Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ КООРДИНАЦІЇ ЗАВДАНЬ У КОМАНДНИХ ПРОЄКТАХ З ВИКОРИСТАННЯМ GRADLE KOTLIN DSL

DEVELOPMENT OF AN AUTOMATED TASK COORDINATION SYSTEM FOR TEAM PROJECTS USING GRADLE KOTLIN DSL

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Мартинчук Ярослав Олександрович

Керівник:
Повстяна Юлія Славомирівна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«*20*» *12* 20*24* р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Мартинчуку Ярославу Олександровичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *«Розробка автоматизованої системи координації завдань у командних проєктах з використанням Gradle Kotlin DSL».*
Керівник роботи: *доцент Повстяна Юлія Славомирівна.*
затверджені наказом закладу вищої освіти від *«19» грудня 2024 р.* № *474/01-02.*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра *«10» червня 2025 р.*
3. Вихідні дані до роботи: *Kotlin, Android Studio, Gradle Kotlin DSL, Jetpack Compose, ViewModel, LiveData/StateFlow, Room, Firebase, SharedPreferences, Git/GitHub; аналіз предметної області, формування вимог і ролей, архітектура з офлайн-синхронізацією, модульна розробка з UX/UI, тестування, документування.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз предметної області; вимоги до системи; архітектура з ролями та офлайн-режимом; реалізація на Kotlin; основний функціонал задач; безпечне збереження та синхронізація; тестування і оптимізація.*

5. Перелік графічного матеріалу: *19 рисунків, 1 додаток.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Повстяна Ю. С.*		
Специфікація вимог до розробленої системи	*Повстяна Ю. С.*		
Розробка об`єкта проектування	*Повстяна Ю. С.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	1,67 %		
Академічна доброчесність	*Повстяна Ю. С.*		

7. Дата видачі завдання *«20» грудня 2024 р.*

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	Виконано
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 01.03.2025 р.	Виконано
3	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	Виконано
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 29.03.2025 р.	Виконано
5	Практична реалізація об'єкта проектування та розробка бази даних	до 26.04.2025 р.	Виконано
6	Тестування та налагодження об'єкта проектування	до 03.05.2025 р.	Виконано
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	Виконано

Здобувач вищої освіти _____ Ярослав МАРТИНЧУК

Керівник кваліфікаційної роботи _____ Юлія ПОВСТЯНА

АНОТАЦІЯ

Мартинчук. Я. О. Розробка автоматизованої системи координації завдань у командних проєктах з використанням Gradle Kotlin DSL. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків і пропозицій, списку використаних джерел та додатків.

У роботі досліджено сучасні підходи до розробки мобільних додатків для організації командної роботи, зокрема особливості створення Android-застосунків із використанням мови програмування Kotlin, системи збирання Gradle Kotlin DSL, архітектурних шаблонів, а також засобів хмарної взаємодії на базі Firebase. Описано процес розробки автоматизованої системи, що дозволяє керівникам команд створювати завдання, призначати їх учасникам, контролювати виконання, проводити опитування та спільно координувати дії в межах групи. Особливу увагу приділено реалізації бази даних за допомогою Firebase Firestore, функціоналу авторизації через Firebase Authentication, а також забезпеченню зручного інтерфейсу для ефективної командної взаємодії. Проведено аналіз існуючих систем для управління завданнями, визначено їхні сильні та слабкі сторони, на основі чого сформульовано технічні вимоги та реалізовано функціональний прототип мобільного застосунку з урахуванням потреб тімлідів та виконавців.

Ключові слова: автоматизована система, керування завданнями, командна робота, Kotlin, Android, Gradle Kotlin DSL, Firebase, Firestore, аутентифікація, мобільний застосунок, to-do list.

ABSTRACT

Martynchuk Y. Development of an Automated Task Coordination System for Team Projects Using Gradle Kotlin DSL. Manuscript. Bachelor's qualification work in the educational program "Software Engineering" of specialty 121 "Software Engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification work consists of an introduction, three chapters, conclusions and recommendations, a list of references, and appendices.

This work explores modern approaches to mobile application development for organizing team collaboration, in particular the specifics of creating Android applications using the Kotlin programming language, the Gradle Kotlin DSL build system, architectural design patterns, and cloud-based interaction tools based on Firebase. The development process of an automated system is described, which enables team leaders to create tasks, assign them to team members, monitor their completion, conduct polls, and jointly coordinate actions within a team. Special attention is paid to the implementation of the database using Firebase Firestore, user authentication using Firebase Authentication, and the creation of a user-friendly interface for efficient team interaction. An analysis of existing task management systems was conducted, their strengths and weaknesses identified, technical requirements were formulated, and a functional prototype of a mobile application was developed to meet the needs of both team leaders and team members.

Keywords: automated system, task management, team collaboration, Kotlin, Android, Gradle Kotlin DSL, Firebase, Firestore, authentication, mobile application, to-do list.

ЗМІСТ

ВСТУП

Актуальність теми. Сучасна розробка програмного забезпечення є складним і динамічним процесом, що вимагає високої ефективності в управлінні командними завданнями та координації дій між учасниками. Зі збільшенням складності проєктів і кількості членів команд традиційні методи контролю через неструктуровані канали комунікації (чати, електронну пошту, усні обговорення) стають неефективними. У таких умовах виникає потреба в інструментах, які забезпечують централізоване управління завданнями, розподілення їх серед учасників, контроль строків виконання, відстеження статусу і оперативну реакцію на зміни в реальному часі.

Зокрема, важливими є мобільні додатки, які дозволяють залишатись на зв’язку з поточним станом проєкту, незалежно від місця перебування. У цьому контексті мобільний додаток на платформі Android, розроблений за допомогою мови програмування Gradle Kotlin DSL, забезпечує зручність використання, масштабованість і підтримуваність, а також можливість інтеграції з іншими сервісами.

Мета цієї кваліфікаційної роботи полягає в розробці автоматизованої системи координації завдань у командних проєктах, яка реалізує функціональність інтелектуального To-Do List для керівника і його команди. Система повинна забезпечити централізоване управління завданнями, відображення статусів, встановлення дедлайнів та зручну взаємодію між учасниками.

Об’єктом кваліфікаційної роботи є процеси управління завданнями в командних проєктах, де важлива координація та оптимізація взаємодії між учасниками з використанням сучасних технологій мобільного програмного забезпечення.

Предметом кваліфікаційної роботи є автоматизовані системи координації завдань, зокрема мобільні додатки для платформи Android, що розробляються з

використанням мови програмування Gradle Kotlin DSL для управління завданнями в командних проєктах.

Актуальність розробки визначається потребою в простих, ефективних і водночас гнучких інструментах, що сприяють оптимізації робочого процесу команди, знижують навантаження на керівника та забезпечують високу якість виконання завдань без необхідності постійного ручного контролю.

Для реалізації мети кваліфікаційної роботи необхідно виконати наступні конкретні цілі:

– провести аналіз предметної області та існуючих програмних рішень для організації координації завдань у командних проєктах, виявити їх сильні та слабкі сторони;

– сформулювати функціональні та нефункціональні вимоги до автоматизованої системи координації завдань для команди, орієнтованої на використання мобільних пристроїв під управлінням Android;

– розробити архітектуру застосунку з урахуванням моделі ролей користувачів (керівник та учасники команди) та підтримкою офлайн-режиму із синхронізацією даних;

– реалізувати мобільний застосунок за допомогою мови програмування Kotlin, забезпечивши нативну продуктивність, сучасні принципи UX/UI дизайну та оптимальну зручність використання;

– забезпечити основний функціонал застосунку: створення, редагування, делегування, моніторинг та завершення завдань, а також налаштування статусів і пріоритетів завдань;

– реалізувати механізми безпечного збереження даних локально на пристрої з подальшою їх синхронізацією при наявності мережевого підключення;

– провести тестування застосунку на предмет відповідності вимогам, виявити можливі помилки та оптимізувати продуктивність і зручність використання.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У сучасних умовах стрімкого розвитку IT-індустрії, цифрової трансформації бізнесу та зростаючої популярності гнучких методологій управління проєктами (Agile, Scrum, Kanban) проблема ефективної координації завдань у командній роботі набуває особливого значення.

На сьогодні все більше команд розробників, інженерів, дизайнерів, тестувальників та менеджерів стикаються з необхідністю синхронізованої роботи у спільних проєктах, часто за умов географічної віддаленості учасників. Тому забезпечення чіткої комунікації, розподілу ролей та завдань, відстеження статусів виконання та оперативного реагування на зміну пріоритетів стає критичним для досягнення поставлених цілей.

Класичні інструменти управління проєктами часто виявляються надмірно складними, перевантаженими функціоналом і не завжди придатними для потреб невеликих або кросфункціональних команд. У дослідженні Alsewari та Emran «Task Management Systems for Agile Software Development: A Review» (2021) проведено систематичний аналіз популярних систем управління завданнями, зокрема таких як Trello, Jira, Asana та Redmine, у контексті їх застосування в Agile-середовищах. Автори вказують, що попри широкий функціонал, більшість таких платформ мають низку обмежень у роботі з малими командами: «ці системи часто занадто складні в налаштуванні або надто перевантажені функціоналом, який не використовується в рамках невеликих гнучких проєктів» [1]. Дослідження підкреслює, що інтерфейс і користувацький досвід таких інструментів нерідко орієнтовані на масштабовані корпоративні процеси з формалізованими етапами розробки, що не відповідає реаліям невеликих команд. У цьому контексті автори рекомендують створення більш легких, адаптивних рішень, орієнтованих на простоту, швидке налаштування, мобільність та

зручність зворотного зв’язку. Такий підхід особливо актуальний для ролі керівника, якому необхідно швидко делегувати задачі, отримувати статус виконання в режимі реального часу та мати змогу адаптуватися до змін у проєкті без складної документації та довгих процесів узгодження.

У дослідженні Farooq, Aslam і Ahmad «Kotlin vs Java: A Comparative Study for Android Applications» (2020) проведено ґрунтовне порівняння мов Kotlin та Java щодо використання у мобільній розробці під Android [2]. Метою дослідження було визначити, яка з мов забезпечує вищу ефективність розробки, кращу підтримку сучасних технологій та кращу якість коду. Автори розглянули ключові аспекти, серед яких: продуктивність виконання, обсяг та складність коду, легкість навчання, інструментальна підтримка, безпека при роботі з нульовими значеннями (null safety) та сумісність із сучасними фреймворками й бібліотеками.

Результати експериментів продемонстрували, що Kotlin дозволяє суттєво зменшити обсяг коду (у деяких випадках – до 30-40 % менше, ніж у Java) завдяки лаконічному синтаксису, функціональним можливостям (наприклад, лямбда-виразам, розширенням функцій, data-класам) та автоматизованому управлінню типами. Це, у свою чергу, позитивно впливає на читабельність, підтримуваність та швидкість розробки. Kotlin також має вбудовану підтримку null safety, що дозволяє уникати поширених помилок, які часто призводять до крашів додатків у Java (NullPointerException).

У дослідженні також наголошується на повній інтероперабельності Kotlin із Java, що дозволяє поєднувати обидві мови в одному проєкті – важливий фактор для великих організацій, які мають наявну Java-кодову базу. Окрім цього, Kotlin краще підтримує сучасні архітектурні підходи, зокрема MVVM, які є стандартом у сучасній Android-розробці. Автори також підкреслюють переваги Kotlin при роботі з популярними Android-бібліотеками та сервісами: Room для роботи з локальними БД, Firebase для аутентифікації, повідомлень і хмарного збереження, а також Kotlin Coroutines для ефективної роботи з багатопоточністю та асинхронними задачами.

У висновках дослідження зазначено, що Kotlin є більш доцільним вибором для нових Android-проєктів завдяки своїй зручності, безпечності, продуктивності та активній підтримці з боку Google, яка офіційно оголосила Kotlin як основну мову для розробки під Android ще у 2019 році.

