НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Комп'ютерних наук
(назва кафедри)

_____ Голуб Б.Л.
(підпис) (ПІБ)

"2" червня 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Мобільний додаток системи тестування безпілотників»

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
доцент к.т.н. _____ Вайганг Г.О.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
доцент к.т.н. _____ Пархоменко І.І.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Губенок В.Г.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____комп'ютерних наук _____
__доцент к.т.н._____ _Голуб Б. Л. __
(науковий ступінь, вчене звання) (підпис) (ПІБ)
“____” _____20_____р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студентці

_____Губенка Владислава Геннадійовича_____
(прізвище, ім’я, по батькові)

Спеціальність 121 – «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення мобільного органайзера

Затверджена наказом ректора НУБіП України від “_16_” _Грудня_2024 р. №2249 “С”

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи

Дослідження щодо впливу програмного забезпечення на процеси тестування безпілотників. Аналіз ефективності мобільного застосунку для організації сесій, збору телеметрії, управління сценаріями та формування звітності..

Перелік питань, які потрібно розробити:

1) Аналіз систем тестування БПЛА як об’єкта дослідження
2) Проєктування інформаційного та програмного забезпечення
3) Розробка інформаційного та програмного забезпечення
4) Тестування та введення в експлуатацію системи

Дата видачі завдання “_____”_____20___ р.

Керівник бакалаврської кваліфікаційної роботи _____ ___Пархоменко І.І.
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ _____Губенок В.Г.
(підпис) (прізвище та ініціали студента)

Календарний план

№ з/п	Назва етапів виконання бакалаврської кваліфікаційної роботи	Строк виконання етапів бакалаврської кваліфікаційної роботи	Примітка
	Дослідження систем тестування БПЛА	01.05.2025-03.05.2025	
	Аналіз вимог до системи	04.05.2025	
	Моделювання предметної області	05.05.2025-07.05.2025	
	Огляд існуючих рішень	08.05.2025	
	Постановка задачі	09.05.2025	
	Проєктування бази даних	10.05.2025-12.05.2025	
	Проектування програмного забезпечення	12.05.2025-15.05.2025	
	Розробка бази даних	15.05.2025-21.05.2025	
	Розробка алгоритмів програмного забезпечення	22.05.2025-29.05.2025	
	Тестування системи	25.05.2025-30-05.2025	
	Проходження нормо контролю	01.06.2025	
	Попередній захист	02.06.2025	
	Захист	12.06.2025	

АНОТАЦІЯ

У бакалаврській кваліфікаційній роботі на тему “Програмне забезпечення системи тестування безпілотників” досліджено особливості створення цифрового рішення для організації та автоматизації тестування БПЛА. Розглянуто проблематику управління тестовими сесіями, збирання телеметрії, збереження сценаріїв тестування та генерації звітів.

Програмне забезпечення було реалізовано мовою програмування **Kotlin** з використанням фреймворків для створення мобільних додатків (Android SDK). Як систему управління базами даних застосовано **SQLite**.

Робота складається з чотирьох розділів. Перший розділ присвячено аналізу предметної області та постановці задачі. У другому розділі описано проєктування інформаційного та програмного забезпечення: побудовано ER-діаграми, діаграми класів і компонентів. Третій розділ містить опис реалізації програмного забезпечення. У четвертому розділі наведено результати тестування системи та рекомендації щодо її впровадження.

ABSTRACT

In the bachelor's qualification work on the topic “Software for UAV Testing System,” the development of a digital solution for organizing and automating UAV testing was explored. The work examines the challenges of managing test sessions, collecting telemetry data, storing test scenarios, and generating reports.

The software was developed using the **Kotlin** programming language with the **Android SDK** framework for mobile application development. **SQLite** was used as the database management system.

This work consists of four sections. The first section covers the analysis of the subject domain and the problem statement. The second section focuses on designing the information and software components, including ER diagrams, class diagrams, and component diagrams. The third section describes the implementation of the software. The fourth section presents system testing results and provides recommendations for deployment.

ЗМІСТ

ВСТУП..... 8
1 СИСТЕМНИЙ АНАЛІЗ СИМУЛЯЦІЇ РОБОТИ ДРОНІВ ЯК ОБ'ЄКТУ ДОСЛІДЖЕННЯ..... 10
1.1 Опис сфери тестування БПЛА..... 10
1.2 Аналіз вимог до програмної системи..... 11
1.3 Моделювання предметної області..... 13
1.4 Огляд інформаційних джерел та існуючих рішень..... 15
1.5 Постановка завдання..... 19
2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 23
2.1 Логічна модель даних у вигляді ER-діаграми..... 23
2.2 Діаграма класів..... 27
2.3 Діаграма пакетів..... 30
2.4 Діаграма компонентів..... 31
3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ..... 33
3.1 Система управління базою даних..... 33
3.2 Розробка об'єктів бази даних..... 35

3.2.1 Створення бази даних.....35
3.2.2 Створення таблиць.....36
3.2.3 Створення процедур.....36
3.2.4 Створення тригерів.....37
3.2.5 Створення користувачів.....38
3.3 Вибір інструментарію для створення мобільного застосунку тестування.....39
3.4 Алгоритмізація та програмування програмних модулів.....40
4 Рекомендації щодо впровадження та експлуатації системи.....44
4.1 Тестування системи.....44
4.2 Вимоги до апаратного та програмного забпезпечення.....59
4.3 Склад інсталяційного пакету.....60
ВИСНОВки.....61
Список використаних джерел.....62

ВСТУП

Актуальність даної розробки полягає в тому, що безпілотні літальні апарати (БПЛА) активно використовуються в різних галузях, і з цим зростає потреба у відлагоджених системах тестування їхніх параметрів та поведінки. Проте не завжди є можливість проводити тестування з використанням реального обладнання — через вартість дронів, обмеження простору, погодні умови чи інші чинники. У зв'язку з цим особливої актуальності набуває створення мобільного додатку, який дозволяє симулювати процеси тестування БПЛА без необхідності фізичного підключення до пристрою[1]. Такий підхід дозволяє перевірити сценарії поведінки, обробку даних, логіку інтерфейсу та роботу з умовними параметрами.

Метою проєкту є розробка мобільного застосунку на базі Android Studio, який імітує процес тестування дрона. У додатку передбачено можливість створення віртуальних польотних сесій, генерації симульованих даних (наприклад, швидкості, висоти, температури, заряду акумулятора), побудови графіків на основі цих даних, збереження результатів у локальну базу, а також формування текстових звітів. Такий підхід дозволить користувачам протестувати логіку аналізу, візуалізації та обліку даних без потреби в реальному апараті.

Реалізація проєкту здійснюється з використанням Android Studio[2], мовою Kotlin. Для зберігання даних планується використання SQLlite[3] , а для візуалізації — бібліотека MPAndroidChart[4]. Генерація симульованих даних реалізується через вбудовані алгоритми, які створюють змінні параметри польоту залежно від заданих сценаріїв. Користувач зможе налаштовувати тривалість симуляції, частоту зміни параметрів, а також обирати тип моделі дрона, яка буде тестуватись.

Таким чином, мобільний додаток стане зручним інструментом для навчання, тестування логіки обробки даних, а також демонстрації роботи програмного забезпечення у сфері безпілотної авіації. Навіть без реального дрона, користувач

зможе проводити віртуальні тести та аналізувати поведінку системи в різних умовах, що значно знижує витрати на розробку та дозволяє виявляти помилки на ранніх етапах.

Розробка програмного забезпечення включала етапи моделювання, проєктування та реалізації. Структуру системи описано за допомогою UML-діаграм (прецедентів, активності, послідовності, класів, компонентів, пакетів та розгортання). Такий підхід дозволяє краще зрозуміти архітектуру ПЗ, забезпечити її розширюваність та підтримку.

Структуру даної пояснювальної записки до дипломного проекту можна прелствити в такій послідовності:

1) перший розділ, де розглядається предметна область та формуються вимоги до програмного забезпечення;
2) другий розділ, присвячений проектуванню програмного забезпечення;
3) третій розділ, що обґрунтовує вибір інструментарію та описує розробку програмного забезпечення;
4) четвертий розділ, де розглянуто як було проведено тестування якості та відповідності вимогам, сформованим в першому розділі, та впровадження і експлуатацію системи.

1 СИСТЕМНИЙ АНАЛІЗ СИМУЛЯЦІЇ РОБОТИ ДРОНІВ ЯК ОБ'ЄКТУ ДОСЛІДЖЕННЯ

1.1 Опис сфери тестування БПЛА

У сучасному світі все більше сфер діяльності потребують ефективних цифрових рішень для обробки та аналізу великих обсягів даних. Однією з таких сфер є тестування безпілотних літальних апаратів (БПЛА), що дедалі активніше застосовуються в логістиці, сільському господарстві, військовій справі, геодезії та екологічному моніторингу[5]. Але через високу вартість дронів, обмеження у фізичному доступі до техніки, а також ризики при тестуванні, виникає потреба у створенні симуляційного середовища, яке дозволяє моделювати поведінку БПЛА без використання реального пристрою.

Типова ситуація — коли розробник хоче перевірити логіку маршруту дрона або протестувати сценарій польоту з перепадами висоти, температури, швидкості, але не має змоги запускати справжній апарат. У таких випадках необхідне програмне забезпечення, яке дозволяє:

1) Імітувати типові сценарії польотів,
2) Змінювати параметри симуляції в реальному часі,
3) Будувати графіки польоту, таблиці з показниками,
4) Зберігати результати в базу даних або експортувати у звіт,
5) Демонструвати користувачеві поведінку дрона без фізичного запуску.

Особливо корисною така система є для:

1) Студентів, які вивчають робототехніку або інженерні спеціальності,
2) Розробників, які створюють ПЗ для дронів, але не мають доступу до реального пристрою,
3) Навчальних закладів, де важлива безпечна демонстрація роботи дронів,
4) Тестувальників, які хочуть перевірити правильність роботи алгоритмів.

Ще однією важливою проблемою є складність існуючих систем: більшість інструментів для тестування дронів вимагають реального підключення, знань програмування, складних налаштувань або доступу до зовнішніх бібліотек. У той же час звичайний користувач потребує простого застосунку з інтуїтивним інтерфейсом, у якому можна вибрати модель дрона, запустити симуляцію, подивитися зміну параметрів і створити звіт.

Рішенням цієї проблеми є мобільний застосунок, що дозволяє:

1) Запускати симульовані польоти з вибором сценарію (наприклад, оглядова місія, доставка вантажу, обліт перешкод),
2) Відображати в режимі реального часу параметри польоту: висоту, швидкість, заряд акумулятора, температуру,
3) Автоматично будувати графіки та діаграми на основі симульованих даних,
4) Формувати звіти, що зберігаються у внутрішню базу,
5) Експортувати результати в PDF

Розробка ведеться на базі Android Studio з використанням мови Kotlin. Для візуалізації використовується бібліотека MPAndroidChart. Дані моделюються вбудованими алгоритмами — генерація значень відбувається з урахуванням випадкових змін, допустимих меж для параметрів дронів та заданих сценаріїв польоту.

Таким чином, симуляційне програмне забезпечення для тестування безпілотників дає можливість навчатися, досліджувати, тестувати та аналізувати польотні сценарії навіть тим, хто не має доступу до реального обладнання. Це особливо важливо у сучасному цифровому суспільстві, де програмна частина все більше переважає над апаратною на етапах навчання, проєктування та валідації.

1.2 Аналіз вимог до програмної системи

Перед тим як розпочати розробку будь-якого програмного забезпечення, особливо якщо мова йде про симуляцію роботи складних систем (таких як

безпілотники), необхідно чітко визначити основні вимоги до системи. Ці вимоги традиційно поділяються на два типи: функціональні та нефункціональні.

Функціональні вимоги описують, що саме повинно вміти робити ПЗ. Вони визначають перелік доступних дій, з якими параметрами працює система, як реагує на введені користувачем значення, та які результати формуються. У випадку нашого мобільного додатку для симуляції тестування дронів це особливо важливо, адже без реального пристрою усе покладається саме на логіку програмного модуля.

Нефункціональні вимоги описують якісні характеристики роботи системи. Вони не стосуються конкретних функцій, але впливають на зручність, стабільність, безпеку та загальну придатність застосунку для щоденного використання.

Функціональні вимоги до мобільного ПЗ для симуляції тестування дронів:

1) Запуск симуляції польоту з заданими параметрами (висота, швидкість, температура, рівень заряду).
2) Вибір типу сценарію польоту: доставка вантажу, сканування місцевості, обхід перешкод.
3) Генерація даних польоту у реальному часі (на основі псевдовипадкових алгоритмів).
4) Відображення основних параметрів на екрані (таблична форма або графік): висота, швидкість, навантаження, координати.
5) Збереження результатів симуляції у локальну базу даних
6) Експорт звіту симуляції у PDF або CSV формат.

Нефункціональні вимоги:

1) Продуктивність — застосунок не повинен "глючити" або зависати при обробці великого обсягу симульованих даних (до 100 тис. точок за сеанс).
2) Швидкодія — побудова графіків та збереження даних мають займати не більше 1–2 секунд на середньостатистичному Android-пристрої.

3) Розширюваність — архітектура має дозволяти додавати нові типи сценаріїв, параметрів або джерел даних без повної переробки логіки.
4) Надійність — система повинна стійко працювати навіть при некоректному введенні даних (наприклад, негативній швидкості чи нульовій висоті).
5) Кросплатформеність — можливість портування додатку на iOS у майбутньому (архітектура має бути модульною).
6) Безпека зберігання — всі дані симуляцій зберігаються локально у зашифрованій формі або з обмеженням доступу до бази.
7) Зручність інтерфейсу — інтерфейс повинен бути зрозумілим навіть для користувача без технічної освіти (підказки, інструкції, кнопка "назад").
8) Автоматичне збереження — при аварійному закритті застосунку симуляція автоматично зберігається з можливістю продовження.

1.3 Моделювання предметної області

Моделювання являється важливим етапом при розробці програмної системи, бо дозволяє краще зрозуміти, як саме має функціонувати ПЗ, яке ми створюємо. UML (Unified Modeling Language) – уніфікована мова моделювання, яка використовується для візуалізації структури та поведінки системи. Вона є досить зручною для побудови як логічної, так і технічної архітектури програми[6].

UML-діаграми умовно поділяються на два основні типи[7]:

1) структурні діаграми (діаграма розгортання, діаграма пакетів, діаграма класів, діаграма компонентів);
2) поведінкові діаграми (діаграма прецедентів, діаграма активності, діаграма послідовності).

Для моделювання предметної області мобільного застосунку для тестування безпілотників було обрано поведінкові діаграми, які дозволяють показати логіку роботи системи з точки зору користувача та послідовність виконання дій.

Діаграма прецедентів — це одна із базових UML-діаграм, яка дозволяє уявити ролі користувачів в системі та їх основні взаємодії з ПЗ. Така діаграма складається з 4 основних елементів[8]:

1) актор (тобто користувач або зовнішня система);
2) прецедент (функція, яку виконує система);
3) система (прямокутник, який обмежує контекст);
4) зв’язки (асоціації, включення, розширення).

На рис. 1 представлено діаграму прецедентів для мобільного застосунку системи тестування безпілотників. Вона включає одного актора – оператора (користувача), який може авторизуватися, обрати безпілотник для тесту, запустити симуляцію польоту, отримати телеметрію, зберегти лог тесту та сформувати звіт. Кожен прецедент пов’язаний з оператором асоціативним зв’язком.

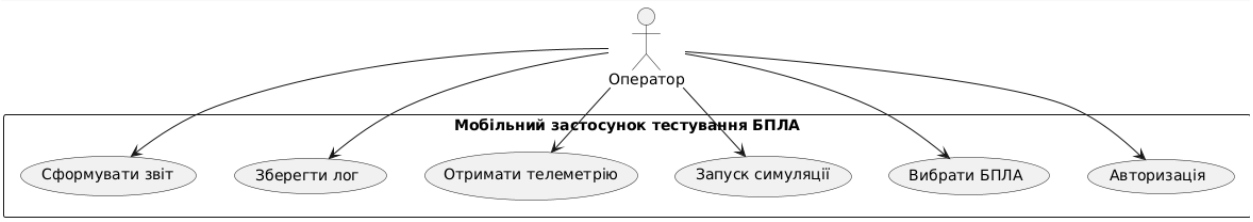


Рис. 1 Діаграма прецедентів

Діаграма активності — це ще один вид UML-діаграми, який зображує логіку виконання дій у вигляді послідовності кроків, із можливими розгалуженнями, паралельними потоками та об'єднанням дій. Вона дозволяє краще зрозуміти, які саме дії виконує система в залежності від вхідних даних або рішень користувача[9].

У ДОДАТКУ А наведено діаграму активності для сценарію симуляції польоту. Спочатку відбувається авторизація, далі — вибір моделі БПЛА. Якщо модель підтримується — відбувається ініціалізація симуляції. Паралельно система показує телеметрію і записує лог. Після завершення — система пропонує зберегти результат та сформувати звіт.

Діаграма послідовності — це тип поведінкової діаграми, яка показує взаємодію між об'єктами у вигляді обміну повідомленнями з плином часу. Основні елементи тут:

1) лінія життя об'єкта;
2) смуга активації (коли об'єкт щось виконує);
3) повідомлення (синхронні чи асинхронні виклики).

У ДОДАТКУ Б наведена діаграма послідовності, яка показує, як користувач авторизується, передає команду на запуск симуляції, мобільний застосунок надсилає запит до симуляційного ядра, потім отримує телеметрію, зберігає лог в базу, і після завершення формує звіт. Передбачено також альтернативний шлях — якщо авторизація не пройшла успішно, користувач отримає повідомлення про помилку.

Загалом, використання цих трьох типів діаграм дало змогу краще спроєктувати логіку програми, продумати сценарії взаємодії з користувачем та уникнути помилок ще до початку програмування.

