МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ANDROID ЗАСТОСУНКУ З ВИКОРИСТАННЯМ FIREBASE СЕРВІСІВ»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Я.Л. Толстенков
(ініціали та прізвище)

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

Рецензент в.о. завідувача кафедри комп'ютерних наук, доцент, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Толстенкову Ярославу Леонідовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка Android застосунку з використанням Firebase сервісів

керівник роботи Кудін Олексій Володимирович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 30.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка Android застосунку з використанням Firebase сервісів.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 05.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	16.05.2023	
2.	Збір вихідних даних.	07.06.2023	
3.	Обробка методичних та теоретичних	23.06.2023	
	джерел.		
4.	Розробка першого та другого розділу.	30.08.2023	
5.	Розробка третього розділу.	27.10.2023	
6.	Оформлення та нормоконтроль	24.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	15.12.2023	

Студент _____ Я.Л. Толстенков
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка Android застосунку з використанням Firebase сервісів»: 54 с., 23 рис., 20 джерел.

РОЗРОБКА ANDROID ЗАСТОСУНКУ, ХМАРНІ СЕРВІСИ, ANDROID STUDIO, FIREBASE, FLUTTER, JAVA, KOTLIN.

Об’єкт дослідження – процес створення мобільних застосунків із використання хмарних сервісів.

Предмет дослідження – бібліотеки розробки Android та Firebase.

Мета роботи: розробка Android застосунку із використанням сервісів Firebase.

Метод дослідження – аналіз.

У кваліфікаційній роботі розглянуто інструменти та засоби для розробки Android застосунків, проаналізовано існуючі сервіси Firebase та Google для роботи географічними даними. Розглянуто платформу Firebase, інструменти, що надаються розробникам. Обґрунтовано використання сервісів Firebase при розробці Android застосунків. Розглянуто методи та засоби інтеграції сервісів Firebase при розробці застосунків. На основі цього матеріалу був виконаний аналіз вимог та проєктування програмного застосунку. Виконана реалізація Android додатку з використанням сервісів Firebase та його тестування.

Структурно робота складається з трьох розділів. У першому розділі виконано технічний огляд засобів розробки. Другий розділ містить проєктування застосунку засобами UML. До третього розділу входять приклади роботи та тестування.

SUMMARY

Master’s qualifying paper «Development of the Android Application Using Firebase Services»: 54 pages, 23 figures, 20 references.

ANDROID APPLICATION DEVELOPMENT, CLOUD SERVICES, ANDROID STUDIO, FIREBASE, FLUTTER, JAVA, KOTLIN.

The object of the study is the process of creating mobile applications using cloud services.

The subject of the study is Android and Firebase development libraries.

The aim of the research is development of an Android application using Firebase services.

Method of research is analysis.

The qualification work explores the tools and resources available for developing Android applications, and analyzes the existing Firebase and Google services for working with geographic data. The study considers the Firebase platform and the tools provided to developers, and justifies the use of Firebase services in the development of Android applications. It also examines the methods and means of integrating Firebase services in application development. Based on this analysis, the software application’s requirements were identified and its design was developed. The Android application was implemented using Firebase services and tested.

Structurally, the work consists of three sections. The first section provides a technical overview of development tools. The second section contains the design of the application using UML. The third section includes examples of work and testing.

ЗМІСТ

ВСТУП

Використання хмарних сервісів в розробці мобільних застосунків має численні переваги, які можуть поліпшити продуктивність, масштабованість та ефективність додатка. Зокрема, хмарні сервіси дозволяють зручний та швидкий спільний доступ до даних між різними пристроями та користувачами. Це особливо важливо для мобільних додатків, які можуть використовувати спільні дані. Також суттєво оптимізується масштабування інфраструктури та ресурсів, що дозволяє мобільним застосункам ефективно працювати навіть у випадку зростання обсягу користувацького трафіку або обробки даних. Інтеграція Firebase сервісів у Android надає зручний та доступний інструментарій для розробки програмних систем. Отже, програмування Android застосунків з поєднанням різних сервісів Firebase є актуальною задачею.

