НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
_____ Голуб Б. Л.

«___»_____2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ІНТЕЛЕКТУАЛЬНИЙ ПОМІЧНИК ПЛАНУВАННЯ РУХУ ТЕХІНКИ ПРИ ВИКОНАННІ АГРООПЕРАЦІЙ »

Спеціальність 122 «Комп'ютерні науки»

Гарант освітньої програми
Д. е. н., професор _____ Руденський Р. А.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Боровик В. І.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Камінник Д. О.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
_____ Голуб Б. Л.

«___»_____2025 р.

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

КАМІННИКУ ДМИТРУ ОЛЕКСАНДРОВИЧУ

Спеціальність 122 «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи Інтелектуальний помічник планування руху техніки при виконанні агрооперацій
затверджена наказом ректора НУБіП України від «16» 12 2024 р. № 2246 "С"

Термін подання завершеної роботи на кафедру 2025.06.02
(рік, місяць, число)

Вихідні дані до бакалаврської кваліфікаційної роботи:
опис програмного забезпечення
Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір засобів для розробки системи.
3. Розробка інформаційної системи.
4. Висновок

Дата видачі завдання «___» _____ 2025 р.
Керівник бакалаврської кваліфікаційної роботи

_____ _____ Боровик В. І.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Камінник Д. О.
(підпис) (ПІБ студента)

Календарний план

№ з/п	Назва етапів виконання бакалаврської кваліфікаційної роботи	Строк виконання етапів бакалаврської кваліфікаційної роботи	Примітка
1	Аналіз предметної області	18.02.2025	
2	Реалізація структури бази даних	12.03.2025	
3	Проектування програмного забезпечення	02.04.2025	
4	Проектування та реалізація інтерфейсу користувача	15.04.2025	
5	Реалізація інтерфейсу з базою даних	23.04.2025	
6	Розробка алгоритму плану руху техніки	10.05.2025	
7	Впровадження програмної системи	15.05.2025	
8	Оформлення пояснювальної записки	22.05.2025	

Студент _____ Камінник Д. О.
(підпис) (ПІБ студента)

Керівник бакалаврської кваліфікаційної роботи

_____ Боровик В. І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У роботі представлено розробку мобільного Android-додатку, що виступає як інтелектуальний помічник для планування руху сільськогосподарської техніки при виконанні агрооперацій.

Актуальність дослідження зумовлена потребою аграрного сектору в цифрових рішеннях для підвищення ефективності та зменшення витрат на виконання польових робіт.

У процесі реалізації проєкту здійснено системний аналіз предметної області, моделювання основних сутностей, розробку логічної та фізичної моделі бази даних. Для реалізації застосовано стек технологій Kotlin, Jetpack Compose, Firebase (Authentication, Firestore), Mapbox SDK. Архітектура додатку побудована на основі підходу MVVM і принципів Clean Architecture. Особливу увагу приділено реалізації алгоритму побудови маршруту руху техніки з урахуванням параметрів поля, типу агрооперації та технічних характеристик обладнання. Додаток дозволяє генерувати кілька варіантів маршруту, проводити їх аналітичне порівняння та працювати в офлайн-режимі.

Отримані результати свідчать про доцільність впровадження системи AgroPlan у сільськогосподарське виробництво малого та середнього масштабу, з можливістю її масштабування і подальшого розвитку функціоналу.

ABSTRACT

This thesis presents the development of a mobile Android application that acts as an intelligent assistant for planning the movement of agricultural machinery during agricultural operations.

The relevance of the study is driven by the agricultural sector's need for digital solutions to increase efficiency and reduce the cost of field work.

In the course of the project implementation, a systematic analysis of the subject area, modelling of the main entities, development of a logical and physical database model were carried out. The project was implemented using the Kotlin technology stack, Jetpack Compose, Firebase (Authentication, Firestore), and Mapbox SDK. The application architecture is based on the MVVM approach and Clean Architecture principles.Particular attention is paid to the implementation of the algorithm for building a route for the movement of equipment, taking into account the parameters of the field, the type of agricultural operation and the technical characteristics of the equipment. The application allows generating several route options, conducting their analytical comparison and working in offline mode.

The results obtained indicate the feasibility of implementing the AgroPlan system in small and medium-sized agricultural production, with the possibility of scaling it up and further developing its functionality.

ЗМІСТ

ВСТУП

У сучасному світі цифрові технології охоплюють дедалі більше сфер людської діяльності, включаючи аграрну галузь. Сільське господарство впроваджує інноваційні ІТ-рішення з метою підвищення ефективності використання ресурсів, зниження витрат та покращення якості виконання агроопераций. Одним із пріоритетних напрямів є оптимізація руху сільськогосподарської техніки під час виконання операцій у полі.

Актуальність теми пояснюється тим, що траєкторії руху техніки часто формуються вручну, без врахування просторових характеристик поля, типу агроопераций та технічних параметрів обладнання. Це призводить до перевитрат пального, зайвого зносу техніки та втрат продуктивного часу. Існує потреба у створенні програмного рішення, яке дозволяє формувати ефективні маршрути автоматизовано — з урахуванням карти поля та заданих параметрів.

Метою даної бакалаврської кваліфікаційної роботи є розробка мобільного Android-додатку AgroPlan, який виступає як інтелектуальний помічник у плануванні руху сільськогосподарської техніки при виконанні агроопераций.

Для досягнення цієї мети необхідно реалізувати наступні функціональні можливості:

- авторизацію та реєстрацію користувачів з використанням Firebase Authentication;
- додавання інформації про поля, побудову їх контурів за допомогою Mapbox SDK;
- введення технічних характеристик агротехніки;
- створення та перегляд плану маршруту руху техніки;
- побудову декількох варіантів маршруту за різними критеріями
- порівняння ефективності реалізованих планів;
- підтримку офлайн-режиму перегляду.

Розробка додатку базується на сучасних підходах до Android-розробки: використання Kotlin, Jetpack Compose, Firebase та Mapbox. Проєкт реалізовано із дотриманням архітектурної моделі MVVM та принципів Clean Architecture.

Структура пояснювальної записки охоплює:

- опис предметної області та постановку задачі;
- аналіз аналогів і обґрунтування вибору інструментів;
- проектування логічної та фізичної моделі даних;
- реалізацію структури додатку та алгоритмів побудови маршруту;
- вимоги до впровадження та тестування;
- висновки щодо отриманих результатів.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка завдання

Сучасне сільське господарство перебуває на етапі активної цифрової трансформації, що охоплює такі напрями як точне землеробство, аграрна аналітика, автоматизоване управління технікою та застосування геоінформаційних технологій (ГІС). Одним з найбільш витратних і водночас критичних процесів є виконання агроопераций — сівби, обприскування, обробки ґрунту, збирання врожаю. Ці операції передбачають систематичне пересування техніки по полю, і точність планування її руху напряму впливає на витрати пального, час виконання, знос агрегатів та врожайність.

На практиці агрооперації здебільшого плануються вручну — оператор, агроном або інженер самостійно оцінює форму поля, вказує напрям руху, обирає відправну точку і метод покриття території. Такий підхід не враховує важливих факторів:

- **неоптимальний напрям руху** призводить до надлишкових поворотів, перекриттів або неохоплених зон;
- **ігнорування технічних характеристик** (ширини захвату, швидкості, витрати пального) призводить до неправильного розрахунку кількості проходів;
- **неврахування форми поля** — особливо при неправильних контурах або складній геометрії — ускладнює ручне планування;
- **відсутність порівняння варіантів** — оператор не бачить, чи можна виконати роботу краще за іншим маршрутом.

У результаті підприємства несуть втрати через перевитрати пального, збільшення навантаження на техніку, неефективне використання часу і зниження точності обробки поля. Особливо це критично для великих господарств, де площа оброблюваних полів сягає десятків і сотень гектарів. Для вирішення цієї проблеми пропонується розробка мобільного Android-додатку AgroPlan, що виступатиме в ролі інтелектуального помічника. Додаток дозволяє автоматизувати ключові етапи:

- **Створення електронної карти поля**: користувач відображає контур на карті вручну за допомогою позначення країв поля.

- **Введення параметрів техніки**: ширина захвату, швидкість пересування, витрата пального на 100 кілометрів.
- **Обрання типу агрооперації**: додаток адаптує логіку побудови маршруту залежно від обраного типу.
- **Генерація маршруту**: алгоритм формує траєкторію покриття з урахуванням параметрів поля і техніки.
- **Порівняння декількох варіантів**: за критерієм мінімальної кількості проходів, найменших витрат пального, найкоротшого часу виконання.
- **Збереження та перегляд у офлайн-режимі**: що є критично важливим для роботи у польових умовах.
- Цільовими користувачами системи є:
- **фермери та керівники господарств**, які хочуть скоротити витрати і підвищити ефективність;
- **агрономи**, які відповідають за планування робіт і мають потребу у простому візуальному інструменті;
- **оператори техніки**, які отримують зрозумілий маршрут для виконання агрооперацій.

Завдяки використанню сучасного стеку технологій — Firebase Firestore, Mapbox SDK, Jetpack Compose, Kotlin — реалізація рішення забезпечує як масштабованість, так і простоту використання, а також підтримку офлайн-режиму, що є важливою перевагою для агросектору.

1.2 Огляд інформаційних джерел та існуючих рішень

1.2.1 Загальна характеристика напрямів розвитку IT в агросфері

У сучасному аграрному виробництві зростає потреба в цифрових рішеннях, що забезпечують точність, ефективність та зниження витрат. Основними напрямами цифрової трансформації є: точне землеробство (precision farming), застосування IoT-пристроїв, аграрна аналітика, супутниковий моніторинг і автоматизація техніки.

Ці напрями об’єднує потреба в зборі та обробці великого обсягу геопросторових і технічних даних у реальному часі, що, своєю чергою, вимагає впровадження інтелектуальних систем керування та ефективних мобільних платформ для польових умов.

1.2.2 Аналіз існуючих рішень

У процесі дослідження предметної області було виявлено, що напрямок цифрової трансформації аграрного сектору активно розвивається, зокрема у сфері точного землеробства (precision agriculture) та інтелектуальних систем керування технікою.

Крім того, було проведено аналіз існуючих рішень, що представлені як мобільні додатки, так і повноцінні хмарні платформи для керування технікою. Деякі з найпоширеніших прикладів:

Trimble Ag Software — орієнтований на великі агрохолдинги та пропонує комплексний набір функцій: трекінг техніки, планування обробітку, моніторинг урожайності. Основний недолік — складність налаштування та висока вартість для малого фермера.

FieldBee — мобільне рішення з функціоналом автопілота, GPS-навігації та запису пройдених маршрутів. Незважаючи на популярність серед фермерів, продукт не дозволяє автоматично формувати маршрут залежно від параметрів поля та техніки.

John Deere Operations Center — потужне корпоративне рішення з широкою підтримкою техніки John Deere, повною інтеграцією з GPS, телеметрією, картами. Основна обмеженість — прив’язаність до техніки конкретного виробника.

Порівняння існуючих сервісів зображено у таблиці 1

Таблиця 1

Назва	Платформа	Потрібне Обладнання	Підтримка карт	Порівняння планів	Офлайн	Ціна
John Deere Operations Center	Web, мобільна	Так	Google Maps, внутрішній ГІС	Так	Частково	Висока
Trimble Ag Software	Web, мобільна	Так	Власні карти, Shape-файли	Так	Ні	Висока
AgroPilot	Android	Частково	Leaflet / OSM	Ні	Так	Середня
AgroPlan (розробка)	Android	Ні	Mapbox + GeoJSON	Так	Так	Безкоштовно

Попри наявність вищенаведених систем, на ринку також існує низка мобільних або локальних додатків, які пропонують часткову автоматизацію польових операцій, але не охоплюють повний цикл планування з урахуванням характеристик поля та техніки. Наприклад:

- **AgriBus-NAVI** — японський додаток для навігації тракторів із підтримкою паралельного водіння. Простий у використанні, але не має функцій аналітики та побудови маршрутів за заданими параметрами.
- **eFarmer** — доступне рішення з базовими можливостями GPS-навігації, карти полів, журналу операцій. Орієнтоване переважно на ручне введення й не генерує маршрути автоматично.
- **Cropio (Syngenta)** — хмарна платформа з широким спектром функцій для аналітики, моніторингу техніки та супутникового нагляду. Вимагає складної інтеграції й найчастіше застосовується великими агрокомпаніями.

Ці приклади демонструють, що більшість рішень мають вузьку спеціалізацію, обмежені можливості конфігурації планів або не забезпечують гнучкого, інтуїтивного інтерфейсу для польових умов.

Таким чином, можна зробити висновок, що більшість наявних рішень або:

- орієнтовані на великих виробників техніки й мають обмежену сумісність;
- не підтримують гнучке створення планів руху техніки на основі параметрів поля та типу агрооперації;
- не надають користувачеві інструментів перебудови плану залежно від критеріїв.

Отже, створення власного мобільного додатку з функціональністю генерації, перегляду, порівняння та оптимізації планів дає змогу заповнити наявну нішу серед рішень для малого та середнього агровиробника, що не має доступу до дорогих корпоративних платформ.

1.2.3 Обґрунтування вибору технологій

Для реалізації додатку було проаналізовано низку технологічних стеків. Вибір зупинився на:

- **Jetpack Compose** — новий декларативний підхід до побудови UI в Android, який дозволяє ефективно керувати станом інтерфейсу, швидко прототипувати і забезпечує просту підтримку тем, навігації та адаптації до

різних розмірів екрану. Compose має підтримку LiveData, ViewModel, Koin — що ідеально підходить для архітектури MVVM.

- **Firebase Firestore** — хмарна NoSQL база даних, яка забезпечує зберігання і синхронізацію даних у реальному часі. Вона підтримує вкладені колекції, дозволяє працювати в офлайн-режимі та автоматично синхронізує дані після відновлення зв'язку. Це ідеальний варіант для польових умов.
- **Firebase Authentication** — забезпечує безпечну та зручну авторизацію користувачів за допомогою email-паролю або Google Sign-In.
- **Mapbox SDK** — гнучкий інструмент для роботи з картографічними даними. Має підтримку GeoJSON-формату, можливість кастомізації шарів, побудову маршрутів, візуалізацію полів, напрямків руху та зон покриття.
- **GeoJSON** — відкритий формат для представлення геопросторових даних. Використання цього формату дозволяє:

1. зберігати межі полів та маршрути;
2. легко інтегруватися з агротехнікою, яка підтримує завантаження карт;
3. експортувати або передавати дані до сторонніх систем.

1.3 Моделювання предметної області

Моделювання предметної області — це важливий етап проєктування програмної системи, який дозволяє сформулювати функціональні та інформаційні вимоги до системи, відобразити логіку взаємодії користувачів з інтерфейсом, визначити структуру даних та взаємозв'язки між елементами. У процесі розробки додатку AgroPlan було реалізовано повний цикл моделювання: від аналізу предметної області до формування структурних та поведінкових моделей за допомогою UML-діаграм.

1.3.1 Ключові сутності предметної області

Основу інформаційної структури AgroPlan становлять такі сутності:

- **Користувач (User)** — містить дані про ім'я, електронну пошту, тип підписки. Дозволяє створювати власні поля, техніку та плани агроопераций.

