Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка та тестування мобільного застосунку супроводу ремонтних робіт з використанням мови програмування Kotlin

Виконав: студент IV курсу, групи СП-42 спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Огінський Н. А.
(підпис) (прізвище та ініціали)

Керівник Мудрик І. Я.
(підпис) (прізвище та ініціали)

Нормоконтроль Стоянов Ю. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри Петрик М. Р.
(підпис) (прізвище та ініціали)

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Комп'ютерно інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Огінському Назару Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та тестування мобільного застосунку супроводу ремонтних робіт з використанням мови програмування Kotlin

Керівник роботи д. ф., доц. Мудрик І. Я.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 06 » квітня 2026 року № _____

2. Термін подання студентом завершеної роботи 22.06.2026

3. Вихідні дані до роботи Предметна область, технічне завдання, вимоги та специфікація, програмне рішення

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області мобільного застосунку для супроводу ремонтних робіт. Розділ 2. Проєктування мобільного застосунку Renovum.
Розділ 3. Реалізація та тестування мобільного застосунку Renovum.
Розділ 4. Безпека життєдіяльності, основи охорони праці. Висновок.
Список використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Схеми та діаграми, знімки екрану з проробленою роботою, ілюстративні зображення, інформативні зображення для доповнення тексту, QR код-посилання на віддалений репозиторій GitHub

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			
Нормоконтроль			

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Розробка технічного завдання	06.05 - 12.04	
2	Робота над першим розділом «Аналіз предметної області мобільного застосунку супроводу ремонтних робіт»	13.04 - 26.04	
3	Робота над другим розділом «Проєктування мобільного застосунку Renovum»	27.04 - 05.05	
4	Робота над третім розділом «Реалізація та тестування мобільного застосунку Renovum»	06.05 - 20.05	
5	Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»	21.05 - 26.05	
6	Оформлення пояснювальної записки і графічного матеріалу	27.05 - 7.06	
7	Перевірка на академічний плагіат, перевірка керівником та консультантами	8.06 - 14.06	
8	Попередній захист кваліфікаційної роботи бакалавра	15.06 - 21.06	
9	Захист кваліфікаційної роботи бакалавра		

Студент _____ Огінський Н. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Мудрик І. Я.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка та тестування мобільного застосунку супроводу ремонтних робіт з використанням мови програмування Kotlin // Кваліфікаційна робота освітнього рівня "Бакалавр" // Огінський Назар Анатолійович // ТНТУ ім. Івана Пулюя, ФІС, кафедра програмної інженерії, група СП-42 // Тернопіль, 2026 // Ст. – 83, рис. – 20, табл. – 7 , додат. – 4, бібліогр. – 40.

Ключові слова: Android, Kotlin, Jetpack Compose, Room (SQLite), Apache POI, MVVM, автономний мобільний застосунок, ремонтні роботи, кошторис, звітність, тестування.

Метою кваліфікаційної роботи є розробка автономного локального мобільного застосунку «Renovum» для автоматизації обліку, геометричних розрахунків та фінансового супроводу ремонтних робіт з використанням мови програмування Kotlin та сучасних технологій Android-розробки.

Під час виконання роботи у першому розділі здійснено всебічний аналіз предметної області та сформовано комплекс функціональних вимог до системи. У другому розділі спроєктовано автономну архітектуру застосунку на основі шаблону MVVM (Model–View–ViewModel), деталізовано користувацькі сценарії використання, а також розроблено обчислювальні моделі даних кімнат різної конфігурації та структуру локальної реляційної бази даних Room (SQLite). У третьому розділі виконано практичну програмну реалізацію та тестування мобільного клієнта, а також інтегровано підсистему асинхронного експорту звітів. У четвертому розділі досліджено аспекти безпеки життєдіяльності та охорони праці розробника.

Реалізація виконана мовою Kotlin із використанням декларативного фреймворку Jetpack Compose, СУБД Room (SQLite) та інструментарію Apache POI. Застосунок забезпечує створення списку кімнат, розрахунок їх параметрів, збереження виконаних робіт для кожної кімнати, обрахунок кошторису усіх кімнат окремо та ремонту в цілому, формування звітності в форматі Microsoft Word Document (.docx) та перегляд історії звітності.

ABSTRACT

Development and testing of a mobile application for monitoring renovation using Kotlin // Qualification Thesis for the Bachelor's Degree // Ohinskiy Nazar Anatoliyovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, Group SP-42 // Ternopil, 2026 // Pages – 83, Figures – 20, Tables – 7, Appendices – 4, References – 40.

Keywords: Android, Kotlin, Jetpack Compose, Room (SQLite), Apache POI, mobile application, renovation, foreman, reporting, testing.

The goal of this thesis is to develop a standalone local mobile application “Renovum” for automating accounting, geometric calculations, and financial management of renovation projects using the Kotlin programming language and modern Android development technologies.

During the course of the work, the first chapter provides a comprehensive analysis of the subject area and establishes a set of functional requirements for the system. In the second chapter, an autonomous application architecture based on the MVVM (Model–View–ViewModel) pattern was designed, user scenarios were detailed, and computational models for rooms of various configurations, as well as the structure of the local relational database Room (SQLite), were developed. In the third chapter, the practical software implementation and testing of the mobile client were performed, and a subsystem for asynchronous report export was integrated. In the fourth chapter, aspects of developer safety and occupational health were examined.

The implementation is written in Kotlin using the Jetpack Compose declarative framework, the Room (SQLite) database management system, and the Apache POI toolkit. The application enables the creation of a list of rooms, the calculation of their parameters, the saving of completed work for each room, the calculation of estimates for all rooms individually and for the renovation as a whole, the generation of reports in Microsoft Word Document (.docx) format, and the viewing of report history.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

CRUD (Create, Read, Update, Delete) – базові операції роботи з даними: створення, читання, оновлення та видалення.

Gson – бібліотека для серіалізації та десеріалізації об’єктів, яка використовується у застосунку для зчитування статичних даних про послуги з JSON-файлів.

Jetpack Compose – сучасний декларативний UI-фреймворк, який використовується для побудови адаптивного та інтуїтивно зрозумілого інтерфейсу користувача.

Kotlin Coroutines/Flow – механізм для асинхронного програмування, який використовується для виконання фонових операцій без блокування основного потоку (UI).

MVVM (Model – View – ViewModel) – архітектурний шаблон, відповідно до якого застосунок поділяється на модель даних, інтерфейс користувача та компонент керування станом і логікою представлення.

Renovum – мобільний застосунок для супроводу ремонтних робіт, розроблений для платформи Android.

Room (SQLite) – бібліотека для роботи з локальними базами даних, яка використовується у застосунку для ефективного зберігання, структурування та швидкого доступу до даних про кімнати та виконані роботи.

StateFlow – компонент Kotlin Coroutines для зберігання та спостереження за станом даних у реальному часі.

UI (User Interface) – користувацький інтерфейс програмного застосунку.

UML (Unified Modeling Language) – уніфікована мова моделювання, яка використовується для опису структури та поведінки програмної системи.

UX (User Experience) – сукупність вражень користувача від взаємодії із застосунком, зокрема зручність, зрозумілість і логічність використання.

ViewModel – компонент архітектури Android, який відповідає за збереження стану екрана та обробку логіки взаємодії між інтерфейсом і даними.

ЗМІСТ

ВСТУП

Цифрова трансформація суттєво змінює способи організації бізнес-процесів у будівельній та ремонтно-оздоблювальній галузях. Сучасні проєкти з оновлення інтер'єрів потребують швидкого розрахунку вартості, зручної взаємодії між виконавцями та замовниками, прозорого планування етапів робіт та точної фіксації витрат матеріалів.

Актуальність теми зумовлена тим, що розрахунки ремонтних робіт часто виконуються вручну або за допомогою неспеціалізованого програмного забезпечення, що створює ризик помилок, ускладнює підготовку кошторисів та знижує ефективність управління проєктами. Така фрагментарність планування ускладнює контроль за виконанням робіт та ведення звітності.

Одним із шляхів вирішення цієї проблеми є створення спеціалізованого мобільного застосунку, який об'єднує основні процеси розрахунку та координації ремонтних робіт в одній системі. Мобільний формат є найбільш доцільним, оскільки смартфон завжди знаходиться під рукою в майстра чи замовника безпосередньо на об'єкті, що дозволяє виконувати операції в режимі реального часу. Такий застосунок дозволяє користувачам додавати приміщення, вносити параметри для розрахунків, формувати кошторисні звіти та автоматично генерувати документацію у форматі .docx для подальшого погодження.

У межах кваліфікаційної роботи розроблено мобільний застосунок Renovum, призначений для автоматизації формування звітності та спрощення обліку виконаних ремонтних робіт. Система орієнтована на індивідуальне використання та забезпечує повний цикл роботи з кошторисною документацією. Користувач має можливість створювати перелік приміщень, визначати обсяги робіт, вносити результати їх виконання та автоматично генерувати фінальні звіти. Крім того, застосунок реалізує функцію архівації, що дозволяє зберігати історію проведених ремонтів для подальшого швидкого доступу до попередніх розрахунків та звітів.

Особливість Renovum полягає в оцифруванні обліку ремонтних робіт, що замінює паперові носії структурованими записами. Застосунок автоматизує

формування звітів та кошторисів, мінімізуючи час на роботу з документацією та забезпечуючи впорядковане зберігання історії ремонтів.

Технічною основою проєкту є платформа Android та мова програмування Kotlin. Для побудови інтерфейсу використано Jetpack Compose та Material Design 3. Архітектура застосунку реалізована за шаблоном MVVM. Для роботи з локальними даними застосовано бібліотеку Room, а формування звітної документації реалізовано засобами Apache POI. Для навігації та взаємодії між компонентами використано Navigation Component.

Метою кваліфікаційної роботи є розробка та тестування мобільного застосунку Renovum для супроводу ремонтних робіт з використанням мови програмування Kotlin.

Об’єктом дослідження є процеси автоматизації обліку та планування ремонтно-будівельної діяльності.

Предметом дослідження є програмні засоби та технології розробки Android-застосунку для управління даними про об’єкти ремонту, фіксації виконаних робіт та формування звітної документації.

Для досягнення мети виконано такі завдання: проаналізовано предметну область; визначено основні вимоги до системи; обґрунтовано вибір технологій; спроєктовано архітектуру та структуру даних; реалізовано основні функціональні модулі; виконано тестування ключових сценаріїв; розглянуто питання впровадження, підтримки та безпеки життєдіяльності.

Практичне значення роботи полягає у створенні мобільного застосунку для автоматизації обліку ремонтних робіт. Розроблена система структурує дані про об’єкти, спрощує фіксацію виконаних процесів та мінімізує витрати часу на формування звітної документації.

Кваліфікаційна робота містить вступ, чотири розділи, висновки, список використаних джерел і додатки. У першому розділі розглянуто предметну область та вимоги до системи. У другому розділі описано архітектуру та моделі даних застосунку. У третьому розділі наведено реалізацію та тестування застосунку. У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОБІЛЬНОГО ЗАСТОСУНКУ СУПРОВОДУ РЕМОНТНИХ РОБІТ

Перший розділ присвячено аналізу предметної області, визначенню проблем супроводу ремонтних робіт, огляду існуючих цифрових інструментів, обґрунтуванню необхідності застосунку Renovum та формуванню вимог до програмної системи.

1.1 Аналіз предметної області

Ремонтно-будівельна діяльність передбачає виконання комплексу складних процесів, що потребують чіткого планування, розрахунків та контролю за використанням ресурсів. У цифровому середовищі така діяльність потребує не тільки оперативного внесення змін, а й структурованого накопичення інформації про приміщення, обсяги робіт, поточні статуси об’єктів, архівну історію розрахунків та фінансову звітність.

Для програмного продукту Renovum предметна область розглядається як сукупність взаємопов’язаних дій користувача (майстра або власника приміщення) з планування та документування ремонтів. Користувач створює об’єкт, додає перелік приміщень, визначає обсяги робіт, вносить дані про їх виконання, контролює прогрес та формує підсумкові звіти у форматі .docx. Система забезпечує централізоване зберігання цих даних, що дозволяє користувачу в будь-який момент повернутися до історії ремонтів, проаналізувати виконані завдання та отримати структуровану документацію для подальшого погодження чи архівування.

Особливістю предметної області є формування довіри до результатів робіт через верифіковану історію дій та систематизацію даних про об’єкти. Для майстра важливою є деталізація кожного етапу ремонту, наявність архіву виконаних кошторисів та відповідність фактичних витрат плановим показникам. Для замовника ключовими є прозорість розрахунків, контроль прогресу робіт та доступ до документального підтвердження всіх витрат. Тому застосунок повинен не

просто зберігати окремі дані, а формувати аналітичний контекст довкола кожного ремонтного проєкту.

Проблемою традиційного ведення ремонтних робіт є значні витрати часу на розрахунок параметрів об’єктів та неефективність їх зберігання. Багато майстрів обчислюють розміри кімнат на папері, у форматі Excel або ментально. Кожен із цих способів має суттєві недоліки: паперові носії вразливі до пошкоджень в умовах будівельного майданчика, Excel незручний для оперативного редагування зі смартфона, а розрахунки «в голові» неточні та фрагментарні. Подібна відсутність стандартизованих інструментів обліку призводить до помилок у кошторисах та затримки виконання проєктів [17].

Ще однією проблемою, яка тісно пов’язана з попередньою, є людський фактор. Процес розрахунків та фіксування прогресу ремонтних робіт потребує високої точності та надійності, що не може бути повною мірою забезпечене при ручному веденні записів. Висока ймовірність арифметичних помилок під час підрахунку площ або витрат матеріалів, випадкова втрата даних чи неоднозначне тлумачення власних чернеток створюють ризики невідповідності кошторису фактичним результатам. Відсутність автоматизації в цьому ланцюзі призводить до суб’єктивності оцінок, що ускладнює прозору комунікацію із замовником та знижує загальну ефективність управління проєктом [19].

Мобільний формат є критично важливим для цієї предметної області, оскільки параметри об’єкта та фіксація результатів робіт повинні бути доступними безпосередньо під час виконання завдань. Користувач має отримувати змогу переглянути дані приміщення та зафіксувати прогрес у будь-який момент, навіть за відсутності стабільного інтернет-з’єднання. Оскільки смартфон є інструментом, що завжди знаходиться під рукою, Renovum було реалізовано як Android-застосунок, а не як настільну чи веб-орієнтовану програму [4].

Отже, предметна область Renovum поєднує точність розрахунків, легкість використання, автономність та ефективність. Ці характеристики визначають вимоги до архітектури, моделей даних, інтерфейсу та бізнес-логіки застосунку.

1.2 Огляд існуючих цифрових рішень

Для супроводу ремонтних робіт існує низка цифрових інструментів, проте більшість із них орієнтовані на великі будівельні компанії. Поодинокі майстри та невеликі команди найчастіше використовують традиційні методи: паперові носії, Excel-таблиці або рішення, що не повністю відповідають специфічним умовам праці на об’єкті. Серед наявних продуктів можна виділити SimplyWise та Rabotniki.ua [17].

SimplyWise це американський мобільний застосунок, який дозволяє сформувати приблизний кошторис ремонту приміщення та його фінальний вигляд лиш сфотографувавши його. Переваги цієї програми полягають у використанні штучного інтелекту для аналізу зображення кімнати та формування кошторису для її ремонту у той самий момент, що дозволяє користувачу швидко зорієнтуватися в ціні ремонту та його якості. Недоліками є використання штучного інтелекту, що при формуванні кошторису може допустити декілька помилок, які можуть зробити цей його досить невідповідним реальній ситуації, а також у застосунку відсутні функції розрахунку параметрів кімнат та фіксації прогресу ремонту зі збереженням точних даних, відсутність української локалізації та обмежений функціонал у безкоштовній версії.

Rabotniki.ua – це всеукраїнська онлайн платформа для майстрів, яка дозволяє їм пропонувати свої послуги замовникам, а також підтримує великий список можливих ремонтно-будівельних робіт з щоденним оновленням актуальних цін на кожну з них в усіх областях. Також цей вебсайт має вбудований калькулятор кошторисів, який дозволяє за введеними параметрами кімнати (як от площа стін та підлоги, довжина відкосу тощо) переглянути ціни на будь-який ремонтний процес, скласти їх список та сформувати орієнтовний кошторис. Недоліками є те, що цей калькулятор орієнтований на замовників, які бажають оцінити приблизну вартість ремонту та знайти майстрів, що його проведуть, також немає функціоналу проведення повних розрахунків даних об’єкту ремонту чи введення параметрів кімнати для автоматичної підстановки даних в калькулятор.

Таблиця 1.1 – Порівняння цифрових інструментів для супроводу ремонтних робіт

Інструмент	**Переваги**	**Недоліки**
SimplyWise	Фотосканер, візуалізація (до/після).	Збільшена похибка в розрахунках через використання ШІ, відсутність локалізації, обмежений функціонал без платної підписки.
Rabotniki.ua	Точність, актуальність цін	Орієнтація розрахунків кошторису на замовника, відсутність калькулятора параметрів кімнат.
Renovum	Автономність, експорт звітів, пооб'єктний облік	Новий продукт, потребує часу на розширення функціоналу

Проведений аналіз показує, що наявні на ринку рішення демонструють фрагментарний підхід до автоматизації ремонтних робіт: вони або закривають лише вузькі потреби (як-от візуалізація або пошук виконавців), або є технічно незручними для «польових» умов роботи майстра безпосередньо на об'єкті. Більшість існуючих систем вимагають постійного доступу до інтернету, мають складний інтерфейс, неадаптований до швидкої фіксації даних, або позбавлені функцій автоматизованої генерації звітної документації, що змушує майстрів дублювати роботу в паперових нотатках.

Саме тому розробка Renovum є доцільною, адже проєкт пропонує інтегрований підхід до управління ремонтним процесом. Застосунок поєднує інструменти для точного математичного розрахунку параметрів приміщень, систематизації виконаних етапів у зручні покімнатні списки та автоматизованого формування звітності у форматі .docx, що суттєво мінімізує витрати часу на рутинну роботу з документацією. Створення такого інструменту дозволяє не лише структурувати дані про об'єкт, а й забезпечити їх збереження та легкий доступ до історії попередніх розрахунків навіть у режимі офлайн. Таким чином, Renovum виступає не просто як черговий програмний продукт, а як цілісна інформаційна система, що забезпечує прозорість взаємодії, підвищує професійну ефективність майстра та мінімізує ризики, пов'язані з людським фактором при підготовці кошторисів [18].

1.3 Обґрунтування необхідності розробки Renovum

Необхідність створення програмного продукту Renovum зумовлена стрімкою цифровізацією сфери ремонтних послуг та потребою майстрів у спеціалізованому інструментарії, адаптованому під специфічні умови праці. На відміну від універсальних офісних програм чи паперових нотатників, Renovum орієнтований саме на логіку будівельного процесу: від замірів приміщень до формування фінальної звітності.

Основна ідея Renovum полягає у забезпеченні повного циклу обліку. Користувач створює об’єкт, додає приміщення, вносить параметри робіт, контролює їх виконання та автоматично генерує документацію для погодження. Це дозволяє майстру мати «цифрового помічника», який завжди під рукою [4, 18].

Важливою перевагою такого підходу є чітка структурованість даних. У застосунку реалізовано ієрархічну модель: Об’єкт → Приміщення → Список робіт. Така архітектура спрощує навігацію та робить процес обліку зрозумілим навіть під час інтенсивної роботи на об’єкті. Система не перевантажена сторонніми функціями, що дозволяє користувачу зосередитися на головному — точності кошторису та дотриманні графіку робіт [18].

Окремою функцією є автоматизація формування звітності. Замість ручного перенесення даних у текстові редактори, застосунок генерує фінальні звіти у форматі .docx на основі внесених параметрів. Це не лише економить час, а й мінімізує ймовірність помилок, спричинених людським фактором при підрахунках або оформленні документів. Для дипломного проєкту це демонструє роботу з бізнес-правилами, обробку даних у режимі реального часу та інтеграцію із засобами формування звітності [13].

Renovum також враховує потребу у гнучкості розрахунків. Ремонтні роботи часто мають різну специфіку — від чорнових процесів до фінішного оздоблення. Тому в застосунку передбачено можливість гнучкого введення параметрів для кожного типу робіт. Збереження історії попередніх ремонтів дозволяє майстру

формувати власну базу знань, швидко звертатися до архівних розрахунків та аналізувати динаміку своєї професійної діяльності.

Система дозволяє зберігати дані локально, що є критично важливим для умов, де інтернет-з'єднання може бути нестабільним. Застосунок забезпечує надійність зберігання інформації та швидкий доступ до неї в будь-який час, а також фіксацію прогресу у вигляді звітів .docx [8, 10].

Отож, Renovum є доцільним програмним рішенням для теми кваліфікаційної роботи, оскільки об'єднує актуальну галузеву проблему, практичні сценарії використання, сучасний Android-стек (Kotlin, Jetpack Compose), архітектуру MVVM, роботу з локальними базами даних та повноцінну систему автоматизації бізнес-правил [2, 3, 5].