Щодо реалізації спільної роботи в рамках застосунку – було проаналізовано дослідження авторів Halaweh та El Massry (2021), у якому детально розглянуто вимоги, виклики та принципи проєктування колаборативних мобільних застосунків [3]. У статті підкреслюється, що ефективні мобільні системи для командної роботи мають забезпечувати: синхронізацію даних через хмару, підтримку кількох користувацьких ролей (наприклад, керівник – виконавець), механізми зворотного зв'язку та оновлення даних у реальному часі. Це дозволяє досягти високої узгодженості між членами команди, особливо у швидкозмінному середовищі розробки.

Автори зазначають, що архітектура подібних застосунків повинна базуватися на централізованому зберіганні даних, з чітко визначеними правами доступу до функцій та інформації відповідно до ролі користувача. Важливим є також реалізація системи пріоритетів, обміну коментарями та інших елементів мікрокомунікації в команді. Дослідження вказує на необхідність застосування сучасних архітектурних шаблонів і технологій (наприклад, REST API, WebSockets, Firebase), які забезпечують гнучкість та масштабованість рішення.

Окрему увагу автори приділяють інтерфейсу користувача. Наголошується, що інтуїтивний UX, адаптований під специфіку ролей користувачів, суттєво знижує когнітивне навантаження та сприяє швидкому прийняттю рішень. Особливо це важливо в умовах роботи в невеликих командах, де швидкість реакції та простота комунікації критично важливі.

Окрім аналізу досліджень, патентний пошук також є важливою складовою етапу розробки програмного забезпечення, що дозволяє визначити рівень новизни проєкту, унікальність його функціональних характеристик, а також уникнути порушення чинного законодавства щодо інтелектуальної власності. У даному випадку об'єктом дослідження виступає мобільний застосунок типу To-

Do List, призначений для автоматизації управління завданнями у командних проєктах, що реалізується мовою програмування Kotlin під платформу Android.

Методикою патентного пошуку стало використання відкритих міжнародних баз патентної документації, такі як: Google Patents, Espacenet (база Європейського патентного відомства) та United States Patent and Trademark Office (USPTO). Використовувались наступні ключові слова та фрази: task coordination mobile app, team project management, to-do application, task assignment system, Android project task system, Kotlin mobile coordination.

У результаті патентного пошуку було виявлено декілька патентів, які частково перетинаються з ідеєю розроблюваного програмного продукту, але не повністю дублюють його функціонал або архітектуру.

Патент US8146104B2 описує систему, яка інтегрує списки завдань із календарними додатками та автоматично створює завдання на основі даних з інших застосунків [4]. Проте він не адаптований для сучасних мобільних платформ і не використовує актуальні технології, такі як Kotlin або Gradle. Його функціонал обмежений інтеграцією з календарем і не охоплює комплексну координацію командних завдань або автоматизацію управління проєктами.

Патент US10977621B2 фокусується на автоматичному створенні завдань на основі дій користувача, але він не передбачає глибокої інтеграції в роботу команд та не має достатньої гнучкості для налаштувань під різні ролі й рівні керівництва [5].

Патент US8806613B2 описує інтелектуальне призначення завдань і авторизацію, але не містить детальної інтеграції з мобільними платформами, зокрема Android, і не передбачає гнучких налаштувань для команд різного розміру [6].

Патент US7596416B1 є інструментом управління проєктами, але орієнтований на стаціонарні або веб-системи, не оптимізований для мобільних пристроїв, і не підтримує автоматичне налаштування проєктів із використанням Kotlin або Gradle, що обмежує його гнучкість [7].

Патент US9239719B1 описує систему управління завданнями з віртуальною дошкою, але не має глибокої мобільної інтеграції і не використовує сучасних технологій Kotlin і Gradle [8].

Аналіз показав що жоден із знайдених патентів не повністю покриває задуману концепцію автоматизованої системи координації завдань для керівника, яка реалізується на Kotlin. У патентах не використовуються архітектури MVVM, не міститься інтерфейс командної взаємодії, немає розмежування ролей керівника та учасника, системи не дозволяють офлайн-роботу з синхронізацією через Firebase та не мають мінімалістичний мобільний інтерфейс, адаптований під Android. Таким чином, розроблювана система має достатній ступінь унікальності, що дозволяє вважати її оригінальною розробкою, яка не порушує прав чинних власників інтелектуальної власності.

Хоча на ринку існують схожі рішення у сфері мобільного управління завданнями, розроблюваний застосунок відрізняється унікальною комбінацією аспектів: використанням Kotlin для нативної Android-реалізації, зосередженням на командній координації саме з боку керівника, простим мобільний інтерфейсом, орієнтованим на малі та середні команди; гнучкою системою ролей та зберігання даних через хмару з офлайн-доступом. Усе це дає змогу позиціонувати розробку як оригінальну інновацію в межах тематики мобільного менеджменту задач, а також відкриває можливості для її подальшої реєстрації як об’єкта авторського права або патентування.

Одним із ключових етапів підготовки до розробки автоматизованої системи координації завдань у командних проєктах стало проведення ретельного аналізу існуючих рішень, які вже реалізовані на ринку. Метою даного етапу було виявлення сильних і слабких сторін існуючих систем, зокрема тих, що орієнтовані на мобільне середовище, а також оцінка можливостей використання певних концептуальних і технічних підходів при створенні власного продукту.

Одним із найпоширеніших інструментів для управління завданнями на ринку є Trello (рис. 1.1) [9]. Цей застосунок базується на концепції канбан-дошок, що забезпечує візуальне відображення процесів виконання завдань у вигляді

карток, які можна переміщати між різними етапами. Простота використання Trello і його зрозумілий інтерфейс дозволили цьому інструменту здобути широку популярність серед різних користувачів, від окремих фрілансерів до великих команд. Крім того, система підтримує можливість командної роботи, дозволяючи призначати завдання окремим учасникам та контролювати виконання через спільний простір.

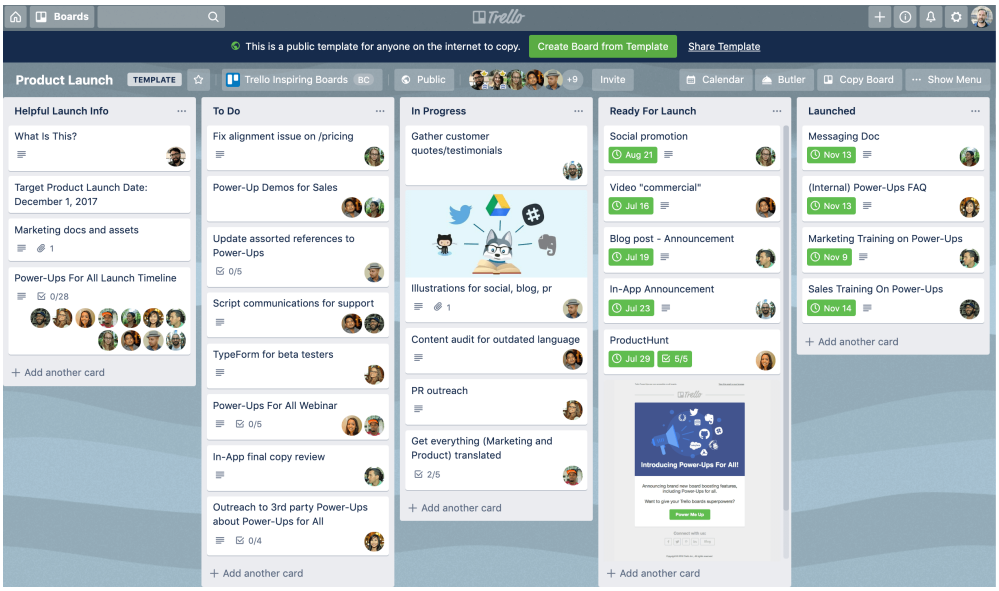

Рисунок 1.1 – Інтерфейс інструменту Trello [9]

До основних переваг Trello можна віднести простоту освоєння, зручність використання мобільної версії для Android (рис. 1.2) [10], а також доступність базового функціоналу безкоштовно.

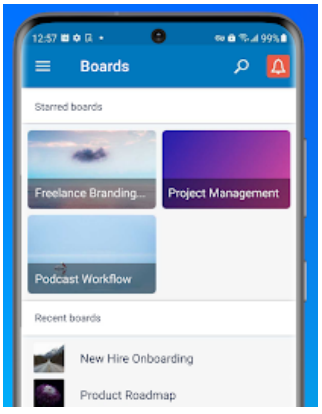

Рисунок 1.2 – Інтерфейс застосунку Trello на Android [10]

Водночас аналіз виявив і суттєві недоліки, серед яких варто відзначити обмежені можливості в налаштуванні ролей та прав доступу всередині командного простору.

При роботі з великою кількістю завдань інтерфейс втрачає свою зручність, що негативно позначається на ефективності управління. Окрім того, застосунок не підтримує повноцінну роботу в офлайн-режимі, що обмежує його використання в умовах нестабільного або відсутнього інтернет-з'єднання. З технічної точки зору, Trello розроблений із використанням кросплатформених технологій (React Native), що позначається на оптимізації роботи під Android-пристрої й унеможливлює повноцінну реалізацію на Kotlin.

Ще одним відомим аналогом є Asana (рис. 1.3) [11] – платформа, орієнтована переважно на корпоративний сегмент управління завданнями та проєктами.

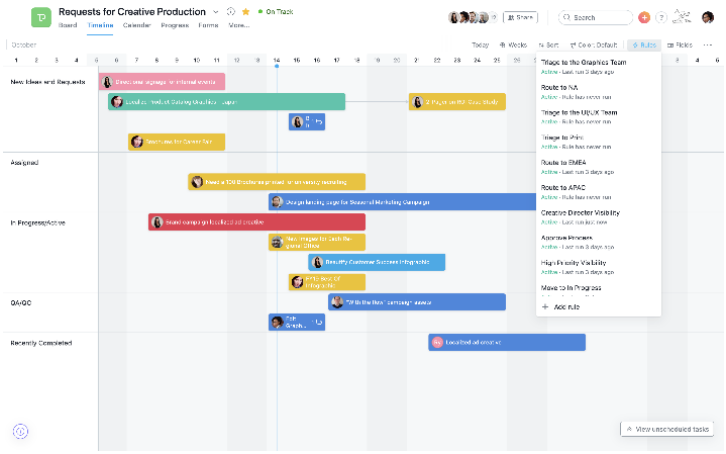

Рисунок 1.3 – Інтерфейс інструменту Asana [11]

Система дозволяє користувачам створювати завдання, призначати відповідальних осіб, визначати терміни виконання, групувати задачі у проєкти та підпроєкти, а також контролювати прогрес їх виконання через різноманітні візуальні інструменти, такі як діаграми Ганта або списки. Одним із ключових плюсів Asana є можливість інтеграції з великою кількістю сторонніх сервісів, серед яких Google Drive, Slack, Dropbox тощо, що значно підвищує ефективність командної співпраці.

Однак попри багатий функціонал Asana має і певні обмеження. Серед недоліків слід відзначити надмірну складність інтерфейсу для невеликих команд та новачків, що ускладнює початкове освоєння продукту. Крім того, багато функцій доступні лише за умов оформлення платної підписки, що не завжди виправдано для малих і середніх командних проєктів. Щодо мобільної версії, варто зазначити, що застосунок створений на основі кросплатформеної технології Flutter (рис. 1.4) [12], що також не відповідає вимогам нативної оптимізації під Android за допомогою Kotlin.

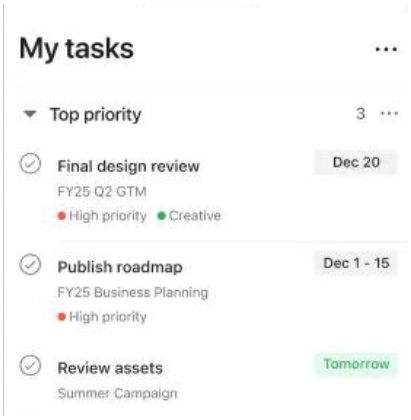

Рисунок 1.4 – Інтерфейс застосунку Asana на Android [12]

Третім проаналізованим рішенням стала система ClickUp (рис. 1.5) [13], яка позиціонує себе як універсальна платформа для організації всієї роботи команди в одному просторі.