1.4 Огляд інформаційних джерел та існуючих рішень

На сучасному ринку існує велика кількість програмних рішень, які частково або повністю покривають функціонал систем тестування безпілотних літальних апаратів (БПЛА). Більшість із них орієнтовані на тестування окремих модулів, таких як навігація, керування польотом, телеметрія або безпека, проте комплексних і водночас гнучких систем, які охоплюють повний цикл автоматизованого тестування, поки що небагато.

Одним з найвідоміших рішень у цій сфері є Dronecode SDK(рис 2)— бібліотека з відкритим кодом, що підтримує управління та тестування БПЛА. Переваги Dronecode SDK:

1) Підтримка широкого спектру апаратних платформ на базі PX4;
2) Можливість автоматизованого тестування польотних сценаріїв;

3) Інтеграція з симуляторами (наприклад, Gazebo або AirSim).

Недоліки:

1) Високий поріг входу для початківців;
2) Обмежена документація для розробників сторонніх додатків;
3) Не всі типи тестів (наприклад, стрес-тести або регресійні) реалізовані з коробки.

Рис. 2.Dronecode

Іншим розповсюдженим рішенням є **AirSim(Рис 3)** від Microsoft — симулятор для тестування автономних транспортних засобів, зокрема безпілотників.

Переваги AirSim:

1) Реалістичне фізичне моделювання та рендеринг;
2) Підтримка API для автоматичного тестування та машинного навчання;
3) Інтеграція з Unreal Engine.

Недоліки:

1) Велика ресурсомісткість (високі вимоги до системи);

2) Відсутність вбудованогоінтерфейсу для створення сценаріїв тестування без програмування;
3) Неповна підтримка польотних контролерів.

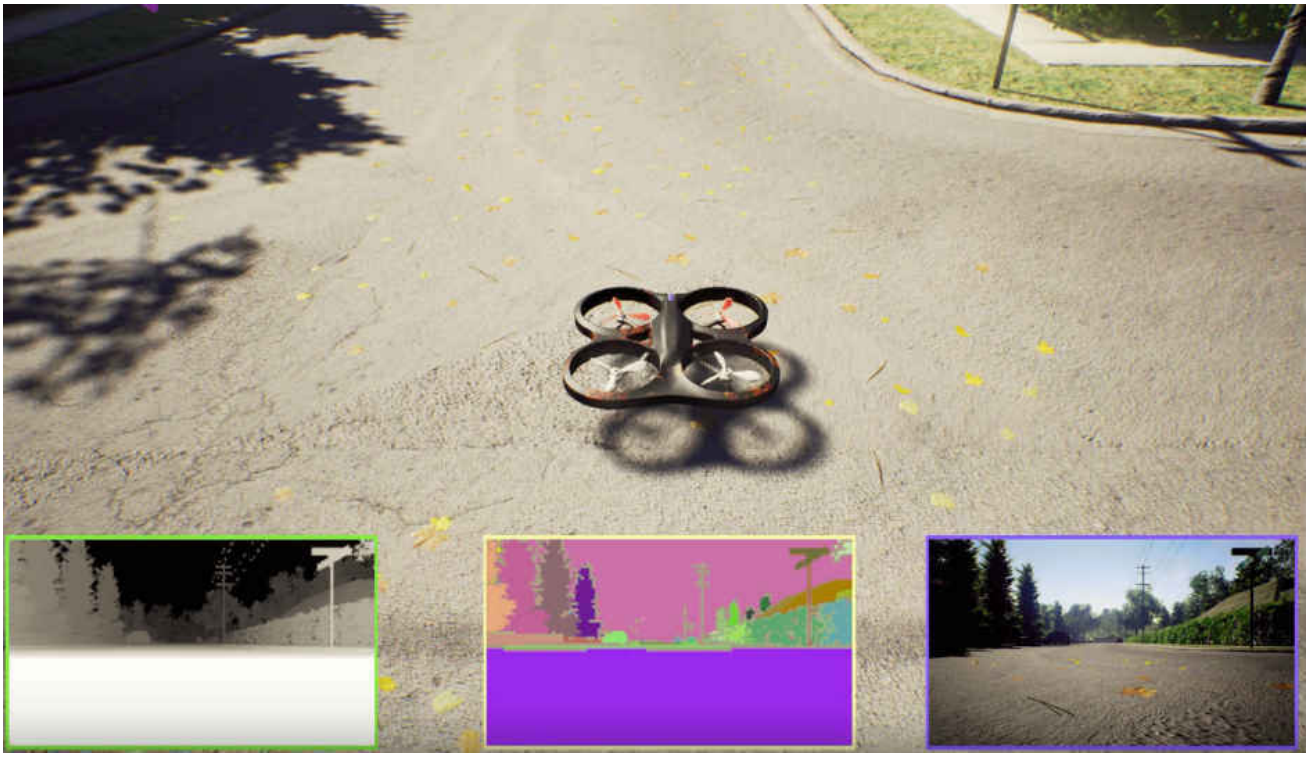


Рис 3. AirSim

Серед промислових рішень виділяється Auterion Suite(рис 4) — комерційна система з повним стеком інструментів для управління, тестування та моніторингу БПЛА.

Переваги**:**

1) Професійна підтримка та масштабованість;
2) Інтеграція з хмарними службами;
3) Високий рівень безпеки.

Недоліки:

1) Висока вартість використання;
2) Закритий код і обмежені можливості кастомізації;
3) Орієнтація переважно на великих корпоративних клієнтів.

Рис. 4

1.4.3 Порівняння існуючих рішень. Для порівняння існуючих рішень, що були представлені раніше, було вирішено створити порівняльну таблицю на основі метрик(табл. 1).

Таблиця 1

Порівняння існуючих рішень

Метрика	**Dronecode SDK**	**AirSim**	**Auterion Suite**
Автоматизоване тестування	+	+	+
Підтримка симуляції	-	+	+
Інтеграція з апаратними модулями	+	-	+
Створення сценаріїв тестування	+	-	+
Підтримка API/SDK	+	+	+

Наявність графічного інтерфейсу	-	+	+

Для порівняння існуючих рішень було вирішено обрати такі метрики:

1) **Автоматизоване тестування** – наявність підтримки для запуску серій тестів без втручання людини;
2) **Підтримка симуляції** – можливість тестування віртуально без потреби у фізичному апараті;
3) **Інтеграція з апаратними модулями** – здатність взаємодіяти з реальними сенсорами, камерами, GPS тощо;
4) **Створення сценаріїв тестування** – чи є можливість легко задавати умови польоту, дії та перевірки результатів;
5) **Підтримка API/SDK** – наявність програмного інтерфейсу для інтеграції з іншими системами або додатками;
6) **Наявність графічного інтерфейсу** – візуальне середовище керування тестами;

1.5 Постановка завдання

Метою розробки програмного забезпечення мобільного додатку тестування безпілотників є створення програмного рішення, яке надасть зручні та ефективні інструменти для керування тестовими польотами, завданнями та моніторингом результатів, що полегшить роботу операторів і інженерів, які потребують контролю над процесом тестування безпілотних літальних апаратів у реальному часі.

Для забезпечення цього система має покривати такий функціонал:

1) Авторизація користувача в системі
2) Реєстрація користувача в системі
3) Управління безпілотниками

a. Перегляд списку дронів із деталями (модель, серійний номер, версія прошивки)
b. Додавання, редагування та видалення дронів
c. Підключення та відключення дронів
d. Оновлення прошивки
e. Перегляд архіву тестів та статусів

4) Тестові сесії
   a. Створення нової сесії тестування з прив’язкою до дрона та користувача
   b. Перегляд списку сесій із сортуванням за датами
   c. Відображення сесій на поточний день, включно з виконаними
   d. Редагування та видалення сесій
   e. Позначка сесії як завершеної
   f. Повтор запуску тесту
5) Тестові сценарії
   a. Перегляд та вибір сценаріїв, згрупованих за типами
   b. Додавання та редагування сценаріїв у сесії
   c. Виконання або скасування сценарію
6) Телеметрія
   a. Збір і відображення точкових даних телеметрії (координати, висота, швидкість, заряд батареї)
   b. Валідація та нормалізація телеметрії
7) Події
   a. Додавання, редагування та видалення подій
   b. Повідомлення про критичні події
   c. Підтвердження або відхилення повідомлень
8) Звіти

a. Автоматична генерація звітів після тестів
b. Відправка звітів електронною поштою
c. Перегляд збережених звітів із резюме та файлами

Вхідною інформацією у системі є:

1) Інформація про тестове завдання
    a. Ідентифікатор сесії
    b. Дата створення
    c. Дедлайн
    d. Статус
    e. Оператор, який проводить тестування
    1. Безпілотник, що тестується
2) Інформація про тестові сценарії
    a. Ідентифікатор сценарію
    b. Назва сценарію
    c. Опис сценарію
    d. Категорія або тип сценарію
3) Інформація про безпілотник
    a. Ідентифікатор безпілотника
    b. Назва моделі
    c. Серійний номер (опис, ідентифікація)
    d. Версія програмного забезпечення
    e. Дата початку та завершення експлуатації (можна додати)
4) Інформація про користувача
    a. Ідентифікатор користувача
    b. Ім'я користувача
    c. Email
    d. Пароль

5) Інформація про події
    a. Ідентифікатор події
    b. Дата створення
    c. Категорія події
    d. Опис події
6) Інформація про телеметричні дані
    a. Ідентифікатор запису
    b. Час запису
    c. Географічні координати (широта, довгота)
    d. Висота
    e. Швидкість
    f. Рівень заряду батареї
7) Інформація про звіти
    a. Ідентифікатор звіту
    b. Дата створення звіту
    c. Короткий опис
    d. Посилання на файл звіту

Вихідна інформація з системи

1) Список тестових завдань
    a. Виконані та невиконані сесії
    b. Інформація про оператора і безпілотник
2) Список безпілотників
    a. Статуси і основна інформація про дрони

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Важливою частиною програмного забезпечення є база даних, яка відіграє критичну роль у збереженні, управлінні та доступі до даних. База даних є основою функціонування багатьох систем не залежно від варіанту реалізації. Вона забезпечує надійність, цілісність та безпеку даних. [1]

Першим етапом в проєктуванні бази даних є створення логічної моделі даних. Вона є дуже важливим інструментом, що допомагає візуалізувати структуру даних та зрозуміти які взаємозв'язки є між ними. Логічна модель не тільки описує сутності, атрибути та взаємозв'язки між ними в логічному контексті, але й спрощує розуміння проєкту. Вона дозволяє перевірити коректність проєктуємої структури даних та запобігти дублюванню чи втраті даних. За допомогою логічної моделі розробники можуть створювати логічну основу для фізичної реалізації, що дозволяє передбачити, які дані потрібні для забезпечення коректної роботи всього функціоналу системи. [2]

Логічна модель даних складається з трьох основних елементів:

1) сутності (*entities*) – це конкретні об‘єкти, про які зберігається інформація, вони відображають таблиці в базі даних та їх поля;

2) атрибути (*attributes*) – це властивості сутностей, які описують їх певні характеристики, вони визначаються в межах сутностей, і допомагають зрозуміти, яка інформація має зберігатись в кожній сутності;

3) відношення (*relationships*) – це зв’язки між сутностями, що визначають характер зв’язків такі як один до одного, один до багатьох, багато до багатьох:

а) один до одного – це тип зв’язку, що вказує на те, що одна сутність може бути пов’язана лише з однією іншою сутністю;

b) один до багатьох – це тип зв’язку, що вказує на те, що одна сутність може мати відношення з багатьма екземплярами іншої сутності;

c) багато до багатьох – це тип зв’язку, що вказує на те, що багато сутностей однієї таблиці можуть бути пов’язаними з багатьма сутностями іншої таблиці. [3]

На основі аналізу предметної області та постановки завдання було спроєктовано логічну модель даних для програмного забезпечення тестування дронів(рис. 5).

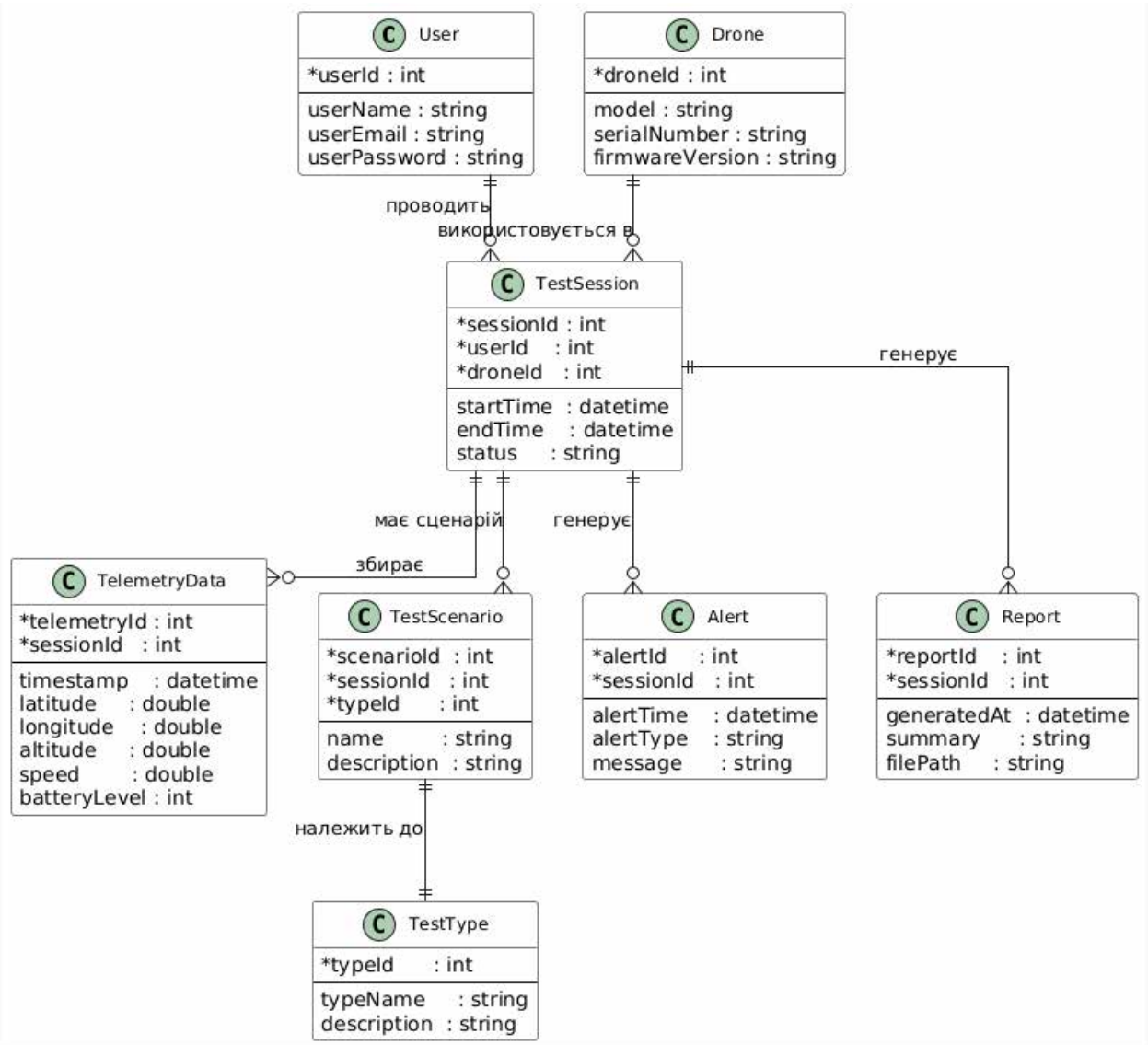


Рис. 5 Логічна модель даних

На рис. 10 зображено вісім сутностей:

1) User – користувач системи
    a. ключовий атрибут: userId (PK)
    b. неключові атрибути:
        userName – ім’я оператора
        userEmail – електронна пошта
        userPassword – пароль
2) Drone – безпілотник
    a. ключовий атрибут: droneId (PK)
    b. неключові атрибути:
        model – модель дрона
        serialNumber – серійний номер
        firmwareVersion – версія прошивки
3) TestSession – сесія тестування
    a. ключові атрибути:
        sessionId (PK)
        userId (FK)
        droneId (FK)
    b. неключові атрибути:
        startTime – час початку сесії
        endTime – час завершення сесії
        status – статус тестування (наприклад, «успішно», «помилка»)
4) TelemetryData – телеметрія
    a. ключовий атрибут: telemetryId (PK)
    b. зовнішній ключ: sessionId (FK)
    c. неключові атрибути:
        timestamp – мітка часу запису

latitude – широта

longitude – довгота

altitude – висота

speed – швидкість

batteryLevel – рівень заряду акумулятора

5) TestScenario – сценарій тесту
    a. ключовий атрибут: scenarioId (PK)
    b. зовнішній ключ: sessionId (FK)
    c. зовнішній ключ: typeId (FK)
    d. неключові атрибути:

        name – назва сценарію

        description – опис сценарію
6) TestType – тип сценарію
    a. ключовий атрибут: typeId (PK)
    b. неключові атрибути:

        typeName – назва типу (наприклад, «політ за маршрутом»)

        description – опис типу
7) Alert – повідомлення про подію під час тестування
    a. ключовий атрибут: alertId (PK)
    b. зовнішній ключ: sessionId (FK)
    c. неключові атрибути:

        alertTime – час спрацювання

        alertType – тип повідомлення (наприклад, «низький заряд»)

        message – текст повідомлення
8) Report – звіт за результатами сесії
    a. ключовий атрибут: reportId (PK)
    b. зовнішній ключ: sessionId (FK)

c. неключові атрибути:

generatedAt – дата й час створення звіту

summary – короткий опис результатів

filePath – шлях до файлу звіту

2.2 Діаграма класів

Діаграма класів це один із інструментів моделювання в об’єктно-орієнтованому програмуванні, необхідний для візуалізації структури класів системи та їх взаємозв’язків між собою. Основними елементами діаграми класів є: класи, атрибути, методи, зв’язки (спадкування, асоціація, композиція, агрегація).