1.4 Формування функціональних вимог

Функціональні вимоги визначають ключові можливості системи, необхідні для ефективного вирішення завдань користувача [18]. Для мобільного застосунку Renovum функціональні вимоги сформовано на основі потреб майстра в оперативному управлінні ремонтно-будівельними процесами:

1) Система повинна забезпечувати створення проєкту ремонту залежно від введеної користувачем адреси.
2) Система повинна забезпечувати можливість створення списку кімнат та введення їх базових даних.
3) Система повинна підтримувати нестандартні форми приміщень (Г-подібна, Т-подібна тощо)
4) Система повинна проводити точні розрахунки усіх необхідних для ремонту параметрів.
5) Система повинна надавати зручний для перегляду список можливих до виконання на об'єкті робіт та їх діапазону цін.
6) Система повинна надавати користувачу можливість зберегти будь-яку роботу зі списку як виконану окремо для кожної кімнати.

7) Система повинна дозволити користувачу вводити власну ціну за одиницю роботи під час її збереження.
8) Система повинна надавати користувачу можливість змінювати список виконаних робіт та виставляти знижку на загальну вартість кошторису.
9) Система повинна дозволяти користувачу перетворювати список виконаних робіт у покімнатний або загальний кошторис у форматі Microsoft Word Document.
10) Система повинна дозволяти користувачу переглядати та змінювати історію сформованих ним кошторисних звітів.

Для наочного представлення зафіксованих функціональних вимог та визначення меж взаємодії користувача з підсистемами застосунку було розроблено діаграму прецедентів (Use Case Diagram) відповідно до стандарту UML (рис. 1.2). Вона відображає склад функцій мобільного застосунку та демонструє, що єдиним ініціатором усіх процесів у системі є майстер (користувач.

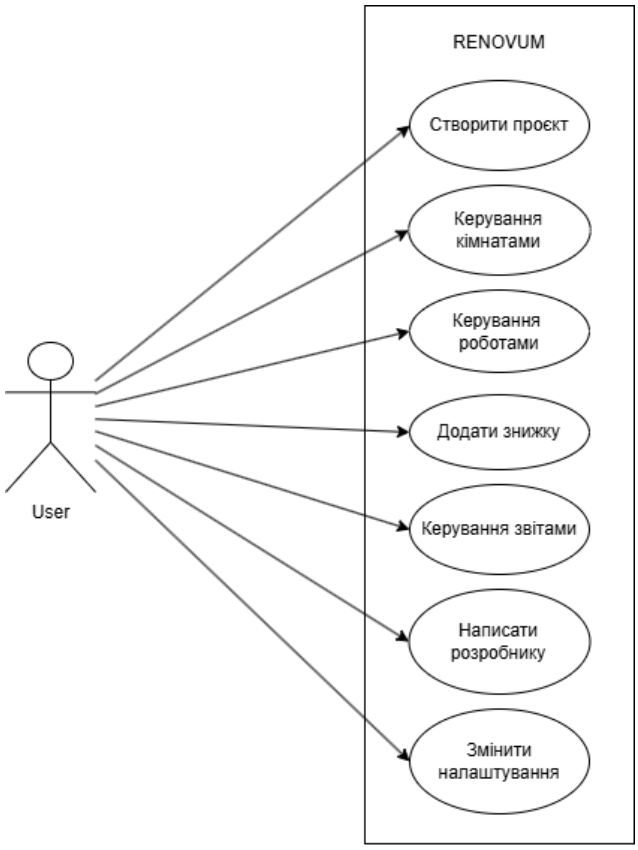


Рисунок 1.1 – Діаграма прецедентів користувача

Ключовою особливістю системи є послідовний ланцюжок формування даних, що забезпечує точність обліку на кожному етапі ремонту. Робочий процес майстра побудований на логіці ієрархічного структурування: користувач створює об’єкт, до якого додає перелік приміщень, після чого для кожної окремої кімнати формує перелік виконаних або запланованих робіт. Така архітектура дозволяє деталізувати кошторис та уникнути фрагментарності даних, що було неможливим при використанні традиційних методів [18].

Логіка обробки даних гнучко адаптована під різні типи завдань та умови об'єкта. Під час фіксації виконаної роботи майстер має змогу не лише обирати її зі стандартного списку, а й коригувати ціну за одиницю, враховуючи специфіку конкретного приміщення чи умови виконання. Система підтримує роботу як із простими, так і з нестандартними формами приміщень, виконуючи при цьому точні математичні розрахунки, що мінімізує ймовірність помилок у підсумковому кошторисі. Для додаткового управління вартістю передбачено можливість виставлення загальної знижки на проєкт, що забезпечує фінансову гнучкість при роботі із замовником.

Важливим компонентом є автоматизований механізм формування звітності. Застосунок трансформує структурований перелік робіт у повноцінний покімнатний або загальний кошторис у форматі Microsoft Word, що значно скорочує час на підготовку документації [13]. Крім того, система забезпечує постійний доступ до історії звітів, дозволяючи майстру оперативно змінювати дані або звертатися до архівних проєктів для аналізу власної продуктивності.

З огляду на специфіку «польових» умов, значну увагу приділено ергономіці та надійності системи. Інтерфейс користувача адаптований для зручного керування однією рукою, а функціонал зворотного зв’язку дозволяє майстру оперативно повідомляти розробника про будь-які технічні труднощі. Такий підхід робить Renovum не просто інструментом для розрахунків, а надійним помічником, який структурує професійну діяльність майстра на всіх етапах — від перших замірів до передачі готового кошторису замовнику [4].

1.5 Формування нефункціональних вимог

Нефункціональні вимоги визначають якісні характеристики програмної системи: швидкість відгуку, надійність, ергономічність, підтримуваність та масштабованість [18]. Для мобільного застосунку Renovum ці вимоги мають визначальне значення, оскільки застосунок орієнтований на роботу в специфічних «польових» умовах будівельного майданчика, де стабільність інтерфейсу та зручність керування є критично важливими.

Список нефункціональних вимог:

1) Застосунок повинен працювати на широкому колі Android-пристроїв [2].
2) Застосунок повинен бути реалізований як native Android-рішення.
3) Інтерфейс користувача повинен бути зручним, сучасним і зрозумілим.
4) Архітектура застосунку повинна бути розділена на незалежні логічні шари.
5) Застосунок повинен коректно працювати з асинхронними операціями та оновленням даних.
6) Дані в системі повинні залишатися узгодженими під час складних операцій.
7) Система повинна коректно працювати при відсутності постійного підключення до мережі інтернет.
8) Система повинна забезпечувати зрозумілу обробку помилок, порожніх станів і процесів завантаження.
9) Застосунок повинен бути зручним для подальшої підтримки та розширення.
10) Система повинна забезпечувати користувача можливістю адаптувати UI для зручного використання лівою або правою рукою.
11) Система повинна забезпечувати користувачу можливість повідомлення розробнику про можливі проблеми в роботі застосунку.

Для забезпечення високої підтримуваності та масштабованості архітектури програмного продукту Renovum застосовується принцип чіткого розділення

відповідальності між логічними шарами системи. Інтерфейси користувача (UI-екрани) виступають виключно як шари відображення, що не повинні безпосередньо взаємодіяти із джерелами даних чи виконувати бізнес-логіку. Відповідно до архітектурного патерну MVVM, представлення передають користувацькі події до ViewModel, які, у свою чергу, делегують операції з обробки та збереження даних спеціалізованим репозиторіям. Такий підхід суттєво спрощує проведення модульного тестування, мінімізує дублювання коду та забезпечує високий рівень його читабельності для подальшої підтримки. Надійність роботи з даними та їх цілісність у локальній базі Room гарантується використанням транзакцій, що запобігає виникненню помилок у разі переривання процесів запису або оновлення.

Крім архітектурних рішень, пріоритетним завданням є забезпечення максимальної зручності користувача, що досягається шляхом проектування інтуїтивно зрозумілих інтерфейсів. Кожен сценарій роботи майстра — від створення проєкту до формування фінального кошторису — спроектований таким чином, щоб основні дії виконувалися за мінімальну кількість кроків без перевантаження навігаційної структури. Система забезпечує зрозумілу візуалізацію станів: при виконанні складних операцій відображаються прогрес-індикатори, а при виникненні будь-яких проблем користувач отримує вичерпні повідомлення про помилки. Додатково, для швидкого доступу до необхідної інформації в межах великих обсягів даних, застосунок підтримує зручні механізми пошуку та фільтрації, що дозволяє майстру ефективно працювати в умовах обмеженого часу на будівельному об’єкті.

1.6 Вибір технологій розробки

Для розробки Renovum було обрано нативний стек Android-технологій, який гарантує не лише високу продуктивність застосунку, але й стабільну роботу в умовах повної відсутності інтернет-з’єднання, а також забезпечує широкі можливості для подальшого масштабування системи [4]. Обраний підхід дозволяє максимально ефективно використовувати апаратні ресурси мобільного пристрою,

що є критично важливим для забезпечення плавності роботи інтерфейсу під час виконання складних математичних розрахунків.

Основною мовою розробки є Kotlin [1, 2], яка забезпечує високий рівень безпеки завдяки механізму null-safety, а також надає розробнику потужні інструменти для асинхронного програмування через Coroutines та Flow [2]. Це дозволяє ефективно обробляти складні обчислювальні процеси та оновлювати стан інтерфейсу без затримок, що підвищує загальний комфорт використання.

Для побудови графічного UI обрано Jetpack Compose [3] — сучасний декларативний фреймворк, який дозволяє інтерфейсу автоматично реагувати на будь-які зміни в стані ViewModel, що є ключовим аспектом для коректного та миттєвого відображення оновлених кошторисів чи покімнатних списків робіт у режимі реального часу.

Оскільки однією з критичних вимог є забезпечення стабільної роботи без постійного підключення до глобальної мережі, основним сховищем обрано бібліотеку Room [8]. Вона забезпечує надійне та безпечне збереження об’єктів, приміщень та структурованих списків робіт безпосередньо у локальній базі даних SQLite [9], підтримуючи цілісність інформації через транзакції [10] навіть під час виконання складних ітеративних операцій [12].

Проєкт базується на класичному архітектурному патерні MVVM (Model-View-ViewModel) [18] у щільному поєднанні з Repository Pattern [21]. ViewModel повністю керують станом екранів та обробкою користувацьких подій, тоді як репозиторії виступають надійною абстракцією над усіма джерелами даних, що робить систему модульною, спрощує тестування логіки розрахунків та дозволяє без зайвих зусиль додавати нові функціональні модулі у майбутньому [19].

Для реалізації функції експорту сформованих кошторисів у стандартному форматі Microsoft Word обрано спеціалізовану бібліотеку Apache POI [13]. Вона дозволяє динамічно генерувати документи, заповнюючи заздалегідь підготовлені шаблони розрахованими даними про об’єкт та виконані роботи безпосередньо на самому пристрої користувача, що суттєво економить час.

Використання принципів Material Design 3 [7] дозволяє реалізувати гнучкі та інтуїтивно зрозумілі інтерфейси, які легко адаптуються під різні розміри екранів, включно з можливістю зручного керування однією рукою, що є надзвичайно важливим під час роботи на будівельному об’єкті.

Для наочного представлення взаємодії обраних засобів розробки та розподілу їхніх зон відповідальності в межах системи, було спроєктовано концептуальну схему інтеграції технологій за архітектурними шарами (рис. 1.1).

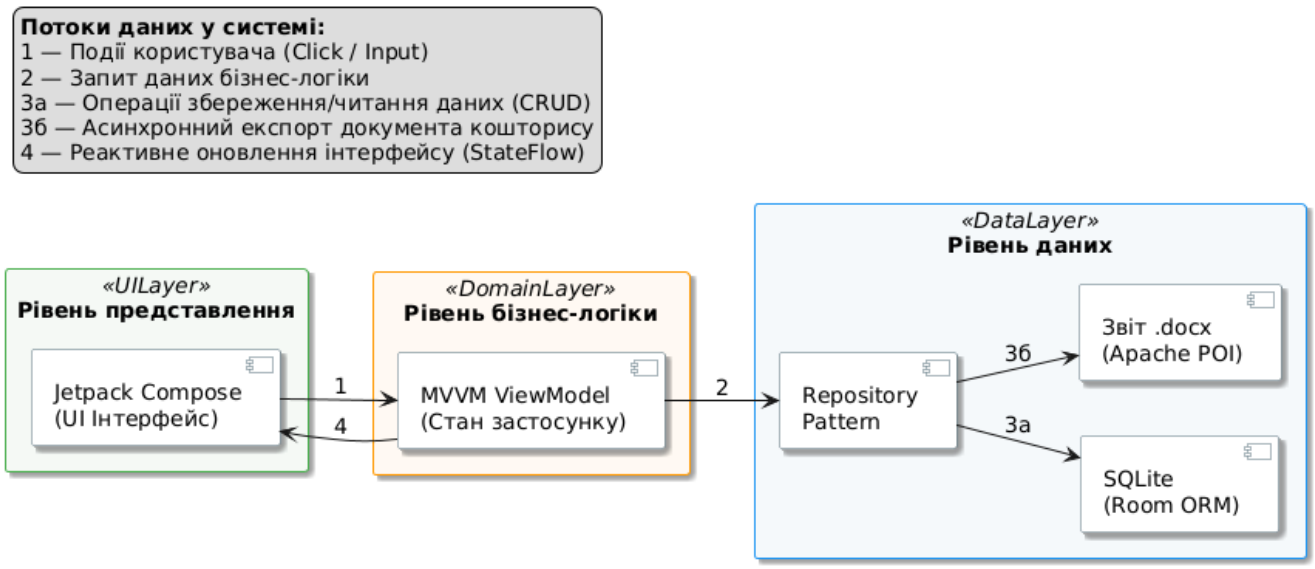


Рисунок 1.2 – Схема інтеграції технологій та взаємодії шарів застосунку

Взаємодія між рівнями базується на принципі односпрямованого потоку даних (UDF). Рівень представлення (Jetpack Compose, Material 3) передає події користувача у ViewModel (Domain Layer), яка повертає оновлений стан через потоки StateFlow.

Зв’язок із Data Layer ізольовано паттерном Repository. Репозиторії координують роботу двох підсистем: ORM Room для SQL-транзакцій у SQLite та бібліотеки Apache POI для асинхронної генерації кошторисів у форматі .docx. Така архітектура забезпечує повну офлайн-автономність застосунку, поєднуючи високу швидкість обчислень із гнучкістю його використання в польових умовах [17].

2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ RENOVUM

Другий розділ присвячено детальному проєктуванню архітектури системи Renovum, що охоплює розробку структури даних, визначення основних сценаріїв використання, моделювання бази даних, а також архітектуру навігації та інтерфейсу. Особливу увагу приділено логіці автоматизованих розрахунків параметрів приміщень, механізмам формування звітної документації та правилам обробки бізнес-даних, що в сукупності забезпечують ефективну функціональність застосунку в «польових» умовах роботи майстра.

2.1 Загальна архітектура системи

Систему Renovum побудовано з використанням багаторівневої архітектури та патерну MVVM (Model-View-ViewModel) [18, 19]. Такий підхід забезпечує чітке розділення відповідальності між інтерфейсом користувача, бізнес-логікою та рівнем збереження даних, що сприяє високій підтримуваності та масштабованості системи [18, 20].

Архітектура застосунку складається з наступних взаємопов'язаних шарів:

1. UI / View Layer (Рівень представлення): реалізований за допомогою NavGraph для управління навігацією між екранами. Складається з нижньої навігаційної панелі BottomNav та бокового меню AppDrawer, а також додаткових екранів для роботи з даними. Всі вони побудовані на базі Jetpack Compose [3, 5].
2. Domain / ViewModel Layer (Рівень стану): центральним компонентом є RoomViewModel, який виступає єдиним джерелом істини для UI-шару. Він обробляє користувацькі події та координує потік даних між рівнем представлення та рівнем даних.
3. Data Layer (Рівень даних): поділяється на дві основні гілки:
    - Repositories: класи RoomRepository та WorkRepository, які забезпечують взаємодію з локальною базою даних.

- Assets: репозиторій WorkDataRepository, що відповідає за зчитування статичних даних з локального файлу Services.json.

4. Local Database (Рівень збереження): базується на бібліотеці Room (SQLite) [8]. Включає DAO-інтерфейси та базу даних AppDatabase [10].

Така структура забезпечує ізоляцію бізнес-логіки від особливостей реалізації UI, що є критично важливим для стабільності роботи в «польових» умовах [18, 19]. Використання репозиторіїв дозволяє ефективно керувати потоками даних, забезпечуючи їх узгодженість між локальним сховищем та статичними ресурсами застосунку.

MainActivity запускає кореневий Compose-компонент застосунку — RenovumApp, який виступає глобальним контейнером для всього інтерфейсу користувача [3, 5]. Перед цим у методі onCreate ініціалізуються базові залежності шару даних, зокрема статичний репозиторій WorkDataRepository, що завантажує структуру робіт із локального файлу конфігурації. Далі компонент NavGraph керує всіма переходами між екранами системи, використовуючи NavHostController [5]. Він визначає стартовий маршрут Screen.Rooms.route для відображення списку кімнат, а також забезпечує гнучку навігацію усіх інших екранів застосунку залежно від подій та стану, що транслюються через єдине джерело істини RoomViewModel[4, 18]. При цьому передача даних між архітектурними шарами відбувається у вигляді реактивних потоків StateFlow, що гарантує миттєве оновлення графічних компонентів Jetpack Compose при зміні стану моделей. Завдяки повній ізоляції шару даних через патерн Repository, бізнес-логіка застосунку залишається абсолютно незалежною від платформних засобів відображення інтерфейсу користувача. Таке архітектурне рішення мінімізує зв'язність модулів системи та забезпечує її безперебійне й стабільне функціонування в офлайн-режимі.

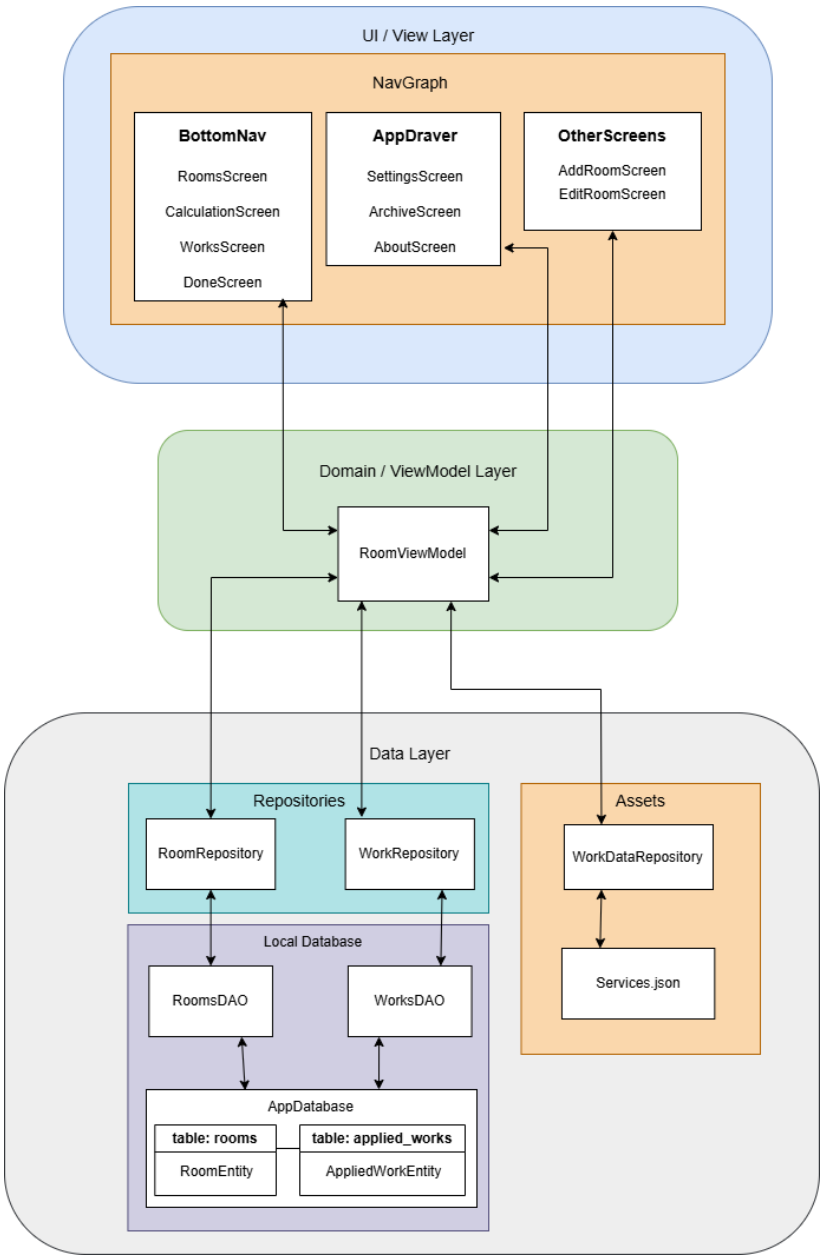


Рисунок 2.1 – Архітектура мобільного застосунку Renovum

Перевагою такої архітектури є те, що UI-шар не містить прямого доступу до бази даних чи локальних файлів конфігурації, що повністю відповідає принципу інкапсуляції [3, 18, 19]. Наприклад, екран додавання кімнати AddRoomScreen не звертається безпосередньо до локальної бази даних Room. Замість цього він передає введені користувачем дані (геометричні параметри та форму приміщення) у RoomViewModel, яка координує бізнес-логіку та викликає RoomRepository [4]. Якщо ж користувачеві необхідно обрати будівельні роботи для конкретного приміщення, RoomViewModel взаємодіє із WorkDataRepository для зчитування

статичних технологічних процесів із локального файлу конфігурації, а після фіксації виконання — передає сформовані дані у WorkRepository для збереження транзакції в базі даних AppDatabase [8, 10, 21].

