розширений функціонал для користувачів.

Метою платформи Android є надання розробникам інструментів та середовища для створення інноваційних мобільних застосунків, які працюють на різних пристроях, таких як смартфони, планшети, телевізори, годинники тощо.

Для розробників платформа Android надає такі можливості як:

- гнучкість у створенні інтерактивних користувацьких інтерфейсів. За допомогою різноманітних компонетнів, розробники можуть створювати списки, меню, вклади, елементи керування та інші елементи інтерфейсу;
- використання апаратних компонент пристрою, таких як камера, GPS, сенсори, мікрофон тощо;
- робота з мережевими протоколами (HTTP, TCP/IP та ін.)
- різні способи для зберігання та керування даними в застосунках. Розробники можуть використовувати локальну базу даних, файлову систему пристрою та хмарні сервіси

- інтеграція зі сторонніми сервісами та API, такими як соціальні мережі, платіжні системи, картографічні сервіси та іншими.

Надання такого широкого спектру можливостей сприяє створенню потужних та інноваційних мобільних застосунків.

2.2 Середовище розробки Android Studio

Android Studio – інтегроване середовище розробки (IDE), створене компанією Google для розробки програмного забезпечення під платформу Android [1]. Базується на популярній платформі IntelliJ IDEA.

Android Studio надає широкі можливості для створення, редагування та відлагодження Android-застосунків. Функціональні можливості включають:

- редактор коду з підсвічуванням синтаксису, автодоповненням, перевіркою помилок та іншими корисними функціями;
- вбудований дизайнер інтерфейсу, що дозволяє візуально створювати та редагувати користувацький інтерфейс застосунків.
- інструменти для відлагодження Android-застосунків;
- підтримка системи збірки Gradle, що дозволяє керувати залежностями, додавати сторонні бібліотеки та інші ресурси до проєкту;
- вбудовані емулятори Android, які дозволяють запускати та тестувати застосунки на віртуальних Android-пристроях.

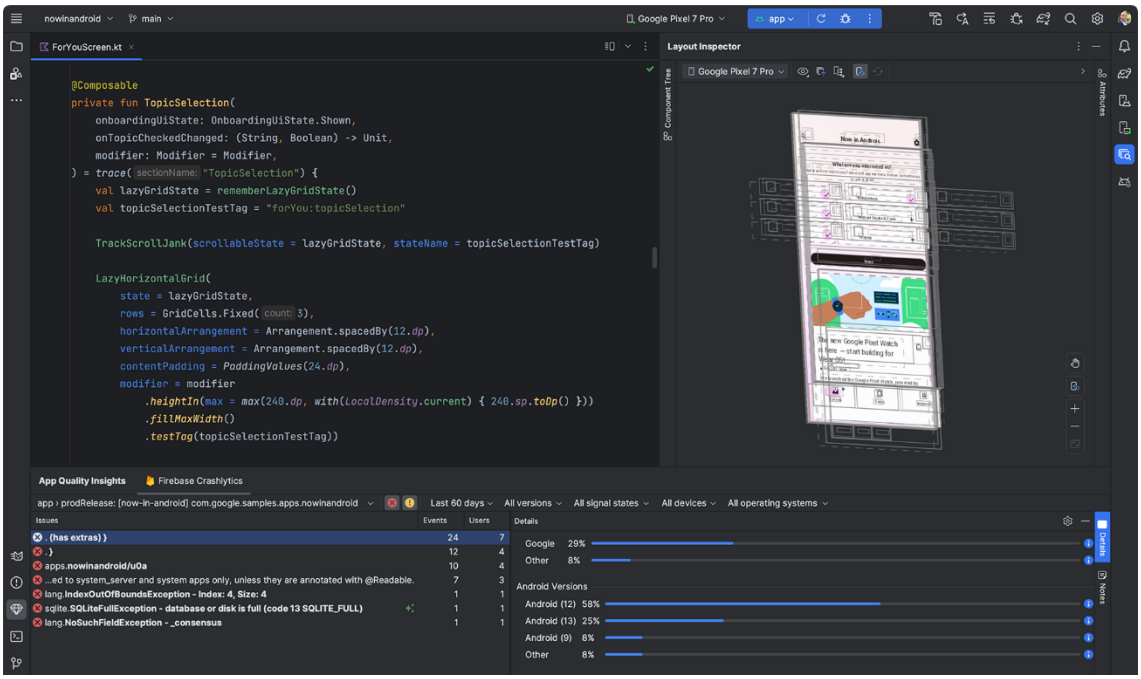


Рисунок 2.1 – Інтерфейс Android Studio

2.3 Мова програмування Kotlin

Kotlin – мова програмування, що працює на базі Java Virtual Machine (JVM). Вона повністю сумісна з мовою Java та має на меті поєднати в собі переваги інших мов. У 2017 році стала офіційною мовою розробки для платформи Android.

З головних переваг можна виділити:

- надання потужних функціональних можливостей, такі як лямбда-вирази, функціональні типи, операції зі звуженням та інші, що сприяють більш компактному коду;
- безпека. Kotlin має строгу типізацію та перевірки на етапі компіляції, що допомагає уникнути багатьох типових помилок;
- можливість розширення функціональності існуючих класів, за допомогою функцій-розширень;
- простий та логічний синтаксис, що дозволяє писати код більш продуктивно.

2.4 Платформа Firebase

Для збереження інформації було обрано Firebase - платформу хмарних послуг від Google. Firebase надає широкі можливості для розробки мобільних застосунків, включаючи аутентифікацію користувачів, зберігання та синхронізацію даних в режимі реального часу, хмарні повідомлення та багато іншого. Використання Firebase дозволить швидко та надійно забезпечити функціональність збереження та синхронізації даних для застосунку.

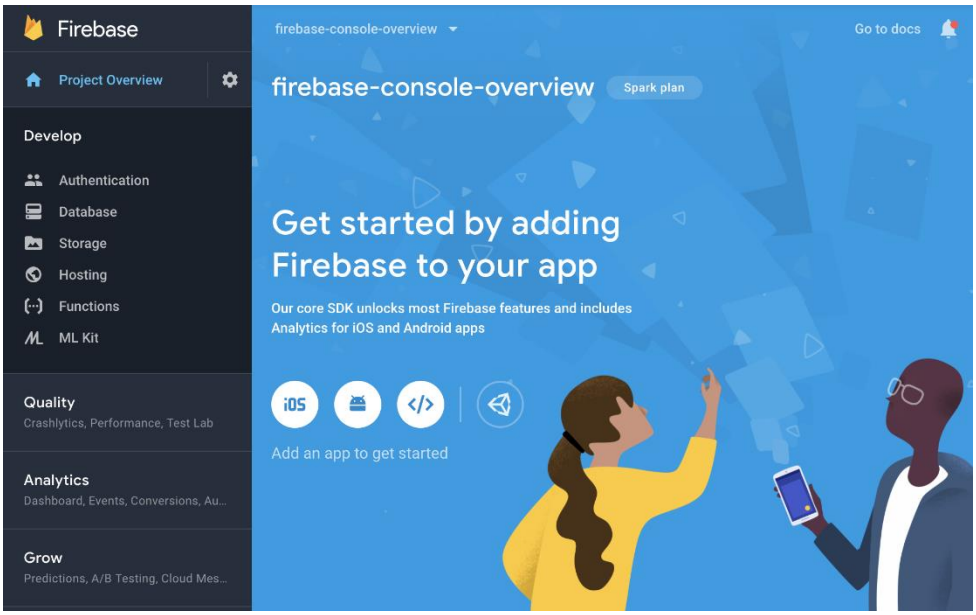


Рисунок 2.2 – Головна сторінка Firebase

Функції платформи Firebase:

- Realtime Database (база даних реального часу). Забезпечує швидку та миттєву синхронізацію даних між підключенимми клієнтами. Зміни в базі даних негайно розсилаються до всіх підписаних пристроїв. Дані зберігаються у вигляді JSON дерева з ключами та значеннями.

- Firestore (документ-орієнтовна база даних). База даних, що дозволяє зберігати структуровані дані у вигляді колекцій, документів та підколекцій. Також Firestore дозволяє виконувати багатофільтровані запити до даних.
- Authentication (аутентифікація). Сервіс аутентифікації та керування користувачами в Firebase. Він підтримує різні способи аутентифікації, такі як електронна пошта та пароль, облікові записи соціальних мереж, номер телефону тощо.
- Cloud Messaging (хмарні повідомлення). Сервіс, що в реальному часі надає можливість надсилати повідомлення на зареєстровані пристрої. FCM підтримує різні типи повідомлень, такі як сповіщення, дані та контрольовані повідомлення.