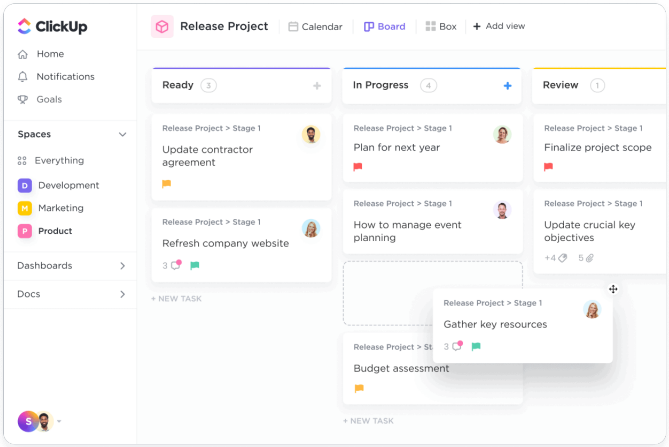

Рисунок 1.5 – Інтерфейс інструменту ClickUp [13]

ClickUp дозволяє не лише керувати завданнями, а й створювати документи, проводити обговорення, налаштовувати автоматизацію процесів та генерувати звіти. Основними перевагами ClickUp є висока гнучкість налаштувань під різні типи проєктів та команд, наявність великої кількості шаблонів для швидкого старту роботи, а також підтримка роботи в офлайн-режимі з подальшою синхронізацією даних при підключенні до мережі.

Разом з тим, ClickUp має певні недоліки. Зокрема, інтерфейс застосунку є надмірно складним, що може стати перешкодою для нових користувачів. Деякі функції в мобільній версії працюють нестабільно, що негативно впливає на досвід використання. Додатково слід зауважити, що мобільний застосунок ClickUp побудований із застосуванням кросплатформених технологій, а не нативної розробки на Kotlin, що не дозволяє максимально використати можливості Android-платформи (рис. 1.6) [14].

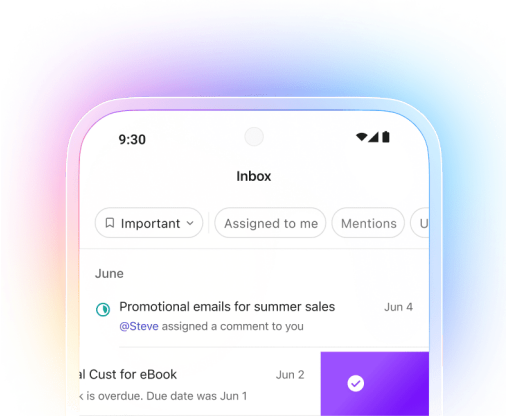

Рисунок 1.6 – Інтерфейс застосунку ClickUp на Android [14]

Проведений аналіз існуючих аналогів систем координації завдань у командних проєктах засвідчили, що на сучасному ринку представлено низку ефективних рішень, таких як Trello, Asana та ClickUp. Кожен із них має певні переваги, зокрема зручність інтерфейсу, підтримку командної роботи та можливість інтеграції зі сторонніми сервісами. Водночас було виявлено низку істотних недоліків, серед яких – складність освоєння для новачків, відсутність нативної реалізації для платформи Android засобами мови Kotlin, обмеженість

офлайн-функціоналу та перенавантаження інтерфейсу зайвими функціями для невеликих команд.

Аналіз підтвердив актуальність і обґрунтованість розробки власної автоматизованої системи координації завдань, яка буде орієнтована на середовища невеликих і середніх командних проєктів, реалізована за допомогою мови Kotlin для мобільних пристроїв Android, матиме простий інтуїтивний інтерфейс, чітке розмежування ролей між керівником та виконавцями, а також підтримку повноцінної роботи в офлайн-режимі. Запропонована система покликана об’єднати кращі практики існуючих рішень, водночас уникаючи їхніх основних недоліків, що дозволить створити конкурентоспроможний продукт для організації ефективної командної взаємодії.

Тож проведений огляд літературних джерел з предметної області продемонстрував актуальність проблеми організації ефективної координації завдань у командних проєктах. Сучасні підходи до управління завданнями орієнтуються на гнучкість процесів, підтримку спільної роботи та доступність інформації у реальному часі. При цьому особлива увага у наукових роботах приділяється простоті використання мобільних технологій, які дозволяють здійснювати управління завданнями незалежно від місця перебування учасників команди, що стає особливо важливим в умовах розвитку дистанційної роботи.

Аналіз результатів патентного пошуку показав, що на даний момент не існує запатентованих рішень, які б комплексно поєднували мобільність, автономність, простоту інтерфейсу та спеціалізовану орієнтацію на рольову модель «керівник – команда» у контексті нативної розробки для Android-платформи з використанням мови Kotlin. Відомі запатентовані системи здебільшого зосереджені на широкофункціональних корпоративних рішеннях або покривають лише окремі аспекти планування завдань, не враховуючи специфіку невеликих команд.

Порівняння з існуючими аналогами, такими як Trello, Asana та ClickUp, виявило кілька суттєвих проблем. Хоча сучасні системи забезпечують користувачам багатий функціонал управління завданнями, інтеграцію зі

сторонніми сервісами та підтримку командної роботи, проте жодне з рішень не об’єднує в собі нативної розробки на Kotlin для Android-пристроїв, простого й інтуїтивно зрозумілого інтерфейсу, оптимізованого під потреби невеликих і середніх команд, чіткого розмежування прав і ролей між керівниками та учасниками проєктів, а також повноцінної можливості роботи в офлайн-режимі із надійною синхронізацією даних після відновлення мережевого підключення. Крім того, використання кросплатформених технологій, таких як React Native або Flutter, негативно впливає на продуктивність мобільних застосунків і ускладнює глибоку інтеграцію з системними ресурсами Android, що знижує загальну якість користувацького досвіду.

Особливої уваги потребує також той факт, що більшість сучасних застосунків перенавантажені надлишковими функціональними модулями, які є актуальними для великих корпоративних проєктів, проте не є необхідними для роботи невеликих команд. У результаті такі системи стають складними для початкового освоєння, вимагають додаткових ресурсів на адміністрування і часто передбачають платну підписку для розблокування ключового функціоналу.

Ураховуючи зазначене вище, доцільність створення нової автоматизованої системи координації завдань у командних проєктах є очевидною. Пропонована система має заповнити наявну ринкову нішу та задовольнити потреби малих і середніх командних проєктів, які прагнуть використовувати мобільні технології для ефективної організації роботи. Вона буде нативно розроблена на мові Kotlin, що дозволить забезпечити високу швидкодію, стабільність та безпеку роботи на платформі Android. Застосунок орієнтуватиметься на швидке створення та розподіл завдань, чітке розмежування ролей між керівниками та виконавцями, легкий контроль статусу виконання завдань, а також мінімалістичний дизайн, що фокусуватиме увагу користувачів на основних діях без перевантаження інтерфейсу зайвими елементами. Крім того, система забезпечуватиме повноцінну роботу в офлайн-режимі із надійною синхронізацією даних після відновлення інтернет-з’єднання, що є критично важливим для забезпечення безперервності робочого процесу.

Таким чином, розробка нової системи має як теоретичне обґрунтування відповідно до сучасних наукових тенденцій, так і практичну необхідність з огляду на недоліки існуючих аналогів. Запропоноване рішення дозволить створити ефективний, стабільний та зручний інструмент для організації командної роботи у сфері мобільних застосунків.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

У сучасних умовах розвитку технологій та поширення дистанційної праці ефективна організація командної роботи вимагає застосування інструментів для планування та координації завдань. Існуючі рішення не завжди враховують специфіку потреб невеликих команд і оптимізацію під мобільні пристрої. У зв’язку з цим виникає необхідність створення нового мобільного застосунку, який би забезпечував гнучке керування завданнями та спрощував робочі процеси в командних проєктах.

Для реалізації мети кваліфікаційної роботи необхідно виконати наступні конкретні цілі:

– провести аналіз предметної області та існуючих програмних рішень для організації координації завдань у командних проєктах, виявити їх сильні та слабкі сторони;

– сформулювати функціональні та нефункціональні вимоги до автоматизованої системи координації завдань для команди, орієнтованої на використання мобільних пристроїв під управлінням Android;

– розробити архітектуру застосунку з урахуванням моделі ролей користувачів (керівник та учасники команди) та підтримкою офлайн-режиму із синхронізацією даних;

– реалізувати мобільний застосунок за допомогою мови програмування Kotlin, забезпечивши нативну продуктивність, сучасні принципи UX/UI дизайну та оптимальну зручність використання;

– забезпечити основний функціонал застосунку: створення, редагування, делегування, моніторинг та завершення завдань, а також налаштування статусів і пріоритетів завдань;

– реалізувати механізми безпечного збереження даних локально на пристрої з подальшою їх синхронізацією при наявності мережевого підключення;

– провести тестування застосунку на предмет відповідності вимогам, виявити можливі помилки та оптимізувати продуктивність і зручність використання.

Виконання зазначених цілей дозволить створити ефективний інструмент для організації командної роботи у вигляді сучасного мобільного застосунку, що відповідає актуальним потребам ринку мобільних технологій.

У першому розділі було проведено комплексний аналіз предметної області, що охоплює сучасні підходи до організації командної роботи та управління завданнями із застосуванням мобільних технологій. Було розглянуто основні наукові дослідження, які підтверджують актуальність створення простих, зручних та ефективних систем для координації завдань у командах. Огляд літературних джерел засвідчив, що на сьогоднішній день існує стійка тенденція до розвитку мобільних застосунків, орієнтованих на підтримку гнучких моделей командної взаємодії та мінімізації складності інтерфейсу.

Проведений патентний пошук показав відсутність запатентованих рішень, які б одночасно відповідали вимогам нативної розробки на Kotlin для Android, підтримували офлайн-роботу та були спеціалізованими для моделі «керівник – команда». Існуючі аналоги, такі як Trello, Asana та ClickUp, хоча і мають широкий функціонал, але характеризуються перенавантаженістю інтерфейсів, недостатньою оптимізацією для невеликих команд і часто вимагають підключення до інтернету для повноцінної роботи.

Аналіз аналогів виявив їхні сильні сторони, серед яких гнучкість налаштувань та інтеграція зі сторонніми сервісами, проте також були ідентифіковані ключові недоліки – складність освоєння, обмежена офлайн-

функціональність та залежність від платних підписок для розширеного доступу до функцій.

На основі проведеного аналізу було обґрунтовано доцільність створення нової автоматизованої системи координації завдань у командних проєктах. Запропонований застосунок має бути нативно розробленим на Kotlin для Android, забезпечувати простоту інтерфейсу, підтримувати чітке розмежування ролей користувачів, працювати в офлайн-режимі та синхронізувати дані після відновлення підключення до мережі.

У результаті сформульовано конкретні цілі кваліфікаційної роботи, що передбачають аналіз предметної області, формування вимог до системи, проєктування архітектури, реалізацію мобільного застосунку з необхідним функціоналом, проведення тестування та підготовку експлуатаційної документації.

Висновки до розділу 1

Проведено аналіз предметної області та сучасних мобільних рішень для командної роботи. Виявлено, що існуючі системи (Trello, Asana, ClickUp) не враховують потреби малих команд, не мають зручного офлайн-режиму та не реалізовані нативно на Kotlin. Патентний пошук підтвердив відсутність аналогів із таким функціоналом. Обґрунтовано доцільність створення нової системи координації завдань із простим інтерфейсом, підтримкою ролей та автономної роботи, що і стало метою кваліфікаційної роботи.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Аналіз і визначення вимог до автоматизованої системи координації завдань у командних проєктах, зокрема для створення To-Do List застосунку для керівника та його команди на мові Kotlin, включає як функціональні, так і нефункціональні аспекти, що забезпечують ефективність, зручність використання та безпеку системи.

