МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ПЕРСОНАЛЬНОГО ОРГАНАЙЗЕРА ЗАДАЧ ДЛЯ ПЛАТФОРМИ ANDROID»**

Виконав: студент 4 курсу, групи 6.1210-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.В. Левданський
(ініціали та прізвище)

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

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Левданському Михайлу Владиславовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка персонального органайзера задач для платформи Android

керівник роботи Горбенко Віталій Іванович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

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

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Розробка проєкту.
3. Демострація.

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

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	22.01.2024	
3.	Обробка методичних та теоретичних	23.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	09.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2024	

Студент _____ М.В. Левданський
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.І. Горбенко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка персонального органайзера задач для платформи Android»: 63 с., 36 рис., 4 табл., 9 джерел, 2 додатки.

ВИСНОВКИ, ДІАГРАМИ, МЕТА РОЗРОБКИ, ПРИКЛАДИ КОДУ, СКРІНШОТИ РОБОТИ ДОДАТКУ, ANDROID, REALM, ORGANISER, FIREBASE, ADMOD, GRADLE, KOTLIN.

Об’єкт дослідження – реалізація функцій персонального органайзера в мобільному застосунку для платформи Android.

Мета роботи: виконати проєктування та реалізацію застосунку для платформи Android

Метод дослідження – аналіз літературних джерел та функціональності аналогів застосунку, експерименти щодо реалізації функціональності застосунку.

У сучасному цифровому світі персональні органайзери задач стали невід’ємною частиною повсякденного життя людей. Розробка таких додатків для платформи Android відкриває широкі можливості для зручного та ефективного керування часом та завданнями. Саме такий додаток було розроблено у цій дипломній роботі. Також було продемострovано функціональність застосунку та надано пояснення щодо певних моментів розробки.

Розробка персонального органайзера задач для Android вимагає ретельного аналізу потреб користувачів, вивчення їх вимог та вподобань. Процес включає в себе проєктування зручного та інтуїтивно зрозумілого інтерфейсу, розробку функціоналу для створення, редагування та видалення завдань, а також можливості налаштування нагадувань. Для цього було обрано середовище розробки Android Studio, створено шаблони користувацького інтерфейсу, та проведено аналіз існуючих рішень У роботі було проведено всі необхідні розрахунки і проєктування для досягнення поставленої мети.

SUMMARY

Bachelor's qualifying paper "Development of a Personal Task Organizer for the Android": 63 pages, 36 figures, 4 tables, 9 references, 2 supplements.

CONCLUSIONS, DIAGRAMS, DEVELOPMENT GOAL, CODE EXAMPLES, APP WORKING SCREENSHOTS, ANDROID, REALM, ORGANISER, FIREBASE, ADMOD, GRADLE, KOTLIN.

The object of the study is implementation of personal organizer functions in a mobile application for the Android platform.

The aim of the study is to design and implement an application for the Android platform.

The methods of research are analysis of literary sources and functionality of application analogs, experiments on implementation of application functionality.

In today's digital world, personal task organizers have become an integral part of people's everyday lives. The development of such applications for the Android platform opens up wide opportunities for convenient and effective time and task management. Such application was developed in this thesis. The functionality of the application was also demonstrated and explanations were provided regarding certain aspects of the development.

Development of a personal task organizer for Android requires careful analysis of user needs, study of their requirements and preferences. The process includes designing a convenient and intuitive interface, developing functionality for creating, editing and deleting tasks, as well as the ability to set reminders. For this, the Android Studio development environment was chosen, user interface templates were created, and existing solutions were analyzed. All the necessary calculations and design were carried out in the work to achieve the set goal.

ЗМІСТ

ВСТУП

У сучасному світі швидкість життя стрімко зростає, а потреба в ефективному управлінні часом стає все більш актуальною. Потік інформації, розмаїття обов'язків та завдань вимагають від людини надзвичайної організованості та систематичного підходу до розподілу свого часу.

У цьому контексті велике значення мають інструменти, які допомагають структурувати щоденні справи, нагадувати про важливі події та ефективно керувати часом.

Цією дипломною роботою пропонується розробка персонального органайзера задач для платформи Android, що має на меті створення зручного та функціонального інструменту для керування щоденними обов'язками та планами користувачів. Розроблений персональний органайзер задач буде спрощувати процес планування, нагадування та виконання завдань, що дозволить користувачам більш ефективно використовувати свій час та досягати поставлених цілей.

У рамках цієї роботи буде проведено аналіз існуючих рішень у сфері організації робочого часу, визначено функціональні вимоги до персонального органайзера задач та розроблено програмний продукт, що задовольняє встановлені критерії ефективності та зручності використання. Такий підхід сприятиме покращенню продуктивності користувачів та підвищенню їхнього рівня організованості.

Ця дипломна робота складається з таких основних розділів: аналіз існуючих рішень на ринку, формулювання вимог до програмного забезпечення, опис архітектури та розробка програмного продукту, тестування та валідація розробленого органайзера. В кінці роботи буде надано висновки щодо отриманих результатів та визначено можливі напрямки подальшого розвитку розробленого продукту.

Ця робота спрямована на підвищення рівня організованості та продуктивності користувачів за допомогою створення ефективного та зручного інструменту для керування задачами та планами у сфері особистого та професійного життя.

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧ

1.1 Аналіз існуючих рішень

Для того щоб розуміти чи не буде створений додаток клоном вже існуючого за функціоналом чи дизайном, і для того щоб розуміти які особливості створеного мною додатку зможуть надати перевагу на ринку, треба провести аналіз конкурентоспроможності, або інакше кажучи дослідити додатки-аналоги.

Для проведення цього аналізу було обрано 4 аналоги додатків які дозволяють організовувати задачі:

– власний додаток: Universal Task Organiser;
– аналог 1: Todoist;
– аналог 2: Microsoft To Do;
– аналог 3: Google Tasks;
– аналог 4: Any.do.

Після чого необхідно описати кожний аналог та створюваний мною додаток за критеріями:

– назва, компанія-виробник;
– основні функціональні можливості;
– переваги;
– недоліки.

Опишемо обрані мною додатки.

Назва, компанія-виробник: **kerumit bsl**.

Основні функціональні можливості:

– створення завдань та списків завдань;
– планування та призначення термінів виконання;
– можливість додавати медіа та іншу додаткову інформацію;
– можливість створювати лічильники;

– можливість додавати підзадачі та коментарі до завдань.

Переваги:

– простий та інтуїтивно зрозумілий інтерфейс;
– дуже різноманітний функціонал;
– розширені функції, такі як нагадування про терміни, підтримка файлів тощо.

Недоліки:

– відсутні.

Назва, компанія-виробник: **doist, inc**.

Основні функціональні можливості:

– створення завдань та списків завдань;
– планування та призначення термінів виконання;
– спільний доступ до списків завдань для співробітників та можливість призначення завдань;
– можливість додавати підзадачі та коментарі до завдань.

Переваги:

– простий та інтуїтивно зрозумілий інтерфейс;
– можливість синхронізації між різними пристроями;
– розширені функції у версії premium, такі як нагадування про терміни, підтримка файлів тощо.

Недоліки:

– обмежена кількість функцій у безкоштовній версії;
– деякі функції доступні лише за підпискою.

Назва, компанія-виробник: **microsoft corporation**.

Основні функціональні можливості:

– створення завдань та списків завдань;
– планування та призначення термінів виконання;
– інтеграція зі службами microsoft, такими як outlook, exchange, microsoft teams тощо;
– можливість додавати завдання з електронної пошти та з програми

outlook.

Переваги:

– інтеграція з екосистемою microsoft;
– можливість спільного доступу до списків завдань для користувачів office 365;
– безкоштовний доступ для користувачів outlook та office 365.

Недоліки:

– можливість інтеграції обмежена до продуктів microsoft;
– менша кількість розширених функцій порівняно з іншими аналогами.

Назва, компанія-виробник: **google llc.**

Основні функціональні можливості:

– створення завдань та списків завдань;
– планування та призначення термінів виконання;
– інтеграція з gmail, google календарем та google диском;
– можливість додавати завдання з gmail та google календаря.

Переваги:

– інтеграція з екосистемою google;
– безкоштовний доступ для користувачів облікового запису google;
– простий та зрозумілий інтерфейс.

Недоліки:

– обмежена функціональність порівняно з іншими аналогами;
– відсутність деяких розширених функцій, таких як підтримка файлів.

Назва, компанія-виробник: **any.do.**

Основні функціональні можливості:

– створення завдань та списків завдань;
– планування та призначення термінів виконання;
– можливість додавати співробітників та надавати їм доступ до завдань;
– інтеграція зі сторонніми сервісами, такими як google календар, slack, amazon alexa тощо.

Переваги:

– можливість спільного доступу до списків завдань для користувачів;
– розширені функції в платній версії, такі як нагадування про терміни, підтримка групової роботи тощо.

Недоліки:

– деякі розширені функції доступні лише за підпискою;
– можливості інтеграції обмежені до платформ та сервісів, підтримуваних розробником.

Після цього було обрано 5 класифікацій за якими будуть порівнюватись всі додатки, та проводитись інтегральне порівняння аналогів та свого програмного забезпечення за методом Сааті.

Метод Сааті, або аналіз ієрархій (Analytic Hierarchy Process, AHP), розроблений математиком Томасом Сааті, є методом прийняття рішень, який допомагає ранжувати та порівнювати альтернативи за складними критеріями. Основна ідея полягає у розбитті складних рішень на більш прості частини, порівнянні їх за вагомістю та важливістю, а потім об’єднанні результатів в остаточне рішення.

Процес методу Сааті включає наступні етапи:

а) створення ієрархії критеріїв та альтернатив;
б) парне порівняння кожної пари критеріїв або альтернатив за їх важливістю на шкалі;
в) розрахунок вагових коефіцієнтів для критеріїв та альтернатив на основі порівнянь;
г) обчислення консистентності та перевірка коректності вибору.

Метод Сааті широко використовується в управлінні, бізнесі, науці та інших сферах для прийняття об’єктивних рішень при розгляді різних факторів та аспектів. Цей метод дозволяє систематизувати та узгодити різні точки зору, що допомагає зробити кращий вибір у складних ситуаціях, отже:

а) інтерфейс та користування:
    1) universal task organiser:

- простий та інтуїтивно зрозумілий інтерфейс;
- швидкий доступ до функцій завдяки зручному дизайну;
- елегантний та зручний інтерфейс;

2) todoist:
   - простий та інтуїтивно зрозумілий інтерфейс;
   - швидкий доступ до функцій завдяки зручному дизайну;
3) microsoft to do:
   - інтерфейс, який відповідає стандартам дизайну microsoft;
   - легка інтеграція з іншими продуктами microsoft;
4) google tasks:
   - мінімалістичний та простий інтерфейс;
   - інтеграція з іншими сервісами google для швидкого доступу;
5) any.do:
   - елегантний та зручний інтерфейс;
   - великий вибір кастомізації та персоналізації;

б) функціональні можливості безкоштовної версії:

1) universal task organiser:
   - усі функції доступні у безкоштовній версії;
   - будь-які обмеження відсутні;
2) todoist:
   - обмежений набір функцій, включаючи кількість списків та завдань;
3) microsoft to do:
   - доступ до основних функцій, але обмежена інтеграція з іншими сервісами microsoft;
4) google tasks:
   - безкоштовний доступ до основних функцій, але обмежена інтеграція з іншими сервісами google;
5) any.do:

- доступ до основних функцій, але обмежена кількість списків та завдань;

в) можливості спільного доступу та співпраці:

1) universal task organiser:
   - наразі знаходяться на етапі розробки;
2) todoist:
   - можливість спільного доступу та співпраці для користувачів платної версії;
3) microsoft to do:
   - інтеграція зі службами office 365 дозволяє спільно працювати над завданнями;
4) google tasks:
   - обмежена можливість спільного доступу, в основному обмін завданнями через gmail;
5) any.do:
   - можливість спільно працювати над завданнями та списками для користувачів платної версії;

г) інтеграція з іншими сервісами та платформами:

1) universal task organiser:
   - наразі знаходяться на етапі розробки;
2) todoist:
   - інтеграція з багатьма сторонніми сервісами та платформами;
3) microsoft to do:
   - інтеграція з іншими продуктами microsoft, такими як outlook, office 365, teams тощо;
4) google tasks:
   - інтеграція з екосистемою google, включаючи gmail, календар та диск;
5) any.do:
   - інтеграція зі сторонніми сервісами, такими як google

календар, slack, amazon alexa тощо;

д) наявність розширених функцій у платній версії:

1) universal task organiser:
   - усі функції доступні у безкоштовній версії;
   - будь-які обмеження відсутні;
2) todoist:
   - розширені функції доступні у версії premium, включаючи нагадування, підтримку файлів тощо;
3) microsoft to do:
   - деякі додаткові функції доступні для користувачів office 365;
4) google tasks:
   - відсутність розширених функцій у безкоштовній версії;
5) any.do:
   - розширені функції доступні у платній версії, такі як нагадування про терміни, підтримка групової роботи тощо.

Для проведення інтегрального порівняння аналогів за методом Сааті [1] необхідно визначити вагу кожної з характеристик та провести порівняння кожного аналогу з урахуванням цих ваг (див. табл. 1.1).