РОЗДІЛ 3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Загальні вимоги до застосунку

Перед початком розробки мобільного застосунку для організації спільної роботи, необхідно чітко визначити вимоги до застосунку. Вимоги є основою для подальшої розробки, тестування та оцінки успішності застосунку. Вони визначають функціональні та нефункціональні характеристики, що повинні бути реалізовані.

Застосунок повинен бути надійним і стабільним у роботі. Це включає відсутність помилок та відповідну обробку винятків, що можуть виникнути під час роботи застосунку. Повинен мати зручний та інтуїтивно зрозумілий інтерфейс, який забезпечує зручну навігацію та взаємодію користувача з застосунком. А також повинен забезпечувати захист персональних даних та бути сумісним з різними версіями операційної системи Android. та пристроями різного типу.

Функціональні вимоги до застосунку:

1. Реєстрація та авторизація користувачів. Застосунок повинен мати механізми для реєстрації нових користувачів та авторизації існуючих. Це забезпечить безпеку та ідентифікацію користувачів, а також можливість налагодження різних рівнів доступу до функцій застосунку.
2. Створення та керування дошками. Користувачам має бути надана можливість створювати нові дошки, а також додавати та видаляти учасників.
3. Створення та призначення завдань. Користувачі повинні мати можливість створювати нові завдання, встановлювати терміни виконання, визначати пріоритети та призначати завдання різним членам команди.
4. Додання і видалення учасників. Застосунок повинен надавати можливість адміністраторам додавати нових учасників до дошки проєкту та видаляти їх.
5. Призначення та звільнення адміністраторів. Повинна бути можливість призначати користувачів в ролі адміністраторів та звільняти їх від цієї ролі.
6. Україномовний інтерфейс. Застосунок повинен мати інтерфейс, який повністю локалізований на українську мову.
7. Чат для користувачів. Застосунок повинен мати функціонал, який дозволяє користувачам спілкуватися в режимі реального часу.
8. Сповіщення. Застосунок повинен мати можливість надсилання сповіщень про оновлення та зміни.

Визначення цих вимог є важливим етапом у розробці застосунку, оскільки вони встановлюють основу для подальшої.ііф

Логіку виконання програми зобразимо за допомогою діаграми прецедентів:

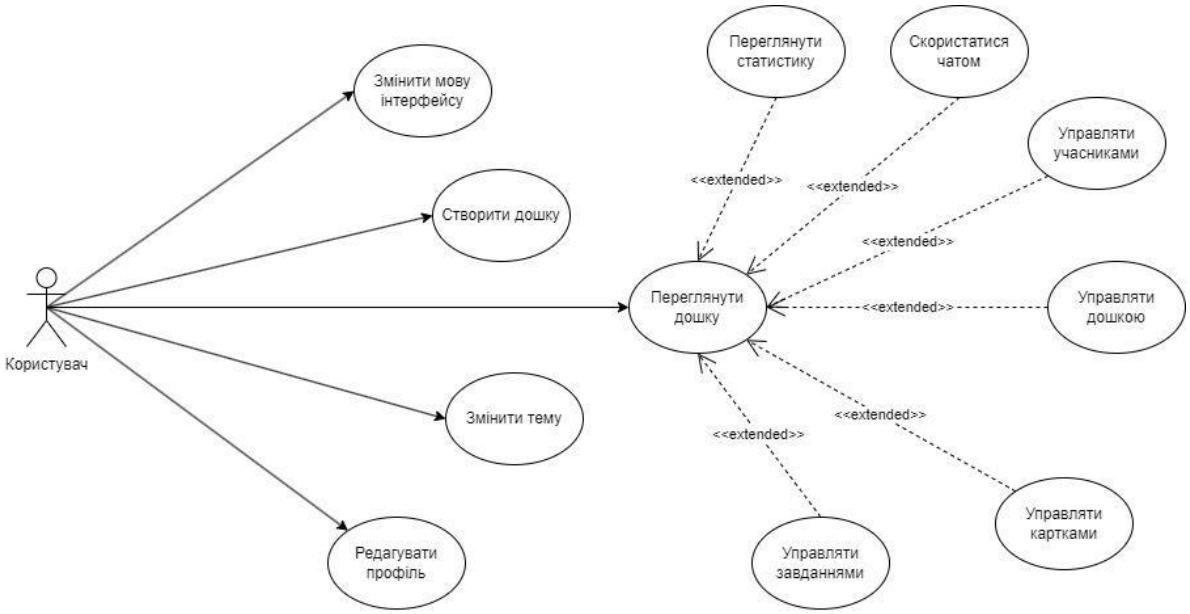


Рисунок 3.1 - Діаграма прецедентів для користувача

3.2 База даних Firebase

Firebase Firestore - це гнучка, швидка та легко масштабована база даних, яка пропонує багато можливостей для зберігання та структурування інформації. База даних включає в себе дві колекції: «boards» та «users». Тут зберігається інформація про дошки та користувачів [10]. У сховищі Firebase Storage – розташовуються всі зображення (користувачів та дошок) [4], [11].

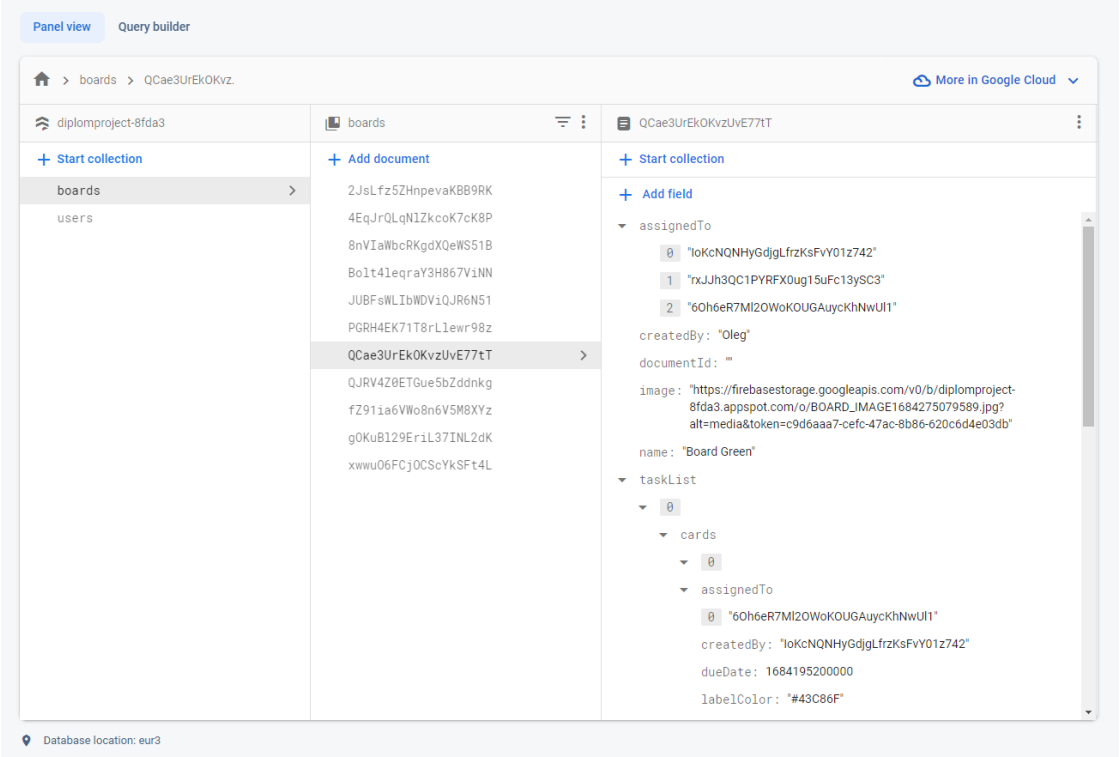


Рисунок 3.2 - Дані про існуючі дошки

У колекції «boards» знаходиться інформація про існуючі дошки. Кожен документ колекції представляє окрему дошку. Дошки мають свої властивості, такі як назва, список учасників, список карток, завдань та інші. Застосунок може додавати нові дошки, видаляти та редагувати існуючі.

У колекції «users» знаходиться інформація про всіх користувачів [9]. Кожен документ колекції представляє окремого користувача. Кожен користувач має свої дані, такі як ім’я, електронна адреса, пароль та інші.

Firestore дозволяє легко створювати та отримувати дані про користувачів, забезпечуючи автоматичну синхронізацію даних між пристроями.