Класи представляють об’єкти системи і мають певні методи та зв’язки з іншими класами. Атрибути дозволяють визначати стан класу, методи ж описують поведінку класу.

Діаграми класів є важливим елементом проєктування програмного забезпечення, адже дозволяють поглибити розуміння архітектури системи, виявити можливі проблеми в проєктування та полегшують подальшу розробку програмного продукту.

Діаграма класів (Рис.6) для програмного забезпечення тестування дронів включає наступні класи та зв'язки:

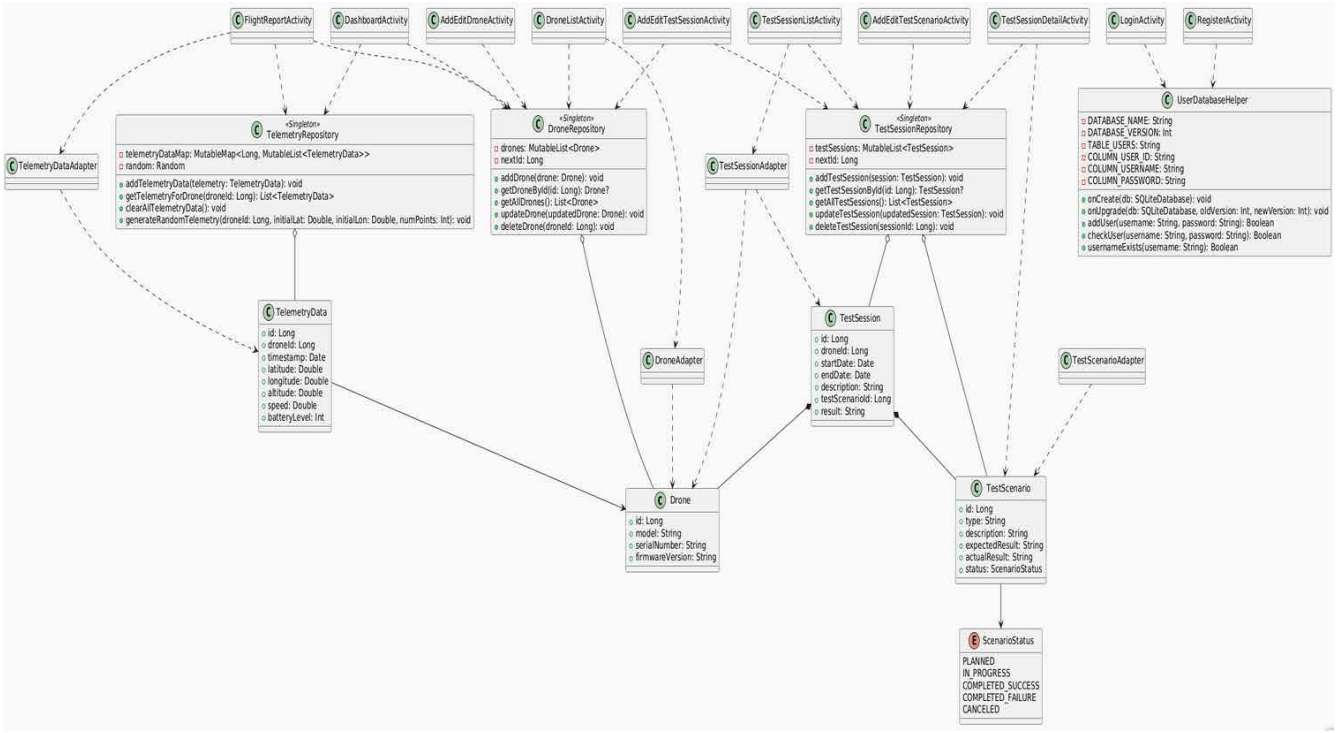


Рис. 6. Діаграма класів

1) Клас User Призначення: Описує оператора або інженера, який взаємодіє з системою. Його дані управляються через UserDatabaseHelper. Атрибути: userId, userName, userEmail, userPassword. Методи: login(), logout(), updateProfile(). Взаємодія: LoginActivity та RegisterActivity використовують UserDatabaseHelper для взаємодії з даними користувача.
2) Клас Drone Призначення: Модель безпілотника, яка реєструється та керується в системі. Один Drone може брати участь у багатьох TestSession. Атрибути: droneId, model, serialNumber, firmwareVersion. Методи: connect() (логічна імітація підключення), disconnect() (логічна імітація відключення), updateFirmware(). Взаємодія: Керується за допомогою DroneRepository. DashboardActivity імітує підключення до Drone. TestSession асоціюється з одним Drone.
3) Клас TestSession Призначення: Відповідає за сесію тестування дрона, агрегуючи дані про її початок, кінець та результат. Атрибути: sessionId, startTime, endTime, status, testScenarioId (ідентифікатор сценарію), result (результат сесії). Методи: startSession(), endSession(), retry(). Взаємодія: TestSessionRepository керує створенням та управлінням TestSession. TestSession пов'язаний з одним Drone (який тестується) та одним

TestScenario (який виконується). TestSession також відповідає за агрегацію TelemetryData.

4) Клас TelemetryData Призначення: Представляє точкові дані телеметрії (наприклад, з симуляції польоту) для певного дрона в конкретний момент часу. Атрибути: telemetryId, timestamp, latitude, longitude, altitude, speed, batteryLevel. Методи: validate(), normalize() (для обробки даних). Взаємодія: TelemetryRepository зберігає та надає доступ до TelemetryData. Генерується під час симуляції польоту в DashboardActivity і відображається у FlightReportActivity.
5) Клас TestScenario Призначення: Описує сценарій або конкретний етап тестування, який виконується в рамках TestSession. Атрибути: scenarioId, type, description, expectedResult, actualResult, status (статус виконання сценарію). Методи: execute(), abort(). Взаємодія: TestScenario пов'язаний з TestSession і може бути частиною TestType.
6) enum class ScenarioStatus Призначення: Перелік можливих статусів виконання TestScenario (PLANNED, IN_PROGRESS, COMPLETED_SUCCESS, COMPLETED_FAILURE, CANCELED).
7) Клас TestType Призначення: Довідкова сутність, що визначає тип сценарію тестування (наприклад, "Перевірка батареї", "Тест дальності"). Атрибути: typeId, typeName, description. Методи: Відсутні (це чиста довідкова сутність). Взаємодія: TestScenario асоціюється з одним TestType.
8) Клас Alert Призначення: Відповідає за повідомлення про критичні події або відхилення під час тестування або польоту. Атрибути: alertId, alertTime, alertType, message. Методи: acknowledge(), dismiss(). Взаємодія: Може генеруватися TestSession при виникненні аномалій.
9) Клас Report Призначення: Містить звіт за результатами тестування або симуляції польоту. Атрибути: reportId, generatedAt, summary, filePath. Методи: generatePDF(), sendByEmail(). Взаємодія: FlightReportActivity відповідає за відображення та генерацію PDF-звітів. Концептуально, TestSession (або TelemetryRepository) генерує дані, на основі яких формується Report.
10) Клас UserDatabaseHelper Призначення: Управляє локальним зберіганням даних користувачів. Методи: onCreate(), onUpgrade(), addUser(), checkUser(), usernameExists(). Взаємодія: Використовується LoginActivity та RegisterActivity для автентифікації та реєстрації.
11) Клас DroneRepository Призначення: Синглтон-репозиторій для управління колекцією об'єктів Drone (додавання, отримання, оновлення, видалення). Методи: addDrone(), getDroneById(), getAllDrones(),

updateDrone(), deleteDrone(). Взаємодія: Використовується DroneListActivity, AddEditDroneActivity та DashboardActivity.

12) Клас TestSessionRepository Призначення: Синглтон-репозиторій для управління колекцією об'єктів TestSession (додавання, отримання, оновлення, видалення). Методи: addTestSession(), getTestSessionById(), getAllTestSessions(), updateTestSession(), deleteTestSession(). Взаємодія: Використовується TestSessionListActivity, AddEditTestSessionActivity, TestSessionDetailActivity та AddEditTestScenarioActivity.
13) Клас TelemetryRepository Призначення: Синглтон-репозиторій для зберігання та надання доступу до згенерованих даних TelemetryData, пов'язаних з дронами. Методи: addTelemetryData(), getTelemetryForDrone(), clearAllTelemetryData(), generateRandomTelemetry(). Взаємодія: Використовується DashboardActivity для симуляції та FlightReportActivity для відображення звітів.
14) UI-класи (Activities та Adapters): Активності: LoginActivity, RegisterActivity, DashboardActivity, DroneListActivity, AddEditDroneActivity, TestSessionListActivity, AddEditTestSessionActivity, TestSessionDetailActivity, AddEditTestScenarioActivity, FlightReportActivity. Вони відповідають за відображення інтерфейсу користувача та взаємодію з репозиторіями. Адаптери: DroneAdapter, TestScenarioAdapter, TestSessionAdapter, TelemetryDataAdapter. Відповідають за відображення колекцій даних у RecyclerView.

2.3 Діаграма пакетів

Діаграма пакетів – це один із інстументів моделювання, що дозволяє спроектувати структуру програмного забезпечення на рівні пакетів. Мета даної діаграми полягає в групування класів, інтерфейсів та інших програмних елементів у логічні пакети.

У мобільному застосунку тестування безпілотників реалізовано компонентну архітектуру, яка забезпечує чіткий розподіл відповідальностей між частинами системи. Основні компоненти організовані наступним чином:

1) UI — відповідає за взаємодію з користувачем. Містить екрани застосунку (наприклад, LoginActivity, DashboardActivity, тощо), обробляє події користувача, викликає відповідні сервіси або методи репозиторіїв.

2) Adapter — допоміжні компоненти, які адаптують моделі до відображення у списках або таблицях. Вони тісно пов’язані з UI та моделями.
3) Model — визначає основні сутності предметної області: Drone, TestSession, TelemetryData, TestScenario, TestType, Alert, Report, User, тощо.
4) Repository — реалізує логіку доступу до даних. Використовується UI для отримання або збереження даних. Репозиторії взаємодіють із моделями та базою даних.
5) Database — відповідає за фізичне збереження даних. Містить класи доступу до локального сховища (наприклад, UserDatabaseHelper).
6) Service — фонові або технічні сервіси, які можуть забезпечувати симуляцію, з'єднання з дроном або іншу обробку незалежно від UI.
7) Util — набір допоміжних функцій (наприклад, форматування дат, константи), які можуть використовуватись усіма іншими компонентами.

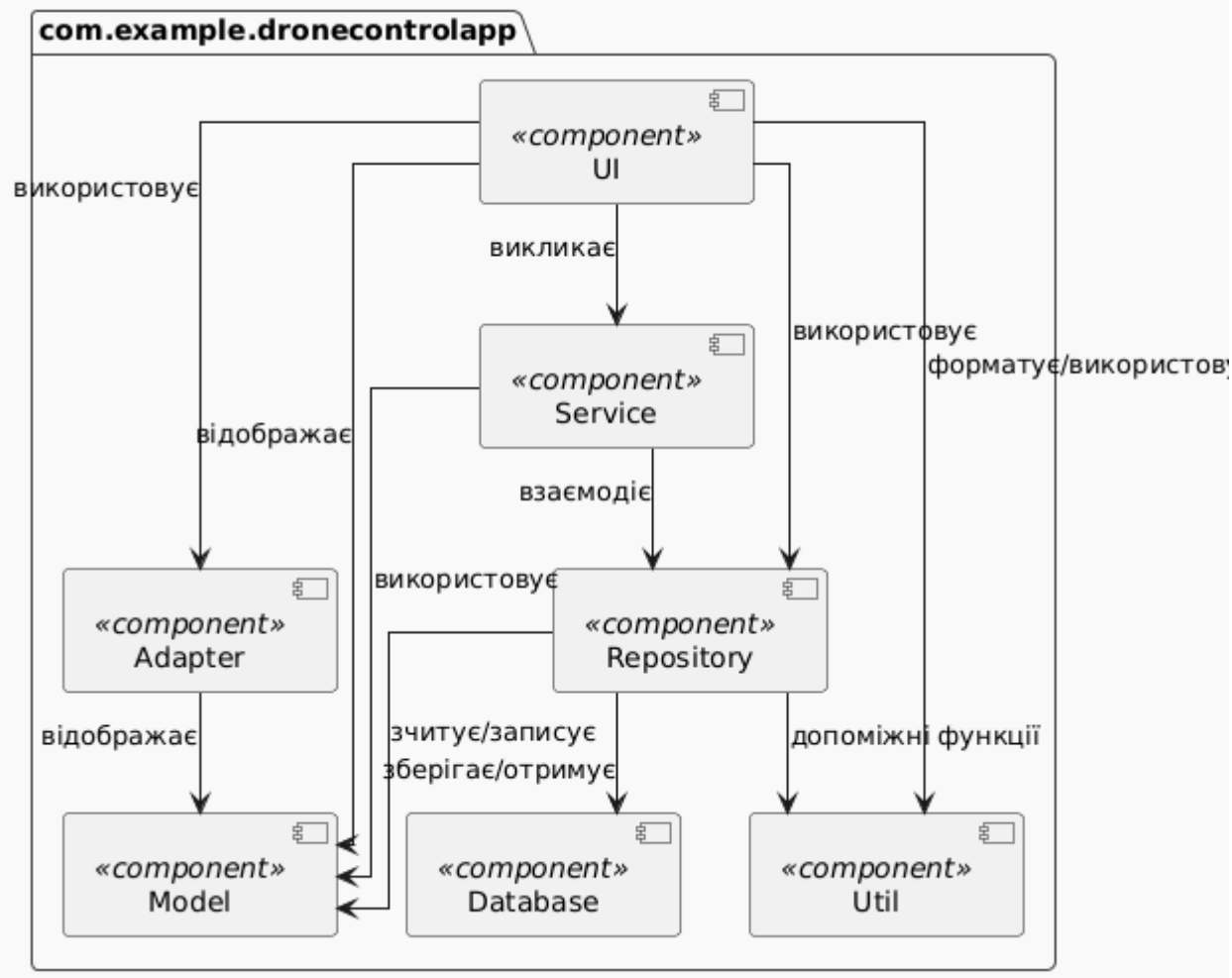


Рис. 7 Діаграма пакетів

2.4 Діаграма компонентів

Діаграма компонентів є одним з типів UML-діаграм і використовується для моделювання структури програмної системи з точки зору компонентів та залежностей між ними. Вона дозволяє відобразити, як компоненти системи взаємодіють між собою та як вони організовані в логічну архітектуру.

Основними елементами діаграми компонентів є компоненти, інтерфейси, залежності та артефакти. Компоненти представляють фізичні або логічні частини системи, які можуть бути реалізовані як окремі модулі або підсистеми. Інтерфейси визначають спосіб взаємодії між компонентами. Залежності вказують на зв'язки між компонентами, такі як використання одного компонента іншим. Артефакти

представляють фізичні ресурси, такі як виконувані файли або бібліотеки, що використовуються компонентами.

Діаграма компонентів використовується для визначення структури системи та допомагає розробникам зрозуміти, як компоненти взаємодіють між собою. Вона допомагає виявити модульність системи, ідентифікувати залежності та розподілити відповідальності між компонентами. Діаграма компонентів також може бути використана для аналізу потенційних проблем в системі, таких як залежності, перевантаження або недостатня модульність.

На рис. 8представлено діаграму компонентів для програмного забезпечення мобільного застосунку для тестування дронів

Діаграма компонентів, яку ми розглядаємо, надає важливу інформацію про структуру нашої програмної системи. Головним компонентом є .exe файл, який є основним виконуваним файлом нашої програми. Він складається з різних менших компонентів, які відповідають за різні частини функціональності програми.

Окрім того, на діаграмі видно наявність важливих файлів, які визначають основні елементи дизайну нашої програми. Ці файли можуть включати графічні ресурси, шаблони, стилі та інші складові, що використовуються для створення інтерфейсу та взаємодії з користувачем. Крім того, діаграма показує наявність файлу конфігурації, який містить параметри налаштування програми.

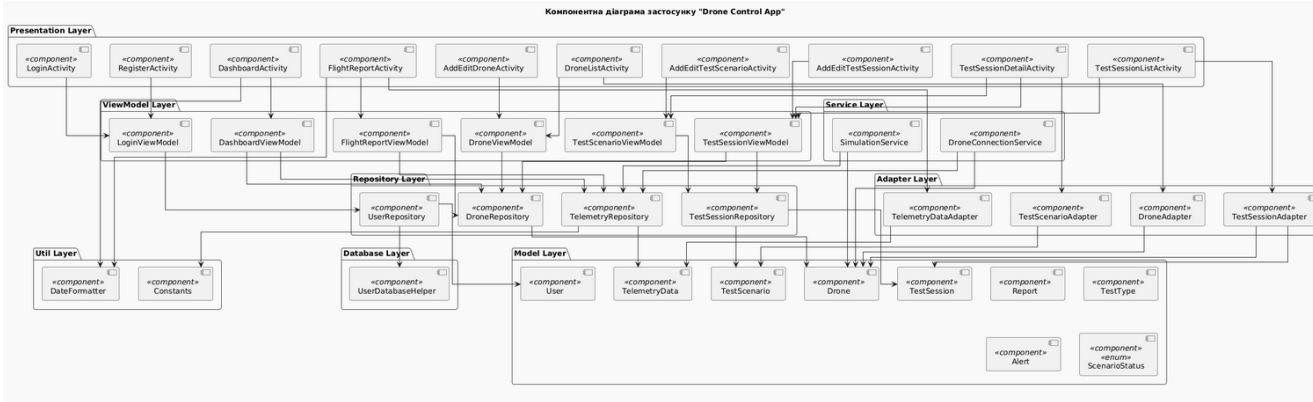


Рис. 8 Діаграма компонентів

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління базою даних

Для програмного забезпечення мобільного застосунку тестування дронів необхідне збереження та обробка даних. Для забезпечення цього було обрано реляційну базу даних.