- **Поле (Field)** — містить назву, геометрію у форматі GeoJSON (список координат), культуру. Прив’язане до користувача.
- **Техніка (Machinery)** — характеризується назвою, типом, швидкістю, шириною захвату, витратою палива. Зберігається у вкладеній колекції користувача.
- **Операція (Operation)** — відображає тип агрооперації (сівба, обробка, обприскування, жнива) та описує її специфіку.
- **План (Plan)** — зберігає маршрут у вигляді списку координат, ID техніки та операції, обчислені метрики: площа, кількість проходів, довжина маршруту, витрата пального тощо.
- **Аналітика (Analytics)** — використовується для збереження результатів порівняння альтернативних планів за різними критеріями (найкоротший маршрут, мінімум поворотів, найменша витрата часу або пального).

Структура бази даних реалізована у Firestore з вкладеними колекціями: техніка та поля належать конкретному користувачу, а всередині кожного поля зберігаються плани і аналітика. Такий підхід забезпечує логічну ізоляцію та швидкий доступ до потрібної інформації без зайвих запитів.

1.3.2 Взаємозв’язки між сутностями

У системі реалізовано такі логічні зв’язки:

1. один користувач може мати багато полів і техніки;
2. кожне поле може містити багато планів;
3. кожен план прив’язаний до конкретної техніки та операції;
4. об’єкти зберігаються ідентифіковано, з автоматично згенерованими ID документів або вкладених колекцій.

Таке моделювання забезпечує модульність і масштабованість системи: кожен користувач ізольований, що дозволяє працювати незалежно з його даними, а також зберігати історію планування для кожного поля.

1.3.3 Діаграма прецедентів

Діаграма прецедентів (рис. 1) відображає основні сценарії використання додатку AgroPlan з точки зору користувача. У центрі уваги взаємодія

користувача із системою. Зліва зображено актора «Користувач», а зправа набір прецедентів (use cases), які описують доступні дії.

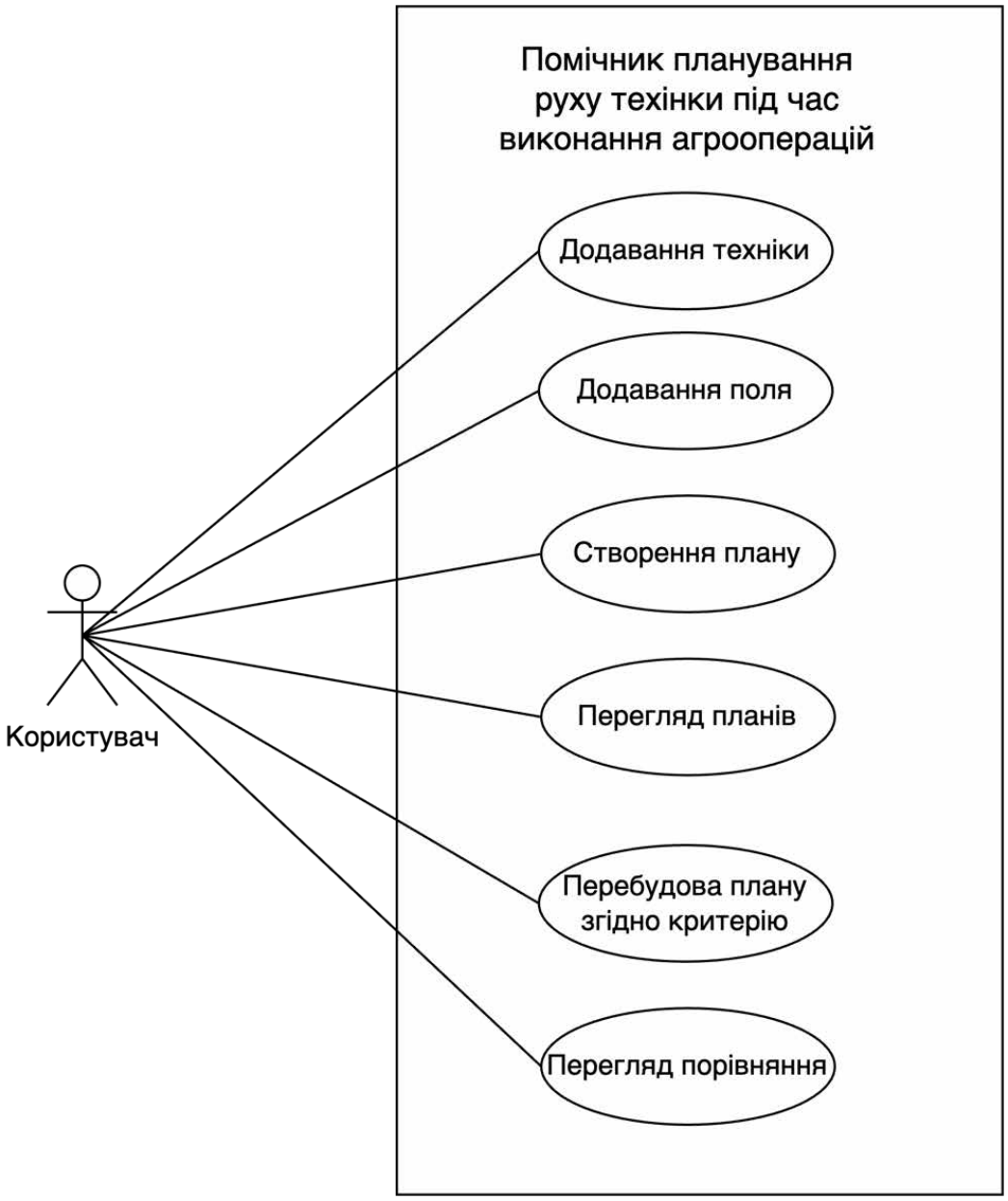

Рис. 1 Діаграма прецедентів додатку AgroPlan

Зокрема, користувач може:

- додавати поле, задаючи його геометрію та культуру;
- додавати техніку, що використовується для виконання агрооперацій;
- створювати план руху техніки відповідно до параметрів поля й техніки;
- переглядати список раніше створених планів;

- перебудовувати план відповідно до певного критерію оптимізації (наприклад, найменша витрата пального, мінімум проходів або найкоротший час);
- переглядати результат порівняння між базовим і оптимізованим планом.

Ця діаграма дозволяє наочно зрозуміти, які функціональні модулі передбачено у додатку, і як саме користувач взаємодіє з кожним із них. Вона також ілюструє логіку побудови інтерфейсу, де кожен із прецедентів може бути реалізований як окремий екран або розділ в мобільному застосунку.

1.3.4 Діаграма діяльності

Діаграма діяльності (рис. 2) відображає логіку взаємодії користувача з додатком під час створення плану агрооперації. Вона дозволяє побачити загальну послідовність дій, розгалуження залежно від вибору або наявності даних, а також кінцеву мету — генерацію оптимального маршруту техніки.

Процес починається з авторизації користувача в системі. Після входу він переходить у меню планування агрооперацій. Далі реалізовано перевірку: якщо поле вже існує — користувач вибирає його зі списку, якщо ні — додає нове поле, вказуючи назву та геометрію. Аналогічно відбувається з технікою: система перевіряє наявність збереженої техніки, або користувач створює новий запис.

Після цього відбувається вибір типу обробки (наприклад, сівба, обприскування, обробка, жнива), що задає алгоритм планування. На завершення система генерує та виводить план руху техніки на полі у вигляді маршруту, збереженого в GeoJSON-форматі та відображеного на карті.

Така діаграма дозволяє зрозуміти, як виглядає типовий користувацький сценарій і де в системі відбувається логічне розгалуження або перевірка умов.

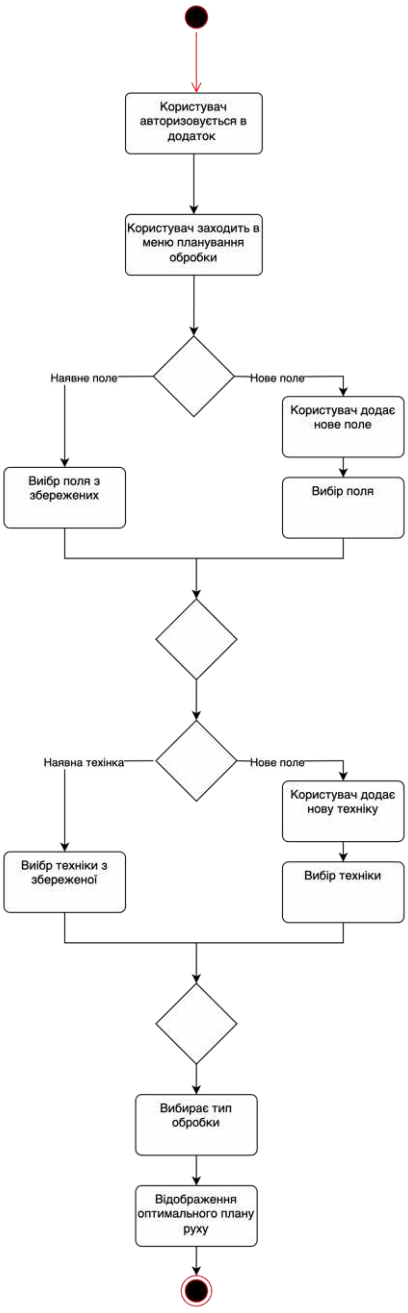

Рис. 2 Діаграма діяльності

1.3.5 Діаграма послідовності

Ця діаграма описує взаємодію між користувачем, інтерфейсом додатку. Вона демонструє динамічний характер обміну даними при формуванні плану: як запит від користувача ініціює вибір поля, підвантаження техніки та операції, генерацію маршруту, збереження результатів у Firestore і побудову карти. Таким чином, діаграма підтверджує цілісність архітектури взаємодії компонентів.

Діаграма послідовності (рис. 3) детально відображає етапи взаємодії між користувачем та системою під час створення та перегляду плану агрооперації. Вона побудована як послідовність обміну повідомленнями між об'єктами: актором (Користувач), інтерфейсом додатку та внутрішнім компонентом Планувальник, який відповідає за обробку даних і логіку формування плану.

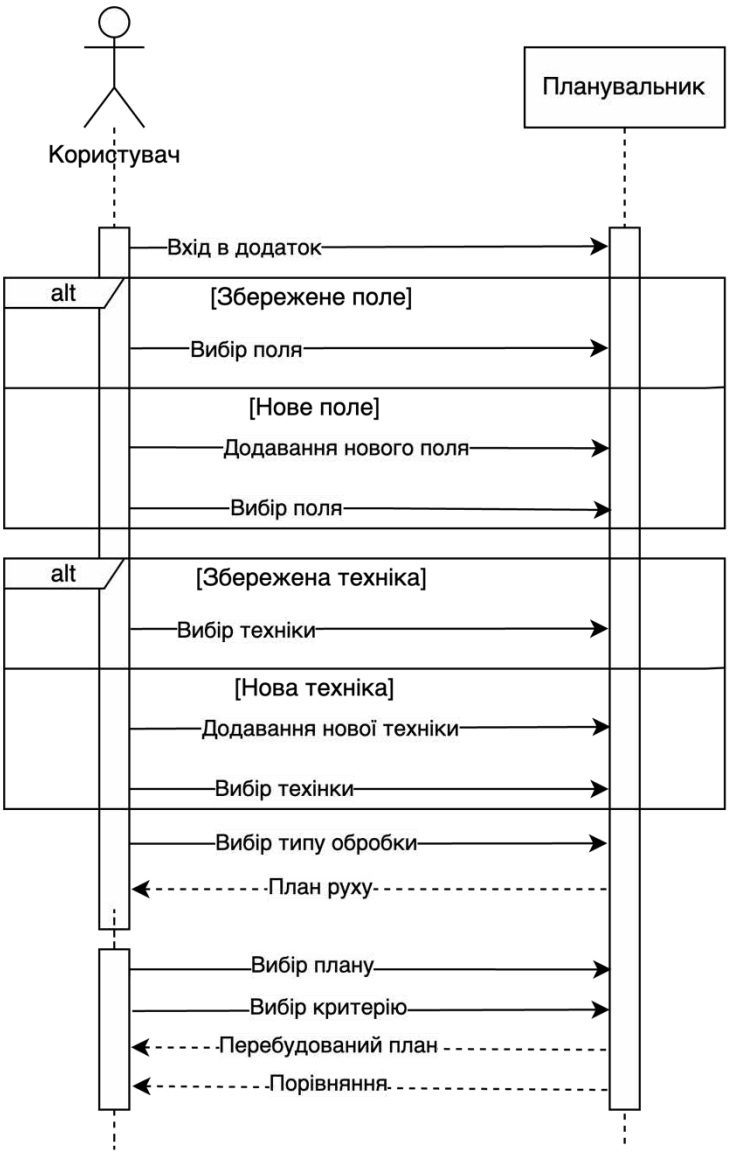

Рис. 3 Діаграма послідовності

Процес починається з входу користувача в додаток, після чого відбувається вибір поля: або з переліку вже збережених, або шляхом додавання нового. Далі аналогічно обирається техніка: або зі списку, або нова — зі збереженням її параметрів. Користувач обирає тип агрооперації (наприклад, сівба), і відбувається генерація первинного плану руху техніки.

На наступному етапі користувач має можливість переглянути план, задати критерій оптимізації (наприклад, найменша витрата пального), і система генерує альтернативний план, що зберігається як окрема версія. Після цього реалізується функція порівняння — користувач може побачити, чим відрізняється оптимізований маршрут від базового.

Діаграма також відображає умовні блоки (alt), які вказують на можливі варіанти сценаріїв залежно від дій користувача. Така модель дозволяє проаналізувати послідовність викликів і виявити критичні точки взаємодії з іншими сервісами.

1.4 Функціональні та нефункціональні вимоги до системи

У процесі розробки програмної системи AgroPlan було сформульовано ряд вимог, які визначили її функціональну придатність, надійність і зручність для кінцевого користувача — агроспеціаліста, який працює в реальних польових умовах. Усі вимоги поділяються на функціональні та нефункціональні.

Функціональні вимоги описують, що саме система повинна робити:

- Реєстрація та автентифікація користувача. Користувач повинен мати можливість створити обліковий запис і входити в систему за допомогою електронної пошти та пароля. Додатково підтримується автентифікація через Google-акаунт.
- Створення і редагування сільськогосподарських полів. Система має дозволяти користувачу позначати контури полів на карті, вказувати назву поля та культуру, обчислювати площу і зберігати всі дані у базі.
- Додавання сільськогосподарської техніки. Користувач може створювати картки техніки, вказуючи її тип, ширину захвату, швидкість руху та витрату пального. Ці параметри використовуються в алгоритмах побудови плану обробки.
- Побудова маршруту руху техніки. На основі форми поля та характеристик техніки система має генерувати маршрут руху у вигляді паралельних проходів, враховуючи напрям обробки, кількість проходів і розвороти.
- Збереження та перегляд планів. Кожен згенерований маршрут (план) повинен зберігатися в базі даних з можливістю подальшого перегляду, редагування або видалення.

- Відображення геоданих на карті. Користувач має змогу переглядати поля та маршрути на інтерактивній карті, масштабувати її, переміщуватись та змінювати стиль відображення.
- Підтримка офлайн-режиму. У випадку втрати інтернет-з’єднання система повинна зберігати останні доступні дані локально й забезпечувати їх синхронізацію після відновлення підключення.