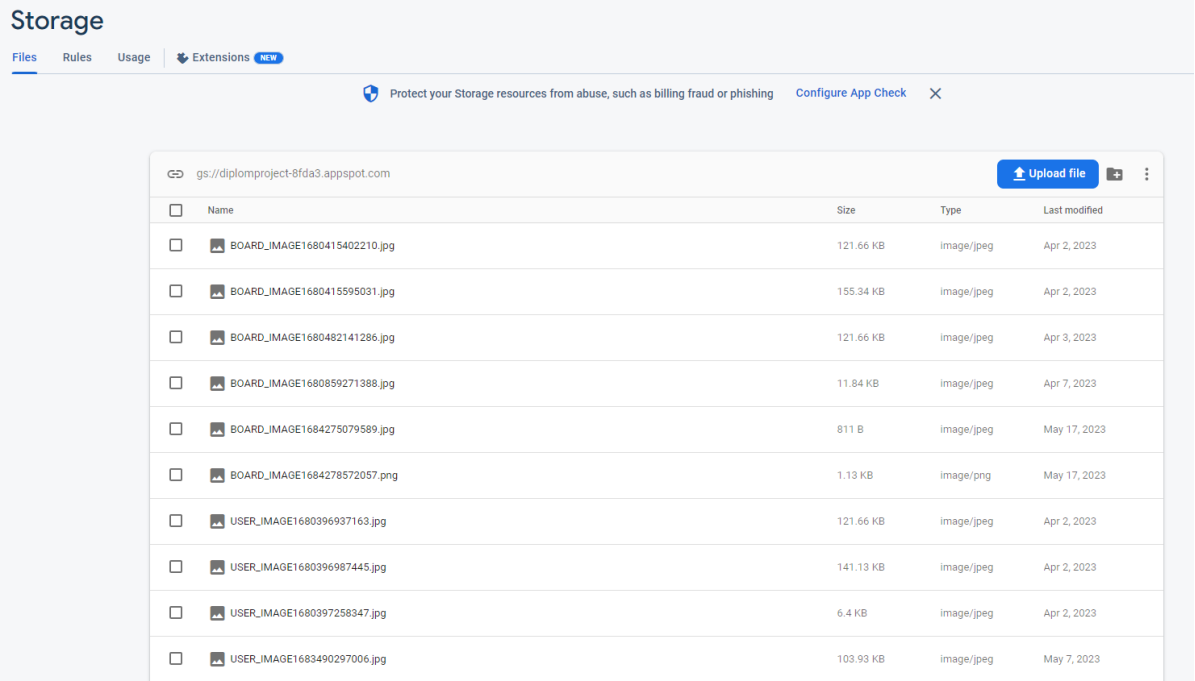


Рисунок 3.3 – Сховище Firebase Storage

Для зберігання зображень дошок та зображень профілю користувачів використовується Firebase Storage. Він дозволяє безпечно та надійно завантажувати, зберігати та керувати файлами.

При створенні нової дошки, користувач може обрати зображення із галереії свого пристрою, яке буде завантажене до сховища. В подальшому це зображення буде прив’язано до відповідної дошки для її візуальної ідентифікації.

Також, кожен користувач має можливість завантажити власне зображення профілю, що зберігатиметься у Firebase Storage. Воно може використовуватися для ідентифікації користувача та покращення особистого досвіду використання застосунку.

3.3 Реєстрація користувачів

Функція реєстрації – важлива частина розробки застосунку, адже саме вона дозволяє новим користувачам отримати доступ до всіх функцій та

можливостей. Реєстрація забезпечує ідентифікацію користувача, створення нового облікового запису та збереження його в системі.

```
private fun SignUp(){
    val etName = findViewById<EditText>(R.id.et_name)
    val etEmail = findViewById<EditText>(R.id.et_email)
    val etPassword = findViewById<EditText>(R.id.et_password)

    val name: String = etName.text.toString().trim(){it <= ' '}
    val email: String = etEmail.text.toString().trim(){it <= ' '}
    val password: String = etPassword.text.toString().trim(){it <= ' '}

    if (validateForm(name, email, password)) {
        auth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    val user_auth = auth.currentUser
                    val firebaseUser: FirebaseUser = task.result!!.user!!
                    val registeredEmail = firebaseUser.email!!
                    val user = User(firebaseUser.uid, name, registeredEmail)
                    FirestoreClass().registerUser( activity: this, user)
                    startActivity(Intent( packageContext: this,IntroActivity::class.java))
                } else {
                    Toast.makeText(baseContext,  text: "Authentication failed.",
                        Toast.LENGTH_SHORT).show()
                }
            }
    }
}
```

Рисунок 3.4 - Функція реєстрації облікового запису

Функція SignUp() використовується для реєстрації нового користувача в застосунку. Зчитуючи з полів EditText таку інформацію як ім’я, електронну адресу та пароль, функція здійснює наступні кроки:

- перевіряє чи не використовувалася для реєстрації електронна адреса раніше;
- створює новий обліковий запис в Firebase Authentication;
- інформацію про обліковий запис також записує в Firebase Firestore, щоб в подальшому її можна було використовувати та змінювати;
- якщо під час реєстарції сталася помилка, функція SignUp() її обробляє , надаючи відповідне повідомлення про помилку «Authentication failed»

```
data class User (val id: String = "",
    val name: String = "",
    val email: String = "",
    val image: String = "",
    val mobile: Long = 0,
    val fcmToken: String = "",
    var selected: Boolean = false
) : Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readString()!!,
        parcel.readString()!!,
        parcel.readString()!!,
        parcel.readString()!!,
        parcel.readLong(),
        parcel.readString()!!
    ) {
    }
```

Рисунок 3.5 – Модель користувача

Щоб зберегти дані нового користувача, небхідно створити його модель за допомогою класа User(). Ця модель в подальшому буде збережена в базі даних Firebase Firestore в колекції «users». На рис. 8 зображена панель із зарєстрованими користувачами в Firebase Authentication.

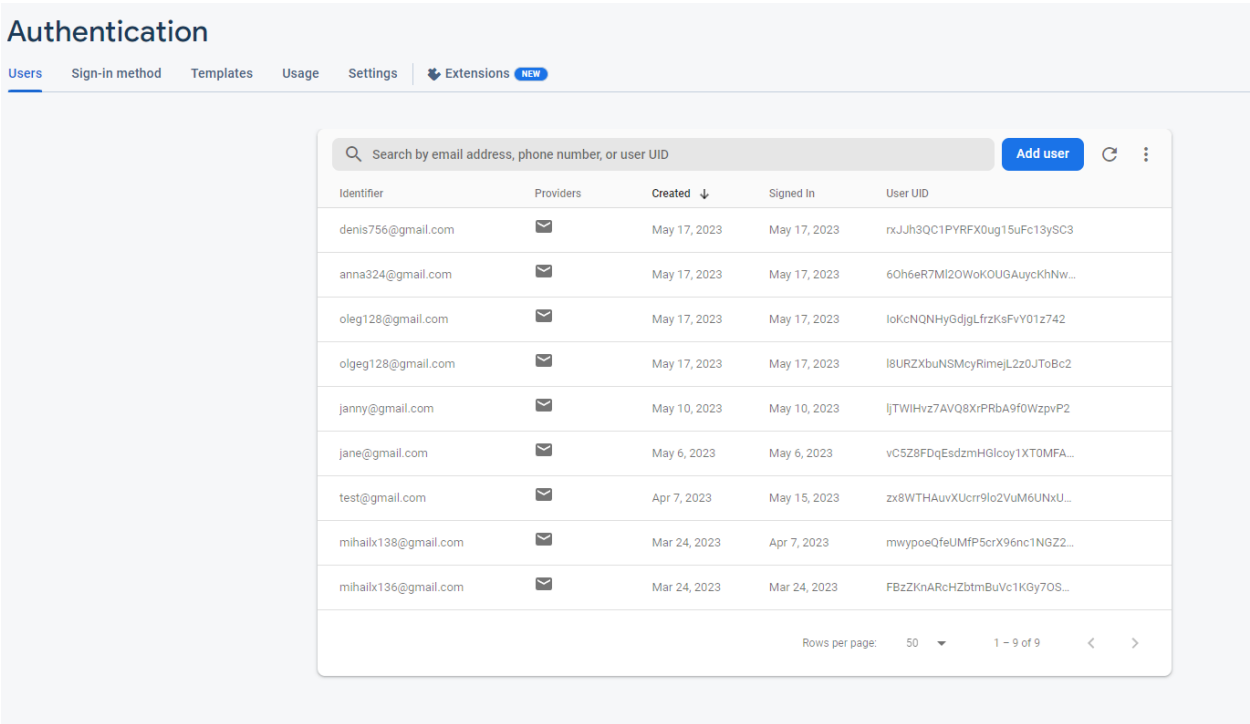


Рисунок 3.6 – Панель із зареєстрованими користувачами

3.4 Функція створення дошки

Функція створення дошки – одна з ключових функцій застосунку, адже саме завдяки дошці користувачі зможуть створювати та управляти картками,

```
private fun createBoard(){
    val assignedUsersArrayList: ArrayList<String> = ArrayList()
    assignedUsersArrayList.add(getCurrentUserID())

    var board = Board(
        et_board_name.text.toString(),
        mBoardImageURL,
        mUserName,
        assignedUsersArrayList
    )
    FirestoreClass().createBoard( activity: this,board)
}
```

Рисунок 3.7 – Функція створення дошки

Функція createBoard() створює модель класу Board, передаючи як аргументи: назву, посилання на зображення (що попередньо було завантажене у сховище Firestore), ім’я користувача, та список користувачів, що були додані на дошку.