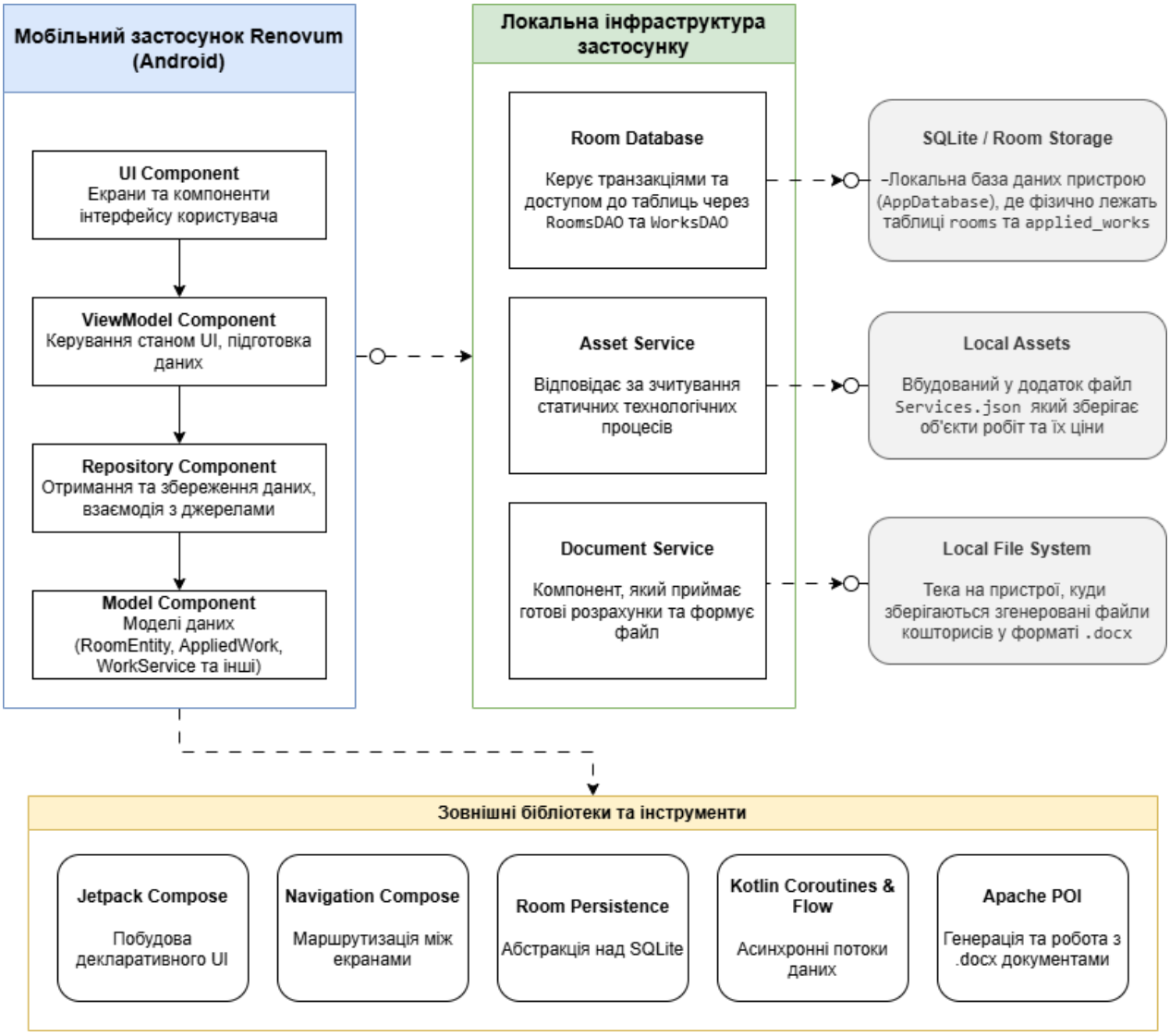


Рисунок 2.2 – Діаграма компонентів застосунку

Для систематизації архітектурних та технічних рішень, прийнятих на етапі проєктування системи, було сформовано цілісний набір інструментів розробки[4,17]. Повний перелік компонентів, що складають технологічний стек мобільного застосунку Renovum, із зазначенням конкретних версій програмного забезпечення та їх цільового призначення в системиі, наведено в таблиці 2.1 [14,15].

Таблиця 2.1 – Технологічний стек Renovum

Компонент	**Технологія**	**Призначення**
Мова програмування	Kotlin 2.4.0	Основна мова реалізації
UI	Jetpack Compose (BOM 2026.05.01), Material3 1.4.0	Побудова екранів і компонентів
Навігація	Navigation Compose 2.9.8	Переходи між екранами
Архітектура	MVVM, Repository Pattern	Розділення відповідальності
Управління станом	StateFlow, Kotlin Flow	Реактивне оновлення UI
Асинхронність	Kotlin Coroutines	Фонові операції
База даних	Room Persistence Library 2.8.4	Збереження вимірів, кімнат та виконаних робіт
Робота з локальними активами	Gson 2.14.0	Робота з вбудованим конфігураційним файлом Services.json зі списком послуг
Збереження конфігурацій	DataStore Preferences 1.2.1	Локальне збереження користувацьких налаштувань та системних прапорців
Формування звітності	Apache POI / POI-OOXML 5.5.1	Динамічна локальна генерація кошторисів у форматі .docx
Збірка проєкту	Gradle Kotlin DSL, AGP 9.0.1	Конфігурація, керування залежностями та автоматизація збірки Android-проєкту

Наведений технологічний стек базується на офіційно рекомендованих інструментах сучасної Android-розробки, що гарантує високу продуктивність системи та тривалий життєвий цикл її підтримки. Використання виключно нативних рішень у поєднанні з потужними бібліотеками асинхронної обробки й локального збереження дозволило створити повністю автономне інженерне середовище, оптимізоване під обмежені апаратні ресурси мобільних пристроїв.

2.2 Проєктування сценаріїв використання

Основні сценарії використання мобільного застосунку Renovum охоплюють повний життєвий цикл взаємодії користувача з локальною системою розрахунків та менеджменту будівельних робіт [18, 19]. Оскільки програмний комплекс розроблено як інструмент персональної продуктивності майстра, всі сценарії орієнтовані на максимальну автономність, швидкість обробки даних та мінімізацію

ручного введення інформації «в польових» умовах [17, 32].Логіка роботи системи деталізується у двох ключових користувацьких сценаріях.

Сценарій основного робочого циклу є базовим процесом функціонування застосунку й складається з наступних послідовних кроків:

- Ініціалізація та конфігурація об'єкта: Після запуску застосунку користувач повинен створити ремонт, ввівши адресу ремонту у відповідне поле, після чого він переходить до створення картки приміщення на екрані RoomsScreen. За замовчуванням система пропонує базову прямокутну форму кімнати, де майстер вводить назву приміщення та його лінійні параметри (довжину, ширину, висоту), проте доступні також Г-подібна та Т-подібна форми.
- Врахування архітектурних особливостей: Для забезпечення високої точності розрахунків користувач додає параметри технологічних прорізів (вікон та дверей), які згодом будуть автоматично відняті від загальної площі стін під час обчислення обсягів робіт.
- Фіксація та верифікація вимірів: Після збереження даних у локальній базі даних Room [8], об'єкт з'являється у загальному списку. Шляхом кліку на картку кімнати або через перехід до екрана CalcScreen користувач може миттєво переглянути всі введені лінійні виміри та автоматично обчислені геометричні параметри (площу підлоги, периметр, чисту площу стін та інші).
- Призначення технологічних процесів: На екрані WorksScreen майстер обирає необхідні типи ремонтних робіт, що завантажуються із системного файлу Services.json. Система реалізує інтелектуальну підтримку введення: додаток демонструє середню ціну та рекомендований діапазон вартості для кожної послуги. При збереженні роботи без ручного заповнення полів, застосунок автоматично підставляє середню ціну та пов'язаний параметр кімнати (наприклад, площу стін для штукатурки або площу підлоги для укладання плитки).

- Фінансовий аналіз та генерація звітності: На екрані виконаних робіт DoneScreen акумулюється покімнатний та загальний кошторис об'єкта. Майстер може бачити вартість окремих операцій, сумарну вартість по кожній кімнаті та підсумкову ціну всього ремонту, на яку за потреби можна нарахувати відсоткову знижку. Кінцевою точкою сценарію є локальний запуск процесу генерації детального неофіційного звіту у форматі Microsoft Word за допомогою бібліотеки Apache POI [13].

Для концептуального аналізу взаємодії компонентів системи під час виконання кроків основного сценарію було розроблено узагальнену діаграму послідовності (рис. 2.3).

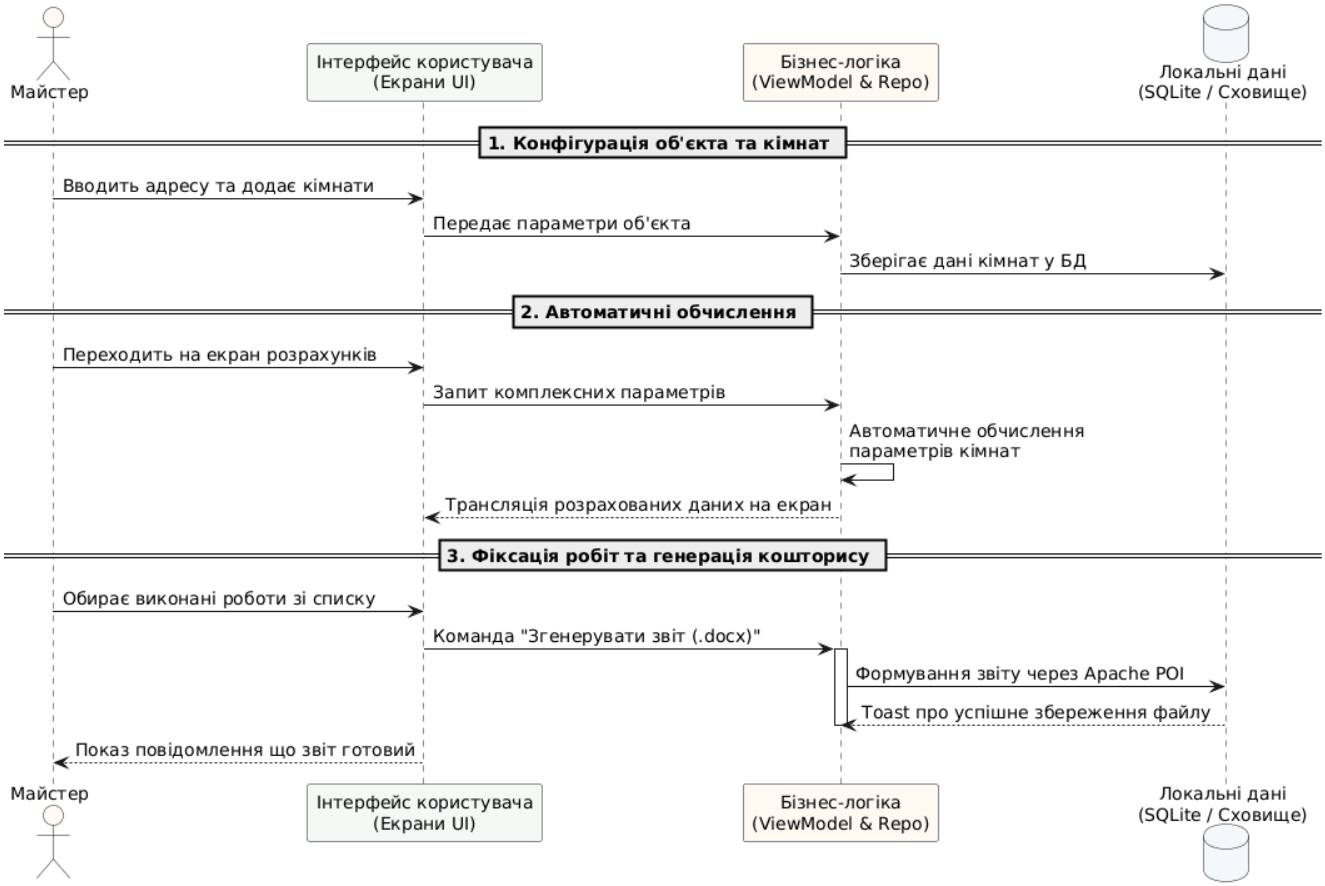


Рисунок 2.3 – Узагальнена діаграма послідовності сценарію основного робочого циклу

Представлена діаграма (рис. 2.3) відображає високорівневу динаміку передачі даних між ключовими архітектурними рівнями застосунку без

перевантаження схеми низькорівневим кодом. На схемі зафіксовано, як користувач взаємодіє з екранами UI, вводячи геометричні дані та параметри прорізів. Шар представлення делегує обробку подій модулю бізнес-логіки (ViewModel & Repository), який запускає утиліти автоматичних математичних розрахунків площ і периметрів приміщень. Обчислені доменні моделі асинхронно записуються в базу даних, після чого реактивні потоки оновлюють стан інтерфейсу. Повна інженерна діаграма послідовності із деталізацією внутрішніх викликів функцій (зокрема addRoom, saveAppliedWork, generateWordReportInBackground) та взаємодії між конкретними екранами наведена у Додатку В.

Сценарій керування архівом та дистрибуції звітів стає доступним після успішного виконання основного робочого циклу та створення хоча б одного документа. Він описує механізми взаємодії з готовою документацією:

- Користувач переходить на спеціалізований екран ArchiveScreen, де система відображає інтерактивний список усіх раніше згенерованих кошторисів.
- Для кожного файлу в архіві реалізовано функції швидкого перегляду, повного видалення запису з бази даних та повторного завантаження документа у виділену системну директорію пристрою за шляхом /Documents/Renovum [32].
- Експорт та поширення документа реалізовано через інтеграцію з системним компонентом Android SDK — Intent.createChooser за підтримки механізму безпечного доступу FileProvider [4, 6]. Це дозволяє користувачеві безпосередньо з інтерфейсу застосунку передати згенерований .docx файл у будь-які зовнішні канали комунікації: надіслати через месенджери, відправити електронною поштою клієнту, зберегти на особистий хмарний диск або передати по бездротовому протоколу Bluetooth.

Для візуалізації логіки взаємодії застосунку із системними компонентами Android OS під час роботи з архівом побудовано концептуальну діаграму послідовності (рис. 2.4).

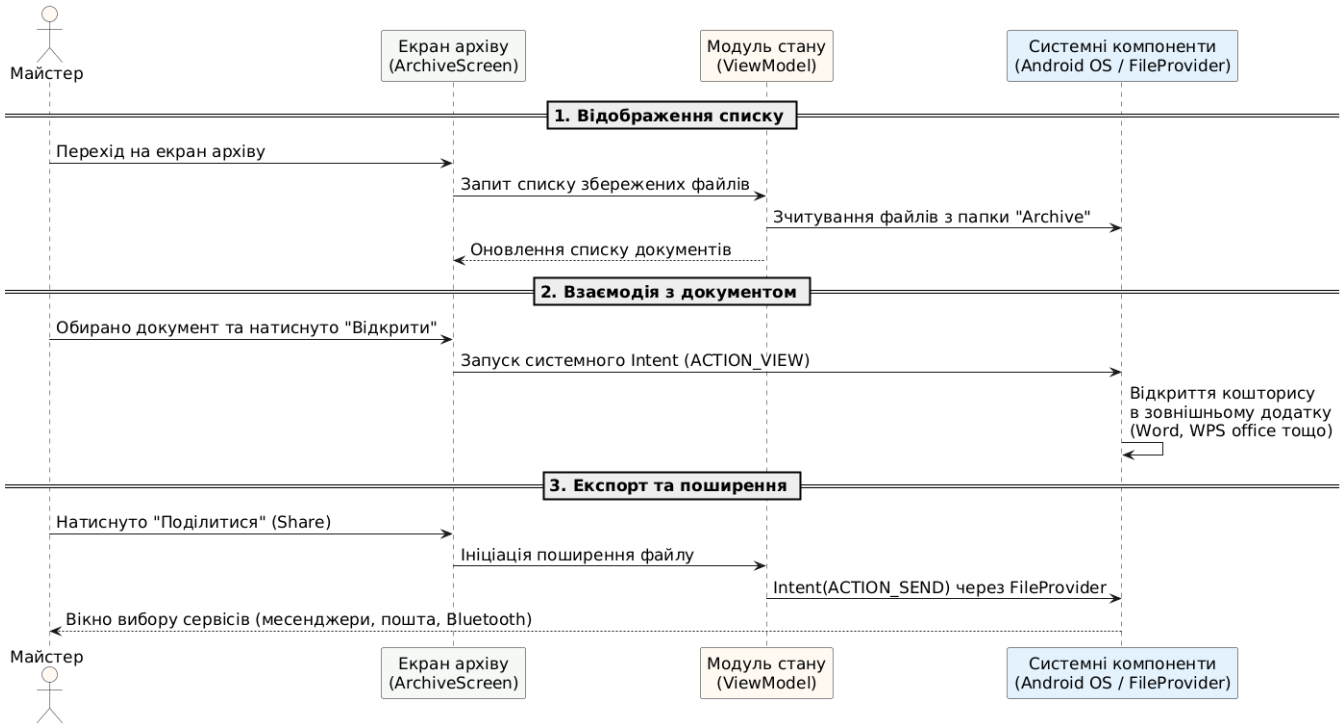


Рисунок 2.4 – Узагальнена діаграма послідовності процесу експорту та поширення звіту з архіву

Ця діаграма (рис. 2.4) ілюструє узагальнену логіку роботи підсистеми архівування та поширення звітів. Під час ініціації команди «Поділитися» або «Відкрити», модуль стану (ViewModel) зв'язується через репозиторій із файловою системою пристрою та перевіряє наявність сформованого файлу .docx [13]. Далі шар представлення через контекст застосунку комунікує з операційною системою Android OS. За допомогою механізму безпечного доступу FileProvider створюється захищений URI-посилач на файл, що передається у системне вікно вибору програм (Intent.createChooser), яке делегує фінальну дистрибуцію документа стороннім мобільним сервісам. Повну технічну діаграму послідовності з відображенням життєвого циклу діалогових вікон, утиліт та специфічних методів (loadArchiveFiles, openFile, shareFile) наведено у Додатку Д.

2.3 Моделювання доменної області та моделей даних

Архітектура мобільного застосунку Renovum базується на концепції розділення відповідальності, де усі сутності чітко розмежовані за своєю роллю в системі [18, 22]. На відміну від систем із централізованими хмарними базами даних, де консистентність даних контролюється сервером, у розробленому офлайн-застосунку об'єктна модель безпосередньо відображає структуру локального сховища та логіку математичних обчислень геометрії приміщень [8, 10].

Усі моделі даних застосунку можна класифікувати на три ключові категорії:

- Об'єктно-реляційні сутності баз даних (@Entity): Ключові моделі, що зберігаються у локальній базі даних Room (RoomEntity, AppliedWork).
- Моделі бізнес-логіки та звітів (data class, sealed class): Ключові класи, що обслуговують обчислювальні процеси та експорт (CalculatedData, ReportData, UserSettings, RoomParams).
- Технологічні переліки (enum class): Сутності, що типізують геометричні форми, типи прорізів, одиниці виміру та категорії будівельних робіт.

Узагальнене призначення основних моделей даних мобільного застосунку наведено у таблиці 2.2.

Таблиця 2.2 – Основні моделі даних Renovum

Модель даних	**Тип структури**	**Призначення моделі в системі**
1	2	3
RoomEntity	data class (@Entity)	Картка приміщення об'єкта (назва, форма, лінійні виміри, список прорізів)
OpeningEntity	data class	Параметри технологічного прорізу в стінах (вікно або двері)
AppliedWork	data class (@Entity)	Фіксація виконаної технологічної операції, прив’язана до конкретної кімнати
WorkService	data class	Опис будівельної послуги з довідника (цінові діапазони, тип виміру)
RoomParams	sealed class	Поліморфна математична модель для обчислення площ прямокутних, Г- та Т-кімнат
CalculatedData	data class	Агреговані результати чистих геометричних обчислень об'єкта

Продовження таблиці 2.2

1	2	3
ReportData	data class	Структурований набір даних, підготовлений для передачі в генератор файлів Word
UserSettings	data class	Локальні параметри профілю майстра, адреси об'єкта та налаштувань інтерфейсу

Низькорівнева реалізація структури локального сховища Room визначається описом головної сутності кімнати RoomEntity. Вона містить унікальний ідентифікатор у форматі тексту UUID, назву, тип форми, вкладений об'єкт параметрів та динамічний список архітектурних прорізів.

Типізація прорізів здійснюється за допомогою стандартного переліку OpeningType, який містить у собі рядкову назву елемента для виведення в інтерфейс програми користувача.

Для забезпечення гнучкості та можливості розширення підтримуваних архітектурних конфігурацій об'єктів без зміни структури таблиць бази даних, обчислення площі підлоги, стелі, брутто та нетто площі стін реалізовано через поліморфний базовий клас RoomParams з використанням механізмів sealed-класів в мові Kotlin. Для типізації форм приміщень застосовується перелік RoomShapeType.

Результати проміжних вимірів, які не потребують персистентного зберігання в базі даних SQLite, але динамічно використовуються під час роботи екрана CalcScreen та логіки автоматичного призначення обсягів, інкапсульовані в проміжну модель CalculatedData.