Метою кваліфікаційної роботи є розробка Android застосунку із використанням сервісів Firebase. Для досягнення поставленої мети сформульовано такі задачі:

- проаналізувати існуючи сервіси Firebase та Google для роботи з географічними даними;
- виконати аналіз вимог та проєктування програмного застосунку;
- виконати реалізацію та тестування Android додатка.

Об’єктом дослідження є процес створення мобільних застосунків із використання хмарних сервісів.

Предметом дослідження є бібліотеки розробки Android та Firebase.

Структурно робота складається з трьох розділів. У першому розділі виконано технічний огляд засобів розробки. Другий розділ містить проєктування застосунку засобами UML. До третього розділу входять приклади роботи та тестування.

1 ОГЛЯД ЗАСОБІВ РОЗРОБКИ

1.1 Інструменти розробки додатків для Android

У процесі розробки додатків для Android, одну з головних ролей займає середовище розробки. У ньому проходить процес написання коду, налагодження, його тестування та інше. Офіційним середовищем для розробки додатків для Android є Android Studio [1].

Це середовище підтримує мови програмування Java та Kotlin та C++. Програма доступна на популярних операційних системах таких як Windows, MacOS та Linux. У ньому доступна велика кількість шаблонів, що надають розробнику обрати один з наданих заздалегідь макетів.

Збірка у Android Studio автоматизується за допомоги інструменту Gradle Build Tool. Це швидкий та надійний інструмент автоматизації збірки з відкритим вихідним кодом. Gradle є популярнішою системою збірки для JVM та є системою за замовчуванням для Android, Kotlin та мульти-платформних проєктів, має багату екосистему плагінів спільноти. Він надає високорівневу, декларативну та виразну мову збірки, що спрощує читання та написання логіки збірки. Gradle швидкий, масштабований і може створювати проєкти будь-якого розміру та складності [1].

Також важливим є редактор коду, що спрощує процес розробки. Можна виділити декілька основних характеристик редактору коду у Android Studio: автоматичне завершення коду; інструменти навігації та гарячі клавіши для переходу між файлами, методами та класами; вбудовані інструменти рефакторингу, що дозволяють вносити зміни у структурі коду, назви змінних та інше; наявність шаблонів коду; звернення коду, що дозволяє скрити частину коду; вбудований аналіз коду, що виділяє потенційні проблеми у роботі коду [1].

Android Studio має вбудований емулятор пристроїв. Це дозволяє запускати та тестувати додатки без наявного фізичного пристрою. Слід виділити його

окремі особливості. Конфігурацію пристрою задають самі розробники, для перевірки роботи з різними версіями Android, від ранніх до стабільних релізів, та типами пристроїв. Емулятор має доступні сервіси Google, для тестування залежних від нього функцій, емулює роботу камери пристрою, різних сенсорів, GPS, акселерометр та інше.

Також це середовище розробки має інші, не менш корисні інструменти, що використовуються при розробці:

- відладчик, який надає можливість відстежувати змінні, кроки відладки, відстеження журналів;
- інтеграція з платформою Firebase, що надає можливість використовувати сервіси для автентифікації, роботою з базами даних, хмарних функцій та інших;
- тестування за допомоги спеціальних фреймворків, наприклад JUnit, Espresso та інших.

Для розробки додатків для Android можуть використовуватися такі мови програмування як Java та Kotlin. Порівняємо їх технічні аспекти.

Java є об’єктно-орієнтованою мовою розробки для Android. Відомий тим, що програми написані на ньому можуть бути виконані на будь-якому пристрої, що підтримує віртуальну машину Java [2].

Для розробки Android-додатків використовується Android SDK, що має усі необхідні інструменти, бібліотеки для створення додатків. Виконання додатків проходить на віртуальній машині Dalvik, Android Runtime, що спеціально для цього призначені. Зазвичай, проєкти під Android мають організовану структуру, що відокремлює вихідний код та ресурси. У процесі розробки, спеціалісти мають змогу використовувати Java API, що надається у Android SDK та інші сторонні Java бібліотеки.

Також Java має підтримку роботи з потоками, що дозволяє створювати потоки та працювати з багатопотоковістю. Для відстеження дій користувача використовується обробка подій, будь це жести або натискання на екран тощо. Віртуальна машина Java має збірник «сміття», що автоматично керує пам’яттю,

що знижує ризики витоку пам’яті [2]. Java залишається широко використовуваною мовою для розробки додатків для Android, має широку підтримку, та є стабільним вибором для розробників.