Функція uploadBoardImage() відповідає за збереження зображення дошки.

```
private fun uploadBoardImage(){
    val storageReference: StorageReference = FirebaseStorage.getInstance().reference.child(
        pathString: "BOARD_IMAGE" + System.currentTimeMillis() + "."
                + Constants.getFileExtension( activity: this, mSelectedImageFileUri))

    storageReference.putFile(mSelectedImageFileUri!!) UploadTask
        .addOnSuccessListener { taskSnapshot ->
            Log.e(
                tag: "Firebase Board Image URL",
                taskSnapshot.metadata!!.reference!!.downloadUrl.toString()
            )
            taskSnapshot.metadata!!.reference!!.downloadUrl
                .addOnSuccessListener { uri ->
                    Log.e( tag: "Downloadable Image URL", uri.toString())

                    mBoardImageURL = uri.toString()
                    createBoard()
                }
        } StorageTask<UploadTask.TaskSnapshot!>
        .addOnFailureListener { exception ->
            Toast.makeText(
                context: this,
                exception.message,
                Toast.LENGTH_LONG
            ).show()
        }
}
```

Рисунок 3.8 - Функція збереження зображення дошки

Функція uploadBoardImage() зчитує зображення, яке завантажив користувач та відправляє цей файл до сховища Firebase Storage. Як ім’я файлу, генерується унікальний ідентифікатор, щоб уникнути конлфікту імен фійлів при збереженні. Якщо результат успішний – функція отримає посилання на файл зі сховища та викликає метод створення дошки createBoard().

3.5 Додавання учасника

Функція getMemberDetals() необхідна у застосунку для того, щоб знайти потрібного користувача у базі даних, в колекції «users». У випадку, якщо його знайдено, дані користувача зберігаються та передаються як аргумент у функцію memberDetails()

```
fun getMemberDetails(activity: MembersActivity, email: String){
    mFirestore.collection(Constants.USERS).whereEqualTo(Constants.EMAIL, email).get().addOnSuccessListener {
        document ->
        if(document.documents.size > 0){
            val user = document.documents[0].toObject(User::class.java)!!
            activity.memberDetails(user)
        } else{
            activity.showErrorSnackBar("No such member found")
        }
    }
}
```

Рисунок 3.9 - Пошук користувача в базі даних

Функція memberDetails() зчитує передану інформацію про нового користувача та додає його id в список доданих до дошки користувачів. Викликаючи метод assignMemberToBoard(), оновлений список використовується як один із аргументів виклику.

```
fun memberDetails(user: User){
    mBoardDetails.assignedTo.add(user.id)
    FirestoreClass().assignMemberToBoard( activity: this, mBoardDetails, user)
}
```

Рисунок 3.10 - Додавання нового учасника до дошки

```
fun assignMemberToBoard(activity: MembersActivity, board: Board, user: User){
    val assignedToHashMap = HashMap<String, Any>()
    assignedToHashMap[Constants.ASSIGNET_TO] = board.assignedTo

    mFirestore.collection(Constants.BOARDS).document(board.documentId).update(assignedToHashMap).addOnSuccessListener {  it: Void!
            activity.memberAssignSuccess(user)
        }
}
```

Рисунок 3.11 – Внесення змін до бази даних

Функція assignMemberToBoard() отримує дані про користувача та дошку з оновленим списком доданих користувачів. Інформація зі списку зберігається у змінній класу HashMap<String, Any>, яка в свою чергу використовується як аргумент для оновлення інформації про дошку у базі даних, в колекції boards. Якщо результат успішний – інформація про учасників дошки оновлюється.

3.6 Створення завдання

```
fun addTaskToCardList(position: Int, taskName: String){
    mBoardDetails.cardList.removeAt(mBoardDetails.cardList.size - 1)
    val taskAssignedUserList: ArrayList<String> = ArrayList()

    val task = Task(taskName, FirestoreClass().getCurrentUserId(), taskAssignedUserList)

    val tasksList = mBoardDetails.cardList[position].tasks
    tasksList.add(task)

    val card = Card(mBoardDetails.cardList[position].title,
        mBoardDetails.cardList[position].createdBy,mBoardDetails.cardList[position].adminAccess, tasksList)

    mBoardDetails.cardList[position] = card
    FirestoreClass().addUpdateCardList(this, mBoardDetails)
}
```

Рисунок 3.12 - Функція збереження зображення дошки

Функція addTaskToCardList() зчитує назву завдання від користувача, на основі якої, створює новий об’єкт класу Task. З бази даних дістається список існуючих завдань і додається щойно створене. В кінці створюється об’єкт класу Card (в який передається новий список завдань в якості аргумента) та зберігається у дошці.

```
data class Task(
    val name: String = "",
    val createdBy: String = "",
    val assignedTo: ArrayList<String> = ArrayList(),
    val labelColor: String = "",
    val dueDate: Long = 0,
    val progress: Int = 0,
    val completed: Boolean = false
) : Parcelable {
    @RequiresApi(Build.VERSION_CODES.Q)
    constructor(source: Parcel) : this(
        source.readString()!!,
        source.readString()!!,
        source.createStringArrayList()!!,
        source.readString()!!,
        source.readLong()!!,
        source.readInt()!!,
        source.readBoolean()!!
    )
```

Рисунок 3.13 – Клас моделі Task

Клас Task виступає як модель, що представляє завдання в застосунку. Основними його параметрами є:

- назва;
- користувач, що створив завдання;
- список призначених учасників;
- пріоритет;
- дата завершення;
- прогрес.

3.7 Функція призначення адміністратора

```
fun getAdminDetails(activity: MembersActivity, email: String){
    mFirestore.collection(Constants.USERS).whereEqualTo(Constants.EMAIL, email).get().addOnSuccessListener {
            document ->
        if(document.documents.size > 0){
            val user = document.documents[0].toObject(User::class.java)!!
            activity.adminDetails(user)
        } else{
            activity.showErrorSnackBar("No such member found")
        }
    }
}
```

Рисунок 3.14 – Функція пошуку користувача в базі даних

Функція getAdminDetails() отримує на вхід електронну адресу та шукає існуючого користувача в колекції users бази даних Firebase Firestore. У випадку, якщо користувача знайдено – його дані зберігаються та передаються як аргумент при виклиці adminDetails(). Якщо ж електронна адреса не збігається ні з одним існуючим користувачем – з’явиться повідомлення про помилку «No such member found».

```
fun adminDetails(user: User){
    mBoardDetails.admins.add(user.id)
    FirestoreClass().addAdminToBoard( activity: this, mBoardDetails, user)
}
```

Рисунок 3.16 – Функція додавання нового адміністратора до дошки

Функція adminDetails() зчитує дані про нового адміністратора та додає його id до списку адміністраторів дошки. Цей оновлений список використовується як аргумент для виклику функції addAdminToBoard().

```
fun addAdminToBoard(activity: MembersActivity, board: Board, user: User){
    val adminsHashMap = HashMap<String, Any>()
    adminsHashMap[Constants.ADMINS] = board.admins

    mFirestore.collection(Constants.BOARDS).document(board.documentId).update(adminsHashMap).addOnSuccessListener {
        activity.AdminAssignSuccess(user)
    }
}
```

Рисунок 3.15 – Функція внесення змін до бази даних

Функція addAdminToBoard() отримує дані про нового адміністратора та дошку з оновленим списком. Ця інформація зберігається у змінній класу HashMap<String, Any> після чого, використовується як аргумент при оновленні інформації про дошку в базі даних Firebase Firestore. Якщо результат успішний – вікно з інформацією про учасників дошки оновлюється.

РОЗДІЛ 4. ОГЛЯД ЗАСТОСУНКУ

4.1 Огляд дошок

У застосунку кожна дошка представляє окремий проєкт, над яким працює команда. Вона має свою назву та зображення і містить в собі картки із завданнями, два чати (для учасників та адміністраторів), а також загальну статистику. Важливими функціями дошки є можливість додавання та видалення учасників, а також призначення та звільнення адміністраторів. Щоб створити нову дошку, користувачу необхідно натиснути кнопку «Add».

Відкриється нове вікно з полем для введення назви та формою для завантаження зображення.

