МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ANDROID ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ПРОЄКТАМИ»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
І.В. Гавриш
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Гавришу Івану Васильовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка Android застосунку для управління проєктами

керівник роботи Кудін Олексій Володимирович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка програмного забезпечення для управління проєктами.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	05.02.2024	
3.	Обробка методичних та теоретичних	04.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	29.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2024	

Студент _____ І.В. Гавриш
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка Android застосунку для управління проєктами»: 71 с., 29 рис., 2 табл., 22 джерела, 1 додаток.

ЗАСТОСУНОК, СИСТЕМИ УПРАВЛІННЯ ПРОЄКТАМИ, УПРАВЛІННЯ ПРОЄКТАМИ, ANDROID, FIREBASE, FIRESTORE, KANBAN, KOTLIN.

Об’єкт дослідження – процес розробки системи управління проєктами.

Предмет дослідження – Android SDK, бібліотеки для роботи з ним, засоби обробки даних на стороні клієнта.

Мета роботи: розробка Android застосунку засобами мови програмування Kotlin для управління проєктами.

Метод дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання та тестування програмного забезпечення.

В процесі розробки застосунку було проаналізовано предметну область, використовуваний інструментарій та бібліотеки, спроєктовано структуру бази даних та архітектуру системи за допомогою UML діаграм. Серверну частину реалізовано з використанням сервісу Firebase, а саме таких його інструментів, як Cloud Firestore, Firebase Authentication і Storage. Клієнтська частина була розроблена з використанням Android SDK з використанням мови програмування Kotlin.

В результаті роботи отримано мобільний застосунок для управління проєктами.

SUMMARY

Bachelor’s qualifying paper “Development of an Android Application for the Project Management”: 71 pages, 29 figures, 2 tables, 22 references, 1 supplement.

APPLICATION, PROJECT MANAGEMENT SOFTWARE, PROJECT MANAGEMENT, ANDROID, FIREBASE, FIRESTORE, KANBAN, KOTLIN.

The object of the study is the process of developing a project management system.

The subject of the study is the Android SDK, libraries for working with it, and client-side data processing tools.

The aim of the study is to develop an Android application using the Kotlin programming language for project management.

The methods of research are methods of collecting and analyzing software requirements, methods of modelling, designing, constructing, and testing software.

During the application development process the subject domain, utilized software frameworks and libraries were analyzed, the database structure and system architecture were designed using UML diagrams. The server part was implemented using the Firebase service, namely its tools such as Cloud Firestore, Firebase Authentication and Storage. The client side was developed using the Android SDK with the Kotlin programming language.

The result is a mobile project management application.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

БД	База даних
JSON	JavaScript Object Notation
MVVM	Model-View-ViewModel
SDK	Software Development Kit
UI	User Interface
UML	Unified Modeling Language

ВСТУП

Планування та управління є невід’ємними процесами життєвого циклу майже кожного проєкту. Від чітко побудованого плану залежить загальна тривалість, якість, бюджет, продуктивність команди, вчасне виконання проєкту та його завдань. Використання сучасних методологій та систем для управління проєктами дозволяє оптимізувати роботу, забезпечити розподілення задачі між працівниками, заощадити ресурси та зменшити ризики. Швидкий та зручний доступ до таких систем забезпечують мобільні застосунки, які хоч і можуть мати обмежений функціонал у порівнянні з їх веб або настільними версіями, проте дозволяють переглядати та керувати основними задачами проєкту «на ходу». Завдяки доступності мобільних пристроїв та гаджетів, проєктування та реалізація мобільних додатків для управління проєктами залишається актуальним завданням для розробників.

Метою кваліфікаційної роботи є розробка Android застосунку засобами мови програмування Kotlin для управління проєктами. Для досягнення мети було визначено для вирішення наступні задачі:

– аналіз систем та методологій управління проєктами;
– визначення вимог до розробки програмного забезпечення;
– аналіз і опис обраного інструментарію;
– проєктування застосунку та його компонентів;
– реалізація застосунку з використанням мови програмування Kotlin;
– тестування та перевірка роботи застосунку.

Об’єкт дослідження – процес розробки системи управління проєктами.

Предмет дослідження – Android застосунок (для управління проєктами).

Структурно робота складається з трьох розділів.

У першому розділі проаналізовано системи та методології управління проєктами, визначено вимоги до розробки застосунку, описано обраний інструментарій, який використовувався при реалізації програмного забезпечення.

У другому розділі описано проєктування застосунку та бази даних, наведено UML діаграми з описом структурних компонентів програми.

У третьому розділі наведено приклади реалізації застосунку, його тестування та роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБРАНОГО ІНСТРУМЕНТАРІЮ, ВИЗНАЧЕННЯ ВИМОГ

1.1 Аналіз предметної області

Предметною областю даної кваліфікаційної роботи є системи управління проєктами. При виконанні аналізу визначено поняття «управління проєктами», описано його цілі, підходи, методології та програмне забезпечення, використовуване для керування проєктами.

1.1.1 Управління проєктами

Управління проєктами – це процес застосування принципів, методів, навичок, знань та досвіду для успішного досягнення конкретних цілей проєкту відповідно до поставлених вимог в межах погоджених норм і стандартів. Основними показниками, які відрізняють управління проєктами від звичайного «управління», є кінцевий результат і обмежений часовий проміжок, на відміну від менеджменту, який є безперервним процесом [1]. Перш ніж перейти до аналізу цілей керування проєктами, доцільно дати визначення самого поняття «проєкту».

Проєкт – це унікальний набір процесів із обмеженими вимогами до часу, ресурсів і якості, спрямованих на досягнення цілей і завдань для створення продукту, послуги або результату. Проєкти існують в межах організації і не функціонують як замкнені системи. Вони потребують вхідних даних від організації та інших сторін, а також забезпечують повернення результатів, які вони можуть використовувати [2, с. 3].

Управління проєктами спрямоване на створення кінцевого продукту,

який забезпечить певні результати на користь організації, що ініціювала проєкт. До основних завдань керування проєктами відносять [3]:

– досягнення цілей проєкту відповідно до визначених вимог – передбачає ретельне планування та розподіл ресурсів для забезпечення досягнення цілей проєкту, не виходячи за визначені межі обсягу, часу та фінансів;
– покращення співпраці та комунікації в команді – забезпечує проведення регулярних зустрічей, обговорень та консультацій, узгодження вимог з клієнтами та обмін ідеями, щоб упевнитись, що всі учасники команди працюють над досягненням одних і тих самих цілей;
– розподіл бюджету та ресурсів – для використання виділених коштів у найбільш ефективний спосіб для зменшення витрат та збільшення прибутку;
– керування ризиками та змінами – полягає у виявленні та усуненні можливих ризиків, швидкому пристосуванні до змін і забезпеченні досягнення цілей проєкту в умовах непередбачуваних (невизначених) ситуацій.

1.1.2 Методології управління проєктами

Методологія управління проєктами – це набір правил, принципів, інструментів і технік, які використовуються для планування, виконання та управління проєктами. Вибір єдиної та універсальної методології є ефективним підходом для об’єднання попередніх результатів з управління проєктами в організації [4]. Незважаючи на те, що різні методології мають різні процеси та результати, вони всі дотримуються базових принципів: планування проєкту, виконання роботи, відстеження прогресу та управління змінами обсягу робіт. Існує багато різних методологій управління проєктами, одні з найпопулярніших яких описано у таблиці 1.1 [5, 6].

Таблиця 1.1 – Методології управління проєктами

Назва	**Опис**	**Кому підійде**
Waterfall	Традиційний лінійний підхід, в якому задачі виконуються послідовно. Тобто, кожен етап має бути завершений до того, як буде розпочато наступний. Передбачає, що всі вимоги будуть зібрані заздалегідь перед початком роботи.	Невеликим проєктам з чітко визначеними і більш передбачуваними результатам и, оскільки зазвичай до попередніх етапів вже не повертаються, і на будь-які зміни потрібно буде витратити велику кількість грошей і часу.
Agile	Гнучкий підхід, що передбачає виконання роботи короткими циклами, які називаються спринтами. Спринти дають змогу постійно переоцінювати та вдосконалювати проєкт в залежності від змін вимог замовника.	Проєктам, які не мають чітко визначених кінцевих вимог, потребують гнучкості у виконанні, і замовники яких готові до постійного діалогу. Також підійде тим, кому важливі швидкі, а не досконалі результати.
Scrum	Підхід, який використовує спринти для створення проєктного циклу, що тривають від одного до двох тижнів і виконуються командами з 10 або менше осіб. Команди очолює Scrum-майстер, який проводить щоденні та інші зустрічі.	Проєктам із різних галузей, які потребують гнучкості у реалізації, і не мають заздалегідь остаточного плану. Також може підійти командам, які знайомі, або працювали, із Agile підходом.
Kanban	Гнучкий підхід, що ґрунтується на використанні Kanban дошок для візуалізації	Завдяки універсальності даного підходу, його можна використовувати у проєктах

Продовження таблиці 1.1

Назва	Опис	Кому підійде
	поставлених задач. Дошки складаються з колонок із завданнями, які члени команди, по мірі їх виконання, переміщують між цими стовпчиками, поки всі етапи проєкту не будуть завершені.	будь-якого розміру та напрямку. Можлива комбінація із Scrum підходом – Scrumban – використання принципів Scrum у поєднанні із візуалізацією Kanban-дошок.

Через існування досить великої кількості методологій та їх гібридних варіантів, командам можем бути важко визначитись з тим, яку краще застосовувати. Вибір підходящої методології є важливим етапом будь-якого проєкту, адже він безпосередньо впливатиме на ефективність, якість та результати його виконання. Тому, перед тим як вибрати одну з них, слід розглянути такі питання [7]:

– структура, сильні та слабкі сторони команди й компанії (організації);
– динамічні чи фіксовані вимоги у замовника;
– кількість зацікавлених сторін у проєкті;
– тип, розмір, бюджет, складність, умови виконання та часові обмеження проєкту;
– чи має команда досвід використання будь-якої з методологій управління проєктами.

В залежності від відповідей та думок працівників на поставлені питання, керівники разом з командою зможуть обрати найбільш слушну методологією для майбутнього проєкту. Дотримання принципів та технік обраної методології дозволить не тільки підвищити якість роботи, а й зменшити виникнення потенційних ризиків, оптимізувати діяльність і взаєморозуміння команди.

1.1.3 Системи управління проєктами

Одним із основних інструментів, який допомагає у менеджменті проєктами, є система управління проєктами. Це програмне забезпечення, яке, в залежності від його складності та функціональності, дозволяє користувачам керувати задачами та бюджетом, створювати плани проєктів, розподіляти та призначати ресурси, обмінюватись файлами, спілкуватись тощо. Використання програм, сервісів та інструментів, які відповідають потребам команди та проєкту, має на меті покращити взаємодію між працівниками та їх завданнями, полегшити відслідковування фінансів та часу, обмінюватись необхідними документами тощо.

Станом на сьогодні існує велика кількість прикладних, веб та мобільних застосунків для керування проєктами. Вибір правильного програмного забезпечення є важливим організаційним етапом, адже від обраної системи буде залежати взаємодія команди, процес виконання задач, відстеження наявних ресурсів тощо. Таблиця 1.2 надає порівняння популярних програм для керування проєктами, а саме: переваги та недоліки, вартість та типи планів інструменту [8].

При виборі системи управління проєктами, варто зосередити увагу на визначених замовником бізнес-вимогах, технічних характеристиках, які задовольняють ці вимоги, зручності та можливостях користувацького інтерфейсу, функціональності та кривої навчання інструменту, а також проявити об’єктивність під час його вибору. Слід зазначити, що, хоча, використання якісного та ефективного програмного забезпечення й збільшує ймовірність успіху проєкту, але все таки не гарантує його. Врешті решт, система управління проєктами – це звичайний інструмент, який в руках недосвідченого менеджера, чи керівника, не створить дива. Тому, до вибору тієї чи іншої програми хоч і потрібно підходити свідомо та відповідально, проте потрібно розуміти, що навіть багатофункціональні сервіси не зможуть вирішити проблеми, пов’язані з поганим плануванням, чи розподілом задач [9].

Загалом, системи управління проєктами, які являють собою сукупність процесів, інструментів, функцій та методологій, допомагають менеджеру у керуванні проєктом та його ефективному доведенні до завершення. При виборі підходящої програми потрібно враховувати визначені бізнес-вимоги, їх особливості, часові терміни, необхідний інструментарій, присутній функціонал та зручність програми для роботи команди.

Таблиця 1.2 – Порівняння систем управління проєктами

Сервіс	**Переваги**	**Недоліки**	**Вартість планів (user/month)**
Jira [10]	Широкий набір функцій; підтримка методологій Scrum і Kanban; велика кількість інтеграцій; зручне відслідковування прогресу; можливість генерації звітів; підходить для різних типів команд.	Трохи складний у освоєнні інтерфейс; обмежений у функціоналі мобільний додаток; дорогий для невеликих команд; повільне завантаження запитів.	В залежності від розміру команди: безкоштовний; стандартний – 4.87-8.15$ на місяць; преміум – 7.40-16$ на місяць; корпоративний (від 800 користувачів) – від 141000$ на рік.
Asana [11]	Зручний та зрозумілий інтерфейс; широкий набір інструментів візуалізації; інтеграція зі сторонніми сервісами; зручний мобільний додаток.	Можливість призначити одну людину на задачу; підтримка експорту лише у JSON і CSV; більшість функцій доступні лише у платних підписках.	Безкоштовний; початковий – 11$; розширений – 25$.

Продовження таблиці 1.2

Сервіс	**Переваги**	**Недоліки**	**Вартість планів (user/month)**
Trello [12]	Використання Kanban-дошок, як основний інструмент; ефективна система сповіщень; підтримка офлайн режиму; зручний мобільний застосунок.	Обмежена кількість дошок у безкоштовній версії; проблеми з масштабуванням; обмежене хмарне сховище.	Безкоштовний; стандартний – 5$; преміум – 10$; корпоративний (від 50 користувачів) – від 17.50$.
Monday .com [13]	Підтримка тайм-трекеру; багато інструментів для візуалізації; зручне планування та управління проєктами; можливість відстеження завдань на кількох дошках.	Не дуже зручний мобільний застосунок; може бути важким у вивченні; дещо складний інтерфейс.	Безкоштовний; базовий – 12€; стандартний – 14€; про – 24€.
Kanban Board [14]	Зручний інтерфейс; встановлення часу на виконання задач; можливість візуалізації задач	Не працює календар для встановлення термінів виконання задач; відсутня можливість додавання учасників до дошок та задач.	Безкоштовний; преміум – 2$.

1.2 Визначення вимог

Метою даної кваліфікаційної роботи є розробка Android застосунку для управління проєктами. Як було розглянуто у пункті 1.1.3 цього розділу, такі програми можуть мати багатий інструментарій та функціонал, або бути зосередженими на підтримці якоїсь однієї методології, як наприклад Trello. Для даної системи було вирішено розробити застосунок для управління проєктами методом Kanban. Створюваний додаток повинен надавати користувачам можливість керувати проєктами та їх задачами з використанням візуальних Kanban-дошок.

Загалом, можна виділити наступні функціональні можливості системи:

- реєстрація та автентифікація користувачів у системі;
- редагування облікового запису користувача;
- створення/редагування/перегляд/видалення робочих просторів (workspaces);
- створення/редагування/перегляд/видалення Kanban-дошок;
- створення/редагування/переміщення/видалення списків задач;
- створення/редагування/перегляд/переміщення/видалення задач;
- додавання/видалення користувачів до робочих просторів;
- додавання/видалення користувачів до Kanban-дошок;
- призначення/видалення користувачів до задач;
- налаштування прав доступу користувача у робочому просторі;
- налаштування прав доступу користувача у Kanban-дошці;
- підтримка роботи системи в режимі офлайн;
- перегляд всіх задач призначених користувачу.