Другим центральним вузлом бази даних є сутність AppliedWork. Вона моделює реляційний зв'язок «один-до-багатьох» між кімнатою та доданими до неї технологічними операціями з будівельного довідника послуг. Сутність містить обмеження ForeignKey з каскадним видаленням даних CASCADE для очищення пов'язаних робіт у разі видалення кімнати майстром.

Будівельні послуги, що зчитуються з файлу конфігурації Services.json, описуються за допомогою класу даних WorkService. Зв'язок між роботою та типом

вимірюваної поверхні встановлюється за допомогою переліків TargetSurface та WorkUnit.

Дані профілю користувача та поточного стану активного будівельного об'єкта об'єднані в модель UserSettings. Відсутність заповненого поля currentObjectAddress сигналізує архітектурним компонентам системи про відсутність активного кошторису і перемикає інтерфейс головного екрана в стан очікування створення нового проєкту.

2.4 Проєктування структури локальної бази даних Room

Для надійного збереження даних у мобільному застосунку Renovum використано об'єктно-реляційну бібліотеку-обгортку Room, яка функціонує поверх вбудованої в операційну систему Android реляційної бази даних SQLite [8, 10]. Структура локальної бази даних побудована на окремих таблицях (сутностях), кожна з яких відповідає певному типу об'єктів предметної області.

Основними таблицями в локальному сховищі є:

- rooms;
- applied_works.

Таблиця rooms зберігає створені користувачем об'єкти приміщень. Таблиця applied_works використовується для фіксації обсягів будівельних робіт, призначених майстром для конкретної кімнати.

Таблиця 2.3 – Структура таблиць локальної СУБД Room

Назва таблиці	**Призначення**	**Основні дані**
rooms	Картки приміщень об'єкта	id, name, shapeType, params (JSON), openings (JSON)
applied_works	Призначені будівельні роботи	id, workId, roomId (FK), quantity, priceAtTime

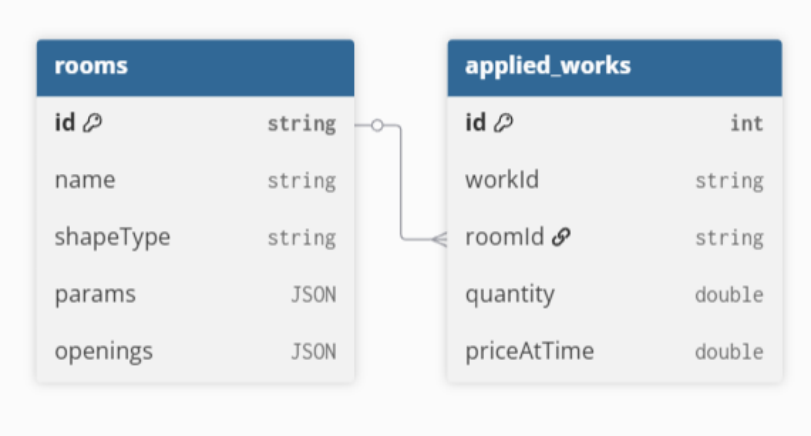


Рисунок 2.8 – Структура локальних таблиць бази даних Room

Реляційна модель сховища Room має свої архітектурні особливості, які безпосередньо вплинули на проєктування доменного шару додатка [9]. З погляду класичної теорії реляційних баз даних, спроєктована структура відповідає вимогам третьої нормальної форми (3NF): усі неключові атрибути таблиць є взаємно незалежними та повністю залежать лише від первинного ключа (Primary Key). Зв'язок між сутностями реалізовано за реляційним принципом «один до багатьох» (один об'єкт rooms містить декілька записів applied_works) за допомогою зовнішнього ключа roomId (Foreign Key) [40].

Оскільки СУБД SQLite нативно не підтримує збереження комплексних об'єктно-орієнтованих структур мови Kotlin, таких як поліморфні класи параметрів геометрії RoomParams чи динамічні списки прорізів List<OpeningEntity>, у систему було впроваджено сучасний підхід гібридизації даних за допомогою конвертацій типів (TypeConverters) [12]. За допомогою бібліотеки Google Gson складні вкладені об'єкти автоматично серіалізуються у текстовий формат JSON і записуються в базу даних як звичайні рядкові значення типу TEXT (String). Під час читання даних Room автоматично десеріалізує ці рядки у зворотні об'єктні типи даних мови Kotlin.

Таке інженерне рішення дозволило зберегти фізичну простоту таблиць та уникнути надлишкового створення додаткових 3-4 реляційних таблиць зв'язків. Це повністю позбавило систему потреби виконання важких операцій об'єднання (JOIN) на обмежених ресурсах мобільного процесора

Для забезпечення цілісності даних та оптимізації швидкодії на рівні таблиць використовуються реляційні обмеження та індексація [8]. Для запобігання появі «аномалій» та записів-сиріт для зовнішнього ключа налаштовано правило каскадного видалення onDelete = ForeignKey.CASCADE, завдяки якому видалення кімнати користувачем автоматично очищує всі прив'язані до неї кошторисні роботи. Додатково для поля roomId створено неколізійний індекс, що прискорює операції пошуку та фільтрації даних при побудові покімнатних звітів з лінійної складності $O(N)$ до логарифмічної $O(\log N)$ [10].

Окремого архітектурного рішення потребувало збереження та менеджмент базового довідника будівельних послуг, який містить фіксовані цінові діапазони, категорії (WorkCategory) та одиниці виміру (WorkUnit). Оскільки цей довідник є константним (статичним) та не підлягає модифікації користувачем у процесі роззрахунків, з метою оптимізації дискового простору пристрою та спрощення майбутніх міграцій бази даних, його було винесено із реляційного сховища Room у локальний конфігураційний файл services.json, що зберігається в директорії assets застосунку.

Доступ до цих даних організовано через паттерн проектування «Репозиторій» в об'єкті WorkDataRepository [18, 20]. Під час ініціалізації застосунку метод init() одноразово зчитує та десеріалізує JSON-структуру будівельних послуг у оперативну пам'ять додатка, інкапсулюючи логіку фільтрації робіт за секціями (WorkSection) та цільовими поверхнями обчислень (TargetSurface).

Таким чином, розроблена локальна архітектура збереження є повністю достатньою та оптимальною для мобільної системи. Вона успішно поєднує строгі транзакційні переваги реляційної СУБД SQLite для динамічних даних користувача із високою швидкістю зчитування статичних ресурсів без виконання накладних SQL-запитів до дискового простору пристрою [40].

2.5 Проєктування логіки навігації та інтерфейсу користувача

Логіка переміщення користувача між функціональними компонентами мобільного застосунку Renovum побудована на основі декларативного фреймворку Navigation Compose [5]. На відміну від класичного імперативного підходу з використанням ідентифікаторів ресурсів (R.id), дане архітектурне рішення використовує концепцію строго типізованих маршрутів, описаних за допомогою ієрархії запечатаних класів (sealed class), що забезпечує високу надійність контролю типів на етапі компіляції програми [4].

Центральним ядром опису навігаційних точок у застосунку є клас Screen, який інкапсулює у собі строковий ідентифікатор маршруту (route), текстову назву для відображення у графічних елементах (title) та відповідну векторну іконку з пакету Material Icons [7].

Ініціалізація та керування станом відображення екранів здійснюється у головному контейнері RenovumApp. На етапі запуску застосунку відбувається асинхронне зчитування налаштувань майстра через об'єкт UserSettingsManager, який використовує реактивні потоки даних Flow.

Для запобігання аномалій рендерингу (наприклад, відображення порожніх екранів до моменту зчитування конфігурації з диска), у системі реалізовано блокуючий фільтр: якщо об'єкт налаштувань повертає стан null, застосунок тимчасово відмальовує порожній компонент Box і очікує завершення операції введення-виведення. Граф навігації NavGraph використовує як початковий маршрут за замовчуванням екран списку об'єктів. Для деталізації топології переходів та візуалізації зв'язків між усіма екранами системи було розроблено загальну навігаційну схему застосунку (рис. 2.9). Вона наочно демонструє розподіл користувацьких маршрутів, а також показує точки інтеграції контекстних компонентів керування із глобальним графом навігації фреймворку

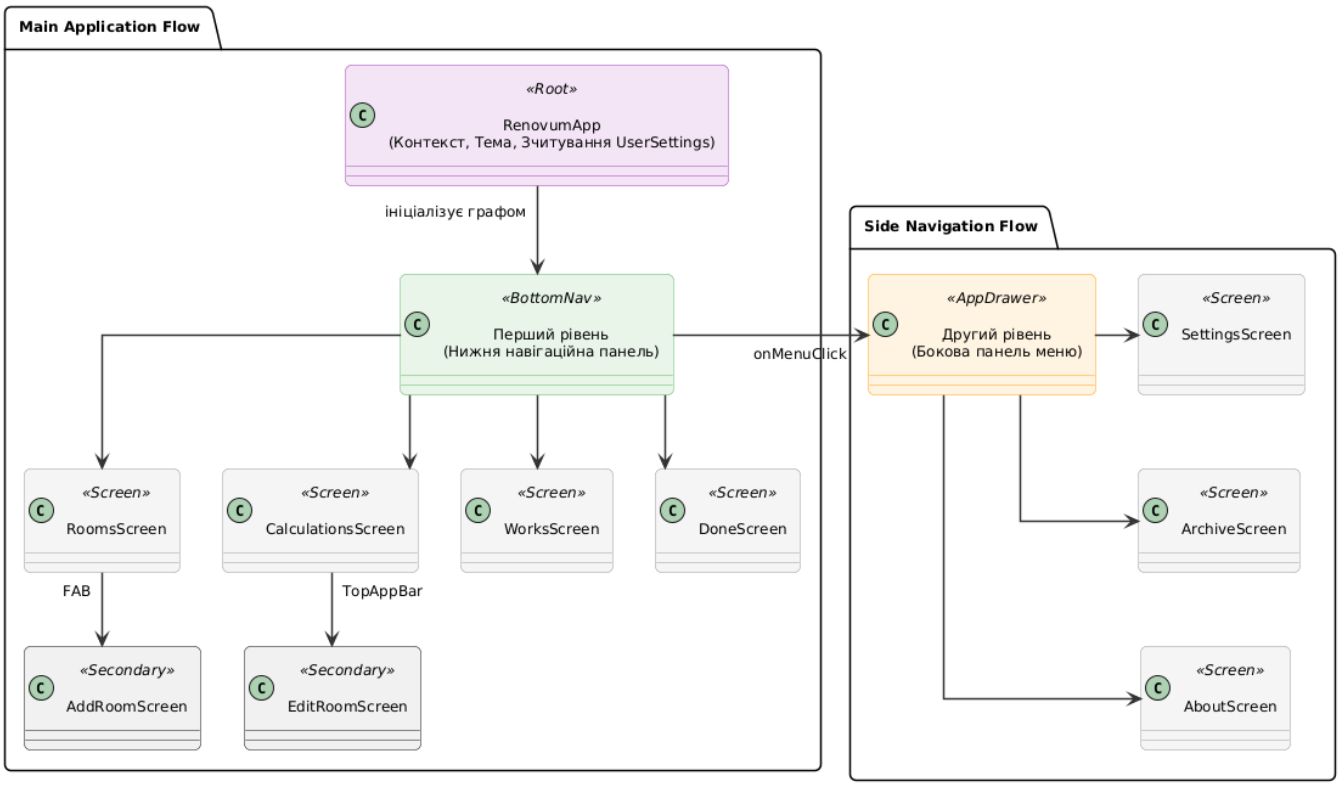


Рисунок 2.9 – Навігаційна схема застосунку

Графічний інтерфейс користувача побудовано за трирівневою системою розподілу навігаційного простору згідно з гайдлайнами Material Design 3, що дозволяє уникнути перевантаження екрана елементами управління [7].

- Перший рівень — BottomNav (Нижня навігаційна панель): Забезпечує миттєвий доступ до основних циклічних екранів розрахунку об'єкта. Сюди винесено чотири базові кроки технологічного процесу: керування кімнатами (RoomsScreen), перегляд геометричних вимірів приміщень (CalculationsScreen), призначення будівельних робіт з довідника (WorksScreen) та формування фінального кошторису з урахуванням знижок (DoneScreen). Для запобігання роздуванню стеку переходів при циклічних кліках по нижньому бару, в системі застосовано прапорець launchSingleTop = true та збереження стану через saveState = true/restoreState = true.
- Другий рівень — AppDrawer (Бокова панель навігації): Реалізований через компонент ModalNavigationDrawer. Він інкапсулює у собі глобальні

системні функції та екрани утилітарного призначення: налаштування інтерфейсу додатка (SettingsScreen), архів виконаних об'єктів (ArchiveScreen) та інформаційне вікно з даними про розробника (AboutScreen).

- Третій рівень – функціональні екрани: цей рівень містить у собі додаткові екрани для додавання кімнат (AddRoomScreen) та їх редагування (EditRoomScreen). Навігація на ці екрани здійснюється натисканням відповідних кнопок на екранах RoomsScreen (FloatingActionButton: AddRoom) та CalculationsScreen (TopAppBar: NavigateToEditRoom).

Унікальною ергономічною особливістю проєктування інтерфейсу користувача в Renovum є підтримка адаптивної доступності для майстрів із різною провідною рукою (лівша / правша). Будівельні обчислення часто проводяться безпосередньо на об'єктах ремонту, де взаємодія з мобільним пристроєм відбувається однією рукою в екстремальних умовах. Для цього у глобальний стейт додатка інтегровано параметр userSettings.isLeftHanded.

Завдяки механізму CompositionLocalProvider додаток динамічно змінює напрямок макетування інтерфейсу (LayoutDirection.Ltr або LayoutDirection.Rtl) виключно для керуючих елементів навігації.

Верхня панель застосунку RenovumTopAppBar також виступає інтерактивним контекстним елементом управління. Вона автоматично підлаштовує свій функціонал під поточний відкритий маршрут. На екрані Rooms та Archive вона активує тригер isEditMode, дозволяючи користувачеві масово видаляти записи чи переходити в режим пакетного редагування сутностей. На екранах Works верхній бар додає випадаючий швидкий селектор кімнат, що покроково спрощує користувацький сценарій.

На екрані Done панель виводить фінальну суму кошторису з урахуванням знижки (currentDiscountedSum) та дозволяє по кліку викликати діалогове вікно редагування відсотків дисконту showDiscountDialog. Така щільна інтеграція компонентів навігації з реактивним архітектурним шаром мобільних рішень

дозволяє побудувати надійний, відмовостійкий та сучасний інтерфейс автоматизації розрахунків будівельних робіт.

2.6 Проєктування бізнес-правил та алгоритмів розрахунку

Функціональна складова мобільного застосунку Renovum базується на забезпеченні точних інженерно-будівельних обчислень, автоматизації калькуляції витрат та суворому контролі цілісності введених користувачем даних. Оскільки застосунок орієнтований на автономне локальне використання без залучення зовнішніх серверних обчислювальних потужностей, уся бізнес-логіка, математичні моделі та алгоритми валідації інтегровані безпосередньо в архітектурний шар представлення (ViewModel) та репозиторії мобільного клієнта.

Математична основа обчислювальних алгоритмів додатка розподілена відповідно до геометричної конфігурації приміщень, що описується переліком типів RoomShapeType. Розрахунок площі підлоги ($S_{підлоги}$), периметра (P) та загальної площі стін ($S_{стін}$) виконується за чіткими геометричними правилами:

- Для базової прямокутної форми кімнати обчислення здійснюються за класичними лінійними формулами (2.1, 2.2, 2.3) на основі введених параметрів довжини (a), ширини (b) та висоти стелі (h):

$$S_{\text{підлоги}} = a \times b, \tag{2.1}$$

$$P = 2 \times (a + b), \tag{2.2}$$

$$S_{\text{стін}} = P \times h, \tag{2.3}$$

- Для складних архітектурних форм (Г-подібні, Т-подібні приміщення) алгоритм, закладений у бізнес-модель RoomParams, виконує автоматичну декомпозицію (розбиття) кімнати на кілька простих ортогональних прямокутників, після чого сумує їхні локальні площі та периметри для отримання фінального значення.

Для запобігання перевитрати будівельних матеріалів та формування коректного фінансового кошторису, в системі реалізовано алгоритм динамічного віднімання технологічних прорізів (вікон, дверей, арок). Чиста площа поверхні стін, яка безпосередньо підлягає обробці (штукатурка, шпалери, плитка), розраховується за наступним бізнес-правилом:

$$S_{\text{чиста}} = S_{\text{стін}} - \sum_{i=1}^{n} S_{\text{прорізу\_}i} \tag{2.4}$$

де $S_{\text{прорізу}} = w \times h$ (добуток ширини та висоти конкретного вікна чи дверей, доданих майстром до поточної кімнати).

Формування фінансових показників та розрахунок вартості кожної призначеної будівельної операції у RoomViewModel підпорядковується алгоритму калькуляції вартості за одиницю об'єму. Підсумковий кошторис усього об'єкта розраховується з урахуванням глобальної проєктної знижки ():

$$Cost_{\text{роботи}} = Q \times Price_{\text{фікс}}\,, \tag{2.5}$$

$$Sum_{\text{фінальна}} = \left(\sum_{j=1}^{m} C\ ost_{\text{роботи\_}j}\right) \times \left(1.0 - \frac{D}{100.0}\right) \tag{2.6}$$

де Q — обсяг виконаної роботи (введений вручну або прив'язаний до чистої площі поверхні), $\text{Price}_{\text{фікс}}$— фіксована вартість послуги, імпортована з локального довідника services.json на момент її призначення, а D — відсоток глобальної знижки на ремонтний об'єкт (projectDiscountPercent).

Для забезпечення стабільної роботи системи та виключення крашів додатка через некоректне введення даних користувачем, у застосунку Renovum визначено наступний перелік суворих бізнес-правил та обмежень:

1. Правило додатної геометрії: Будь-які лінійні виміри кімнати (довжина, ширина, висота) та габарити технологічних прорізів повинні бути строго

більшими за нуль. Введення від'ємних або нульових значень блокується шаром UI.

2. Правило фізичної цілісності поверхонь: Сумарна площа всіх доданих у кімнату вікон та дверей не може перевищувати або дорівнювати розрахованій загальній площі стін цієї кімнати. При спробі порушення цього ліміту система видає помилку валідації.
3. Правило фіксації комерційних цін: Ціна будівельної послуги копіюється з каталогу і жорстко записується в таблицю applied_works. Подальше оновлення або зміна цін у глобальному файлі services.json не має впливати на вже сформовані та збережені кошториси поточних об'єктів.
4. Правило лімітування дисконту: Значення проєктної знижки обмежується діапазоном від 0% (відсутня знижка) до 100% (безкоштовне виконання робіт).
5. Правило каскадного очищення пам'яті: Видалення сутності кімнати з локальної бази даних автоматично ініціює каскадний тригер СУБД SQLite для повного видалення всіх зв'язаних із нею призначених робіт із таблиці applied_works. Це запобігає появі «завислих» даних-сиріт.
6. Правило унікальності ідентифікації: Назви кімнат у межах одного об'єкта ремонту не обов'язково мають бути унікальними (майстер може створити дві кімнати з назвою «Коридор»). Первинна ідентифікація та зв'язування сутностей у базі даних відбувається виключно за прихованим унікальним системним індексом UUID.

Таким чином, запропонована система математичних моделей та логічних обмежень повністю покриває вимоги предметної області автоматизації будівельних замірів, гарантує точність обчислень та забезпечує високу стабільність і автономність роботи мобільного застосунку.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ RENOVUM

Третій розділ присвячено практичній розробці та тестуванню Android-застосунку Renovum. У ньому розглянуто структуру пакетів проєкту, програмну реалізацію реактивного UI на Jetpack Compose (включаючи адаптацію для лівші) та організацію локального збереження даних за допомогою Room DB. Також розділ містить лістинги ключових компонентів системи та результати тестування розробленого програмного забезпечення.

3.1 Структура Android-проєкту

Мобільний застосунок Renovum реалізовано сучасною мовою програмування Kotlin. Базовим пакетом проєкту є com.void_dev_ua.renovum. Для збирання проєкту, менеджменту залежностей та конфігурації модулів використовується інструментарій Gradle на базі декларативного скрипту Kotlin DSL та механізму Version Catalogs (libs.versions.toml) [39].

Програмна архітектура проєкту організована за чітким пакетним принципом розподілу логічних шарів системи:

- пакет data: відповідає за локальне збереження та обробку даних. Включає підпакет local/daos з інтерфейсами доступу до бази даних (RoomDao, AppliedWorkDao), головний клас бази даних AppDatabase та Converters. Також містить шар repositories для управління потоками інформації та менеджер експорту звітів WordExportManager. Локальні налаштування інтерфейсу зберігаються через UserSettings та UserSettingsManager.kt.
- пакет model: інкапсулює доменні моделі та сутності бази даних Room, такі як RoomEntity, OpeningEntity.kt, RoomParams, CalculationsData та специфічні переліки RoomShapeType, TargetSurface.
- пакет navigation: містить декларативну логіку переходів між екранами у файлі NavGraph.kt та типізовані маршрути класу Screen.

- пакет viewmodel: містить центральний компонент управління станом інтерфейсу та обчислювальними алгоритмами — RoomViewModel.
- пакет ui: об'єднує в собі повторно використовувані елементи (components), діалогові вікна (dialogs), кастомні відмальовщики (RoomSchemaPainter.kt), системні панелі (AppDrawer.kt, BottomNav.kt) та безпосередньо Compose-екрани застосунку (screens).
- пакет utility: відповідає за допоміжний системний функціонал, включаючи логування (Logger.kt), роботу з файловою системою (RenovumFileProvider) та локальні сповіщення (RenovumNotificationManager).
- пакет assets: містить статичний файл локального каталогу будівельних послуг та базових розцінок services.json.