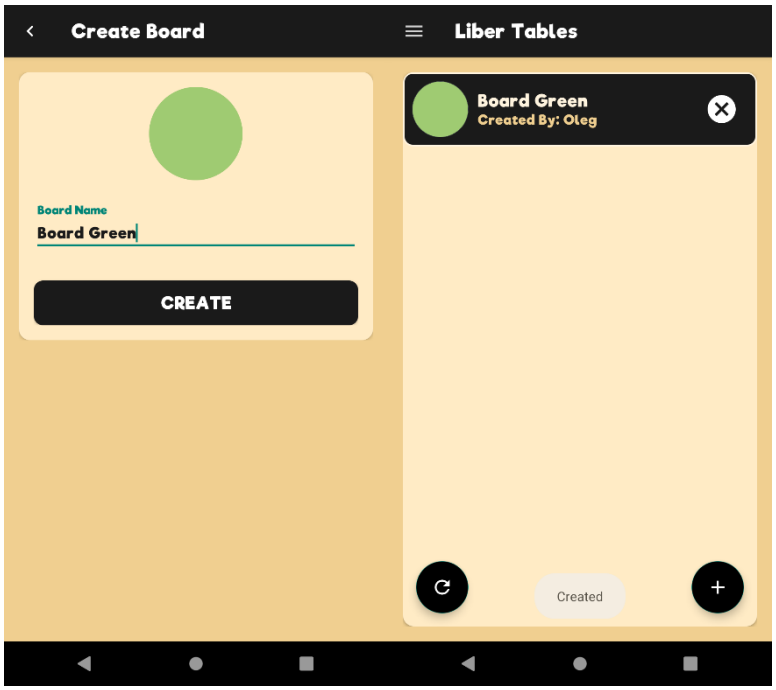


Рисунок 4.1 – Створення нової дошки

Для того, щоб видалити існуючу дошку, необхідно мати відповідні права. Ними володіє лише користувач, що є безпосереднім автором.

4.2 Огляд карток

Картка – основний елемент для організаціїї та управління завданнями в застосунку. Представляється у вигляді окремого блоку, розташованого на дошці. Містить основну інформацію про кожне завдання, що було додане на цю картку, а саме: назва, колір мітки (пріоритетність), статус виконання, термін виконання. Картки дозволяють управляти завданнями шляхом додавання, видалення, редагування та переміщення між собою.

При натисканні на нову дошку, відкриється вікно з основною інформацією. Тут можна створювати картки із завданнями, а також додавати/видаляти уачсників.

Для того щоб створити нову картку, необхідно натиснути кнопку «Add Card», після чого ввести назву.

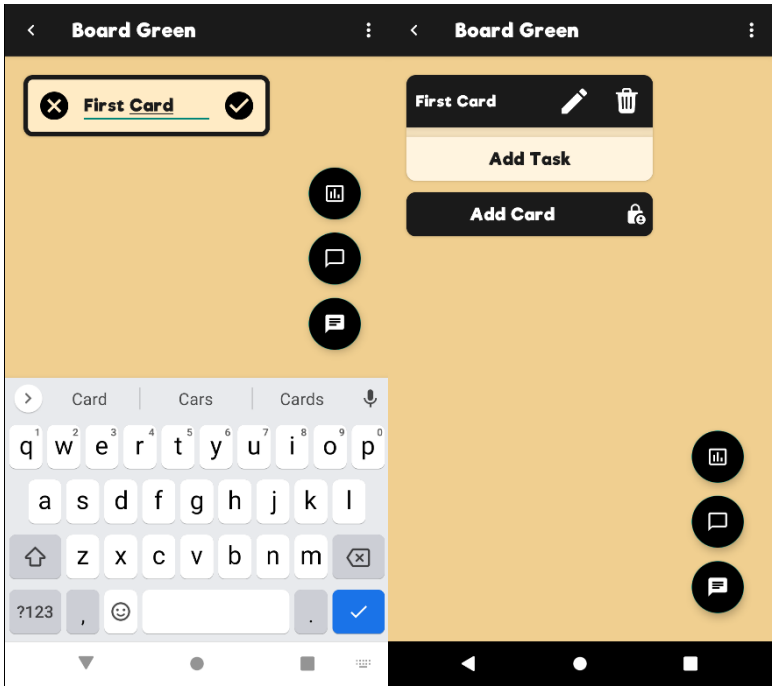


Рисунок 4.2 – Створення нової картки

За необхідності, в будь-який момент користувач має можливість змінити назву картки або ж і зовсім видалити її.

Приватна картка – картка, яку може бачити лише адміністратор та творець дошки. Це дає можливість створювати та працювати над конфіденційними або обмеженими проєктами, які не призначені для загального доступу. Щоб її створити необхідно натиснути на значок із зображенням замка, що розташований на кнопці «Add Card».

4.3 Огляд керування користувачами

Додавання, видалення учасників, а також призначення адміністраторів - важлива функція застосунку, що забезпечує гнучкість та керованість у роботі з дошками. Це дозволяє плідно та ефективно управляти командними проєктами та розподіляти задачі.

Щоб додати нових учасників до дошки, необхідно натиснути кнопку «Details» у правому верхньому куту. На цьому місці з’явиться кнопка «Members», при нажатті на яку, відкриєтсья вікно з інформацією про учасників дошки.

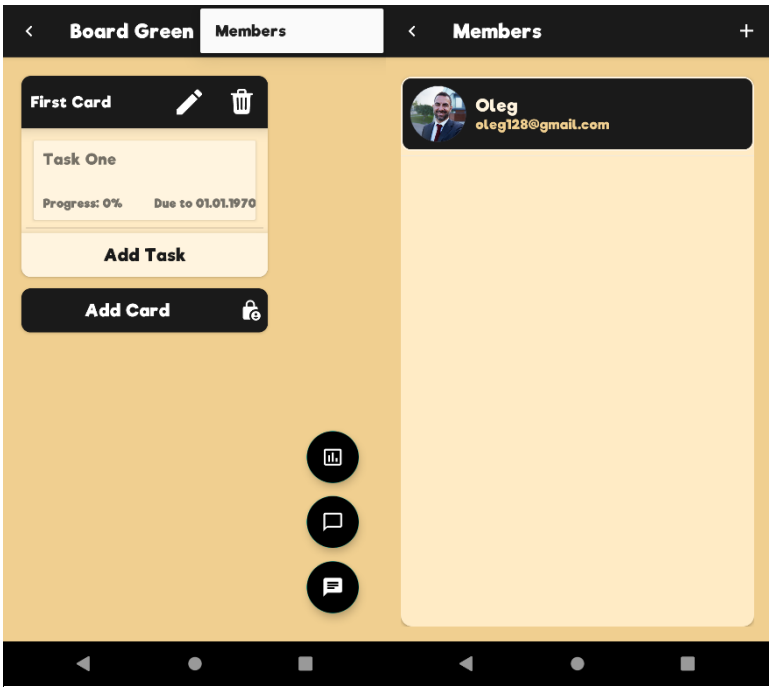


Рисунок 4.3 – Відкриття вікна з учасниками дошки

Для того щоб додати нового учасника, необхідно нажати кнопку «Add» в правому верхньому куту. В результаті відкриється вікно з полем для вводу електронної адреси. Користувачу потрібно ввести Email користувача, якого він бажає додати.

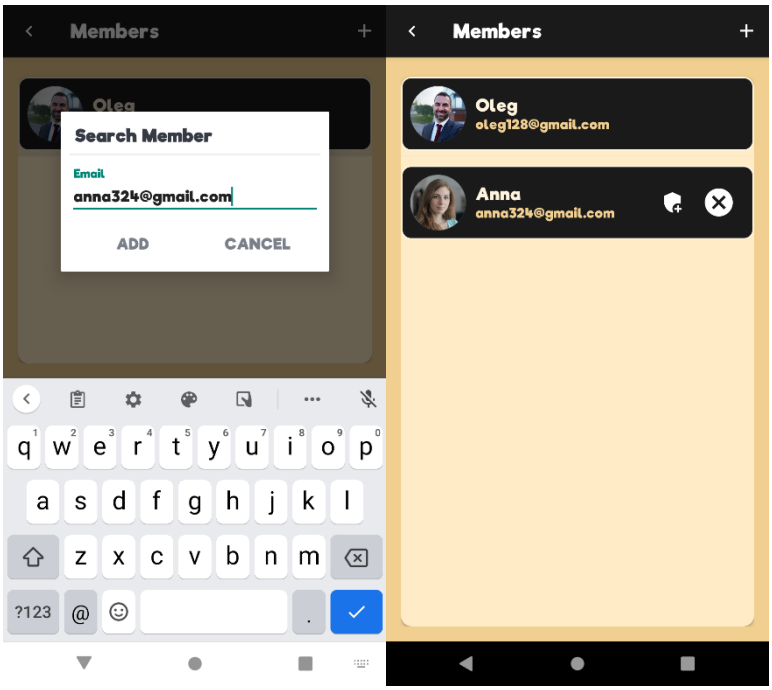


Рисунок 4.4 – Додавання нового учасника

Щоб призначити адміністратора – необхідно натиснути на зображення щитка з плюсом. У випадку, якщо необхідно навпаки звільнити адміністратора – необхідно нажати на зображення перекресленого щитка.