При створені задачі користувачі мають змогу задавати її назву, опис, встановлювати дату початку та закінчення виконання задачі, створювати та додавати теги, призначати та видаляти відповідальних за виконання цієї задачі.

У робочому просторі має бути два типи ролей користувачів –

адміністратор і учасник. Адміністратори повинні мати змогу створювати, редагувати та видаляти робочий простір і дошки; додавати до користувачів до простору, видаляти їх, керувати їх ролями, окрім власника простору. Учасники можуть тільки переглядати дошки, а редагувати тільки ті, до яких вони приєднані.

Як і в робочому просторі, дошки повинні підтримувати два типи ролей користувачів – адміністратор і учасник. Адміністратори можуть створювати, редагувати, видаляти списки задач і самі задачі; додавати користувачів до дошки, видаляти їх, керувати їх ролями, окрім власника дошки; створювати, редагувати та видаляти теги. Учасники можуть створювати, редагувати, видаляти списки задач і самі задачі; створювати, редагувати та видаляти теги.

Розроблюваний застосунок повинен підтримувати та бути сумісним із смартфонами з версією Android не нижче ніж 8.0 «Oreo» (рівень API 26), мати зручний та зрозумілий користувацький інтерфейс, працювати без збоїв у 90 відсотках випадків використання протягом місяця.

1.3 Опис обраного інструментарію

Завданням даної роботи є розробка Android застосунку для управління проєктами. Для реалізації додатку було вирішено використовувати мову програмування Kotlin. Це статично типізована, багатоцільова, високорівнева мова програмування, розроблена компанією JetBrains, яка з 2017 року підтримується Google для Android розробки. Однією з головних особливостей мови Kotlin є його повноцінна сумісність з Java, що дає змогу використовувати вже її існуючі бібліотеки та фреймворки при розробці. Серед інших характеристик можна виділити: більш лаконічний синтаксис, підтримку функціонального програмування (лямбда-вирази, функції вищого порядку тощо), розширення функцій, співпрограми тощо.

Під час розгляду мови програмування Kotlin варто звернути увагу на

бібліотеку kotlinx.coroutines, яка надає підтримку співпрограм. Співпрограма (англ. Coroutine) – це шаблон паралельного програмування, який можна використовувати для спрощення написання асинхронного коду [15]. При розробці Android застосунків, співпрограми допомагають керувати виконанням довготривалих операцій, які потенційно могли б заблокувати головний потік, що призвело б до зависання додатку. До таких операцій можна віднести, наприклад, виконання інтернет запитів або запитів до бази даних, запис до або зчитування з файлу, виконання важких обчислень тощо.

Однією із складових бібліотеки kotlinx.coroutines є flows. Flow, або потік, відображає потік значень, які обчислюються асинхронно. Для створення та отримання значень Flow використовує функції призупинення (англ. Suspend functions), які виконуються не блокуючи головний потік програми. Flow можуть бути корисні для роботи з потоком даних, наприклад, в результаті виконання запиту до бази даних для отримання оновлень в режимі реального часу.

У якості backend-сервісу для розроблюваного застосунку було вирішено використовувати Firebase-сервіси. Firebase – платформа, що належить компанії Google, і пропонує розробникам сервіси та інструменти для реалізації таких функцій у додатках, як автентифікація користувачів, зберігання файлів у хмарному сховищі, зберігання інформації у базі даних, аналітика роботи програми тощо. Простота розгортання сервісів Firebase, та їх активна підтримка командою розробників, робить його вдалим вибором для використання в розробці Android застосунку. Розглянемо інструменти Firebase, обрані для впровадження в програмній реалізації [16].

Одним із сервісів, який надає платформа Firebase, є Firebase Authentication. Даний інструмент надає можливість автентифікації користувачів у системі і підтримує цілу низку відповідних методів входу в акаунт: за допомогою пошти і пароля, номеру телефона, облікового запису Google, GitHub, LinkedIn тощо. Завдяки забезпеченню такого широко кола провайдерів, розробники можуть обрати для реалізації ті способи, що

задовольняють вимоги користувачів і замовника.

Сервіс Firebase Firestore надає можливість зберігати та синхронізувати дані клієнта для використання на кількох пристроях. Firestore являє собою хмарну NoSQL документо-орієнтовану базу даних для зберігання інформації у режимі реального часу, що також підтримує офлайн режим. Перелічені характеристики роблять цей інструмент підходящим для використанні у застосунку з управління проєктами, де всі оновлення даних повинні відбуватись швидко та одразу відображатись у всіх користувачів системи. Окрім того, можливість перегляду даних без доступу до мережі є важливою особливістю додатку, яку Firestore теж забезпечує.

Для підтримки можливості додавання вкладень у задачі, а також зміни світлини профілю користувача і обкладинки дошки, було вирішено використовувати такий Firebase-сервіс, як Cloud Storage. Він забезпечує зберігання та завантаження файлів у простий та зручний спосіб, що робить його важливою частиною застосунку, адже обмін файлами є невід'ємною складовою системи управління проєктами.

При проєктуванні класів розробники часто зіштовхуються з питанням впровадження залежностей. Ін'єкція залежностей (англ. Dependency Injection) – це шаблон проєктування, який передбачає надання залежностей ззовні, замість створення їх конкретних реалізацій всередині класу. Для централізованого та зручного керування залежностями та програмними компонентами, які їх потребують, у розроблюваному Android застосунку було вирішено використовувати бібліотеку Hilt. Hilt забезпечує зручний спосіб впровадження залежностей, надаючи контейнери для кожного класу Android у проєкті та автоматично керуючи їхніми життєвими циклами.

Для відображення зображень у застосунку, наприклад, фото профілю користувача, використовуватиметься бібліотека Glide. Вона підтримує отримання, декодування та відображення відео, зображень та анімованих GIF-файлів як з локального сховища, так і з мережі. Увага бібліотеки Glide зосереджена на забезпеченні високої продуктивності та ефективному

використанні ресурсів системи задля збереження швидкодії роботи застосунку при роботі з зображеннями.

За зберігання певних налаштувань користувача, які змінюються під час використання застосунку, буде відповідати бібліотека DataStore. Вона забезпечує зберігання та отримання як простих, так і складних типів даних у вигляді пар ключ-значення. Сумісність даної бібліотеки із такими особливостями мови програмування Kotlin, як Coroutines та Flow, дозволяє виконувати запити асинхронно, що значно покращує ефективність та швидкість роботи додатка.

2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1 Моделювання варіантів використання

Моделювання варіантів виконується для відображення системних або програмних вимог до нового програмного забезпечення, і відбувається з використанням діаграми прецедентів. Діаграма прецедентів (англ. Use case diagram) – це поведінкова UML діаграма, на якій зображуються різні варіанти використання (прецеденти) системи різними типами користувачів (актори). Ключовою особливістю застосування цього типу діаграми є те, що вона допомагає спроєктувати систему з точки кінцевого користувача. Важливо відмітити, що модель використання повинна бути відносно простою і відображати лише зв’язки між прецедентами та акторами, без зазначення конкретних етапів для досягнення цілей кожного варіанту використання.

На рисунку 2.1 представлена діаграма варіантів використання, на якій зображено можливі дії користувача в системі та їх зв’язки з Firebase-сервісами, відповідальними за виконання backend запитів кожного варіанту використання.

Виділимо та розглянемо більш детально один із основних варіантів використання – «Керувати Kanban-дошками». Увага розроблюваного застосунку зосереджена на впровадженні та використанні основного інструменту методології Kanban – Kanban-дошках. Після реєстрації у системі та створенні свого першого робочого простору, користувачі можуть створювати дошки для розміщення задач, співпраці з іншими учасниками та відстеження прогресу виконання проєкту. Кожна дошка динамічно реагує на всі зміни, що в ній відбуваються, наприклад, зміни її налаштувань. Щодо самих налаштувань дошки, можливо змінити її назву, опис, додати або видалити учасника, керувати тегами, видалити дошку. У кожній дошці

користувач може створювати довільну кількість списків задач (стовпців), а у кожному користувач може розміщувати необмежену кількість задач (карток). Для створення списку необхідно вказати лише його назву. Меню редагування списку надає можливість лише змінити назви та видалення. При створенні задачі користувач обов'язково має вказати її назву. Серед інших полів, доступних до заповнення, є опис задачі, дата початку та кінця її виконання, поле для вибору та призначення учасників дошки до задачі, вибір або створення тегів. Для видалення задачі використовуються меню на верхній панелі керування. На дошці можливо переміщувати як стовпці задач, так і задачі у межах певного стовпця, або між стовпцями.

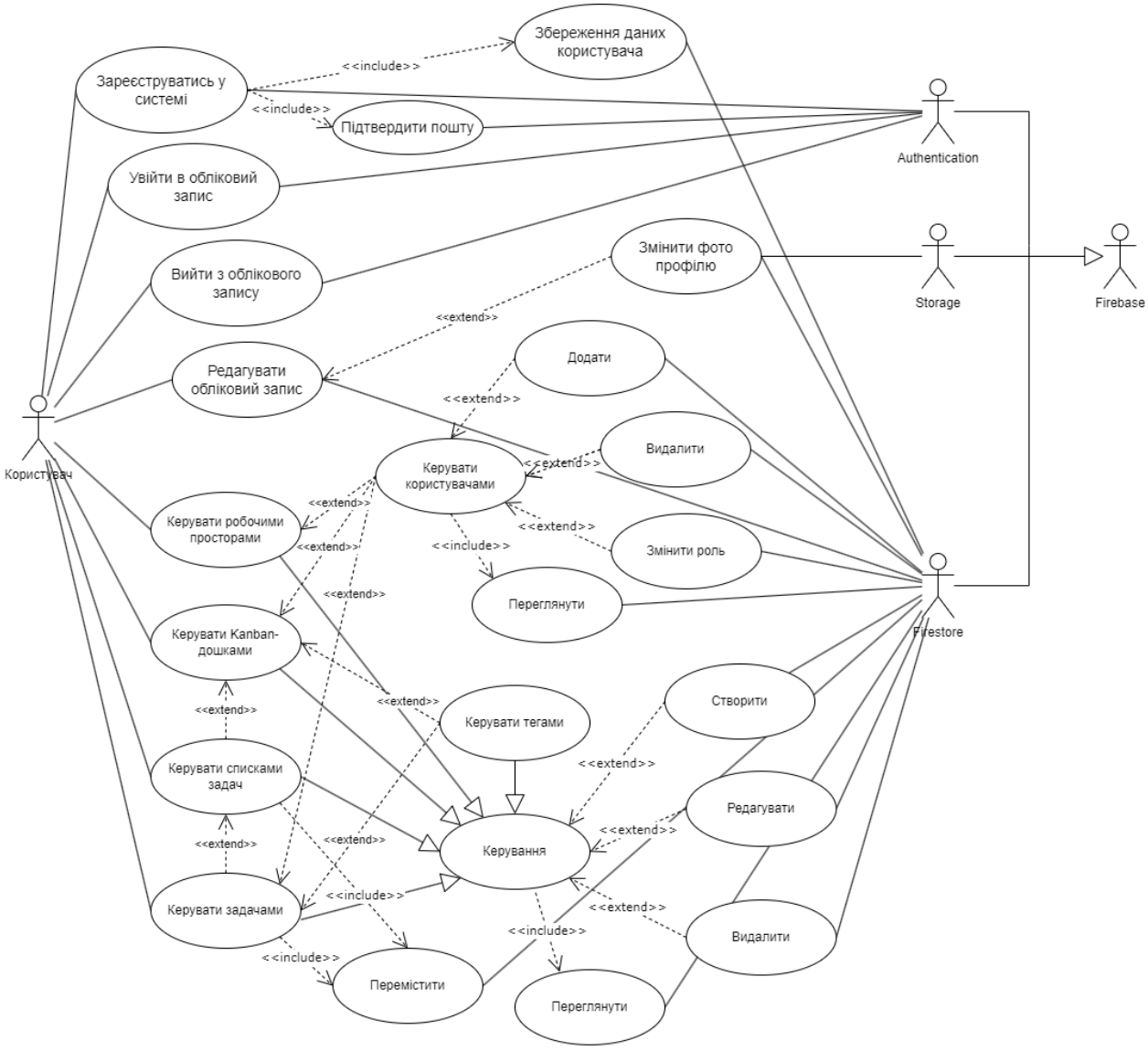


Рисунок 2.1 – Діаграма варіантів використання

На рисунку 2.2 представлено діаграму, яка зображує варіанти взаємодії користувачів із робочими просторами згідно їх ролей. Таким чином, кожен користувач може створювати робочі простори, однак, в залежності від його ролі, будуть обмежуватись його дії в них. За замовчуванням кожен власник робочого простору, тобто користувач, який його створив, є його адміністратором, і не може надалі змінити цю роль. Адміністратори мають повний контроль над робочим простором, тобто можуть його редагувати, видаляти, додавати та видаляти користувачів., змінювати їх роль. У свою чергу, учасники робочого простору можуть лише переглядати створені у ньому Kanban-дошки.

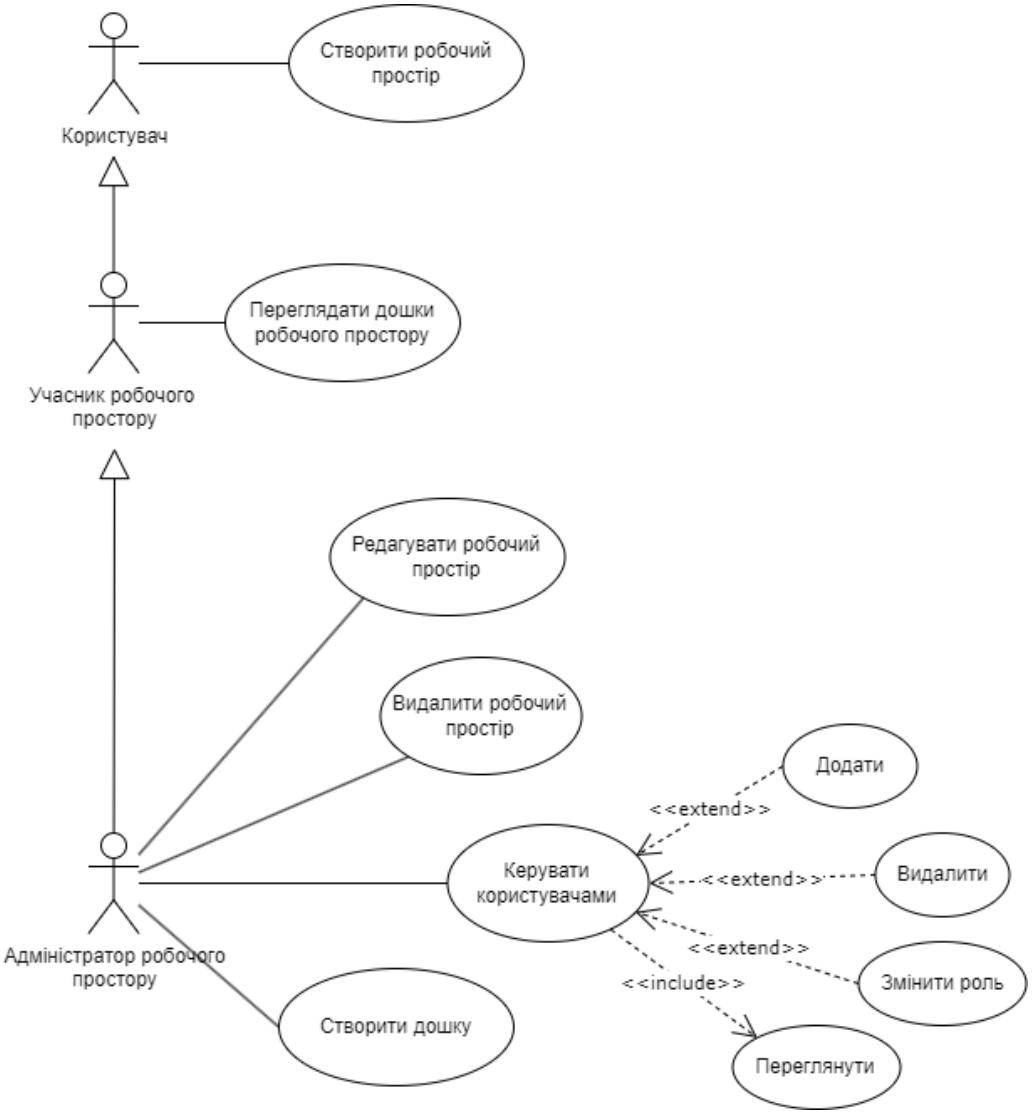


Рисунок 2.2 – Діаграма варіантів використання робочих просторів користувачами з різними ролями

На рисунку 2.3 змальовано діаграму варіантів використання Kanban-дошок користувачами із різними ролями як на самих дошках, так і в робочих просторах, в яких вони розміщуються.