Функціональні вимоги до системи охоплюють основні можливості, які забезпечують комфортне та ефективне управління завданнями в команді. Система повинна включати реєстрацію та авторизацію користувачів, що дозволяє кожному учаснику зареєструвати обліковий запис та увійти в систему. Система передбачає рольову модель, де визначено дві основні ролі: керівник команди (адміністратор) та виконавець (співробітник). Керівник має повний доступ до всіх завдань, може їх створювати, редагувати та призначати виконавців. Виконавець отримує завдання від керівника та має можливість оновлювати їх статус, проте не може редагувати чи видаляти завдання.

Основною функціональністю застосунку є створення та управління завданнями. Керівник може створювати завдання, вказуючи його назву, опис, термін виконання та пріоритет, а також призначати конкретного виконавця. Користувачі можуть переглядати завдання, змінювати їхній статус, наприклад, від «Не розпочато» до «У процесі» або «Завершено» чи «На перевірці». Система також повинна фільтрувати завдання за параметрами виконання, а саме: переносити виконані завдання у кінець сторінки. Особистий кабінет користувача дозволяє переглядати його завдання, відстежувати та редагувати статус виконання завдань.

Щодо нефункціональних вимог, система повинна забезпечувати високу продуктивність, щоб швидко обробляти запити користувачів та мінімізувати час

завантаження даних. Це важливо для ефективної роботи команди, адже затримки можуть впливати на швидкість виконання завдань. Безпека є ключовим аспектом, тому передбачено використання HTTPS для шифрування даних, хешування паролів та захист від атак, таких як SQL-ін’єкції та CSRF. Масштабованість системи також має бути важливою, адже при збільшенні кількості користувачів чи задач система повинна бути готова до збільшення навантаження без зниження ефективності.

Інтерфейс застосунку має бути інтуїтивно зрозумілим і зручним для використання. Він повинен підтримувати адаптивний дизайн, що забезпечує коректне відображення на різних пристроях – смартфонах та планшетах. Важливо, щоб користувачі швидко знаходили необхідні функції і не витрачали час на пошук інструментів для управління завданнями. Крім того, зручність користування забезпечується через застосування елементів матеріального дизайну, які забезпечують сучасний і зрозумілий інтерфейс.

Визначення ролей користувачів у системі допомагає забезпечити чітке розмежування доступів. Керівник має доступ до всіх функцій і можливість управляти командою, створювати, призначати, редагувати та видаляти завдання, видаляти інформацію користувачів, створювати опитування та переглядати результати. Виконавець має можливість тільки виконувати завдання, змінювати їхній статус, але не може редагувати самі завдання. Адміністратор займається управлінням користувачами, налаштуванням усієї бази даних та налаштуванням входу.

Для відображення архітектури системи та взаємозв’язків між її складовими я використовую діаграму прецедентів (Use Case Diagram) (рис. 2.1). Така діаграма дозволяє наочно представити логіку роботи застосунку, показати взаємодію між користувачами та функціональністю системи, а також формує зрозуміле уявлення про принципи роботи платформи для всієї команди розробників.

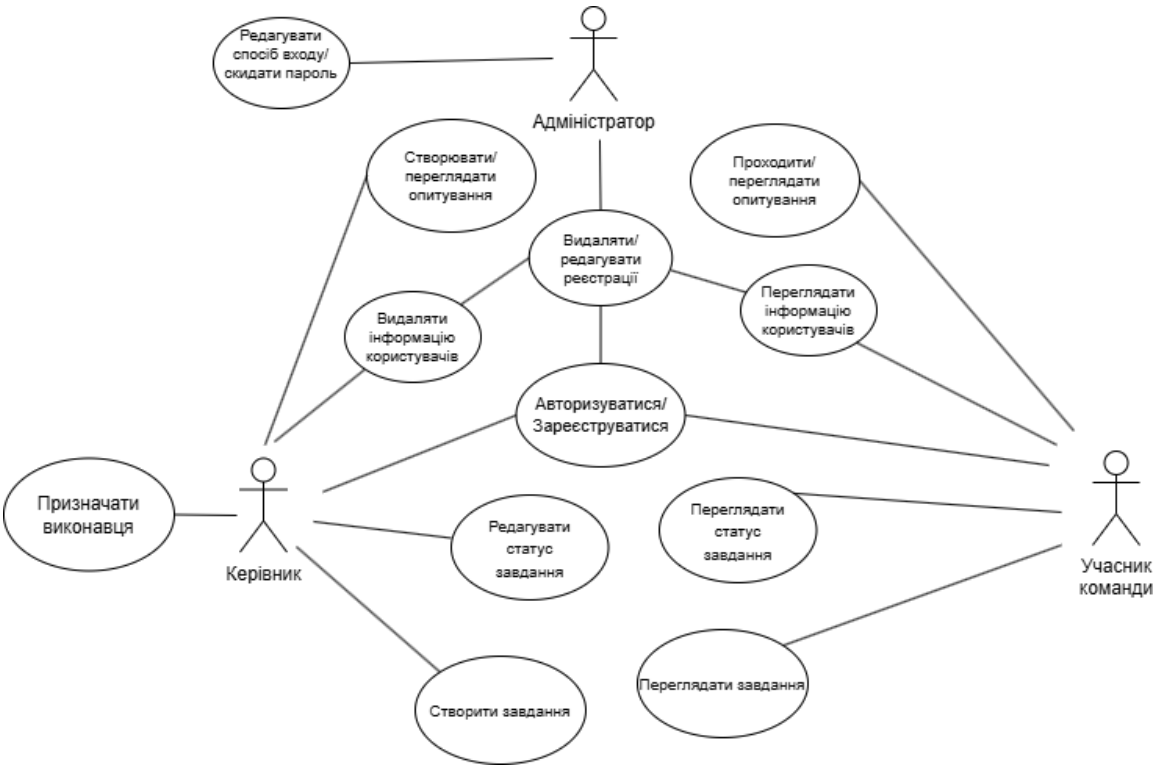

Рисунок 2.1 – UML-діаграма прецедентів

Ця діаграма прецедентів відображає логіку роботи застосунку, де участь беруть три ролі: адміністратор, керівник команди та учасник команди. У центрі розміщено основний сценарій – «Авторизуватися / Зареєструватися», який відкриває доступ до решти функціональності залежно від ролі. Адміністратор відповідає за загальне управління системою: він керує користувачами (редагує або видаляє інформацію), налаштовує способи входу, скидку паролів, а також створює та редагує реєстрації. Керівник команди зосереджений на роботі з завданнями: створює задачі, редагує їхній статус і призначає виконавців, створює та редагує опитування та видаляє реєстрації. Учасник команди має доступ до перегляду своїх завдань, їх статусу, а також може проходити й переглядати опитування та бачити інформацію про інших користувачів.

Таким чином, діаграма чітко демонструє розподіл відповідальностей у межах команди: адміністратор керує системою, керівник – робочими процесами, а учасник – виконує призначені задачі.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

На основі проведеного аналізу в розділі 1, що охоплює сучасні тенденції у сфері командної роботи та мобільних технологій, було визначено оптимальний технологічний стек для розробки мобільного застосунку, орієнтованого на ефективну координацію завдань у проєктах малого та середнього масштабу. Основною платформою стала Android, а мовою розробки – Kotlin, яка вирізняється лаконічним синтаксисом, високою безпекою за рахунок null safety, підтримкою сучасних архітектурних підходів і активною підтримкою з боку Google. Це забезпечує зручність реалізації нативного застосунку, високу продуктивність та сумісність з Android-середовищем.

Для конфігурації проєкту використано Gradle Kotlin DSL, який дозволяє гнучко налаштовувати проєкт і сприяє кращій структурованості коду. Обробка асинхронних подій реалізується за допомогою Kotlin Coroutines, що підвищує ефективність виконання задач у реальному часі. Для роботи з локальною базою даних застосовується бібліотека Room, яка надає зручний ORM-рівень взаємодії з локальним сховищем.

У контексті зберігання й обміну даними ключову роль відіграє хмарний сервіс Firebase. Його використання дозволяє реалізувати зберігання інформації в реальному часі, синхронізацію між користувачами, а також роботу в офлайн-режимі з подальшим оновленням після відновлення з’єднання. Авторизація користувачів у системі реалізована через Firebase Authentication, що забезпечує безпеку доступу до функціональності відповідно до ролей. Такий підхід дозволяє уникнути складного налаштування власного серверного бекенду на початковому етапі та забезпечує масштабованість у майбутньому.

Оскільки одним із ключових елементів є зручність використання мобільного застосунку, інтерфейс розробляється з урахуванням принципів Material Design. Це забезпечує інтуїтивну навігацію, привабливий вигляд та адаптивність до різних розмірів екранів. Застосунок буде підтримувати дві

основні ролі: керівника, який має можливість створювати та призначати завдання, контролювати їх виконання й редагувати дані; та виконавця, який оновлює статуси отриманих задач. При цьому архітектура системи дозволяє працювати без доступу до мережі, з наступною синхронізацією даних – що критично важливо для забезпечення безперервності роботи.

Висновки до розділу 2

Проведено аналіз вимог до системи координації завдань та визначено її основний функціонал, ролі користувачів і критерії зручності, безпеки та продуктивності. Обґрунтовано вибір Kotlin, Gradle DSL, Firebase, Room і Jetpack Compose як технологічної бази. Сформовано архітектуру застосунку з підтримкою ролей, офлайн-режиму та синхронізації, що відповідає сучасним вимогам до мобільних командних рішень.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКА

3.1 Практична реалізація об’єкта проєктування

Одним із ключових етапів побудови автоматизованої системи координації завдань є створення механізму авторизації та реєстрації користувачів. У застосунку TaskFlow, реалізованому для Android на мові програмування Kotlin із використанням архітектури MVVM та фреймворку Jetpack Compose, ці механізми побудовано на базі сервісу Firebase Authentication та бази даних Firebase Firestore.

При вході в застосунок Вас зустрічають вікна авторизації та реєстрації. У застосунку реалізовано окрему активність AuthorizationActivity, яка містить форму входу з полями для введення електронної пошти та пароля, а також кнопки для переходу до реєстрації або входу. Інтерфейс розроблений із використанням Jetpack Compose.

У застосунку TaskFlow вхід користувача реалізовано через Firebase Authentication. Користувач вводить електронну пошту та пароль, після чого ViewModel передає дані в Firebase для перевірки.

На початковому етапі розробки розглядалися різні варіанти реалізації UI-елементів для введення даних, зокрема, використання звичайного TextField. Однак з міркувань зручності та відповідності Material Design, було обрано компонент OutlinedTextField (рис. 3.1), який забезпечує чітку візуальну межу поля вводу та підтримує підказку (label), що автоматично анімується при фокусі.

Рисунок 3.1 – Поле введення пошти

Цей підхід дозволяє покращити UX і підтримує масштабованість інтерфейсу при додаванні нових полів (ліст. 3.1).

Лістинг 3.1 – Поле введення електронної пошти у вікні входу

```
OutlinedTextField(
    value = email,
    onValueChange = { email = it },
    label = { Text("Електронна пошта") }
)
```

кінець лістингу 3.1

Таким чином дані зберігаються у змінній email, яка оновлюється кожного разу при введенні символу. Використовується Jetpack Compose.

Ще одним важливим елементом вікна входу стало поле для введення пароля. З міркувань безпеки та відповідності стандартам UX, було прийнято рішення використати PasswordVisualTransformation() (ліст. 3.2), що приховує символи, які вводить користувач. Це забезпечує базову конфіденційність під час введення пароля, особливо на загальнодоступних пристроях.

Лістинг 3.2 – Поле для введення пароля «PasswordVisualTransformation()»

```
OutlinedTextField(
    value = password,
    onValueChange = { password = it },
    label = { Text("Пароль") },
    visualTransformation = PasswordVisualTransformation()
)
```

кінець лістингу 3.2

Саме поле для введення пароля, PasswordVisualTransformation() приховує введені символи, забезпечуючи конфіденційність (рис. 3.2).