За необхідності, користувач, що створив дошку може в будь-який момент видалити непотрібного учасника.

4.4 Огляд завдань

Завдання – конкретна задача або обов'язкок, який можна додати на картку. Представлена у вигляді коремого блоку і містить в собі загальну інформацію щодо виконання, а саме:

- Назва (Короткий опис завдання);
- Колір мітки (пріоритетність);
- Призначені виконавці (учасники, що визначені як відповідальні за виконання поставленої задачі);
- Статус виконання (від 0% до 100%);

- Термін виконання.

Завдання відображають конкретні обов’язки або ж робочі елементи, які необхідно виконати в рамках проєкту. Надання зручного способу організації та відстеження прогресу виконання різних завдань у межах проєкту – головна їх функція.

Для створення нового завдання, необхідно натиснути «Add Task», після чого, ввести назву.

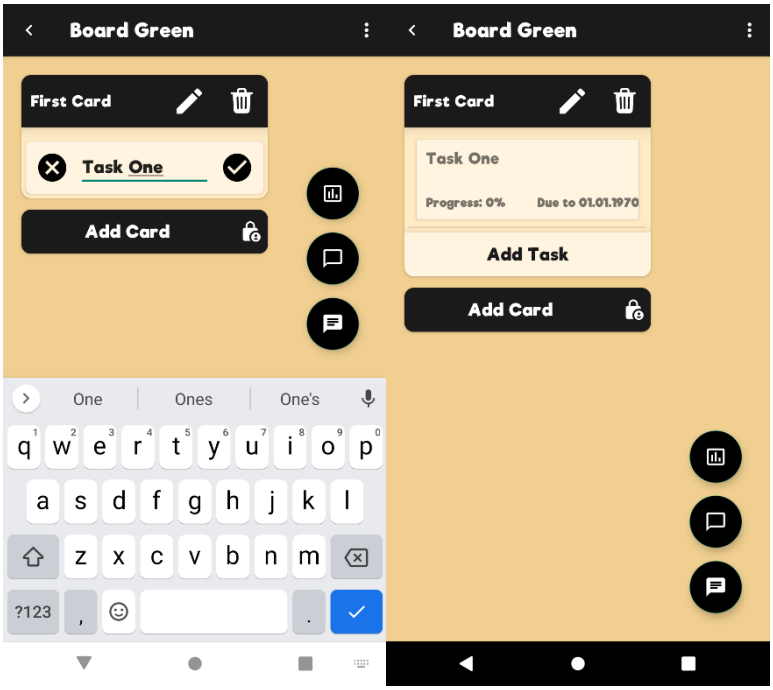


Рисунок 4.5 – Створення нового завдання

Щоб встановити термін виконання, колір мітки, а також відповідальних за завдання – необхідно обрати і натиснути на потрібний Task. Відкриється нове вікно із детальною інформацією про завдання, з можливістю її редагування.

Для втсановлення кольору мітки, необхідно обрати пункт «Select Color». Користувач може обрати один з доступних кольорів, а саме:

- Червоний (Критичний пріоритет) - використовується для позначення найважливіших завдань, що потребують негайної уваги;
- Помаранчевий (Високий пріоритет) - використовується для позначення важливих завдань, якщо терміни їх виконання не є критичними;
- Жовтий (Нормальний пріоритет) - використовується для звичайних завдань, які мають стандартний пріоритет. Вони не є терміновими, тому можуть бути виконані в межах планового часу;
- Зелений (Низький пріоритет) - використовується для завдань, що не є важливими, або їх виконання може бути відкладене на деякий час;
- Блакитний (Опціональний пріоритет) - використовується для неважливих завдань, які можуть бути виконані, якщо дозволяють час і ресурси;

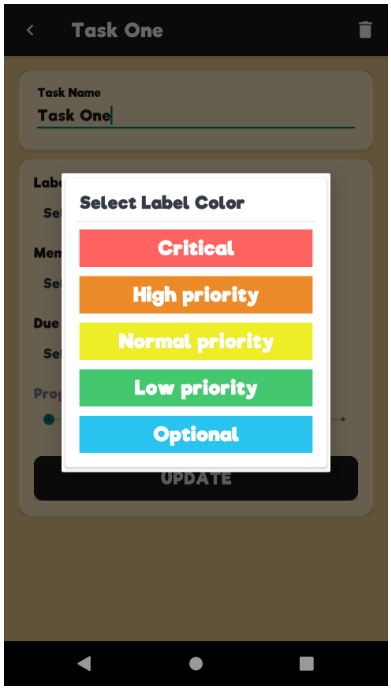


Рисунок 4.6 – Встановлення кольору мітки

Важливою функцією застосунку є встановлення відповідальних за завдання. Це дозволяє призначити конкретного користувача або групу користувачів на виконання певного завдання.

Щоб встановити відповідальних за завдання, потрібно обрати пункт «Select Members». У спливаючому вікні користувач може призначати та звільняти користувачів, що попередньо були додані в дошку.

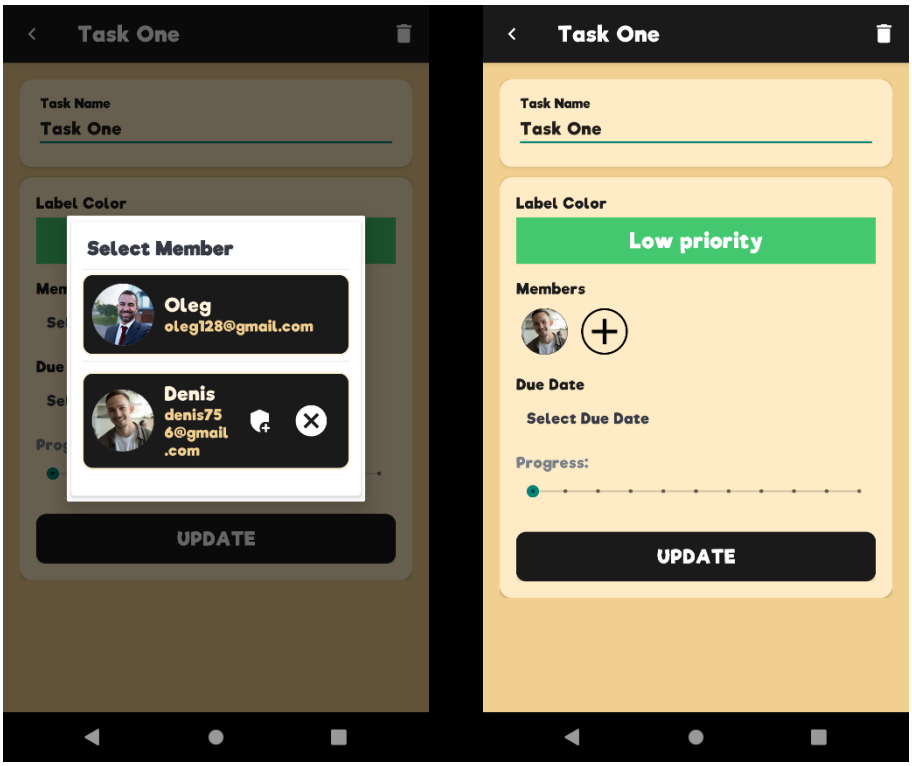


Рисунок 4.7 – Встановлення відповідального за завдання

Встановлення терміну завдання - важливий етап управління проєктом. Він дозволяє визначити часові рамки для виконання поставленого завдання. Це допомагає забезпечити вчасне виконання роботи, а також покращує планування та сприяє досягненню цілей проєкту.

Задля встановлення терміну виконання, необхідно обрати пункт «Due Date». На екрані з’явиться інтерактивний календар в якому можна обрати потрібне число.

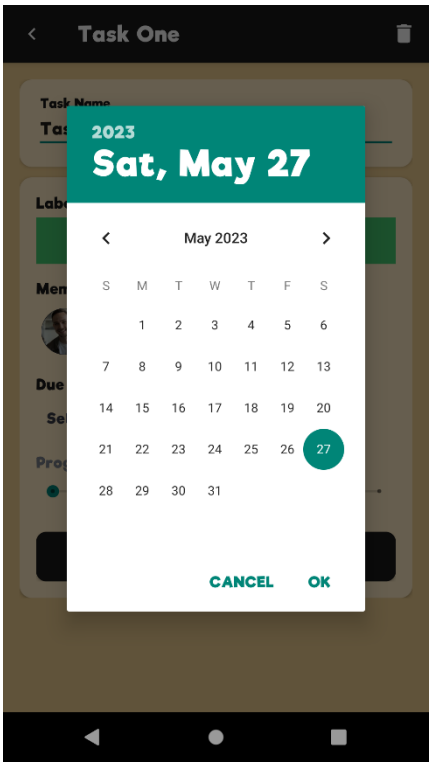


Рисунок 4.8 – Встановлення терміну виконання

Щоб змінити прогрес виконання – у пункті «Progress» (прогрес) необхідно встановити потрібне значення (від 0% до 100%). У випадку, якщо завдання виконане на 100% - його статус буде змінено на «Completed» (виконано).