Реляційна база даних – це один із основних типів баз даних, що використовує реляційну модель даних. Особливостями даної бази даних є те, що дані організовані у вигляді таблиць з рядками та стовпцями, де кожен рядок представляє окремий запис, а стовпці відповідають певному типу даних, яких зберігається. Саме реляційна база даних дозволяє зберігати дані у пов’язаних таблицях та ефективно маніпулювати цими даними, виконуючи складні запити.

Серед переваг реляційних баз даних можна виділити:

1) структурованіть даних, за рахунок таблиць, що спрощує роботу з даними;

2) високий рівень незалежності даних, тобто зміна структури таблиць не впливає на роботу з даними;

3) цілісність даних, забезпечена шляхом використання ключів.

Але при роботі з реляційними базами даних варто пам’ятати про деякі особливості, наприклад, перед створенням об’єктів бази даних варто визначити відношення між ними та структуру, потрібно правильно використовувати ключі, щоб не виникало проблем під час виконання запитів.

Для даного програмного забезпечення в якості бази даних було обрано SQLite, через низку переваг:

1) легкість використання – дана СУБД є простою у використанні та має простий інтерфейс, що дозволяє швидко розпочати роботу, не витрачаючи час на вивчення системи;

2) низький рівень системних вимог – SQLite вимагає доволі мало ресурсів, таких як пам’ять, що є неймовірно важливим при розробці мобільного додатку;

3) вбудована база даних – SQLite входить в стандартну конфігурацію Android, тому не потребує додаткового встановлення та налаштування;

4) швидкість – SQLite має оптимізовані алгоритми, що дозволяють системі швидко оброблювати запити з мінімальною кількістю помилок;

5) транзакційна безпека – SQLite підтримує такі властивості як: атомарність, консистентність, ізольованість, стійкість, що гарантує безпеку та цілісність даних;

6) підтримка SQL – дана СУБД є повністю сумісною з SQL-стандартом, що дозволяє використовувати мову запитів при роботі з даними;

Але, звісно, SQLite має певні недоліки:

1) обмежена масштабованість – SQLite може бути неефективною, коли потрібно працювати з дуже великими наборами даних;

2) відсутність підтримки серверного режиму – SQLite працює лише як вбудована база даних, тобто не підтримує клієнт-серверну архітектуру;

3) відсутність додаткових функцій – SQLite має менший функціонал в порівнянні з MySQL чи PostgreSQL, він не підтримує складні операції, як наприклад вбудовані процедури.

Серед варіантів, які були розглянуті та відкинуті варто згадати:

1) *«Realm»* – це мобільна СУБД, яка пропонує доволі легкий та швидкий спосіб зберігання та синхронізації даних між мобільними пристроями та сервером. Але дана база даних має певні недоліки, зокрема це доволі обмежена підтримка запитів та структури даних; [34]

2) *«Firebase Realtime Database»* – це хмарна СУБД, яка може забезпечити синхронізацію даних в реальному часі між клієнтом та сервером. Дана СУБД є затратною та має доволі обмежений функціонал порівняно з аналогами; [35]

3) *«PostgreSQL»* – це реляційна СУБД з повною підтрикою SQL-стандарту, але має дуже високі вимоги до системи, особливо до пам’яті та процесору, і має складне налаштування, в порівнянні зі вбудованими СУБД; [36]

4) *«MongoDB»* – це документо-орієнтована СУБД, яка зберігає дані у вигляді документів у форматі JSON. Але, дана СУБД, немає повноцінної транзакційної підтримки та має високі вимоги до пам’яті, де повинні зберігатись дані. [37]

3.2 Розробка об’єктів бази даних

3.2.1 Створення бази даних. Для створення бази даних та її об’єктів було використано SQL. Це стандартна мова запитів, яка необхідна для взаємодії з реляційними базами даних. Вона дозволяє створювати, модифікувати та управляти даними в базі даних. На рис. 9 показано код створення бази даних для програмного забезпечення тестування БПЛА

```
CREATE DATABASE DroneTestDB;
```

Рис. 9.Код створення бази даних

3.2.2 Створення таблиць. На основі логічної моделі з пункту 2.1, було розроблено код для створення таблиць. На рис. 10 наведено код для створення таблиці User на основі сутності з логічної моделі даних.

Код створення всіх інших таблиць наведено в ДОДАТКУ Б.

```
CREATE TABLE [User] (
    userId INT PRIMARY KEY IDENTITY(1,1),
    userName NVARCHAR(100) NOT NULL,
    userEmail NVARCHAR(100) NOT NULL UNIQUE,
    userPassword NVARCHAR(255) NOT NULL
);
```

Рис. 10 Код створення таблиці User

3.2.3 Створення процедур. Процедури – це певний набір інструкцій SQL, які можуть бути викликані для виконання певних операцій. Основною метою процедур є оптимізація роботи з базою даних, адже вони дозволяють не повторювати певні операції, що полегшує управління кодом.

Процедура proc_ShowArchivedTestSessions показує користувачу сесії тестування безпілотників, які вже завершені.. На рис. 11 зображена її реалізація, де в нас є дві умови, чи задача виконана, чи задача належить списку задач, а також задачі в нас відсортовані за датою, для показу лише тих задач, що були виконані сьогодні.

```
CREATE PROCEDURE proc_ShowArchivedTestSessions
AS
BEGIN
    SELECT sessionId, startTime, endTime, status
    FROM TestSession
    WHERE status = 'Completed' AND CAST(endTime AS DATE) = CAST(GETDATE() AS DATE)
    ORDER BY endTime DESC;
END;
```

Рис. 11 Код створення процедури proc_ShowArchivedTestSessions

На рис. 12 зображено код процедури proc_UpdateFirmwareVersion для оновлення версії прошивки безпілотника.

```
CREATE PROCEDURE proc_UpdateFirmwareVersion
    @droneId INT,
    @newFirmwareVersion NVARCHAR(50)
AS
BEGIN
    UPDATE Drone
    SET firmwareVersion = @newFirmwareVersion
    WHERE droneId = @droneId;

    SELECT * FROM Drone WHERE droneId = @droneId;
END;
```

Рис. 12 Код створення процедури proc_UpdateFirmwareVersion

3.2.4 Створення тригерів. Тригери – це спеціальні об’єкти бази даних, які виконуються автоматично при відповідності певним умовам, таким як INSERT, DELETE, UPDATE. Тригери можуть бути прив’язаними до певних таблиць і виконуватись автоматично після виконання дій з цією таблицею. Вони використовуються для забезпечення цілісності даних та мінімізації можливих помилок.

На рис. 13 представлений код створення тригеру UpdateTestSessionTimestamp, який оновлює поле lastUpdated в таблиці TestSession при зміні запису.

```
CREATE TRIGGER UpdateTestSessionTimestamp
ON TestSession
AFTER UPDATE
AS
BEGIN
    UPDATE TestSession
    SET lastUpdated = GETDATE()
    FROM inserted
    WHERE TestSession.sessionId = inserted.sessionId;
END;
```

Рис. 13 Код створення тригера UpdateTestSessionTimestamp,

На рис. 14 представлений тригер PreventInvalidTelemetryData який забороняє додавання телеметрії з некоректними координатами.

```
CREATE TRIGGER PreventInvalidTelemetryData
ON TelemetryData
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (
        SELECT * FROM inserted
        WHERE latitude NOT BETWEEN -90 AND 90
        OR longitude NOT BETWEEN -180 AND 180
    )
    BEGIN
        RAISERROR('Invalid telemetry coordinates.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END

    INSERT INTO TelemetryData SELECT * FROM inserted;
END;
```

Рис. 14 Код створення тригеру PreventInvalidTelemetryData

3.2.5 Створення користувачів. Для забезпечення безпеки системи та доступу користувачів до бази даних було створено роль користувача, який може лише змінювати записи в таблиці (рис. 15).

```
CREATE LOGIN TestUser WITH PASSWORD = 'SecurePass123!';
CREATE USER TestUser FOR LOGIN TestUser;
ALTER ROLE db_datawriter ADD MEMBER TestUser;
ALTER ROLE db_datareader ADD MEMBER TestUser;
```

Рис. 15 Код створення користувача та надання йому прав

3.3 Вибір інструментарію для створення мобільного застосунку тестування

Для розробки інтерфейсу програмного забезпечення мобільного застосунку для тестування дронів було обрано інструмент *Figma*. Серед переваг даного програмного забезпечення можна виділити:

1) Для розробки інтерфейсу програмного забезпечення мобільного додатку тестування безпілотників було обрано інструмент **Figma**. Серед переваг даного програмного забезпечення можна виділити:
2) професійний дизайн — Figma дозволяє створювати зручний та інтуїтивний інтерфейс для користувачів, що важливо для ефективного тестування безпілотних літальних апаратів, завдяки широким можливостям макетування, стилізації та створення векторних графічних елементів;
3) прототипування — можливість створення інтерактивних прототипів допомагає моделювати сценарії роботи додатку та вчасно виявляти недоліки у взаємодії користувача із системою;
4) безкоштовний план — Figma є безкоштовною для невеликих команд, що дозволяє зекономити кошти на початкових етапах розробки

Для розробки самого програмного забезпечення мобільного додатку було обрано мову програмування **Kotlin**, через такі переваги:

1) кросплатформованість — Kotlin забезпечує роботу додатку на різних пристроях та версіях Android, що дозволяє охопити широку аудиторію користувачів;
2) багатий набір бібліотек та фреймворків — доступні високоякісні інструменти, зокрема Android SDK, які значно спрощують розробку функціоналу для тестування безпілотників, таких як відображення телеметрії, керування дроном та збір даних;

3) підтримка Kotlin-розробки в Android Studio — офіційне середовище розробки надає зручні інструменти для написання, налагодження та тестування мобільного додатку

Для розробки було обрано інтегроване середовище **Android Studio**, яке має такі переваги:

1) повна підтримка Android-платформи — забезпечує всі необхідні інструменти для створення та тестування додатків на Android;
2) інтеграція з Android SDK — дає змогу легко управляти бібліотеками, API та різними компонентами системи;
3) висока продуктивність та зручність — швидка робота середовища і наявність багатьох корисних функцій прискорює розробку та підвищує якість кінцевого продукту.

3.4 Алгоритмізація та програмування програмних модулів

Для кращого розуміння алгоритму дій системи було змодельовано основні бізнес процеси у вигляді блок-схеми. Адже даний вид діаграми є корисним для візуалізації алгоритму роботи модуля програми та виявлення помилок та проблем в логіці.

На рис. 16 зображено алгоритм авторизації користувача в системі.

При запуску додатку користувач вводить логін і пароль. Система перевіряє наявність облікового запису в базі даних. У разі успішної авторизації користувач отримує доступ до функцій додатку, інакше — відображається повідомлення прozпомилку.

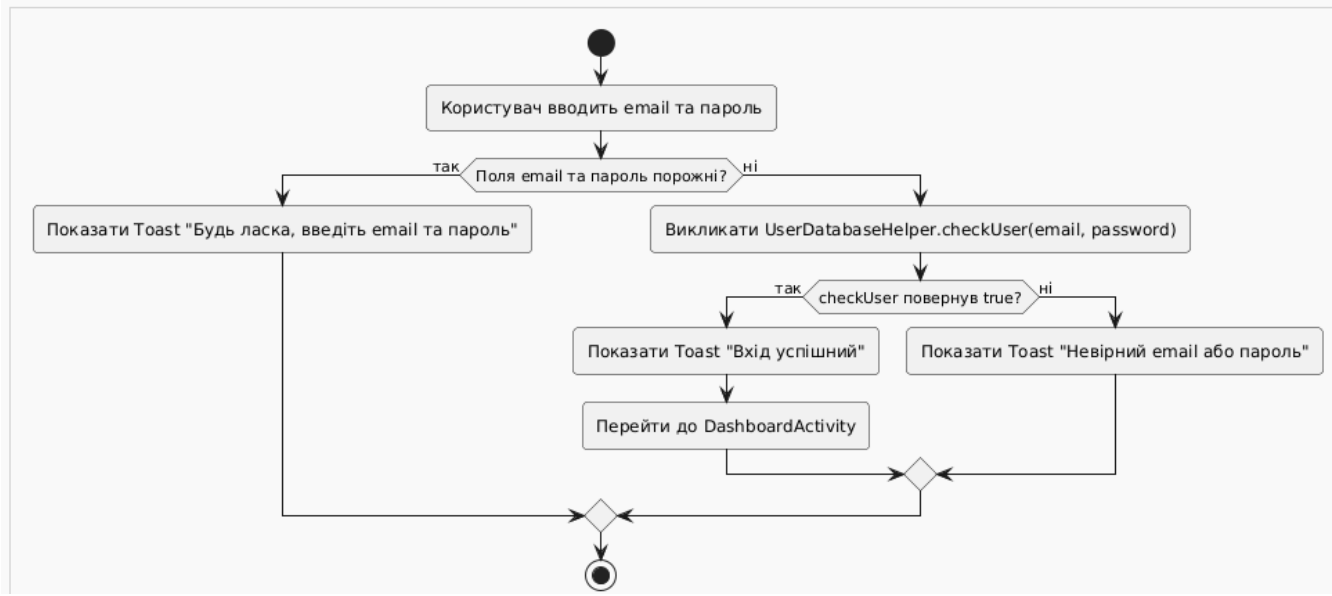


Рис. 16 Алгоритм авторизації користувача в системі

На рис. 17 зображено частину коду для авторизації користувача в системі.

```
val db = UserDatabaseHelper( context: this)
binding.registerButton.setOnClickListener {
    val email = binding.emailInput.text.toString().trim()
    val password = binding.passwordInput.text.toString().trim()

    if (email.isNotEmpty() && password.length >= 6) { // Перевірка довжини пароля
        if (!db.doesUserExist(email)) { // Перевірка, чи користувач вже існує
            val success = db.registerUser(email, password)
            if (success) {
                Toast.makeText( context: this,  text: "Реєстрація успішна", Toast.LENGTH_SHORT).show()
                val intent = Intent( packageContext: this, LoginActivity::class.java) // Повернення на LoginActivity
                startActivity(intent)
                finish() // Закрити RegisterActivity
            } else {
                Toast.makeText( context: this,  text: "Помилка при реєстрації. Спробуйте ще раз.", Toast.LENGTH_SHORT).show()
            }
        } else {
            Toast.makeText( context: this,  text: "Користувач з таким email вже існує.", Toast.LENGTH_SHORT).show()
        }
    } else {
        Toast.makeText( context: this,  text: "Будь ласка, введіть коректний email та пароль (мінімум 6 символів)", Toast.LENGTH_LONG).show()
    }
}
```

Рис. 17 Алгоритм авторизації користувача в системі

Ще одни прикладом є алгоритм для реалізації методу для додавання сценарію(рис. 18). Спочатку система отримує інформацію про сценарій і дані від користувача, після чого створює новий запис у базі даних із характеристиками).

Потім оновлює статус тестування та надсилає повідомлення про успішне додавання.