Таблиця 1.1 – Матриця порівняння

Характе-ристика	Вага (від 1 до 5)	Todo list	Microsoft To Do	Google Tasks	Any.do	Universal Task Organiser
Інтерфейс та користування	5	4	3	4	5	5
Функціональні можливості	4	3	4	3	3	5
Спільний доступ та співпраця	3	4	4	2	4	3

Продовження таблиці 1.1

Характе-ристика	Вага (від 1 до 5)	Todo list	Microsoft To Do	Google Tasks	Any.do	Universal Task Organiser
Інтеграція з іншими сервісами	4	5	5	4	4	3
Розширені функції	3	4	3	2	4	5

На рисунку 1.1 представлено діаграму «Інтегральне порівняння за методом Сааті».

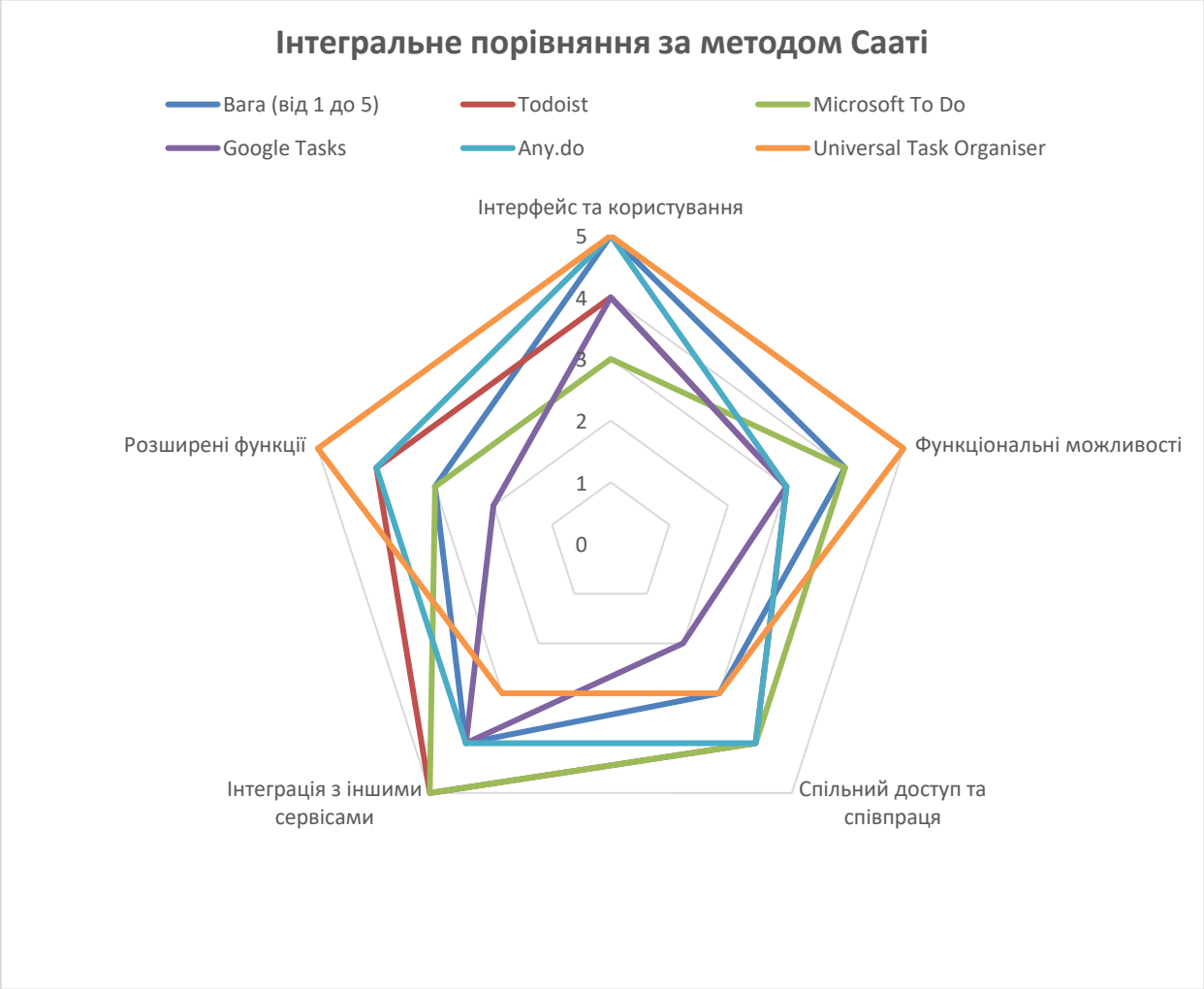


Рисунок 1.1 – Діаграма «Інтегральне порівняння за методом Сааті»

Розрахунок вагованих оцінок: для кожного аналогу необхідно розрахувати ваговану оцінку, перемноживши оцінку за кожною характеристикою на її вагу та підсумувавши всі значення:

а) universal task organiser: 5×5+5×4+3×3+3×4+5×3=81;

б) todoist: 4×5+3×4+4×3+5×4+4×3=76;

в) microsoft to do: 3×5+4×4+4×4+5×4+3×3=76;

г) google tasks: 4×5+3×4+2×3+4×4+2×3=60;

д) any.do: 5×5+3×4+4×3+4×4+4×3=77.

Після чого на основі проведених розрахунків і отримали діаграму з рисунку 1.1.

Висновок: згідно інтегрального порівняння за методом Сааті, найбільш високий рейтинг отримав Universal Task Organiser (81 бал), що вказує на його відносну перевагу у порівнянні з іншими аналогами.

1.2 Постановка задач

Виходячи з попереднього аналізу, для розробки конкурентоспроможного додатку, який буде охоче використовуватись користувачами та зможе в повну міру досягти поставлених цілей і охопить весь необхідний функціонал було створено такі задачі:

- необхідно створити базову архітектуру яка забезпечити можливість розширення проєкту у майбутньому;
- необхідно розробити зручний, інтуітивно зрозумілий інтерфейс, який не буде викликати проблем у користувачів під час використання додатку;
- необхідно розробити локальну базу даних яка дозволить зберігати усю необхідну інформацію на пристрої;
- необхідно впровадити показ реклами, для того щоб отримувати дохід від створеного додатка;

– необхідно впровадити можливість контролювати показ реклами для того щоб не створювати незручностей у користувачів.

2 ПЛАНУВАННЯ ПРОЄКТУ

2.1 Аналіз проблеми. Постановка задачі

Для того щоб зрозуміти сенс навіщо потрібно розробляти органайзер задач, і яким чином і для чого це робити, потрібно провести аналіз поставленої задачі. Для цього потрібно скласти список зацікавлених осіб. Для виявлення зацікавлених осіб необхідно розуміти наступні речі:

– користувачами системи є звичайні люди;
– замовником (покупцем) системи є власник;
– результати роботи системи не вплинуть більше ні на кого;
– оцінювати і приймати систему, коли вона буде представлена і розгорнута буде власник;
– існують інші внутрішні чи зовнішні користувачі системи, чиї потреби необхідно врахувати;
– займатися супроводом нової системи є власник;
– бенефіціаром створеної системи є власник.

Після чого необхідно провести анкетування або інтерв’ю з кожним із зацікавлених осіб. Для виявлення потреб замовника і опису об’єктів автоматизації можна проводити як анкетування, так і інтерв’ю. Найбільший ефект можна отримати при одночасному проведенні анкетування і інтерв’ю.

Анкета для опитування зацікавлених осіб включає:

а) ім’я – Михайло;
б) найменування організації – kerumit bsl;
в) найменування структурного підрозділу – відділ розробки;
г) посада – розробник;
д) посадові обов’язки – розробляти програми;
е) продукт діяльності – додатки;
ж) документи або інформація які є вхідними для діяльності – технічне

завдання, дизайн та документація;

з) документи які можна вважати вихідними – інструкція щодо використання;

и) критерії якими вимірюється успіх діяльності – якість створеного додатка, його відповідність поставленим задачам.

Після чого необхідно скласти перелік питань для інтерв'ю. Це оцінка проблеми. По кожній проблемі потрібно вияснити наступне:

– ця проблема існує через потребність людей організовувати свої плани;
– на даний момент ця проблема вирішується таким чином – я розробляю для цього додаток;
– замовник хоче вірішити цю проблему розробивши додаток;
– замовник переслідує наступні цілі – створення додатку для заробітку на ньому.

Також необхідно з'ясувати зрозуміле використання ПК:

– навички роботи на пк у працівників дуже гарні;
– працюють з наступними типами додатків – android-додатки.

Після чого потрібно скласти документ «Опис постановки задачі (комплексу задач)» (див. табл. 2.1).

Таблиця 2.1 – Структура документу «Опис постановки задачі (комплексу задач)»

1. Проблема (проблеми), що викликала необхідність у розробці
За власним бажанням
2. Визначення цілей продукту та потреб користувачів
Цілі: збагачення власника Потреби: за бажанням користувачів
3. Зацікавлені персони та користувачі
Звичайні люди

Продовження таблиці 2.1

4. Межі системи (середовище, платформа та оточення, в якому буде працювати)
Операційна система «Android»
5. Обмеження системи (задачі, які не будуть вирішуватись)
Усі крім передбачених
6. Цільовий сегмент ринку
Всесвітньо

Визначити функціональні та нефункціональні вимоги (див. табл. 2.2).

Таблиця 2.2 – Таблиця функціональних і нефункціональних вимог

1. Нефункціональні вимоги
Додаткові речі непередбачені вимогами
2. Функціональні вимоги
Робота додатка запланованим чином

Визначити профілі користувачів та сценарії використання ними системи (див. табл. 2.3).

Таблиця 2.3 – Таблиця профілів користувачів та сценарії використання ними системи

1. Профілі користувачів
Звичайна людина
2. Сценарії використання
Створення та організація справ

2.2 Створення діаграм

Будь-який високотехнологічний продукт характеризується своїми функціональними можливостями. Забезпечення необхідної функціональності часто є досить складним завданням через потреби ринку, конкуренцію, жорсткі терміни, обмежені бюджети, тощо. Однак успіх системи цілковито залежить від НВС.

Роль архітектури – це забезпечення НВС на рівні архітектурних блоків: компонентів, з'єднувачів, конфігурації. Ефективність – це якість, яка відображає здатність програмного забезпечення(далі ПЗ) до задоволення вимог продуктивності при одночасній мінімізації використання його ресурсів. Складність вказує до якої міри ПЗ або один з його компонентів, містить проєктні рішення чи реалізацію, які важко зрозуміти і перевірити.

Масштабованість ПЗ – це можливість системи бути зміненою з урахуванням нових вимог. Неоднорідність – це якість ПЗ, що передбачає, що ПЗ складається з декількох різнорідних компонентів або функціонує в декількох різнорідних обчислювальних середовищах одночасно.

Пристосовуваність – є здатністю ПЗ до задоволення нових вимог і пристосовуватися до нових умов роботи під час його життєвого циклу.

Надійність – це набір властивостей ПЗ, що дозволяє розраховувати, що ПЗ буде функціонувати так, як було заплановано. Функціональна схема будується з метою розуміння всіх функцій, що виконує ПЗ (див. рис. 2.1).

В UML для представлення специфікацій загальної структури програмного коду системи використовують діаграму компонентів. Діаграма компонентів забезпечує узгоджений перехід від логічного до фізичного представлення системи у вигляді програмних компонентів. Деякі компоненти можуть існувати тільки на етапі компіляції програмного коду, інші – на етапі його виконання. Основними елементами діаграми є компоненти, інтерфейси та залежності між ними, а також вона може вміщувати ключові класи, що входять до складу компонентів (див. рис. 2.2 та 2.3).

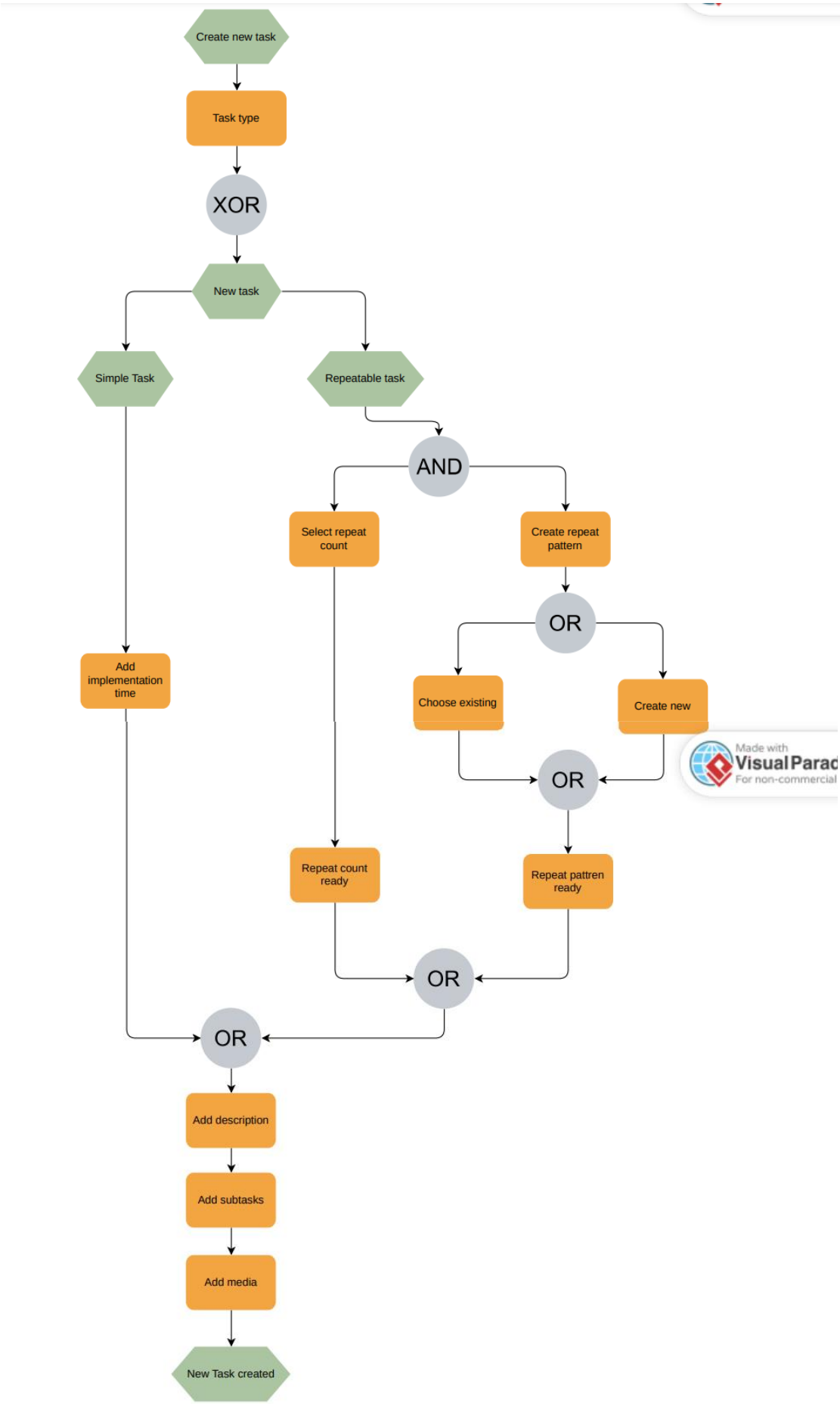


Рисунок 2.1 – Діаграма «функціональна схема архітектури ПЗ»