Тепер розглянемо Kotlin. Він оснований на Java і має сумісність з нею, що дає можливість інтегрувати його у вже існуючі проєкти на Java [3]. Так само, об’єктно-орієнтований та зі стандартною типізацією, але на відміну від Java, надає широкі можливості для функціонального програмування, високі порядки функцій, інлайнові функції та інше. Також він являється офіційною мовою програмування для Android та має підтримку від Google. Так як він оснований на Java, так само компілюється у байт-код та виконується на JVM. Структура проєктів на Kotlin має аналогічну структуру із Java. Код на Kotlin є більш компактним у порівнянні з Java та краще читається. Спільними є також керування пам’яттю, але має деякі вдосконалення, наприклад, корутини для ефективного керування асинхронними функціями.

Значною відмінністю є наявність безпечної роботи з нулевими значеннями, що дозволяє значно зменшити вірогідність помилок через нулеві значення і запобігти збою додатку. Kotlin має плагін Android Extensions, який надає можливість взаємодіяти з елементами інтерфейсу користувача через автоматичну генерацію коду. Також відмінністю від Java є DLS-підхід для створення DLS(Domain-Specific Language), через що код стає більш читабельним [3]. З урахуванням усього цього, можна зробити висновки, що обидві мови можна обрати для розробки додатків для Android, але Kotlin є більш привабливим вибором. Він надає більше сучасних та просунутих можливостей, що дозволяють розробникам упроваджувати його код у проєкти на Java та створювати нові.Java та Kotlin відносяться до нативної розробки, націлену на конкретну платформу. Але існують інший підхід до розробки додатків. Такою є розробка додатків за допомоги Flutter. Це фреймворк для розробки крос-платформних додатків, створений компанією Google.

Він використовує мову програмування Dart, що так само як і попередньо розглянуті мови, є об’єктно-орієнтовною мовою зі статичною типізацією.

Особливістю Flutter є те, що після компіляції отримується нативний код для обраних заздалегідь платформ [4]. Це може бути Android, iOS, web та desktop платформи. Це дозволяє зекономити час та ресурси у процесі розробки. За допомоги Dart та Flutter створюються додатки на основі реактивної моделі – парадигма програмування, орієнтована на потоки даних і поширення змін [4].

Віджети – являють собою блоки користувацького інтерфейсу та логіки додатку, є основною концепцією Flutter. Він надає можливість використовувати бібліотеку власних віджетів так і створювати їх самостійно. Однією з зручних особливостей фреймворку є наявність «гарячого перезавантаження» додатку, що дозволяє вносити зміни та бачити їх у додатку без необхідності повністю перезавантажувати його. За допомоги власного движка відтворення, Flutter дозволяє відтворювати користувацькі інтерфейси однаково на всіх платформах [5].

Пакетний менеджер для керування залежностями у Flutter – Pub. Він дозволяє розробникам інтегрувати інші бібліотеки та пакети та тим самим розширювати його функціональність. Використання модулів та сенсорів пристроїв досягається за допомоги доступу до платформних API.Можна зробити висновки, що Flutter є потужнім інструментом для створення багатоплатформних додатків з високою продуктивністю та однаковим інтерфейсом для всіх платформ.При огляді Android Studio згадувався Firebase. Це платформа для розробки мобільних та вебдодатків, що дає змогу створювати високоякісні додатки, при цьому не маючи власних серверів. Далі розглянемо їх більш детально, визначимо їх переваги та недоліки при використанні їх у розробці мобільних додатків.

1.2 Переваги та недоліки Firebase

У кожної платформи є свої переваги та недоліки, тому необхідно їх визначити перед тим як задіяти її у своєму проєкті.

Це необхідно для того, щоб впевнитися, що у подальшому не виникне незручностей при розробці. Так само це стосується й Firebase сервісів. Виокремимо переваги та недоліки та розберемо їх детальніше.

До переваг використання сервісів Firebase можна віднести:

– наявність безкоштовного плану;
– швидкість розробки;
– наскрізна платформа для розробки додатків;
– працює на платформі Google;
– розробники можуть зосередитись на клієнтській частині;
– не потребує використання серверу;
– закладень можливості машинного навчання;
– генерація трафіку для вашого додатку;
– моніторинг помилок;
– безпека.