В разі необхідності, користувач може в будь-який момент видалити непотрібне завдання. Для цього небхідно натиснути на іконку смітника у правому верхньому куту. Одразу ж система запитає підтвердження.

Кількість завдань для однієї картки – необмежена. Якщо користувачу знадобиться змінити порядок, в якому розташовані завдання – необхідно натиснути та утримувати потрібний Task, після чого перемістити у більш придатне місце.

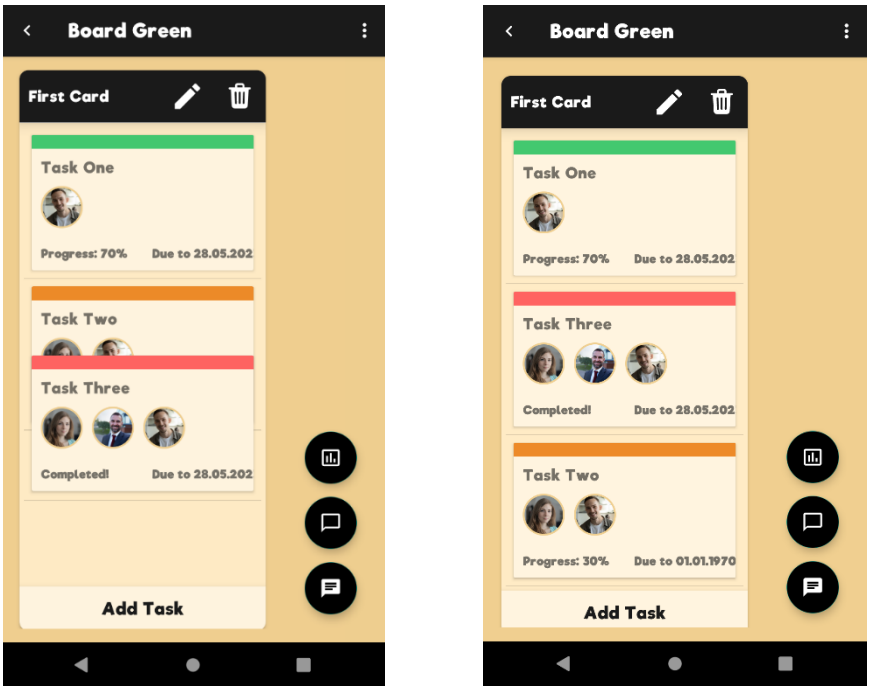


Рисунок 4.9 – Переміщення завдання

4.5 Огляд чатів

Комунікація – важлива складова будь-якого колективного проєкту. Саме тому застосунок надає можливості та інструменти для ефективної комунікації між користувачами: чат для користучачів та приватний чат [15], [12].

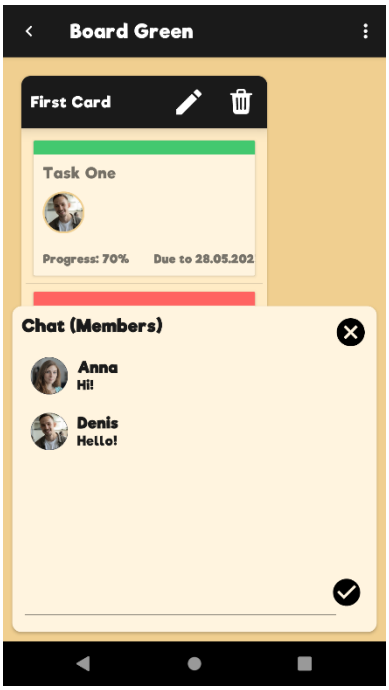
Чат для користувачів дозволяє учасникам команди спілкуватись, а також ділитися ідеями та проблемами, що виникають під час роботи. Це забезпечує зручну платформу для спілкування та співпраці між учасниками проєкту.

Приватний чат – створений з метою забезпечити приватну комунікацію між адміністраторами та творцем дошки. Це надає змогу обговорювати важливі питання та приймати рішення без участі інших користувачів. Адміністратори можуть обмінюватися конфіденційною інформацією, обговорювати стратегії та плани дій.

Обидва чати мають зручний мінімалістичний інтерфейс, що дозволяє без проблем обмінюватися повідомленнями.

Щоб відкрити чат користувачів – необхідно натиснути на кнопку із зображенням білого чату у правому нижньому кутку.

Щоб відкрити приватний чат – необхідно нажати на кнопку із зображення прозорого чату.



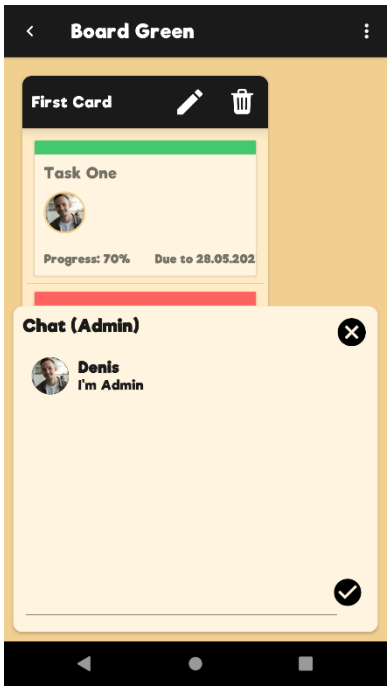


Рисунок 4.10 – Чат користувачів *Рисунок 4.11 – Приватний чат*

4.6 Вкладка Theme

Вкладка «Theme» в застосунку надає користувачу можливість налаштувати зовнішній вигляд та оформлення застосунка, а саме тему і шрифт [14].

Обрання теми – користувач може обрати одну з доступних тем, а саме: бежеву, червону або жовту. В залежності від вибору, буде змінено його кольорову палітру.

Зміна шрифту – окрім теми, користувач також може змінити й шрифт застосунка. Це впливає на типографіку тексту в застосунку, роблячи його більш читабельним або ж змінюючи його характер. З доступних варіантів: Classic (Sans-serif), Times New Roman, Disket Mono та Riffic.

Після вибору теми та шрифту, система застосовує зміни до всього інтерфейсу, змінюючи його кольорову схему та стиль тексту.

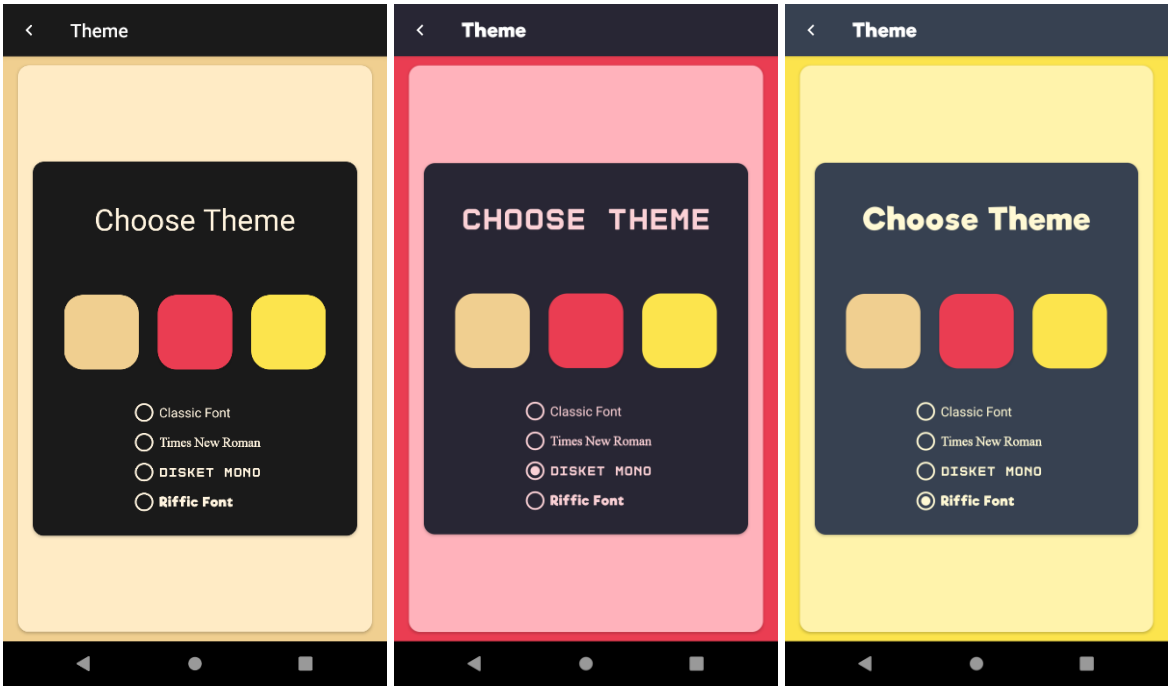


Рисунок 4.12 – Зміна теми і шрифту

4.7 Вкладка Language

Вкладка «Language» в застосунку надає користувачу можливість змінити мову інтерфейсу на одну з доступних [6]:

- Англійська;
- Українська
- Німецька;
- Французька.