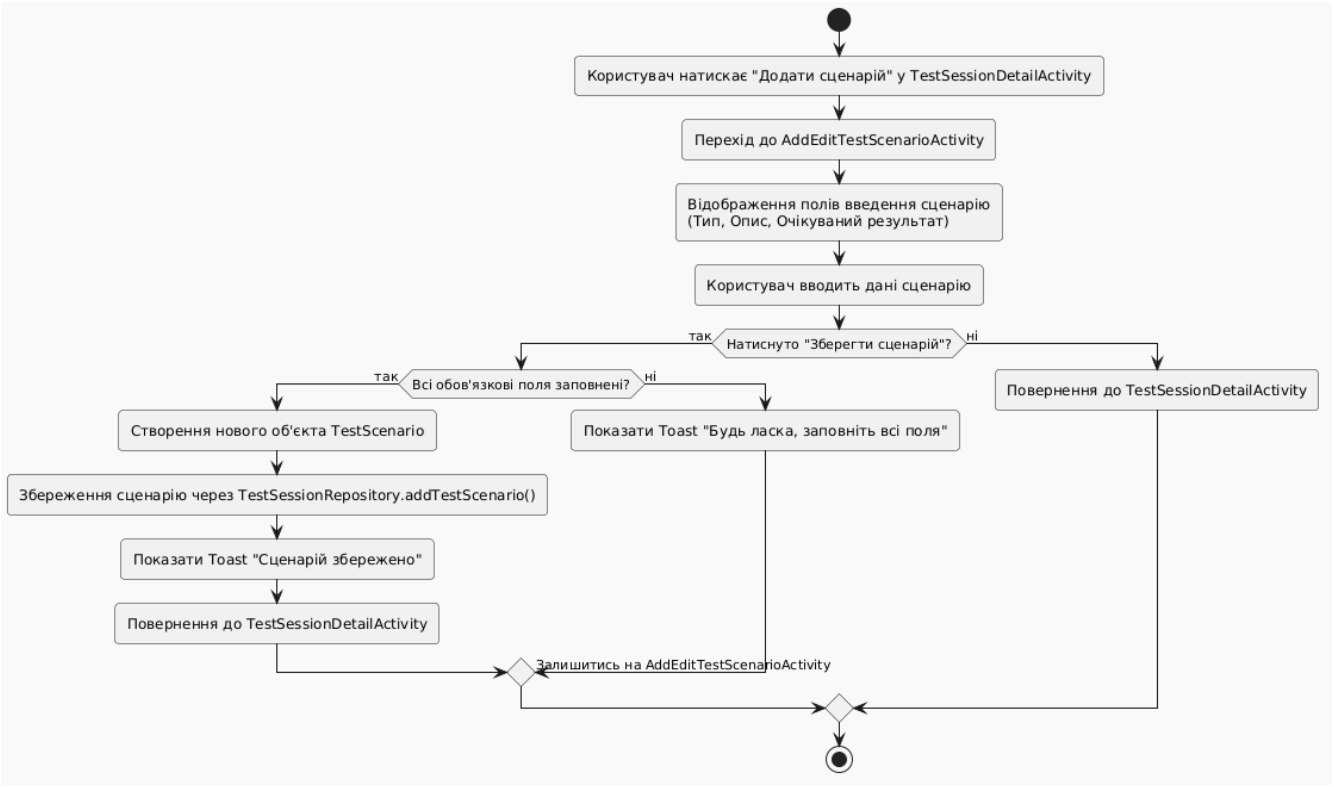
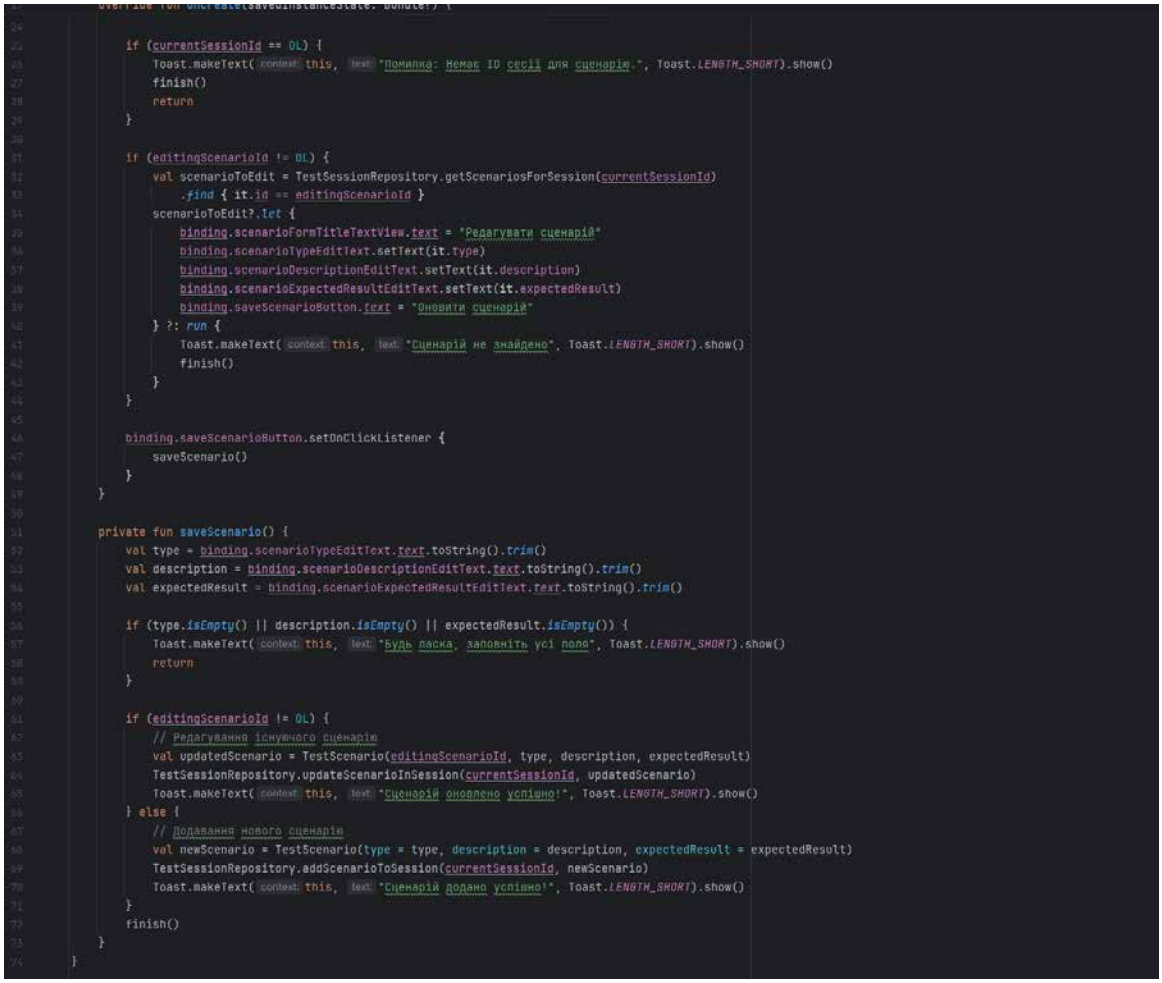


Рис. 18 Алгоритм додавання тестового польоту

На рис. 19 представлено частину коду для реалізації даного алгоритму.



```
        override fun onCreate(savedInstanceState: Bundle?) {

            if (currentSessionId == 0L) {
                Toast.makeText( context: this,  text: "Помилка: Немає ID сесії для сценарію.", Toast.LENGTH_SHORT).show()
                finish()
                return
            }

            if (editingScenarioId != 0L) {
                val scenarioToEdit = TestSessionRepository.getScenariosForSession(currentSessionId)
                    .find { it.id == editingScenarioId }
                scenarioToEdit?.let {
                    binding.scenarioFormTitleTextView.text = "Редагувати сценарій"
                    binding.scenarioTypeEditText.setText(it.type)
                    binding.scenarioDescriptionEditText.setText(it.description)
                    binding.scenarioExpectedResultEditText.setText(it.expectedResult)
                    binding.saveScenarioButton.text = "Оновити сценарій"
                } ?: run {
                    Toast.makeText( context: this,  text: "Сценарій не знайдено", Toast.LENGTH_SHORT).show()
                    finish()
                }
            }

            binding.saveScenarioButton.setOnClickListener {
                saveScenario()
            }
        }

        private fun saveScenario() {
            val type = binding.scenarioTypeEditText.text.toString().trim()
            val description = binding.scenarioDescriptionEditText.text.toString().trim()
            val expectedResult = binding.scenarioExpectedResultEditText.text.toString().trim()

            if (type.isEmpty() || description.isEmpty() || expectedResult.isEmpty()) {
                Toast.makeText( context: this,  text: "Будь ласка, заповніть усі поля", Toast.LENGTH_SHORT).show()
                return
            }

            if (editingScenarioId != 0L) {
                // Редагування існуючого сценарію
                val updatedScenario = TestScenario(editingScenarioId, type, description, expectedResult)
                TestSessionRepository.updateScenarioInSession(currentSessionId, updatedScenario)
                Toast.makeText( context: this,  text: "Сценарій оновлено успішно!", Toast.LENGTH_SHORT).show()
            } else {
                // Додавання нового сценарію
                val newScenario = TestScenario(type = type, description = description, expectedResult = expectedResult)
                TestSessionRepository.addScenarioToSession(currentSessionId, newScenario)
                Toast.makeText( context: this,  text: "Сценарій додано успішно!", Toast.LENGTH_SHORT).show()
            }
            finish()
        }
    }
```

Рис. 19 Частина функції додавання тестового польоту

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

4.1 Тестування системи

Тестування – це важлива складова процесу розробки програмного забезпечення. Воно допомагає перевірити коректність роботи системи та забезпечити високу якість готового програмного забезпечення. Існує багато видів тестування, які відрізняються підходами та методами, але всі вони спрямовані на перевірку функціональності, надійності та відповідності вимогам.

Серед основних видів тестування можна виділити:

1) функціональне тестування – це перевірка програми з точки зору виконання сценаріїв, воно тестує відповідність функціональним вимогам програмного забезпечення;

2) навантажувальне тестування – це оцінка роботи програми при великому навантаженні, воно визначає обсяг ресурсів, які витрачає програма при великому навантаженні;

3) перевірка на помилки (debugging) – це пошук, ідентифікація та виправлення помилок у програмі, які можуть призводити до некоректної роботи системи;

4) регресійне тестування – це перевірка програми, вже після внесення змін, щоб переконатись, що програмне забезпечення продовжує працювати правильно та не виникло нових дефектів;

5) автоматизоване тестування – це тестування за допомогою спеціальних інструментів та скриптів, щоб швидко та ефективно виконувати повторювані сценарії та скоротити час потрібний для тестування;

6) юніт-тестування – це тестування окремих компонентів та функцій програми за допомогою юніт-тестів, що перевірити коректність оброблювання значень;

7) верифікація та валідація – це перевірка відповідності програмного забезпечення вимогам та специфікаціям описаних в технічному завданні;

8) тестування безпеки – це тестування спрямоване на перевірку потенційних вразливостей програми та захисних механізмів, воно допомагає виявити можливі проблеми з безпекою даних;

9) тестування з використанням реальних даних – це тестування, що перевіряє коректність роботи системи за допомогою реальних даних;

10) тестування користувацького інтерфейсу – це перевірка реакції інтерфейсу програми на взаємодію користувача з системою, тобто перевіряється правильність відображення елементів, реакція на дії користувача, розташування елементів тощо;

11) тестування сумісності – це перевірка роботи програми на різних платформах, пристроях та операційних системах, щоб переконатись в коректності відображення всіх елементів та їх роботі;

12) тестування відмовостійкості – це перевірка поведінки програми про помилках, що можуть виникати.

Для тестування програмного забезпечення було вирішено використати два типи тестування: юніт тестування та тестування користувацького інтрефейсу, для перевірки коректності роботи як програмного коду, так і правильності побудови користувацького інтерфейсу.

Переваги юніт тестування:

1) виявлення помилок – юніт тести допомагають виявити помилки в коді ще на стадії розробки та швидко їх виправити;

2) зручна зміна коду – можна вносити зміни в код без страху непередбачуваних наслідків;

3) документація – юніт тести можна використовувати як документацію до програмного коду, описуючи очікувані результати;

4) покращення архітектури – для написання юніт-тестів є необхідною правильна структура коду та розбиття його на незалежні елементи, такі як методи.

Переваги тестування користувацького інтерфейсу:

1) впевненість в коректності роботи програму – можна виявити такі помилки як некоректне відображення графічних елементів, неправильне їх розташування, некоректні відповіді на запити користувача;

2) вдосконалення користувацького досвіду – дозволяє виявити можливі проблеми такі як неочікувані дії елементів, неправильні навігаційні шляхи тощо;

3) підвищення якості продукту – допомагає виявити проблеми з інтерфейсом перед релізом продукту, що дозволяє покращити якість програмного забезпечення.

На рис. 20 представлені юніт тести для методів авторизації користувача в системі, що були наведені на рис. 16.

Даний юніт тест перевіряє коректність роботи методу authenticateUser. Його можна поділити на три етапи виконання:

1) arrange – на цьому етапі йде підготовка даних до тесту, ми встановлюємо значення змінних «username» та «password» для створення валідних даних облікового запису;

2) act – на цьому етапі ми викликаємо наш метод, що ми тестуємо, передаємо в нього змінні з попереднього етапу, та зберігаємо результат;

3) assert – на цьому етапі ми перевіряємо, що значення результату є вірним, та вказує на успішну авторизацію користувача.

```
@Test
void testAuthenticateUser_ValidCredentials_ReturnsTrue() {
    // Arrange
    String username = "testuser";
    String password = "password123";

    // Act
    boolean result = authService.authenticateUser(username, password);

    // Assert
    assertTrue(result, "Користувач з правильними обліковими даними повинен пройти авторизацію");
}

@Test
void testAuthenticateUser_InvalidPassword_ReturnsFalse() {
    // Arrange
    String username = "testuser";
    String password = "wrongpassword";

    // Act
    boolean result = authService.authenticateUser(username, password);

    // Assert
    assertFalse(result, "Користувач з неправильним паролем не повинен пройти авторизацію");
}
```

Рис. 20 Юніт тест для методу authenticateUser

Тестування користувацього інтерфейсу. На рис. 21 зображено вікно входу в систему.

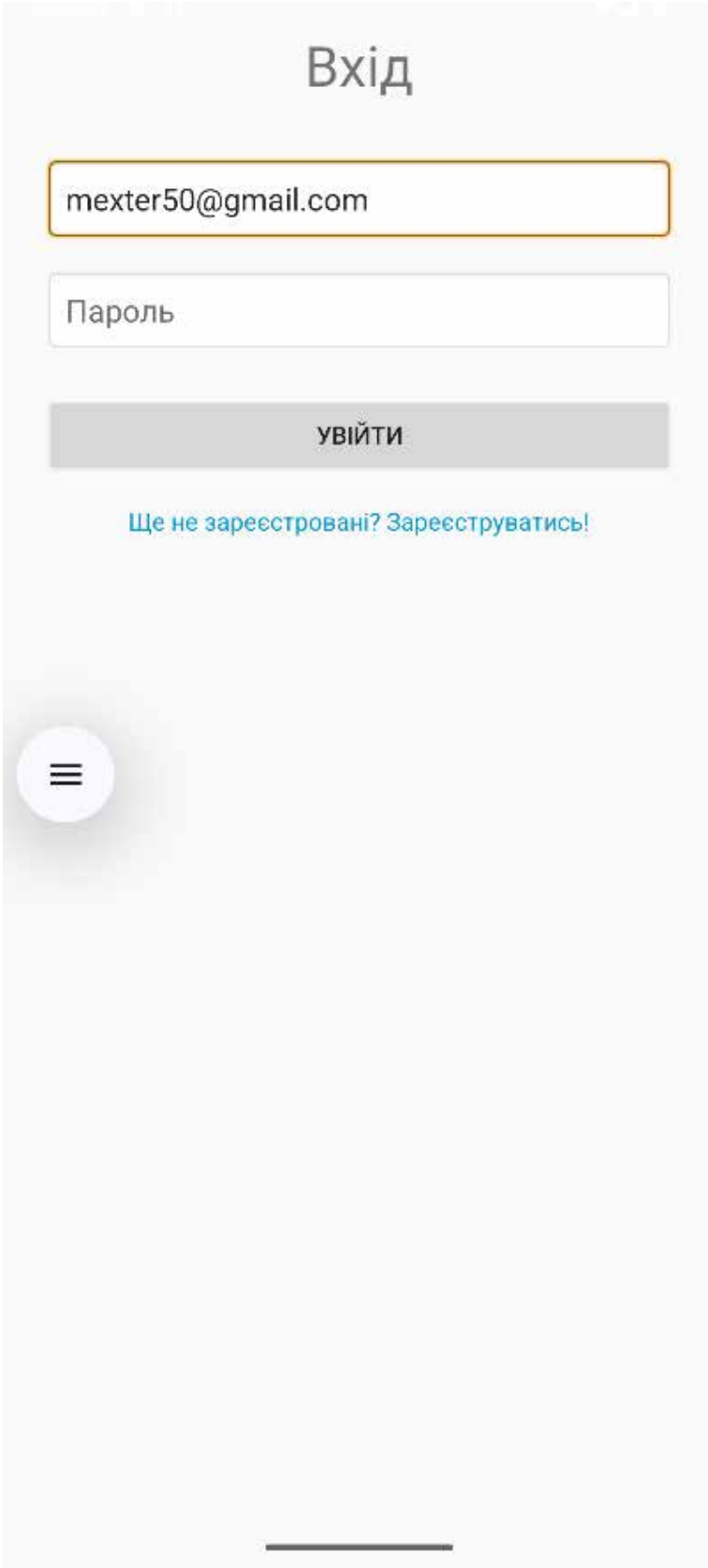


Рис. 21 Вікно входу до системи

При натисканні на кнопку зареєсруватися має відкриватись вікно авторизації (рис. 22).

Рис. 22 Вікно реєстрації

При натисканні на кнопку зареєструватися та введенні даних має відкриватись сторінку дашборду (рис. 23).

Рис. 23 Дрони

При переході в керування дронами має відкрити вікно з дронами (рис. 24).

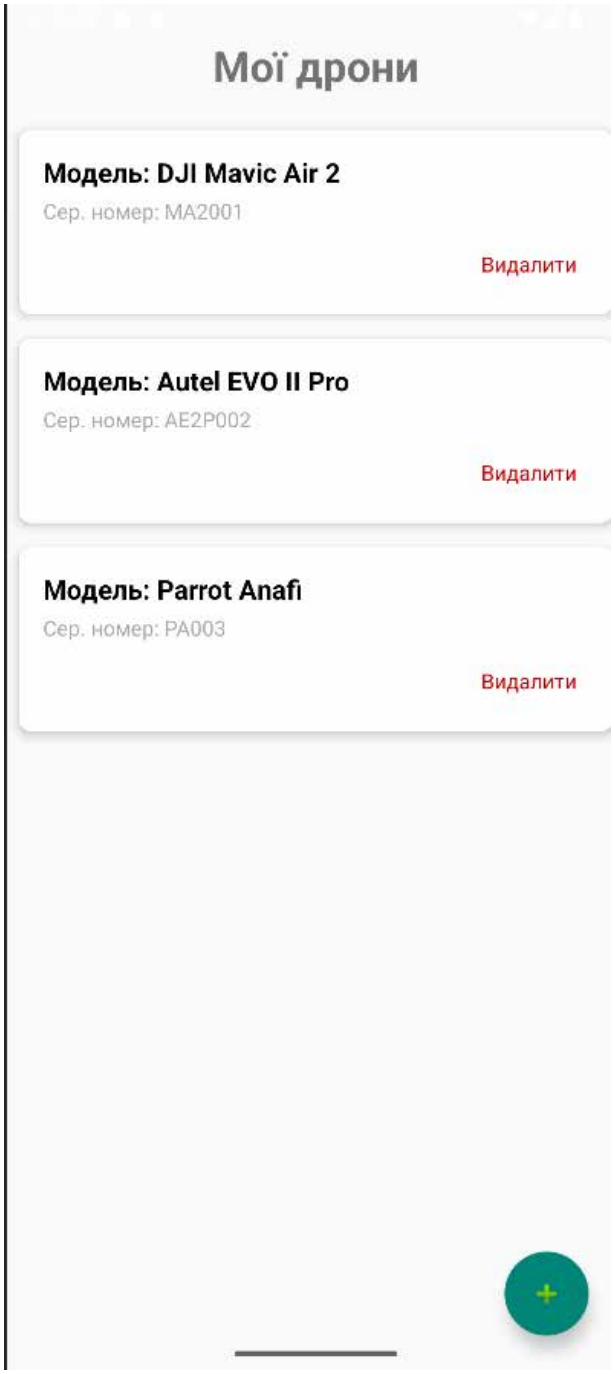


Рис. 24 Вікно дронів при видаленні

При видалені дрону він має зникати з списку (рис. 25)