Для початку користування сервісами Firebase не треба витрачати ресурси, увійти в систему можна зразу зі свого Google аккаунту. На платформі є два тарифних плани: Spark та Blaze. Тариф Spark є безкоштовним. Для початку роботи з сервісами його достатньо, та якщо вимоги при розробці будуть змінюватися – завжди є можливість обрати план Blaze. Це є однією з причин популярності Firebase.

Розробники мають можливість почати працювати з платформою без затрат. Зокрема, план Spark має безкоштовний ліміт на роботу з базою даних. Також слід зауважити, що тариф Blaze не буде потребувати оплату за безкоштовні ліміти плану Spark, і якщо в якомусь з сервісів за них не вийшли – вони не потребують оплати.

Щодо швидкості розробки, використання сервісів Firebase може дозволити скоротити час на розробку та виходу додатку на ринок. Зазвичай, для того щоб обслуговувати бази даних та серверних служб, необхідний доступ до серверу. Тому при розробці додатків, навіть у невеликих проєктах, необхідно мати як мінімум одного frontend та backend розробника. Через це можуть виникати

проблеми та помилки при розробці, через які погіршується якість додатків, зростає час на розробку що впливає на вартість та складність розробки. Використання сервісів Firebase вирішує цю проблему, надаючи frontend розробникам самостійно керувати роботою та скорочувати час для її виконання. Також платформа має велику кількість готових сервісів, які створені для того щоб уникнути шаблонної роботи і не витрачати час на написання серверної частини. Серед таких сервісів можна виділити систему реєстрації, аналітику та повідомлення.

Google Firebase надає розробникам необхідні продукти, які можуть знадобитися у процесі розробки. Платформа надає інструменти, що можуть бути застосовані на всіх етапах розробки додатків, такі як: створення, випуск, моніторинг. Також має інструменти для утримання та залучення користувачів.

Сервіси працюють на базі Google, з моменту їх придбання компанією вони пройшли ряд модернізацій, наразі є модернізованими та надійними. Вони використовують усі переваги хмарних та інших сервісів Google. Наразі Firebase є частиною хмарної платформи Google, добре поєднуються з їх іншими хмарними сервісами та сторонніми.

Багато розробників обирають саме Firebase сервіси, бо завдяки їм з’являється можливість зосередитись на створенні коду та фронтенду додатків. Замінюючи сервісами написання власного коду для бекенду скорочуються терміни розробки додатків.

Таким чином вони є кращими як для розробників так і для замовників. Завдяки цьому розробники та компанії можуть стандартизувати середу розробки бекенду. Використання шаблонів для бекенду дозволяють розробникам, що роблять фронтенд, виконувати більше дій.

Використання Firebase не потребує власних серверів, тому необхідність слідкувати на ними відсутня. Сервіси поставляються з архітектурою, в якій немає серверів, оплата за використання потребується тільки при використанні сервісів, на основі запитів. Тому немає необхідності керувати інфраструктурою серверів та слідкувати за нею. Це допомагає уникнути проблем з масштабуванням,

наприклад, кластеру баз даних, та робіт з оптимізації для безперервної роботи при великих навантаженнях.

Зараз активно розвивається машинне навчання, за деякими оцінками, близько тридцяти відсотків компаній будуть використовувати його в своїх процесах. Firebase також має інструменти для застосування технології машинного навчання. Її можна використовувати Android та iOS розробникам.

Платформа має комплект машинного навчання зі зрозумілим та легким у використанні API для використання на мобільних платформах. Її можна використовувати для розпізнавання облич, тексту, скануванні штрих-кодів, маркуванні зображень тощо. Залежно від потреб у розробці, є можливість обрати між хмарними або вбудованими API-інтерфейсами.

Щоб спростити потенційним користувачам знайти додаток в Google пошуку, Firebase спрощує індексацію додатків. Також за рахунок індексації може бути покращено ранжування додатків, що допомагає швидше залучити нових користувачів які зможуть їх встановити. Для доступу до контенту додатку можливо у Android Instant App.