Нефункціональні вимоги визначають якість реалізації функцій:

- Продуктивність. Система повинна реагувати на основні дії користувача (відкриття карт, побудова маршруту, збереження) протягом часу, не більшого за 500 мілісекунд у більшості сценаріїв.
- Надійність. Критичні дії, такі як видалення даних або перезапис маршрутів, мають супроводжуватись підтвердженням від користувача, щоб уникнути втрати інформації.
- Зручність використання (юзабіліті). Інтерфейс має бути інтуїтивно зрозумілим, оптимізованим для використання в польових умовах: великі кнопки, контрастні кольори, мінімалістичний дизайн.
- Масштабованість. Архітектура системи повинна дозволяти обслуговування великої кількості користувачів і зберігання значного обсягу геоданих без втрати продуктивності.
- Безпека. Доступ до даних дозволяється лише авторизованим користувачам. Аутентифікація реалізована на базі Firebase Auth, із захистом сесій і перевіркою ролей.
- Переносимість. Застосунок повинен стабільно працювати на Android-пристроях з версії 8.0 і вище, а також коректно адаптуватися до екранів різного розміру і співвідношення сторін.

1.5 Висновки до розділу

У процесі моделювання предметної області було розроблено комплексну систему структурних і поведінкових моделей, що охоплюють всі рівні взаємодії між користувачем і додатком AgroPlan. Визначено ключові сутності, логічні зв’язки між ними, реалізовано поділ відповідно до принципів Clean Architecture, візуалізовано основні сценарії роботи за допомогою UML-діаграм.

Таке моделювання дозволило не лише структурувати внутрішню логіку додатку, а й забезпечити гнучкість, масштабованість та можливість розширення функціоналу в майбутньому. Створені моделі стали надійною основою для реалізації архітектури, побудови бази даних, механізмів генерації планів і взаємодії з зовнішніми сервісами.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

Інформаційне забезпечення є важливою складовою будь-якої програмної системи, оскільки визначає способи, принципи та технології роботи з даними, що забезпечують виконання її функціональних можливостей. Для додатку AgroPlan інформаційне забезпечення включає створення ефективної логічної моделі даних, вибір відповідної бази даних, розробку фізичної структури зберігання, реалізацію формату геопросторових даних, забезпечення безпеки та аналіз продуктивності системи.

2.1 Логічна модель даних

Логічна модель даних є фундаментом функціонування інформаційної системи AgroPlan, оскільки вона визначає, які об’єкти зберігаються, як вони між собою взаємодіють і в якій формі використовуються під час реалізації функціональності додатку. Ретельно продумана логічна структура дозволяє забезпечити узгодженість, цілісність та масштабованість даних.

Основу логічної моделі AgroPlan складають наступні сутності:

- **Користувач (User)** — кожен користувач, зареєстрований через Firebase Authentication, має унікальний UID, який є ключем до доступу до його особистих даних у Firestore. Для кожного користувача створюється окрема колекція, що містить його поля, техніку, плани, а також профільні дані (email, ім’я, дата реєстрації тощо). Користувач може додавати, редагувати і видаляти будь-які власні сутності, але не має доступу до даних інших користувачів.
- **Поле (Field)** — сутність, що представляє фізичну ділянку землі. Поле має такі атрибути: назва, культура (тип рослини), площа (розрахована на основі координат), геометрія у форматі масиву точок країв поля. Поле може мати складну форму з декількома вершинами, які задаються списком координат. З кожним полем асоціюється колекція планів обробки, які формуються на його основі.
- **Техніка (Machinery)** — описує одиницю сільськогосподарської техніки, яка може бути застосована для виконання агрооперацій. Атрибути: назва,

тип (трактор, обприскувач, сівалка), ширина захвату, середня швидкість, витрата пального. Кожен користувач має власний набір техніки, яка не дублюється між акаунтами.

- **Агрооперація (Operation)** — сутність, що задає тип роботи, яка буде виконуватися на полі: сівба, культивація, обприскування тощо. Тип операції впливає на логіку побудови маршруту: наприклад, для сівби критичною є рівномірність покриття, тоді як для обприскування — мінімальна кількість розворотів.
- **План обробки (Plan)** — центральна сутність, яка містить результат роботи алгоритму побудови маршруту. Вона включає: унікальну назву, посилання на техніку (machineryId), операцію (operationId), геометрію маршруту (LineString у форматі GeoJSON), площу покриття (areaProcessed), довжину маршруту (totalDistance), кількість проходів, витрати пального, час виконання, ознаку первинності (isBase) та ідентифікатор батьківського плану (parentPlanId) у разі, якщо план створено як оптимізований варіант на основі іншого.
- **Критерій порівняння (AnanliticCriteria)** —використовується для побудови альтернативних планів. Можливі критерії: мінімальна довжина маршруту, мінімальна витрата пального, мінімальна кількість розворотів. У майбутньому цю логіку можна винести в окрему сутність, щоб надавати користувачу можливість керувати власними критеріями.

Зв’язки між сутностями побудовано за принципом вкладеності:

- Користувач → Колекція полів;
- Поле → Колекція планів;
- Кожен план містить посилання на об’єкт техніки та тип операції.

Це дозволяє швидко знаходити всі плани, що стосуються конкретного поля, а також порівнювати ефективність планів, побудованих із використанням різної техніки або під різні агрооперації. Вкладена структура також забезпечує логічну ізоляцію даних і спрощує реалізацію правил доступу.
Для ілюстрації використовується ER-діаграма (рис. 4), яка візуалізує сутності, їх атрибути та зв’язки. Кожна сутність має первинний ключ, що відповідає ідентифікатору документа у Firestore, та зовнішні ключі, представлені у вигляді ID-посилань на інші документи.

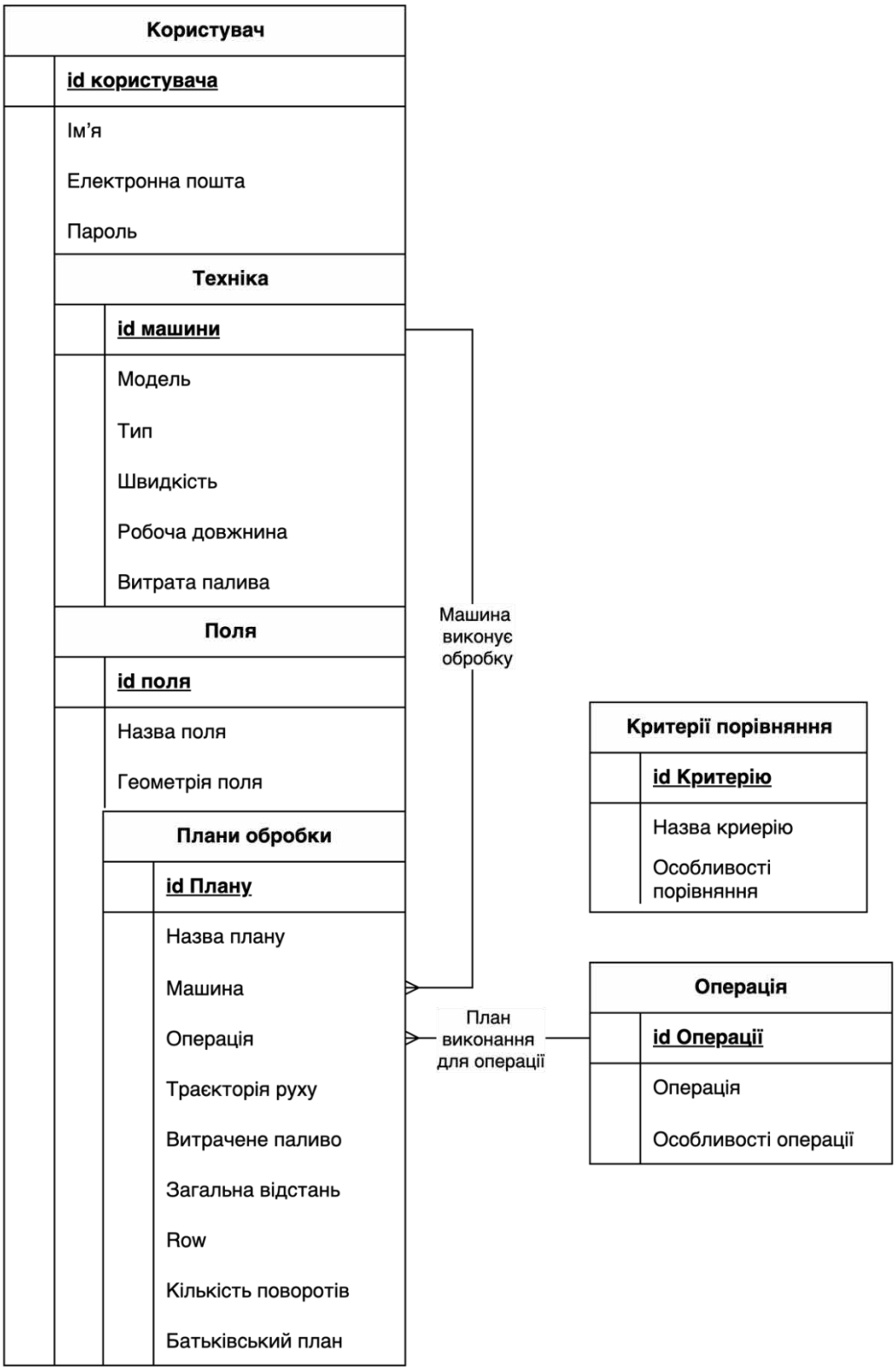

Рис. 4 ER діаграма

Завдяки такій логічній моделі система може забезпечувати стабільну та швидку роботу навіть при великій кількості користувачів і полів. Усі дані легко масштабуються, сортуються, фільтруються та обробляються як у реальному часі, так і в офлайн-режимі.

2.2 Вибір системи управління базами даних

У контексті розробки мобільного застосунку AgroPlan вибір системи управління базами даних (СУБД) є критично важливим рішенням, оскільки база даних є не лише сховищем для даних користувачів, але й основою для обчислень, побудови маршрутів, аналітики та синхронізації з хмарою. Враховуючи специфіку предметної області — робота в полі, можливі перебої з інтернетом, необхідність швидкого доступу до геоданих та складну структуру об’єктів — було розглянуто кілька варіантів СУБД, які могли б відповідати вимогам додатку.

2.2.1 Порівняння можливих СУБД

1. SQLite

- Переваги: проста вбудована база, не потребує мережевого з’єднання, швидка для базових CRUD-операцій.
- Недоліки: не підтримує складну ієрархічну структуру даних, відсутня синхронізація між пристроями, не масштабована.

2. PostgreSQL (через REST API)

- Переваги: потужна реляційна СУБД, підтримка просторових даних (PostGIS), висока надійність.
- Недоліки: складна у розгортанні для мобільних додатків, потребує сервера, немає вбудованого офлайн-режиму, погано підходить для мобільної клієнтської архітектури.

3. Realm DB

- Переваги: мобільна NoSQL СУБД, має підтримку офлайн-режиму, зручний API.
- Недоліки: складна інтеграція з Firebase, не підтримує кастомні права доступу та спільну авторизацію, обмежена масштабованість без Realm Cloud.

4. MongoDB Atlas (через Realm або API)

- Переваги: документна модель, гнучка структура, підтримка хмарного зберігання.
- Недоліки: потребує окремого беку, складніше забезпечити офлайн-роботу на клієнті, обмежена інтеграція з Android SDK.

5. Firebase Firestore

- Переваги: документно-орієнтована, масштабована, з підтримкою офлайн-режиму, реактивних потоків, ієрархічного зберігання даних, безпечного доступу, інтеграції з Firebase Auth.

- Недоліки: складніше реалізувати складні транзакції, обмеження за розміром документів.

2.2.2 Обґрунтування вибору Firestore

Після порівняльного аналізу було прийнято рішення використати Firebase Firestore — хмарну, документно-орієнтовану NoSQL СУБД, розроблену компанією Google спеціально для мобільних та веб-застосунків.
Firestore працює за принципом зберігання структурованих JSON-подібних документів у колекціях. Документи можуть містити вкладені об’єкти та колекції, що дозволяє зручно відтворити ієрархію сутностей (наприклад: користувач → поле → плани). Для AgroPlan це стало критично важливим, оскільки дозволило прив’язати всі об’єкти до унікального userId, зберігаючи повну ізоляцію даних.

Firestore має вбудовану підтримку офлайн-режиму. Це означає, що всі дані автоматично кешуються на пристрої користувача. Якщо підключення до мережі відсутнє, додаток продовжує працювати з локальними даними. Після відновлення з’єднання Firestore автоматично виконує синхронізацію змін. Цей механізм є особливо важливим для агросектору, де стабільне покриття мобільного інтернету не завжди доступне.

Ще однією важливою перевагою є реактивність: усі зміни, що відбуваються у базі даних, миттєво доступні у вигляді потоків (streams), які оновлюються в реальному часі. Завдяки цьому додаток може відображати оновлену інформацію без необхідності ручного оновлення або повторних запитів. Це значно покращує UX.

Firestore також має високу масштабованість і здатна обробляти тисячі одночасних запитів без втрати продуктивності. Це важливо для потенційного масштабування AgroPlan у SaaS-сервіс із сотнями або тисячами користувачів.

Окрім того, Firebase надає вбудовані засоби авторизації, контролю доступу (через Security Rules), аналітики, хостингу, що зменшує потребу у зовнішніх сервісах і спрощує підтримку проекту.
Таким чином, Firestore задовольняє такі критичні вимоги:

- Можливість роботи офлайн;

- Ієрархічне зберігання даних із вкладеними колекціями;
- Висока швидкість роботи і масштабованість;
- Реактивне оновлення даних;
- Безшовна інтеграція з Firebase Authentication та Android SDK;
- Безпечний доступ і контроль прав;
- Простота налаштування та обслуговування.

Вибір Firestore був зроблений свідомо, з урахуванням технічних і бізнес-вимог, і є ключовим фактором успіху реалізації архітектури AgroPlan.

2.3 Структура даних у Firestore

Firebase Firestore забезпечує ієрархічну структуру даних, яка дозволяє зберігати вкладені об'єкти у вигляді колекцій і документів. Для системи AgroPlan це має вирішальне значення, оскільки кожен користувач працює з власними ізольованими наборами даних, а логіка структури напряму відображає відношення між сутностями.

Уся інформація організована за принципом: **користувач → поля → плани**, а також **користувач → техніка**. Це реалізується через вкладені колекції у Firestore, наприклад:

```
Users/
  └── {userId}/
     ├── Profile (document)
     ├── Machinery/ (collection)
     │   └── {machineId} (document)
     └── Fields/ (collection)
        └── {fieldId} (document)
           └── Plans/ (collection)
              └── {planId} (document)
```

Кожен документ у цій структурі містить набір полів, що відповідають атрибутам відповідної сутності. Наприклад, machineId має поля name, type, workingWidth, fuelRate, а planId містить trajectoryGeom, totalDistance, areaProcessed, fuelUsed, isBase, parentPlanId тощо.

Така структура дозволяє:

- ізолювати дані кожного користувача;
- швидко виконувати запити за допомогою індексів Firestore;

- обробляти колекції незалежно, наприклад, отримати всі плани для конкретного поля без звернення до інших частин БД.

Firestore не є реляційною БД, тому всі зв’язки зберігаються через явні посилання (ID) між документами, а не через зовнішні ключі. Це зменшує час обробки даних і підвищує гнучкість структури при майбутньому масштабуванні. Крім того, структура дозволяє легко реалізувати правила безпеки — доступ до кожної колекції визначається рівнем вкладеності й UID користувача.