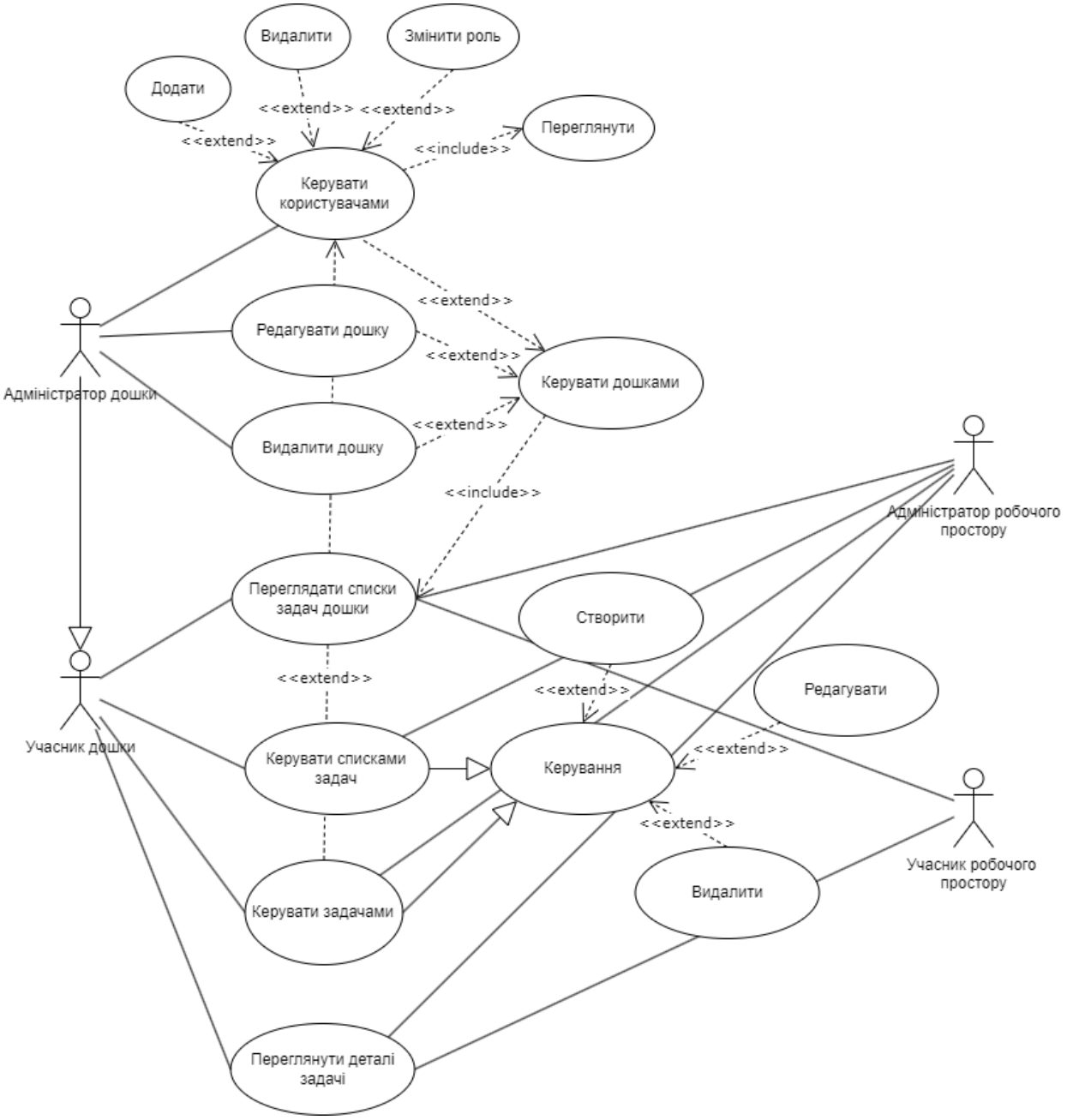


Рисунок 2.3 – Діаграма варіантів використання Kanban-дошок користувачами з різними ролями

Розглянемо можливі дії користувачів на дошках в залежності від їх ролі у робочому просторі, до яких їх додано. Адміністратори можуть створювати нові дошки, редагувати списки задач і самі задачі на вже існуючих дошках,

однак не можуть змінювати налаштування дошок. Учасники робочого простору можуть лише переглядати існуючі дошки та їх вміст без можливість внесення будь-яких змін. Розглядаючи варіанти використання дошок користувачами в залежності від їх ролі на них варто відмітити, що відмінність між учасником та адміністратором дошки полягає лише в можливості зміни її налаштувань останнім. Як адміністратори, так і учасники можуть вільно створювати. редагувати та видаляти списки задач і самі задачі, керувати їх учасниками та тегами.

2.1.1 Опис основних варіантів використання

У даному пункті описано основні варіанти використання розроблюваної системи з управління проєктами згідно рисунку 2.1.

Прецедент «Зареєструватись у системі».

Призначення: реєстрація облікового запису користувача у системі.

Основний потік подій: на екрані реєстрації користувач вводить необхідні дані для створення акаунту, і натискає кнопку «Sign Up». При успішній реєстрації виконується зберігання даних користувача (пошта, ім’я користувача, посилання на зображення профілю) до Firestore БД.

Альтернативний потік подій: якщо користувач вже зареєстрований, система відобразить відповідне повідомлення.

Виняткова ситуація 1: введено неправильні дані – система відобразить відповідне повідомлення.

Прецедент «Увійти в обліковий запис».

Призначення: вхід користувачем в обліковий запис системи.

Основний потік подій: користувач вводить дані облікового запису даний варіант використання починає виконуватися, коли користувач натискає кнопку «Sign in» на екрані реєстрації або входу в застосунок. При успішному вході в акаунт користувача буде перенаправлено на головну сторінку системи.

Альтернативний потік подій: якщо користувач не зареєстрований, система відобразить відповідне повідомлення.

Передумова: користувач має бути зареєстрований у системі.

Виняткова ситуація 1: введено неправильні дані – система відобразить відповідне повідомлення.

Прецедент «Вийти з облікового запису».

Призначення: вихід користувачем із облікового запису системи.

Основний потік подій: користувач натискає кнопку «Sign out» на боковому меню на головному екрані застосунку. При успішному виконанні операції, користувача буде виведено з акаунту системи та перенаправлено на екран реєстрації.

Передумова: користувач повинен увійти до системи.

Прецедент «Редагувати обліковий запис».

Призначення: редагування користувачем інформації його облікового запису.

Основний потік подій: користувач натискає кнопку «Edit profile» на екрані налаштувань. Після редагування профілю (зміна імені, оновлення зображення профілю тощо) валідними даними, користувач натискає кнопку «Save», система зберігає внесені зміни, та повертає користувача назад на екран налаштувань.

Альтернативний потік подій: якщо користувач не виконав ніяких змін і натиснув кнопку «Save», система перенаправляє користувача на екран налаштувань.

Передумова: користувач повинен увійти до системи.

Виняткова ситуація 1: введено неправильні дані – система відобразить відповідне повідомлення. Користувач може змінити дані.

Прецедент «Керувати робочими просторами».

Призначення: керування користувачем робочими просторами.

Основний потік подій: користувач знаходиться на головному екрані системи, на якому він має доступ до створення та редагування робочих

просторів. Обравши робочий простір, користувач може переглянути його вміст, а саме: створені в ньому Kanban-дошки.

Для створення робочого простору використовується кнопка «Create workspace» на боковому меню.

Для зміни налаштувань робочого простору, використовується кнопка «Workspace settings» на панелі інструментів. Після її натискання користувача буде перенаправлено на екран налаштувань робочого простору, де він може змінити назву робочого простору, керувати його учасниками, видалити робочий простір.

Альтернативний потік подій: якщо не створено жодного робочого простору, система відобразить відповідне повідомлення.

Передумова: користувач повинен увійти до системи.

Прецедент «Керувати Kanban-дошками».

Призначення: керування користувачем Kanban-дошками.

Основний потік подій: користувач обирає дошку в поточному робочому просторі на головному екрані системи. Після вибору дошки, користувача буде перенаправлено на екран дошки, де він може переглянути та керувати її вмістом.

Для створення дошки в обраному робочому просторі, на головному екрані використовується кнопка «Create board».

Для зміни налаштувань дошки, використовується кнопка «Board settings» на панелі інструментів. Після її натискання користувача буде перенаправлено на екран налаштувань дошки, де він може змінити назву та опис дошки, керувати її учасниками та тегами, видалити дошку.

Альтернативний потік подій: якщо в обраному робочому просторі не створено жодної дошки, система відобразить відповідне повідомлення.

Передумова: користувач знаходиться на головному екрані застосунку, попередньо обравши робочий простір.

Прецедент «Керувати списками задач».

Призначення: керування користувачем списками задач на Kanban-

дошках.

Основний потік подій: користувач знаходиться на екрані обраної Kanban-дошки, на якому він може створювати, переміщати та редагувати списки задач.

Для створення списку задач використовується кнопка «Create list», яка завжди знаходиться в кінці всіх списків.

Редагування списків відбувається шляхом натискання кнопки меню вгорі кожного списку і вибором елементу «Edit name». Пункт меню «Delete» використовується для видалення списку задач.

Переміщення списку задач відбувається шляхом довгого натискання його назви та перетягування пальцем на нову позицію.

Передумова: користувач знаходиться на екрані Kanban-дошки.

Прецедент «Керувати задачами».

Призначення: керування задачами на Kanban-дошках.

Основний потік подій: користувач знаходиться на екрані обраної Kanban-дошки. Створення задач відбувається шляхом натискання кнопки «Create task» внизу списку задач. На екрані редактора задач користувач заповнює дані нової задачі та натискає кнопку «Create task». Після успішного створення задачі, користувача буде перенаправлено назад на екран дошки.

При натисканні на задачу відбувається перехід на екран з інформацією про завдання, на якому користувач може переглянути всі відомості задачі. Також на цьому екрані він може видалити задачу, використавши елемент меню на панелі інструментів «Delete», або перейти до її редагування шляхом натискання кнопки «Edit task».

Переміщення задач відбувається шляхом довгого натискання на картку задачі у списку задач та перетягування її пальцем на нову позицію у поточному або новому списку.

Передумова: користувач знаходиться на екрані Kanban-дошки із щонайменше одним списком задач на ній.

2.2 Архітектура застосунку

Одним із етапів проєктування програмного забезпечення є вибір архітектури, яка буде використовуватись у розроблюваному продукту. Архітектура ПЗ забезпечує абстрактний погляд на структуру та дизайн системи. Вона розглядає зовнішні характеристики всіх компонентів і їх взаємодій, не враховуючи внутрішні деталі реалізації. Метою якісної архітектури є розробка системи, яка відповідає бізнес вимогам, і є надійною, продуктивною, зручною, масштабованою, і підтримуваною [17].

Зазвичай Android-застосунок складається з багатьох компонентів, таких як діяльності (англ. Activity), фрагменти, сервіси тощо, кожен з яких виконує чітку роль під час виконання програми. Для уникнення можливих проблем, пов’язаних із життєвим циклом цих компонентів, покращення ефективності та надійності роботи застосунку, важливо дотримуватись принципу розподілу відповідальностей (англ. Separation of concerns) [18]. Дотримання цього принципу під час розробки застосунку передбачає розмежування бізнес логіки, моделей даних та специфічних для Android компонентів. Такий підхід дозволяє розробникам зосередитись на реалізації основного функціоналу без залежності від Android платформи, що полегшує тестування і повторне використання частин коду. Одним із архітектурних шаблонів, який задовольняє наведеним вимогам є Clean Architecture, запропонований Робертом Мартіном (див. рис. 2.4).

Кожне коло на схемі представляє різні рівні, або слої, програмного забезпечення. Зовнішнє коло відображає конкретні реалізації, які є характерними для платформи. Рухаючись всередину, кожне коло є все більш абстрактним і високорівневим. Центральне коло є найбільш загальним (абстрактним) і містить бізнес-логіку, яка не залежить від середовища або фреймворку, який використовується. Такий підхід називається принципом абстракції. Ще одним принципом, або правилом, Clean Architecture, є правило залежності (англ. Dependency rule). Воно визначає, що залежності повинні йти

тільки всередину, до абстракцій вищого рівня, а не назовні, до елементів нижчого рівня, тобто конкретних реалізацій. Це гарантує, що зміни в компонентах нижчого рівня (зовнішні кола) не впливають на бізнес-правила вищого рівня (внутрішні кола), що сприяє модульності, гнучкості та підтримуваності системи [20].

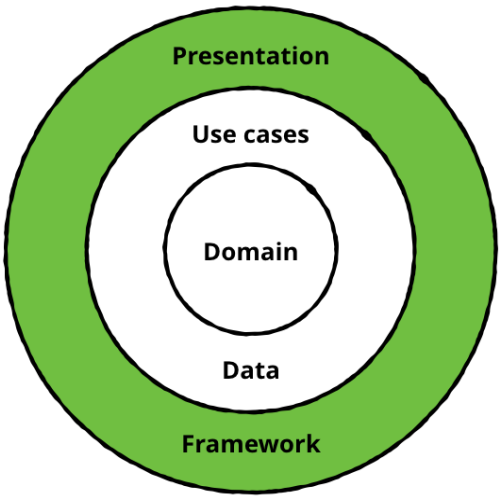


Рисунок 2.4 – Clean Architecture в Android (позначений рівень залежить від Android SDK) [19]

Кожен рівень архітектури має своє призначення.

Рівень представлення (англ. Presentation layer) взаємодіє з користувацьким інтерфейсом системи, відповідає за обробку дій користувача, відображення даних, збереження стану UI тощо. В Android-застосунку цей рівень зазвичай складається з Activities, Fragments, Views, View Models, та інших компонентів інтерфейсу. Важливо відмітити, що цей рівень залежить від доменного рівня і не повинен мати жодних залежностей до рівня даних.

Доменний рівень (англ. Domain layer) містить основну бізнес-логіку і правила програми. Він зазвичай складається з сутностей (англ. Entities) і варіантів використання (англ. Use cases). Цей рівень має залежати від абстракцій, а не конкретних реалізацій або технологій.

Рівень даних (англ. Data layer) відповідає за керування даними, їх

збереження, пошук та взаємодію з ними. Він складається з реалізації інтерфейсів, визначених на доменному рівні, для взаємодії з зовнішніми джерелами даних, такими як бази даних, локальні сховища, вебсервіси або API.

Рівень представлення в архітектурі програмного забезпечення часто використовує різні шаблони проєктування для управління користувацьким інтерфейсом програми, обробки дій користувача і відображення даних. Одним із таких найпоширеніших шаблонів є MVVM. Model-View-ViewModel – це архітектурний шаблон, використовуваний в розробці програмного забезпечення для відокремлення логіки користувацького інтерфейсу від моделі даних. «Model» відповідає за дані та бізнес-логіку програми (доменний рівень та рівень даних), «View» – за користувацький інтерфейс, а «ViewModel» діє як міст між «Model» та «View». «ViewModel» отримує дані з «Model» і оброблює їх для відображення у «View». Вона також обробляє будь-яку взаємодію користувача з «View» і оновлює «Model» відповідним чином.