Рисунок 3.2 – Конфеденційність паролю

Використання OutlinedTextField у поєднанні з PasswordVisualTransformation дозволяє створити зручний та безпечний елемент

введення, який легко масштабувати та доповнити додатковими функціями, наприклад, перемикачем видимості пароля або перевіркою складності введених даних.

Завершальним елементом у вікні входу стала кнопка, що запускає процес авторизації користувача. Вона відіграє ключову роль в інтерфейсі, оскільки саме з її натисканням відбувається перевірка введених облікових даних. При реалізації використовувався стандартний компонент Button з бібліотеки Jetpack Compose, який забезпечує інтерактивну поведінку та вбудовану анімацію натискання (ліст. 3.3).

Лістинг 3.3 – Кнопка для входу

```
Button(onClick = { onLoginClick(email, password) }) {
    Text("Увійти")
}
```

кінець лістингу 3.3

При натисканні самої кнопки викликається onLoginClick, яка передає введені дані у ViewModel для перевірки користувача через Firebase.

Після натискання кнопки «Увійти» відбувається безпосередній виклик логіки аутентифікації користувача. Для цього реалізована функція login, яка використовує стандартний механізм авторизації через Firebase Authentication. Вона приймає електронну пошту та пароль як вхідні параметри, а також зворотний виклик onResult, який дозволяє обробити результат – успішну або невдалу спробу входу (ліст. 3.4).

Лістинг 3.4 – Аутентифікація користувача через Firebase

```
fun login(email: String, password: String, onResult: (Boolean, String?)
-> Unit) {
    firebaseAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                onResult(true, null)
            } else {
                onResult(false, task.exception?.message)
```

```
            }
        }
}
```

кінець лістингу 3.4

Метод signInWithEmailAndPassword намагається аутентифікувати користувача з Firebase. У разі успіху (task.isSuccessful) – викликається onResult(true, null), тобто повертається повідомлення, що вхід успішний. Якщо щось пішло не так – помилка передається через task.exception?.message, наприклад: Invalid password or no user record (рис. 3.3).

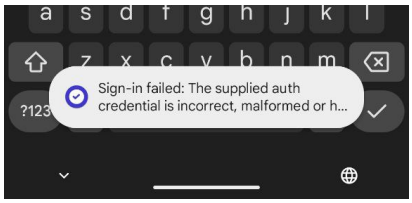

Рисунок 3.3 – Помилка при вході

Після успішного входу в систему реалізується перенаправлення користувача до відповідного екрану залежно від його ролі в системі (ліст. 3.5).

Лістинг 3.5 – Перехід після входу

```
if (user.role == Role.LEADER) {
    startActivity(Intent(this, LeaderDashboardActivity::class.java))
} else {
    startActivity(Intent(this, EmployeeTasksActivity::class.java))
}
```

кінець лістингу 3.5

У застосунку передбачено два типи користувачів: керівник команди (LEADER) та звичайний учасник (EMPLOYEE) (рис. 3.4). На основі ролі, що зберігається у моделі user, виконується умовна перевірка та ініціюється запуск відповідної активності за допомогою Intent: керівник бачить інтерфейс управління всіма задачами, працівник – лише призначені йому.

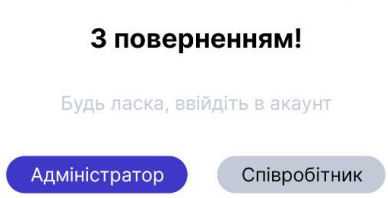

Рисунок 3.4 – Кнопки обрання ролі

Для підтримки повноцінного функціоналу доступу до застосунку була реалізована окрема форма реєстрації, яка відкривається після натискання відповідної кнопки на екрані входу (рис. 3.5).

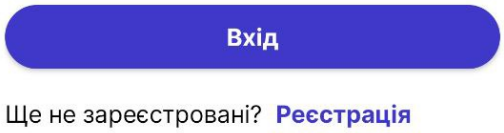

Рисунок 3.5 – Кнопка переходу на вікно реєстрації

У цьому вікні новий користувач має можливість ввести базову інформацію, необхідну для створення облікового запису: ім’я, електронну пошту, пароль і роль у команді (адміністратор чи співробітник). Дані спершу передаються в Firebase для створення облікового запису, а потім – зберігаються у Firestore (ліст. 3.6).

Лістинг 3.6 – Реєстраційна форма користувача

```
OutlinedTextField(value = name, onValueChange = { name = it }, label =
{ Text("Ім’я") })
OutlinedTextField(value = email, onValueChange = { email = it }, label =
{ Text("Email") })
OutlinedTextField(
    value = password,
    onValueChange = { password = it },
    label = { Text("Пароль") },
    visualTransformation = PasswordVisualTransformation()
)
```

кінець лістингу 3.6

Користувач вводить особисті дані. Ці значення зберігаються у змінних name, email, password для подальшої обробки. У рамках реєстраційної форми важливо надати користувачеві можливість вказати свою роль у системі, оскільки від цього залежить подальший доступ до функціональності застосунку (ліст. 3.7).

Лістинг 3.7 – Вибір ролі користувача через DropdownMenu

```
DropdownMenu(role = role, onRoleSelected = { role = it })
```

кінець лістингу 3.7

Цей компонент дозволяє вибрати роль користувача зі списку значень перерахування Role, наприклад EMPLOYEE або LEADER. Це визначає доступ і функціональність користувача після входу. Такий підхід дозволяє реалізувати чітке розмежування прав користувачів вже на етапі реєстрації та спрощує логіку переходу до відповідних екранів після входу.

Реалізовано функціонал централізованої координації завдань на основі унікальних ідентифікаторів груп (Group ID). Система надає можливість керівнику створювати нові групи з автоматично генерованим Group ID, який служить ключем для реєстрації та приєднання працівників до відповідного робочого простору (рис. 3.6).

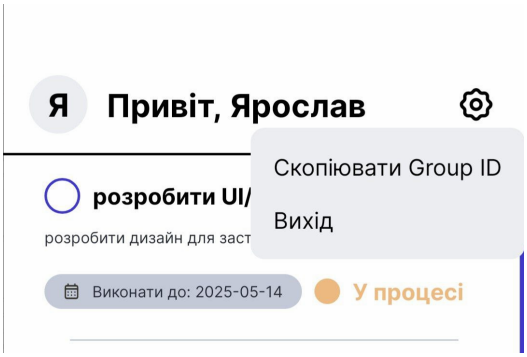

Рисунок 3.6 – Group ID

Процес реєстрації організовано таким чином, що після отримання Group ID працівники можуть вказати його при реєстрації у застосунку, що забезпечує

їх автоматичну прив’язку до конкретної групи. Це дозволяє централізовано керувати розподілом і контролем завдань, забезпечує синхронізацію даних між усіма учасниками групи в реальному часі, а також зберігає історію виконання робіт.

Клас MainActivity (ліст. 3.8) є центральним елементом інтерфейсу користувача в мобільному застосунку TaskFlow. Його реалізація охоплює логіку завантаження та відображення списку завдань, персоналізацію інтерфейсу під користувача, а також навігацію між основними екранами програми.

Лістинг 3.8 – Метод onCreate у MainActivity

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
 ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v,
insets ->
        val systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars())
        v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom)
        insets
    }
}
```

кінець лістингу 3.8

У методі onCreate() відбувається ініціалізація інтерфейсу через ViewBinding, встановлюються системні відступи для коректного відображення, а також викликаються методи, що відповідають за налаштування UI, обробку подій користувача та завантаження даних з Firestore.

У методі setupUI() (ліст 3.9) здійснюється персоналізація інтерфейсу.

Лістинг 3.9 – Метод setupUI

```
private fun setupUI() {
    binding.txUserName.text = user.name.first().toString().uppercase()
    var name = SharedPreferencesRepository.getUser()?.name
    binding.txTitle.text = getString(R.string.hello, name)
```

```
    binding.btnAddTask.visibility =
        if (user.role == Role.ADMIN) View.VISIBLE else View.INVISIBLE
}
```

кінець лістингу 3.9

Відображається перша літера імені користувача, формується привітальне повідомлення та визначається доступність кнопки додавання завдань, що відображається лише для адміністратора (рис. 3.7).

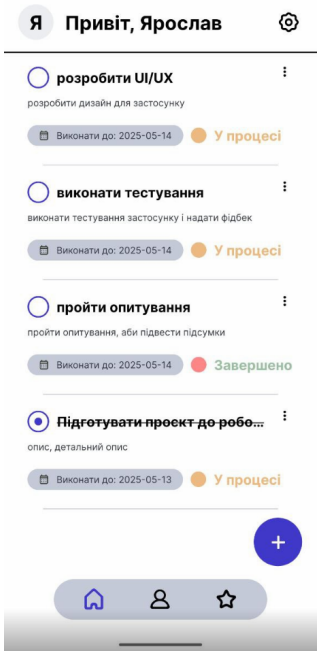

Рисунок 3.7 – Головна сторінка адміністратора

Взаємодія з елементами інтерфейсу забезпечується методом setListener() (ліст. 3.10), де встановлено обробники для кнопок: відкриття меню налаштувань, переходу до створення завдання, перегляду групи та опитувань.

Лістинг 3.10 – Метод setListener

```
private fun setListener() {
    binding.btnAddTask.setOnClickListener {
        navigateToAddTask()
    }
    binding.btnSettings.setOnClickListener {
        showSettingsPopupMenu(binding.btnSettings)
    }
```

```
    binding.btnMyGroup.setOnClickListener {
        navigateToMyGroup()
    }
    binding.btnPolls.setOnClickListener {
        navigateToPolls()
    }
}
```

кінець лістингу 3.10

Для отримання завдань з Firestore використовується метод getTasks() (ліст. 3.11). Залежно від ролі користувача (адміністратор чи звичайний користувач) завантажуються всі завдання певної групи або лише завдання конкретного користувача. Отримані завдання передаються адаптеру для ініціалізації або оновлення списку.

Лістинг 3.11 – Метод getTasks

```
private fun getTasks() {
    lifecycleScope.launch {
        val result =
            if (user.role == Role.ADMIN) {
                firestoreRepository.getListOfTasks(user.groupId)
            } else {
            firestoreRepository.getListOfTasksCurrentUser(user.id)
            }
        result.onSuccess { tasks ->
            if (isFirstGetTasks) {
                initRecyclerViewTasks(tasks)
                isFirstGetTasks = false
            } else {
                taskAdapter.updateTasks(tasks)
            }
        }.onFailure { error ->
            showToast(error.message.toString())
        }
    }
}
```

кінець лістингу 3.11

Ініціалізація списку завдань виконується через метод initRecyclerViewTasks() (ліст. 3.12), де створюється адаптер TaskAdapter, що підтримує функції редагування, перегляду, зміни статусу тощо.

Лістинг 3.12 – Метод initRecyclerViewTasks