Один з інструментів для пошуку проблем та їх виправлення є Firebase Crashlytics. Завдяки ньому можливо відслідковувати як фатальні так і критичні помилки. Генерація звітів будується на тому, як помилки впливають на використання додатку користувачем.

За рахунок регулярного резервного копіювання, платформа забезпечує безпеку та доступність даних. Firebase має функцію автоматичного резервного копіювання, що забезпечує захист від втрати даних. У плані Blaze можливо автоматичне резервне копіювання для бази даних Firebase Realtime.

До недоліків використання сервісів Firebase можна віднести:

- не має відкритого вихідного коду;
- залежність від Постачальника;
- firebase не присутній у багатьох країнах;
- доступні тільки бази даних NoSQL;
- повільні запити;

- не всі служби працюють безкоштовно на базовому тарифі;
- це не дешева платформа зі складно прогнозованою ціною;
- працює тільки в Google Cloud;
- виділені Сервери та Корпоративна техпідтримка відсутні;
- відсутні GraphQL API.

У зв’язку з тим, що Firebase не є продуктом з відкритим вихідним кодом для розробки додатків, він не підходить для деяких розробників. Через те, що він є закритим, користувачі не можуть змінювати вихідний код. Через це багато розробників використовують інші платформи, в яких немає такого обмеження. Проте багато бібліотек та SDK доступні на GitHub. Це заважає спільноті робити платформу краще, гнучкіше та використовувати незалежно від хостингу розробників.

Факт того, що ви повністю залежите від одного постачальника є ще одним стримуючим фактором для розробників. Це суттєва проблема, яка зупиняє частину розробників від вибору платформи. Відсутність доступу до вихідного коду може не влаштовувати деяких розробників. Для великих додатків перехід до інших постачальників є складним завданням, тому що доведеться заново розробляти серверну частину. Також проблемою є те, що Firebase працює на піддомені Google, який заблокований у багатьох країнах. Пошукова система Google заблокована та не може бути використана з іншими сервісами, наприклад, у Китаї.

Платформа пропонує тільки NoSQL бази даних, Firebase, Firestore и Firebase Realtime Database. Також користувачі сервісу, які використовують NoSQL базу зіштовхуються із проблемою реалізації складних запитів. Не дивлячись на те, що у Cloud Firestore порівняно з RTDB внесли значні покращення, виконання складних запитів залишається проблемою. Через деякі обмеження бази даних Firebase підходять не для всіх варіантів використання. До таких обмежень відносяться: запити знижують продуктивність; гнучкість запитів; максимальна кількість одночасних підключень – один мільйон; обмеження розміру документа один MiB; максимальний розмір запиту за API

десять МіВ; відсутні власні запити агрегації; частота запису має обмеження один раз на секунду.

У безкоштовному плані Spark багато доступних сервісів, проте деякі з них залишаються недоступними, наприклад машинне навчання та хмарні функції. Для того, щоб користуватися ними, необхідний план Blaze. Не дивлячись на те, що у безкоштовному тарифі є можливість використання хмарних функцій, після перевищення кількості безкоштовних запитів буде зніматися плата.

Також оплата за використання сервісів заснована на тому, як реально буде використовуватися платформа, та немає засобів для фіксування підсумкової вартості. Тому використання Firebase сервісів може виявитись дорогим. Послуги надаються по моделі «інфраструктура як послуга» та відсутні фіксовані тарифні плани. Оцінити підсумкову вартість може бути важко, особливо коли неможливо передбачити масштабування.

Також відсутня можливість використання Firebase сервісів з іншими хмарними постачальниками послуг. Це пов’язано з тим, що платформа є частиною Google та її інфраструктура працює на Google Cloud. Firebase обмежує гнучкість з точки зору вибору хостингу. Firebase не надає доступ на рівні сервера, і за необхідності індивідуально налаштувати параметри сервера можуть виникнути проблеми. Використання платформи можливо тільки з використанням її безсерверної архітектури. Це є ще одним обмеженням, що забезпечую меншу гнучкість та у деяких випадках є причиною для використання інших постачальників послуг. Це може бути зручно, коли навантаження на сервер постійно змінюється. Але якщо навантаження на сервер стабільне, але буде необхідний доступ на серверному рівні, це буде неможливо.