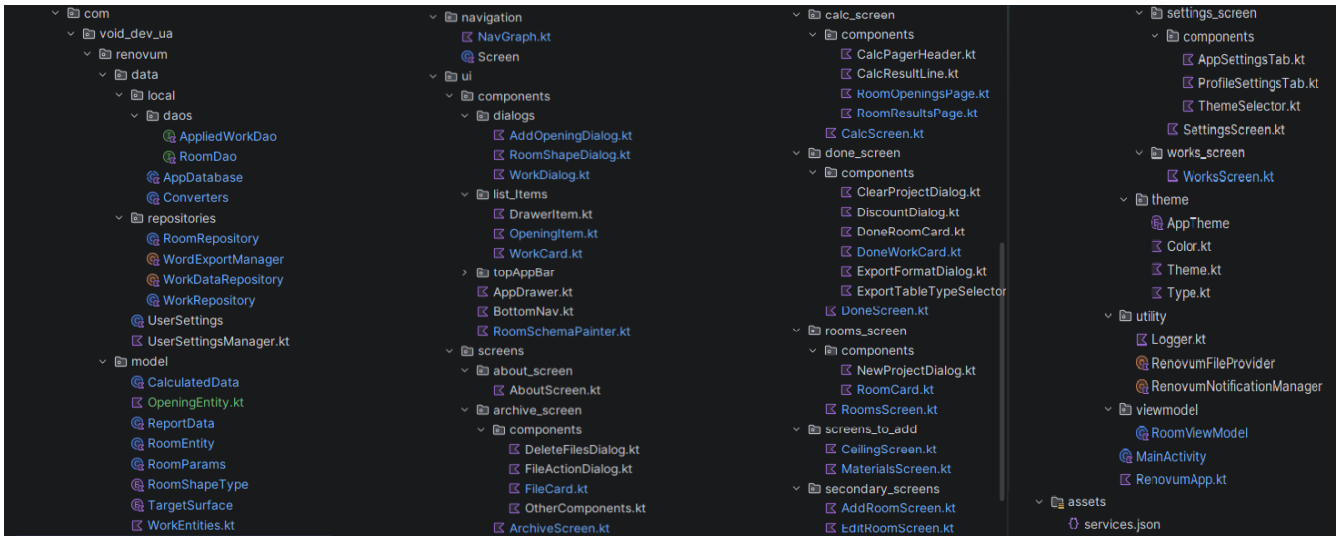


Рисунок 3.1 - Структура проєкту в Android Studio

Така модульна структура забезпечує незалежність компонентів (чисту архітектуру). Наприклад, модифікація дизайну карток будівельних робіт або діалогових вікон у пакеті ui жодним чином не зачіпає логіку калькуляції у RoomViewModel або структуру збереження таблиць у data [39].

3.2 Реалізація додавання ремонту та керування кімнатами

Створення нового будівельного об'єкта та формування структури приміщень реалізовано на першому ієрархічному рівні застосунку за допомогою взаємодії екрана RoomsScreen та діалогових компонентів валідації даних. Коли користувач запускає застосунок вперше, репозиторій повертає порожній список об'єктів, а інтерфейс відображає початковий стан екрана RoomsScreen із пропозицією ініціювати новий проєкт та ввести його адресу.

Процес ініціалізації об'єкта ремонту виконується через запуск компонента NewProjectDialog. Майстер вводить у текстове поле адресу або умовну назву об'єкта, після чого бізнес-логіка валідації перевіряє рядок на заповнення обов'язкових полів і наявність некоректних символів. Після підтвердження RoomViewModel через відповідний репозиторій фіксує новий будівельний майданчик, переводячи інтерфейс у стан готовності до макетування кімнат.

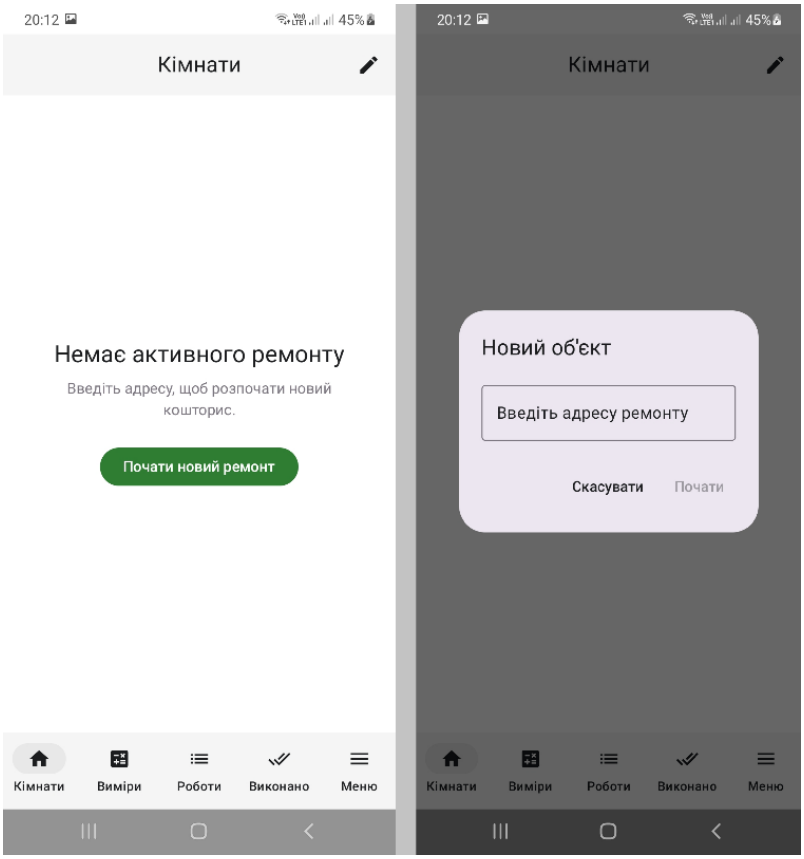


Рисунок 3.2 – Інтерфейс ініціалізації проєкту ремонту та діалогове вікно додавання адреси

Подальша розробка кошторису передбачає наповнення створеного об'єкта конкретними приміщеннями, що реалізовано через перехід на екран AddRoomScreen. Форма додавання кімнати вимагає від користувача введення назви (наприклад, «Кухня» або «Спальня») та вибору форми приміщення через RoomShapeDialog, що безпосередньо впливає на подальші геометричні розрахунки.

Для забезпечення високої надійності інтерфейсу на етапі додавання параметрів кімнати діє вбудована система валідації:

- Перевіряється обов'язкове текстове заповнення назви приміщення;
- Блокується збереження картки, якщо лінійні розміри нульові;
- Помилки та системні підказки відображаються українською мовою, що робить застосунок максимально орієнтованим на українських майстрів.

Після успішного натискання кнопки підтвердження, дані трансформуються в об'єкт RoomEntity і записуються в базу даних Room. Екран RoomsScreen реактивно оновлюється та відображає додані приміщення у вигляді структурованого списку інформаційних карток RoomCard із динамічним виведенням розрахованої площі та периметра для кожного елемента.

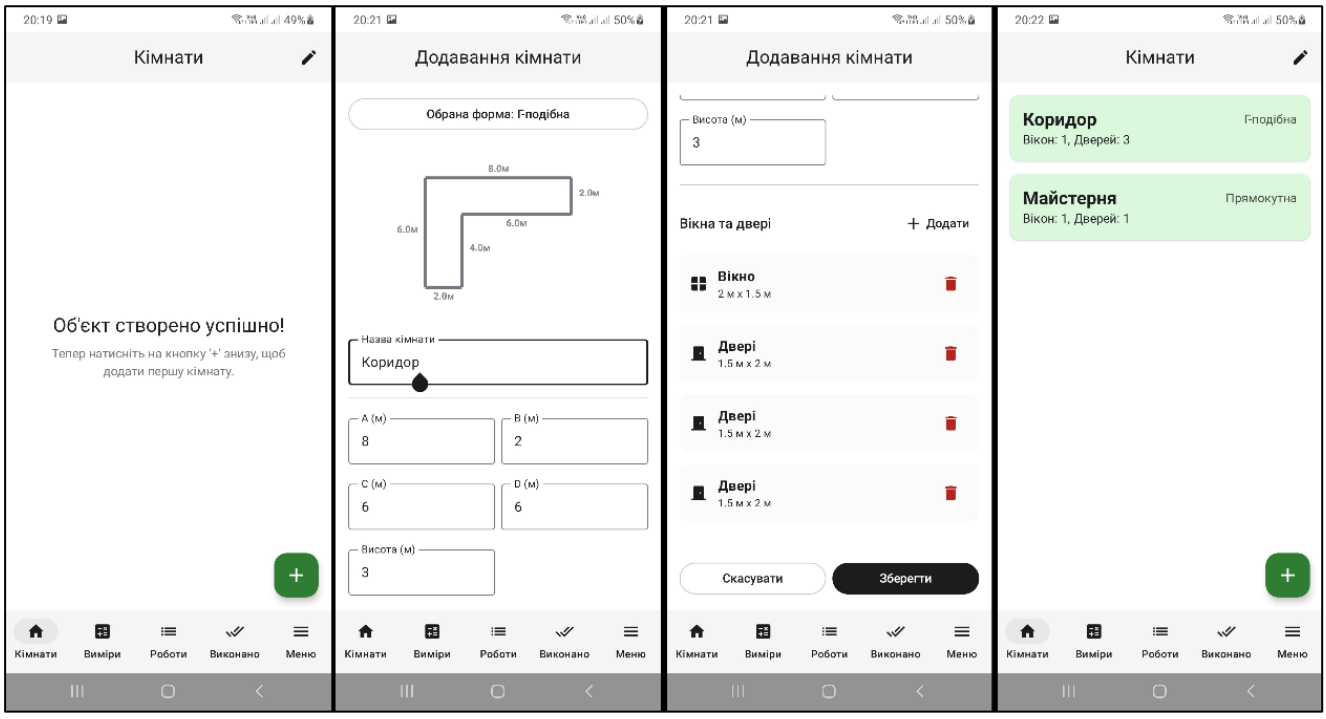


Рисунок 3.3 – Еволюція екрана RoomsScreen: порожній стан, вікно додавання кімнати та список створених приміщень

3.3 Реалізація обчислювального модуля та екрана геометричних замірів

Центральним функціональним ядром застосунку, відповідальним за математичну обробку лінійних параметрів приміщень, є екран CalcScreen. З метою оптимізації простору екрану для його компоновки використано HorizontalPager, що ділить робочий простір на дві взаємопов'язані функціональні вкладки:

- Сторінка підсумкових результатів замірів (RoomResultsPage): відображає інтерактивну таблицю з розрахованими геометричними показниками чистої та загальної площ поверхонь, об'єму кімнати, додаткових параметрів та введених користувачем базових значень.
- Сторінка технологічних прорізів (RoomOpeningsPage): призначена для динамічного формування списку вікон та дверей у межах поточної кімнати з можливістю виклику діалогу додавання AddOpeningDialog.

Коригування лінійних розмірів кімнати та прорізів здійснюється на окремому допоміжному екрані EditRoomScreen. Модифікація будь-якого параметра на цьому екрані ініціює реактивне оновлення об'єкта RoomParams у базі даних.

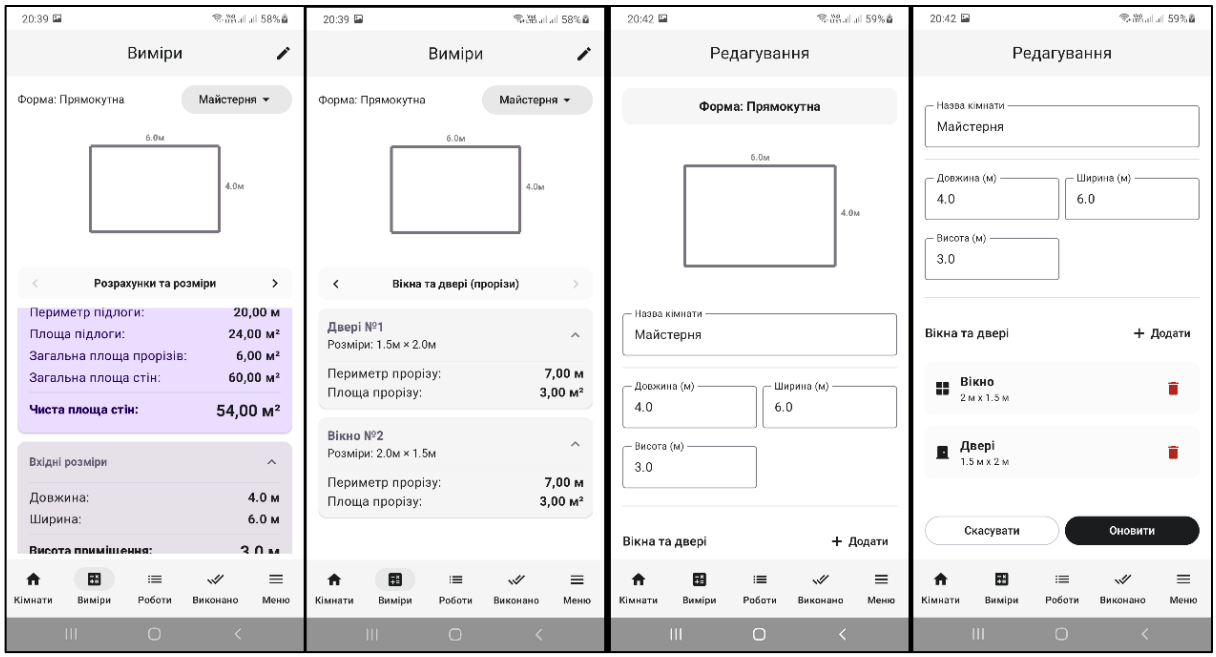


Рисунок 3.4 – Візуальний інтерфейс CalcScreen (вкладки прорізів та результатів) та екран редагування кімнати

Безпосередній математичний розрахунок площ та периметрів інкапсульовано в архітектурному шарі бізнес-логики застосунку [4]. Функція calculateRoomData отримує поточний об'єкт сутності кімнати, агрегує масив доданих прорізів за допомогою вбудованого оператора мови Kotlin sumOf [1], обчислює загальну площу стін на основі отриманого периметра та висоти стелі, а потім проводить операцію віднімання для формування фінальної чистої площі під подальше будівельне оздоблення. Програмну реалізацію цього калькуляційного алгоритму наведено в лістингу 3.1.

Лістинг 3.1 – Метод обчислення геометричних показників приміщення

```
fun calculateRoomData(room: RoomEntity): CalculatedData {
    val p = room.params
    val floorArea = p.getFloorArea()
    val perimeter = p.getPerimeter()

    val openingsArea = room.openings.sumOf { it.width.toDouble() *
it.height.toDouble() }
    val wallArea = (perimeter * p.roomHeight)
    val cleanWallArea = wallArea - openingsArea

    return CalculatedData(floorArea, wallArea, cleanWallArea,
openingsArea, perimeter, p.getExtraResults())
}
```

Обчислювальний конвеєр у шарі RoomViewModel використовує гнучку поліморфну модель даних на базі ізольованого класу (sealed class) RoomParams [2]. Таке архітектурне рішення дозволяє описати загальний інтерфейс для отримання базових інженерних характеристик приміщення (висоти стелі, периметра та площі підлоги), приховуючи специфіку обчислень конкретних архітектурних форм, що реалізовані через механізм спадкування в окремих вкладених дата-класах:

- RectangleParams — реалізує обчислення для стандартних ортогональних прямокутних кімнат на основі лінійних параметрів довжини та ширини;
- LShapedParams — забезпечує калькуляцію Г-подібних кімнат, де площа підлоги вираховується через декомпозицію та різницю площ суміжних секторів, а також додатково формує окремі результати для специфічних зон («площа шапки» та «площа виступу»);

- TShapedParams — інкапсулює найбільш складну геометричну логіку розрахунку Т-подібних приміщень, автоматично вираховуючи приховану висоту правого боку верху на основі перепадів довжини лівої та правої «ніжок», а також сумує площі трьох окремих частин для отримання підсумкового показника.

Зв'язок між шаром інтерфейсу представлення, де дані від користувача збираються у вигляді простої асоціативної карти рядків (Map), та типізованою бізнес-моделлю забезпечується фабричним методом супутнього об'єкта (companion object), який здійснює безпечний парсинг значень на основі обраного типу конфігурації RoomShapeType. Завдяки розробленому методу безпечної конвертації, додаток ізольовано обробляє некоректні текстові введення: у разі помилки розбору рядка система фіксує безпечне нульове значення, записує інформацію у внутрішній логер та запобігає аварійному завершенню роботи програми [18].

Сформований об'єкт розрахованих параметрів через механізм реактивних потоків даних StateFlow негайно транслюється в компоненти графічного інтерфейсу [3]. Це дозволяє шару представлення динамічно адаптувати свій стан і автоматично перебудовувати фінальні таблиці результатів на екрані CalcScreen.kt, забезпечуючи миттєве візуальне оновлення при будь-яких маніпуляціях майстра з розмірами чи прорізами об'єкта.

3.4 Реалізація модуля призначення технологічних робіт та формування розцінок

Управління фінансово-технологічним наповненням кошторису здійснюється на екрані WorksScreen. Програмна логіка цього модуля тісно пов'язана із поточним станом формування структури приміщень у RoomViewModel. Для забезпечення строгої узгодженості та цілісності даних у системі реалізовано динамічне бізнес-правило доступності інтерфейсу:

- Неактивний стан: якщо в поточному будівельному проєкті ще не створено жодної кімнати, у верхній частині екрана відображається попереджувальна картка з червоним контейнером помилки, а всі інтерактивні елементи та картки доступних послуг WorkCard примусово деактивуються, унеможливлюючи хаотичне призначення робіт без прив'язки до простору.
- Активний стан: після додавання хоча б одного об'єкта RoomEntity інтерфейс повністю розблоковується. Користувачеві відкривається ієрархічний деревоподібний каталог будівельних послуг, імпортований із локального ресурсу services.json та розділений на великі технологічні секції (наприклад, оздоблювальні роботи, демонтаж тощо) за допомогою горизонтальних перемикачів та випадаючих списків.

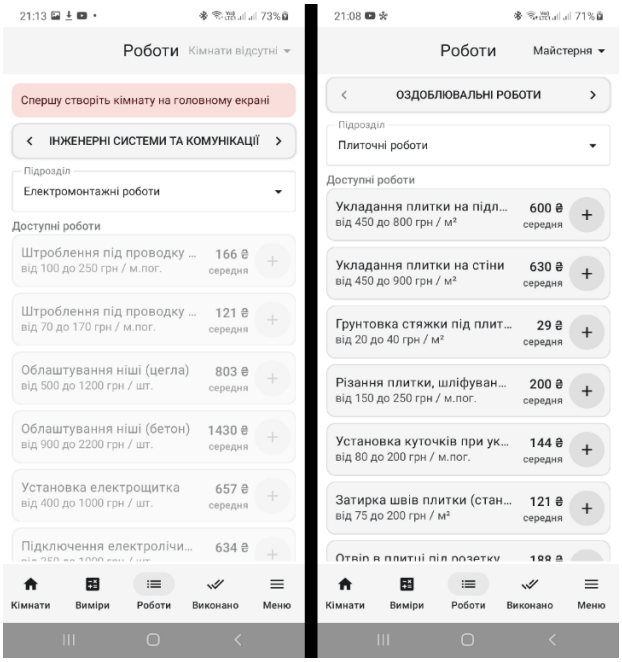


Рисунок 3.5 – Екран WorksScreen: стан блокування за відсутності кімнат та активний режим вибору послуг

При виборі конкретнго технологічного процесу та натисканні на кнопку додавання запускається двосторінковий діалоговий компонент WorkDialog. Його конструкція базується на горизонтальному пейджері HorizontalPager, що дозволяє майстру швидко перемикатися між двома інформаційними екранами ковзанням:

- Перша сторінка (Дані приміщення): відображає інтерактивний блок-акордеон (AccordionHeader) із рохрахованими параметрами та переліком вікон і дверей поточної кімнати, що вираховуються «на льоту» через виклик функції calculateRoomData у RoomViewModel.
- Друга сторінка (Введення параметрів): містить поля вводу для встановлення ціни за одиницю та об'єму робіт. Особливістю архітектури є автоматичне підтягування рекомендованого об'єму через метод getSurfaceValue (наприклад, якщо вибрано фарбування стін, додаток запропонує підставити значення чистої площі стін). Також на основі введених параметрів динамічно вираховується підсумкова сума.