```
private fun initRecyclerViewTasks(tasks: List<Task>) {
    var isAdmin = user.role == Role.ADMIN
    var onCheckBoxClick: ((Task) -> Unit) = { task -> updateTask(task,
null) }
    var onBtnMoreClick: ((anchor: View, Task) -> Unit) =
        { anchor, task -> showTaskPopupMenu(anchor, task) }
    var onBtnInfoClick: ((Task) -> Unit) = { task ->
showAlertDialogTask(task) }
    taskAdapter =
        TaskAdapter(
            tasks,
            isAdmin,
            onCheckBoxClick,
            onBtnMoreClick,
            onBtnInfoClick
        )
    binding.rvTasks.adapter = taskAdapter
    hideLoading()
}
```

кінець лістингу 3.12

Окрему увагу приділено виводу інформації про завдання у вигляді діалогового вікна. Метод showAlertDialogTask(), що наведено у додатку А, реалізує відображення даних завдання з можливістю змінити його статус. Для вибору статусу використовується кастомний спінер PowerSpinnerView (рис. 3.8).

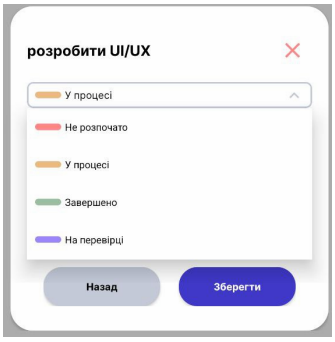

Рисунок 3.8 – PowerSpinnerView

Усі зміни статусу або видалення завдань автоматично синхронізуються з Firestore, а список оновлюється. Також реалізовано методи для переходу до

інших активностей: navigateToAddTask(), navigateToMyGroup(), navigateToPolls() та navigateToAuthorization().

Таким чином, клас MainActivity виступає основною точкою взаємодії користувача із застосунком, забезпечуючи адаптивність, гнучкість і масштабованість інтерфейсу завдяки сучасному підходу до реалізації логіки та UI.

Функціональність створення нового завдання в застосунку TaskFlow реалізовано в окремій активності AddTaskActivity (рис. 3.9).

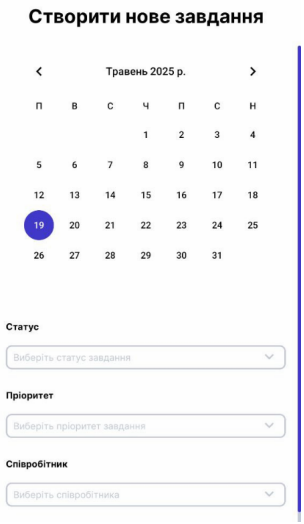

Рисунок 3.9 – Активність AddTaskActivity

Ця активність відкривається при натисканні на кнопку додавання завдання з головного екрану застосунку та використовується як для створення нового завдання, так і для редагування вже існуючого. Після запуску відбувається ініціалізація елементів інтерфейсу за допомогою ViewBinding, що забезпечує зручний доступ до компонентів розмітки. Далі виконується перевірка на наявність переданого об'єкта завдання через Intent: якщо дані передано, активується метод «populateForm()», який автоматично заповнює відповідні поля форми значеннями з отриманого об’єкта, забезпечуючи зручне редагування без потреби ручного введення інформації (ліст. 3.13).

Лістинг 3.13 – Перевірка переданого завдання у Intent

```
val json = intent.getStringExtra("task")
json?.let {
    selectedTask = Gson().fromJson(it, Task::class.java)
    populateForm(selectedTask!!)
}
```

кінець лістингу 3.13

Інтерфейс активності побудовано на базі Jetpack ViewBinding та підтримує введення таких даних: заголовок завдання, його опис, дедлайн (дата виконання), а також вибір статусу зі списку, пріоритету та обрання співробітника на виконання. Для покращення користувацького досвіду як компонент для вибору статусу використано PowerSpinnerView, який дозволяє обирати значення з адаптованими іконками відповідно до обраного стану.

Список статусів створюється під час ініціалізації інтерфейсу (ліст. 3.14). Кожному статусу відповідає іконка, що спрощує візуальне сприйняття.

Лістинг 3.14 – Формування списку статусів для спінера

```
val items = arrayListOf(
    IconSpinnerItem(R.drawable.shape_1, "Не розпочато"),
    IconSpinnerItem(R.drawable.shape_2, "У процесі"),
    IconSpinnerItem(R.drawable.shape_3, "Завершено"),
    IconSpinnerItem(R.drawable.shape_4, "На перевірці")
)
```

кінець лістингу 3.14

Крім того, у поле дедлайну за замовчуванням підставляється поточна дата, що дозволяє зменшити кількість кроків для створення типового завдання.

Після заповнення форми користувач натискає кнопку «Зберегти», після чого виконується перевірка на заповненість полів. У разі успішного введення створюється або оновлюється об’єкт класу Task (в залежності від того, чи був переданий на вхід об’єкт), після чого цей об’єкт готується до збереження (ліст. 3.15).

Лістинг 3.15 – Створення об’єкта Task для збереження

```
val task = Task(

    title = title,
    description = description,
    deadline = deadline,
    status = status,
    userId = user.id,
    groupId = user.groupId
)
```

кінець лістингу 3.15

Збереження в базу даних відбувається в корутині, що дозволяє не блокувати основний потік програми. Для цього використовується метод репозиторію, що взаємодіє з Firebase Firestore (ліст. 3.16).

Лістинг 3.16 – Збереження завдання у Firebase Firestore

```
lifecycleScope.launch {
    val result = firestoreRepository.addOrUpdateTask(task)
    result.onSuccess {
        showToast("Завдання збережено")
        finish()
   }
}
```

кінець лістингу 3.16

У разі успіху виводиться повідомлення про збереження, і активність закривається (рис. 3.10).

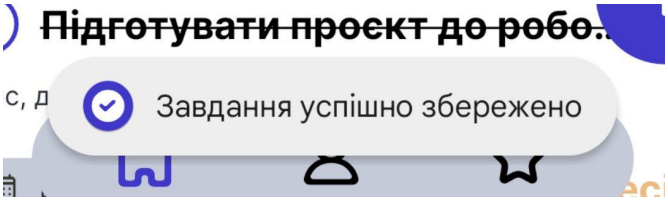

Рисунок 3.10 – Повідомлення про збереження завдання

Таким чином, вікно створення завдання реалізовано відповідно до сучасних принципів зручності, безпеки та адаптивності. Реалізація враховує роль

користувача, дозволяє як створювати нові задачі, так і редагувати наявні, забезпечуючи зручну взаємодію з інтерфейсом та базою даних.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

У процесі створення інформаційно-комп’ютерної системи для автоматизованої координації завдань у командних проєктах було проведено комплексне тестування та налагодження програмного забезпечення. Особливу увагу приділено виявленню критичних помилок, покращенню стабільності взаємодії з Firebase, а також забезпеченню належної взаємодії між окремими модулями застосунку, що реалізований із використанням Kotlin та Gradle Kotlin DSL.

У ході розробки застосовувалися поєднані підходи тестування – зокрема, ручне тестування інтерфейсу користувача, інтеграційне тестування компонентів та поведінкове тестування логіки роботи. Наприклад, після реалізації функціоналу створення та оновлення завдань (через AddTaskActivity та обробники в MainActivity) виконувалася перевірка коректності валідації введених даних, відправки їх у Firebase Firestore та їх подальшого відображення в RecyclerView списку завдань.

Під час ручного тестування було виявлено декілька помилок. Однією з них була некоректна обробка стану полів форми після повороту екрану пристрою. Цю проблему було вирішено за допомогою ViewModel, яка дозволила зберігати стан введених даних незалежно від життєвого циклу активності.

Також було зафіксовано затримку при оновленні списку завдань після редагування. Проблема виникала через те, що адаптер RecyclerView не оновлювався коректно при зміні даних. В результаті налагодження до функції getTasks() у MainActivity було додано перевірку isFirstGetTasks, яка відрізняє першу ініціалізацію адаптера від подальшого оновлення списку завдань (ліст. 3.17). Такий підхід дозволив уникнути дублювання завдань у списку.

Лістинг 3.17 – Оновлення завдань після першого запуску

```
if (isFirstGetTasks) {
    initRecyclerViewTasks(tasks)
    isFirstGetTasks = false
} else {
    taskAdapter.updateTasks(tasks)

}
```

кінець лістингу 3.17

Мінімальні системні вимоги для стабільної роботи застосунку були визначені на основі специфікації Android-пристроїв, підтримуваних Firebase SDK. Мінімальна версія операційної системи – Android 8.0 (API level 26). Рекомендовані параметри: Android 10+, 2 ГБ оперативної пам’яті, доступ до Інтернету для синхронізації з Firebase, а також наявність облікового запису Google на пристрої для роботи з Firebase Authentication.

У межах налагодження було також реалізовано систему виведення повідомлень користувачу у вигляді Toast, що дозволило оперативно інформувати про результат виконання дій, як-от успішна авторизація, помилка під час видалення або оновлення завдання (ліст. 3.18).

Лістинг 3.18 – Виведення повідомлення користувачу після оновлення завдання

```
showToast(R.string.task_updated_successfully)
```

кінець лістингу 3.18

Додатково у проєкті були реалізовані три підпункти, які демонструють специфіку створення системи.

По-перше, було впроваджено рольову систему, яка обмежує або відкриває функціонал залежно від типу користувача. Наприклад, лише адміністратор має доступ до створення нових завдань або редагування існуючих. Це реалізовано на рівні View (btnAddTask.visibility = View.VISIBLE) та логіки роботи з Firebase (ліст. 3.19).

Лістинг 3.19 – Відображення кнопки додавання завдань для адміністратора

```
binding.btnAddTask.visibility =
    if (user.role == Role.ADMIN) View.VISIBLE else View.INVISIBLE
```

кінець лістингу 3.19

По-друге, для оптимізації повторного доступу до даних користувача було застосовано SharedPreferences. Це дало змогу зменшити кількість запитів до Firebase та підвищити швидкодію інтерфейсу.

По-третє, було впроваджено кастомні спливаючі меню (PopupWindow) (рис. 3.11), які відкриваються в контексті елемента керування та містять функції редагування, видалення, копіювання тощо.

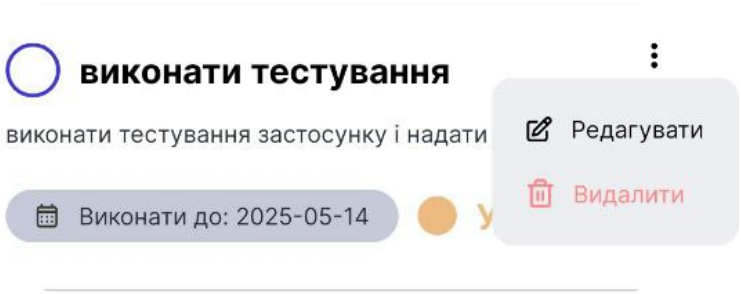

Рисунок 3.11 – Спливаюче меню

Це значно покращило UX, зробивши керування завданнями зручнішим та інтуїтивно зрозумілим (ліст. 3.20).

Лістинг 3.20 – Відкриття кастомного контекстного меню для завдання

```
popupWindow.showAsDropDown(anchor)
```

кінець лістингу 3.20

Таким чином, у процесі тестування було виявлено та усунуто критичні недоліки, що дозволило забезпечити стабільну роботу застосунку в межах визначених системних параметрів, а також сформувати гнучку архітектуру, придатну до масштабування.

3.3 Розробка бази даних

Розробка бази даних є ключовим етапом проєктування автоматизованої системи координації завдань у командних проєктах. У рамках створення мобільного застосунку To-Do List для командної роботи, який розробляється з використанням Kotlin та Gradle Kotlin DSL, було прийнято рішення використовувати хмарну NoSQL-базу даних Firebase Firestore. Таке рішення обумовлено потребою у гнучкому та масштабованому сховищі даних, яке забезпечує інтеграцію з мобільною платформою, автоматичну синхронізацію у реальному часі, а також простоту в адмініструванні.

Firebase Firestore є документно-орієнтованою базою даних, у якій інформація організована у вигляді колекцій і документів. У застосунку визначено чотири основні колекції: users, groups, tasks і polls (рис. 3.12).

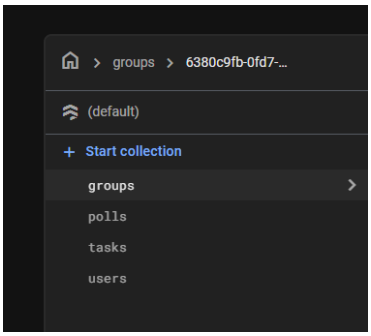

Рисунок 3.12 – Колекції бази даних

У колекції users зберігається інформація про користувачів системи, включаючи їхній унікальний ідентифікатор, ім’я, адресу електронної пошти, групу, до якої вони належать, та роль (керівник або працівник) (рис. 3.13).

Колекція groups містить дані про робочі групи, зокрема назву та опис. Завдання, призначені користувачам, зберігаються у колекції tasks, а функція опитувань реалізована через колекцію polls.

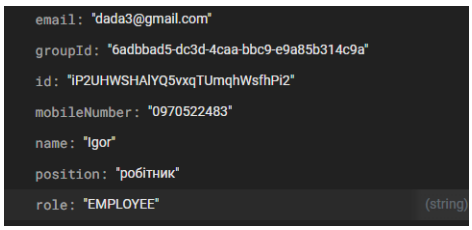

Рисунок 3.13 – Інформація користувачів

Для взаємодії з Firestore у застосунку реалізовано окремий клас FirebaseFirestoreRepository, який зосереджує всю бізнес-логіку доступу до бази даних. Застосовано підхід асинхронного програмування з використанням корутин (suspend-функції) та розширення await() для очікування результату операції (ліст. 3.21).

Лістинг 3.21 – Ініціалізація Firestore та доступ до колекцій