1.3 Огляд Firebase сервісів

Для збереження та синхронізації даних клієнтів у реальному часі використовується NoSQL база даних Firebase Realtime.

Розміщається ця база даних у хмарі у форматі JSON. Дані синхронізуються між усіма клієнтами та залишаться доступними при виключенні додатку.

Незалежно від платформи, наприклад, Apple чи Android, всі клієнти використовують єдиний екземпляр бази даних та автоматично отримують оновлення з новими даними. Синхронізація даних використовуються замість HTTP-запитів та дозволяє швидко актуалізувати дані, коли вони змінюються. Через те, що Firebase Realtime зберігає дані користувачів на диск, додаток залишається функціональним навіть в автономному режимі.

Синхронізація даних з сервером виконається як тільки виконається підключення, отримуються усі пропущені зміни. Завдяки тому, що база знаходиться у хмарі, немає необхідності у власному сервері додатків, клієнти отримують доступ до бази у самому мобільному додатку або через веббраузер.

Також сервіс має інструменти для безпеки та перевірки даних. Правила виконуються при зчитуванні або запису даних, будуються на основі виразів. Також Firebase Realtime працює з іншими сервісами, наприклад, Firebase Authentication [6].

Cloud Firestore так само синхронізує дані між клієнтськими додатками. Але також надає можливості безшовної інтеграції з іншими сервісами Firebase та Google Cloud. Сервіс надає можливість зберігати дані у організовані колекції, які можуть містити складні вкладені об’єкти у додатку до вкладених колекцій.

Є можливість створювати більш гнучкі запити, отримувати окремі документи або цілі колекції. Зв’язні фільтри, фільтрація та сортування можуть застосовуватися у запитах. У додатку, Cloud Firestore має автоматичну реплікацію даних у декількох регіонах, підтримку реальних транзакцій, атомарні пакетні операції [7].

Firebase Machine Learning – це набір інструментів, що дозволяє використовувати машинне навчання у своїх додатках для вирішення задач з додатками Android та Apple. Використання цього сервісу не потребує від розробника бути спеціалістом у машинному навчанні. Для того, щоб додати необхідний функціонал до додатку не обов’язково мати глибоких знань в області

нейронних мереж. Але для досвідчених розробників машинного навчання Firebase ML надає API для використання власних моделей. Для цього достатньо розгорнути свою модель у Firebase. Сервіс динамічно надає користувачам останню версію моделі та не потребує для цього втручання розробника.

За допомоги альфа та бета тестування можливо проводити експерименти та знаходити більш ефективні моделі. А з Remote Config можливо надавати різним сегментам користувачів різні моделі. Залежно від вимог, використання моделі може відбуватися як на мобільному пристрої так і у хмарі. Для цього існують хмарний API та пристрою. Для більш точної роботи моделі краще використовувати хмару, вона має більше ресурсів для обчислень ніж пристрій. Але якщо мова йде про роботу у реальному часі, наприклад, обробка відео, краще використовувати мобільний API.

Користувацькі моделі не потребують підключення до мережі та працюють досить швидко щоб обробляти кадри відео у режимі реального часу. Firebase ML надає можливість завантажувати моделі на сервери сервісу для розгортання на пристроях користувачів. За запитом додатку буде завантажено модель на пристрій. Це дозволяє змінювати модель машинного навчання на нову без потреби повторної публікації та зберегти початковий розмір додатку [8].

До набору готових до використання моделей на пристроях входять:

- розпізнавання тексту;
- маркування зображення;
- виявлення та відстеження об’єктів;
- розпізнавання облич і трасування контурів;
- сканування штрих-коду;
- ідентифікація мови;
- переклад;
- розумна відповідь;
- хмарні функції.

Cloud Functions for Firebase – це платформа, що надає можливість виконувати серверний код автоматично, у відповідь на події, що викликаються

Firebase або запитами. У хмарі Google зберігається код у вигляді JavaScript або TypeScript та виконується у керованому середовищі. Через відсутність серверу, зникає необхідність у керуванні власним сервером та масштабуванні. Події для виконання коду можуть бути запущені через функції Firebase, Google Cloud, тригерами аутентифікації або сховища.