Поєднання Clean Architecture і MVVM (див. рис. 2.5) дозволяє уникнути дублювання коду, забезпечити його повторне використання, покращити розподіл відповідальностей компонентів програми шляхом використання «UseCases».

Винесення бізнес-логіки у «UseCases» дозволяє зменшити завантаженість та загальну складність класів «ViewModel». Однак необхідно бути обачним під час використання такого підходу, особливо якщо «UseCase» лише викликають, наприклад, функцію репозиторію, не виконуючи при цьому ніякої додаткової логіки. В таких випадках слід обміркувати доцільність використання «UseCase», якщо можливо здійснити виклик цієї функції безпосередньо з «ViewModel».

Отже, використання архітектури (архітектурних шаблонів) при розробці програмного забезпечення може допомогти створити більш надійні, продуктивні, підтримуванні та масштабовані застосунки.

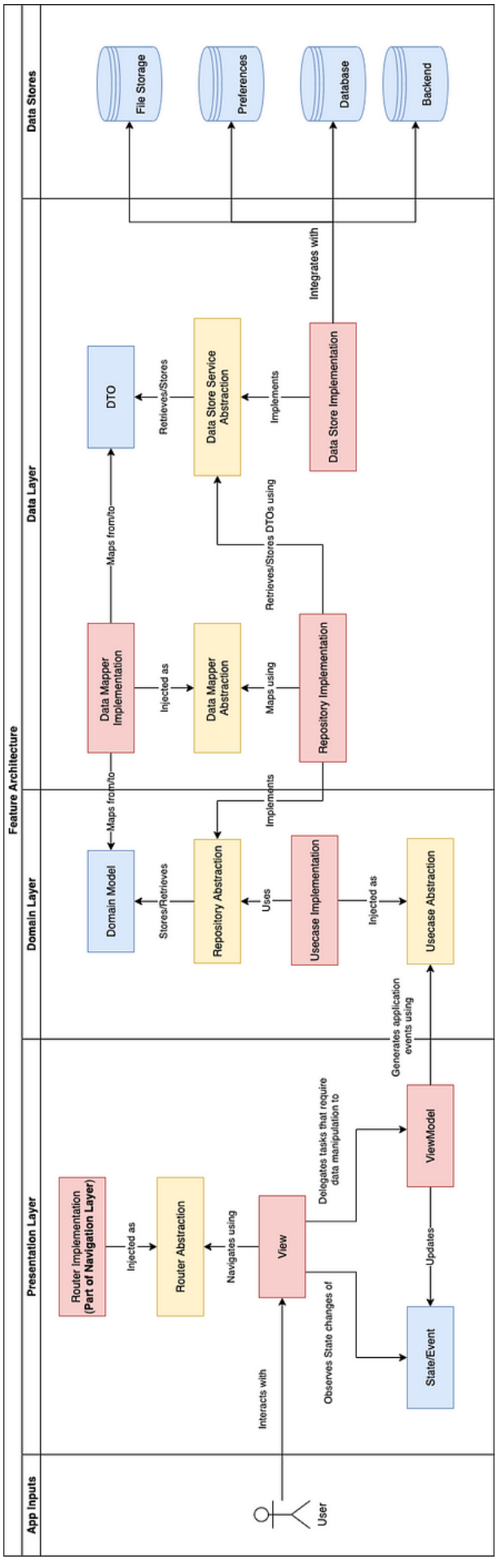


Рисунок 2.5 – Clean Architecture із MVVM в Android [21]

2.3 Проєктування ER-діаграми бази даних

База даних є невід'ємним компонентом системи управління проєктами, який забезпечує централізоване зберігання всіх даних, що курсують у системі, та надає можливість ефективного керування ними. Завдяки йому користувачі можуть швидко вносити та зручно отримувати всю необхідну інформацію про проєкти, їх задачі, учасників тощо. Платформа Firebase, інструменти якої активно використовуватимуться у розроблюваному застосунку, пропонує дві хмарні бази даних – Realtime Database та Cloud Firestore. Обидві є нереляційними БД, які використовують об'єкти JSON для зберігання даних, забезпечують безпечний доступ до них, надають низьку затримку синхронізації між клієнтом і сервером, підтримують використання без інтернету. Серед відмінностей варто відмітити модель, яку вони використовують для зберігання даних. Realtime Database являє собою одне велике дерево JSON, в той час як Cloud Firestore використовує колекції документів із можливістю створення підколекцій, які полегшують структурування складних ієрархічних даних [22]. Так як розроблювана система фактично матиме багаторівневу структуру (робочий простір зберігає дошки, що містять списки задач із, власне, самими задачами) було вирішено обрати саме Firestore БД.

Як було зазначено раніше, Cloud Firestore – це NoSQL, документо-орієнтована база даних призначена для зберігання ієрархічних структур даних, які зазвичай називаються документи і зберігаються в колекціях. Фактично документ у Cloud Firestore – це іменований файл JSON, що містить набір пар ключ-значення, які називаються поля, і який не може бути більше 1 МБ у розмірі. На відміну від реляційних баз даних, Firestore не потребує схеми для організації даних, що зберігаються, тому їх структура може відрізнятись від документа до документа. Такий підхід забезпечує гнучкість у впроваджені можливих змін у вимогах до системи. Ще однією особливістю розглядаємої бази даних є можливість створювати колекції у документах, які дозволяють

ієрархічно групувати дані, оптимізувати їх зберігання та доступ до них. Оскільки максимальний розмір кожного документа становить лише 1 МБ, підколекції забезпечують зберігання схожих за структурою даних, які з часом можуть зайняти весь документ. Таким чином, ми гарантуємо, що користувачеві не доведеться турбуватися про доступний простір, і дозволяємо йому зберігати стільки інформації, скільки йому необхідно.

Незважаючи на те, що Firestore не потребує схеми, дотримання конкретних моделей забезпечує цілісність та узгодженість інформації між застосунком та базою даних. Основну структуру моделей, тобто сутностей, та взаємозв’язки між ними можна зобразити за допомогою ER-діаграми, як показано на рисунку 2.6. Таким чином, всі запити до Firestore, які здійснюватимуть завантаження або отримання даних, будуть приводитись до моделей цих сутностей.

Наведені сутності у Firestore фактично являють собою окремі колекції, а атрибути – поля їх документів. Зв’язки відображають взаємодію між сутностями, однак не позначають, які є підколекціями інших. Для детального опису елементів даної діаграми див. 2.3.1 – 2.3.2.

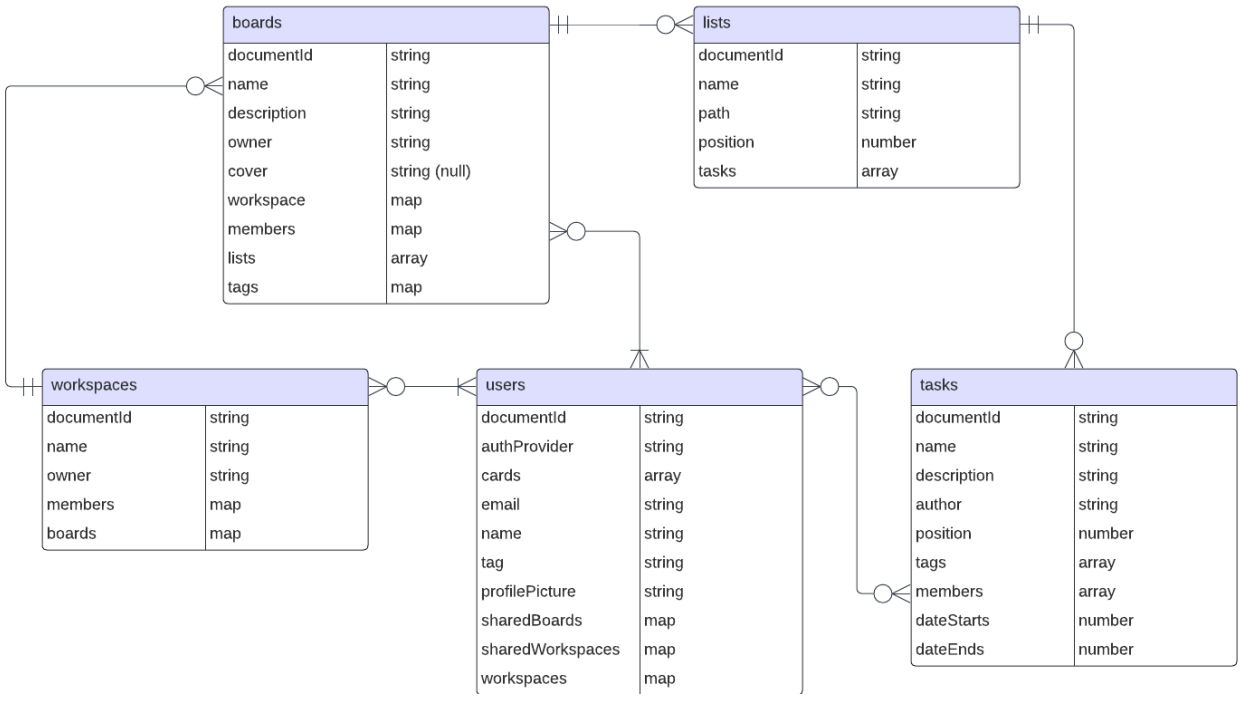


Рисунок 2.6 – Діаграма взаємозв’язків сутностей

2.3.1 Опис сутностей

Сутність «workspaces».

Призначення: дана сутність призначена для зберігання інформації про робочі простори користувачів. Робочий простір є кореневою та центральною колекцією для створення та зберігання Kanban-дошок.

Атрибути:

- owner – ID користувача, який створив робочий простір;
- name – назва робочого простору;
- members – набір пар ключ-значення, де ключ – ID користувача, значення – роль користувача у робочому просторі;
- boards – набір об’єктів по кожній дошці робочого простору. Містить такі поля: name – назва дошки, cover – обкладинка (шлях до зображення у Cloud Storage).

Сутність «users».

Призначення: дана сутність є кореневою колекцією, і призначена для зберігання інформації про користувачів.

Атрибути:

- authProvider – назва провайдеру авторизації;
- cards – масив ID значень задач, призначених користувачу;
- email – поштовий адрес користувача;
- name – ім’я користувача;
- tag – унікальний тег користувача;
- profilePicture – фото профілю користувача (шлях до зображення у Cloud Storage);
- sharedBoards – набір пар ключ-значення, де ключ – ID Kanban-дошки до якої додано користувача, значення – ID робочого простору дошки;
- sharedWorkspaces – набір пар ключ-значення, де ключ – ID робочого простору до якого додано користувача, значення – назва робочого простору;

– workspaces – набір пар ключ-значення, де ключ – ID робочого простору до якого додано користувача, значення – назва робочого простору.

Сутність «boards».

Призначення: дана сутність є підколекцією колекції «workspaces», і призначена для зберігання інформації про Kanban-дошки у робочому просторі.

Атрибути:

– name – назва дошки;
– description – опис дошки;
– owner – ID користувача, який створив дошку;
– cover – обкладинка дошки (шлях до зображення у Cloud Storage);
– workspace – об’єкт, який зберігає інформацію по робочому простору, де створено дошку. Містить такі поля: id – ID робочого простору, name – назва робочого простору;
– members – набір пар ключ-значення, де ключ – ID користувача, значення – роль користувача на дошці;
– lists – масив ID значень списків, створених на дошці;
– tags – набір об’єктів по кожному тегу створеному на дошці. Містить такі поля: color – HEX-кодування кольору тегу, name – назва тегу.

Сутність «lists».

Призначення: дана сутність є підколекцією колекції «boards», і призначена для зберігання інформації про списки задач на Kanban-дошці.

Атрибути:

– name – назва списку задач;
– path – шлях до підколекції;
– position – позиція списку задач на дошці;
– tasks – масив ID значень задач, створених у списку задач.

Сутність «tasks».

Призначення: дана сутність є підколекцією колекції «lists», і призначена для зберігання інформації про задачі в списку задач.

Атрибути:

– name – назва задачі;
– description – опис задачі;
– author – ID користувача, який створив задачу;
– position – позиція задачі в списку задач;
– tags – набір ID значень встановлених тегів;
– members – набір ID значень користувачів, призначених для виконання задачі;
– dateStarts – дата початку виконання задачі (у форматі Unix timestamp);
– dateEnds – дата закінчення виконання задачі (у форматі Unix timestamp).

2.3.2 Опис зв’язків

Зв’язок «workspaces»-«boards».

Тип: один до багатьох.

Опис: у робочому просторі можна створити довільну кількість Kanban-дошок. Дошка може належати тільки робочому простору, звідки її створено;

Зв’язок «workspaces»-«users».

Тип: багато до багатьох.

Опис: робочий простір обов’язково повинен мати власника, тобто користувача, який його створив. До робочого простору може бути додано довільну кількість користувачів, зареєстрованих у системі. Користувачі можуть мати безліч робочих просторів (або бути доданими до них), або жодного.

Зв’язок «boards»-«lists».

Тип: один до багатьох.

Опис: на Kanban-дошці можна створити довільну кількість списків задач. Кожен список задач належить тільки тій дошці, на якій його було створено.

Зв’язок «boards»-«users».

Тип: багато до багатьох.

Опис: кожна Kanban-дошка повинна мати власника, тобто користувача, який її створив. До дошки може бути додано користувачів як зі списку учасників робочого простору, де міститься дошка, так і будь-яких з зареєстрованих у системі. Кожен користувач може створити, або бути доданим до довільної кількості дошок.

Зв’язок «lists»-«tasks».

Тип: один до багатьох.

Опис: у списку задач може бути створено довільну кількість задач, однак кожна задача може знаходитись одночасно лише в одному списку.

Зв’язок «tasks»-«users».

Тип: багато до багатьох.

Опис: задачі може бути призначено довільну кількість виконавців з переліку учасників дошки. Користувач може створювати безліч задач, та бути призначеним на виконання до будь якої з них.

2.4 Моделювання діаграм класів

Однією із основних UML діаграм, використовуваних при проєктуванні програмного забезпечення, є діаграма класів. Як один із типів структурних діаграм, вона описує структуру системи, зображуючи класи даної системи, їх атрибути, операції та взаємозв’язки між об’єктами. Завдяки тому, що діаграми класів надають візуальне високорівневе представлення структури системи, всі зацікавлені сторони можуть оцінити та зрозуміти складність і структури системи. Окрім того, вони дозволяють розробниками та іншим членами команди створити візуальне уявлення про архітектуру системи, її компоненти та взаємодію між ними, які в подальшому можуть допомогти під час реалізації розроблюваної програми, її налагодженні та супроводі. Діаграми класів

задають базову модель системи, яка може використовуватись в інших структурних діаграмах, передбачених UML, таких як діаграми об’єктів, діаграми компонентів або діаграми розгортання.

Змоделюємо діаграми класів для одного з основних прецедентів «Керувати Kanban-дошками» (див. рис. 2.1). Так як обраний варіант використання складається з інших варіантів використання, для повного їх відображення буде створено декілька діаграм залежно від екранів програми на яких вони виконуються.

2.4.1 Діаграма класів для прецеденту створити Kanban-дошку

На діаграмі, наведеній на рисунку 2.7, зображено основні компоненти програми та їх зв’язки задіяні для створення користувачем нової Kanban-дошки.