```
private val db: FirebaseFirestore = FirebaseFirestore.getInstance()
private fun getCollection(collectionName: String) =
db.collection(collectionName)
private val collectionUsers = "users"
private val collectionGroups = "groups"
private val collectionTasks = "tasks"
private val collectionPolls = "polls"
```

кінець лістингу 3.21

Методи цього класу дозволяють виконувати повний спектр CRUD-операцій: додавання, оновлення, отримання та видалення користувачів, груп, завдань і опитувань. Наприклад, для збереження нового користувача у базі даних використовується функція addUser, яка звертається до відповідної колекції та виконує операцію set() над документом з ID користувача (ліст. 3.22).

Лістинг 3.22 – Додавання користувача у Firestore

```
suspend fun addUser(user: User): Result<Unit> {
    return try {
  getCollection(collectionUsers).document(user.id).set(user).await()
        Result.success(Unit)
    } catch (e: FirebaseFirestoreException) {
```

```
        Result.failure(Exception("Failed to save user to Firestore: ${e.message}"))
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

кінець лістингу 3.22

Аналогічно, для зчитування списку завдань конкретної групи реалізовано функцію getListOfTasks, у якій здійснюється фільтрація за groupId та сортування за полем verified (ліст. 3.23).

Лістинг 3.23 – Отримання списку завдань групи

```
suspend fun getListOfTasks(groupId: String): Result<List<Task>> {
    return try {
        val document =
            getCollection(collectionTasks)
                .orderBy("verified", Query.Direction.ASCENDING)
                .whereEqualTo("groupId", groupId).get().await()
        val tasks = document.documents.mapNotNull { it.toObject(Task::class.java) }
        Result.success(tasks)
    } catch (e: FirebaseFirestoreException) {
        Result.failure(Exception("Failed to get task from Firestore: ${e.message}"))
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

кінець лістингу 3.23

Реалізація аутентифікації здійснюється у класі FirebaseAuthRepository, який забезпечує вхід користувача, реєстрацію та відновлення пароля. Всі методи працюють асинхронно за допомогою suspend-функцій (ліст. 3.24).

Лістинг 3.24 – Реєстрація користувача через Firebase Authentication

```
suspend fun registerWithEmailAndPassword(email: String, password: String): Result<String> {
    try {
```