2.4 Фізична реалізація бази даних

Фізична реалізація бази даних у мобільному додатку AgroPlan побудована на основі хмарної NoSQL СУБД Firebase Firestore. Уся взаємодія з базою даних відбувається через Kotlin-класи, які відповідають структурі документів у Firestore, з використанням асинхронних запитів і корутин. Завдяки цьому забезпечується гнучка модель зберігання, масштабованість та адаптивність до змін структури без необхідності міграції схем.

У додатку реалізовано патерн Repository, який абстрагує логіку доступу до бази даних і забезпечує чіткий поділ між Presentation та Data шарами. Такий підхід сприяє легкому тестуванню, повторному використанню коду та мінімізації залежностей між модулями. Для кожної ключової сутності створено окремий репозиторій:

- FieldRepository — зчитування, додавання та оновлення інформації про поля;
- PlanRepository — збереження та отримання маршрутів техніки, включаючи GeoJSON-траєкторії;
- MachineryRepository — взаємодія з технікою (додавання, вибірка, оновлення параметрів);
- AuthRepository — робота з Firebase Authentication та базовими даними користувача.
- Кожен репозиторій реалізує методи на основі suspend функцій Kotlin Coroutines, що дозволяє:
- виконувати всі запити у фоновому потоці без блокування UI;
- використовувати .await() для зручного оброблення Task-об'єктів Firebase;
- обробляти винятки через try/catch у ViewModel або безпосередньо в репозиторії.

Всі дані з Firestore зчитуються у вигляді QuerySnapshot або DocumentSnapshot і трансформуються у Kotlin-об'єкти за допомогою вбудованого механізму серіалізації або вручну — через мапінг полів документа. Зворотне збереження об'єктів у Firestore виконується або автоматично через .set(), або вручну у вигляді Map<String, Any>, що забезпечує контроль над структурою збережених документів.
Для демонстрації основних сценаріїв доступу до Firestore побудовано дві блок-схеми: зчитування (рис.5) та додавання даних(рис. 6).

Перша схема демонструє процес зчитування даних:

- метод репозиторію надсилає запит до Firestore;
- перевіряється наявність документів у колекції;
- у разі наявності — дані перетворюються у Kotlin-об’єкти;
- результат передається до ViewModel, яка оновлює стан UI.
- у разі відсутності документів — повертається порожній список.

Рис. 5 Схема зчитування даних

Друга схема описує процес додавання об’єкта до бази:

- створюється об’єкт Kotlin data class на основі введених користувачем даних;
- передається у відповідний метод репозиторію;
- формується запит до Firestore на запис документа;
- у разі успіху — повертається підтвердження, і UI відображає результат;
- у разі помилки — генерується повідомлення, яке обробляється ViewModel або UI.

Рис. 6 Схема додавання даних

Подібна реалізація дозволяє гнучко контролювати потік даних, повторно використовувати логіку зчитування/збереження та дотримуватися принципів MVVM. Особливістю є також ієрархічна структура зберігання документів: кожен користувач має власну гілку (Users/{uid}/...), в якій зберігаються

пов’язані документи (поля, техніка, плани). Це дозволяє забезпечити безпеку даних, масштабованість і чітку ізоляцію користувацьких сесій без потреби в окремому бекенді.
Фізична модель бази даних повністю відповідає логічній моделі, підтримує розподілену та реактивну обробку даних у реальному часі та забезпечує ефективне зберігання, пошук, оновлення та видалення даних у мобільному середовищі.

2.5 Безпека даних

Безпека даних у Firebase Firestore базується на використанні гнучкої системи правил доступу — Firebase Security Rules, яка дозволяє контролювати всі операції зчитування та запису даних на рівні документів і колекцій. У порівнянні з традиційними підходами до захисту баз даних (через авторизаційні токени або серверну логіку), Firestore надає можливість прямо вказати умови доступу до кожного документа на основі параметрів авторизації користувача.

У Firestore існує кілька варіантів побудови правил безпеки:
1. **Глобальні правила для всієї бази** (наприклад, лише читання, повна заборона на запис).
2. **Правила на рівні колекцій** — дозволяють встановлювати доступ лише для авторизованих користувачів.
3. **Динамічні правила, що базуються на UID користувача** — найгнучкіші, дозволяють реалізувати доступ «тільки до своїх даних».