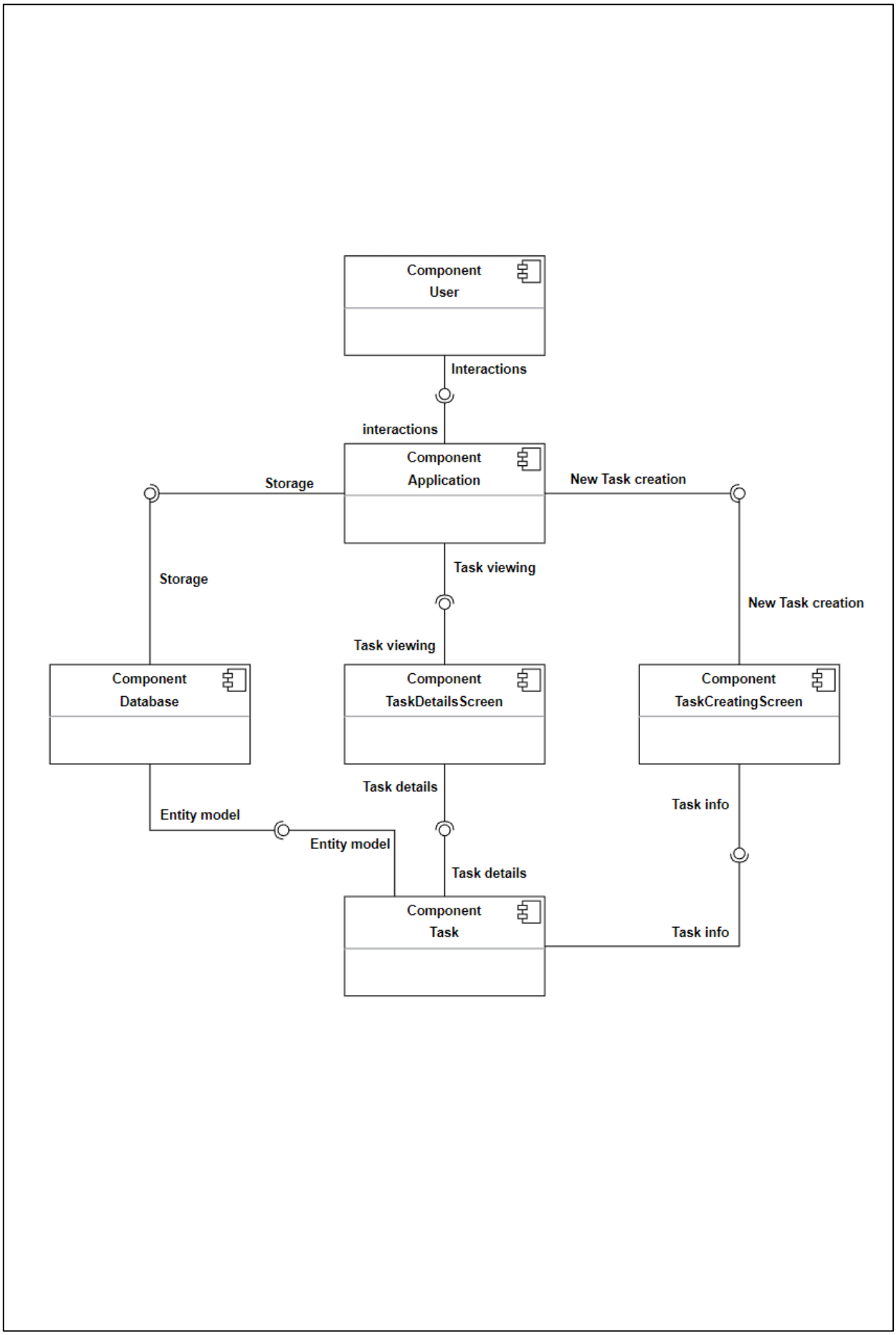


Рисунок 2.2 – Діаграма компонентів для системи

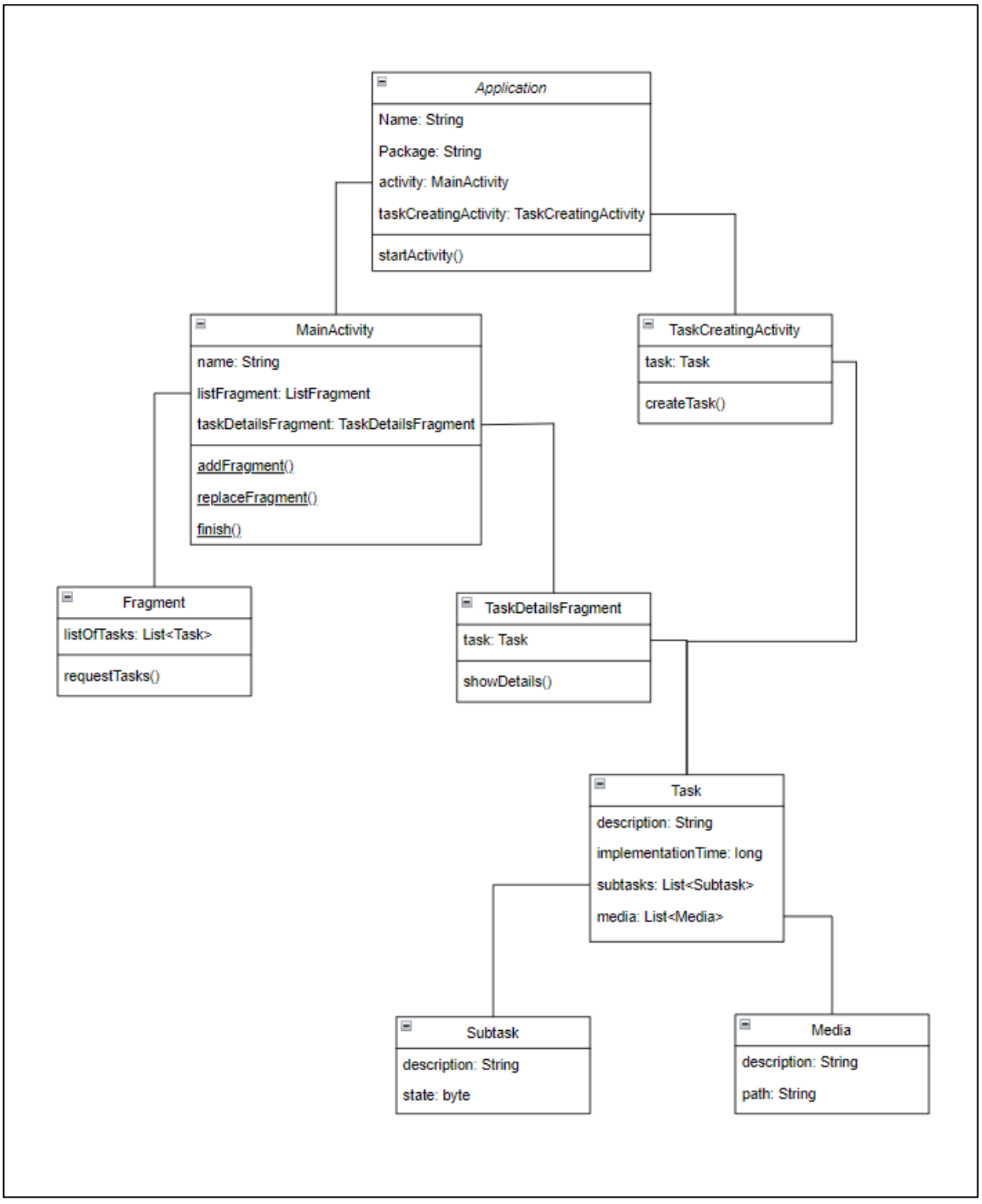


Рисунок 2.3 – Діаграма «представлення одного з структурного компонента у вигляді діаграми класів»

Крім визначення структурних елементів будь-яка архітектура визначає взаємодію між цими структурними елементами, що забезпечує бажану поведінку системи (див. рис. 2.4).

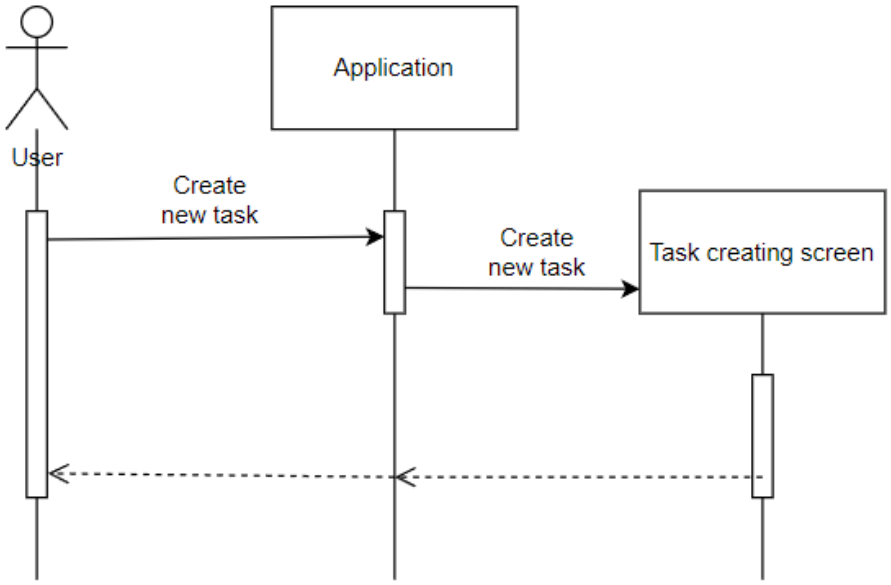


Рисунок 2.4 – Діаграма «представлення поведінки системи діаграмою послідовностей»

2.3 Архітектура додатку

Для створення проєкту було обрано шаблони проєктування Clean Architecture та MVVM (Model View ViewModel).

Clean Architecture дозволяє відокремити проєкт на такі частини як UI (Інтерфейс користувача), Core (Операції з внутрішнім сховищем, запити до серверу та ін.) та шар через який відбувається взаємодія між цими двома частинами. Чиста архітектура (Clean Architecture) – це архітектурний підхід у розробці програмного забезпечення, який ставить основний акцент на розділенні коду на окремі шари з мінімальними залежностями між ними. Цей підхід сприяє збереженню високої рівності гнучкості та розширюваності програмного забезпечення, полегшує тестування та підтримку коду [5].

MVVM це шаблон який рекомендується застосовувати у android-застосунку для з’єднання частини UI з Core, саме він і є тим шаром у Clean Architecture через який дві інші частини взаємодіють одна з одною. MVVM є архітектурним шаблоном для розробки програмного забезпечення, особливо

популярним у мобільній розробці. У цій архітектурі Модель (Model) відповідає за управління даними, Вид (View) – за відображення графічного інтерфейсу користувача, а Модель перегляду (ViewModel) – за логіку взаємодії між Моделлю та Видом. MVVM сприяє розділенню обов'язків між компонентами програмного забезпечення, полегшує тестування, забезпечує більшу гнучкість та розширюваність коду [6].

Обрані архітектурні шаблони проєктування (Clean Architecture та MVVM) передбачають усі зазначені вимоги (розширення, здатність до змін, простота та ефективність), тож ніяких змін по мірі розвитку проєкту не буде відбуватись, будуть лише додаватись нові компоненти, наприклад такі як: нові вью-моделі, нові запити до серверу або операції з локальною базою даних, та нові інтерфейси користувача.

2.4 Об'єктно-орієнтовані метрики коду

Об'єктно-орієнтовані метрики вводяться з метою:

– поліпшити розуміння якості продукту;
– оцінити ефективність розробки;
– поліпшити якість роботи на етапі проєктування.

Одними з найбільш поширеніших наборів об'єктно-орієнтованих метрик є метрики Чидамбера-Кемерера [2] та метрики Фернанда Абреу (MOOD) [3, с. 372].