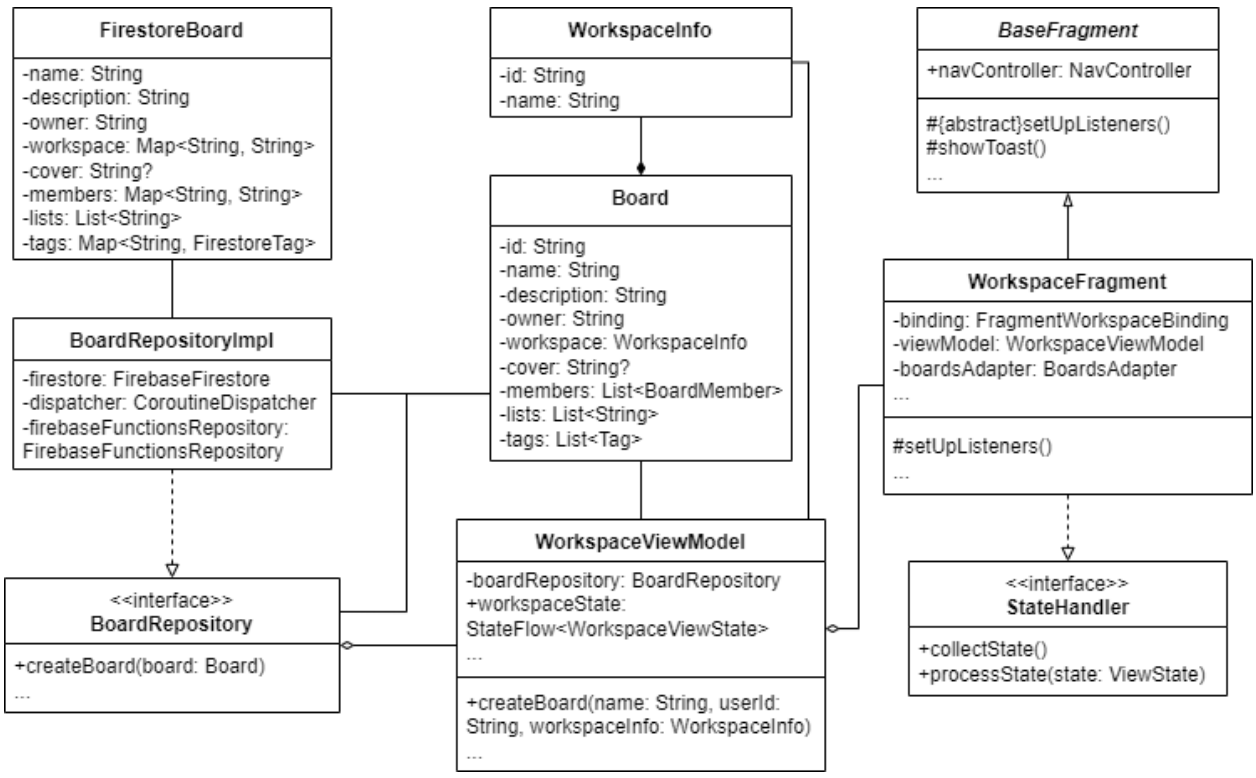


Рисунок 2.7

Для створення дошки користувачу необхідно натиснути відповідну кнопку «Створити дошку» , після чого з’явиться вікно для вводу назви створюваної дошки. Поле має бути обов’язково заповненим. Після введення назви, користувач натискає кнопку «Створити». Нова Kanban-дошка повинна одразу відобразитись у списку дошок на екрані робочого простору.

Опис класів, зображених на діаграмі, наведено нижче.

Клас «BaseFragment».

Опис: абстрактний базовий клас для визначення спільних атрибутів та методів для всіх фрагментів програми.

Батьківський клас: Fragment.

Атрибути:

– navController – об’єкт класу NavController для забезпечення навігації між фрагментами.

Методи:

– setUpListeners – абстрактний захищений метод для встановлення оброблювачів подій до інтерактивних елементів інтерфейсу;
– showToast – захищений метод для створення та відображення спливаючого повідомлення (англ. Toast message).

Клас «WorkspaceFragment».

Опис: визначає та керує користувацьким інтерфейсом екрану робочого простору для відображення існуючих дошок та створення нових.

Батьківський клас: BaseFragment.

Реалізовані інтерфейси: StateHolder.

Атрибути:

– binding – об’єкт класу FragmentWorkspaceBinding для доступу до елементів макету екрану робочого простору;
– viewModel – об’єкт класу WorkspaceViewModel для обробки стану UI та дій користувача;
– boardsAdapter – об’єкт класу BoardsAdapter для управління даними та елементами віджету RecyclerView для відображення дошок.

Методи:

– setUpBoardsAdapter – закритий метод для ініціалізації об’єкту boardsAdapter і його прив’язки до віджету RecyclerView.

Інтерфейс «StateHolder».

Опис: визначає абстрактні методи для збирання та оброблення даних стану користувацького інтерфейсу.

Методи:

– collectState – метод для збирання даних стану;

– processState – метод для оброблення стану, який визначено в аргументі state типу ViewState.

Клас «WorkspaceViewModel».

Опис: відповідає за підготовку та управління даними користувацького інтерфейсу та обробку дій користувача. Фактично діє як міст між UI та рештою програми (наприклад, виклики методів рівня бізнес-логіки або даних), і є компонентом архітектурного шаблону MVVM.

Батьківський клас: ViewModel.

Атрибути:

– boardRepository – об’єкт для виклику методів інтерфейсу BoardRepository;

– workspaceState – об’єкт класу, який реалізує інтерфейс StateFlow, для збереження потоку даних про стан користувацького інтерфейсу екрану робочого простору.

Методи:

– createBoard – метод для створення нової Kanban-дошки. Викликає метод createBoard інтерфейсу BoardRepository.

Інтерфейс «BoardRepository».

Опис: визначає методи для доступу та керування Kanban-дошками у системі.

Методи:

– createBoard – метод для створення Kanban-дошок у робочому просторі.

Клас «BoardRepositoryImpl».

Опис: реалізує інтерфейс BoardRepository для виконання запитів до бази даних Firestore для керування дошками.

Реалізовані інтерфейси: BoardRepository.

Атрибути:

– firebaseFirestore – об’єкт класу FirebaseFirestore, який представляє базу даних Cloud Firestore і є точкою входу для всіх її операцій;
– dispatcher – об’єкт класу CoroutineDispatcher, який визначає контекст для виконання асинхронних операцій з використанням корутин;
– firebaseFunctionsRepository – об’єкт для виклику методів інтерфейсу FirebaseFunctionsRepository.

Клас «Board».

Опис: модель доменного рівня, що представляє сутність Kanban-дошки з такими атрибутами, як назва дошки, опис, власник, учасники тощо.

Клас «WorkspaceInfo».

Опис: модель доменного рівня, для збереження ідентифікатору та назви робочого простору моделі Kanban-дошки.

Клас «FirestoreBoard».

Опис: модель рівня даних, що діє як DTO для передачі даних Kanban-дошки між програмою та базою даних Firestore.

2.4.2 Діаграма класів для прецеденту переглянути Kanban-дошку

На діаграмі, представленій на рисунку 2.8, зображено основні компоненти програми та їх зв’язки для перегляду користувачем обраної Kanban-дошки із списку дошок на екрані робочого простору. На екрані для перегляду дошки відображаються всі списки задач та їх відповідні задачі, якщо їх, звичайно, створено. Відповідно, користувачі можуть взаємодіяти із списками та задачами переміщуючи їх, створюючи нові задачі, або перейменовуючи та видаляючи списки задач.

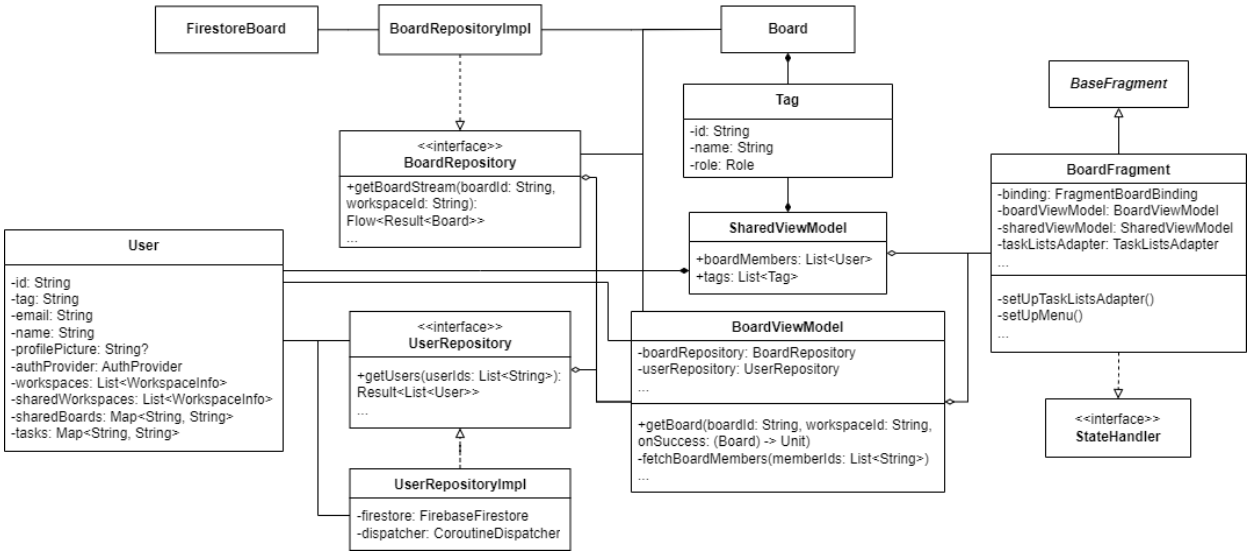


Рисунок 2.8

Для опису класів «BaseFragment», «BoardRepositoryImpl», «Board», «FirestoreBoard» та інтерфейсу «StateHandler» див. 2.4.1.

Опис інших класів, зображених на діаграмі, наведено нижче.

Клас «BoardFragment».

Опис: визначає та керує користувацьким інтерфейсом екрану Kanban-дошки.

Батьківський клас: BaseFragment.

Реалізовані інтерфейси: StateHolder.

Атрибути:

– binding – об’єкт класу FragmentBoardBinding для доступу до елементів макету екрану Kanban-дошки;
– boardViewModel – об’єкт класу BoardViewModel для обробки стану UI та дій користувача;
– sharedViewModel – об’єкт класу SharedViewModel для зберігання даних про користувачів та тегів переглядуваної Kanban-дошки;
– taskListsAdapter – об’єкт класу TaskListsAdapter для управління даними та елементами віджету RecyclerView для відображення списків задач.

Методи:

– setUpTaskListsAdapter – закритий метод для ініціалізації об’єкту

taskListsAdapter і його прив’язки до віджету RecyclerView;

– setUpMenu – закритий метод для налаштування меню на панелі керування для переходу до екрану налаштувань дошки.

Клас «BoardViewModel».

Опис: відповідає за підготовку та управління даними для користувацького інтерфейсу, обробку дій користувача.

Батьківський клас: ViewModel.

Атрибути:

– boardRepository – об’єкт для виклику методів інтерфейсу BoardRepository;

– userRepository – об’єкт для виклику методів інтерфейсу UserRepository;

Методи:

– getBoard – метод для отримання даних Kanban-дошки з потоку даних Flow. Викликає метод getBoardsStream інтерфейсу BoardRepository;

– fetchBoardMembers – закритий метод для отримання учасників Kanban-дошки. Викликає метод getUsers інтерфейсу UserRepository.

Клас «SharedViewModel».

Опис: відповідає за зебреження даних про користувачів та теги обраної Kanban-дошки.

Батьківський клас: ViewModel.

Атрибути:

– boardMembers – список учасників Kanban-дошки;

– tags – список створених тегів у Kanban-дошці.

Інтерфейс «BoardRepository».

Опис: визначає методи для доступу та керування Kanban-дошками у системі.

Методи:

– getBoardStream – метод для отримання потоку даних (Flow) Kanban-дошки.

Інтерфейс «UserRepository».

Опис: визначає методи для доступу та керування даними користувачів системі.

Методи:

– getUsers – метод для отримання користувачів для переданого, у якості параметра, списку ідентифікаторів користувачів.

Клас «UserRepositoryImpl».

Опис: реалізує інтерфейс «UserRepository» для виконання запитів до бази даних Firestore для керування даними користувачів.

Реалізовані інтерфейси: UserRepository.

Атрибути:

– firebaseFirestore – об’єкт класу FirebaseFirestore, який представляє базу даних Cloud Firestore і є точкою входу для всіх її операцій;
– dispatcher – об’єкт класу CoroutineDispatcher, який визначає контекст для виконання асинхронних операцій з використанням корутин.

Клас «User».

Опис: модель доменного рівня, що представляє сутність користувача з такими атрибутами, як ім’я користувача, адреса електронної пошти, тег користувача тощо.

Клас «Tag».

Опис: модель доменного рівня, що представляє сутність тегу (візуальний елемент для категоризації задач на дошці) з такими атрибутами, як назва та ідентифікатор встановленого кольору тегу.

2.4.3 Діаграма класів для прецедентів оновити, видалити Kanban-дошку, керувати її учасниками

На діаграмі класів, змальованій на рисунку 2.9, зображено основні компоненти програми та їх зв’язки для зміни налаштувань, редагування та керування учасниками Kanban-дошки. Для переходу на екран налаштувань

користувач має натиснути на відповідний елемент меню на панелі управління з екрану перегляду дошки.

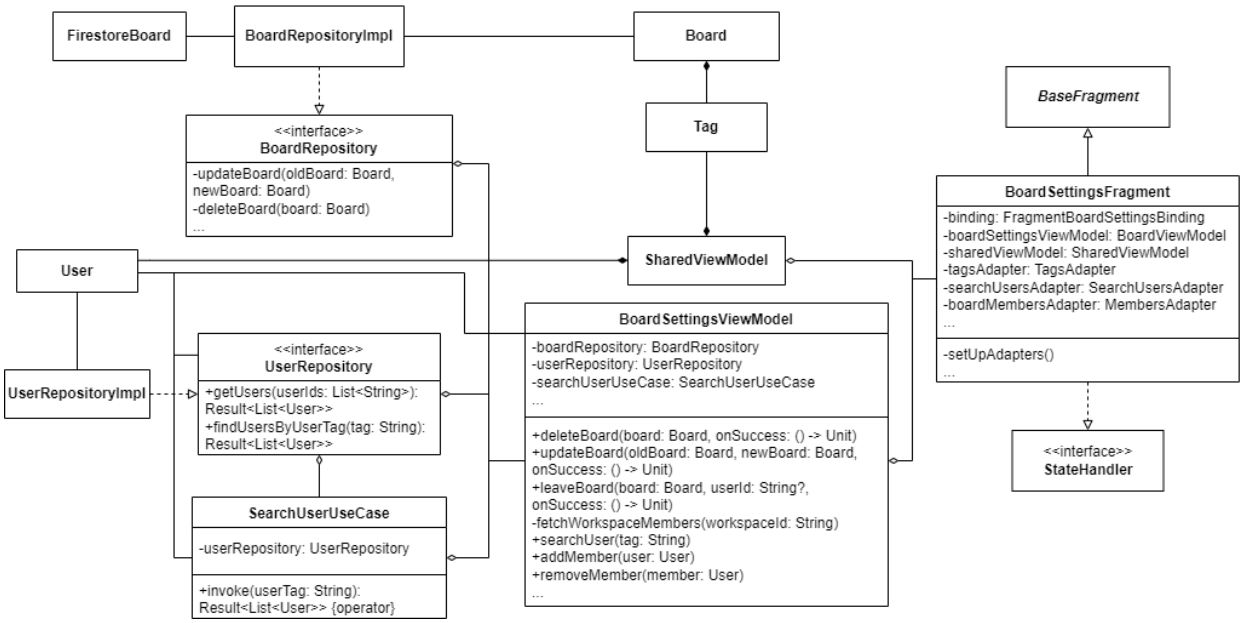


Рисунок 2.9

На екрані налаштувань обраної Kanban-дошки користувач може змінювати її назву, опис, керувати учасниками та тегами, видалити або покинути дошку. Додавання учасників до дошки здійснюється шляхом вибору користувачів із списку учасників робочого простору, або за допомогою пошуку користувачів у системі за їх тегами. Видаляти дошку можуть тільки адміністратори, або власник дошки. Покинути Kanban-дошку може будь-який користувач, окрім її власника.