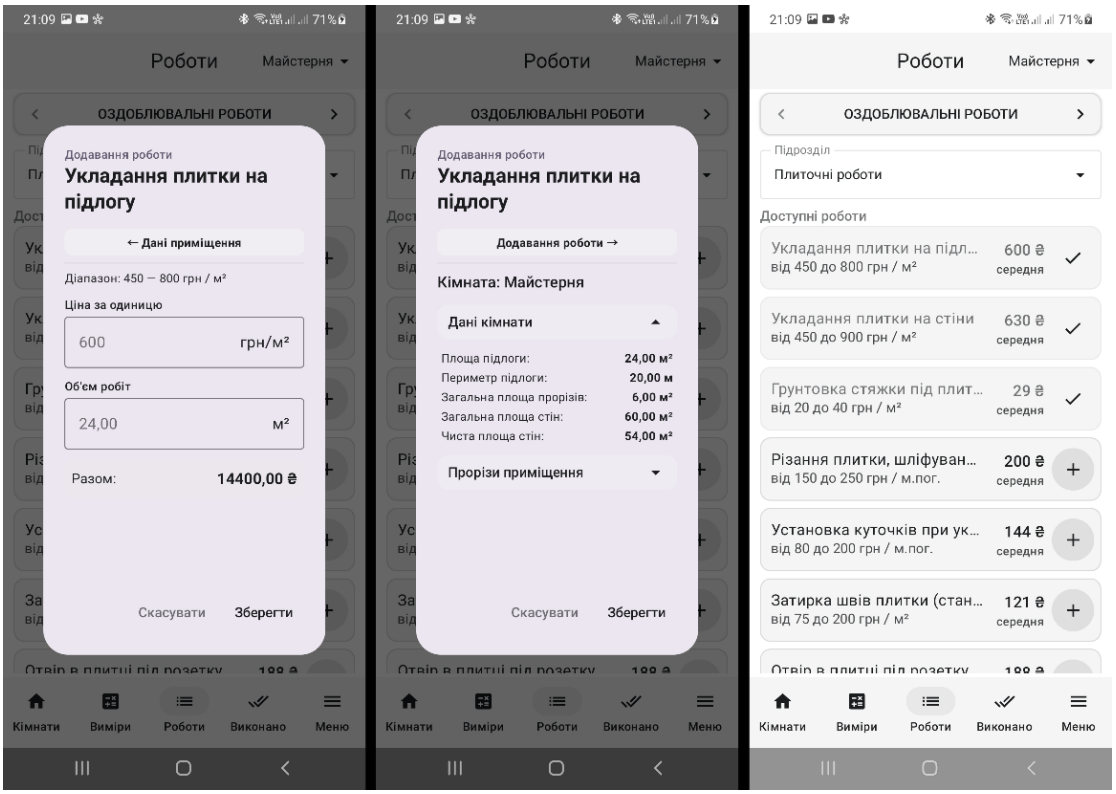


Рисунок 3.6 – Інтерфейс діалогового вікна WorkDialog та WorkScreen з призначенми роботами

Після підтвердження введення метод saveAppliedWork трансформує дані у сутність AppliedWork і фіксує її в локальній базі даних Room, що викликає реактивний перерахунок загальної вартості об'єкта (totalRawSumState) за допомогою реактивних потоків даних у межах концепції Clean Architecture.

3.5 Реалізація екрана фінального кошторису та інтерактивних інструментів управління проєктом

Кінцевим етапом консолідації та аналізу всіх проведених замірів і призначених будівельно-монтажних операцій у межах об'єкта є екран DoneScreen. Інтерфейс цієї сторінки спроєктовано як зведений інтерактивний звіт, який у реальному часі відображає фінансову структуру поточного проєкту.

Основною функціональною особливістю екрана є використання реактивного стану groupedWorksState з архітектурного шару RoomViewModel. Цей механізм автоматично групує всі виконані та збережені в базі даних Room технологічні операції (AppliedWork) за конкретними приміщеннями об'єкта (RoomEntity), паралельно підтягуючи текстові описи та метадані з репозиторію послуг WorkDataRepository. Користувач бачить чітку ієрархію у вигляді карток кімнат (DoneRoomCard) та вкладених у них карток робіт (DoneWorkCard) із зазначенням підсумкової вартості по кожній зоні (наприклад, окремо для майстерні чи коридору).

У верхньому лівому кутку екрана інтегровано динамічний індикатор загальної вартості, який відображає поточний стан розрахунків. Натискання на цей елемент активує діалогове вікно DiscountDialog, що запускає логіку коригування бюджету. Інтерфейс вікна виводить базову вартість, поле для введення відсотка знижки та автоматично прораховує фінальну суму до оплати, викликаючи метод updateDiscount у ViewModel для реактивного оновлення стану інтерфейсу.

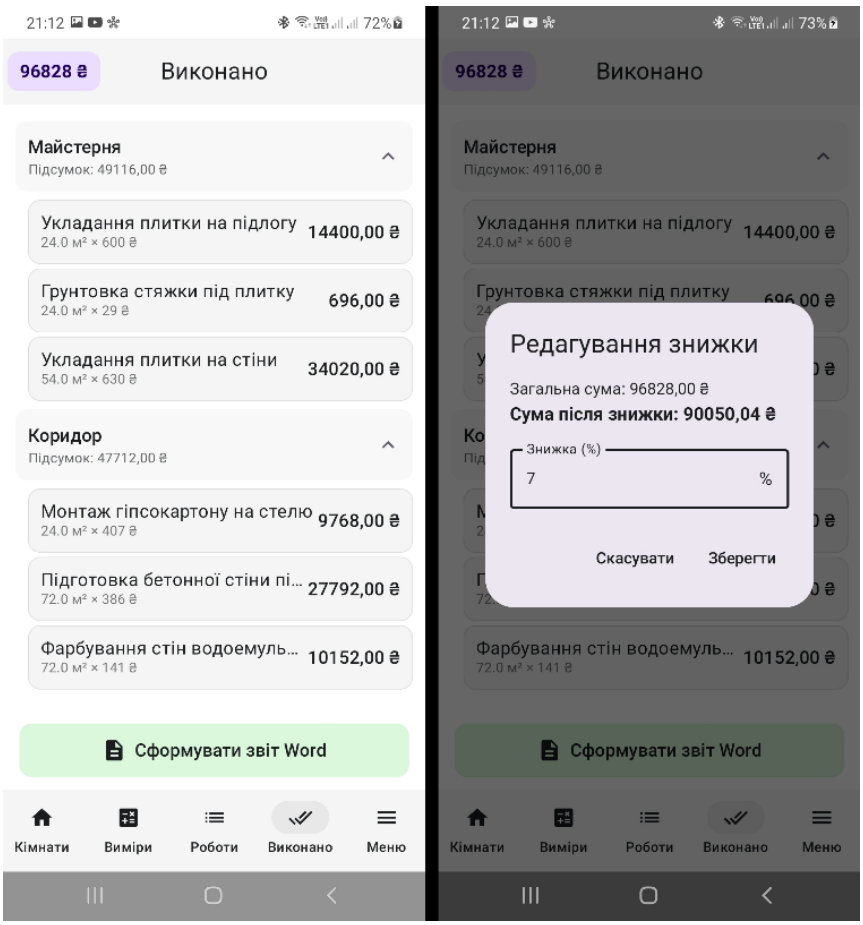


Рисунок 3.7 – Загальний інтерфейс екрана DoneScreen та діалогове вікно редагування знижки

Взаємодія з результатами проєкту на екрані фінального кошторису підтримує два критично важливі системні сценарії, які реалізовані через відповідні діалогові компоненти:

- Експорт сформованих даних: натискання на нижню кнопку «Сформувати звіт Word» ініціює запуск діалогу ExportFormatDialog. Це вікно дозволяє майстру перевірити чи відкоригувати поточну адресу будівельного майданчика, змінити назву підсумкового документа, а також гнучко налаштувати тип виводу даних за допомогою перемикача ExportTableTypeSelector — обравши або деталізований «Покімнатний» звіт, або консолідований «Загальний» варіант. При підтвердженні виклику запускається фоновий метод generateWordReportInBackground та створюється сповіщення генерації файлу.

- Повне очищення робочого простору: для завершення роботи над об'єктом або швидкого скидання параметрів передбачено виклик діалогу ClearProjectDialog одразу після успішного формування файлу. Програма виводить модальне попередження з вимогою підтвердити безповоротне видалення інформації. За умови згоди користувача метод clearCurrentProject у фоновому потоці Dispatchers.IO каскадно очищає базу даних від усіх зв'язаних робіт та кімнат, а також повністю скидає адресу та встановлені знижки.

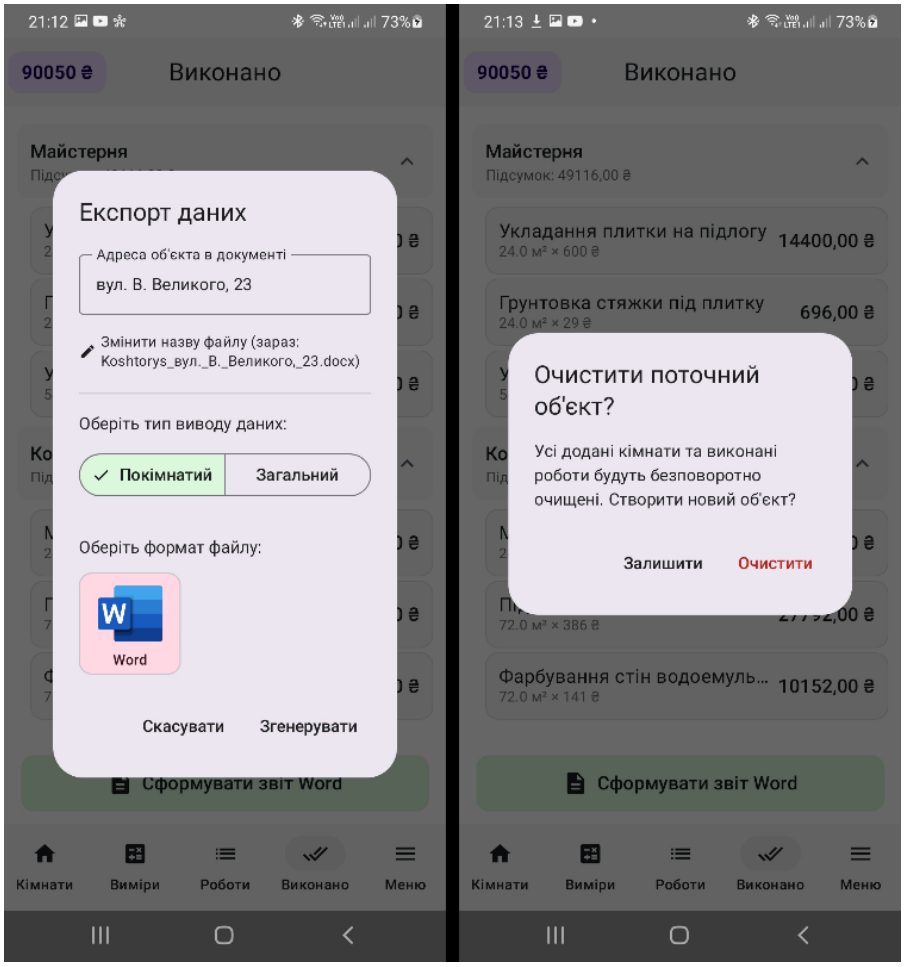


Рисунок 3.8 – Діалогові компоненти DoneScreen: налаштування параметрів експорту та вікно підтвердження очищення даних проєкту

Ці модальні компоненти забезпечують гнучке налаштування звітності та безпечне керування локальними даними, захищаючи майстра від випадкової втрати робочої інформації

3.6 Реалізація підсистеми фонової генерації звітів та взаємодії з файловою системою ОС Android

Процес формування фінальної звітності у вигляді файлів документів формату .docx є ресурсомісткою обчислювальною операцією, яка вимагає взаємодії з файловою системою, системними каналами сповіщень та динамічного парсингу об'єктних моделей [12]. Для забезпечення плавності графічного інтерфейсу користувача й запобігання блокуванню головного потоку застосунку, логіку генерації інтегровано в архітектурний компонент RoomViewModel за допомогою асинхронних корутин [2, 4].

Ініціалізація процедури створення документа відбувається через фоновий метод generateWordReportInBackground. Даний метод інкапсулює роботу з диспетчером обчислень Dispatchers.IO за допомогою конструкції withContext, що повністю ізолює важкі дискові операції введення-виведення [18]. Програмну реалізацію архітектурного мосту між реактивним станом інтерфейсу та менеджером експорту наведено в лістингу 3.2.

Лістинг 3.2 – Фонова ініціалізація процесу генерації документа

```
viewModelScope.launch {
   RenovumNotificationManager.showProgressNotification(appContext,
notificationId)
   val wordFile = withContext(Dispatchers.IO) {
      try {
         val reportData= ReportData(/*ініціалізація об'єкта даних*/)
         WordExportManager.createWordDocument(appContext,
reportData, isGroupedByRooms, customFileName, userSettings)
      } catch (e: Exception) { null }
   }
   if (wordFile != null && wordFile.exists()) {
      RenovumNotificationManager.showSuccessNotification(appContext,
wordFile, notificationId)
   } else {
RenovumNotificationManager.cancelExportNotification(appContext,
notificationId) }}
```

Безпосереднє низькорівневе конструювання структури файлу Word реалізовано всередині об'єкта WordExportManager із використанням

спеціалізованої бібліотеки Apache POI [13]. Метод createWordDocument ініціалізує новий екземпляр класу XWPFDocument, після чого жорстко конфігурує поля сторінки, адаптуючи документ під вимоги стандартів оформлення текстових звітів[29]. Фрагмент коду, що відповідає за програмне встановлення меж сторінки (ліве поле — 20 мм, інші — по 12 мм), представлено в лістингу 3.3.

Лістинг 3.3 – Програмне налаштування полів сторінки документа в Apache POI

```
val document = XWPFDocument()
val sectPr = document.document.body.let { if (it.isSetSectPr)
it.sectPr else it.addNewSectPr() }
val pageMar = if (sectPr.isSetPgMar) sectPr.pgMar else
sectPr.addNewPgMar()

pageMar.top = BigInteger.valueOf(850)    // 42.5pt
pageMar.bottom = BigInteger.valueOf(850) // 42.5pt
pageMar.right = BigInteger.valueOf(850)  // 42.5pt
pageMar.left = BigInteger.valueOf(1417)  // 70.85pt
```

Генерація таблиць технологічних операцій підтримує два поліморфні сценарії розгортання, залежно від прапорця isGroupedByRooms [20]. У разі покімнатної ієрархії дані ітеративно розділяються за об'єктами RoomEntity, де для кожного створюється окремий абзац із використанням системної табуляції зміщення вправо (STTabJc.RIGHT) для відображення проміжних сум [13]. Якщо вибрано консолідований звіт, усі роботи зіставляються в єдиний плоский масив за допомогою оператора flatten, групуються за ідентифікатором базової послуги WorkService, а їхні об'єми математично агрегуються за допомогою вбудованої функції sumOf [1].

Збереження готового документа здійснюється у внутрішню захищену директорію додатка (appContext.filesDir/Archive) з обов'язковим циклічним алгоритмом перевірки наявності файлу з аналогічним ім'ям для виключення ризику випадкового перезапису раніше згенерованих звітів.

Паралельно з роботою обчислювального конвеєра, користувач отримує асинхронний зворотний зв'язок через утиліту RenovumNotificationManager [4]. Після успішного запису файлу сповіщення трансформується: воно набуває статусу

відхильного та інтегрує захищений інтент PendingIntent для швидкого відкриття сформованого документа через зовнішні офісні додатки [6]. Логіку зв'язування файлу зі сповіщенням наведено в лістингу 3.4.

Лістинг 3.4 – Конфігурація PendingIntent для відкриття файлу зі сповіщення

```
val fileUri: Uri = FileProvider.getUriForFile(appContext,
"${appContext.packageName}.fileprovider", file)
val openFileIntent = Intent(Intent.ACTION_VIEW).apply {
   setDataAndType(fileUri, "application/vnd.openxmlformats-
officedocument.wordprocessingml.document")
   addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION or
Intent.FLAG_ACTIVITY_NEW_TASK)
}
val contentIntent = PendingIntent.getActivity(
   appContext, notificationId, openFileIntent,
   PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
)
```

Подальша взаємодія користувача з файлами кошторисів у локальному архіві спирається на компонент безпеки RenovumFileProvider [16]. Безпечне спільне використання файлів між ізольованою пісочницею додатка та зовнішніми системними утилітами реалізовано за допомогою генерації контентних посилань Uri через вбудований інструмент FileProvider.

З огляду на суворі обмеження безпеки сховища (Scoped Storage) [16], логіка збереження у публічну директорію розділена за версіями SDK. Для операційних систем Android 10 (API 29) і вище реалізовано взаємодію через системний контент-резолвер MediaStore, який виконує транзакційний запис даних у відносну публічну директорію Documents/Renovum [2]. Програмну реалізацію логіки експорту для сучасних версій ОС Android наведено в лістингу 3.5.

Лістинг 3.5 – Запис документа в публічне сховище через MediaStore (Android 10+)

```
val contentValues = ContentValues().apply {
   put(MediaStore.MediaColumns.DISPLAY_NAME, file.name)
   put(MediaStore.MediaColumns.MIME_TYPE,
"application/vnd.openxmlformats-
officedocument.wordprocessingml.document")
   put(MediaStore.MediaColumns.RELATIVE_PATH,
"${Environment.DIRECTORY_DOCUMENTS}/Renovum")
```

```
}
val resolver = context.contentResolver
val uri= resolver.insert(MediaStore.Files.getContentUri("external"),
contentValues)
uri?.let { contentUri ->
   resolver.openOutputStream(contentUri).use { out ->
file.inputStream().use { it.copyTo(out!!) } }
   true
} ?: false
```

Таким чином, розроблена архітектурна підсистема повністю покриває весь цикл роботи з файлами кошторисів — від безпечної асинхронної генерації на базі шаблонів бібліотеки Apache POI до гнучкого управління, перегляду та експорту документів без порушення вимог щодо продуктивності та безпеки даних [18, 32].

3.7 Програмна реалізація екрана локального архіву кошторисів та інструментів пакетного управління файлами

Для забезпечення автономності роботи майстра та можливості швидкого доступу до раніше сформованих документів без підключення до мережі Інтернет, у додатку реалізовано підсистему локального архіву ArchiveScreen.kt [5]. Робота з дисковим простором повністю інтегрована в архітектурний шаблон Model-View-ViewModel, де реактивні списки файлів синхронізуються із захищеною внутрішньою директорією пристрою [11].

Головний екран архіву відображає ієрархічний список карток створених документів за допомогою контейнера LazyColumn. Асинхронне зчитування вмісту теки Archive запускається в момент ініціалізації екрана через потік LaunchedEffect(Unit), який викликає метод loadArchiveFiles у RoomViewModel [4].

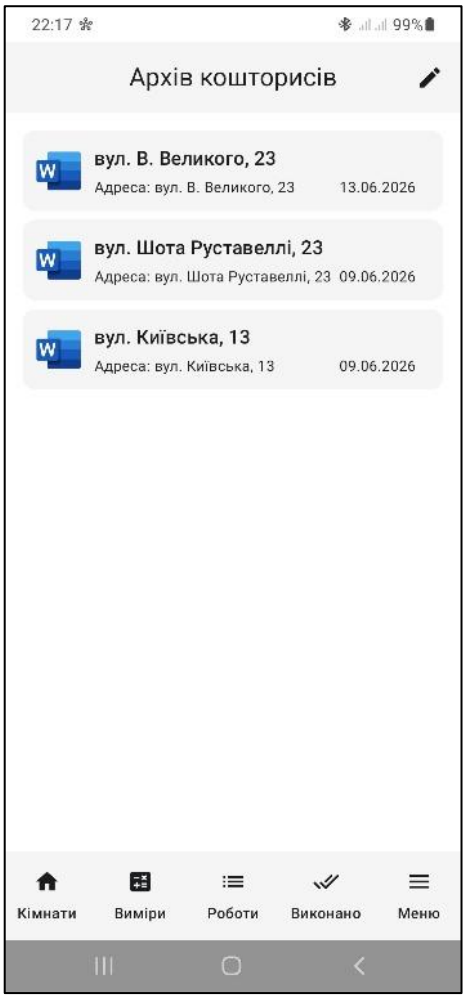


Рисунок 3.9 – Інтерфейс екрана локального архіву згенерованих кошторисів

Ключовою інженерною особливістю компонента FileCard є використання експериментального API combinedClickable, що дозволяє розділити сценарії взаємодії користувача з елементом списку:

- Одинарне натискання: якщо увімкнено стандартний режим перегляду, додаток ініціалізує стан clickedFileForActions і виводить на екран контекстне модальне вікно дій FileActionDialog.
- Довге натискання (утримання картки): автоматично активує режим множинного виділення (isArchiveSelectMode = true), додаючи поточний файл до динамічного списку selectedArchiveFiles у RoomViewModel

Контекстне діалогове вікно FileActionDialog надає майстру три вектори взаємодії з файлом, реалізовані через виклики методів утиліти RenovumFileProvider. Натискання на центральну кнопку «Відкрити» викликає

системний інтент ACTION_VIEW для рендерингу документа в сторонньому офісному пакеті. Інструменти панелі дозволяють виконати транзакційний запис у загальнодоступну директорію Documents/Renovum через іконку завантаження, активувати системний діалог спільного доступу ACTION_SEND (іконка «Поділитися») або ініціювати видалення конкретного документа через тригер singleFileToDelete. Логіку побудови інтерфейсу вибору дій над файлом представлено в лістингу 3.6.