Метрики Чидамбера-Кемерера орієнтовано на класи, які є фундаментальними елементами об'єктно-орієнтованого підходу. Тому вимірювання та метрики для окремого класу, ієрархії класів та взаємодії класів є важливими для інженера-програміста, якому необхідно оцінити якість проєкту.

Метрика 1: зважені методи на клас WMC (Weighted Methods Per Class) – відносна міра складності класу:

$$WMC = \sum_{i=1}^{n} c_i.$$

Візьмемо для прикладу клас SimpleTaskActivityFragment (див. додаток Б). Необхідно підрахувати методи поточного класу та методи які прямо наслідуються від батьківского класу: n = 14. При складності кожного методу $C_i = 1$ отримаємо $WMC = 14$.

Метрика 2: висота дерева наслідування DIT (Depth of Inheritance Tree) визначається як максимальна довжина шляху від листа до кореня дерева успадкування класів. На прикладі того ж самого класу (див. рис. 2.5).

Рисунок 2.5 – Графічне відображення метрики 2

Метрика 3: кількість дітей NOC (Number of children) – кількість безпосередніх спадкоємців класу в ієрархії класів. У класу з прикладу немає дочірніх класів.

Метрика 4: зчеплення між класами об’єктів CBO (Coupling between object classes) – це кількість взаємодій, передбачених для класу, тобто кількість класів, з якими він з’єднаний. Клас з прикладу наслідується від BaseListFragment, та використовує 1 його метод – reloadContent() (рис. 2.6).

```
abstract class BaseListFragment<T : Any>: BaseFragment() {
    abstract fun getContent(): Deferred<List<T>>
    abstract fun updateContent(list: List<T>)
    override fun onStart() {
        reloadContent()
        super.onStart()
    }
    fun reloadContent() {
        if (childFragmentManager.fragments.isNotEmpty()) {
            clearChildFragments()
        }
        getContent().onComplete { updateContent(it) }
    }
}
```

Рисунок 2.6 – Класс BaseListFragment

Отже маємо що $\mathrm{CBO} = 1$.

Метрика 5: відповідь для класу RFC (Response For a Class).

Множина відповідей класу RS – це множина методів, які можуть виконуватись у відповідь на надходження повідомлень в об’єкт цього класу:

$$RFC = |RS|.$$

На прикладі методу onDoneTask() із вищевказаного класу SimpleTaskActivityFragment (див. рис. 2.7, 2.8 та 2.9).

```
override fun onDoneTask(simpleTaskFragment: SimpleTaskFragment) {
    onSimpleTaskAction(R.string.TASKDONEDstring, simpleTaskFragment, true)
}
```

Рисунок 2.7 – Метод onDoneTask класу SimpleTaskActivityFragment

```
private fun onSimpleTaskAction(
    resId: Int,
    simpleTaskFragment: SimpleTaskFragment,
    isDoned: Boolean?
) {
    countDownTimer?.performImmediately()

    simpleTaskFragment.preLoader?.root?.let {
        showSnackBar(
            it,
            SNACK_BAR_DURATION,
            resId
        ) {
            shortToast(R.string.CANCELEDstring)
            countDownTimer?.cancel()
            countDownTimer = null
            appear(it)
        }
    }

    countDownTimer = object : EnhancedTimer(SNACK_BAR_DURATION, 1000) {
        override fun onTick(millisUntilFinished: Long) {}

        override fun onFinish() {
            onTimerExhausted(simpleTaskFragment, isDoned)
        }
    }

    countDownTimer?.start()
}
```

Рисунок 2.8 – Метод onSimpleTaskAction класу SimpleTaskActivityFragment

```
private fun onTimerExhausted(simpleTaskFragment: SimpleTaskFragment, isDoned: Boolean?) {
   asyncOnDefault {

viewModel.removeSimpleTaskDataContainer(simpleTaskFragment.viewModel.getDataContainer(),
isDoned)
   }
   removeChildFragment(simpleTaskFragment)
   countDownTimer = null
}
```

Рисунок 2.9 – Метод onTimerExhausted класу SimpleTaskActivityFragment

Метод onDoneTask() викликає onSimpleTaskAction() який в свою чергу викликає onTimerExhausted() отже маємо що $\mathrm{RFC} = 3$.

Метрика 6: відсутність зв'язності в методах LCOM (Lack of Cohesion in Methods):

$$NC = \left|\left|\{I_{ij} \middle| I_i \cap I_j = 0\}\right|\right|,$$

$$C = \left|\left|\{I_{ij} \middle| I_i \cap I_j \neq 0\}\right|\right|,$$

$$LCOM = \max(0, NC - C),$$

а) NC – кількість пар методів без загальних змінних екземпляру;

б) C – кількість пар методів із загальними змінними екземпляру;

в) Ij – набір змінних екземпляру, що використовуються методом Mj.

Оскільки кожен метод класу з прикладу має пару по атрибутам класу, то з 100%вою впевненістю можна стверджувати що $LCOM = 0$.

Фактор закритості методу (Method Hiding Factor, MHF):

$$MHF = \frac{\sum_{i=1}^{TC} \mathrm{M}h_i}{\sum_{i=1}^{TC}(\mathrm{M}v_i + \mathrm{M}h_i)},$$

а) Mvi – кількість видимих методів класу;

б) Mhi – кількість прихованих методів класу;

в) TC – кількість класів.

Клас з прикладу має 12 видимих методів і 2 прихованих методів, отже

$$MHF = \frac{\sum_{i=1}^{1} 2}{\sum_{i=1}^{1}(12+2)} = \frac{2}{12} \approx 0{,}166667.$$

Фактор закритості атрибуту (Attribute Hiding Factor, AHF):

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)},$$

а) Ah(Ci) – кількість прихованих атрибутів класу Ci (інтерфейс класу);

б) Ad(Ci) – загальна кількість атрибутів, визначених у класі Ci (безврахування наслідування);

в) TC – загальна кількість класів.

Оскільки кожний атрибут класу з прикладу є прихованим, то $AHF = 100\%$.

Фактор наслідування методу (Method Inheritance Factor, MIF):

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)},$$

а) Mi(Ci) – кількість наслідуваних та не перевизначених методів класу Ci;

б) Ma(Ci) – кількість всіх методів, доступних у класі Ci.

Із трьох наслідуваних методів у класу з прикладу тільки один не перевизначено, отже

$$MIF = \frac{\sum_{i=1}^{1} 1}{\sum_{i=1}^{1} 3} = \frac{1}{3} \approx 0{,}333334.$$

Фактор наслідування властивості (Attribute Inheritance Factor, AIF):

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)},$$

а) Ai(Ci) – кількість наслідуваних та не перевизначених атрибутів класу Ci;

б) Aa(Ci) – загальна кількість атрибутів, визначених у класі Ci;

в) TC – загальна кількість класів.

Оскільки клас з прикладу не наслідує жодних атрибутів, то

$$AIF = 0.$$

Фактор поліморфізму (Polymorphism Object Factor, POF):

$$POF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} M_n(C_i) \times DC(C_i)},$$

а) Mo(Ci) – кількість наслідуваних та перевизначених методів класу Ci;

б) Mn(Ci) – кількість нових методів, доступних у класі Ci;

в) DC – кількість класів, що наслідують клас Ci.

Для класу з прикладу BaseListFragment маємо 1 наслідуваний та перевизначений метод onStart(), десять нових та доступних методів у цьому класі, та один клас що є дочірнім від даного класу, отже

$$POF = \frac{\sum_{i=1}^{1} 1}{\sum_{i=1}^{1} 10 \times 1} = \frac{1}{10} = 0{,}1.$$

Фактор зчеплення (Coupling Factor, COF):

$$COF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client(C_i, C_j)\right]}{TC^2 - TC},$$

а) $is\_client(C_c, C_s)\begin{cases}1\ if\ C_c \to C_s \cap C_c \neq C_s \\ 0\ else\end{cases}$;

б) TC – загальна кількість класів.

Якщо взяти класи SimpleTaskActivityFragment та BaseListFragment то будемо мати:

$$COF = \frac{\sum_{i=1}^{2}\left[\sum_{j=1}^{2} is\_client(C_i, C_j)\right]}{2^2 - 2} = \frac{1}{2^2 - 2} = \frac{1}{2} = 0{,}5.$$

3 РОЗРОБКА ДОДАТКА

Для розробки додатку було використано середовище розробки Android Studio [4].

Додаток було розроблено з використанням двох мов програмування – Java та Kotlin. У сучасних тенденціях розробки android-застосунків рекомендується використовувати Kotlin, проте деякі речі, наприклад анімації все ж таки краще створювати з використанням Java.

Також у додатку було використано локальну база даних, її було створено з використанням бібліотеки Realm. Це продукт компанії MongoDB, створений для полегшення роботи з невеликими локальними базами даних для мобільних додатків.

Авжеж, для отримання доходу, було застосовано рекламу, а саме банери. Для їх впровадження було використано інструменти від Google, а саме платформу AdMod.

Також для контролю показу реклами в середині додатка було використано інструменти Firebase, а саме FirebaseRemoteConfiguration.

3.1 Створення фундаменту

Як вже було сказано вище, для створення архітектури проєкту було обрано шаблон проєктування CleanArchitecture та MVVM. Це передбачає створення двох модулів в середині проєкту – app та organiserCore (див. рис. 3.1).

У модулі app знаходяться класи які відповідають за інтерфейс користувача та класи які з’єднують модулі core та app тобто View-моделі, а у модулі core класи, які представляють собою сутності необхідні для створення схеми локальної бази даних, та допоміжні класи для комутування двох модулів app та core.

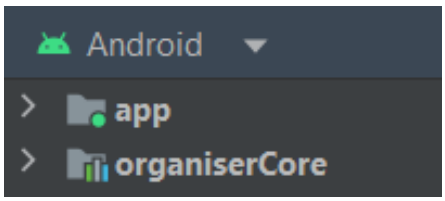


Рисунок 3.1 – Модулі проєкту

Для керування залежностями такими як бібліотеки, версії мов програмування, модулі та конфігурації застосовується стандартний інструмент у android-розробці – Gradle. У кожного модуля та у проєкту загалом є свій конфігураційний файл під назвою build.gradle.kts, у якому вказуються потрібні налаштування модулів і проєкту відповідно (див. рис. 3.2).

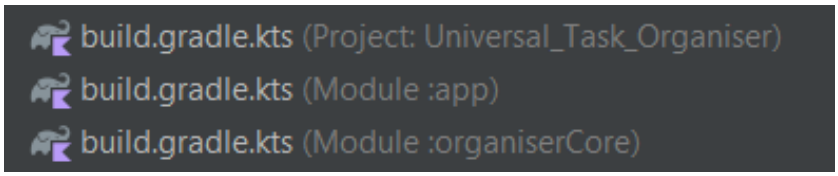


Рисунок 3.2 – Конфігураційні файли Gradle

Завдяки цим модулям інженерам програмного забезпечення не потрібно самотужки під’єднувати усі необхідні залежності при створенні додатку, бо Gradle робить це автоматично, і зберігає все у власному кеші.

Тепер необхідно створити усі необхідні класи за планом. Почнемо з модуля app.

3.1.1 Модуль app

У каталозі «base» зберігаються «базові класи» серед яких присутній і клас BaseListFragment. Ці класи слугують основами для більшості класів у проєкті, з метою спрощення їх функціональних властивостей, та уникання необхідності повторювати написання вже існуючого коду (див. рис. 3.3).

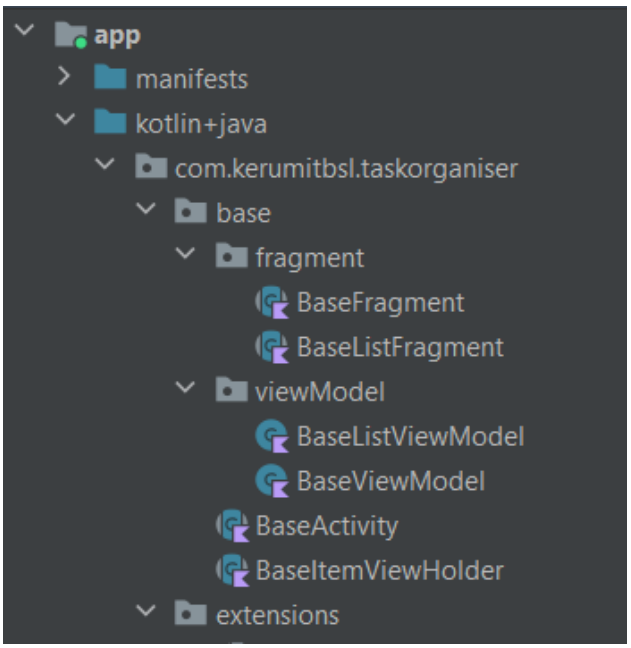


Рисунок 3.3 – Вміст папки base

У каталозі «extensions» знаходяться файли розширень, чудова властивість мови програмування Kotlin яка дозволяє винести глобальні методи або константи у окремий файл. Аналогом у мові програмування Java є статичні методи на змінні (див. рис. 3.4).