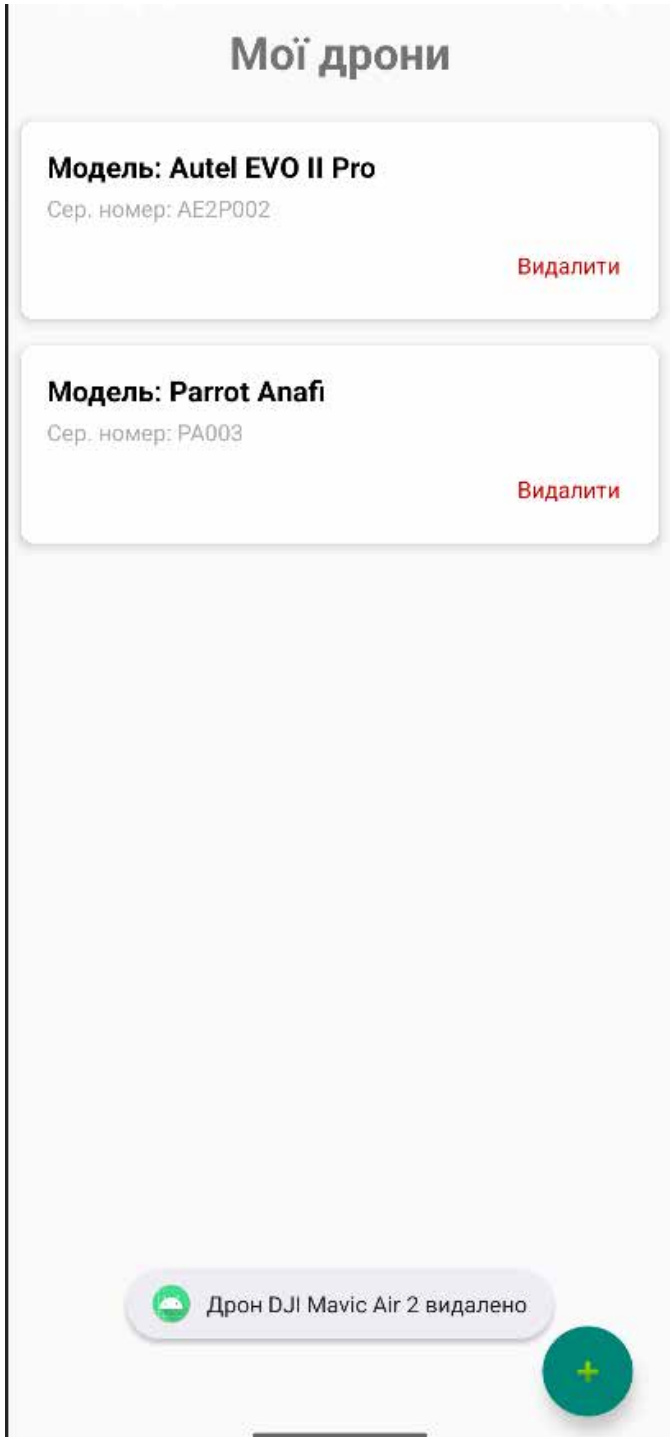


Рис. 25 Видалення

При натисканні на кнопку “+” має відкриватися форма додавання дрона(рис. 26).

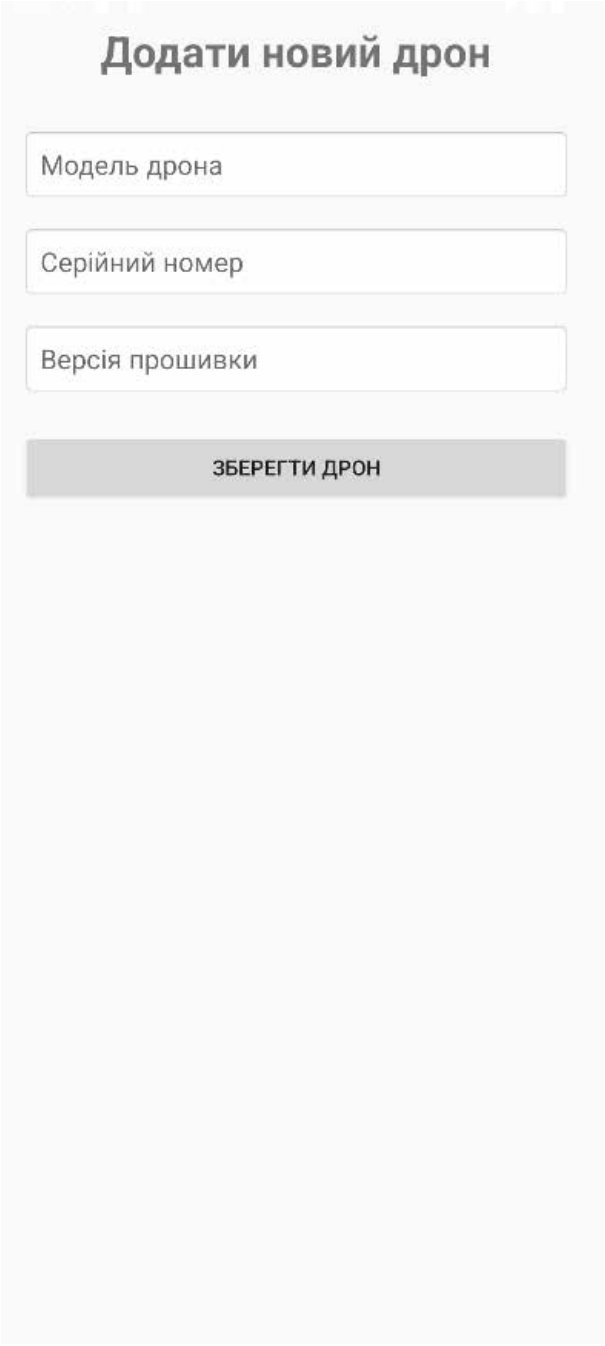


Рис. 26 Сесії, після видалення

При введенні та збереженні даних має в список має додаватися інший дрон(рис. 27).

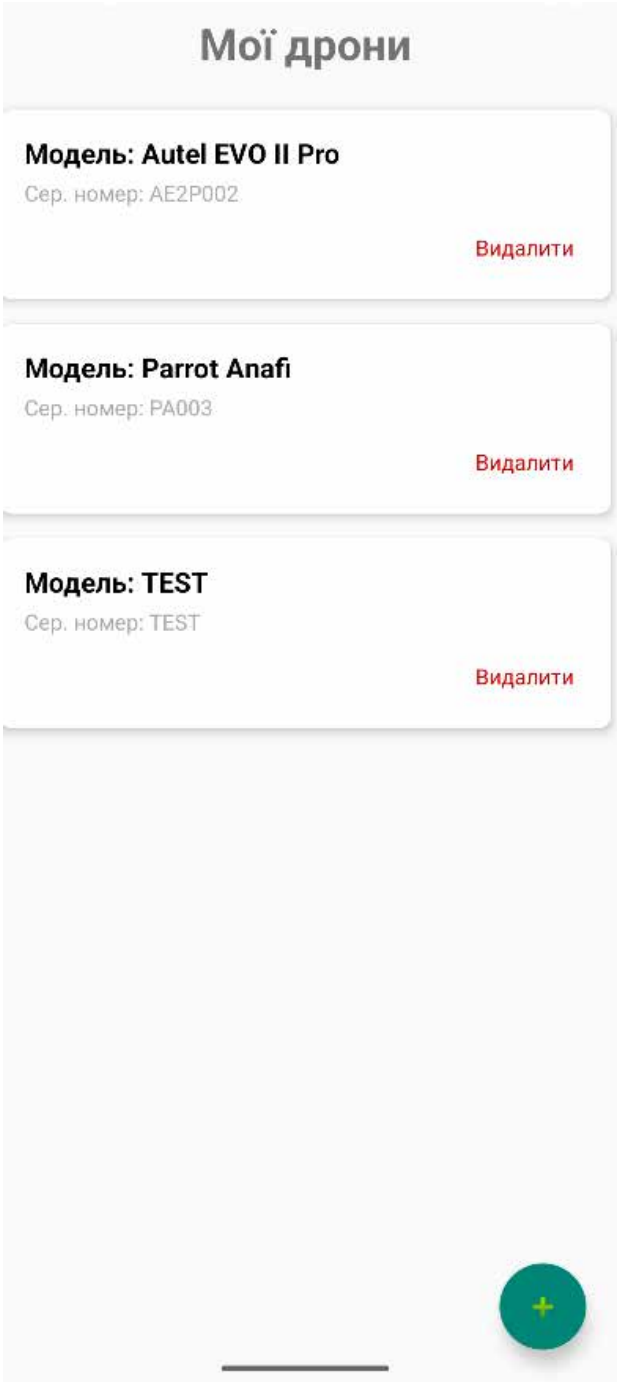


Рис. 27 Додавання дрону

При натисканні на дрон має відкриватися меню редагування (рис. 28).

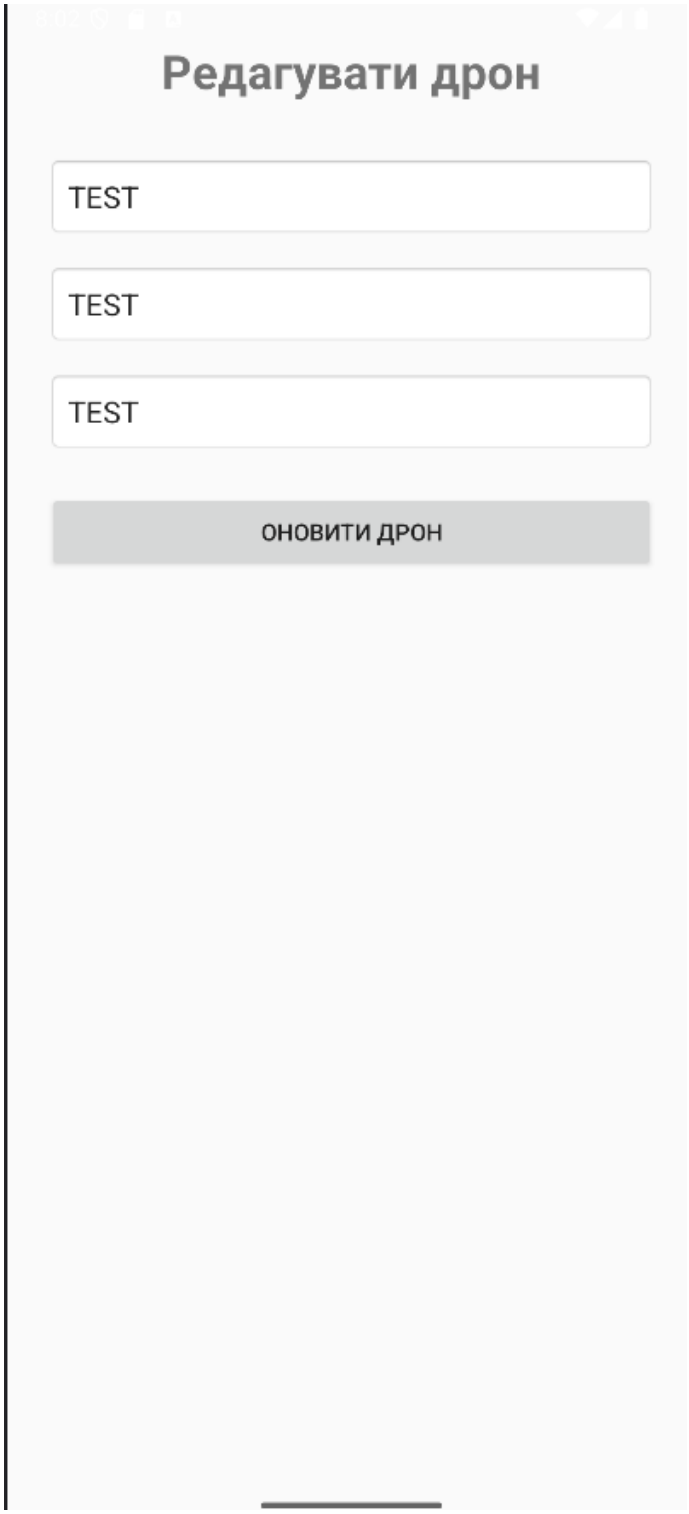


Рис. 28 Редагування

При натисканні на кнопку оновити дрон, має оновлюватись інформацію про нього (рис. 29).

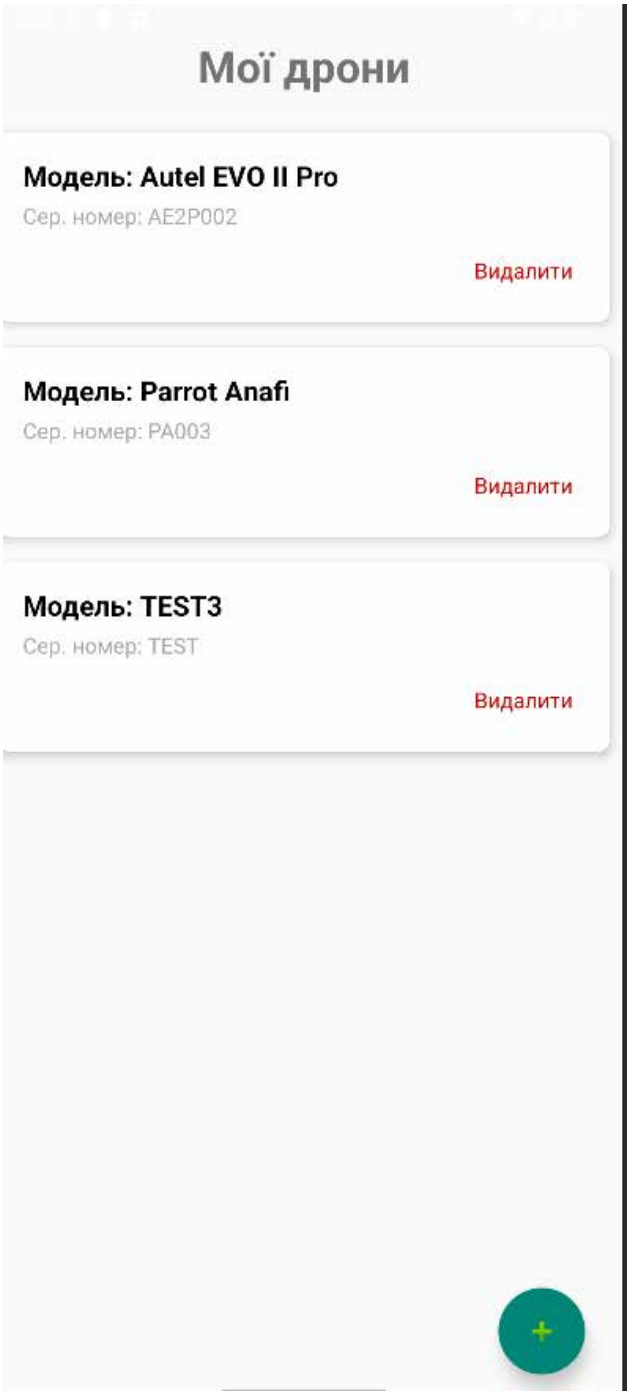


Рис. 29 Телеметрія

В сесіях має відображатися оновлення інформація про дрони(рис. 30).

Рис. 30 Сесії

При перевірці двома видами тестування не було виявлено ніяких помилок, компонент авторизації системи та елементи інтерфейсу працюють коректно.

4.2 Вимоги до апаратного та програмного забпезпечення

Для програмного забезпечення мобільного оргназайзера написаного мовою програмування Kotlin з базою даних SQLite є такі мінімальні вимоги:

1) Процесор – рекомендуються мінімум чотирьохядерний процес, для коректного виконання всіх функцій системи
2) Оперативна пам’ять – мінімум 5 гб
3) Операційна системи –Android
4) Версія операційної системи – Android 10
5) Роздільна здатність дисплею – мінімально 480x800 пікселів
6) Розмір екрану – мінімум 4 дюйми
7) Ємність батареї – мінімум 2000 мАгод
8) Витримка батареї – мінімум 8 годин

4.3 Склад інсталяційного пакету

На етапі проєктування для розуміння фізичної організації програмного забезпечення було спроєктовано діаграму розгортання. Діаграма розгортання – це діаграма для моделювання архітектури та фізичного розгортання системи. Вона надає графічне представлення компонентів системи, фізичних вузлів та зв’язків між ними. [46]

Рис. 41 Діаграма розгортання

На рис. 33 представлена діаграма розгортання для даної системи. Вона є простою і складається лише з двох компонентів, це додаток та локальна база даних.

Тому інсталяційний пакет програмного забезпечення мобільного застосунку для тестування дронів буде складатись з одного файлу – Dronecontrol.apk.

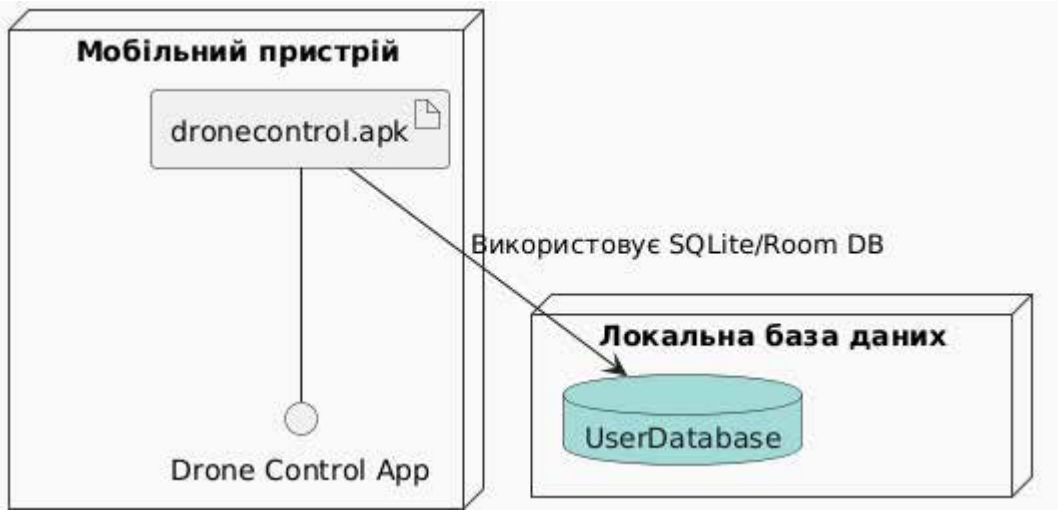


Рис. 31

ВИСНОВКИ

У ході виконання бакалаврської кваліфікаційної роботи на тему «Мобільний додаток для тестування безпілотників» було здійснено комплексний аналіз предметної області, включно з процесами тестування безпілотних апаратів та методами контролю їх технічного стану.