Для опису класів «BaseFragment», «BoardRepositoryImpl», «Board», «FirestoreBoard» та інтерфейсу «StateHandler» див. 2.4.1.

Для опису класів «SharedViewModel», «UserRepositoryImpl», «User», «Tag» див. 2.4.2.

Опис інших класів, зображених на діаграмі, наведено нижче.

Клас «BoardSettingsFragment».

Опис: визначає та керує користувацьким інтерфейсом екрану налаштувань Kanban-дошки.

Батьківський клас: BaseFragment.

Реалізовані інтерфейси: StateHolder.

Атрибути:

- binding – об’єкт класу FragmentBoardSettingsBinding для доступу до елементів макету екрану налаштувань Kanban-дошки;
- boardSettingsViewModel – об’єкт класу BoardSettingsViewModel для обробки стану UI та дій користувача;
- sharedViewModel – об’єкт класу SharedViewModel для зберігання даних про користувачів та тегів переглядуваної Kanban-дошки;
- tagsAdapter – об’єкт класу TagsAdapter для управління даними та елементами віджету RecyclerView для відображення списку тегів;
- searchUsersAdapter – об’єкт класу SearchUsersAdapter для управління даними та елементами віджету RecyclerView для відображення списку знайдених користувачів;
- membersAdapter – об’єкт класу MembersAdapter для управління даними та елементами віджету RecyclerView для відображення списку учасників дошки.

Методи:

- setUpTagsAdapter – закритий метод для ініціалізації об’єктів tagsAdapter, searchUsersAdapter, membersAdapter і їх прив’язок до віджетів RecyclerView.

Клас «BoardSettingsViewModel».

Опис: відповідає за підготовку та управління даними для користувацького інтерфейсу та обробки дій користувача.

Батьківський клас: ViewModel.

Атрибути:

- boardRepository – об’єкт класу, який реалізує інтерфейс BoardRepository для доступу та керування Kanban-дошками;
- userRepository – об’єкт класу, який реалізує інтерфейс UserRepository для керуванням користувачами системи та їх даними;

– searchUserUseCase – об’єкт класу SearchUserUseCase для пошуку користувачів у системі.

Методи:

– deleteBoard – метод для видалення Kanban-дошки. Викликає метод deleteBoard інтерфейсу BoardRepository;
– updateBoard – метод для оновлення даних Kanban-дошки. Викликає метод updateBoard інтерфейсу BoardRepository;
– leaveBoard – метод для видалення користувача із списку учасників Kanban-дошки, який бажає її покинути. Викликає метод updateBoard інтерфейсу BoardRepository;
– fetchWorkspaceMembers – закритий метод для отримання учасників робочого простору. Викликає метод getUsers інтерфейсу UserRepository;
– searchUser – метод для пошуку користувачів у системі за тегом користувача. Викликає метод invoke класу SearchUserUseCase;
– addMember – метод для додавання користувача до списку учасників Kanban-дошки;
– removeMember – метод для видалення користувача із списку учасників Kanban-дошки.

Інтерфейс «BoardRepository».

Опис: визначає методи для доступу та керування Kanban-дошками у системі.

Методи:

– updateBoard – метод для оновлення даних Kanban-дошки;
– deleteBoard – метод для видалення Kanban-дошки.

Інтерфейс «UserRepository».

Опис: визначає методи для доступу та керування даними користувачів системі.

Методи:

– getUsers – метод для отримання користувачів для переданого, у якості параметра, списку ідентифікаторів користувачів;

– findUsersByUserTag – метод для пошуку користувачів у системі за тегом користувача.

Клас «SearchUserUseCase».

Опис: варіант використання для пошуку користувачів у системі.

Атрибути:

– userRepository – об’єкт для виклику методів інтерфейсу UserRepository.

Методи:

– invoke – викликає метод findUsersByUserTag інтерфейсу UserRepository.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Приклади реалізації класів системи

Одним із основних варіантів використання розроблюваного Android-застосунку для управління проєктами є створення задач на Kanban-дошках. Клас CreateTaskFragment (див. рис. 3.1) відповідає за відображення даних на користувацькому інтерфейсі та обробку дій користувача для створення задачі. Він розширює функціональність класу TaskEditorFragment, який визначає слухачів для всіх полів, для заповнення даними задачі (ім’я, опис, дата початку/закінчення тощо), і встановлює оброблювач подій для кнопки «Create task».

```
@AndroidEntryPoint
class CreateTaskFragment : TaskEditorFragment(){
    private val args: CreateTaskFragmentArgs by navArgs()
    override val taskList: TaskList by lazy { args.taskList }
    override val task: Task? by lazy { args.task }
    override val viewModel: CreateTaskViewModel by viewModels()

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        setUpActionBar(binding.toolbar, "Create task")
    }

    override fun setUpListeners() {
        super.setUpListeners()
        binding.btnCreateTask.text = getString(R.string.create_task)
        binding.btnCreateTask.setOnClickListener {
            viewModel.createTask(task = createTask(), taskList) { navController.popBackStack() }
        }
    }
}
```

Рисунок 3.1

Оброблювач подій кнопки «Create task» викликає метод createTask класу CreateTaskViewModel, відповідального за обробку дій користувача та взаємодії з моделлю даних програми. Код даного класу наведено на рисунку 3.2. Метод createTask приймає у якості параметрів створювану задачу моделі «Task», екземпляр класу моделі «TaskList» для оновлення списку задач, і лямбда-функцію, яка викликатиметься в разі успішного виконання запиту. Якщо при виконанні запиту виникла помилка, на екрані зображено відповідне повідомлення. За виконання запиту для створення задачі відповідає метод createTask інтерфейсу TaskRepository.

```
@HiltViewModel
class CreateTaskViewModel @Inject constructor(
    private val taskRepository: TaskRepository,
    upsertTagUseCase: UpsertTagUseCase,
) : TaskEditorViewModel(upsertTagUseCase) {
    fun createTask(task: Task, taskList: TaskList, onSuccessCallback: () -> Unit) =
viewModelScope.launch {
        _isSavingTask.value = true
        val result = taskRepository.createTask(
            task = task.copy(position = taskList.tasks.size.toLong()),
            taskListId = taskList.id,
            taskListPath = taskList.path
        )
        when (result) {
            is Result.Success -> {
                _isSavingTask.value = false
                onSuccessCallback()
            }
            is Result.Error -> {
                _isSavingTask.value = false
                _message.value = result.message
            }
        }
    }
}
```

Рисунок 3.2

Інтерфейс TaskRepository використовується для абстракції основних методів для керування задачами в системі і джерелом даних програми. Фактично він відокремлює логіку доступу до даних від решти програми. Клас TaksRepositoryImpl відповідає за реалізацію даного інтерфейсу, і взаємодію з сервісом Cloud Firestore. Він слугує за формування та виконання запитів до бази даних для створення, видалення, оновлення та отримання задач. Так, приклад реалізації методу createTask для створення задачі, який викликається в класі CreateTaskViewModel наведено на рисунку 3.3. Метод використовує withContext для зміни контексту корутини на виконання у потоці вводу-виводу для виконання потенційно довготривалої операції створення завдання у Firestore. Для створення ідентифікатору задачі використовується метод randomUUID класу UUID. Метод створює запит до Firestore для додавання задачі у список задач за ідентифікатором taskListId, який розташовується за шляхом, визначеним у параметрі taskListPath. Метод addAssignedTaskToUsers викликається в разі успішного виконання запиту, і додає створену задачу до користувачів, які були призначені для її виконання.

```
override suspend fun createTask(
    task: Task,
    taskListId: String,
    taskListPath: String
): Result<Unit> = runCatching {
    withContext(dispatcher) {
        val taskId = UUID.randomUUID().toString()
        firestore.collection(taskListPath)
            .document(taskListId)
            .update("${FirestoreCollection.TASKS}.${taskId}", task.toFirestoreTask())
            .getResult {
                // add the task to assigned users
                addAssignedTaskToUsers(taskId, "$taskListPath/$taskListId", task.members)
            }
    }
}
```

Рисунок 3.3

3.2 Тестування застосунку

Тестування є важливим етапом циклу розробки програмного забезпечення. Воно гарантує, що створюваний застосунок функціонує відповідно до очікуваних вимог, стабільно працює за будь-яких умов і забезпечує належний досвід використання. Існують різні типи тестування, такі як модульне, інтеграційне, тестування користувацького інтерфейсу тощо.

3.2.1 Тестування взаємодії з базою даних Firestore

За виконання запитів до бази даних Cloud Firestore для створення, отримання, оновлення та видалення даних у розроблюваному застосунку відповідають репозиторії. Діючи як посередники між джерелами даних програми і рештою програми, вони забезпечують зручний інтерфейс для доступу до даних і керування ними. Firebase надає набір локальних емуляторів, який називається Firebase Local Emulator Suite, що дозволяє розробляти та тестувати сервіси Firebase у контрольованому та ізольованому середовищі. Для тестування запитів до бази даних Firestore буде використано відповідний емулятор даного сервісу з набору доступних. Після встановлення та налаштування Local Emulator Suite, для запуску емулятора необхідно виконати команду, як наведено на рисунку 3.4.

```
PS E:\firebase-test> firebase.cmd emulators:start --only firestore
```

Рисунок 3.4

Взаємодія з програмою відбувається з використанням графічного інтерфейсу, де можна передивитись всі запущені емулятори по кожному сервісу, і дані, які кожен з них містить (див. рис. 3.5). Також необхідним кроком є встановлення емулятора Android пристрою, і встановлення на ньому

розроблюваного застосунку, який потім можна під’єднати до емулятору сервісу Firestore.

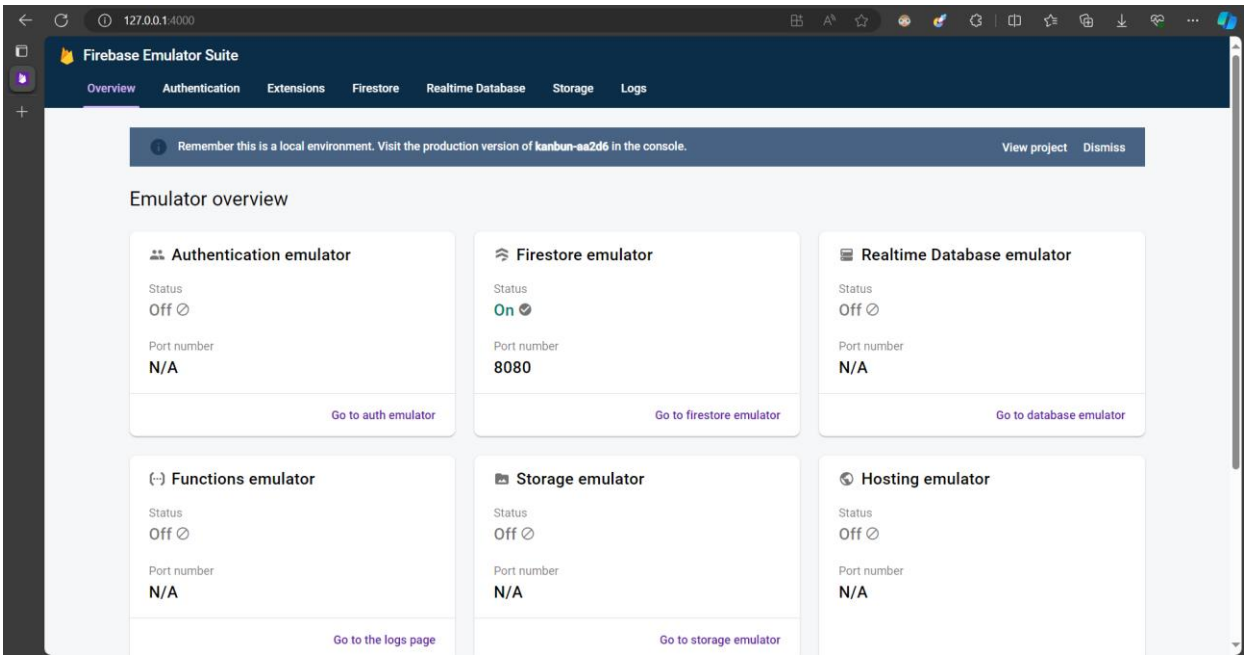


Рисунок 3.5 – Список запущених емуляторів у Local Emulator Suite

Останнім кроком є налаштування об’єкта класу FirebaseFirestore для використання емулятора, який екземпляр репозиторію зможе використовувати для виконання всіх запитів до бази даних (див. рис. 3.6).

```
private const val host = "10.0.2.2"
private const val firestorePort = 8080
val firestore = Firebase.firestore.apply {
    useEmulator(host, firestorePort)
    firestoreSettings = firestoreSettings {
    isPersistenceEnabled = false
  }
}
```

Рисунок 3.6 – Приклад використання емулятору Firestore

Протестуємо запит на додавання даних до бази даних на прикладі створення Kanban-дошки у робочому просторі. На рисунку 3.7 наведено

приклад тестування методу createBoard інтерфейсу BoardRepository, який виконує запит на створення нового документа у колекції «boards» з даними дошки, переданої як аргумент методу. Варто зазначити, що в реалізації наведеного методу ідентифікатор документу генерується сервісом Firestore автоматично. При створені дошки у програмі мають бути вказані її власник, тобто користувач, який створює дошку, та інформація про робочий простір (ідентифікатор та ім’я), в якому створюється дошка.

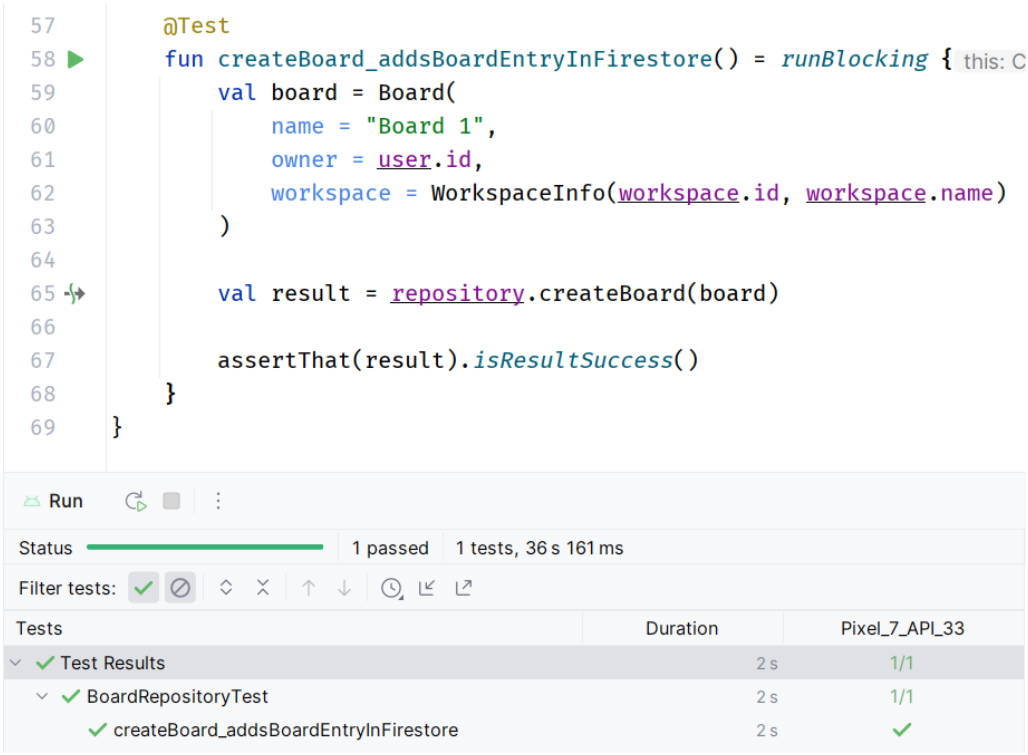


Рисунок 3.7 – Тестування запиту для створення Kanban-дошки

У разі успішного виконання запиту, метод повинен повернути об’єкт класу Result.Success, де Result – це клас обгортка для запитів до бази даних, мережевих запитів тощо, а також створити новий документ в базі даних із атрибутами параметру дошки. В емуляторі Firestore (див. рис. 3.8) можна перевірити стан колекцій бази даних після виконання тесту. В даному випадку, у підколекцію «boards» документу робочого простору має бути додано новий документ із даними створеної дошки.