```
        val authResult = auth.createUserWithEmailAndPassword(email,
password).await()
        val userId = authResult.user?.uid
        if (userId == null) {
            return Result.failure(Exception("Registration failed: User ID
is null"))
        }
        return Result.success(userId)
    } catch (e: FirebaseAuthException) {
        return Result.failure(Exception("Registration failed:
${e.message}"))
    } catch (e: Exception) {
        return Result.failure(e)
    }
}
```

кінець лістингу 3.24

Таким чином, реалізована база даних забезпечує централізоване зберігання всієї необхідної інформації – про користувачів, робочі групи, завдання та опитування. Дані пов’язані між собою через унікальні ідентифікатори. Зокрема, користувачі належать до груп, завдання – до конкретної групи та конкретного виконавця, а опитування – до групи, яка їх ініціює. Така структура дозволяє ефективно фільтрувати, групувати та обробляти інформацію без складних транзакцій, характерних для реляційних БД, що відповідає ідеології Firebase Firestore.

Обрана архітектура спрощує масштабування, забезпечує зручність супроводу та мінімізує потребу у серверній логіці, що є важливою перевагою у мобільному застосунку з обмеженими ресурсами пристроїв. У поєднанні з Kotlin DSL та модульною структурою, така база даних є сучасним, гнучким і продуктивним рішенням для командної взаємодії у рамках To-Do List застосунку.

3.4 Захист інформаційно-комп’ютерної системи

Під час розробки автоматизованої системи координації завдань у командних проєктах особливу увагу було приділено забезпеченню надійного захисту даних користувачів і стійкості системи до потенційних загроз. Оскільки

застосунок орієнтований на командну роботу та обмін завданнями в межах груп, він обробляє персональні облікові дані, внутрішні задачі, результати опитувань тощо. Це зумовлює необхідність запровадження комплексного підходу до захисту інформаційно-комп’ютерної системи.

У розробленому мобільному застосунку реалізовано автентифікацію користувачів із використанням платформи Firebase Authentication. Вона забезпечує вхід на основі електронної пошти та пароля, а також керує сесією користувача, автоматично формуючи токен для підтвердження автентичності. Firebase використовує алгоритм bcrypt для хешування паролів, що включає адаптивний механізм складності та сіль (salt), яка унікальна для кожного користувача. Такий підхід дозволяє ефективно протистояти як brute-force атакам, так і спробам використання rainbow-таблиць. Сам хешований пароль зберігається виключно у хмарному середовищі, а застосунок має доступ лише до результату перевірки автентичності, що виключає компрометацію даних на клієнтській стороні.

Передача даних між клієнтською частиною та сервером Firebase відбувається через захищений канал за протоколом HTTPS, з використанням TLS-шифрування (версії 1.2 або 1.3). Це гарантує, що навіть у випадку перехоплення трафіку зловмисник не зможе розшифрувати вміст повідомлень, оскільки кожне з’єднання встановлюється із застосуванням криптографічного обміну ключами. Крім того, Firebase має вбудовані механізми захисту від перевантаження серверу запитами, які дозволяють ефективно протидіяти DDoS-атакам. Зокрема, система автоматично обмежує частоту звернень від одного користувача або пристрою, а також блокує підозрілу активність.

З метою унеможливлення зворотної інженерії застосунок оптимізовано та захищено за допомогою інструменту ProGuard. Він здійснює обфускацію коду, замінюючи імена змінних та класів випадковими послідовностями, а також видаляє невикористані компоненти. У файлі build.gradle.kts для релізної збірки активовано правила обфускації, що суттєво ускладнює процес декомпіляції APK-файлу, навіть у разі його доступності третім особам. Таким чином,

зловмиснику буде важко відновити початкову логіку роботи програми або отримати доступ до внутрішніх механізмів авторизації.

На локальному рівні застосунок зберігає лише мінімально необхідні дані, наприклад, адресу електронної пошти користувача, у сховищі SharedPreferences. Важливо, що жодні паролі не зберігаються у відкритому вигляді, а вся конфіденційна інформація залишається на сервері. За потреби можливе підключення зашифрованого сховища EncryptedSharedPreferences, яке базується на AES-шифруванні з 256-бітним ключем, що відповідає сучасним стандартам безпеки.

Система також реалізує модель контролю доступу на основі ролей, згідно з якою користувачі можуть мати різні рівні повноважень (наприклад, керівник команди або звичайний учасник). Ці ролі визначають, які дії користувач може виконувати в інтерфейсі, включаючи створення, редагування або видалення завдань. Окрім цього, у Firebase налаштовані правила безпеки (Firebase Security Rules), які обмежують доступ до бази даних виключно автентифікованим користувачам, перевіряючи їхню унікальну ідентифікацію (UID).

У результаті впровадження зазначених заходів система координації завдань виявляється захищеною від основних загроз, характерних для сучасних мобільних застосунків. Вона демонструє високий рівень стійкості до зовнішніх атак, забезпечує конфіденційність і цілісність даних, а також створює безпечне середовище для спільної роботи команди.

3.5 Розробка інсталяційного пакета

У процесі створення автоматизованої системи координації завдань для командних проєктів було реалізовано повноцінний інсталяційний пакет, який забезпечує просту і надійну інсталяцію застосунку на пристрої з операційною системою Android. Для цього використовувалася стандартна технологія Android Package – APK, що є контейнером для всієї необхідної функціональності,

включно з компільованим кодом, ресурсами інтерфейсу, графікою, конфігураційними файлами, цифровим підписом та бібліотеками.

Процес формування інсталяційного пакета керується за допомогою інструменту Gradle із використанням Kotlin DSL, де у файлі build.gradle.kts описуються всі необхідні параметри збірки. У цьому файлі задається структура проєкту, визначається версія застосунку, рівень API, умови обфускації коду для релізної версії, а також конфігурація підпису застосунку. Збірка APK виконується через команду assembleRelease, яка компілює застосунок у готовий до встановлення файл із цифровим підписом. Підпис є обов'язковою умовою для розповсюдження застосунку через Google Play або для встановлення на будь-який реальний пристрій, оскільки гарантує справжність і цілісність додатку.

Інсталяційний файл, створений у процесі збірки, містить компільований байт-код Kotlin у форматі DEX, який обробляється віртуальною машиною Android Runtime. Окрім коду, в APK включаються графічні ресурси, зокрема зображення, макети інтерфейсу та файли локалізації, які використовуються системою Android для відображення застосунку на екрані. Конфігурація додатку, дозволи, список доступних активностей, служб і приймачів подій задаються у спеціальному файлі AndroidManifest.xml, який також входить до складу інсталяційного пакета.

Системного реєстру в Android у звичному для Windows вигляді не існує. Замість цього застосунок зберігає конфігурацію користувача у внутрішньому сховищі пристрою. Для цього використовується механізм SharedPreferences, який дозволяє зберігати прості налаштування у вигляді пар «ключ-значення». Наприклад, після входу в систему застосунок може зберігати адресу електронної пошти або токен автентифікації, щоб забезпечити автоматичне відновлення сесії без повторного введення даних. Ці налаштування зберігаються у внутрішній директорії пристрою, недоступній для інших застосунків, що забезпечує базовий рівень безпеки.

Під час встановлення APK на пристрій операційна система автоматично створює ярлик застосунку на головному екрані або у списку всіх програм. Це

відбувається завдяки тому, що головна активність програми оголошена у маніфесті з відповідними категоріями, які сигналізують системі про наявність точки запуску. Графічна іконка застосунку визначається у ресурсах проєкту, які адаптовані до різних щільностей екранів за допомогою декількох версій зображень (рис. 3.14).

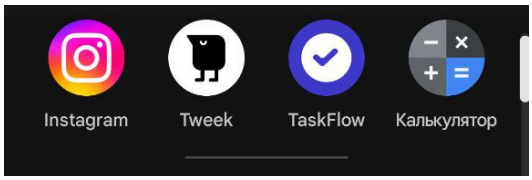

Рисунок 3.14 – Іконка застосунку TaskFlow

Система реєстрації користувачів реалізована через інтеграцію з Firebase Authentication. Під час першого запуску користувач має змогу створити обліковий запис або ввійти до вже існуючого. Після успішної автентифікації інформація про користувача зберігається у хмарі, а локально – лише службові дані для підтримки сесії. Це забезпечує як зручність використання, так і безпеку обробки персональної інформації.

Таким чином, інсталяційний пакет застосунку повністю відповідає стандартам Android і включає в себе всі необхідні компоненти для коректного встановлення, безпечної роботи та захисту даних. Його структура чітко визначена й автоматизована за допомогою Gradle, що дає змогу швидко генерувати як тестові, так і релізні збірки, придатні до розповсюдження або безпосереднього встановлення на пристрої кінцевих користувачів.

Висновки до розділу 3

Було здійснено повноцінну реалізацію мобільного застосунку TaskFlow, призначеного для координації завдань у командних проєктах. Основна увага приділялась практичній реалізації ключових компонентів системи: авторизації, ролей користувачів, створення та редагування завдань, а також їх збереження у

Firebase Firestore. Реалізацію виконано з використанням сучасного технологічного стеку – Kotlin, MVVM-архітектури, Jetpack Compose для побудови інтерфейсу та Firebase для хмарного збереження й автентифікації.

Застосунок підтримує рольову модель із розмежуванням прав доступу між керівником і виконавцем, що дозволяє ефективно делегувати задачі й контролювати їх виконання. Інтерфейс реалізовано з урахуванням принципів Material Design, а логіка управління UI – за допомогою ViewModel та ViewBinding. Завдяки цьому вдалося досягти адаптивності, простоти використання та гнучкості налаштувань.

Особливу увагу приділено інтеграції з Firebase: реалізовано безпечну авторизацію, зберігання користувацьких ролей та задач, синхронізацію даних, а також підтримку офлайн-режиму. Функціональність застосунку доповнена механізмами зворотного зв’язку, перевіркою введених даних, а також повідомленнями про результат дій користувача.

Проведене тестування виявило низку помилок, які було усунуто у процесі налагодження, зокрема щодо збереження стану форм, оновлення інтерфейсу після редагування завдань, а також відображення помилок авторизації. Завдяки використанню ViewModel, корутин і адаптерів RecyclerView вдалося забезпечити стабільну роботу застосунку навіть за змін умов середовища.

Таким чином, реалізована інформаційно-комп’ютерна система відповідає сучасним вимогам до мобільного програмного забезпечення: вона є зручною у використанні, функціонально повною, масштабованою та безпечною. Розробка продемонструвала практичну реалізованість поставлених у проєкті цілей і підтвердила ефективність обраного архітектурного та технологічного підходу.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було реалізовано повний цикл розробки автоматизованої системи координації завдань у командних проєктах із використанням технологій Kotlin та Gradle Kotlin DSL для платформи Android.

На початковому етапі реалізації проєкту було проведено детальний аналіз сучасних інструментів для управління завданнями в команді. Розглянуто популярні системи Trello, Asana та ClickUp. Виявлено, що більшість із них орієнтовані на великі команди, мають складний інтерфейс, обмежену офлайн-функціональність і не реалізовані як нативні Android-застосунки на Kotlin. Крім того, використання кросплатформених технологій (React Native, Flutter) негативно впливає на продуктивність і гнучкість мобільних додатків. Проведений патентний пошук підтвердив відсутність аналогів, які б поєднували нативну розробку на Kotlin, підтримку офлайн-режиму, простий інтерфейс і рольову модель «керівник – виконавець». Це підтвердило доцільність створення нового застосунку, орієнтованого на мобільні команди малого та середнього розміру, із простим UX, розмежуванням ролей, хмарною синхронізацією та високою продуктивністю на Android.

Далі було сформульовано як функціональні, так і нефункціональні вимоги до майбутньої системи. Основні функціональні можливості охоплюють процес реєстрації та авторизації користувачів із розмежуванням ролей на керівника та учасників команди. Керівник отримує повний контроль над створенням, редагуванням і призначенням завдань, визначенням їхніх пріоритетів і термінів виконання, а також може відстежувати поточний статус виконання. Учасники, зі свого боку, мають доступ до призначених їм завдань і можуть оновлювати їхній стан відповідно до прогресу. Система також передбачає персоналізовану взаємодію з користувачем, зокрема фільтрацію завдань і відображення лише актуальної інформації для кожного облікового запису. Нефункціональні вимоги зосереджені на забезпеченні зручності та стабільності використання. Особлива увага приділена побудові інтуїтивного інтерфейсу відповідно до принципів

Material Design, підтримці адаптивного відображення на різних пристроях, а також реалізації безпечного обміну й зберігання даних із використанням сучасних засобів захисту. Важливим критерієм стала підтримка офлайн-режиму, що дозволяє працювати із застосунком без постійного підключення до мережі, з подальшою автоматичною синхронізацією інформації після його відновлення. Таке поєднання вимог забезпечує не лише функціональну повноту, а й високу якість користувацького досвіду та надійність у різних умовах використання.

На основі визначених вимог була розроблена архітектура мобільного застосунку, що враховує рольову модель користувачів із чітким розмежуванням функціональності для керівника та виконавця. Структура проєкту побудована за модульним принципом, що забезпечує зручність підтримки та можливість масштабування. Для зберігання даних застосовано зв’язку Firebase Firestore та Room: перша використовується для хмарної синхронізації та обміну даними між користувачами, друга – для локального збереження, що дозволяє працювати із застосунком навіть без доступу до мережі. Система автоматично виконує синхронізацію при відновленні інтернет-з’єднання, що є критично важливим для стабільної роботи в мобільному середовищі.

Реалізовано мобільний застосунок з використанням мови програмування Kotlin, що забезпечило нативну продуктивність платформи Android. У процесі розробки застосовано інструменти Gradle Kotlin DSL для ефективного керування проєктом, Firebase Authentication і Firestore для надійної авторизації користувачів та обміну даними в режимі реального часу. Для побудови сучасного, адаптивного інтерфейсу відповідно до принципів UX/UI дизайну використано Jetpack Compose. Архітектурні компоненти Jetpack застосовано для підтримки життєвого циклу, стабільності та масштабованості застосунку, що забезпечило зручність використання та високу якість користувацького досвіду.

Забезпечено повний основний функціонал: авторизація, створення, редагування, делегування завдань, відображення статусу й пріоритетів, фільтрація, пошук, а також моніторинг виконання завдань. Система підтримує рольову модель і логіку взаємодії між керівником та учасниками команди.

Також реалізовано локальне збереження даних із використанням SQLite (через Room), що дозволяє працювати із застосунком без доступу до мережі, а при його появі – виконувати автоматичну синхронізацію з хмарним сховищем. Це забезпечує збереження даних, підвищує стабільність роботи та відповідає сучасним вимогам до мобільних рішень.

Після завершення етапу реалізації було проведено тестування мобільного застосунку з метою перевірки відповідності функціоналу попередньо визначеним вимогам. Основну увагу зосереджено на перевірці коректності авторизації, створення та редагування завдань, їх призначення, зміни статусів, а також роботи системи в офлайн-режимі з подальшою синхронізацією даних. У процесі тестування виявлено низку незначних помилок, зокрема некоректне відображення статусів у деяких сценаріях, а також дрібні збої в оновленні інтерфейсу після зміни даних. Ці помилки були оперативно усунуті шляхом доопрацювання логіки ViewModel та оновлення адаптерів RecyclerView. У результаті застосунок продемонстрував відповідність усім основним критеріям якості: зручності користування, стабільності функціонування, адаптивності інтерфейсу та ефективності виконання базових операцій. Тестування підтвердило надійну роботу системи як у звичайному режимі, так і при зміні мережевого підключення, що особливо важливо для мобільних користувачів.

Таким чином, у результаті виконання кваліфікаційної роботи було створено нативний мобільний застосунок для координації завдань у команді, орієнтований на потреби малих і середніх проєктів. Реалізовано повний функціонал: авторизація, управління завданнями, розмежування ролей, офлайн-режим і синхронізація з Firebase. Тестування підтвердило відповідність вимогам щодо зручності, стабільності та ефективності. Застосунок забезпечує надійну роботу як онлайн, так і без підключення до мережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Agile. Task Management Systems for Agile Software Development. URL: https://thesai.org/Publications/ViewPaper?Volume=12&Issue=5&Code=IJACSA&SerialNo=72 (дата звернення: 18.01.2025).

2. Kotlin or Java. A Comparative Study for Android Applications. URL: https://thesai.org/Publications/ViewPaper?Volume=11&Issue=9&Code=IJACSA&SerialNo=52 (дата звернення: 18.01.2025).

3. Collaborative Mobile Applications: Requirements, Challenges, and Design Considerations. URL: https://online-journals.org/index.php/i-jim/article/view/22071 (дата звернення: 18.01.2025).

4. US8146104B2 – System and method for programmatically generating to-do list and creating notification between calendar and other applications. URL: https://patents.google.com/patent/US8146104B2 (дата звернення: 25.02.2025).

5. US10977621B2 – Action-based to-do list. URL: https://patents.google.com/patent/US10977621B2 (дата звернення: 25.02.2025).

6. US8806613B2 – Intelligent task assignment and authorization systems and methods. URL: https://patents.google.com/patent/US8806613B2/en (дата звернення: 25.02.2025).

7. US7596416B1 – Project management tool. URL: https://patents.google.com/patent/US7596416B1/en (дата звернення: 25.02.2025).

8. US9239719B1 – Task management system. URL: https://patents.google.com/patent/US9239719B1/en (дата звернення: 25.02.2025).

9. DjABoo vs Trello. Comparison with Trello. URL: https://djaboo.com/en/comparisons/djaboo-vs-trello/ (дата звернення: 11.03.2025).

10. Trello. URL: https://cafebazaar.ir/app/com.trello?l=en (дата звернення: 10.05.2025).

11. Asana vs Monday. Do They Work as CRM Systems? URL: https://capsulecrm.com/blog/asana-vs-monday-do-they-work-as-crm-systems/ (дата звернення: 11.03.2025).

12. Asana. URL: https://asana.com/download (дата звернення: 11.03.2025).

13. ClickUp Features. URL: https://clickup.com/features (дата звернення: 11.03.2025).

14. ClickUp. URL: https://clickup.com/download (дата звернення: 11.03.2025).

15. Методичні вказівки до виконання кваліфікаційної роботи бакалавра [Текст]: для здобувачів першого (бакалаврського) рівня вищої освіти освітньої програми «Інженерія програмного забезпечення» галузі знань 12 Інформаційні технології спеціальності 121 Інженерія програмного забезпечення денної та заочної форм навчання / уклад. Н. М. Ліщина, А. А. Ящук. Луцьк: ЛНТУ, 2025. 56 с.

ДОДАТКИ

Додаток А

Метод showAlertDialogTask()

```
private fun showAlertDialogTask(selectedTask: Task) {

    val builder = AlertDialog.Builder(this)

    val view = layoutInflater.inflate(R.layout.dialog_task_info, null)

    builder.setView(view)

    val dialog = builder.create()

dialog.window?.setBackgroundDrawable(ColorDrawable(Color.TRANSPARENT))

    var tvTitle = view.findViewById<AppCompatTextView>(R.id.tvTitle)

    var spinnerStatus =
view.findViewById<PowerSpinnerView>(R.id.spinnerStatus)

    tvTitle.text = selectedTask.title

    val itemsStatus = arrayListOf(

        IconSpinnerItem(R.drawable.shape_1,
getString(R.string.not_started)),

        IconSpinnerItem(R.drawable.shape_2,
getString(R.string.in_progress)),

        IconSpinnerItem(R.drawable.shape_3,
getString(R.string.completed)),

        IconSpinnerItem(R.drawable.shape_4,
getString(R.string.under_review))

    )

  spinnerStatus.setSpinnerAdapter(IconSpinnerAdapter(spinnerStatus))

    spinnerStatus.setItems(itemsStatus)

    val selectedIndex = when (selectedTask.status) {

        Status.NOT_STARTED -> 0

        Status.IN_PROGRESS -> 1

        Status.COMPLETED -> 2
```

```
        Status.UNDER_REVIEW -> 3

    }

    spinnerStatus.selectItemByIndex(selectedIndex)

    var selectedStatus = itemsStatus[selectedIndex].text.toString()

    spinnerStatus.setOnSpinnerItemSelectedListener<IconSpinnerItem> { _,
_, _, newItem ->

        selectedStatus = newItem.text.toString()

    }

view.findViewById<AppCompatButton>(R.id.btnSave).setOnClickListener {

        selectedTask.status = Status.fromString(this, selectedStatus)

        updateTask(selectedTask, dialog)

    }

    dialog.show()

}
```