У системі AgroPlan реалізовано третій підхід — доступ до даних дозволено лише у разі, якщо UID користувача, що здійснює запит, збігається з ідентифікатором у структурі бази даних. Наприклад:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /Users/{userId}/{document=**} {
      allow read, write: if request.auth != null && request.auth.uid == userId;
    }
  }
}
```

Це правило забезпечує двофакторну перевірку:

- Перевіряє, чи користувач авторизований через Firebase Authentication.
- Перевіряє, чи має він право доступу саме до тієї гілки даних, що відповідає його UID.

Таким чином, навіть якщо користувач дізнається шлях до чужого документа (наприклад, /Users/INSHYY_UID/Fields/123), він не зможе його прочитати або змінити.

Окрім цього, система безпеки Firestore має ще кілька особливостей:

- **Безпечна авторизація через Firebase Authentication** — підтримка OAuth 2.0, Google Sign-In, email+пароль, анонімний вхід, із забезпеченням унікального токена доступу.
- **Захист від ескалації прав** — користувач не може змінити правила доступу або перейти в іншу роль без адміністрування.
- **Логування запитів і аудит** — через Firebase Extensions можна реалізувати журнал змін.

У системі AgroPlan усі правила безпеки протестовано в середовищі Firebase Emulator Suite, що дозволяє перевірити сценарії доступу ще до розгортання в продакшн. Крім того, вся логіка доступу дублюється на рівні інтерфейсу: користувач не має кнопок чи маршрутів до чужих даних, навіть якщо API дозволив би їх побачити (що не трапляється завдяки правилам).

Таким чином, рівень безпеки системи відповідає сучасним вимогам щодо захисту персональних і геопросторових даних користувача.

2.6 Логіка обробки планів агроопераций

Окрему увагу в додатку AgroPlan приділено обробці планів агроопераций, що включають створення маршруту руху техніки по полю, його перегляд, експорт та порівняння з альтернативними варіантами.

Кожен план містить посилання на техніку, тип агрооперації, список точок маршруту, а також (опційно) ідентифікатор батьківського плану — базової версії, з якою порівнюється оптимізований варіант.

На діаграмі нижче (рис. 7) зображено процес обробки запиту на перегляд плану:

- користувач обирає план із переліку;
- система перевіряє, чи має план батьківський (первинний) варіант;
- якщо має — отримуються обидва документи з Firestore;

- система порівнює метрики та візуально відображає відмінності;
- після цього користувач може експортувати поточний план у GeoJSON-формат для подальшої обробки або перенесення.

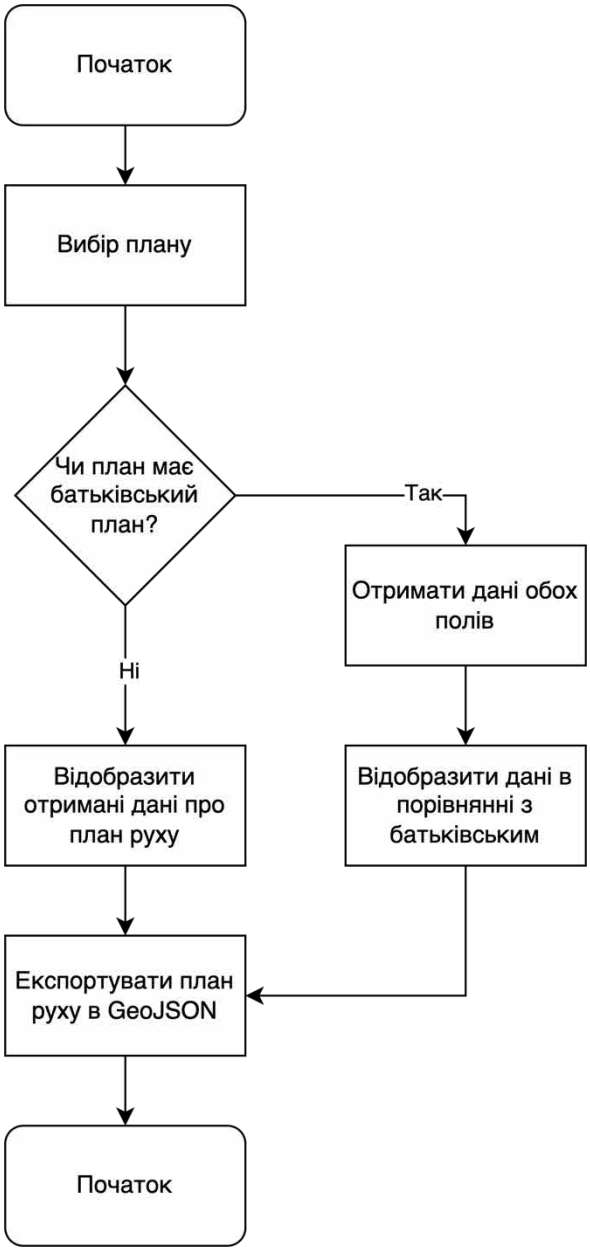

Рис. 7 Схема відбору даних

Ця логіка дозволяє не лише відстежувати результати оптимізації, а й забезпечує прозорість у порівнянні альтернативних рішень — за критеріями довжини маршруту, витрати пального або тривалості виконання.

2.7 Продуктивність та масштабованість

Однією з ключових вимог до інформаційної системи в аграрній сфері є її здатність обробляти великі обсяги даних без зниження продуктивності та з забезпеченням стабільної роботи в умовах зростання кількості користувачів. Firebase Firestore, обрана як основна система зберігання даних, надає всі необхідні механізми для досягнення високого рівня продуктивності й масштабованості, що було підтверджено під час тестування та впровадження системи AgroPlan.

Firestore автоматично оптимізує запити до бази даних через використання індексів. Це забезпечує швидкий доступ до документів навіть у великих колекціях, що особливо важливо для роботи з планами та полями, кількість яких може сягати сотень у межах одного облікового запису. Також важливо, що запити виконуються локально в офлайн-режимі, використовуючи кешовані копії даних. Це не лише прискорює доступ, але й забезпечує стабільність роботи у зонах зі слабким або відсутнім інтернет-з'єднанням.

Горизонтальна масштабованість Firestore дозволяє системі динамічно адаптуватися до збільшення навантаження. Це означає, що зростання кількості користувачів або запитів не потребує ручного втручання або оновлення інфраструктури — всі процеси масштабування відбуваються автоматично на стороні Google Cloud. Таким чином, AgroPlan залишається стабільним навіть у разі активного росту кількості клієнтів, господарств або аналітичних даних, які зберігаються в системі.

Окрім цього, продуктивність системи підтримується архітектурними рішеннями на рівні клієнтського додатку. Асинхронна взаємодія з базою через Kotlin Coroutines, використання репозиторіїв та механізмів обробки потоків дозволяє уникати блокування UI та забезпечує швидке реагування на дії користувача. Навіть при виконанні обчислювально важких операцій, таких як генерація маршруту або обробка геометрії полів, система залишається стабільною та придатною до реального використання.

Завдяки поєднанню всіх вищевказаних рішень, система демонструє високу продуктивність навіть при навантаженні понад середнє, а її архітектура повністю готова до розширення, що робить AgroPlan життєздатним рішенням у довгостроковій перспективі.

2.8 Висновки до розділу

Інформаційне забезпечення системи AgroPlan реалізовано з урахуванням сучасних вимог до розробки мобільних додатків у сфері аграрних технологій. Логічна модель даних дозволяє ефективно управляти зв’язками між об'єктами, такими як користувачі, поля, плани та техніка. Обрана СУБД — Firebase Firestore — повністю задовольняє технічні й бізнес-вимоги щодо гнучкості, масштабованості, продуктивності та безпеки.

Ключовими перевагами реалізованого підходу є:

- підтримка офлайн-режиму для роботи в полі;
- реактивне оновлення даних у реальному часі;
- ієрархічна структура зберігання, яка відображає логіку агроопераций;
- повна ізоляція даних користувача через UID-структуру та правила безпеки;
- можливість масштабування без модифікації архітектури;
- мінімізація серверних витрат за рахунок використання хмарних сервісів Firebase.

Загалом, побудована інформаційна модель є надійною основою для подальшого розвитку AgroPlan. Вона дозволяє не лише зберігати і обробляти дані, але й забезпечує умови для інтеграції з іншими системами, зокрема агрономічними інформаційними платформами, IoT-пристроями, агродронами тощо. Таким чином, AgroPlan має всі передумови для масштабування до комплексної цифрової платформи точного землеробства.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглянуто структуру та реалізацію програмного забезпечення системи AgroPlan. Зокрема описано архітектурні підходи, використані технології, організацію коду, а також алгоритми, які реалізують ключову функціональність — побудову маршруту руху техніки на полі. Розробка додатку виконувалась з урахуванням принципів чистої архітектури, що дозволяє забезпечити масштабованість, гнучкість, надійність та ефективність системи.

3.1 Архітектура програмного забезпечення

Архітектура програмного забезпечення AgroPlan реалізована з дотриманням принципів Clean Architecture та шаблону MVVM (Model–View–ViewModel). Такий підхід забезпечує чітке розділення відповідальності між компонентами додатку, сприяє масштабованості, високій тестованості та спрощує підтримку системи.

Clean Architecture передбачає поділ системи на ізольовані шари:

- **Presentation Layer (UI)** — реалізований за допомогою Jetpack Compose. Цей шар відповідає за відображення стану даних і взаємодію з користувачем. Дані надходять з ViewModel, яка формує відповідь на події користувача та зберігає стан екранів у вигляді об'єктів UiState та UiEvent.
- **Domain Layer** — описує бізнес-логіку програми. У системі AgroPlan цей шар є логічно виокремленим, але use-case класи не використовуються окремо. Бізнес-логіка реалізована безпосередньо у ViewModel через виклики методів репозиторіїв. Це дозволяє зберегти архітектурну прозорість за умов середньої складності додатку.
- **Data Layer** — включає репозиторії, джерела даних і маппери для перетворення DTO. Реалізує взаємодію з Firebase Firestore, Firebase Authentication та Mapbox API. Усі запити виконуються асинхронно з використанням Kotlin Coroutines, що дозволяє не блокувати UI.
- **Dependency Injection Layer** — організований за допомогою бібліотеки Koin, яка забезпечує інжекцію залежностей у ViewModel, репозиторії та

інші компоненти. Це дозволяє уникнути жорсткого зв'язування між класами та спрощує написання модульних тестів.

Завдяки використанню архітектурного шаблону MVVM:

- Користувацький інтерфейс не залежить від логіки обробки даних;
- ViewModel зберігає та обробляє стани у вигляді UiState та UiEvent;
- Усі дані передаються в UI через реактивні об’єкти (StateFlow, LiveData).

Взаємодія між компонентами реалізується відповідно до принципів MVVM:

- Користувач ініціює дію у графічному інтерфейсі (наприклад, створення нового плану агрооперації);
- Компонент View (екран Jetpack Compose) передає подію у відповідну ViewModel;
- ViewModel обробляє подію та викликає метод у відповідному репозиторії;
- Репозиторій звертається до зовнішнього API або Firestore для отримання чи збереження інформації;
- Результати запиту повертаються у ViewModel;
- ViewModel оновлює відповідний стан, який миттєво відображається у UI.

Такий підхід забезпечує низку переваг:

- низький рівень зв’язності між шарами;
- можливість заміни або вдосконалення окремих компонентів без впливу на інші частини системи;
- зручність тестування бізнес-логіки;
- покращена підтримуваність і розширюваність проєкту.

На діаграмі компонентів (рис. 8) зображено архітектуру додатку AgroPlan відповідно до принципів Clean Architecture. Система поділена на логічні функціональні модулі: field_management, auth, core та di.

Кожен модуль (окрім di) складається з трьох типових шарів:

- **domain** — містить базову бізнес-логіку та інтерфейси, ізольовані від реалізацій і сторонніх бібліотек;
- **data** — відповідає за взаємодію з Firebase Firestore, Firebase Auth, Mapbox API, а також містить маппери для перетворення моделей даних;
- **presentation** — реалізує ViewModel та UI-логіку за допомогою Jetpack Compose.

Модуль core виконує роль спільного шару — надає базові утиліти, типи та інтерфейси, які використовуються в інших модулях. Модуль di забезпечує конфігурацію залежностей (через Koin) та ініціалізацію компонентів додатку.

Таким чином, діаграма демонструє чітке логічне розділення архітектурних блоків, слабкий зв’язок між шарами та високу модульність структури додатку. Це спрощує масштабування, супровід та тестування окремих частин системи.

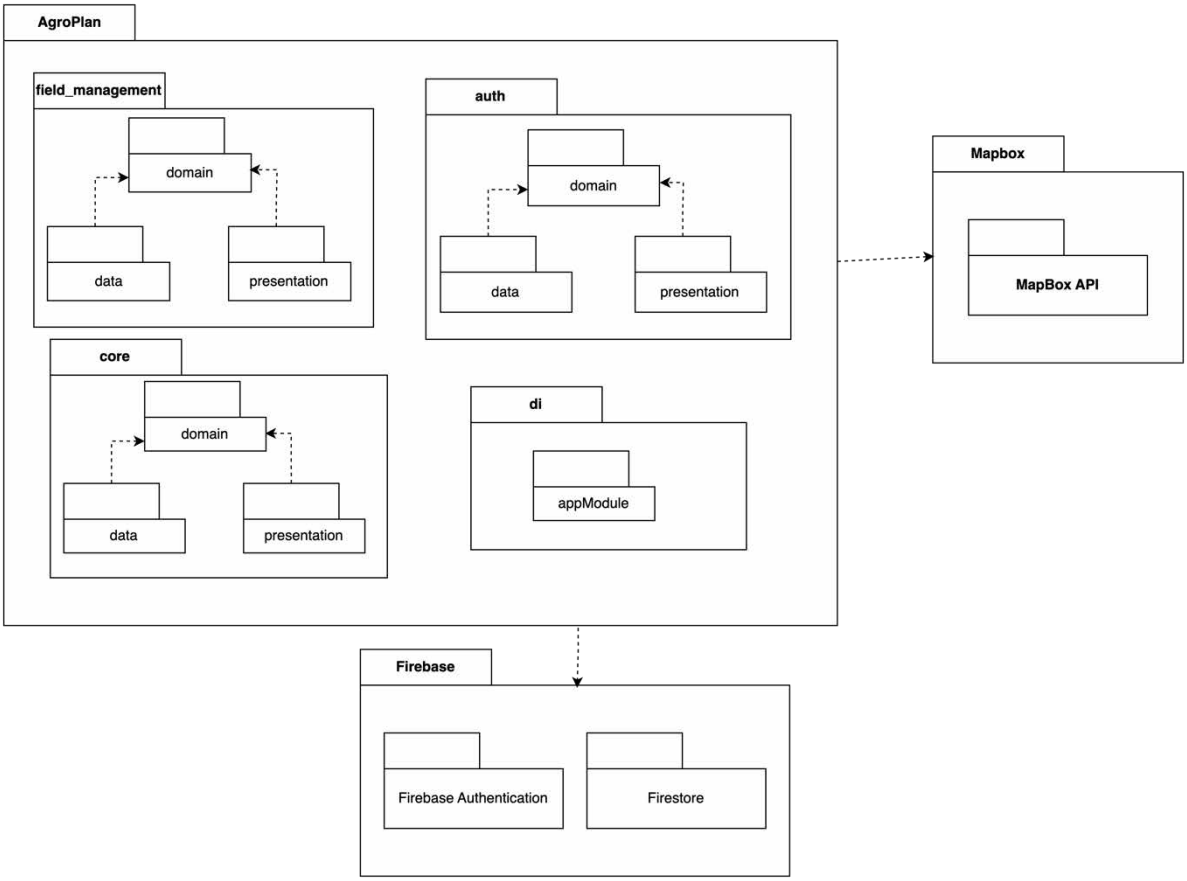

Рис. 8 Діаграма компонентів

Крім того, для демонстрації взаємодії між клієнтським додатком та зовнішніми сервісами, побудовано діаграму вузлів (рис. 9):

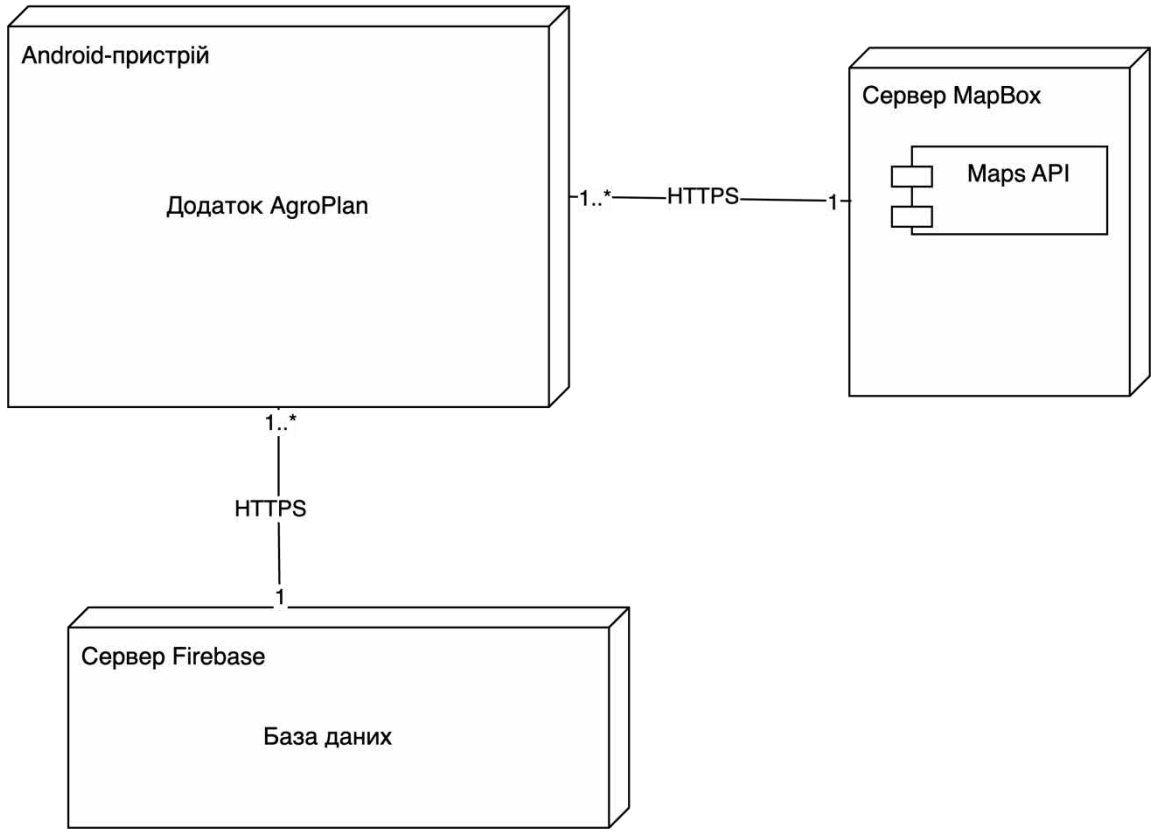

Рис. 9 Діаграма вузлів

На діаграмі вузлів зображено фізичну модель розміщення компонентів системи. Вона демонструє розміщення клієнтського додатку AgroPlan на Android-пристрої, який взаємодіє з серверами Firebase (що забезпечують авторизацію та зберігання даних) і Mapbox (для відображення картографічної інформації). Всі з’єднання здійснюються через HTTPS, а зв'язки описані як множинні (1..*) для підкреслення підтримки декількох клієнтів одночасно. Дана схема акцентує на розподіленому характері системи та забезпеченні взаємодії між її компонентами у реальному часі.

Таким чином, архітектура програмного забезпечення AgroPlan є сучасною, гнучкою і повністю відповідає вимогам до високоякісних мобільних додатків, що використовуються у виробничих умовах.

3.2 Обґрунтування вибору технологій

Розробка мобільного додатку AgroPlan потребувала добору технологічного стеку, який би відповідав вимогам продуктивності, надійності, зручності інтеграції та масштабованості. У цьому підпункті наведено пояснення вибору основних інструментів, мов, бібліотек і платформ, що були застосовані під час реалізації системи.

Kotlin як мова розробки

Основною мовою програмування для розробки додатку обрано Kotlin, яка є офіційною мовою для Android-застосунків і має низку переваг:

- підтримка корутин — асинхронного програмування без колбеків;
- лаконічний синтаксис, що підвищує швидкість розробки;
- безпечна робота з null-змінними (null safety);
- повна сумісність з Java та сучасними Android API.

У порівнянні з Java Kotlin є менш шаблонною, краще інтегрується з сучасним Android SDK, а також забезпечує значно коротший та безпечніший код. Альтернативою могла бути Dart у зв’язці з Flutter, але така комбінація складніше поєднується з Firebase, Firestore і нативними Android API, що ускладнило б реалізацію бізнес-логіки й авторизації.

Завдяки Kotlin стало можливим ефективно реалізувати архітектуру з використанням MVVM, Clean Architecture, а також працювати з Firestore у реактивному стилі.

Jetpack Compose як UI-фреймворк

Для створення користувацького інтерфейсу було використано Jetpack Compose — новітню бібліотеку від Google для декларативного формування інтерфейсу. Порівняно з класичним XML Layout + ViewBinding, Jetpack Compose має такі переваги:

- спрощена структура коду, менше boilerplate;
- зручне управління станами (State);
- підтримка анімацій, темізації, адаптації під різні екрани;
- висока швидкість оновлення інтерфейсу;
- краща інтеграція з MVVM і реактивними потоками.

Іншою альтернативою був Flutter (на базі Dart), який надає мультиплатформену підтримку. Проте Flutter гірше поєднується з Kotlin-проєктами, складніше інтегрується з Firebase та Mapbox. Тому Jetpack Compose став оптимальним вибором для сучасного Android-додатку.

Цей фреймворк дозволив легко реалізувати інтерактивні карти, списки, діалоги та інші UI-компоненти.

Firebase як серверна платформа

У якості серверної частини обрано Firebase, а саме:

- Firebase Firestore — база даних для зберігання документів користувача (поля, плани, техніка);
- Firebase Authentication — для забезпечення безпечної авторизації користувачів;
- Firebase Rules — для контролю доступу на рівні документів.

Firebase забезпечує повноцінний стек для мобільних застосунків, дозволяє працювати офлайн, автоматично синхронізує дані та має просту інтеграцію з Kotlin через офіційний SDK.

Mapbox SDK та Turf для роботи з геоданими

Для візуалізації полів і побудови маршрутів використано Mapbox SDK — потужний картографічний фреймворк із підтримкою кастомних шарів, стилів та GeoJSON. Mapbox дозволяє:

- накладати контури полів у форматі Polygon;
- будувати та виводити маршрути (LineString);
- працювати з кастомним зумом, камерами, векторними шарами.

На відміну від Google Maps, який має обмеження щодо векторних шарів і обробки GeoJSON, Mapbox забезпечує гнучкість і глибшу кастомізацію. Також розглядався варіант OpenStreetMap (через OsmDroid або Leaflet), але він має обмежену підтримку Turf і складну інтеграцію з Firebase.

Для обчислень використовується бібліотека Turf (обгортка над Turf.js):

- розрахунок площі (TurfMeasurement.area());
- обчислення довжини маршруту (TurfMeasurement.length());
- визначення координат розворотів.

Ці бібліотеки є критично важливими для точного аналізу геометрії полів і маршрутів.

Корутинна модель обробки подій

Усі запити до бази, обчислення, завантаження відбуваються в окремих потоках за допомогою Kotlin Coroutines. Це забезпечує:

- асинхронну обробку без блокування головного потоку;
- обробку помилок через try/catch;
- можливість використання suspend та await() при роботі з Firebase SDK.

Інші технології та засоби

- Koin — для впровадження залежностей (Dependency Injection);
- StateFlow — реактивне зберігання та спостереження за станами ViewModel;
- Gradle — система збірки і керування залежностями.
- Переваги обраного стеку
- усі технології активно підтримуються і мають документацію;
- адаптовані для мобільного використання;
- не потребують окремого серверного оточення;
- дозволяють швидко масштабувати функціональність;
- забезпечують стабільну та надійну роботу додатку в полі.

Вибір саме таких технологій обумовлений необхідністю створити зручний, продуктивний та безпечний інструмент для фермерів, агрономів та операторів техніки. Реалізоване рішення підтвердило ефективність і придатність для використання в реальних аграрних умовах.

3.3 Структура програмного коду та модулів

Структура коду системи AgroPlan побудована згідно з принципами чистої архітектури (Clean Architecture) та патерну MVVM. Кожен компонент програми розміщено у відповідному логічному модулі, що забезпечує чітке розмежування обов’язків, зменшує зв’язність і спрощує подальший розвиток та підтримку системи.

Загалом структура додатку поділена на три основні рівні:

1. Модуль ui

Цей модуль містить:

- Composable-функції Jetpack Compose, що відповідають за відображення екрану;
- компоненти користувацького інтерфейсу (текстові поля, кнопки, діалоги);
- обробку станів (наприклад, PlanUiState, FieldUiState);
- підключення до ViewModel для отримання та відображення даних;