Лістинг 3.6 – Обробка дій користувача в контекстному діалозі FileActionDialog

```
AlertDialog(onDismissRequest = onDismiss,
   title = { Text(text = cleanName, style = MaterialTheme.typography
.titleMedium)},
   text = { Row(modifier = Modifier.fillMaxWidth(),
horizontalArrangement = Arrangement.SpaceEvenly) {
         IconButton(onClick = { if (RenovumFileProvider
.saveToPublicDocuments(context, file)) {/*Toast успіху*/}}) {
            Icon(Icons.Default.Download, contentDescription =
"Зберегти")}
         IconButton(onClick = {
RenovumFileProvider.shareFile(context, file); onDismiss() }) {
            Icon(Icons.Default.Share, contentDescription =
"Поділитися")}
         IconButton(onClick = { onDismiss(); onDeleteClick() }) {
            Icon(Icons.Default.Delete, contentDescription =
"Видалити", tint = MaterialTheme.colorScheme.error) }}},
   confirmButton = { Button(onClick = { RenovumFileProvider
.openFile(context, file); onDismiss() }) { Text("Відкрити") } })
```

При переході сторінки в режим множинного вибору, звичайні кліки по картках починають працювати як інвертори стану виділення за допомогою методу toggleArchiveFileSelection. При цьому на екрані з'являється плаваюча кнопка дії FloatingActionButton (FAB) з іконкою кошика. Позиціонування цієї кнопки гнучко адаптується під фізіологічні особливості користувача: прапорець userSettings.isLeftHanded динамічно перемикає вирівнювання FAB між позиціями Alignment.BottomEnd (для праворуких) та Alignment.BottomStart (для ліворуких), що суттєво підвищує ергономіку інтерфейсу.

Натискання на плаваючу кнопку видалення активує модальне вікно попередження DeleteFilesDialog. Програма динамічно підраховує кількість

помічених об'єктів і виводить інформаційне повідомлення з вимогою підтвердження безповоротного очищення дискового простору.

Після підтвердження операції, метод deleteSelectedArchiveFiles ітеративно знищує вибрані файли з пам'яті пристрою у фоновому потоці, після чого викликає повторне сканування папки для синхронізації стану інтерфейсу, а блок DisposableEffect гарантує повне скидання масивів виділення при виході користувача з екрана архіву, запобігаючи витокам пам'яті [11].

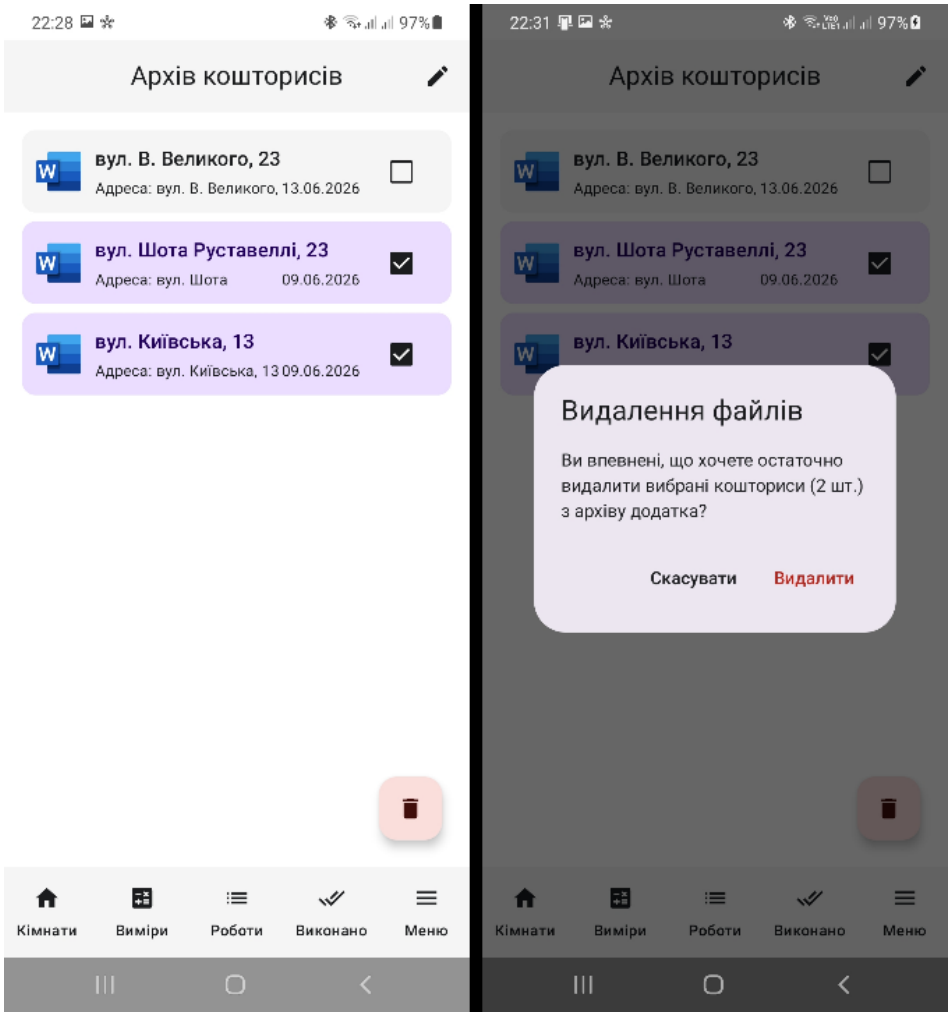


Рисунок 3.10 – Інструменти пакетного керування архівом: режим множинного виділення файлів та вікно підтвердження видалення

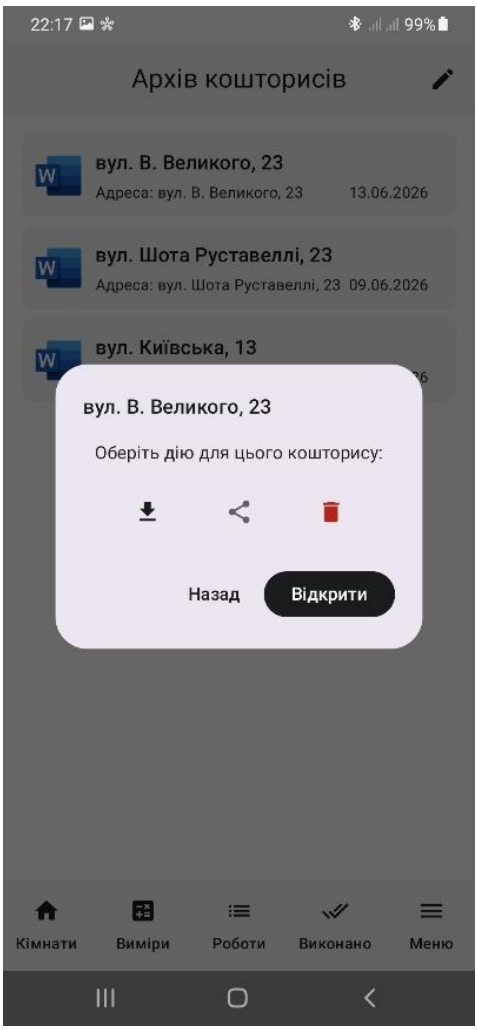


Рисунок 3.11 – Модальний діалог FileActionDialog для вибору операцій над окремим кошторисом

3.8 Верифікація функціональних вимог та управління версіями за допомогою системи Git

Для гарантування якості програмного продукту та відповідності розробленого мобільного застосунку початковим технічним завданням було проведено процедуру верифікації вимог. Верифікація здійснювалася методом наскрізного функціонального тестування основних користувацьких сценаріїв (циклу обчислень, динамічного формування кошторисів та роботи з архівом) безпосередньо на цільових пристроях під управлінням ОС Android. Результати тестування підтвердили, що всі базові функціональні вимоги, зафіксовані на етапі

проєктування системи, реалізовано в повному обсязі, а математичні алгоритми розрахунку складної геометрії приміщень працюють безвідмовно [38].

Організація процесу розробки вихідного коду застосунку та контроль версій здійснювалися на базі відкритого віддаленого репозиторію на платформі GitHub. Загальний вигляд структури репозиторію проєкту Renovum на веб-платформі GitHub представлено на рисунку 3.12.

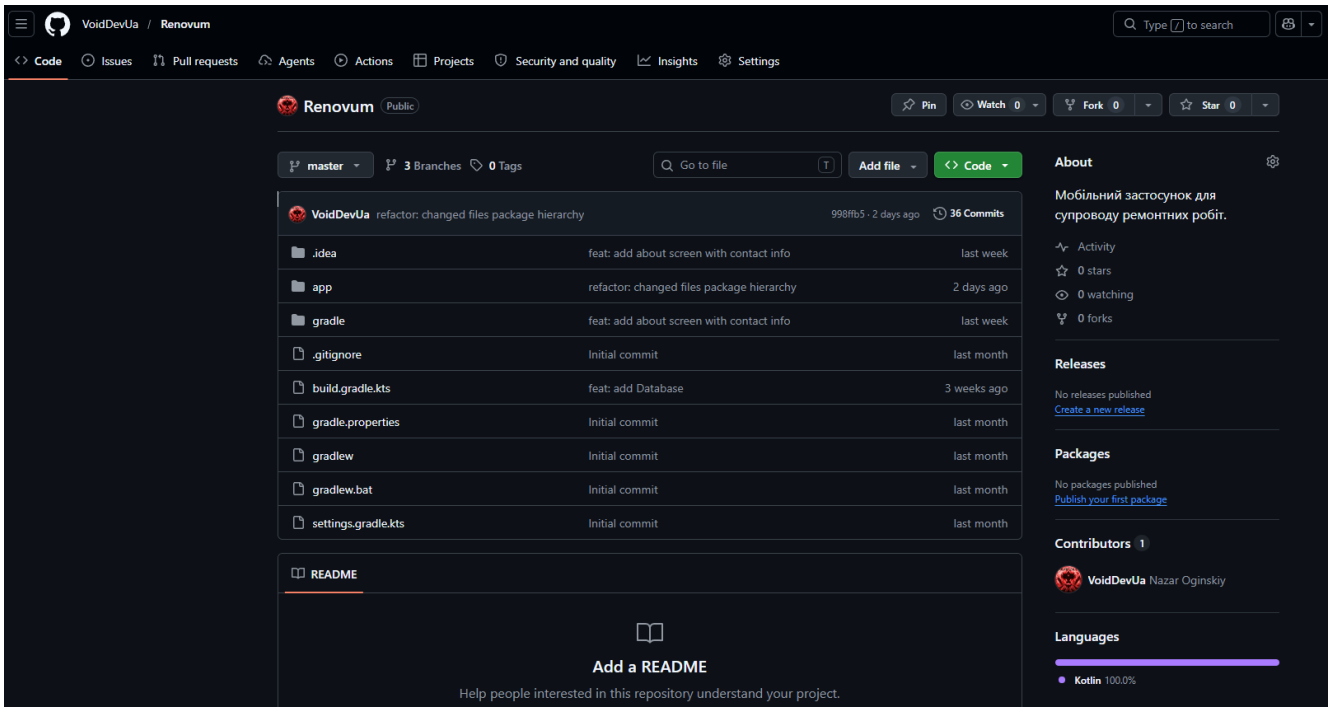


Рисунок 3.12 — Головна сторінка репозиторію мобільного застосунку "Renovum" на GitHub

Для структурування процесу створення архітектури програмного комплексу було застосовано гнучку модель розгалуження Feature Branch Workflow, яка базується на принципах ізольованої розробки функціоналу [36]. Робота з кодовою базою проєкту розподілена між наступними функціональними гілками:

- «master» — центральна стабільна гілка за замовчуванням (Default branch), яка містить виключно перевірені релізні версії коду застосунку, готові до фінального розгортання;
- «prototype» — активна інтеграційна гілка, що використовувалася як основне середовище для злиття проміжних модулів, покрокового

тестування взаємодії компонентів та стабілізації бізнес-логіки перед релізом;

- prototype-word-formation — спеціалізована ізольована гілка, виділена для проєктування, розробки та відлагодження алгоритмів підсистеми фонової генерації текстових звітів у форматі Microsoft Word за допомогою бібліотеки Apache POI.

Інтеграція накопиченого коду з робочих гілок до складу master відбувалася регламентовано — через механізм запитів на злиття (Pull Requests). Кожен Pull Request проходив процедуру верифікації та перевірки цілісності структури перед остаточним комбо-злиттям (Merge) [38].

Важливою інженерною особливістю ведення репозиторію є дотримання суворого стандарту ведення журналів комітів — Conventional Commits. Історія розробки чітко типізована за допомогою семантичних префіксів, що спрощує навігацію кодом:

- feat: — додавання нового функціоналу (наприклад, інтеграція діалогів очищення проєкту, додавання екранів WorksScreen чи створення бази даних);
- fix: — усунення виявлених дефектів та багів (виправлення правил обфускації ProGuard для Apache POI й кастомного логування, коригування відображення нульових значень на екрані підсумків);
- refactor: — реорганізація внутрішньої структури коду та пакетної ієрархії без зміни зовнішньої поведінки застосунку для покращення читабельності архітектури;
- perf: — оптимізація швидкодії окремих компонентів системи (зокрема, кешування статусів у ViewModel для прискорення рендерингу екрана вибору робіт).

3.9 Автоматизована збірка та розгортання релізного пакету застосунку

Процес формування фінального відмовостійкого програмного продукту та його підготовка до релізного розгортання (deployment) повністю автоматизовані засобами декларативної системи збірки Gradle. Постачання мобільного застосунку Renovum кінцевому користувачу в реальних виробничих або «польових» умовах реалізовано шляхом компіляції вихідного коду проєкту в автономний інсталяційний пакет формату .APK (Android Package).

Компіляція та пакування релізної версії виконується за допомогою запуску інструментарію Gradle через термінальну команду:

```
./gradlew assembleRelease
```

Перед безпосереднім формуванням підсумкового файлу розширення .apk, компілятор Gradle в автоматичному режимі ініціює роботу вбудованого системного інструменту оптимізації R8 / ProGuard. Застосування ProGuard є критично важливим для даного проєкту через інтеграцію важковагової зовнішньої бібліотеки Apache POI, яка суттєво збільшує об'єм вихідного байт-коду. Утиліта R8 виконує комплекс процесів:

- Стиснення коду (Shrinking): динамічний аналіз графу викликів та повне видалення з фінального пакету всіх невикористовуваних класів, полів, методів та атрибутів сторонніх бібліотек;
- Оптимізація ресурсів: сканування та вилучення делінійованих чи незадіяних у розмітці UI ресурсів додатка;
- Обфускація (Obfuscation): заміна довгих людиночитаваних назв класів, методів та змінних на короткі нечитавані індекси, що виконує роль захисту інтелектуальної власності від процесів зворотного інженерного проєктування (декомпіляції та зламу вихідного коду).

Завдяки автоматизованому налаштуванню правил оптимізації ProGuard для модулів Apache POI, вдалося досягти кращих показників швидкодії, мінімізувати

вагу кінцевого файлу розгортання та суттєво зменшити споживання оперативної пам'яті на мобільних пристроях. Готовий підписаний цифровим ключем пакет .apk передається замовнику для прямого локального встановлення на смартфони, що забезпечує незалежність від сторонніх маркетів дистрибуції (на кшталт Google Play) та гарантує повну конфіденційність розрахункових даних майстра.

3.10 Тестування застосунку та верифікація функціональних вимог

Фінальним етапом розробки мобільного застосунку «Renovum» є проведення комплексоного тестування для верифікації відповідності реалізованого функціоналу початковим технічним вимогам, забезпечення стабільності обчислювальних алгоритмів та перевірки коректності взаємодії програми з операційною системою Android [22].

З огляду на специфіку архітектури та стислі терміни розгортання проєкту, основним методом верифікації було обрано функціональне тестування за методологією «чорної скриньки» (Black-Box Testing) [24]. Цей підхід дозволяє оцінити працездатність інтерфейсу та бізнес-логіки застосунку з позиції кінцевого користувача, не заглиблюючись у внутрішню структуру вихідного коду.

Для систематизації процесу верифікації було розроблено та виконано серію тест-кейсів, що охоплюють критично важливі користувацькі сценарії: від первинного налаштування профілю до пакетного керування файлами в архіві. Результати успішного проходження тестування наведено в таблиці 3.2.

Таблиця 3.1 – Специфікація тест-кейсів застосунку «Renovum»

Назва тест-кейсу	**Опис кроків тестування**	**Очікуваний результат**	**Статус**
1	2	3	4
Валідація профілю майстра	Перехід у налаштування, введення імені та телефону, збереження стану.	Дані успішно перезаписуються в SharedPreferences та відображаються в шапці звітів.	PASSED
Динамічне створення об'єкта	Натискання кнопки додавання кімнати, введення назви та лінійних розмірів.	Об'єкт додається у БД Room, автоматично вираховується площа та периметр.	PASSED

Продовження таблиці 3.1

Призначення технологічних операцій	Вибір кімнати, перехід до каталогу послуг, введення об'єму для конкретної роботи.	Створюється сутність AppliedWork із прив'язкою до ID кімнати та поточної ціни.	PASSED
Калькуляція знижки проєкту	Перехід на DoneScreen, клік на індикатор вартості, введення значення 10%.	Інтерфейс реактивно оновлює фінальну суму з урахуванням математичного відсотка.	PASSED
Фоновий експорт у формат .docx	Натискання кнопки формування звіту, введення назви файлу, підтвердження.	Запускається корутина, на панелі з'являється прогрес-нотифікація, файл створюється в кеші.	PASSED
Інтеграція з ОС (FileProvider)	Клік на успішне сповіщення або вибір дії «Відкрити» в локальному архіві.	Система успішно генерує безпечний Uri та відкриває документ у сторонньому офісному пакеті.	PASSED
Пакетне очищення дискового простору	Активація режиму множинного вибору в архіві, помітка 2 файлів, клік на FAB кошика.	Файли ітеративно видаляються з пам'яті пристрою, список LazyColumn синхронно оновлюється.	PASSED

Для проведення повноцінного функціонального контролю та оцінки поведінки інтерфейсу Jetpack Compose в реальних умовах було сформовано тестове середовище, що складалося з декількох цільових пристроїв з різними апаратними характеристиками та версіями операційної системи Android [16]. Специфікацію обладнання та системного оточення, використаного під час випробувань, зведено в таблиці 3.2.

Таблиця 3.2 – Апаратне та програмне забезпечення тестового середовища

Тип пристрою / Емулятора	**Версія ОС Android (API Level)**	**Роздільна здатність екрану**	**ОЗП (RAM)**	**Тип перевірки**
Фізичний смартфон	Android 13.0 (API 33)	2400 × 1080 (FHD+)	6 Гб	Основне юзабіліті-тестування, перевірка збереження файлів
Фізичний смартфон	Android 11.0 (API 30)	1600 × 720 (HD+)	4 Гб	Перевірка адаптивності інтерфейсу, оцінка швидкодії
Android Emulator (AVD)	Android 10.0 (API 29)	1920 × 1080 (FHD)	3 Гб	Тестування сумісності зі Scoped Storage через MediaStore

Під час виконання наведеної програми випробувань окрема увага приділялася аналізу граничних значень (Boundary Values) та стійкості системи до некоректних дій користувача [24]. Зокрема, було проведено перевірки на введення екстремально великих лінійних розмірів приміщень (наприклад, довжина стіни понад 100 метрів) та спроби генерації звітів із порожніми текстовими полями адреси об'єкта.

Завдяки інтегрованій у бізнес-логіку RoomViewModel попередній фільтрації та автоматичній заміні порожніх рядків на дефолтні значення (наприклад, автоматичне підтягування назви об'єкта за замовчуванням у WordExportManager), додаток продемонстрував високу толерантність до помилок. Графічні компоненти Jetpack Compose коректно обробляли довгі текстові назви робіт у картках FileCard за рахунок використання властивості TextOverflow.Ellipsis, яка запобігала руйнуванню та виходу елементів за межі екранної сітки [5].

Аналіз результатів інспекційного контролю показав, що мобільний застосунок «Renovum» повністю відповідає критеріям стабільності: фонові потоки корутин Dispatchers.IO забезпечують відсутність мікрофризів інтерфейсу під час генерації .docx документів, а механізми безпеки FileProvider гарантують безпомилкову взаємодію із зовнішнім системним середовищем Android [4, 18]. Критичних дефектів, витоків пам'яті (Memory Leaks) або непередбачуваних аварійних завершень роботи програми (Crashes) під час фінальної серії тестів зафіксовано не було.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У розділі розглянуто інформаційне перевантаження як фактор ризику для життєдіяльності людини, а також висвітлено ергономічні вимоги до організації робочого місця розробника програмного забезпечення, оптимальні параметри виробничого мікроклімату та штучного освітлення відповідно до чинних нормативних санітарних стандартів.

4.1 Інформаційне перевантаження як фактор ризику для життєдіяльності людини

Інформаційне перевантаження виникає тоді, коли обсяг та швидкість надходження інформації перевищують природні психофізіологічні можливості людини щодо її якісного опрацювання, аналізу та використання для прийняття рішень [34]. У сучасному цифровому середовищі цей фактор проявляється через постійний потік повідомлень, електронних листів, технічної документації та робочих завдань.

Під час розумової праці ефект інформаційного перевантаження є особливо відчутним. Програміст у процесі розробки мобільного застосунку змушений одночасно оперувати багатьма різнорідними потоками даних. Він аналізує вихідний код, відстежує стан змінних у середовищі розробки (IDE), реагує на помилки компіляції, звіряється з технічним завданням, вивчає документацію та інтерпретує результати тестування. Така багатозадачність створює підвищене розумове й зорове навантаження на організм оператора [34, 35]. Якщо ці інформаційні потоки не впорядковані, стрімко зростає ризик хронічної втоми, розсіяності уваги, прийняття хибних рішень та появи помилок у коді.

Тривале інформаційне перевантаження негативно впливає не лише на продуктивність, а й створює загрози для безпеки життєдіяльності. Людина у стані постійного розумового напруження повільніше реагує на небезпечні ситуації, гірше оцінює ризики, частіше допускає неуважність і може ігнорувати ознаки

небезпеки [34]. У робочих або побутових умовах це може призвести до неправильного використання обладнання, порушення правил безпеки або травмування.