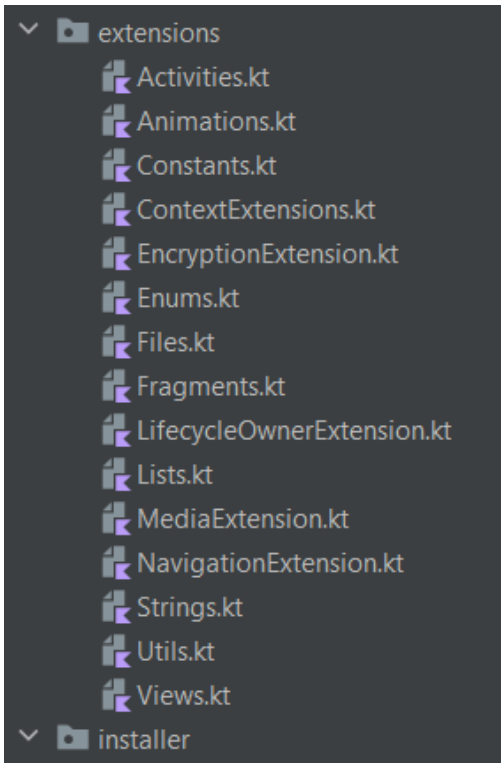


Рисунок 3.4 – Вміст папки extensions

У каталозі «installer» знаходиться клас, необхідний для оперування об’єктами, які буде створено під час роботи програми, та які повинні бути Singleton`ами. Це називається Dependency Injection (зазвичай скорочують DI), для цього було обрано бібліотеку Koin (див. рис. 3.5).

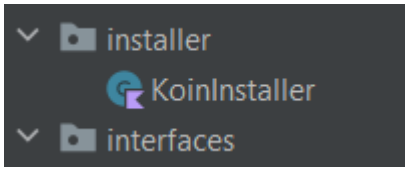


Рисунок 3.5 – Вміст папки installer

У каталозі «interfaces» містяться усі необхідні інтерфейси через які фрагменти можуть комунікувати з активностями у яких вони знаходяться (див. рис. 3.6).

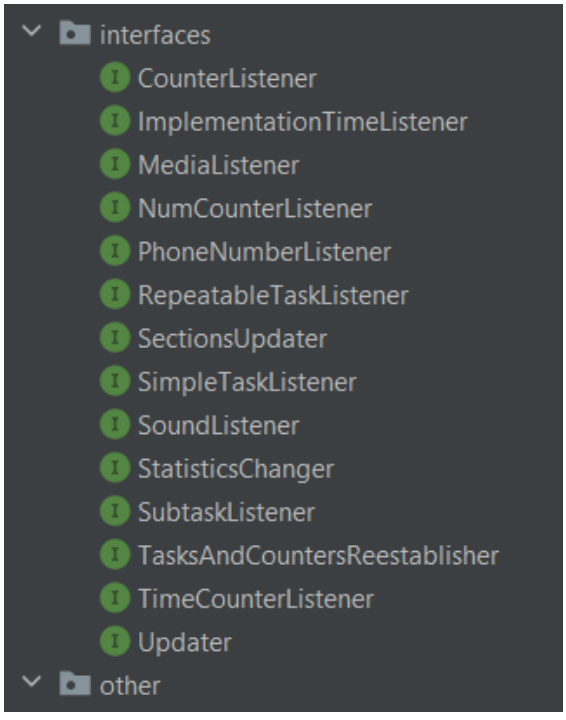


Рисунок 3.6 – Вміст папки interfaces

У каталозі «other» містяться всі допоміжні класи у який винесено певну логіку яка може використовуватись у декількох місцях проєкту, з метою зменшення обсягу коду (див. рис. 3.7).

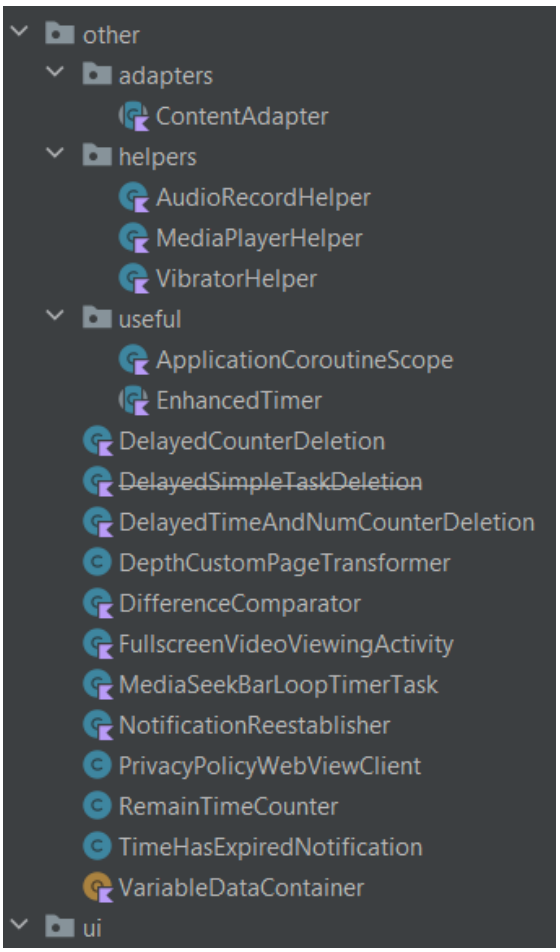


Рисунок 3.7 – Вміст папки other

У каталозі «ui» знаходяться всі класи які відповідають за користувацький інтерфейс, серед них присутній і клас SimpleTaskActivityFragment. Класс AppManger представляє собою відображення додатку, у ньому зазвичай знаходиться логіка яка повинна виконуватись при відкритті його(додатку) користувачем (див. рис. 3.8).

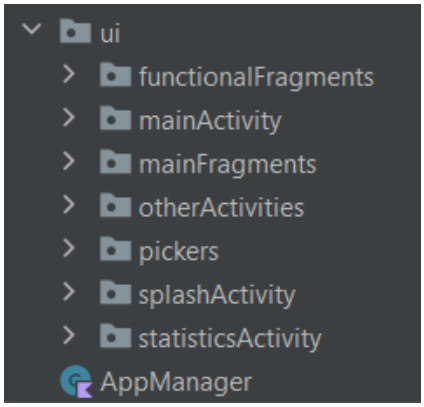


Рисунок 3.8 – Вміст папки ui

3.1.2 Модуль organiserCore

У цьому модулі знаходяться всі необхідні класи які відповідають саме за «підкапотну» логіку додатку. Це переважно сутності локальної бази даних, і класи завдяки яким відбувається передача необхідних даних при їх запиті із модуля app, тобто комунікація між двома модулями.

Класс OrganiserCoreApi представляє собою міст через який модуль app може запитувати у модуля core потрібні йому дані.

Об’єкти OrganiserCoreApiBuilder та RealmGetter є статичними, це також чудова властивість мови програмування Kotlin. Ці об’єкти використовуються у DI для створення сінглтонів.

Також, як і у модулі app тут є допоміжні класи, класи extension і класс-інсталлер для роботи з Koin Dependency Injection (див. рис. 3.9).

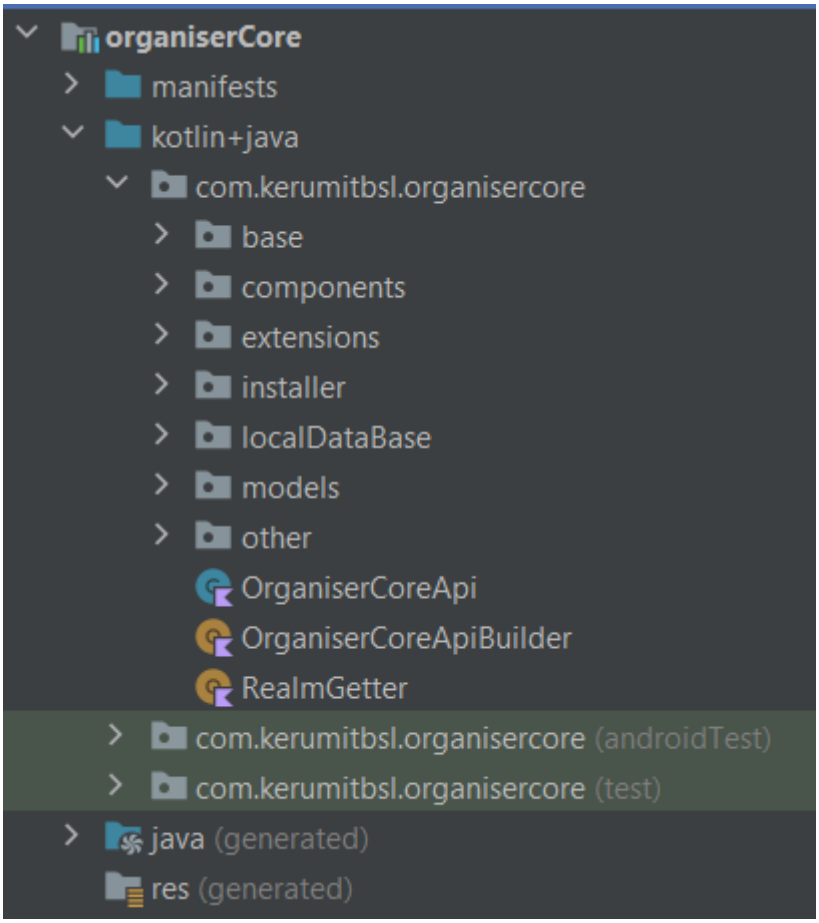


Рисунок 3.9 – Структура модуля organiserCore

3.2 Додавання залежностей

Необхідні бібліотеки, як вже вказувалось вище, було додано у конфігураційних файлах Gradle, а саме: Android SDK (і всі необхідні речі які йдуть у комплекті з ним), Realm, AdMob та FireBase (див. рис. 3.10 – 3.12).

```
buildscript {
    repositories {
        google()
        mavenCentral()
        maven("https://jitpack.io")
    }
    dependencies {
        classpath("io.realm:realm-gradle-plugin:10.15.1")
        classpath("com.android.tools.build:gradle:8.3.2")
        classpath("org.jetbrains.kotlin:kotlin-gradle-plugin:1.7.20")
        classpath("com.google.gms:google-services:4.4.1")
        classpath("androidx.navigation:navigation-safe-args-gradle-plugin:2.7.7")
        classpath("com.google.firebase:firebase-crashlytics-gradle:2.9.9")
    }
}

plugins {
    id("com.android.application") version "8.1.2" apply false
    id("org.jetbrains.kotlin.android") version "1.9.0" apply false
    id("com.android.library") version "8.1.2" apply false
    id("org.jetbrains.kotlin.kapt")  version "1.6.20" apply false
    id("com.google.gms.google-services")  version "4.4.1" apply false
    id("com.google.firebase.crashlytics")  version "2.9.9" apply false
}

allprojects {
    repositories {
        google()
        maven { uri("https://jitpack.io") }
        mavenCentral()
    }
}

tasks.register("clean", Delete::class.java) {
    delete(rootProject.buildDir)
}
```

Рисунок 3.10 – Вміст конфігураційного файлу проєктного рівня

```
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("org.jetbrains.kotlin.kapt")
    id("androidx.navigation.safeargs.kotlin")
    id("com.google.gms.google-services")
    id("com.google.firebase.crashlytics")
}

android {
    namespace = "com.kerumitbsl.taskorganiser"
    compileSdkPreview = "UpsideDownCake"

    defaultConfig {
        applicationId = "com.kerumitbsl.taskorganiser"
        minSdk = 19
        targetSdk = 33
        versionCode = 6
        versionName = "1.5.50 Mona"
        multiDexEnabled = true
        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            isMinifyEnabled = true
            isDebuggable = false
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
            ndk { debugSymbolLevel = "symbol_table" }
        }
        debug {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
            ndk { debugSymbolLevel = "symbol_table" }
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_17
        targetCompatibility = JavaVersion.VERSION_17
    }
    kotlinOptions {
        jvmTarget = "17"
    }
    buildFeatures {
        viewBinding = true
    }
}

dependencies {
```

```
    implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.7.1")

    implementation(project(":organiserCore"))

    implementation(fileTree(mapOf("dir" to "libs", "include" to listOf("*.jar", "*.aar"), "exclude" to
emptyList<String>())))

    implementation("androidx.multidex:multidex:2.0.1")

    implementation("androidx.core:core-ktx:1.12.0")
    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.11.0")
    implementation("androidx.constraintlayout:constraintlayout:2.1.4")
    implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.7.0")
    implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.7.0")
    implementation("androidx.navigation:navigation-fragment-ktx:2.7.7")
    implementation("androidx.navigation:navigation-ui-ktx:2.7.7")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")

    implementation("androidx.legacy:legacy-support-v4:1.0.0")

    val workVersion = "2.9.0"
    implementation("androidx.work:work-runtime:$workVersion")
    implementation("androidx.work:work-multiprocess:$workVersion")

    //google ads
    implementation ("com.google.android.gms:play-services-ads:21.5.0") // do not increase version

    // koin
    api("io.insert-koin:koin-android:3.4.2")
    api("io.insert-koin:koin-core:3.4.2")

    //firebase
    implementation(platform("com.google.firebase:firebase-bom:32.7.0"))
    implementation("com.google.firebase:firebase-analytics-ktx")
    implementation("com.google.firebase:firebase-config-ktx")
    implementation("com.google.firebase:firebase-crashlytics")
    implementation("com.google.firebase:firebase-analytics")
}
```