- навігацію між екранами (екран авторизації, профілю, полів, плану, техніки).

Навігація реалізована на базі Navigation Compose, маршрути описуються у вигляді sealed-класів або констант. Комунікація між екранами забезпечується передачею параметрів через навігаційний контролер.

2. Модуль viewmodel

Цей рівень реалізує посередницьку логіку між UI та бізнес-логікою. Основні компоненти:

- ViewModel-класи (наприклад, PlanViewModel, FieldViewModel);
- реактивне зберігання станів через StateFlow;
- обробка подій через UiEvent та Effect;
- логіка обробки винятків та повідомлень користувачу

Усі ViewModel реалізовані як lifecycle-aware компоненти, що дозволяє уникнути втрат стану при обертанні екрана або зміні життєвого циклу.

3. Модуль domain

Містить виключно інтерфейси репозиторіїв. Цей рівень не залежить від реалізації бази даних чи UI і визначає лише абстракції, з якими взаємодіє бізнес-логіка на рівні ViewModel. Наприклад:

- PlanRepository;
- FieldRepository;
- MachineryRepository;

Інтерфейси визначають набір функцій, які мають реалізовуватись у модулі data, зокрема: створення, оновлення, видалення або отримання сутностей. Завдяки

цьому забезпечується інверсія залежностей, яка дозволяє легко підмінювати реалізації під час тестування або зміни джерела даних.

4. Модуль data

Містить:

- реалізації репозиторіїв: PlanRepositoryImpl, FieldRepositoryImpl, MachineryRepositoryImpl;
- джерела даних: FirestoreDataSource, MapboxService, AuthDataSource;
- маппери: перетворення між DTO, Entity і Domain-моделями;

Використовується патерн Repository, що відділяє бізнес-логіку від способу зберігання даних. Кожен репозиторій має чіткий інтерфейс у domain-рівні.

5. Модуль di

В окремому пакеті описано Koin-модулі, що реєструють залежності:

- viewModel { PlanViewModel(get()) }
- single { PlanRepositoryImpl(get()) as PlanRepository }
- single { FirebaseFirestore.getInstance() }

Це дозволяє гнучко управляти залежностями і легко підмінювати реалізації під час тестування.

Крім цього, у проєкті присутні модулі:

- utils/ — допоміжні функції: форматування чисел, валідація форм;
- navigation/ — маршрути та логіка переходів між екранами;

Загалом структура коду в AgroPlan є логічно впорядкованою, модульною та розширюваною. Вона дозволяє розробникам ефективно орієнтуватися у проєкті, швидко додавати нові функціональні можливості та проводити модульне тестування. Таке структурування особливо важливе при подальшому масштабуванні системи або перенесенні частини функціоналу на інші платформи.

3.4 Алгоритм побудови плану руху техніки

Ключовою функціональністю системи AgroPlan є можливість побудови оптимального маршруту руху сільськогосподарської техніки при виконанні агроопераційй. Алгоритм дозволяє автоматизувати процес формування маршрутів з урахуванням форми поля, ширини захвату, напряму обробки, типу техніки та обраного критерію оптимізації (мінімальна кількість проходів, мінімальні витрати пального або часу).

Побудова маршруту реалізована повністю на стороні клієнта. Це дозволяє формувати плани в режимі офлайн, що особливо важливо в умовах польових робіт, де доступ до Інтернету може бути обмеженим або відсутнім.

3.4.1 Вхідні параметри

1. **Геометрія поля:** GeoJSON-об'єкт типу Polygon, що зберігає координати меж ділянки.

2. **Робоча ширина техніки:** параметр у метрах, залежить від обраної моделі машини.

3. **Тип агрооперації:** визначає критерії ефективності (рівномірність, швидкість).

4. **Напрямок обробки:** AB-лінія або автоматично визначений кут найкращого покриття.

5. **Критерій оптимізації:** задається користувачем або застосовується за замовчуванням.

3.4.2 Алгоритмічна логіка побудови маршруту

Крок 1. Підготовка вхідних даних:

- Отримання координат з Firestore (через Field.geometry).
- Конвертація до об’єкта Polygon з бібліотеки Mapbox.
- Обчислення площі (метрів квадратних) з використанням TurfMeasurement.area.

Крок 2. Визначення напрямку обробки:

- Автоматично: обчислюється довжина сторін поля та обирається найдовша як базова вісь.

Крок 3. Генерація проходів:

- Через заданий напрям формується система паралельних ліній на всьому полі.
- Відстань між лініями = ширина захвату техніки.
- Кожна лінія обрізається за межами поля за допомогою Turf lineSlice + booleanIntersects.

Крок 4. Побудова маршруту:

- Зі згенерованих проходів вибирається послідовність згідно з оптимальним маршрутом (змійка / зворотний хід).
- Створюється об’єкт GeoJSON типу LineString із усіма з’єднаними точками проходів.
- Зберігається в Plan.trajectoryGeom.

Крок 5. Аналітика та оцінка:

- Розраховується довжина (TurfMeasurement.length), кількість проходів, теоретичний час виконання (від швидкості техніки), витрати пального (fuelRate * площа).
- Усі дані округлюються і зберігаються до полів Plan.

Крок 6. Збереження плану:

- Дані передаються в репозиторій PlanRepository.savePlan().
- Записуються до колекції Users/{userId}/Fields/{fieldId}/Plans/{planId} у Firestore.
- У разі похідного плану вказується parentPlanId та критерій.

На рис. 9 наведено основні етапи алгоритму генерації плану руху техніки: починаючи з завантаження геометрії поля та параметрів техніки, завершуючи збереженням сформованого маршруту у Firestore. Алгоритм може бути умовно поділений на три блоки: підготовка даних, генерація маршруту і обчислення аналітики.

Рис. 9 Блок-схема алгоритму побудови маршруту руху техніки

Таким чином, реалізований алгоритм побудови плану руху техніки є достатньо гнучким, масштабованим і придатним до подальшого розвитку. Його використання дозволяє автоматизувати складний процес агропланування, зменшуючи вплив людського фактору, економлячи ресурси та підвищуючи якість обробки полів.

3.4.3 Особливості реалізації в коді

- Для геообчислень використовується бібліотека **Turf.kt**, яка дозволяє працювати з геометрією без необхідності серверної обробки.

- Обробка помилок реалізована через Result тип із success/failure та відображенням повідомлень користувачу.
- Усі геометричні об’єкти сумісні з Mapbox SDK та зберігаються у форматі GeoJSON для потенційного експорту в техніку.

3.4.4 Обмеження та оптимізації

- Алгоритм не враховує поки що рельєф, погодні умови та наявні перешкоди — це потенційний вектор розвитку.
- Не підтримуються поля зі складною внутрішньою геометрією (острови).
- Реалізовано спрощену логіку розворотів — лише мінімізація їх кількості.

3.4.5 Перспективи розвитку

- Додавання підтримки зон із пропуском (наприклад, ставки, дерева).
- Можливість ручного редагування маршруту.
- Побудова маршруту з урахуванням початкової точки в’їзду.
- Виведення оцінки ефективності в порівняльній формі (базовий / похідний план).
- Експорт маршруту у форматах, сумісних із John Deere, Trimble та іншими платформами.

3.5 Фрагменти реалізації та приклади коду

У цьому підпункті описано реалізацію найважливіших функціональних блоків додатку AgroPlan відповідно до архітектури MVVM. Повні фрагменти коду винесено у Додаток А, а тут наведено короткий зміст їхньої логіки.

3.5.1 Реєстрація нового користувача

Використовується Firebase Auth. Метод registerUser() створює обліковий запис за email і паролем, а також оновлює профіль користувача, додаючи його ім’я.

3.5.2 Авторизація користувача

Метод loginUser() реалізує вхід користувача через Firebase Auth із перевіркою помилок та поверненням результату через обгортку Result.

3.5.3 Створення поля

ViewModel приймає введені дані користувача (назва, культура, контур), обчислює площу (calculateArea) і викликає метод репозиторію для створення поля у Firestore.

3.5.4 Додавання техніки

Передані дані про техніку обгортаються у модель Machinery і передаються у machineryRepository.addMachine(...) з обробкою в корутині.

3.5.5 Генерація плану обробки

Метод отримує геометрію поля, генерує маршрут проходів техніки, фільтрує його за межами (через TurfMeasurement.inside(...)), обчислює метрики і створює об’єкт Plan, який зберігається у Firestore.

3.5.6 Відображення маршруту на карті (Mapbox)

Траєкторія плану конвертується у GeoJSON (LineString), після чого створюється Feature і передається у джерело карти (geoJsonSource(...)).

3.5.7 Видалення плану

Метод deletePlan(...) видаляє обраний маршрут із Firestore і оновлює стан за допомогою повторного виклику loadPlans(...).

3.5.8 Робота зі станами ViewModel

Усі стани представлені через sealed class (PlanUiState), що включає варіанти Loading, Success, Error для реактивного оновлення UI через StateFlow.

3.6 Візуалізація інтерфейсу користувача

Мобільний застосунок AgroPlan реалізований з урахуванням специфіки польових умов — мінімалізм, чіткість навігації, велика контрастність елементів і підтримка адаптивного інтерфейсу. У цьому підрозділі наведено ключові екрани додатку з коротким описом їхніх функціональних призначень.

На цьому екрані (рис. 10) реалізовано форму входу до системи за допомогою електронної пошти та пароля, а також кнопку входу через Google. Поля валідуються у режимі реального часу, а помилки — виводяться під формами.

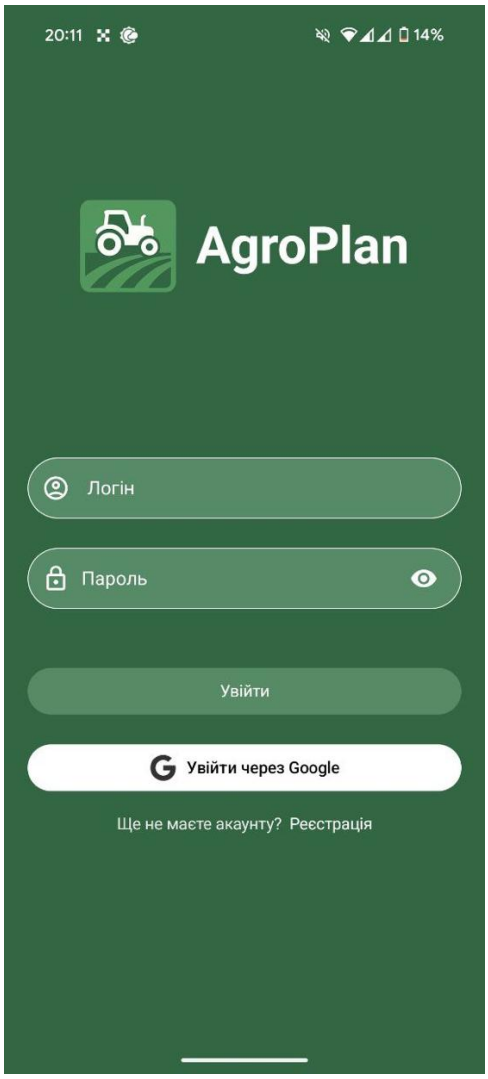

Рис. 10 Екран авторизації користувача

На екрані (рис.11) користувач бачить список усіх полів, які були створені. Кожне поле містить назву, культуру та площу. Доступне швидке перемикання між модулями внизу (Поля, Техніка, Профіль).

Після натискання на «+» користувач переходить до режиму створення поля (рис. 12). Поле обводиться на супутниковій карті. Площа розраховується автоматично, а дані вводяться в інтерактивні поля.

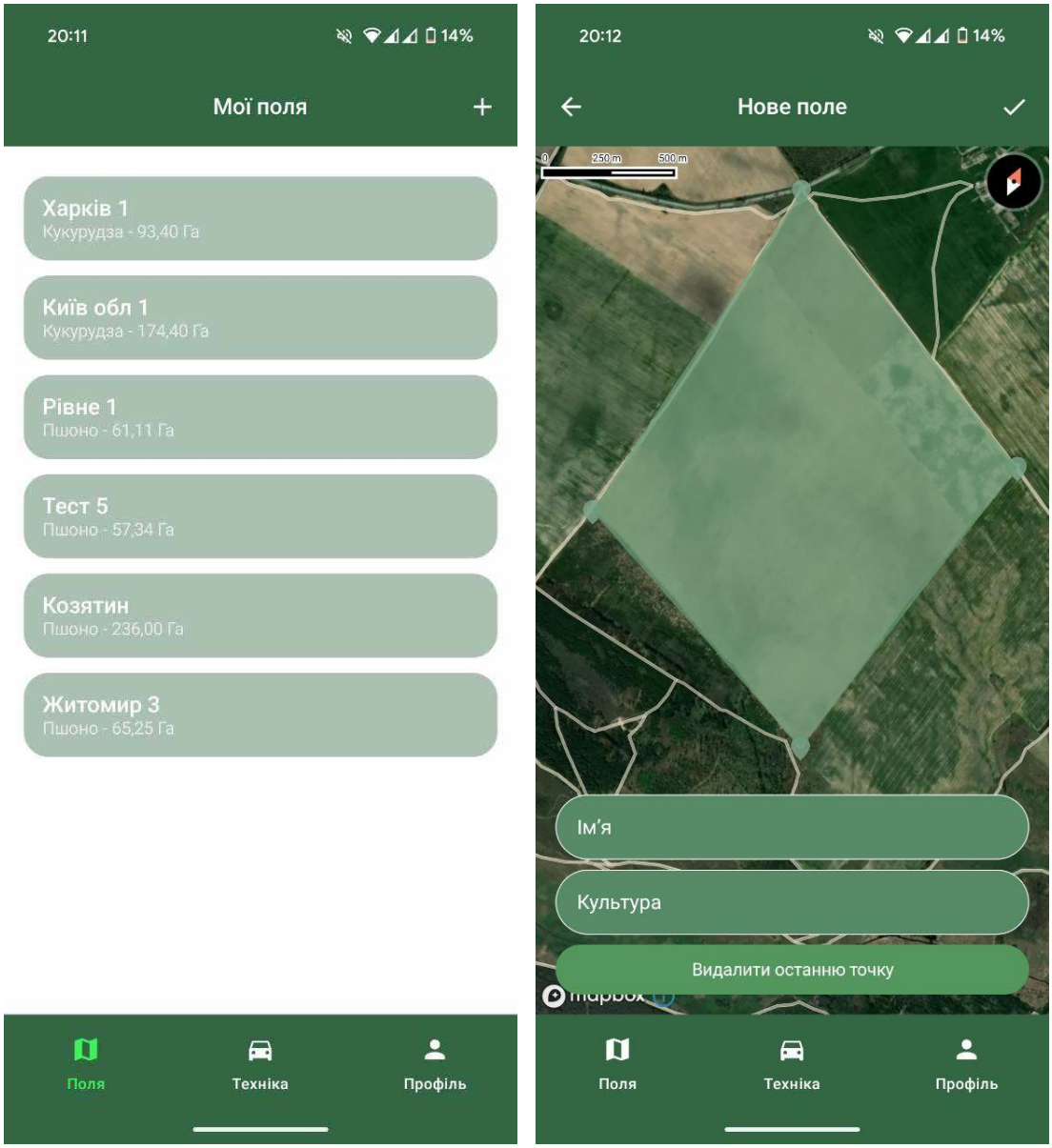

Рис. 11-12 Головний екран зі списком полів та створення нового поля на карті

У розділі «Техніка» (рис. 13) відображаються всі одиниці машин користувача. Для кожної вказано тип, робочу ширину, швидкість і витрату пального. Доступна функція видалення.

Рис. 13 Екран техніки

На екрані перегляду поля(рис. 14) користувач може сформувати план, при цьому він обирає тип обробки, назву плану і техніку. Поле відображається на карті. Натиснувши кнопку, користувач генерує маршрут руху техніки.

Після побудови маршруту користувач бачить фактичні проходи техніки по полю, довжину, площу обробки, кількість розворотів та орієнтовну витрату пального (рис. 15).

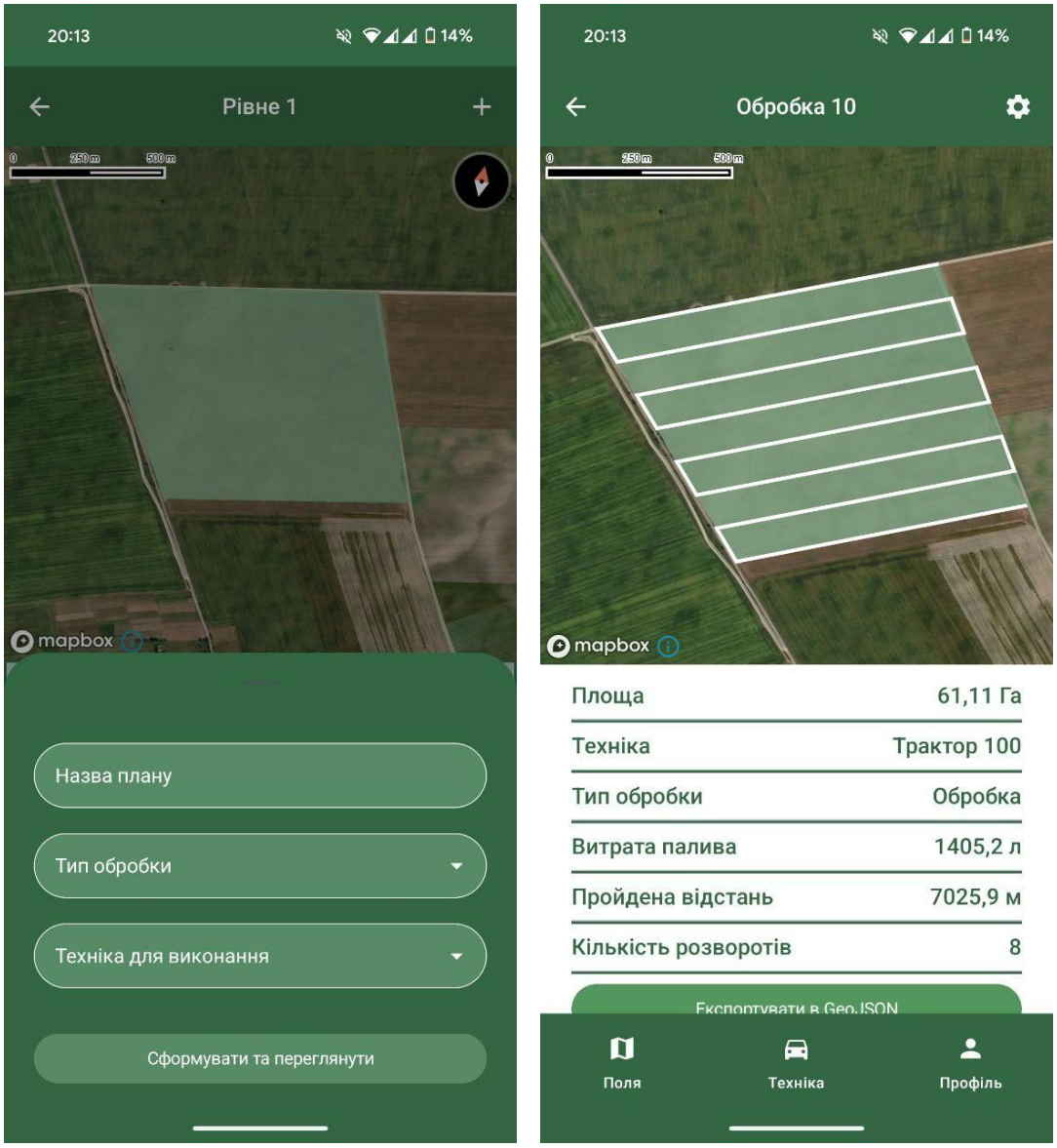

Рис 14-15 Формування плану обробки та візуалізація згенерованого маршруту

3.7 Висновки до розділу

У ході реалізації прикладного програмного забезпечення для мобільного додатку AgroPlan було розроблено повноцінну архітектуру на основі принципів MVVM і структурного поділу Clean Architecture. Такий підхід забезпечив чітке розділення відповідальності між UI, ViewModel, репозиторіями та джерелами даних, що дозволяє системі бути гнучкою, масштабованою та легко підтримуваною.

Використання сучасного технологічного стеку — Kotlin, Jetpack Compose, Firebase, Mapbox, Turf — забезпечило швидку, ефективну і безпечну реалізацію усіх функціональних вимог. Особливу увагу приділено реалізації алгоритму побудови маршруту: він виконується локально, враховує геометрію поля, технічні параметри машин та дозволяє порівнювати базові й оптимізовані плани.

Код проєкту організовано за принципами логічної модульності. Функції авторизації, створення, редагування, візуалізації та збереження даних реалізовані у ViewModel і пов’язані з репозиторіями, які інкапсулюють взаємодію з Firestore та іншими сервісами. Наведені приклади у розділі демонструють практичне використання кожного компонента системи.

Таким чином, прикладне програмне забезпечення AgroPlan демонструє успішну реалізацію архітектурних рішень, повну відповідність функціональним вимогам і технічну готовність до впровадження у виробничі процеси сільськогосподарських підприємств.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

Успішне впровадження програмного забезпечення AgroPlan передбачає не лише його встановлення на пристрої користувача, а й глибоке розуміння особливостей взаємодії із системою, дотримання технічних вимог, проходження повного циклу тестування, а також забезпечення супроводу в реальних умовах. У цьому розділі розглянуто повний цикл експлуатаційної підготовки системи, включаючи тестування, вимоги до середовища виконання, структуру інсталяційного пакета, рекомендації для користувачів, а також напрямки подальшого розгортання.

4.1 Тестування системи

Процес тестування програмного забезпечення AgroPlan є невід’ємною частиною життєвого циклу його розробки. Він охоплює перевірку працездатності окремих компонентів, взаємодії між ними, відображення користувацького інтерфейсу, а також коректність виконання ключових алгоритмів. Проведення всебічного тестування дозволило забезпечити високу стабільність системи та її готовність до впровадження у виробниче середовище.

4.1.1 Модульне тестування

Модульне тестування (unit testing) проводилось на рівні ViewModel. Воно охоплювало:

- обробку подій користувача (наприклад, введення назви поля, вибір техніки);
- формування станів екрану (UiState, PlanUiState);
- обробку виключень при виклику репозиторіїв;
- перевірку переходу між екранами через UI-події (UiEvent).

Тестування проводилось з використанням фреймворків JUnit та MockK. Було перевірено десятки сценаріїв, включно з прикордонними випадками: пусте поле, відсутність техніки, невалідні координати, тощо.

Приклад:
При створенні поля без назви — система повертає стан Error з повідомленням про помилку.

4.1.2 Інтеграційне тестування

Інтеграційне тестування передбачало перевірку зв’язку між ViewModel → Repository → Firebase Firestore. Тестувалися:

- збереження та зчитування об'єктів Field, Plan, Machinery;
- обробка помилок під час втрати з'єднання;
- відновлення даних при поверненні до онлайн-режиму;
- коректність вкладеної структури Firestore (Users/{uid}/Fields/{fieldId}/Plans/...).

Приклад:
При втраті з'єднання під час додавання плану — запис автоматично синхронізується після поновлення мережі.

З метою автоматизації інтеграційних сценаріїв застосовувався Firebase Emulator Suite, що дозволяє тестувати взаємодію з базою даних без реального з’єднання з інтернетом.

4.1.3 UI-тестування

Перевірка користувацького інтерфейсу здійснювалась за допомогою Jetpack Compose Testing API. Тестувалися:

- вивід форм полів, планів, техніки;
- коректна робота валідації даних;
- реакція інтерфейсу на навігацію;
- видимість елементів у різних станах (loading, success, error);
- відображення маршруту на карті.

Тестування проводилось як на фізичних пристроях, так і на Android Emulator, що дозволило перевірити різні діагоналі екранів та версії Android.

4.1.4 Функціональне тестування алгоритму

Особливу увагу приділено перевірці алгоритму побудови маршруту:

- створено набір тестових полів з простою та складною геометрією (включно з полями з виїмками);
- проведено побудову маршрутів з технікою різної ширини захвату;
- перевірено розрахунок площі (TurfMeasurement.area), довжини (TurfMeasurement.length), витрат пального;
- протестовано функцію TurfMeasurement.inside для обрізання сегментів за межами поля.

Порівняння результатів із теоретично очікуваними показало високу точність. Всі маршрути будувались з допустимим відхиленням у межах 1–2% площі поля.

4.1.5 Поведінкове тестування у польових умовах

З метою перевірки стабільності програми було проведено імітацію роботи в реальних умовах:

- запуск додатку без доступу до інтернету: перевірено офлайн-кеш Firebase;
- навмисне переривання мережі під час запису: переконанося, що запис синхронізується після відновлення зв’язку;
- тестування навігації, побудови маршруту на фізичному полігоні за координатами;
- збір зворотного зв’язку від потенційних користувачів (2 агрономи, 1 оператор техніки).

4.1.6 Результати тестування

Результати тестування показали, що система працює стабільно при дотриманні мінімальних вимог до пристрою. Усі критичні помилки були

усунені на етапі внутрішнього рев'ю. Було сформовано чек-листи регресійного тестування, які дозволяють перевіряти основні сценарії при кожному оновленні.

Для ілюстрації, в Додатку А представлено скріншоти інтерфейсу при побудові плану, а в Додатку Б — результат перевірки побудованого маршруту на полігоні за координатами.

Загалом система продемонструвала високу якість реалізації, що дозволяє впевнено рекомендувати її для впровадження в агровиробництво.

4.2 Вимоги до апаратного та програмного забезпечення

Забезпечення належних умов для ефективної роботи програмної системи AgroPlan вимагає дотримання певних апаратних і програмних вимог. Цей підпункт окреслює мінімальні й рекомендовані параметри пристроїв, необхідні дозволи, а також середовище виконання, в якому додаток функціонує найбільш стабільно.

4.2.1 Апаратні вимоги

Для забезпечення стабільної роботи AgroPlan достатньо базових технічних характеристик сучасних мобільних пристроїв:

- Операційна система: Android 8.0 (API level 26) або новіша. На старіших версіях можуть виникати проблеми з Mapbox або Firebase SDK.
- Оперативна пам'ять: мінімум 1 ГБ. Рекомендовано — 2 ГБ і більше для одночасного виконання кількох задач (навігація, побудова, авторизація).
- Процесор: ARMv7 або новіший. Рекомендовано — пристрої з підтримкою 64-бітної архітектури.
- Екран: діагональ від 5.0 дюймів, роздільна здатність не менше 720p. Інтерфейс оптимізовано під HD.
- Наявність GPS-модуля: обов'язкова для точного позиціонування та навігації при побудові плану обробки.
- Пам'ять: щонайменше 100 МБ вільного простору для встановлення програми, кешу Firebase і збереження локальних копій планів.

4.2.2 Програмні вимоги

AgroPlan інтегрує сучасні бібліотеки й сервіси, що потребують наявності певних програмних компонентів:

- Android OS: підтримка Android SDK 26+.
- Google Play Services: для коректної роботи Firebase Auth і геолокації.
- Firebase SDK: вбудований у додаток — користувачу не потрібно встановлювати додаткові сервіси.
- Mapbox SDK: використовується для відображення карт, полів, планів. Сумісність із API версією 10+.
- Інтернет-з'єднання: необхідне для первинної авторизації та синхронізації даних. Після цього більшість функцій працює в офлайн-режимі (за умови попереднього завантаження).
- Дозволи додатку:

1. Доступ до геолокації — обов'язковий для побудови та позиціювання;
2. Доступ до мережі — для авторизації, синхронізації з Firestore;
3. Доступ до зовнішньої пам'яті — опціонально, для збереження резервних копій чи логів (у майбутньому).

4.2.3 Сумісність та тестовані пристрої

Програма тестувалась на таких пристроях:

- Google Pixel 8 (Android 15): стабільна робота, офлайн-функції активні;
- Xiaomi Redmi Note 8 (Android 10): підтримка всіх основних функцій;
- Pixel 4a (Android 13, емулятор): повна сумісність із Firebase Emulator Suite;
- Huawei Y6p (Android 10 без GMS): обмежена робота через відсутність Google Play Services (Firebase Auth не працює).

У випадку запуску на пристроях без Google-сервісів рекомендується реалізувати альтернативні методи автентифікації або працювати через WebView.

4.2.4 Рекомендації щодо середовища використання

- Працювати в польових умовах рекомендовано у режимі із попередньо відкритими полями;
- Для великогабаритної техніки з планшетами (7+ дюймів) рекомендується режим ландшафтного перегляду;
- Застосунок розраховано на однокористувацький режим роботи — синхронізація акаунтів можлива, але адміністрування не реалізоване (в майбутньому можливий мультиакаунт);
- Підключення до Wi-Fi або стабільного мобільного інтернету необхідне для резервного копіювання.

Вимоги до середовища є достатньо помірними, тому система може бути впроваджена навіть на недорогих мобільних пристроях, що розширює доступність цифрових технологій для агросектору.

4.3 Інсталяційний пакет системи

Інсталяційний пакет системи AgroPlan розповсюджується у форматі APK-файлу, що є стандартним для розгортання Android-додатків поза межами Google Play. Крім того, для розробників та тестувальників доступна можливість встановлення через Android Studio безпосередньо з вихідного коду.

Основні компоненти інсталяційного пакета:

- **APK-файл** — головний виконуваний файл додатку, створений за результатами збирання проєкту у форматі Android App Bundle.
- **AndroidManifest.xml** — файл конфігурації, що містить відомості про дозволи (геолокація, мережа), визначає основні компоненти додатку (Activity, Service) та задає точку входу.
- **google-services.json** — конфігураційний файл для підключення до Firebase. Він містить унікальні ідентифікатори проєкту, API-ключі, посилання на базу даних.
- **Gradle-сценарії (build.gradle)** — скрипти, що визначають структуру проєкту, залежності (Mapbox SDK, Firebase SDK), плагіни, параметри компіляції.

- **Ресурсні файли** — включають графіку (іконки, логотипи), кольорові теми, шрифти, стилі оформлення, конфігурації для підтримки багатомовності.

Методи встановлення:

1. **Ручне встановлення APK:**

- передача файлу через USB, електронну пошту, месенджери;
- дозвіл на встановлення з невідомих джерел в Android;
- запуск файлу через файловий менеджер.

2. **Встановлення через Android Studio:**

- використовується в режимі розробника або при тестуванні;
- дозволяє швидко оновлювати код без повного перевстановлення додатку;
- забезпечує перегляд логів, трасування помилок, live preview інтерфейсу.

3. **Майбутнє розміщення у Google Play:**

- передбачає створення release-підпису APK (keystore);
- сертифікацію через Google Play Console (доступність, безпека);
- автоматичне оновлення користувачам;
- публікацію супровідних матеріалів (опис, скріншоти, відео).

Підтримка формату Android App Bundle:

AgroPlan відповідає вимогам сучасного стандарту Android App Bundle (AAB), що дозволяє оптимізувати розмір інсталяційного пакета під кожну конкретну конфігурацію пристрою (density, ABI, language). Це скорочує час встановлення та мінімізує обсяг займаної пам’яті.

Діаграма розгортання: На рисунку 16 наведено схему розгортання додатку AgroPlan, що демонструє архітектурну структуру розміщення програмних компонентів. Вона ілюструє, як клієнтський додаток, встановлений на Android-пристрої, взаємодіє з хмарними сервісами Firebase (Firestore, Authentication) та Mapbox SDK. Також діаграма відображає внутрішню організацію додатку, включаючи APK, конфігураційний файл google-services.json, ресурсні файли та бібліотеки, які зв’язуються з зовнішніми інтерфейсами через HTTPS.

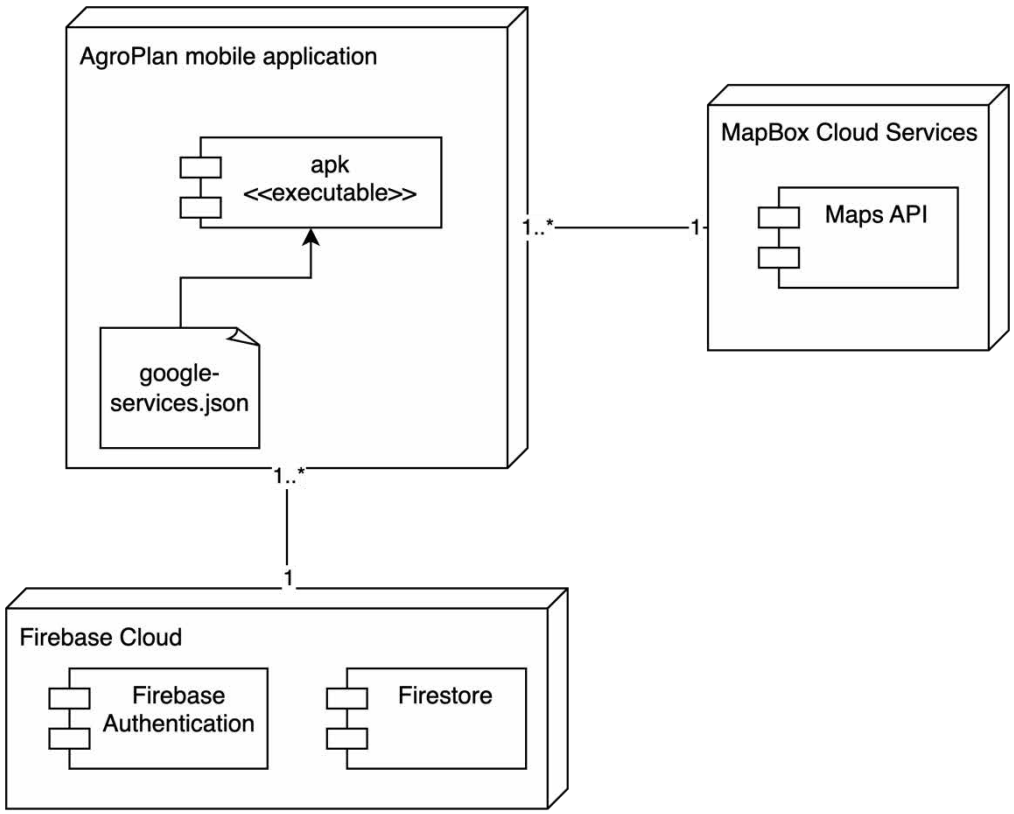

Рис. 16 Діаграма розгортання

Переваги поточного підходу:

- Простота доставки додатку без проміжного сервера;
- Незалежність від зовнішніх магазинів для перших користувачів (альфа-тест);
- Можливість локального встановлення в польових умовах;
- Гнучка підтримка модульної структури для подальшого оновлення.

Таким чином, структура інсталяційного пакета дозволяє як локальне використання на окремих пристроях, так і масштабоване розгортання через офіційні канали Android, забезпечуючи зручність, гнучкість і відповідність сучасним стандартам.

4.4 Рекомендації до подальшого впровадження

Для забезпечення ефективного впровадження та повноцінного функціонування системи AgroPlan в умовах сільськогосподарського

виробництва, необхідно враховувати низку технічних, організаційних та практичних аспектів. Нижче наведено розширений перелік рекомендацій, які базуються на результатах тестування, аналізі середовища використання та зворотному зв’язку з потенційними користувачами.

1. Робота в офлайн-режимі

AgroPlan підтримує повноцінний офлайн-режим завдяки локальному кешуванню Firebase та можливості попереднього завантаження тайлів Mapbox. Для цього необхідно забезпечити:

- перший запуск додатку у середовищі з інтернетом (для авторизації та завантаження полів);
- збереження полів і планів на пристрої перед виїздом у поле;
- завантаження базової карти в межах регіону обробки.

2. Резервне копіювання та синхронізація

Всі дані користувача зберігаються у Firestore. Для захисту від втрат рекомендується:

- активувати автоматичне збереження після кожної зміни плану/поля;
- періодично перевіряти стан синхронізації;
- реалізувати систему логування ключових дій користувача для внутрішнього аудиту (може бути реалізовано як розширення).

3. Навчання користувачів і документація

Ефективне використання системи залежить від розуміння її структури:

- створити покрокову інструкцію (зображення, приклади);
- провести навчальні демонстрації для агрономів, операторів техніки;
- надавати підтримку у вигляді FAQ або консультаційної лінії (телеграм-бот, email).

4. Централізоване керування акаунтами

Для великих агропідприємств доцільно:

- реалізувати розмежування ролей (адміністратор, оператор, аналітик);
- централізовано створювати та поширювати шаблони полів/техніки;
- забезпечити аудит дій користувачів через лог-файли або окрему панель керування (у перспективі).

5. Розширення функціональності

Система може бути доповнена:

- інтеграцією з агродронами для аерофотозйомки полів;
- експортуванням планів у ISO-XML, Shapefile для використання у бортових комп’ютерах техніки;
- підключенням до сторонніх ERP/CRM-систем через API;
- створенням веб-інтерфейсу для перегляду аналітики та масової обробки планів.

6. Підтримка кількох користувачів і синхронізація

У перспективі передбачено впровадження мультиакаунтності:

- підтримка кількох пристроїв з одним акаунтом (синхронізація в реальному часі);
- розмежування доступу до планів між агрономом та виконавцем (механізатором);
- можливість залишення коментарів до плану/поля в режимі реального часу.

Таким чином, дотримання запропонованих рекомендацій дозволить максимально ефективно впровадити систему AgroPlan у реальні агровиробничі процеси, забезпечити її масштабованість, простоту адаптації та подальший розвиток.

4.5 Подальший розвиток системи

У поточній реалізації система AgroPlan охоплює базову функціональність, необхідну для створення маршрутів обробки полів із урахуванням геометрії ділянки та характеристик техніки. Разом з тим, її архітектура та вибрані технології дозволяють поступово масштабувати застосунок, інтегрувати нові сервіси та розширювати функціональність системи у відповідь на потреби аграрного сектору.

Одним із ключових напрямів розвитку є реалізація веб-інтерфейсу для агрономів і менеджерів господарств. Це дозволить переглядати створені плани, координувати техніку, надавати доступ до даних іншим користувачам або адміністраторам, а також формувати звітність за підсумками польових операцій. Враховуючи наявність централізованого зберігання даних у Firestore, створення

веб-версії з використанням фреймворків React або Next.js є технічно обґрунтованим і не потребує змін у бізнес-логіці застосунку.

Ще одним перспективним кроком є розширення мобільної підтримки шляхом реалізації кросплатформенності на базі Kotlin Multiplatform. Це дозволить спільно використовувати бізнес-логіку між Android- і майбутньою iOS-версією, зменшуючи витрати на підтримку двох платформ.

З огляду на специфіку аграрної галузі, доцільною є інтеграція з іншими програмно-технічними системами — зокрема, ERP-системами, службами моніторингу техніки та метеосервісами. Для цього планується реалізація REST API з автентифікацією через токени доступу, що дасть змогу передавати маршрути, поля та аналітичні дані до зовнішніх платформ.

Важливим кроком розвитку є інтеграція з бортовим обладнанням техніки. Це дозволить автоматично отримувати дані про її положення, швидкість, витрати палива та інші параметри, а також надсилати інструкції напряму на техніку. Це значно покращить контроль виконання робіт і забезпечить точність.

Також перспективним є удосконалення самого алгоритму побудови маршруту. У майбутньому планується врахування особливостей рельєфу місцевості шляхом інтеграції цифрових моделей висот (DEM), що дозволить знижувати навантаження на техніку та економити пальне. Крім того, можлива реалізація розширеної аналітики: побудови графіків ефективності, історії обробки полів та прогнозування витрат.

Окремо варто виділити можливість впровадження трекінгу техніки у реальному часі з візуалізацією її переміщення на карті. Це дозволить поєднати планування та фактичне виконання робіт у єдиному інтерфейсі, що є особливо важливим для господарств із кількома одиницями техніки.

Таким чином, система AgroPlan має значний потенціал для подальшого функціонального і технологічного розвитку. Гнучка архітектура, сучасний стек технологій і практичне розуміння потреб кінцевого користувача створюють передумови для трансформації мобільного застосунку в повноцінну платформу для цифрового управління польовими агрооперaціями.

4.6 Висновки до розділу

Проведене тестування, аналіз інфраструктурних вимог і перевірка функціональності AgroPlan у лабораторних та польових умовах свідчать про високу готовність системи до впровадження в реальні виробничі процеси. Система стабільно функціонує як онлайн, так і в автономному режимі, не потребує значних ресурсів і сумісна з більшістю сучасних мобільних пристроїв.

AgroPlan відзначається низьким порогом входу для нових користувачів завдяки зрозумілому інтерфейсу, чіткій структурі взаємодії з полями, технікою та планами, а також підтримці авторизації через Firebase. Інтеграція з Mapbox дозволяє візуалізувати маршрути обробки без потреби у зовнішньому картографічному сервісі. Побудова маршруту відбувається локально, без підключення до зовнішніх серверів, що робить додаток придатним для використання в польових умовах.

Наявність механізмів резервного копіювання через Firestore, підтримка офлайн-режиму та гнучка система збереження геоданих дозволяють рекомендувати AgroPlan не лише для приватних фермерських господарств, але й для середніх агропідприємств.

Подальший розвиток системи, включаючи можливість мультиакаунтного режиму, підтримку нових форматів агроданих (Shapefile, ISO-XML), а також веб-інтерфейс для адміністрування, значно розширює її потенціал. Враховуючи це, AgroPlan є сучасним інструментом для цифровізації агровиробництва, підвищення точності агроопераций та зменшення витрат.

5 ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було розроблено, реалізовано та апробовано інформаційну систему AgroPlan — мобільний додаток, призначений для підтримки прийняття рішень у сфері агровиробництва, зокрема під час планування маршруту руху сільськогосподарської техніки. Розробка здійснювалася з урахуванням специфіки галузі, вимог до автономної роботи, ефективного використання просторових даних, а також потреби у доступності технологій для середніх і малих агропідприємств.

У процесі реалізації було виконано повний життєвий цикл створення програмного продукту:

1. здійснено детальний аналіз предметної області, виявлено існуючі проблеми в ручному плануванні маршрутів техніки та обґрунтовано актуальність автоматизації цього процесу;
2. сформульовано функціональні та нефункціональні вимоги до системи;
3. спроєктовано логічну структуру даних та архітектуру програми згідно з принципами MVVM та Clean Architecture;
4. реалізовано механізм побудови маршруту техніки з урахуванням геометрії поля, ширини захвату техніки, напрямку руху та критерію оптимізації;
5. забезпечено підтримку роботи в офлайн-режимі, що критично важливо для сільських територій з нестабільним інтернет-покриттям;
6. розроблено механізми збереження, перегляду, порівняння та видалення планів агроопераций;
7. інтегровано Firebase Firestore для хмарного зберігання та Mapbox SDK для відображення полів і маршрутів на карті;
8. реалізовано авторизацію користувачів через Firebase Authentication, що дозволяє безпечно розмежовувати доступ до даних;
9. створено інсталяційний пакет у форматі APK, проведено тестування системи, зібрано зворотний зв’язок від користувачів;
10. сформульовано технічні вимоги, інструкції з використання, та рекомендації щодо масштабованого впровадження системи в аграрне середовище.

На основі аналізу функціонування системи можна зробити висновок про її технічну, архітектурну та функціональну зрілість. AgroPlan відзначається простотою в інсталяції та використанні, не потребує складного серверного оточення, має гнучку структуру даних, підтримує офлайн-режим і

синхронізацію через Firebase. Тестування показало стабільність роботи на різних пристроях, у тому числі в умовах нестабільного зв'язку.

AgroPlan є масштабованою платформою, яка у перспективі може бути розширена до повноцінної багатофункціональної системи для агроменеджменту з підтримкою аналітики, веб-інтерфейсу, IoT-інтеграцій, модулів звітності та експорту даних у формати, сумісні з технікою.

Отже, у результаті виконання кваліфікаційної роботи досягнуто всі поставлені цілі. Створено інтелектуальну програмну систему, що сприяє підвищенню ефективності агроопераций, мінімізації витрат і покращенню управління сільськогосподарськими процесами за рахунок цифрових технологій.

ПЕРЕЛІК ЛІТЕРАТУРИ

1. Android Developers Guide. [Електронний ресурс]. — Режим доступу: https://developer.android.com/guide
2. Firebase Documentation. [Електронний ресурс]. — Режим доступу: https://firebase.google.com/docs
3. Mapbox API Documentation. [Електронний ресурс]. — Режим доступу: https://docs.mapbox.com/android
4. Kotlin Language Documentation. [Електронний ресурс]. — Режим доступу: https://kotlinlang.org/docs/home.html
5. Jetpack Compose Documentation. [Електронний ресурс]. — Режим доступу: https://developer.android.com/jetpack/compose/documentation
6. Turf.js Documentation (Mapbox Turf). [Електронний ресурс]. — Режим доступу: https://turfjs.org/docs
7. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. — Pearson Education, 2017. — 432 p.

Додаток А

Фрагменти реалізації ключових функцій додатку AgroPlan

А.1 Реєстрація користувача