Коли користувч обирає вкладку «Language» - він зустрічає перелік доступних для вибору мов. Кожна мова позначена прапорцем відповідної їй країни. Це забезпечує зручну навігацію і використання застосунку на бажаній мові.

Для обрання бажаної мови необхідно натиснути на відповідний пункт зі списку, після чого зберегти зміни за допомогою кнопки «UPDATE» (оновити). В результаті, застосунок переключається на обрану мову і відображає інтерфейс, повністю до неї адаптований. Зміна охоплює всі тексти, кнопки, а також повідомлення.

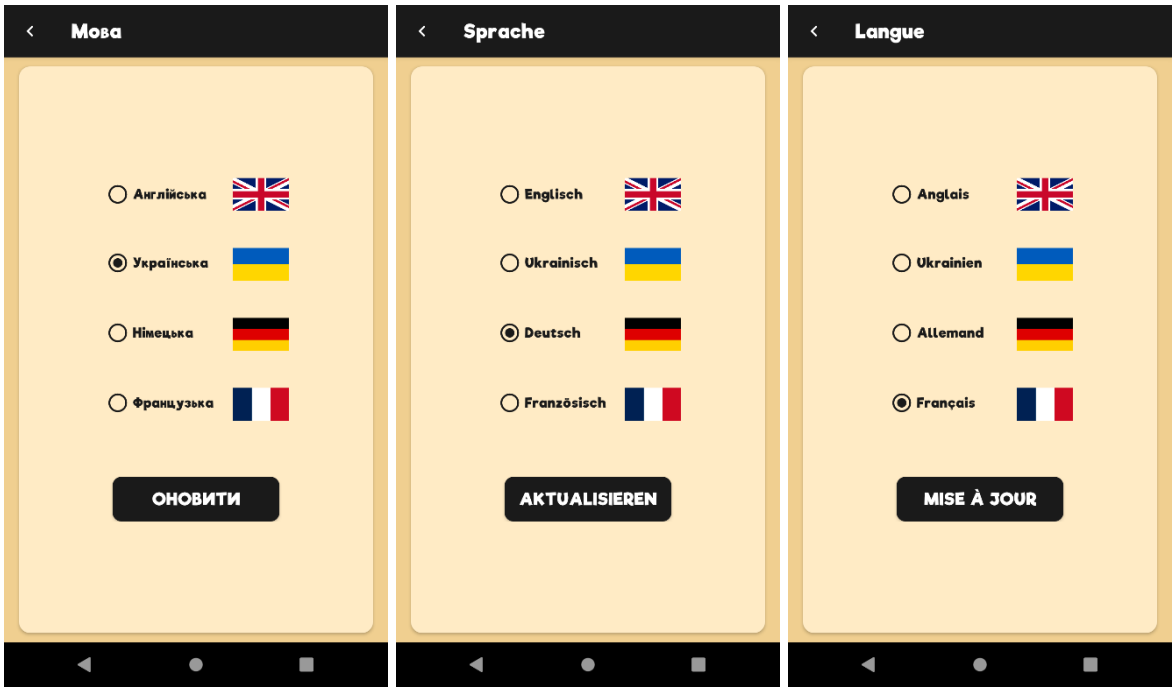


Рисунок 4.13 – Зміна мови інтерфейсу

ВИСНОВКИ

Оцінка одержаних результатів. Розроблено мобільний застосунок для для організації спільної роботи в команді. В процесі дослідження були виявлені основні проблеми та недоліки існуючих застосунків, а також встановлені вимоги до застосунку для організації спільної роботи.

Отже, робота успішно виконала поставлені задачі - застосунок забезпечує планування та контроль завдань, сприяє організації командної роботи та поліпшенню продуктивності проектів.

Інформація щодо створення програмного продукту. На основі аналізу існуючих мобільних застосунків було розроблено концепцію та функціональність застосунку. Вибрані технології та інструменти розробки, такі як Android Studio та Firebase, що забезпечують потужність, надійність та зручність розробки.

Під час розробки була створена архітектура застосунку, а також розроблено інтерфейс користувача, який враховує зручність використання та естетичні аспекти.

Використання результатів роботи. Застосування такого застосунку може покращити організацію та ефективність роботи команди проекту, забезпечуючи зручний доступ до завдань, контроль виконання та зручну спілкування. Використання такого застосунку сприятиме підвищенню продуктивності та організованості проектних команд, зменшенню часу на управління та збільшенню швидкості досягнення поставлених цілей.

Доцільність продовження розробки. Усі розроблені технологічні рішення та результати дослідження можуть бути використані як підґрунтя для подальших досліджень та розробок у галузі управління проектами та розробки мобільних застосунків. Робота зробить свій внесок у практику управління проектами та сприятиме покращенню організації роботи команд у проектних середовищах.

Рекомендації щодо подальших напрямків розвитку застосунку включають розширення функціональності шляхом додавання додаткових модулів, покращення інтерфейсу користувача для поліпшення його зручності та ергономіки, вдосконалення алгоритмів планування та призначення завдань, а також, розробка версій застосунку для інших платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Developers [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Сайт-блог про розробку мобільних застосунків для платформи Android – Режим доступу: https://developer.android.com/studio/intro
2. Технологія firebase [Редакція від 23.02.2023 р.]; [Електронний ресурс] // Сайт компанії Firebase з актуальною документацією – Режим доступу: https://firebase.google.com/docs/reference/android/com/google/firebase/package-summary
3. Технологія firabase.database [Редакція від 23.02.2023 р.]; [Електронний ресурс] // Сайт компанії Firebase з актуальною документацією – Режим доступу:https://firebase.google.com/docs/reference/android/com/google/firebase/database/package-summary
4. Технологія firabase.firestore [Редакція від 02.05.2023 р.]; [Електронний ресурс] // Сайт компанії Firebase з актуальною документацією – Режим доступу:https://firebase.google.com/docs/reference/android/com/google/firebase/firestore/package-summary
5. Kotlinlang [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Сайт компанії Kotlin Foundation з актуальною документацією – Режим доступу: https://kotlinlang.org/docs/home.html
6. Stack Overflow [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Стаття на тему перекладу застосунку на інші мови– Режим доступу: https://stackoverflow.com/questions/23736886/how-to-translate-my-android-application
7. GitHub [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Бібліотека для створення діаграм – Режим доступу: https://github.com/AnyChart
8. Coursera [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Освітній курс на тему розробки Android застосунків – Режим доступу: https://www.coursera.org/projects/android-app-kotlin

9. LinkedIn Learning [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Освітній курс на тему мобільної розробки– Режим доступу: https://www.linkedin.com/learning/android-development-essential-training-5-storage-and-databases
10. Firebase YouTube Channel [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Офіційний YouTube-канал Firebase з освітніми матеріалами– Режим доступу: https://www.youtube.com/watch?v=27BUpiAXt9M&list=PLl-K7zZEsYLnEJf04tHJXScElXcm_xOes
11. Medium [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Стаття про основи роботи з Cloud Firestore – Режим доступу: https://medium.com/firebase-developers/cloud-firestore-basics-in-android-98ccabbc949b
12. Habr [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Сайт спрямований на обмін знаннями та інформацією в галузі програмування – Режим доступу: https://habr.com/ru/articles/344428/
13. Dev.to [Редакція від 16.05.2023 р.]; [Електронний ресурс] // Стаття на тему повідомлень та технології FCM – Режим доступу: https://dev.to/alihakemy/how-to-send-notification-from-android-to-android-using-firebase-fcm-and-volly-44dc
14. Metanit [Редакція від 16.10.2021 р.]; [Електронний ресурс] // Стаття про теми та стилі в Android – Режим доступу: https://metanit.com/java/android/6.2.php
15. Fandroid.info [Редакція від 14.04.2023 р.]; [Електронний ресурс] // Стаття на тему розробки Android застосунку для чату – Режим доступу: https://www.fandroid.info/kak-sozdat-android-prilozhenie-dlya-chata-s-pomoshhyu-firebase/