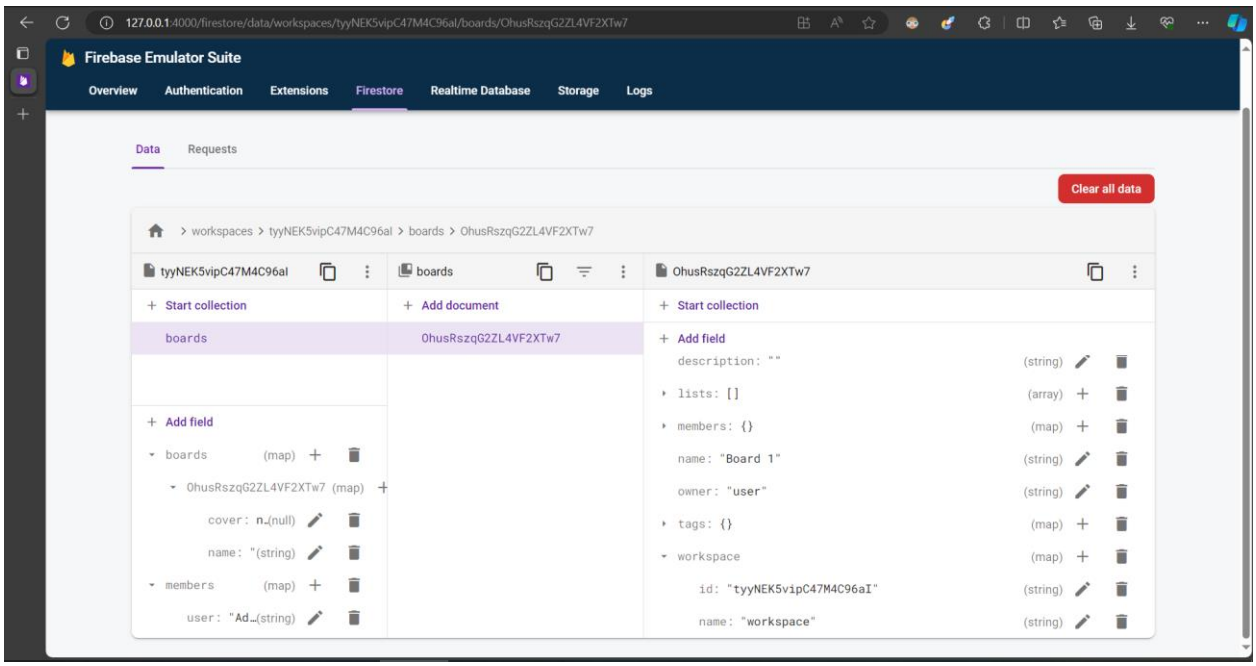


Рисунок 3.8 – Нова Kanban-дошка у Firestore

Таким чином виконується тестування всіх інших методів та репозиторіїв для управління даними в базі даних Cloud Firestore. Використання локального емулятора забезпечує швидке виконання запитів, і спрощує перевірку роботи застосунку шляхом використання окремого середовища для тестування. Такий підхід дозволяє швидко виявляти та вирішувати можливі проблеми, не порушуючи цілісність даних у робочому середовищі.

3.2.2 Тестування варіантів використання (Use Cases).

Варіанти використання відповідають за бізнес-логіку, або дії, які можуть бути виконані в системі, і являються компонентами архітектурного шаблону CleanArchitecture. У розроблюваному застосунку до таких операцій можна віднести створення авторизація користувача, створення Kanbun-дошки, створення задачі тощо. Однією із функцій системи є додавання користувачів до робочого простору, дошки або задачі. Для забезпечення виконання даної операції використовується пошук користувачів у системі, за який відповідає клас «SearchUserUseCase».

Пошук користувачів відбувається за полем «tag» моделі «User», і працює за принципом пошуку за префіксом. Тобто, система буде шукати теги користувачів, які починаються з тексту, який передається в якості аргументу функції. Приклад тестування класу SearchUserUseCase наведено на рисунку 3.9.

```
35      @Test
36      fun searchUser_returnsUsers_withMatchingTags(): Unit = runBlocking { this: CoroutineScope
37          val user1 = User(id = "user1", name = "User 1", tag = "user_test1")
38          val user2 = User(id = "user2", name = "User 2", tag = "userTest2")
39          val user3 = User(id = "user3", name = "User 3", tag = "test3user")
40          // upload users in the Firestore "users" collection
41          createUsers(user1, user2, user3)
42
43          val tag = "user"
44          val expectedUsers = listOf(user1, user2)
45
46          val result = searchUserUseCase(tag)
47
48          assertThat(result).isResultSuccess()
49          assertThat((result as Result.Success).data).containsExactlyElementsIn(expectedUsers)
50      }
```

Run

Status — 1 passed — 1 tests, 30 s 721 ms

Filter tests:

Tests	Duration	Pixel_7_API_33
Test Results	956 ms	1/1
SearchUserUseCaseTest	956 ms	1/1
searchUser_returnsUsersWithMatchingTags	956 ms	✓

Рисунок 3.9 – Тестування пошуку користувачів

SearchUserUseCase повертає об’єкт класу Result.Success зі списком користувачів, які відповідають шуканому тегу, або пустий список, якщо не було знайдено жодної відповідності. В разі виникнення помилки, повертається об’єкт класу Result.Error із повідомленням цієї помилки і об’єктом винятку (англ. Exception).

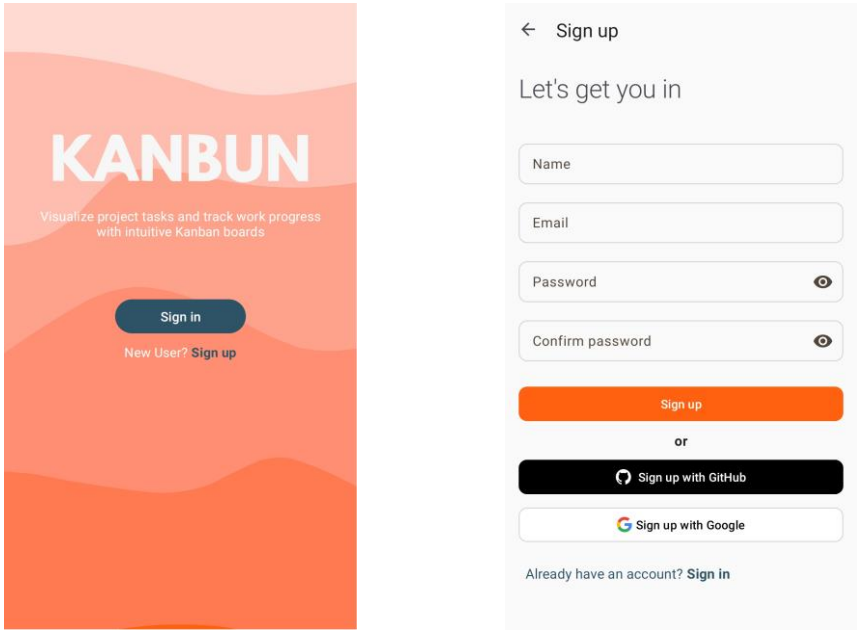
Тестування варіантів використання забезпечує, що бізнес-логіка програми функціонує згідно встановлених вимог. Воно також дозволяє усунути можливі помилки, перевірити цілісність системи та гарантує, що система поводитиметься очікувано для всіх можливих вхідних даних та дій користувача.

3.3 Приклади роботи застосунку

Розроблений застосунок підтримує смартфони, що працюють під управлінням операційної системи Android з версією не нижче 8.0 Oreo. Для реєстрації та авторизації у системі потрібне актине Інтернет з’єднання. Хоча програма підтримує офлайн доступ, для синхронізації даних із сервером та більш ефективної командної роботи рекомендується постійне під’єднання до мережі.

3.3.1 Вхід до системи

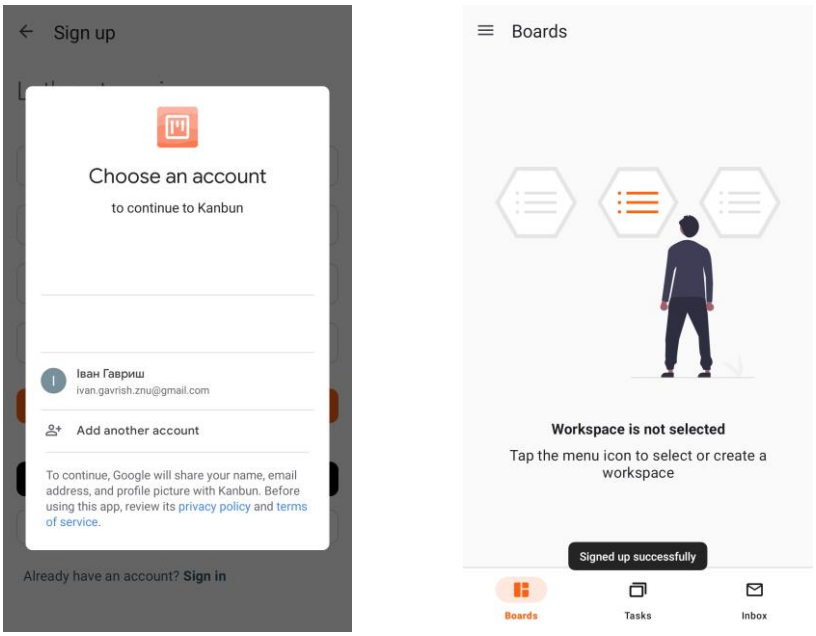
При першому запуску застосунку користувач опиняється на екрані автентифікації, з якого він може перейти до екрану реєстрації, або входу в існуючий обліковий запис (рис. 3.10, а). Програма надає можливість створення акаунту з використанням сервісів Google, GitHub, або за поштою і паролем (рис. 3.10, б).



а) Екран автентифікації б) Екран реєстрації

Рисунок 3.10 – Інтерфейс автентифікації в системі

Наприклад, створимо обліковий запис з використанням існуючого акаунту Google (рис. 3.11, а). Після натискання кнопки «Sign up with Google» з’являється вікно вибору акаунту. Обираємо один із запропонованих. В разі успішної реєстрації, користувача буде перенаправлено на домашній екран застосунку і виведено повідомлення про успішну реєстрацію (рис. 3.11, б).



а) Вибір Google-акаунту б) Успішна реєстрація

Рисунок 3.11 – Реєстрація в системі

Якщо під час реєстрації виникла помилка, наприклад, було відсутнє Інтернет з’єднання, то буде відображено відповідне повідомлення, і користувач залишиться на екрані створення облікового запису (див. рис. 3.12).

3.3.2 Створення Kanban-дошки

На домашньому екрані застосунку відображається список існуючих дошок в обраному робочому просторі, а також міститься кнопка «Create board» для додавання нових.

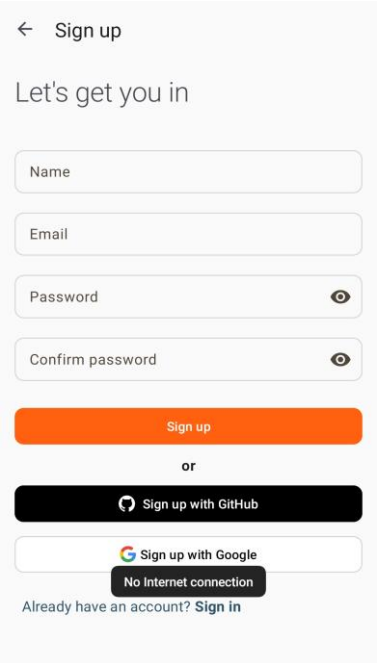
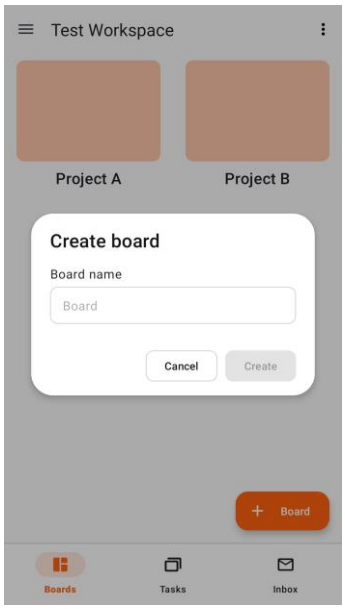
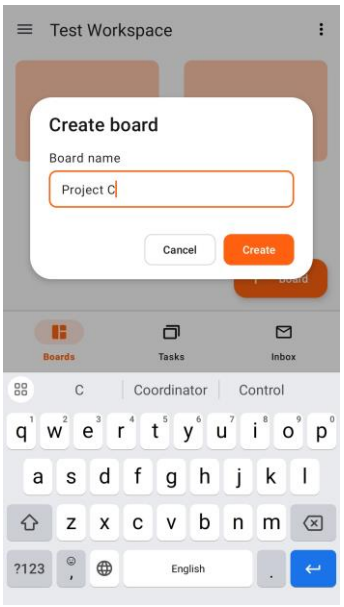


Рисунок 3.12 – Повідомлення про відсутнє Інтернет-з’єднання

При натисканні на кнопку «Create board» з’являється діалогове вікно створення Kanban-дошки із текстовим полем для введення назви дошки, кнопками «Create» та «Cancel» (див. рис. 3.13, а). За замовчування кнопка «Create» неактивна, і змінює стан, якщо введено щонайменше 1 символ (див. рис. 3.13, б).



а) Початковий стан



б) Введення назви дошки

Рисунок 3.13 – Інтерфейс діалогу створення Kanban-дошки

Після введення назви дошки і натискання кнопки «Create» виконується створення дошки, яка повинна одразу з’явитись у списку дошок робочого простору (див. рис. 3.14).

Рисунок 3.14 – Повідомлення про успішне створення Kanban-дошки

3.3.3 Керування задачами на Kanban-дошці

Кожна Kanban-дошка складається зі списків задач (стовпці), до яких користувачі можуть додавати задачі та переміщувати їх між стовпцями. Наприклад, дошка може складатись з таких списків задач, як «To Do», «Doing», «Done», які використовуються для відображення різних статусів виконання задач учасниками дошки. Так, задачі, які потрібно виконати, спочатку додаються до стовпця «To Do», і по мірі їх виконання переміщуються або до стовпця «Doing», або «Done», якщо задачу виконано (див. рис. 3.15).