Рисунок 3.11 – Вміст конфігураційного файлу модуля app

```
plugins {
    id("com.android.library")
    id("org.jetbrains.kotlin.android")
    id("kotlin-kapt")
    id("kotlin-parcelize")
    id("com.google.firebase.crashlytics")
}

apply(plugin = "realm-android")
```

```
android {
    namespace = "com.kerumitbsl.organisercore"
    compileSdkPreview = "UpsideDownCake"

    defaultConfig {
        minSdk = 19

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
        consumerProguardFiles("consumer-rules.pro")
    }

    buildTypes {
        release {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
            ndk { debugSymbolLevel = "symbol_table" }
        }
        debug {
            isMinifyEnabled = true
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
            ndk { debugSymbolLevel = "symbol_table" }
        }
    }
    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_17
        targetCompatibility = JavaVersion.VERSION_17
    }
    kotlinOptions {
        jvmTarget = "17"
    }
}

dependencies {

    implementation("androidx.core:core-ktx:1.12.0")
    implementation("androidx.appcompat:appcompat:1.6.1")
    implementation("com.google.android.material:material:1.11.0")
    testImplementation("junit:junit:4.13.2")
    androidTestImplementation("androidx.test.ext:junit:1.1.5")
    androidTestImplementation("androidx.test.espresso:espresso-core:3.5.1")

    // coroutines
    api("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.7.1")
    api("org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.1")

    // koin
    api("io.insert-koin:koin-android:3.4.2")
    api("io.insert-koin:koin-core:3.4.2")
```

```
    // lifecycle
    api("androidx.lifecycle:lifecycle-extensions:2.2.0")
    api("androidx.lifecycle:lifecycle-viewmodel-ktx:2.7.0")
    api("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")
    api("androidx.lifecycle:lifecycle-livedata-core-ktx:2.7.0")
    api("androidx.lifecycle:lifecycle-livedata-ktx:2.7.0")

    // REST
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    api("com.squareup.retrofit2:converter-gson:2.6.0")
    implementation("com.squareup.okhttp3:okhttp:4.9.1")
    implementation("com.squareup.okhttp3:logging-interceptor:4.9.0")
    api("com.google.code.gson:gson:2.9.0")

    //SDK
    implementation("org.jsoup:jsoup:1.15.3")

    //firebase
    implementation(platform("com.google.firebase:firebase-bom:32.7.0"))
    implementation("com.google.firebase:firebase-analytics-ktx")
    implementation("com.google.firebase:firebase-config-ktx")
    implementation("com.google.firebase:firebase-crashlytics")
    implementation("com.google.firebase:firebase-analytics")
}
```

Рисунок 3.12 – Вміст конфігураційного файлу модуля core

3.3 Створення локальної бази даних

У додатку було використано локальну базу даних, оперування з якою здійснюється за допомогою бібліотеки Realm [7]. Використання цієї бібліотеки дозволяє працювати з порібними даними як із звичайними об’єктами у об’єктно оріентованому програмуванні, інкапсулюючи всю логіку пов’зану із створенням, оновленям, зчитуванням та видаленням даних.

Для створення бази даних необхідно було визначити які типи даних будуть зберігатись, та створити для них відповідні класи, наслідуючись від базової моделі RealmObject. У випадку створюваного додатку, це, наприклад, об’єкт даних задачі – SimpleTaskFragmentDataContainer. Він містить інформацію про назву та опис задачі, підзадачі, медіа та інше (див. рис. 3.13).

```
open class SimpleTaskFragmentDataContainer : RealmObject() {
    var name: String = ""
    var description: String = ""
    var typeOfTask: Byte = 0
    var dates: RealmList<ImplementationDateDataContainer> = realmListOf()
    var countOfDates = 0
    var dateOfCreation: Long = 0
    var datesOfChanges: RealmList<Long> = realmListOf()
    @PrimaryKey
    var identificator = 0
    var pathOfVoiceRecord: String? = null
    var mediaPaths: RealmList<MediaDataContainer> = realmListOf()
    var soundPaths: RealmList<SoundDataContainer> = realmListOf()
    var subtasks: RealmList<SubtaskDataContainer> = realmListOf()
    var links: RealmList<LinkDataContainer> = realmListOf()
    var files: RealmList<FileDataContainer> = realmListOf()
    var phoneNumbers: RealmList<PhoneNumberDataContainer> = realmListOf()
    var coordinates: RealmList<CoordinatesDataContainer> = realmListOf()
    var priority: Byte = 0
    var flags: Byte = 0
    var showEverydayReminders = false
    var timeOfReminders: RealmList<Long> = realmListOf()
    var cycleOfReminders: Long = 0
}
```

Рисунок 3.13 – Клас SimpleTaskFragmentDataContainer

Однією з чудових властивостей бібліотеки Realm є те, що вона дозволяє створювати списки даних, та навіть списки інших об’єктів прямо в середині класу, однією умовою є лише те що список повинен складатися з об’єктів які також є частиною схеми локальної бази даних, тобто наслідуватись від класу RealmObject. Наприклад змінна subtasks це список об’єктів типу SubtaskDataContainer (див. рис. 3.14).

```
open class SubtaskDataContainer : RealmObject() {
    @PrimaryKey
    var id: Int = 0
    var text: String = ""
    var state: Byte = 0
}
```

Рисунок 3.14 – Клас SubtaskDataContainer

Загалом у додатку було створено декілька головних класів-моделей даних, та класів які виступають у ролі типів для списків у головних класах (див. рис. 3.15).

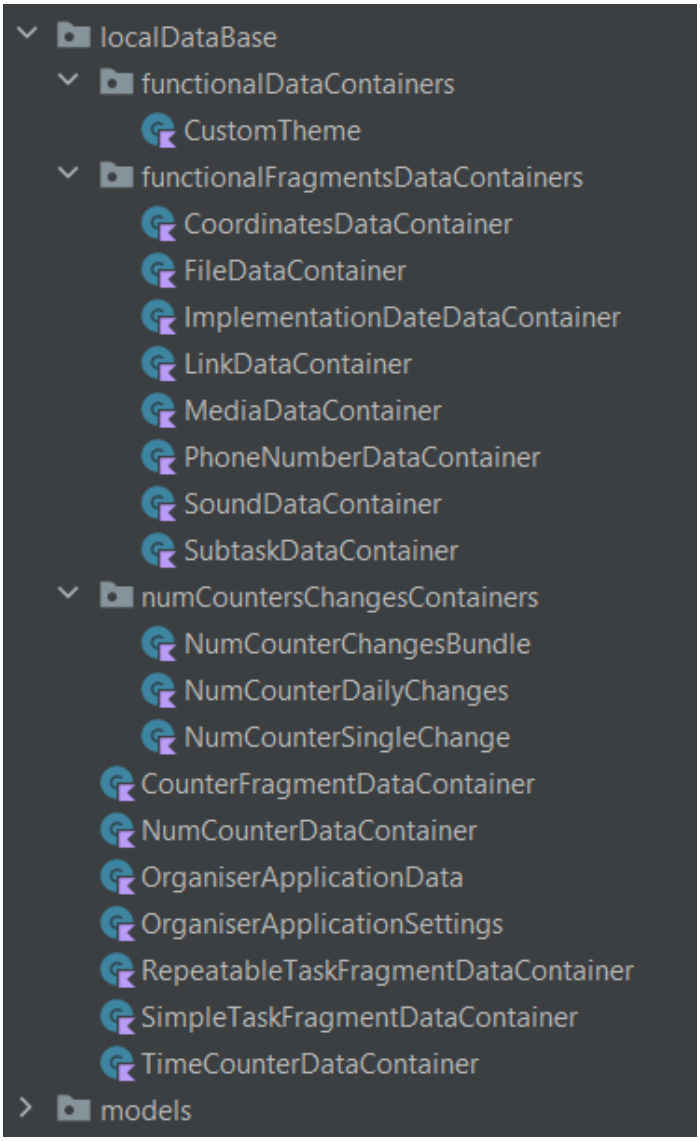


Рисунок 3.15 – Вміст папки localDataBase модуля organiserCore

3.4 Впровадження реклами

Для інтеграції реклами у додаток було застосовано сервіс AdMod від Google [8]. Імплементація цього сервісу додає усі необхідні класи для роботи з рекламою, а також необхідні елементи користувацького інтерфейсу.

Під час розробки було додано рекламний банер, та налаштовано показ реклами у ньому (див. рис. 3.16 та 3.17).

```
<com.google.android.gms.ads.AdView
    xmlns:ads="http://schemas.android.com/apk/res-auto"
    android:id="@+id/ad_view_banner_in_main_activity"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    ads:adSize="BANNER"
    ads:adUnitId="ca-app-pub-5568142015208634/7034673210"
    ads:layout_constraintBottom_toBottomOf="parent"
    ads:layout_constraintStart_toStartOf="parent"
    ads:layout_constraintEnd_toEndOf="parent"/>
```

Рисунок 3.16 – Додавання банеру до користувацького інтерфейсу

```
if (adRequest == null)
    adRequest = AdRequest.Builder().build()

checkAdsAllowed(FRC_SHOW_ADS_IN_MAIN_ACTIVITY) {
    adRequest?.let { binding?.appBarMain?.adViewBannerInMainActivity?.loadAd(it) }
}
```

Рисунок 3.17 – Створення запиту на показ реклами та його впровадження

Після створення та впровадження запиту на показ, рекламні оголошення автоматично з’являться у банері через деякий час, зазвичай впродовж трьох секунд.

3.5 Управління показом реклами

Для регулювання показу реклами у додатку було впроваджено сервіс від Firebase, а саме Firebase Remote Config [9]. Завдяки цьому можливо контролювати чи показувати взагалі оголошення, чи призупинити їх показ. Для цього на сторінці сервісу було створено параметри, які додаток перевіряє на стан, і взалежності від їх стану вирішує чи завантажувати рекламні оголошення, чи ні (див. рис. 3.18).

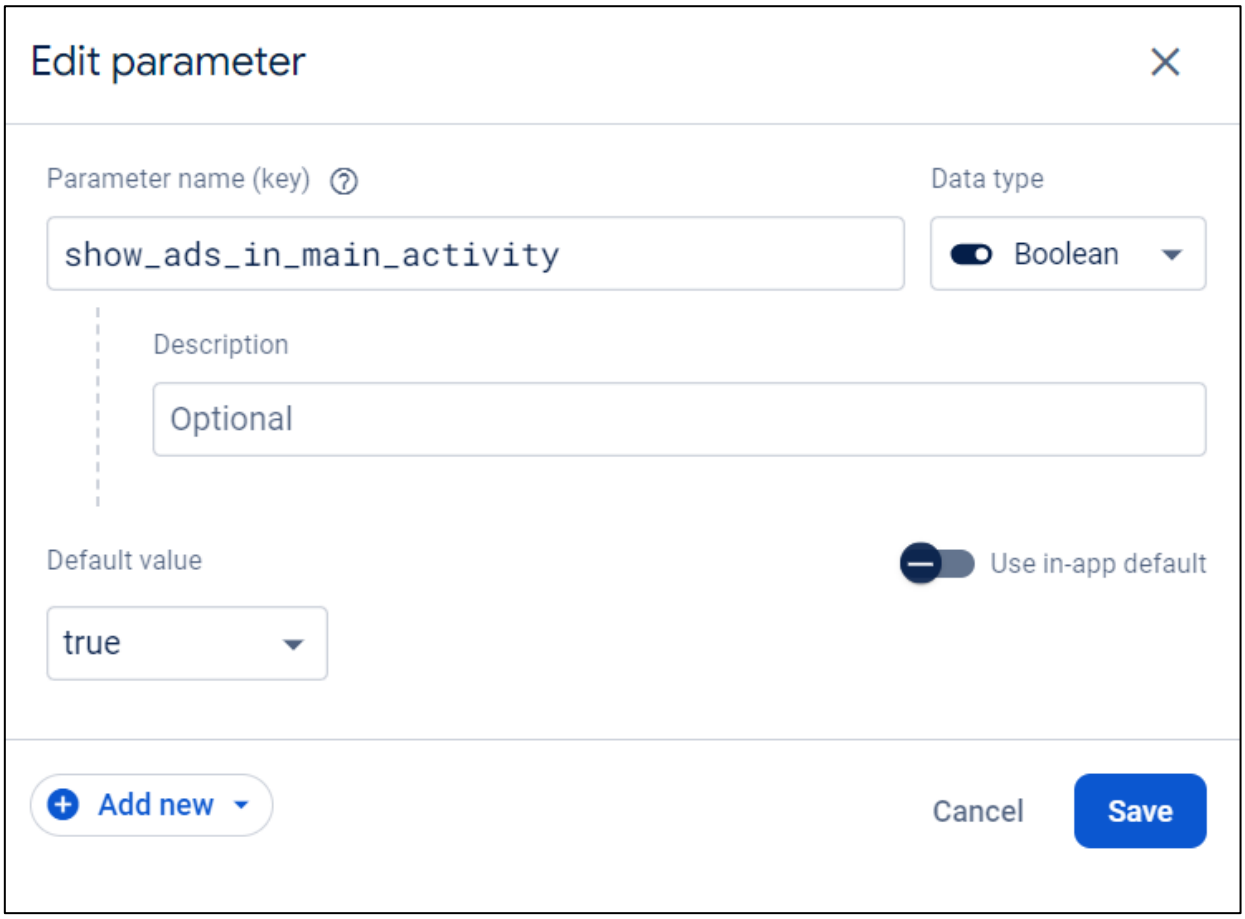


Рисунок 3.18 – Налаштування параметру для показу рекламних оголошень

Перевірка параметрів на стан виконується наступним чином (див. рис. 3.19).