Було сформовано функціональні та нефункціональні вимоги до системи, що лягли в основу моделювання діаграм прецедентів, активності та послідовності для детального опису бізнес-логіки.

Проведено аналіз існуючих рішень, як апаратних, так і програмних, що дозволило чітко окреслити основні завдання системи та описати вхідні й вихідні дані.

Розроблено інформаційну базу у вигляді логічної моделі даних із застосуванням діаграм класів, компонентів та пакетів, що забезпечує зручну структуру зберігання та обробки інформації про тестові сесії, безпілотники, сценарії тестування, події та замітки.

Обґрунтовано вибір технологій (SQLite, Kotlin, Android Studio) для розробки мобільного додатку, що реалізує функціонал системи. Створено базу даних, розроблено алгоритми роботи програмного забезпечення та їх реалізацію.

Проведено тестування програмного продукту за допомогою юніт-тестів та інтерфейсного тестування, що підтвердило відповідність системи заданим вимогам.

Результатом роботи стало створення мобільного програмного забезпечення для тестування безпілотників, що забезпечує ефективне управління тестовими завданнями, моніторингом стану безпілотних апаратів, веденням заміток. Система спрямована на підвищення якості тестування та підтримку операторів у плануванні та контролі тестових польотів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) ArduPilot Dev Team. SITL (Software in the Loop) Simulator. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html
2) Android Developers. Android Studio User Guide. https://developer.android.com/studio/intro
3) SQLite. SQLite Home Page. https://www.sqlite.org/index.html
4) GitHub – PhilJay. MPAndroidChart: A powerful Android chart view / graph view library. https://github.com/PhilJay/MPAndroidChart
5) M. Hassanalian, A. Abdelkefi. Classifications, applications, and design challenges of drones: https://doi.org/10.1016/j.paerosci.2017.04.003
6) booch G., Rumbaugh J., Jacobson I. *The Unified Modeling Language User Guide*. Addison-Wesley
7) OMG (Object Management Group). *UML 2.5 Specification*. https://www.omg.org/spec/UML/2.5/
8) Fowler M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3rd ed. Addison-Wesley
9) Scott W. Ambler. *The Elements of UML 2.0 Style*. Cambridge University Press

ДОДАТОК А

Інтерфейси

Сторінок – 36

2025

AddEditDroneActivity.kt

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.dronecontrolapp.data.DroneRepository
import com.example.dronecontrolapp.data.model.Drone
import com.example.dronecontrolapp.databinding.ActivityAddEditDroneBinding

class AddEditDroneActivity : AppCompatActivity() {

    private lateinit var binding: ActivityAddEditDroneBinding
    private var editingDroneId: Long = 0L // ID дрона, якщо ми редагуємо

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityAddEditDroneBinding.inflate(layoutInflater)
        setContentView(binding.root)

        editingDroneId = intent.getLongExtra("drone_id", 0L)
        if (editingDroneId != 0L) {
            val droneToEdit = DroneRepository.getDroneById(editingDroneId)
            droneToEdit?.let {
                binding.formTitleTextView.text = "Редагувати дрон"
                binding.modelEditText.setText(it.model)
                binding.serialNumberEditText.setText(it.serialNumber)
                binding.firmwareVersionEditText.setText(it.firmwareVersion)
                binding.saveDroneButton.text = "Оновити дрон"
```

```
        } ?: run {
            Toast.makeText(this, "Дрон не знайдено", Toast.LENGTH_SHORT).show()
            finish()
        }
    }

    binding.saveDroneButton.setOnClickListener {
        saveDrone()
    }
}

private fun saveDrone() {
    val model = binding.modelEditText.text.toString().trim()
    val serialNumber = binding.serialNumberEditText.text.toString().trim()
    val firmwareVersion = binding.firmwareVersionEditText.text.toString().trim()

    if (model.isEmpty() || serialNumber.isEmpty() || firmwareVersion.isEmpty()) {
        Toast.makeText(this, "Будь ласка, заповніть усі поля", Toast.LENGTH_SHORT).show()
        return
    }

    if (editingDroneId != 0L) {
        // Редагування існуючого дрона
        val updatedDrone = Drone(editingDroneId, model, serialNumber, firmwareVersion)
        DroneRepository.updateDrone(updatedDrone)
        Toast.makeText(this, "Дрон оновлено успішно!", Toast.LENGTH_SHORT).show()
    } else {
```

```
            // Додавання нового дрона
            val newDrone = Drone(model = model, serialNumber = serialNumber, firmwareVersion
= firmwareVersion)
            DroneRepository.addDrone(newDrone)
            Toast.makeText(this, "Дрон додано успішно!", Toast.LENGTH_SHORT).show()
        }
        finish() // Закрити активність після збереження
    }
}
```

AddEditTestScenarioActivity

```
package com.example.dronecontrolapp.ui

import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.dronecontrolapp.data.TestSessionRepository
import com.example.dronecontrolapp.data.model.TestScenario
import com.example.dronecontrolapp.databinding.ActivityAddEditTestScenarioBinding

class AddEditTestScenarioActivity : AppCompatActivity() {

    private lateinit var binding: ActivityAddEditTestScenarioBinding
    private var currentSessionId: Long = 0L
    private var editingScenarioId: Long = 0L

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        binding = ActivityAddEditTestScenarioBinding.inflate(layoutInflater)
        setContentView(binding.root)

        currentSessionId = intent.getLongExtra("session_id", 0L)
        editingScenarioId = intent.getLongExtra("scenario_id", 0L)

        if (currentSessionId == 0L) {
            Toast.makeText(this, "Помилка: Немає ID сесії для сценарію.",
Toast.LENGTH_SHORT).show()
            finish()
            return
        }

        if (editingScenarioId != 0L) {
            val scenarioToEdit = TestSessionRepository.getScenariosForSession(currentSessionId)
                .find { it.id == editingScenarioId }
            scenarioToEdit?.let {
                binding.scenarioFormTitleTextView.text = "Редагувати сценарій"
                binding.scenarioTypeEditText.setText(it.type)
                binding.scenarioDescriptionEditText.setText(it.description)
                binding.scenarioExpectedResultEditText.setText(it.expectedResult)
                binding.saveScenarioButton.text = "Оновити сценарій"
            } ?: run {
                Toast.makeText(this, "Сценарій не знайдено", Toast.LENGTH_SHORT).show()
                finish()
            }
        }
```

```
        binding.saveScenarioButton.setOnClickListener {
            saveScenario()
        }
    }

    private fun saveScenario() {
        val type = binding.scenarioTypeEditText.text.toString().trim()
        val description = binding.scenarioDescriptionEditText.text.toString().trim()
        val expectedResult = binding.scenarioExpectedResultEditText.text.toString().trim()

        if (type.isEmpty() || description.isEmpty() || expectedResult.isEmpty()) {
            Toast.makeText(this, "Будь ласка, заповніть усі поля", Toast.LENGTH_SHORT).show()
            return
        }

        if (editingScenarioId != 0L) {
            // Редагування існуючого сценарію
            val updatedScenario = TestScenario(editingScenarioId, type, description, expectedResult)
            TestSessionRepository.updateScenarioInSession(currentSessionId, updatedScenario)
            Toast.makeText(this, "Сценарій оновлено успішно!", Toast.LENGTH_SHORT).show()
        } else {
            // Додавання нового сценарію
            val newScenario = TestScenario(type = type, description = description, expectedResult =
expectedResult)
            TestSessionRepository.addScenarioToSession(currentSessionId, newScenario)
            Toast.makeText(this, "Сценарій додано успішно!", Toast.LENGTH_SHORT).show()
```

```
        }
        finish()
    }
}
```

AddEditTestSessionActivity

```
package com.example.dronecontrolapp.ui

import android.app.DatePickerDialog
import android.app.TimePickerDialog
import android.os.Bundle
import android.view.View
import android.widget.AdapterView
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.dronecontrolapp.data.DroneRepository
import com.example.dronecontrolapp.data.TestSessionRepository
import com.example.dronecontrolapp.data.model.Drone
import com.example.dronecontrolapp.data.model.TestSession
import com.example.dronecontrolapp.databinding.ActivityAddEditTestSessionBinding
import java.text.SimpleDateFormat
import java.util.Calendar
import java.util.Date
import java.util.Locale

class AddEditTestSessionActivity : AppCompatActivity() {
```

```
private lateinit var binding: ActivityAddEditTestSessionBinding
private var selectedDroneId: Long = 0L
private var selectedDate: Calendar = Calendar.getInstance()
private var editingSessionId: Long = 0L // ID сесії, якщо ми редагуємо

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityAddEditTestSessionBinding.inflate(layoutInflater)
    setContentView(binding.root)

    setupDroneSpinner()
    setupDateTimePickers()

    editingSessionId = intent.getLongExtra("session_id", 0L)
    if (editingSessionId != 0L) {
        val sessionToEdit = TestSessionRepository.getTestSessionById(editingSessionId)
        sessionToEdit?.let { session ->
            binding.sessionFormTitleTextView.text = "Редагувати сесію"
            binding.descriptionEditText.setText(session.description)
            selectedDate.time = session.date
            binding.saveSessionButton.text = "Оновити сесію"

            // Встановлення вибраного дрона у Spinner
            val drones = DroneRepository.getAllDrones()
            val droneIndex = drones.indexOfFirst { it.id == session.droneId }
            if (droneIndex != -1) {
                binding.droneSpinner.setSelection(droneIndex)
```

```
            }
        } ?: run {
            Toast.makeText(this, "Сесію не знайдено", Toast.LENGTH_SHORT).show()
            finish()
        }
    } else {
        // Для нової сесії встановлюємо час на 1 годину вперед від поточного
        selectedDate.add(Calendar.HOUR_OF_DAY, 1)
    }
    updateDateTimeButtons() // Оновлюємо текст на кнопках при старті

    binding.saveSessionButton.setOnClickListener {
        saveTestSession()
    }
}

private fun setupDroneSpinner() {
    val drones = DroneRepository.getAllDrones()
    val droneNames = drones.map { "${it.model} (${it.serialNumber})" }

    val adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item, droneNames)
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
    binding.droneSpinner.adapter = adapter

    binding.droneSpinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
        override fun onItemSelected(parent: AdapterView<*>, view: View?, position: Int, id:
```

```
Long) {
            selectedDroneId = drones[position].id
        }

        override fun onNothingSelected(parent: AdapterView<*>) {
            selectedDroneId = 0L // Або обробляти випадок, коли нічого не вибрано
        }
    }
}

private fun setupDateTimePickers() {
    binding.selectDateButton.setOnClickListener {
        DatePickerDialog(
            this,
            { _, year, month, dayOfMonth ->
                selectedDate.set(year, month, dayOfMonth)
                updateDateTimeButtons()
            },
            selectedDate.get(Calendar.YEAR),
            selectedDate.get(Calendar.MONTH),
            selectedDate.get(Calendar.DAY_OF_MONTH)
        ).show()
    }

    binding.selectTimeButton.setOnClickListener {
        TimePickerDialog(
            this,
```

```
            { _, hourOfDay, minute ->
                selectedDate.set(Calendar.HOUR_OF_DAY, hourOfDay)
                selectedDate.set(Calendar.MINUTE, minute)
                updateDateTimeButtons()
            },
            selectedDate.get(Calendar.HOUR_OF_DAY),
            selectedDate.get(Calendar.MINUTE),
            true // 24-годинний формат
        ).show()
    }
}

private fun updateDateTimeButtons() {
    val dateFormat = SimpleDateFormat("dd.MM.yyyy", Locale.getDefault())
    val timeFormat = SimpleDateFormat("HH:mm", Locale.getDefault())
    binding.selectDateButton.text = dateFormat.format(selectedDate.time)
    binding.selectTimeButton.text = timeFormat.format(selectedDate.time)
}

private fun saveTestSession() {
    val description = binding.descriptionEditText.text.toString().trim()

    if (selectedDroneId == 0L) {
        Toast.makeText(this, "Будь ласка, виберіть дрон", Toast.LENGTH_SHORT).show()
        return
    }
    if (description.isEmpty()) {
```

```
            Toast.makeText(this, "Будь ласка, введіть опис сесії", Toast.LENGTH_SHORT).show()
            return
        }

        val sessionDate = selectedDate.time
        val userId = 1L // Заглушка: ID користувача, можна пізніше реалізувати отримання реального ID

        if (editingSessionId != 0L) {
            // Редагування існуючої сесії
            val updatedSession = TestSession(editingSessionId, selectedDroneId, userId, sessionDate, description)
            TestSessionRepository.updateTestSession(updatedSession)
            Toast.makeText(this, "Сесію оновлено успішно!", Toast.LENGTH_SHORT).show()
        } else {
            // Додавання нової сесії
            val newSession = TestSession(droneId = selectedDroneId, userId = userId, date = sessionDate, description = description)
            TestSessionRepository.addTestSession(newSession)
            Toast.makeText(this, "Сесію додано успішно!", Toast.LENGTH_SHORT).show()
        }
        finish()
    }
}
```

DashboardActivity

```
package com.example.dronecontrolapp.ui
```

```
import android.content.Intent
import android.graphics.Color
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.dronecontrol.database.UserDatabaseHelper
import com.example.dronecontrolapp.data.DroneRepository // Додайте цей імпорт
import com.example.dronecontrolapp.data.TelemetryRepository // Додайте цей імпорт
import com.example.dronecontrolapp.databinding.ActivityDashboardBinding

class DashboardActivity : AppCompatActivity() {

    private lateinit var binding: ActivityDashboardBinding
    private var isDroneConnected = false
    private val handler = Handler(Looper.getMainLooper())

    private var simulatedConnectedDroneId: Long = 0L

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityDashboardBinding.inflate(layoutInflater)
        setContentView(binding.root)

        updateDroneStatusUI()
```

```
        binding.connectDroneButton.setOnClickListener {
            if (!isDroneConnected) {
                simulateDroneConnection()
            } else {
                Toast.makeText(this, "Дрон вже підключений", Toast.LENGTH_SHORT).show()
            }
        }

        binding.disconnectDroneButton.setOnClickListener {
            if (isDroneConnected) {
                simulateDroneDisconnection()
            } else {
                Toast.makeText(this, "Дрон вже відключений", Toast.LENGTH_SHORT).show()
            }
        }

        binding.logoutButton.setOnClickListener {
            val intent = Intent(this, LoginActivity::class.java)
            intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
            startActivity(intent)
            finish()
        }
        binding.manageDronesButton.setOnClickListener {
            val intent = Intent(this, DroneListActivity::class.java)
            startActivity(intent)
        }
```

```
        binding.manageTestSessionsButton.setOnClickListener {
            val intent = Intent(this, TestSessionListActivity::class.java)
            startActivity(intent)
        }

        binding.simulateFlightButton.setOnClickListener {
            if (isDroneConnected && simulatedConnectedDroneId != 0L) {
                startFlightSimulation(simulatedConnectedDroneId)
            } else {
                Toast.makeText(this, "Будь ласка, підключіть дрон перед симуляцією польоту",
Toast.LENGTH_SHORT).show()
            }
        }

        binding.viewFlightReportsButton.setOnClickListener {
            if (simulatedConnectedDroneId != 0L) { // Можна переглядати звіти, навіть якщо дрон
відключений
                val intent = Intent(this, FlightReportActivity::class.java).apply {
                    putExtra("drone_id", simulatedConnectedDroneId)
                }
                startActivity(intent)
            } else {
                Toast.makeText(this, "Спочатку підключіть дрон або виконайте симуляцію для
перегляду звітів", Toast.LENGTH_SHORT).show()
            }
        }
    }
```

```
    private fun simulateDroneConnection() {
        binding.droneStatusTextView.text = "Статус дрона: Підключення..."
        binding.droneStatusTextView.setTextColor(Color.parseColor("#FFA500"))
        binding.connectDroneButton.isEnabled = false
        binding.disconnectDroneButton.isEnabled = false
        binding.simulateFlightButton.isEnabled = false // Вимкнути під час підключення

        handler.postDelayed({
            isDroneConnected = true
            val availableDrones = DroneRepository.getAllDrones()
            if (availableDrones.isNotEmpty()) {
                simulatedConnectedDroneId = availableDrones.first().id
                binding.droneStatusTextView.text = "Статус дрона: Підключено до
${availableDrones.first().model}"
                Toast.makeText(this, "Дрон ${availableDrones.first().model} успішно підключено!",
Toast.LENGTH_SHORT).show()
            } else {
                simulatedConnectedDroneId = 0L
                binding.droneStatusTextView.text = "Статус дрона: Підключено (без дронів у
списку)"
                Toast.makeText(this, "Дрон успішно підключено, але немає доступних дронів для
симуляції.", Toast.LENGTH_LONG).show()
            }
            updateDroneStatusUI()
        }, 3000)
    }
```