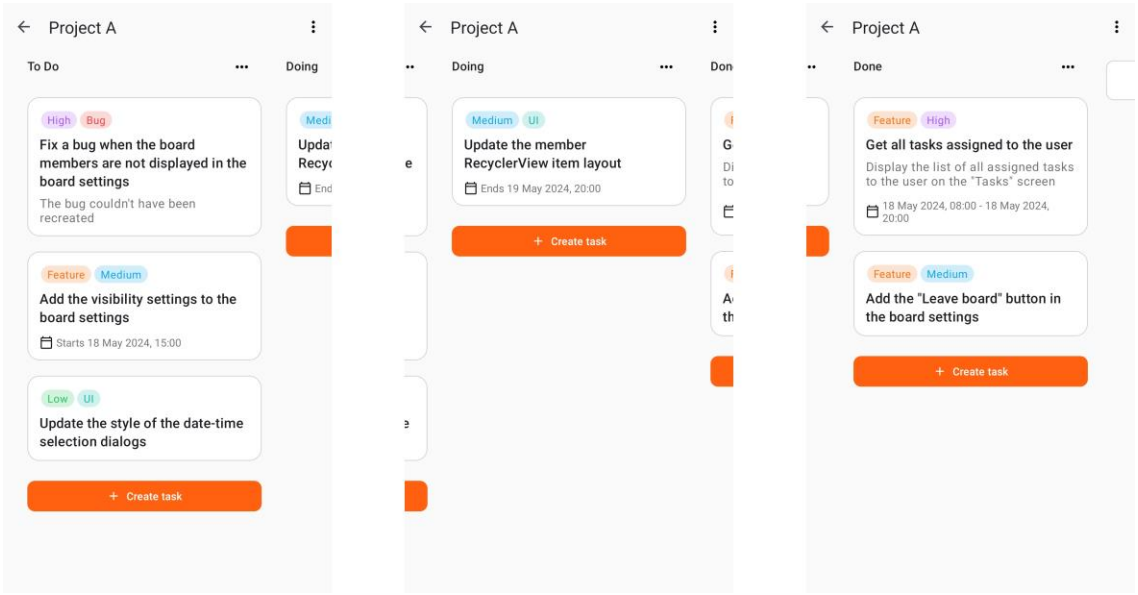
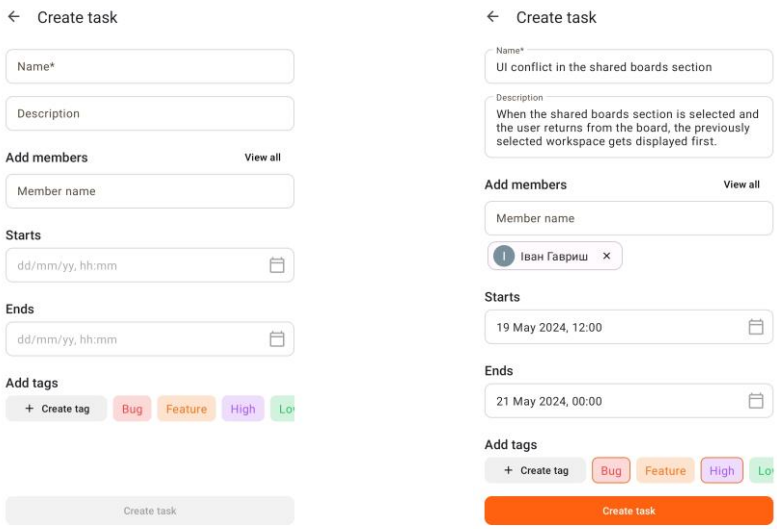


Рисунок 3.15 – Списки задач на Kanban-дошці

Для додавання нової задачі до списку задач необхідно натиснути кнопку «Create task», в результаті чого користувача буде перенаправлено на екран створення задачі (див. рис. 3.16, а). При створенні задачі користувач може вказати її назву, опис, дату початку та/або закінчення, призначити виконавців та встановити теги. Обов’язковим полем для заповнення є «Name». Після заповнення всіх необхідних полів, потрібно натиснути кнопку «Create task» (див. рис. 3.16, б).



а) Стан за замовчуванням б) Заповнення полів

Рисунок 3.16 – Інтерфейс екрану створення задачі

Створена задача повинна одразу з’явитись у списку задач, з якого починалась дія (див. рис. 3.17).

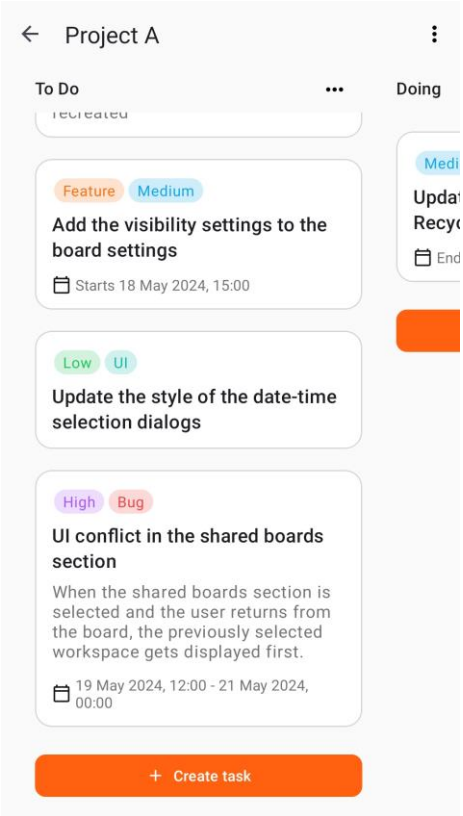
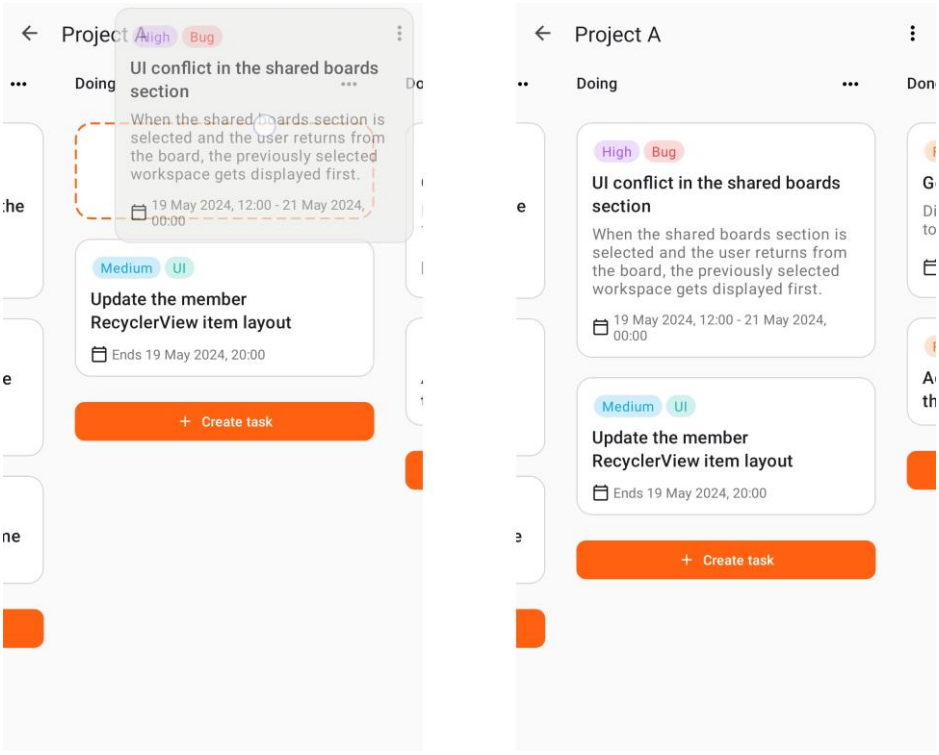


Рисунок 3.17 – Створена задача додана до списку задач

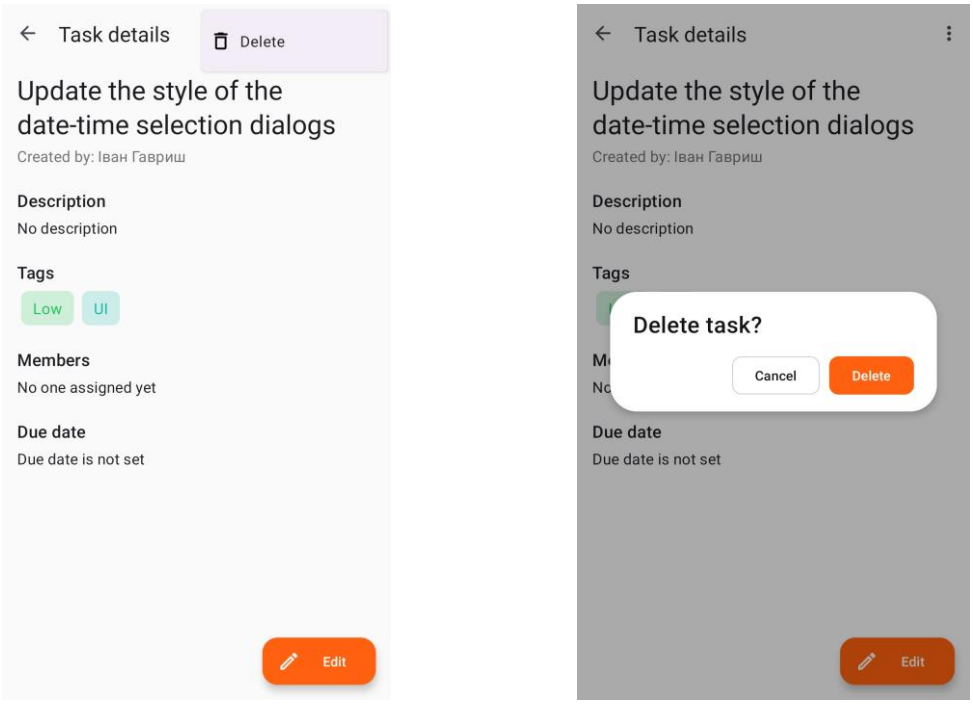
Для переміщення задач необхідно здійснити довге натискання на картку задачі, і перетягнути її на іншу позицію в бажаному списку задач. Під час перетягування, на місці, куди буде переміщено задачу, відображається область скидання (див. рис. 3.18, а). Для завершення дії потрібно перетягнути картку задачі в область скидання і відпустити. Після переміщення, задачі в межах списку буде перевпорядковано (див. рис. 3.18, б).

Для видалення задачі на екрані інформації про задачу використовується опція меню на панелі навігації «Delete» (див. рис. 3.19, а). Після натискання кнопки з’являється діалогове вікно підтвердження дії, на якому користувач потрібен натиснути кнопку «Delete» для продовження (див. рис. 3.19, б). Після видалення, користувача буде перенаправлено на екран Kanban-дошки, а всі інші задачі зі списку буде перевпорядковано.



а) Початок переміщення б) Результат переміщення

Рисунок 3.18 – Переміщення задачі у інший список задач



а) Опція меню «Delete» б) Діалог підтвердження дії

Рисунок 3.19 – Видалення задачі

ВИСНОВКИ

Управління проєктами відіграє важливу роль у життєвому циклі проєкту. Ефективне керування проєктами допомагає досягати поставлених цілей, оптимізувати роботу та взаємодію працівників, дотримуватися встановлених термінів, бюджету та вимог до якості. Використання сучасних методологій та інструментів, таких як системи управління проєктами, допомагає зробити процес проєктного менеджменту більш продуктивним і зручним для всіх зацікавлених сторін.

Метою кваліфікаційної роботи є розробка Android-застосунку для управління проєктами засобами мови програмування Kotlin. Для її досягнення було визначено перелік задач, хід виконання яких було описано у тексті даної роботи. Розглянемо детальніше опис реалізації цих задач по кожному розділу.

Так, у першому розділі було проаналізовано предметну область розроблюваного застосунку, а саме: сферу управління проєктами, описано та порівняно методології, і програмне забезпечення (системи керування проєктами), які можуть використовуватись в процесі менеджменту. Крім того, було визначено основні загальні та функціональні вимоги до розроблюваного застосунку, а також охарактеризовано обраний інструментарій, використовуваний для реалізації програми.

У другому розділі було спроєктовано модель бази даних системи, описано її основні сутності та зв’язки між ними. Також було визначено архітектурні шаблони, використовувані в процесі розробки застосунку, змальовано та описано основні варіанти використання системи різними акторами, наведено діаграми одних з основних класів програми.

У третьому розділі було наведено приклади тестування та роботи основного функціоналу застосунку та його компонентів.

Таким чином, в результаті виконання всіх поставлених задач було досягнуто мети даної кваліфікаційної роботи.

ПЕРЕЛІК ПОСИЛАНЬ

1. What is project management? APM | Chartered Membership Organisation. URL: https://www.apm.org.uk/resources/what-is-project-management/ (дата звернення: 27.04.2024).
2. Project Management Institute. A guide to the project management body of knowledge (PMBOK® guide) : 5th ed. Newtown Square, Pennsylvania : Project Management Institute, 2013. 589 p.
3. Understanding the objectives of project management in 2024. *TimesPro*. URL: https://timespro.com/blog/what-are-the-objectives-of-project-management#what-are-the-objectives-of-project-management (дата звернення: 27.04.2024).
4. Sheffield J., Lemétayer J. Critical success factors in project management methodology fit. *PMI® Global Congress 2010* : Conference paper, Melbourne, 24 February 2010. Newtown Square, 2010.
5. 16 project management methodologies – choose the right one for optimised workflow. Institute of Project Management. URL: https://instituteprojectmanagement.com/blog/project-management-methodologies/ (дата звернення: 28.04.2024).
6. Project Management Methodologies: 12 Best Frameworks. *Asana*. URL: https://asana.com/resources/project-management-methodologies (дата звернення: 28.04.2024).
7. 15 переваг використання методології управління проєктами. *Worksection*. URL: https://worksection.com/ua/blog/benefits-of-project-management-methodology.html (дата звернення: 28.04.2024).
8. Який таск-менеджер вибрати для бізнесу, себе та команди у 2024 році – огляд. *IAMPM*. URL: https://iampm.club/ua/blog/yakij-task-menedzher-obrati-u-2022-roczi/ (дата звернення: 28.04.2024).
9. Kliem R. L. Project management software friend or foe? *PM Network*. 2000.

Vol. 14, no. 7. P. 76–78.

10. Jira | Issue & Project Tracking Software | Atlassian. Collaboration software for software, IT and business teams. URL: https://www.atlassian.com/software/jira (дата звернення: 28.04.2024).
11. Manage your team’s work, projects, & tasks online. *Asana*. URL: https://asana.com/ (дата звернення: 28.04.2024).
12. Manage Your Team’s Projects From Anywhere. *Trello*. URL: https://trello.com/home (дата звернення: 29.04.2024).
13. A new way of working. *Monday.com*. URL: https://monday.com/ (дата звернення: 29.04.2024).
14. DIESEL.app. Kanban Board. *Google Play*. URL: https://play.google.com/store/apps/details?id=app.diesel.kanban_board (дата звернення: 29.04.2024).
15. Kotlin coroutines on Android. *Android Developers*. URL: https://developer.android.com/kotlin/coroutines (дата звернення: 01.05.2024).
16. Фрунза В. Firebase як бекенд для будь-яких застосунків, та як використовувати Firebase-сервіси. *DOU*. URL: https://dou.ua/forums/topic/44058/ (дата звернення: 01.05.2024).
17. What is Software Architecture in Software Engineering? IEEE Computer Society. URL: https://www.computer.org/resources/software-architecture (дата звернення: 04.05.2024).
18. Guide to app architecture. *Android Developers*. URL: https://developer.android.com/topic/architecture (дата звернення: 04.05.2024).
19. Kušt I. Clean Architecture Tutorial for Android: Getting Started. *Kodeco*. URL: https://www.kodeco.com/3595916-clean-architecture-tutorial-for-android-getting-started (дата звернення: 05.05.2024).
20. Clean architecture explored. *Codemagic blog*. URL: https://blog.codemagic.io/clean-architecture-explored/ (дата звернення: 05.05.2024).
21. Brandi D. The “Real” Clean Architecture in Android: S.O.L.I.D. *Medium*. URL:

https://betterprogramming.pub/the-real-clean-architecture-in-android-part-1-s-o-l-i-d-6a661b103451 (дата звернення: 06.05.2024).

22. Choose a Database: Cloud Firestore or Realtime Database. *Firebase*. URL: https://firebase.google.com/docs/database/rtdb-vs-firestore (дата звернення: 07.05.2024).

ДОДАТОК А

Посилання на GitHub-репозиторій проєкту

Посилання на GitHub-репозиторій з вихідним кодом проєкту: https://github.com/givtszs/kanbun.

Рисунок А.1 – QR-код з посиланням на GitHub-репозиторій проєкту