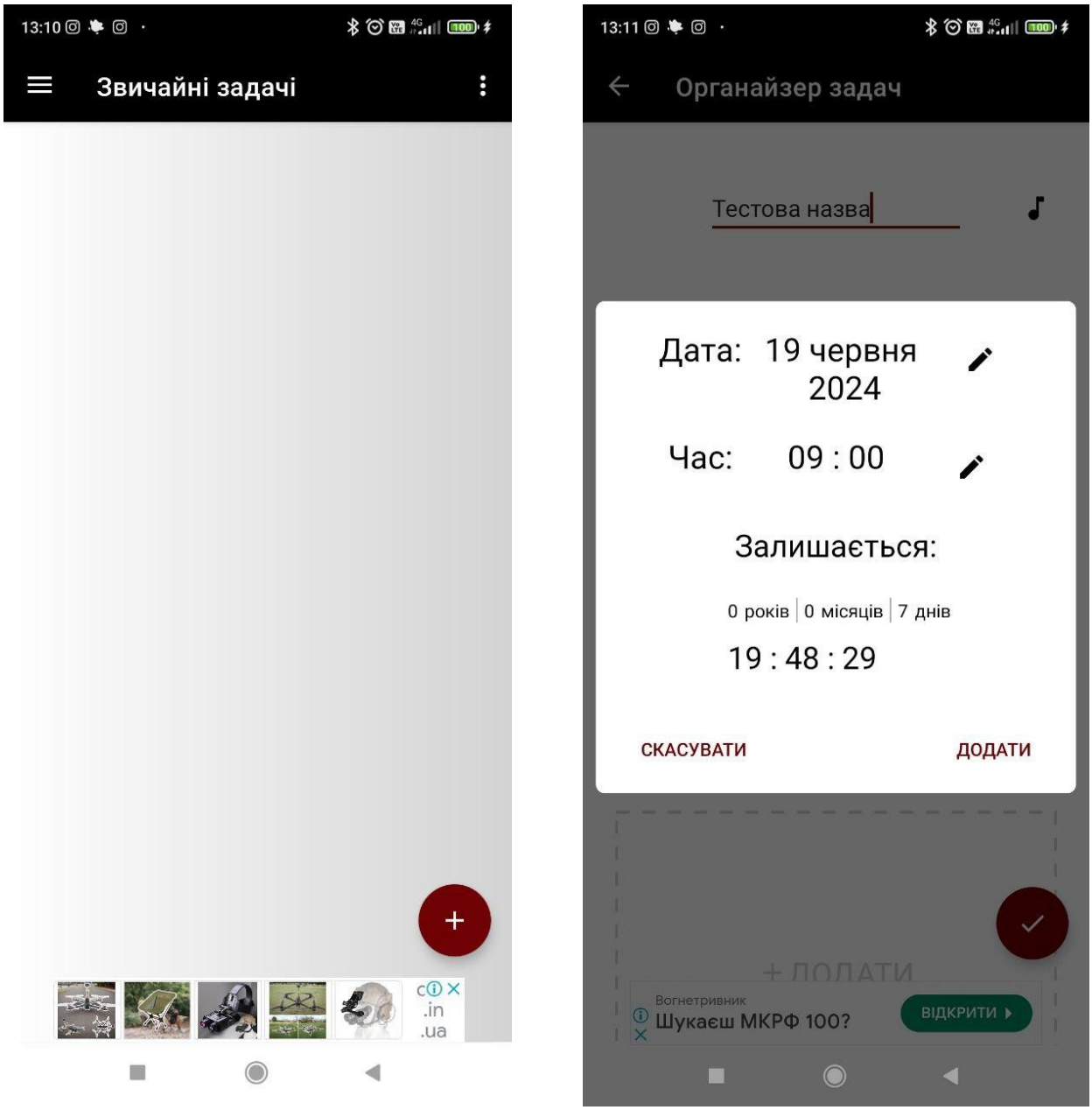
```
fun Activity.checkAdsAllowed(key: String, onAllowed: () -> Unit) =
    Firebase.remoteConfig.fetchAndActivate().addOnCompleteListener(this) {
        if (it.isSuccessful) {
            if (Firebase.remoteConfig.getBoolean(key)) {
                getMainHandler().post { onAllowed() }
            }
        }
    }
```

Рисунок 3.19 – Перевірка стану параметру

Якщо стан параметру дорівнює true, то виконується показ рекламних оголошень, у іншому ж випадку, нічого не відбувається, отже і оголошення не показуються.

3.6 Презентація додатка

Для створення задачі необхідно натиснути на червону кнопку з іконкою «плюс» у нижньому правому куті (див. рис. 3.20, а), після чого відкриється нова активність у якій можна буде задати назву та час коли треба виконати цю задачу (див. рис. 3.20, б).

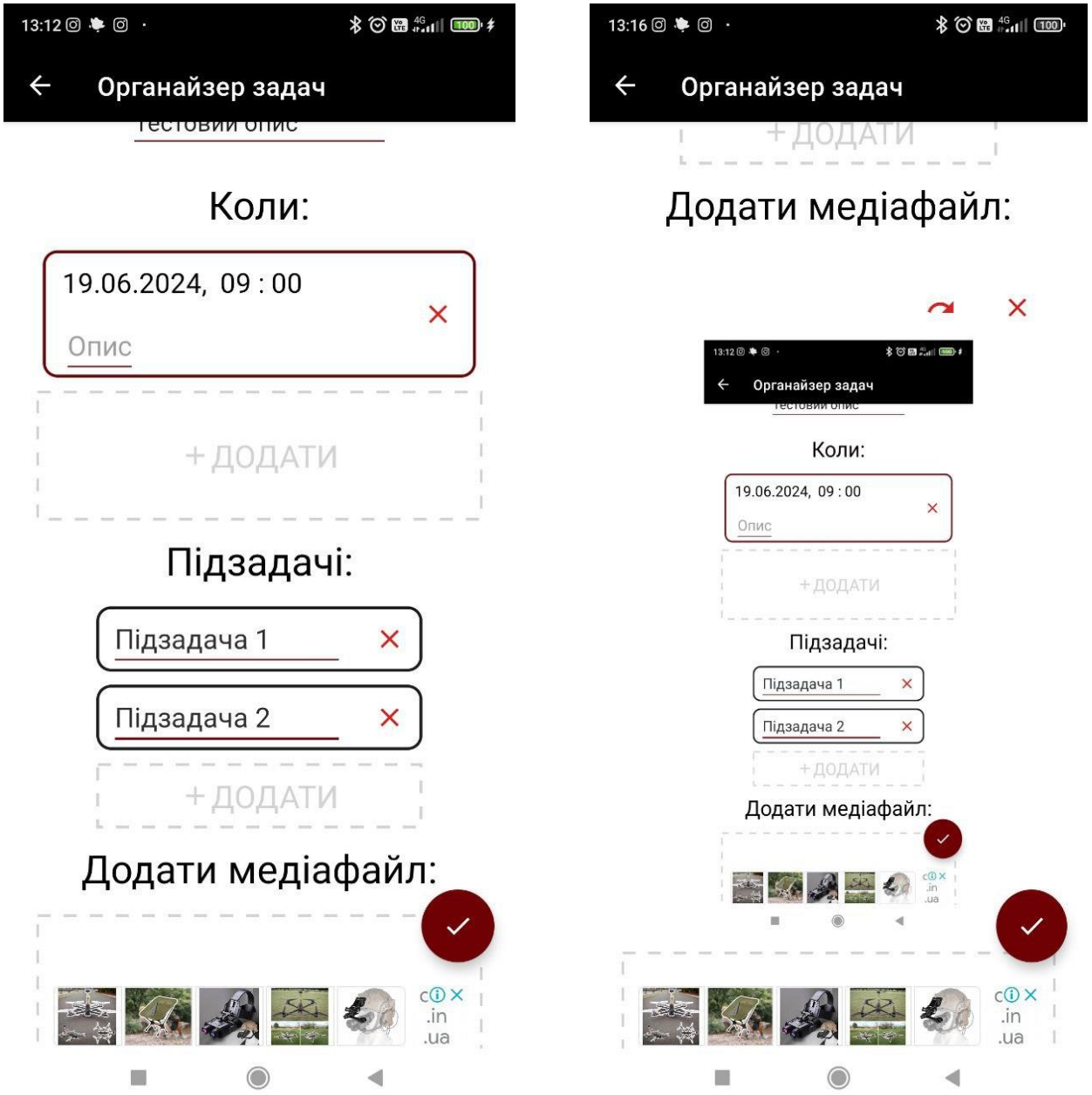


а) Початкова сторінка з кнопкою «додати»

б) Задавання назви та додавання часу виконання

Рисунок 3.20

Додамо підзадачі (див. рис. 3.21, а) та медіафайли (див. рис. 3.21, б).



а) Додавання підзадач б) Додавання медіа

Рисунок 3.21

Також можна додати аудіозапис та номер телефону (див. рис. 3.22).

Після всіх вищепроведених дій можна або натиснути на кнопку «створити» у самому низу, або натиснути на кнопку з іконкою галочки у нижньому правому куті, після чого буде відкрито список всіх задач серед яких буде і новостворена (див. рис. 3.23).

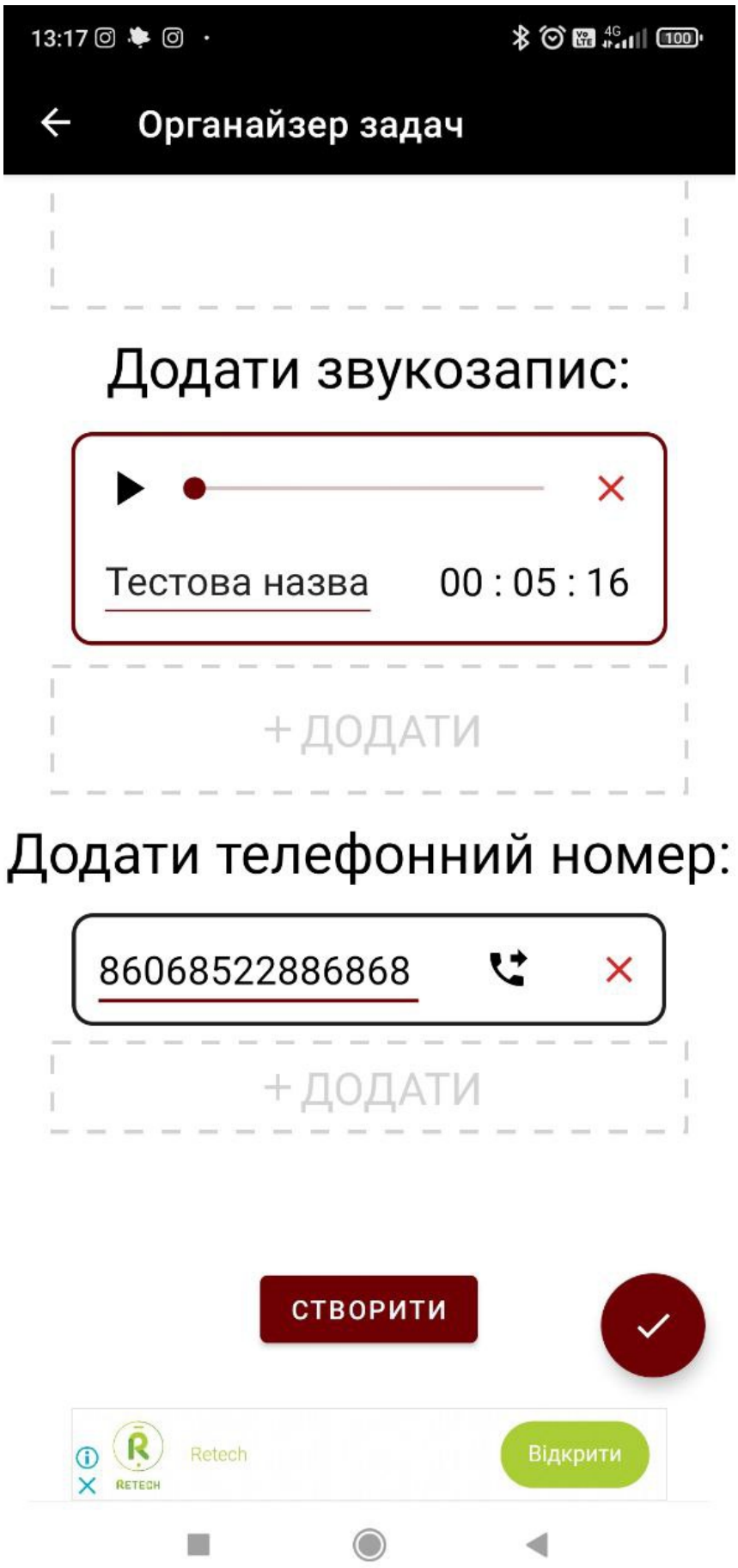
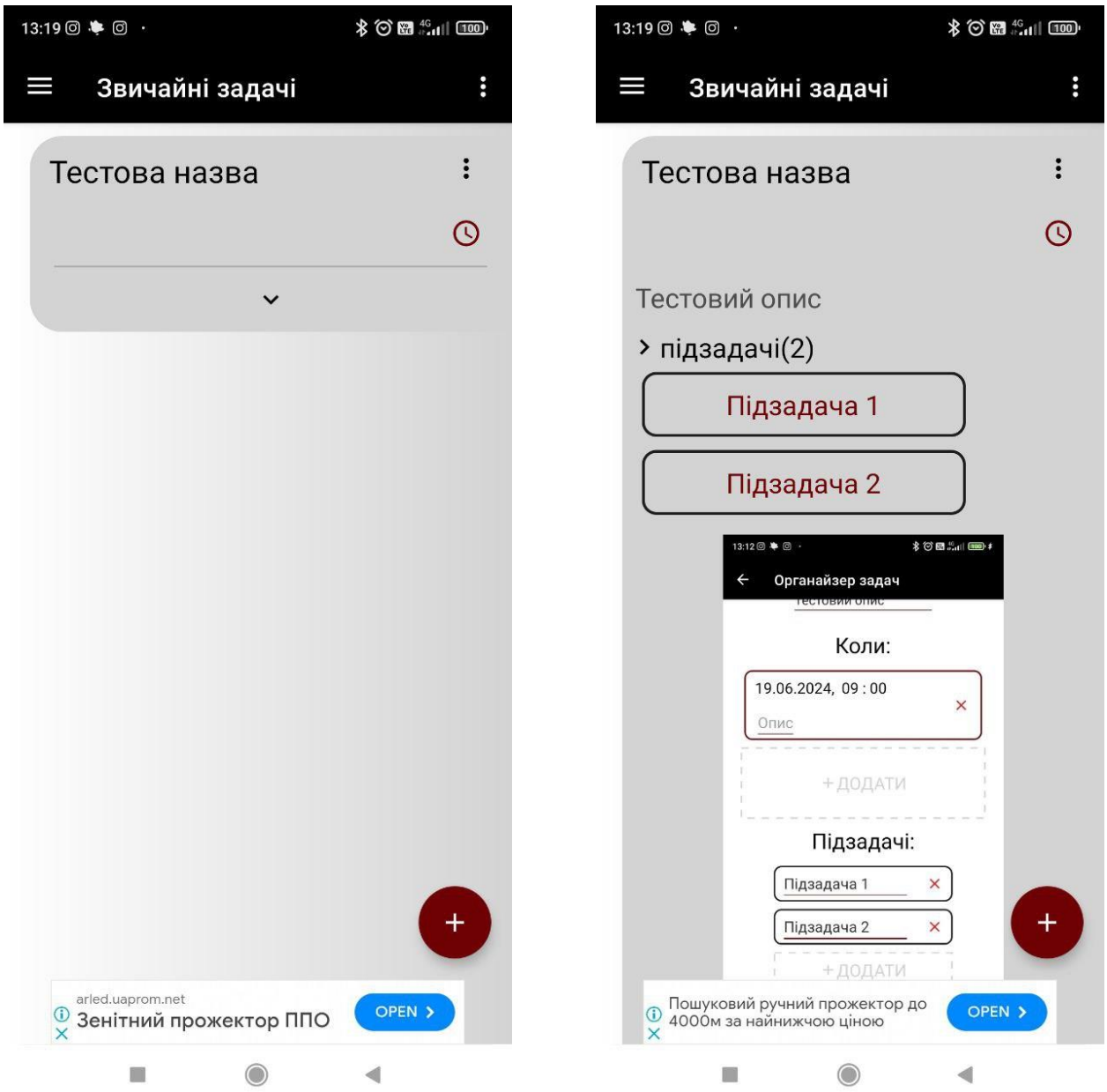