Основними причинами інформаційного перевантаження є надмірна кількість завдань, вимушена багатозадачність, постійні сповіщення, відсутність чіткого плану роботи, тривала робота без перерв, дефіцит сну і невпорядкованість джерел інформації. Основні прояви та наслідки цього впливу наведено у таблиці 4.1.

Таблиця 4.1 – Основні прояви інформаційного перевантаження та їх наслідки

Прояв	**Можливі наслідки**
Зниження концентрації	Помилки в роботі, повільніше виконання завдань, розсіяність
Втома	Загальне стійке зниження інтелектуальної працездатності
Дратівливість	Погіршення емоційного стану, виникнення конфліктів
Погіршення пам’яті	Складність запам’ятовування дрібних технічних деталей
Порушення сну	Хронічна втома, відсутність відновлення, зниження пильності

Для зменшення ризиків інформаційного перевантаження потрібно раціонально організовувати робочий процес. Великі інженерні завдання доцільно ділити на менші етапи, визначати пріоритети, вести список задач і фокусуватися на одному основному завданні у конкретний проміжок часу. Під час виконання складної роботи бажано вимикати другорядні сповіщення, закривати зайві вкладки браузера та залишати відкритими лише потрібні для розробки інструменти.

Важливе значення має режим праці та відпочинку. Після кожної години безперервної роботи за комп’ютером необхідно робити короткі перерви тривалістю 10–15 хвилин [35]. Під час цих пауз слід змінювати положення тіла (встати, пройтися), давати відпочинок очам і виконувати легку розминку м'язів шиї та спини. Це допомагає зменшити втому, підтримати концентрацію та знизити ризик помилок, зумовлених людським фактором [34, 35].

4.2 **Ергономічні вимоги та оптимізація мікроклімату на робочому місці розробника**

Організація робочого місця інженера-програміста у процесі проектування мобільного застосунку має безпосередній вплив на його працездатність та збереження здоров'я. Професійна діяльність розробника характеризується тривалим перебуванням у статичній сидячій позі та високим зоровим і психоемоційним навантаженням. Відповідно до державних санітарних правил і норм ДСанПіН 3.3.2.007-98, параметри робочої зони та мікроклімату підлягають суворому регулюванню для створення безпечних умов праці [31, 35].

Ергономіка робочого місця розробника базується на правильному підборі меблів і комп'ютерного обладнання. Робочий стіл повинен мати висоту в межах 680–800 мм, а його поверхня — забезпечувати достатній простір для зручного розміщення монітора, клавіатури, миші та ліктів розробника. Конструкція робочого стільця (крісла) обов'язково має бути підйомно-поворотною, з можливістю регулювання висоти сидіння та кута нахилу спинки. Це дозволяє підтримувати фізіологічно правильну поставу (кут між стегном і тулубом, а також у колінних суглобах повинен становити приблизно 90°), що мінімізує навантаження на хребет і запобігає розвитку захворювань опорно-рухового апарату [32, 35].

Критично важливим фактором охорони праці є параметри монітора. Екран повинен розташовуватися на відстані 600–700 мм від очей програміста, а його верхня кромка має знаходитися на рівні очей або трохи нижче. Таке розміщення зменшує втому очних м'язів і запобігає виникненню комп'ютерного зорового синдрому. Клавіатуру та мишу слід розміщувати на відстані 100–300 мм від краю столу для забезпечення опори передпліччям та профілактики кистьового тунельного синдрому [31, 32].

Освітлення робочої зони повинно бути комбінованим (природним та штучним). Робочі місця з комп'ютерною технікою слід розташовувати так, щоб природне світло з вікон падало збоку (переважно ліворуч), задля уникнення появи відблисків на екрані. Штучне освітлення має виконуватися у вигляді загального

рівномірного освітлення системи із рівнем освітленості на поверхні столу не менше 300–500 лк [35].

Параметри мікроклімату в приміщенні, де виконується розробка програмного забезпечення, повинні відповідвати оптимальним значенням для категорії робіт легкої фізичної напруженості (категорія Іа) [31, 35]. Основні нормативні показники мікроклімату наведено в таблиці 4.2.

Таблиця 4.2 – Оптимальні параметри мікроклімату для робочих місць операторів ЕОМ

Параметр мікроклімату	**Оптимальне значення в теплий період року**	**Оптимальне значення в холодний період року**
Температура повітря	22 – 25 °С	20 – 22 °С
Відносна вологість	40 – 60 %	40 – 60 %
Швидкість руху повітря	не більше 0,1 м/с	не більше 0,1 м/с

Для підтримки заданих параметрів повітряного середовища приміщення повинно бути обладнане системами опалення, вентиляції або кондиціонування. Протягом робочої зміни обов’язково проводиться регулярне провітрювання кімнати для зниження концентрації вуглекислого газу та очищення повітря від пилу, який інтенсивно притягується статичним полем моніторів [31].

Згідно з вимогами Закону України «Про охорону праці», тривалість безперервної роботи за комп'ютером не повинна перевищувати 2 годин [26, 35]. Регламентований режим праці передбачає організацію коротких перерв тривалістю 10–15 хвилин через кожну годину інтенсивної роботи. Під час перерв розробнику рекомендується виконувати комплекси вправ для очей (фокусування на відстані, колові рухи) та легку фізичну розминку для відновлення кровообігу у нижніх кінцівках і м'язах спини [31, 35].

ВИСНОВКИ

У кваліфікаційній роботі виконано розроблення мобільного застосунку «Renovum» для комп’ютерного розрахунково-будівельного супроводу ремонтних робіт з використанням мови програмування Kotlin. Розроблений застосунок призначений для автоматизації обчислення параметрів приміщень, пооб’єктного обліку виконаних процесів, гнучкого керування ціноутворенням і кошторисами, а також для автоматичного генерування звітної документації.

У першому розділі проаналізовано предметну область ремонтно-будівельного обліку. Визначено недоліки традиційних підходів: ручні заміри, ризик помилок через людський фактор та незручність паперових чи Excel-нотаток на об'єктах. Аналіз аналогів (SimplyWise, Rabotniki.ua) показав їхню фрагментарність та залежність від інтернету. Обґрунтовано розробку автономного Android-застосунку «Renovum» та сформовано вимоги до його інтерфейсу й точності обчислень.

У другому розділі спроєктовано автономну архітектуру застосунку на основі патерну MVVM та Repository Pattern. Навігацію з типізованими маршрутами (sealed-класи) реалізовано через Navigation Compose з адаптацією інтерфейсу під ліву або праву руку майстра. Для локального збереження даних використано СУБД Room (SQLite) із JSON-серіалізацією через Gson та каскадним видаленням залежних сутностей, а статичний каталог послуг винесено в файли assets. Описано сценарії обчислення геометрії кімнат (прямокутних, Г- та Т-подібних форм) із відніманням прорізів вікон/дверей, алгоритми ціноутворення та автоматичного експорту кошторисів у .docx за допомогою Apache POI. Сформовано суворі бізнес-правила валідації для контролю фізичної цілісності даних.

У третьому розділі виконано програмну реалізацію та тестування застосунку «Renovum». Проєкт структуровано за пакетним принципом розподілу логічних шарів чистої архітектури. За допомогою Jetpack Compose розроблено реактивний інтерфейс для ініціалізації об'єктів, геометричних замірів та формування кошторисів. Математичну обробку прямокутних, Г- та Т-подібних приміщень із

відніманням прорізів інтегровано в поліморфних моделях (sealed class) шару бізнес-логіки через потоки StateFlow. На базі корутин та бібліотеки Apache POI реалізовано асинхронну генерацію .docx звітів із дотриманням правил Scoped Storage та обміну через FileProvider. Спроєктовано локальний архів із підтримкою мультивиділення та адаптацією UI під робочу руку користувача. Функціональне тестування «чорної скриньки» на фізичних пристроях та емуляторах підтвердило стабільність обчислень, толерантність до помилок введення та відсутність критичних дефектів.

У четвертому розділі розглянуто питання безпеки життєдіяльності та охорони праці. Описано ергономічні вимоги до організації робочого місця розробника та профілактики психофізіологічного перевантаження під час роботи з комп’ютером.

У перспективі застосунок «Renovum» може бути розширений для реального ринкового впровадження. Основними напрямами розвитку є виправлення дрібних недоліків UI, додавання двох нових екранів та впровадження щотижневого оновлення актуальних цін на ремонтні роботи. Такі покращення не змінюють архітектурну логіку системи, а доповнюють її інструментами для масштабного використання майстрами.

Перед публікацією в Google Play Market доцільно забезпечити всебічний супровід розробленої системи та її підготовку до промислової експлуатації. Це передбачає тестування під навантаженням, моніторинг помилок, резервне копіювання даних та фіксацію політики конфіденційності. Це дозволить перевести «Renovum» до рівня надійного сервісу, готового для реальних об'єктів.

Результатом кваліфікаційної роботи є функціональний Android-застосунок «Renovum», який демонструє практичне застосування сучасних технологій мобільної розробки, локальних баз даних, чистих архітектурних патернів та інженерно-калькуляційної логіки. Поставлену мету повністю досягнуто, а основні завдання кваліфікаційної роботи виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kotlin Documentation [Електронний ресурс]. Режим доступу: https://kotlinlang.org/docs/home.html

2. Android Developers. Kotlin and Android [Електронний ресурс]. Режим доступу: https://developer.android.com/kotlin

3. Android Developers. Jetpack Compose documentation [Електронний ресурс]. Режим доступу: https://developer.android.com/jetpack/compose/documentation

4. Android Developers. Guide to app architecture [Електронний ресурс]. Режим доступу: https://developer.android.com/topic/architecture

5. Android Developers. Navigation with Compose [Електронний ресурс]. Режим доступу: https://developer.android.com/develop/ui/compose/navigation

6. Android Developers. ActivityResultContracts [Електронний ресурс]. Режим доступу: https://developer.android.com/reference/androidx/activity/result/contract/ActivityResultContracts

7. Material Design 3. Design system documentation [Електронний ресурс]. Режим доступу: https://m3.material.io/

8. Android Developers. Room persistence library [Електронний ресурс]. Режим доступу: https://developer.android.com/training/data-storage/room

9. Android Developers. Defining data using Room entities [Електронний ресурс]. Режим доступу: https://developer.android.com/training/data-storage/room/defining-data

10. Android Developers. Accessing data using Room DAOs [Електронний ресурс]. Режим доступу: https://developer.android.com/training/data-storage/room/accessing-data

11. Android Developers. Migrating Room databases [Електронний ресурс]. Режим доступу: https://developer.android.com/training/data-storage/room/migrating-db-versions

12. Android Developers. Referencing complex data using Room [Електронний ресурс]. Режим доступу: https://developer.android.com/training/data-storage/room/referencing-data

13. Overview (POI API Documentation) [Електронний ресурс]. Режим доступу: https://poi.apache.org/apidocs/dev/index.html

14. Gradle User Manual [Електронний ресурс]. Режим доступу: https://docs.gradle.org/current/userguide/userguide.html

15. Android Studio User Guide [Електронний ресурс]. Режим доступу: https://developer.android.com/studio/intro

16. OWASP Mobile Application Security Verification Standard [Електронний ресурс]. Режим доступу: https://mas.owasp.org/MASVS/

17. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 411 p.

18. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.

19. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 422 p.

20. Freeman E., Robson E. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2020. 672 p.

21. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

22. Burns B. Designing Distributed Systems. Sebastopol : O'Reilly Media, 2018. 166 p.

23. Bloch J. Effective Java. 3rd ed. Boston : Addison-Wesley, 2018. 416 p.

24. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.

25. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI [Електронний ресурс]. Режим доступу: https://zakon.rada.gov.ua/laws/show/2297-17

26. Про охорону праці : Закон України від 14.10.1992 № 2694-XII [Електронний ресурс]. Режим доступу: https://zakon.rada.gov.ua/laws/show/2694-12

27. Кодекс цивільного захисту України : Закон України від 02.10.2012 № 5403-VI [Електронний ресурс]. Режим доступу: https://zakon.rada.gov.ua/laws/show/5403-17

28. Правила пожежної безпеки в Україні : наказ Міністерства внутрішніх справ України від 30.12.2014 № 1417 [Електронний ресурс]. Режим доступу: https://zakon.rada.gov.ua/laws/show/z0252-15

29. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. Київ : ДП «УкрНДНЦ», 2016. 31 с.

30. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. Київ : ДП «УкрНДНЦ», 2016. 20 с.

31. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами ЕОМ. Київ, 1998.

32. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д. М., Цуприк Г. Б., Бревус В. М. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2024. 45 с.

33. Репозиторій програмного коду мобільного застосунку Renovum [Електронний ресурс]. Режим доступу: https://github.com/VoidDevUa/Renovum

34. Левченко О. Г., Землянська О. В., Праховнік Н. А., Зацарний В. В. Безпека життєдіяльності та цивільний захист : підручник. 2-ге вид. Київ : Каравела, 2021. 268 с.

35. Левченко О. Г., Полукаров О. І., Арламов О. Ю., Полукаров Ю. О., Землянська О. В. Охорона праці та цивільний захист : підручник для студентів бакалаврату. Київ : Каравела, 2021. 352 с.

36. Chacon S., Straub B. Pro Git. 2nd ed. Berkeley : Apress, 2014. 456 p.

37. Conventional Commits 1.0.0 [Електронний ресурс]. Режим доступу: https://www.conventionalcommits.org/

38. Петрик М. Р., Мудрик І. Я. Проєктування програмного забезпечення на основі об’єктно-орієнтованого аналізу вимог та інструментальних засобів розробки IBM Rational Software Architect. Тернопіль : ТНТУ ім. І. Пулюя, 2022. 144 с.

39. Глива В., Мудрик І. Я. Ентерпрайз патерни для кросплатформної розробки // Матеріали XII науково-технічної конференції „Інформаційні моделі, системи та технології“. Тернопіль : ТНТУ ім. І. Пулюя, 2024. С. 45–46.

40. Яковенко І., Мудрик І. Орієнтований на дані дизайн. Аналіз та переваги // Матеріали IX науково-технічної конференції „Інформаційні моделі, системи та технології“. Тернопіль : ТНТУ ім. І. Пулюя, 2022. С. 28.

ДОДАТКИ

Додаток А

Лістинг коду із кваліфікаційною роботою бакалавра

Лістинг коду розміщений у віддаленому репозиторії GitHub за посиланням: https://github.com/VoidDevUa/Renovum.

Та за QR кодом:

Додаток Б

Апробація результатів

Забезпечення автономності та цілісності даних у мобільних системах управління ремонтними роботами.

УДК 621.395.7 (043.2) Назар Огінський[1]

Тернопільский національний технічний університет імені Івана Пулюя,

[1]nazar_ohinskyi0706@tntu.edu.ua

Сучасна практика менеджменту в будівництві потребує використання спеціалізованих цифрових інструментів, що гарантують конфіденційність розрахунків та стійкість комерційної інформації до зовнішніх загроз безпеці даних. Проблема ефективного супроводу ремонтних робіт полягає у відсутності зручних інструментів для точних обчислень, а також у ризиках використання хмарних сервісів, що можуть призвести до витоку персональної фінансової інформації [1]. Більшість існуючих рішень вимагають постійної синхронізації, що робить дані вразливими до перехоплення у відкритих мережах.

Метою дослідження є проєктування та розробка автономного мобільного застосунку для автоматизації обчислення площ приміщень та формування кошторисів. Актуальність роботи зумовлена необхідністю мінімізації помилок при ручних розрахунках та потребою у конфіденційному інструменті, який забезпечує повний контроль над даними без залучення сторонніх серверів.

Наукова новизна полягає у розробці моделі автономного функціонування прикладного програмного забезпечення, яка базується на принципах ізоляції фінансових обчислень у локальному середовищі мобільної ОС для мінімізації ризиків перехоплення та витоку комерційної інформації. На відміну від аналогів, запропоновано модель локального збереження даних, яка виключає ризики несанкціоного доступу до інформації у хмарних сховищах.

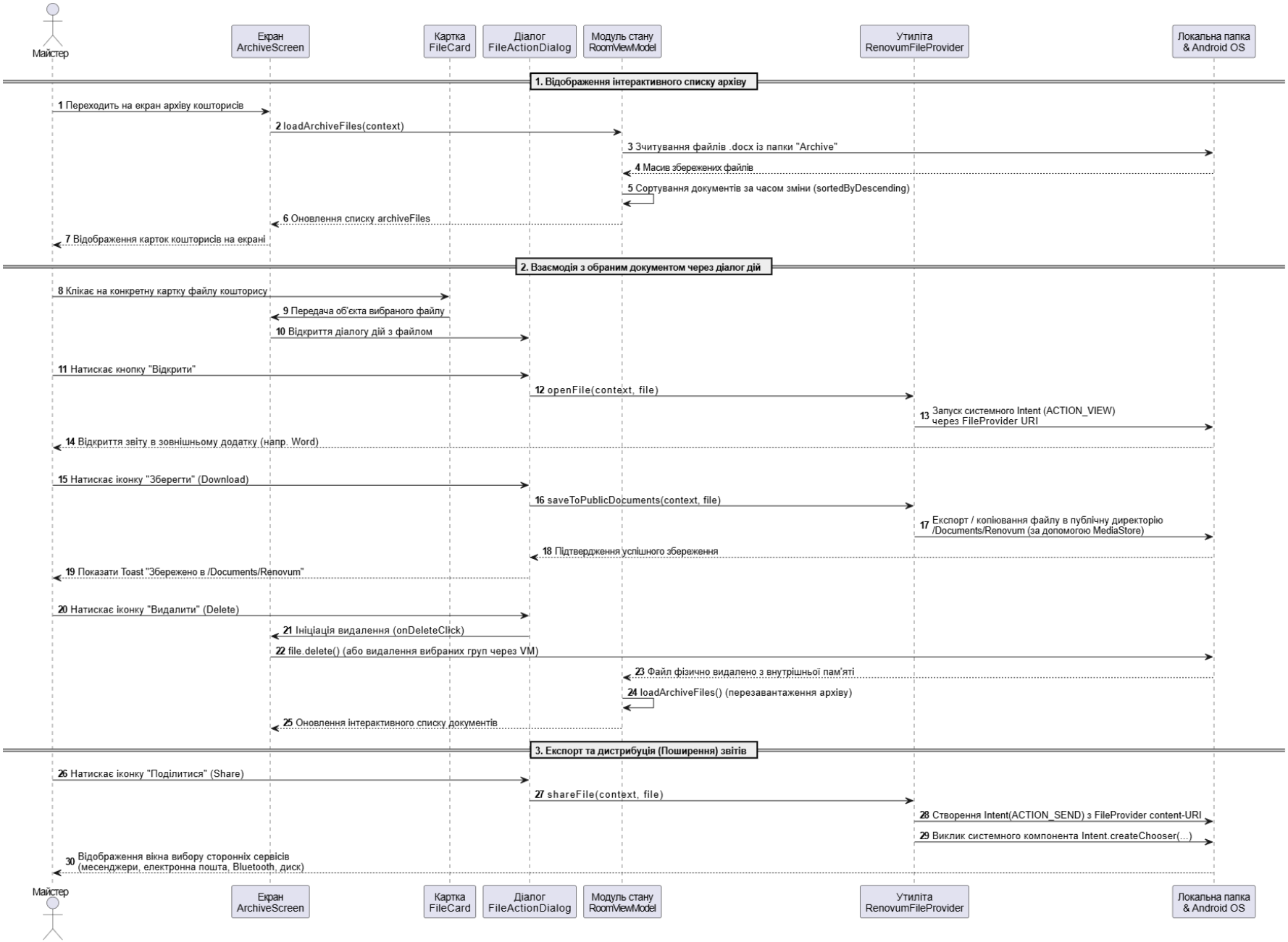
Для реалізації системи обрано мову Kotlin та фреймворк Jetpack Compose [2]. Програмна архітектура побудована на патерні MVVM, що дозволило ізолювати логіку обчислень від інтерфейсу. Розрахунковий модуль автоматизує визначення необхідних для роботи параметрів об'єкту. Локальне збереження даних реалізовано за допомогою бібліотеки Room на базі SQLite, що гарантує автономність роботи та швидкість доступу до інформації. Цей підхід забезпечує цілісність даних та захист від мережевих загроз, оскільки всі дані обробляються виключно на пристрої [3]. Розроблено каталог робіт, який дозволяє гнучко налаштовувати фінансові параметри проекту в ізольованому середовищі.

Розроблений мобільний застосунок забезпечує автоматизацію основних етапів супроводу ремонтних робіт, поєднуючи зручність інтерфейсу з принципами безпечного збереження даних. Отримані результати підтверджують ефективність локальних баз даних для вирішення прикладних задач будівельної інженерії, гарантуючи приватність фінансових розрахунків.

1. OWASP Mobile Application Security (MAS). URL: https://mas.owasp.org/ (дата звернення: 14.05.2026).
2. Bloch J. Effective Java. 3rd ed. Boston: Addison-Wesley Professional, 2017. 412 p.
3. Griffiths D., Griffiths D. Head First Android Development: A Learner's Guide to Building Android Apps with Kotlin. 3rd ed. Sebastopol: O'Reilly Media, 2021. 930 p.

Додаток В

Діаграма послідовності сценарію основного робочого циклу



Додаток Д

Діаграма послідовності процесу експорту та поширення звіту з архіву