```
suspend fun registerUser(email: String, password: String, name: String): Result<Unit> {
    return try {
        val authResult = firebaseAuth.createUserWithEmailAndPassword(email, password).await()
        val user = authResult.user ?: return Result.failure(Exception("User is null"))
        val profileUpdates = UserProfileChangeRequest.Builder()
            .setDisplayName(name)
            .build()
        user.updateProfile(profileUpdates).await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

А.2 Авторизація користувача

```
suspend fun loginUser(email: String, password: String): Result<Unit> {
    return try {
        firebaseAuth.signInWithEmailAndPassword(email, password).await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

А.3 Створення поля

```
val field = Field(
    name = state.nameText.trim(),
    culture = state.cultureText.trim(),
    fieldGeom = coordinates,
    area = calculateArea(coordinates)
)

fs.collection("Users").document(userId)
    .collection("Fields").document().set(field)
```

А.4 Додавання техніки

```
val machinery = Machinery(
    name = state.name,
    type = state.type,
    speed = state.speed,
    workingWidth = state.workingWidth,
    fuelConsumption = state.fuelConsumption
)

machineRepository.addMachine(machinery)
```

А.5 Генерація маршруту

```
val rawRoute = generatePasses(fieldCoords, workingWidth)
val clippedRoute = rawRoute.filter { TurfMeasurement.inside(it, polygon) }
val plan = Plan(
    name = "Generated Plan",
    trajectoryGeom = clippedRoute,
```

```
    areaProcessed =
TurfMeasurement.area(Polygon.fromLngLats(listOf(clippedRoute))),
    fuelUsed = estimateFuel(clippedRoute),
    numberOfReversals = calculateTurns(clippedRoute),
    totalDistance = TurfMeasurement.length(LineString.fromLngLats(clippedRoute)),
    numberOfPasses = clippedRoute.size
)
```

А.6 Відображення маршруту на карті

```
val feature = Feature.fromGeometry(LineString.fromLngLats(plan.trajectoryGeom))
val featureCollection = FeatureCollection.fromFeatures(listOf(feature))
mapboxMap.getStyle {
    it.getSourceAs<GeoJsonSource>("route-source")?.setGeoJson(featureCollection)
}
```

А.7 Видалення плану

```
fun deletePlan(planId: String) {
    firestore.collection("Users").document(userId)
        .collection("Fields").document(fieldId)
        .collection("Plans").document(planId)
        .delete()
}
```

А.8 Стани ViewModel

```
sealed class PlanUiState {
    object Loading : PlanUiState()
    data class Success(val plans: List<Plan>) : PlanUiState()
    data class Error(val message: String) : PlanUiState()
}
```