Рисунок 3.22 – Додавання аудіозапису та номеру телефона



а) Список всіх задач

б) Новостворена задача у розкритому вигляді

Рисунок 3.23

ВИСНОВКИ

У кваліфікаційній роботі виконано проєктування та реалізовано персональний органайзер задач на платформі Android. Було створено базову архітектуру, розроблено користувацький інтерфейс, створено локальну базу даних, впроваджено показ реклами та додано можливість контролювати показ реклами, тобто всі цілі які було створено на початку було досягнуто під час розробки проєкту.

Під час розробки додатка було використано сучасні підходи та технології, зокрема архітектурний патерн MVVM (Model-View-ViewModel), який дозволяє відокремити бізнес-логіку від користувацького інтерфейсу та спрощує тестування. Крім того, для зберігання даних використовувалася база даних Realm, що дозволяє зберігати дані локально на пристрої користувача. Додаток було реалізовано з урахуванням принципів чистої архітектури (Clean Architecture), що сприяє збереженню чіткого розділення між окремими компонентами додатку та полегшує масштабування та підтримку коду в майбутньому.

У результаті розробки додатку було створено зручний і функціональний інструмент для управління завданнями, який включає такі основні функції, як додавання, редагування та видалення завдань, а також встановлення дедлайну для кожної задачі. Крім того, було реалізовано можливість налаштування нагадувань для важливих подій та завдань, що допомагає користувачам не пропустити важливі дедлайни.

Можливі напрямки подальшого розвитку включають розширення функціональності додатку, додавання нових функцій, таких як можливість синхронізації з обліковими записами користувачів, реалізація групування задач за категоріями, покращення користувацького інтерфейсу та інші.

ПЕРЕЛІК ПОСИЛАНЬ

1. Застосування методу Сааті в задачах. Економіка і організація управління. URL: https://jeou.donnu.edu.ua/article/view/2967/3006 (дата звернення: 11.04.2024).
2. Використання метрик Чидамбера-Кемерера. Сайт Луцького національного технічного університету. URL: https://e-tk.lntu.edu.ua/mod/resource/view.php?id=9094&redirect=1 (дата звернення: 18.04.2024).
3. Набір метрик Фернандо Абреу. Сайт Національного авіаційного університету. URL: https://er.nau.edu.ua/bitstream/NAU/25927/1/%D0%92%D0%95%D0%A0%D0%A1%D0%A2%D0%9A%D0%90%201-388.pdf (дата звернення: 20.04.2024).
4. Download Android Studio & App Tools. Офіційний сайт Android Studio. URL: https://developer.android.com/studio (дата звернення: 20.04.2024).
5. What is clean architecture? | Definition from TechTarget. Clean Architecture. *TechTarget*. URL: https://www.techtarget.com/whatis/definition/clean-architecture#:~:text=The%20main%20rule%20of%20clean,in%20the%20more%20inward%20levels (дата звернення: 18.04.2024).
6. Model-View-ViewModel. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Model-View-ViewModel (дата звернення: 18.04.2024).
7. Realm Java SDK. Офіційний сайт Realm. URL: https://www.mongodb.com/docs/atlas/device-sdks/sdk/java/ (дата звернення: 22.04.2024).
8. Монетизація мобільних додатків – Admob – Google. Офіційний сайт Admob. URL: https://admob.google.com/intl/ua/home/ (дата звернення: 25.04.2024).

9. Firebase Remote Config – Google. Офіційний сайт Firebase. Розділ присвячений Firebase Remote Configuration. URL: https://firebase.google.com/docs/remote-config?hl=ua (дата звернення: 25.04.2024).

ДОДАТОК А

Уточнення щодо ліцензії

Додаток, який було розроблено під час написання цієї дипломної роботи, є власністю розробника, тож будь-яке використання фрагментів коду, які є частиною цього додатка або належать до нього, необхідно погоджувати з розробником.

Для зв’язку з розробником звертатись за електронною поштою kerumitbsl.dev@gmail.com.

Посилання на додаток на платформі Google Playmarket: https://play.google.com/store/apps/details?id=com.kerumitbsl.taskorganiser.

Посилання на політику конфіденційності додатка: https://sites.google.com/view/organiser-privacy-policy/%D0%B3%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0.

ДОДАТОК Б

Клас SimpleTaskActivityFragment

```
class SimpleTasksActivityFragment :
    BaseListFragment<SimpleTaskFragmentDataContainer>(),
    SimpleTaskListener {

    private var binder: FragmentSimpleTaskActivityBinding? = null
    private val viewModel: SimpleTasksActivityViewModel by viewModel()

    private var countDownTimer: EnhancedTimer? = null

    private val forResult = ActivityResultContracts.StartActivityForResult()

    private val editTaskLauncher = registerForActivityResult(forResult) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
            longToast(R.string.CHANGESAPPLIEDstring)
            //applyChanges()
        }
    }

    private val quickSoundLauncher = registerForActivityResult(forResult) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
            applyChanges()
        }
    }

    private val createNewSimpleTaskLauncher = registerForActivityResult(forResult) { result ->
        if (result.resultCode == Activity.RESULT_OK) {
            //applyChanges()
        }
    }
```

```
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binder = FragmentSimpleTaskActivityBinding.inflate(inflater, container, false)

        Initialization()
        Valuation()

        return binder?.root
    }

    override fun Valuation() {
        binder?.newSimpleTaskFab?.setOnClickListener {
            createNewSimpleTaskLauncher.launch(Intent(requireContext(),
SimpleTaskCreatingActivity::class.java))
        }
        binder?.newSimpleTaskFab?.setOnLongClickListener {
            if (!checkApplicationPermissions(listOf(Manifest.permission.RECORD_AUDIO))) {
                requestApplicationPermissions(arrayOf(Manifest.permission.RECORD_AUDIO)) {
                    if (checkApplicationPermissions(listOf(Manifest.permission.RECORD_AUDIO))) {
                        quickSoundLauncher.launch(Intent(
                            requireContext(),
                            QuickSoundPickerActivity::class.java
                        ))
                    } else {
                        shortToast(R.string.PERMISSIONNOTGRANTEDstring)
                    }
                }
            } else {
                quickSoundLauncher.launch(Intent(requireContext(),
QuickSoundPickerActivity::class.java))
            }
            return@setOnLongClickListener false
        }
```

```
}

fun applyChanges() = reloadContent()


override fun getContent() =
    viewModel.getSimpleTasks()

override fun updateContent(list: List<SimpleTaskFragmentDataContainer>) {
    viewModel.contentList.clear()
    viewModel.contentList.addAll(list)

    val listOfFragments = mutableListOf<Fragment>()

    if (!viewModel.getViewedTips().contains("SimpleTaskTip")) {
        val tipFragment = TipFragment()

        tipFragment.position = "bottom"
        tipFragment.type = "SimpleTaskTip"
        tipFragment.leftOrRight = true
        tipFragment.textSourceId = R.string.SimpleTaskTip

        tipFragment.onNoted = { removeChildFragment(it) }

        listOfFragments.add(tipFragment)
    }

    listOfFragments.addAll(list/*.data*/.convertElements {
        SimpleTaskFragment().apply {
            viewModel = SimpleTaskViewModel(it)
            onUsing = {
                binder?.scrollViewInSimpleTasksActivity?.requestDisallowInterceptTouchEvent(
                    true
                )
            }
```

```
            launchOnMain { withStarted { applyChanges() } }
        }
    })

    addChildFragments(R.id.ContainerOfSimpleTasks, listOfFragments)
}

override fun createNewNotification(simpleTaskFragment: SimpleTaskFragment) {
    asyncOnDefault {

workManager.cancelAllWorkByTag(simpleTaskFragment.viewModel.getIdentifier().toString())
        simpleTaskFragment.viewModel.getDates().forEachIndexed { index, date ->
            if (date.date >= Calendar.getInstance().timeInMillis) {
                val data =
                    Data.Builder()
                        .putInt("id", simpleTaskFragment.viewModel.getIdentifier())
                        .putString("title", simpleTaskFragment.viewModel.getName())
                        .putString(
                            "text",
                            (index + 1).toString() + " " + getString(R.string.FROMstring) + " " +
simpleTaskFragment.viewModel.getCountOfDates()
                                .toString()
                        ).build()
                val oneTimeWorkRequest = OneTimeWorkRequest.Builder(
                    TimeHasExpiredNotification::class.java
                )
                    .setInputData(data)
                    .setInitialDelay(date.date.minus(getCurrentMillis()),
TimeUnit.MILLISECONDS)
                    .addTag(simpleTaskFragment.viewModel.getIdentifier().toString())
                    .build()
                workManager.enqueue(oneTimeWorkRequest)
            }
        }
    }
```

```
}

override fun onUndoneTask(simpleTaskFragment: SimpleTaskFragment) {
    onSimpleTaskAction(R.string.TASKUNDONEDstring, simpleTaskFragment, false)
}

override fun onDoneTask(simpleTaskFragment: SimpleTaskFragment) {
    onSimpleTaskAction(R.string.TASKDONEDstring, simpleTaskFragment, true)
}

override fun onEditingTask(simpleTaskFragment: SimpleTaskFragment) {
    VariableDataContainer.simpleTaskDataContainer =
        simpleTaskFragment.viewModel.getDataContainer()
    editTaskLauncher.launch(Intent(requireContext(), SimpleTaskCreatingActivity::class.java))
}

override fun onDeleteTask(simpleTaskFragment: SimpleTaskFragment) {
    onSimpleTaskAction(R.string.TASKDELETEDstring, simpleTaskFragment, null)
}

private fun onSimpleTaskAction(
    resId: Int,
    simpleTaskFragment: SimpleTaskFragment,
    isDoned: Boolean?
) {
    countDownTimer?.performImmediately()

    simpleTaskFragment.preLoader?.root?.let {
        showSnackBar(
            it,
            SNACK_BAR_DURATION,
            resId
        ) {
            shortToast(R.string.CANCELEDstring)
            countDownTimer?.cancel()
```

```
                countDownTimer = null
                appear(it)
            }
        }

        countDownTimer = object : EnhancedTimer(SNACK_BAR_DURATION, 1000) {
            override fun onTick(millisUntilFinished: Long) {}

            override fun onFinish() {
                onTimerExhausted(simpleTaskFragment, isDoned)
            }
        }

        countDownTimer?.start()
    }

    private fun onTimerExhausted(simpleTaskFragment: SimpleTaskFragment, isDoned:
Boolean?) {
        asyncOnDefault {

viewModel.removeSimpleTaskDataContainer(simpleTaskFragment.viewModel.getDataContaine
r(), isDoned)
        }
        removeChildFragment(simpleTaskFragment)
        countDownTimer = null
    }



    override fun onPause() {
        countDownTimer?.performImmediately()
        super.onPause()
    }

    override fun onDestroyView() {
        binder = null
```

```
        super.onDestroyView()
    }
}
```