```
    private fun simulateDroneDisconnection() {
        binding.droneStatusTextView.text = "Статус дрона: Відключення..."
        binding.droneStatusTextView.setTextColor(Color.parseColor("#FFA500"))
        binding.connectDroneButton.isEnabled = false
        binding.disconnectDroneButton.isEnabled = false
        binding.simulateFlightButton.isEnabled = false
        handler.postDelayed({
            isDroneConnected = false
            updateDroneStatusUI()
            Toast.makeText(this, "Дрон відключено.", Toast.LENGTH_SHORT).show()
        }, 2000)
    }

    private fun updateDroneStatusUI() {
        if (isDroneConnected) {
            val drone = DroneRepository.getDroneById(simulatedConnectedDroneId)
            binding.droneStatusTextView.text = "Статус дрона: Підключено до ${drone?.model ?:
"Невідомий"}"
            binding.droneStatusTextView.setTextColor(Color.GREEN)
            binding.connectDroneButton.isEnabled = false
            binding.disconnectDroneButton.isEnabled = true
            binding.simulateFlightButton.isEnabled = true
            binding.viewFlightReportsButton.isEnabled = true
        } else {
            binding.droneStatusTextView.text = "Статус дрона: Відключено"
            binding.droneStatusTextView.setTextColor(Color.RED)
```

```
            binding.connectDroneButton.isEnabled = true
            binding.disconnectDroneButton.isEnabled = false
            binding.simulateFlightButton.isEnabled = false
            binding.viewFlightReportsButton.isEnabled = (simulatedConnectedDroneId != 0L)
        }
    }

    private fun startFlightSimulation(droneId: Long) {
        Toast.makeText(this, "Початок симуляції польоту для дрона ID: $droneId",
Toast.LENGTH_SHORT).show()

        TelemetryRepository.generateRandomTelemetry(droneId, 50.4501, 30.5234, 50) // Київські
координати, 50 точок

        Toast.makeText(this, "Симуляція польоту завершена. Згенеровано дані телеметрії.",
Toast.LENGTH_LONG).show()

        val intent = Intent(this, FlightReportActivity::class.java).apply {
            putExtra("drone_id", droneId)
        }
        startActivity(intent)
    }
}
```

DroneListActivity

```
package com.example.dronecontrolapp.ui

import android.content.Intent
```

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.dronecontrolapp.data.DroneRepository
import com.example.dronecontrolapp.data.model.Drone
import com.example.dronecontrolapp.databinding.ActivityDroneListBinding
import com.example.dronecontrolapp.ui.adapter.DroneAdapter

class DroneListActivity : AppCompatActivity() {

    private lateinit var binding: ActivityDroneListBinding
    private lateinit var droneAdapter: DroneAdapter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityDroneListBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setupRecyclerView()

        binding.addDroneFab.setOnClickListener {
            val intent = Intent(this, AddEditDroneActivity::class.java)
            startActivity(intent)
        }
    }
```

```
    override fun onResume() {
        super.onResume()
        droneAdapter.updateDrones(DroneRepository.getAllDrones())
    }

    private fun setupRecyclerView() {
        droneAdapter = DroneAdapter(
            drones = DroneRepository.getAllDrones(),
            onDroneClick = { drone ->
                val intent = Intent(this, AddEditDroneActivity::class.java).apply {
                    putExtra("drone_id", drone.id) // Передаємо ID дрона
                }
                startActivity(intent)
            },
            onDeleteClick = { drone ->
                DroneRepository.deleteDrone(drone.id)
                droneAdapter.updateDrones(DroneRepository.getAllDrones())
                Toast.makeText(this, "Дрон ${drone.model} видалено",
Toast.LENGTH_SHORT).show()
            }
        )
        binding.dronesRecyclerView.apply {
            layoutManager = LinearLayoutManager(this@DroneListActivity)
            adapter = droneAdapter
        }
    }
}
```

FlightReportActivity

```
package com.example.dronecontrolapp.ui

import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.dronecontrolapp.data.DroneRepository
import com.example.dronecontrolapp.data.TelemetryRepository
import com.example.dronecontrolapp.databinding.ActivityFlightReportBinding
import com.example.dronecontrolapp.ui.adapter.TelemetryDataAdapter
import java.text.SimpleDateFormat
import java.util.Locale

class FlightReportActivity : AppCompatActivity() {

    private lateinit var binding: ActivityFlightReportBinding
    private lateinit var telemetryAdapter: TelemetryDataAdapter
    private var droneId: Long = 0L

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityFlightReportBinding.inflate(layoutInflater)
        setContentView(binding.root)

        droneId = intent.getLongExtra("drone_id", 0L)
        if (droneId == 0L) {
```

```
            Toast.makeText(this, "Немає ID дрона для звіту.", Toast.LENGTH_SHORT).show()
            finish()
            return
        }

        val drone = DroneRepository.getDroneById(droneId)
        binding.reportTitleTextView.text = "Звіт польотів для ${drone?.model ?: "Невідомий дрон"}"
        binding.droneInfoTextView.text = "Серійний номер: ${drone?.serialNumber ?: "N/A"}"

        setupRecyclerView()
        loadFlightData()
    }

    private fun setupRecyclerView() {
        telemetryAdapter = TelemetryDataAdapter(emptyList())
        binding.telemetryRecyclerView.apply {
            layoutManager = LinearLayoutManager(this@FlightReportActivity)
            adapter = telemetryAdapter
        }
    }

    private fun loadFlightData() {
        val telemetryData = TelemetryRepository.getTelemetryForDrone(droneId)
        if (telemetryData.isNotEmpty()) {
            telemetryAdapter.updateData(telemetryData)
            // Можна додати зведену статистику тут
```

```
            val totalPoints = telemetryData.size
            val maxAltitude = telemetryData.maxOfOrNull { it.altitude }
            val avgSpeed = telemetryData.map { it.speed }.average()
            val minBattery = telemetryData.minOfOrNull { it.batteryLevel }

            val statsText = """
                Кількість записів: $totalPoints
                Макс. висота: ${"%.2f".format(maxAltitude)} м
                Середня швидкість: ${"%.2f".format(avgSpeed)} м/с
                Мін. рівень батареї: $minBattery %
            """.trimIndent()
            binding.flightStatsTextView.text = statsText
        } else {
            binding.flightStatsTextView.text = "Даних польоту немає."
            Toast.makeText(this, "Для цього дрона немає згенерованих даних польоту.",
Toast.LENGTH_SHORT).show()
        }
    }
}
```

LoginActivity

```
package com.example.dronecontrolapp.ui

import android.content.Intent
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.dronecontrol.database.UserDatabaseHelper
```

```
import com.example.dronecontrolapp.databinding.ActivityLoginBinding

class LoginActivity : AppCompatActivity() {

    private lateinit var binding: ActivityLoginBinding /

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val db = UserDatabaseHelper(this)
        binding.loginButton.setOnClickListener {
            val email = binding.emailInput.text.toString().trim()
            val password = binding.passwordInput.text.toString().trim() /

            if (email.isNotEmpty() && password.isNotEmpty()) {
                if (db.checkUser(email, password)) {
                    Toast.makeText(this, "Вхід успішний", Toast.LENGTH_SHORT).show()
                    val intent = Intent(this, DashboardActivity::class.java) // Перехід на
DashboardActivity
                    startActivity(intent)
                    finish()
                } else {
                    Toast.makeText(this, "Невірний email або пароль",
Toast.LENGTH_SHORT).show()
                }
```

```
            } else {
                Toast.makeText(this, "Будь ласка, введіть email та пароль",
Toast.LENGTH_SHORT).show()
            }
        }

        binding.registerTextView.setOnClickListener {
            val intent = Intent(this, RegisterActivity::class.java) // Перехід на RegisterActivity
            startActivity(intent)
        }
    }
}
package com.example.dronecontrolapp.ui

import android.content.Intent
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.dronecontrol.database.UserDatabaseHelper
import com.example.dronecontrolapp.databinding.ActivityLoginBinding

class LoginActivity : AppCompatActivity() {

    private lateinit var binding: ActivityLoginBinding /

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val db = UserDatabaseHelper(this)
        binding.loginButton.setOnClickListener {
            val email = binding.emailInput.text.toString().trim()
            val password = binding.passwordInput.text.toString().trim() /

            if (email.isNotEmpty() && password.isNotEmpty()) {
                if (db.checkUser(email, password)) {
                    Toast.makeText(this, "Вхід успішний", Toast.LENGTH_SHORT).show()
                    val intent = Intent(this, DashboardActivity::class.java) // Перехід на
DashboardActivity
                    startActivity(intent)
                    finish()
                } else {
                    Toast.makeText(this, "Невірний email або пароль",
Toast.LENGTH_SHORT).show()
                }
            } else {
                Toast.makeText(this, "Будь ласка, введіть email та пароль",
Toast.LENGTH_SHORT).show()
            }
        }

        binding.registerTextView.setOnClickListener {
            val intent = Intent(this, RegisterActivity::class.java) // Перехід на RegisterActivity
```

```
            startActivity(intent)
        }
    }
}
```

MainActivity

```
package com.example.dronecontrolapp.ui

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Перенаправлення на LoginActivity при запуску програми
        val intent = Intent(this, LoginActivity::class.java)
        startActivity(intent)
        finish()
    }
}
```

RegisterActivity

```
package com.example.dronecontrolapp.ui

import android.content.Intent
import android.os.Bundle
import android.widget.Toast
```

```
import androidx.appcompat.app.AppCompatActivity
import com.example.dronecontrol.database.UserDatabaseHelper
import com.example.dronecontrolapp.databinding.ActivityRegisterBinding

class RegisterActivity : AppCompatActivity() {

    private lateinit var binding: ActivityRegisterBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityRegisterBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val db = UserDatabaseHelper(this)
        binding.registerButton.setOnClickListener {
            val email = binding.emailInput.text.toString().trim()
            val password = binding.passwordInput.text.toString().trim()

            if (email.isNotEmpty() && password.length >= 6) { // Перевірка довжини пароля
                if (!db.doesUserExist(email)) { // Перевірка, чи користувач вже існує
                    val success = db.registerUser(email, password)
                    if (success) {
                        Toast.makeText(this, "Реєстрація успішна", Toast.LENGTH_SHORT).show()
                        val intent = Intent(this, LoginActivity::class.java)
                        startActivity(intent)
                        finish()
                    } else {
```

```
                    Toast.makeText(this, "Помилка при реєстрації. Спробуйте ще раз.",
Toast.LENGTH_SHORT).show()
                }
            } else {
                Toast.makeText(this, "Користувач з таким email вже існує.",
Toast.LENGTH_SHORT).show()
            }
        } else {
            Toast.makeText(this, "Будь ласка, введіть коректний email та пароль (мінімум 6
символів)", Toast.LENGTH_LONG).show()
        }
    }
  }
}
```

TestSessionDetailActivity

```
package com.example.dronecontrolapp.ui

import android.content.Intent
import android.graphics.Color
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.dronecontrolapp.data.DroneRepository
import com.example.dronecontrolapp.data.TestSessionRepository
import com.example.dronecontrolapp.data.model.ScenarioStatus
import com.example.dronecontrolapp.data.model.TestSession
```

```
import com.example.dronecontrolapp.databinding.ActivityTestSessionDetailBinding
import com.example.dronecontrolapp.ui.adapter.TestScenarioAdapter
import java.text.SimpleDateFormat
import java.util.Locale
import android.os.Handler
import android.os.Looper

class TestSessionDetailActivity : AppCompatActivity() {

    private lateinit var binding: ActivityTestSessionDetailBinding
    private var currentSessionId: Long = 0L
    private lateinit var scenarioAdapter: TestScenarioAdapter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityTestSessionDetailBinding.inflate(layoutInflater)
        setContentView(binding.root)

        currentSessionId = intent.getLongExtra("session_id", 0L)
        if (currentSessionId == 0L) {
            Toast.makeText(this, "Немає ID сесії", Toast.LENGTH_SHORT).show()
            finish()
            return
        }

        setupScenarioRecyclerView()
```

```
        binding.addScenarioFab.setOnClickListener {
            val intent = Intent(this, AddEditTestScenarioActivity::class.java).apply {
                putExtra("session_id", currentSessionId)
            }
            startActivity(intent)
        }
    }

    override fun onResume() {
        super.onResume()
        loadSessionDetails()

scenarioAdapter.updateScenarios(TestSessionRepository.getScenariosForSession(currentSession
Id))
    }

    private fun loadSessionDetails() {
        val session = TestSessionRepository.getTestSessionById(currentSessionId)
        session?.let {
            val drone = DroneRepository.getDroneById(it.droneId)
            val dateFormat = SimpleDateFormat("dd.MM.yyyy HH:mm", Locale.getDefault())

            binding.detailDescriptionTextView.text = "Опис: ${it.description}"
            binding.detailDroneInfoTextView.text = "Дрон: ${drone?.model ?: "Невідомий"}
(${drone?.serialNumber ?: ""})"
            binding.detailDateTextView.text = "Дата: ${dateFormat.format(it.date)}"
```

```
        if (it.isCompleted) {
            binding.detailStatusTextView.text = "Статус: Завершено"
            binding.detailStatusTextView.setTextColor(Color.GREEN)
        } else {
            binding.detailStatusTextView.text = "Статус: Незавершено"
            binding.detailStatusTextView.setTextColor(Color.RED)
        }
    } ?: run {
        Toast.makeText(this, "Сесію не знайдено", Toast.LENGTH_SHORT).show()
        finish()
    }
}

private fun setupScenarioRecyclerView() {
    scenarioAdapter = TestScenarioAdapter(
        scenarios = TestSessionRepository.getScenariosForSession(currentSessionId),
        onScenarioClick = { scenario ->
            // Перехід до редагування сценарію
            val intent = Intent(this, AddEditTestScenarioActivity::class.java).apply {
                putExtra("session_id", currentSessionId)
                putExtra("scenario_id", scenario.id)
            }
            startActivity(intent)
        },
        onDeleteClick = { scenario ->
            TestSessionRepository.deleteScenarioFromSession(currentSessionId, scenario.id)
```

```
scenarioAdapter.updateScenarios(TestSessionRepository.getScenariosForSession(currentSession
Id))
            Toast.makeText(this, "Сценарій '${scenario.type}' видалено",
Toast.LENGTH_SHORT).show()
        },
        onExecuteClick = { scenario ->
            // Імітація виконання сценарію
            scenario.status = ScenarioStatus.IN_PROGRESS
            TestSessionRepository.updateScenarioInSession(currentSessionId, scenario)

scenarioAdapter.updateScenarios(TestSessionRepository.getScenariosForSession(currentSession
Id))
            Toast.makeText(this, "Виконання сценарію '${scenario.type}'...",
Toast.LENGTH_SHORT).show()

            // Імітація завершення через 2 секунди
            Handler(Looper.getMainLooper()).postDelayed({
                val randomResult = listOf(ScenarioStatus.COMPLETED_SUCCESS,
ScenarioStatus.COMPLETED_FAILURE).random()
                scenario.status = randomResult
                scenario.actualResult = if (randomResult ==
ScenarioStatus.COMPLETED_SUCCESS) "Успішно виконано" else "Виконано з помилкою"
                TestSessionRepository.updateScenarioInSession(currentSessionId, scenario)

scenarioAdapter.updateScenarios(TestSessionRepository.getScenariosForSession(currentSession
Id))
                Toast.makeText(this, "Сценарій '${scenario.type}' ${if (randomResult ==
```

```
ScenarioStatus.COMPLETED_SUCCESS) "успішно" else "з помилкою"} завершено",
Toast.LENGTH_SHORT).show()
                }, 2000)
            },
            onCancelClick = { scenario ->
                scenario.status = ScenarioStatus.CANCELLED
                TestSessionRepository.updateScenarioInSession(currentSessionId, scenario)

scenarioAdapter.updateScenarios(TestSessionRepository.getScenariosForSession(currentSession
Id))
                Toast.makeText(this, "Сценарій '${scenario.type}' скасовано",
Toast.LENGTH_SHORT).show()
            }
        )
        binding.scenariosRecyclerView.apply {
            layoutManager = LinearLayoutManager(this@TestSessionDetailActivity)
            adapter = scenarioAdapter
        }
    }
}
```

TestSessionListActivity

```
package com.example.dronecontrolapp.ui

import android.content.Intent
import android.os.Bundle
```

```
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.dronecontrolapp.data.TestSessionRepository
import com.example.dronecontrolapp.databinding.ActivityTestSessionListBinding
import com.example.dronecontrolapp.ui.adapter.TestSessionAdapter

class TestSessionListActivity : AppCompatActivity() {

    private lateinit var binding: ActivityTestSessionListBinding
    private lateinit var sessionAdapter: TestSessionAdapter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityTestSessionListBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setupRecyclerView()

        binding.addTestSessionFab.setOnClickListener {
            val intent = Intent(this, AddEditTestSessionActivity::class.java)
            startActivity(intent)
        }
    }

    override fun onResume() {
        super.onResume()
```

```
        sessionAdapter.updateTestSessions(TestSessionRepository.getAllTestSessions())
    }

    private fun setupRecyclerView() {
        sessionAdapter = TestSessionAdapter(
            testSessions = TestSessionRepository.getAllTestSessions(),
            onSessionClick = { session ->
                val intent = Intent(this, TestSessionDetailActivity::class.java).apply {
                    putExtra("session_id", session.id)
                }
                startActivity(intent)
            },
            onDeleteClick = { session ->
                TestSessionRepository.deleteTestSession(session.id)

sessionAdapter.updateTestSessions(TestSessionRepository.getAllTestSessions())
                Toast.makeText(this, "Сесію '${session.description}' видалено",
Toast.LENGTH_SHORT).show()
            },
            onMarkCompleteClick = { session, isCompleted ->
                TestSessionRepository.markSessionCompleted(session.id, isCompleted)

sessionAdapter.updateTestSessions(TestSessionRepository.getAllTestSessions())
                val statusText = if (isCompleted) "завершено" else "незавершено"
                Toast.makeText(this, "Сесію '${session.description}' позначено як
$statusText", Toast.LENGTH_SHORT).show()
            }
```

```
        )
        binding.testSessionsRecyclerView.apply {
            layoutManager = LinearLayoutManager(this@TestSessionListActivity)
            adapter = sessionAdapter
        }
    }
}
```