Завдяки сервісу мінімізується шаблонний код, спрощується використання Firebase та Google Cloud всередині функції. Код можливо розгорнути завдяки консолі Firebase. При розгортанні коду, сервіс автоматично виділяє необхідні ресурси для його роботи. Використання хмарних функцій виключає можливість втручання у логіку додатку на боці клієнта. Вони повністю ізольовані від клієнта, що забезпечує безпеку та конфіденційність [9].

Для того, щоб безпечно зберігати дані користувачів у хмарі та забезпечувати однаковий досвід використання на різних пристроях необхідно знати особистість користувача. Для цього є сервіс Firebase Authentication, який має прості у використанні інструменти та бібліотеки користувацького інтерфейсу для автентифікації у додатку. Є підтримка автентифікації з паролем, телефоном, з допомоги популярних постачальників таких як Google, Facebook та інших.

Він використовує стандарти OAuth 2.0 та OpenID Connect. Це дозволяє легко інтегрувати його з бекендом, що налаштовується та з іншими сервісами Firebase. Також є можливість додатково використовувати багатофакторну автентифікацію, функцію блокування, ведення аудиту та інших, за умови оновлення до Firebase Authentication with Identity Platform. Можна використовувати FirebaseUI у якості рішення для автентифікації та інструменти Firebase Authentication, для того щоб обрати методи для входу. Слід зазначити, що FirebaseUI має відкритий вихідний код. Це дозволяє налаштувати його для того, щоб він відповідав візуальному стилю вашого застосунку. Має інструменти не тільки для входу, а також для реєстрації та відновлення доступу [10].

Firebase Cloud Messaging – це сервіс, який дозволяє безкоштовний обміну повідомленнями з пристроями клієнтів. За допомоги нього можна відправляти

повідомлення, утримувати користувачів та залучувати повторно. Повідомлення можливо направляти для окремого пристрою та групи пристроїв. Також сервіс надає можливість відправляти підтвердження, чати, інші повідомлення на сервер [11].

Для зберігання користувацького контенту та обслуговувати його існує хмарне сховище для Firebase. Так ми можемо зберігати у сховищі, наприклад, відео або фото. Сервіс забезпечує безпеку для скачування та завантаження файлів.

Сервіс дозволяє зберігати аудіо, відео та інші типи користувацьких даних. Існують інструменти, що дозволяють розробникам керувати сегментами, створювати URL-адреси скачування, використовувати API Google Cloud Storage для доступу до власних файлів. Також, користувачі зможуть скачувати та завантажувати незалежно від якості мережі. При виникненні розриву підключення, після повторного підключення файли будуть завантажуватися або скачуватися з місця зупинки. API-інтерфейси Google Cloud Storage дозволяють виконувати обробку на стороні серверу, наприклад, транскодування відео. Масштабування хмарного сховища проходить автоматично [12].

Відслідковувати збої у роботі додатку можливо з Firebase Crashlytics. Сервіс займається створенням звітів про збої в режимі реального часу, допомагає відслідкувати проблеми, які впливають на стабільність та якість додатку, та вирішити їх. Завдяки тому, що він вміє групувати збої та виділяти умови їх виникнення, пошук та вирішення проблеми займає менше часу.

Також сервіс інтегрований з Analytics, помилки в ньому фіксуються як події, які можна проглядати у списку з іншими подіями для пошуку взаємозв’язків. Сповіщення про нові проблеми, регресивні, приходять у режимі реального часу [13].

Для того, щоб мати уяву про використання додатку, залученості користувачів, існує сервіс Analytics. Він інтегрується з усіма Firebase функціями, надає необмежені звіти для п’ятисот різних подій, що визначаються інструментами Firebase. Завдяки звітам можливо зрозуміти, як користувачі

взаємодіють з додатком. Є можливість поділити користувачів на сегменти, та залежно від потреб використовувати з іншими функціями Firebase [14].

Сервіс, Firebase Performance Monitoring, надає представлення про проблеми з продуктивністю додатку. Інструменти моніторингу використовуються для збору даних продуктивності додатку, на основі отриманих даних проводиться аналіз у консолі Firebase.

Завдяки тому, що сервіс працює у режимі реального часу, можна краще зрозуміти, в якій частині необхідно зробити покращення. Деякі показники відстежуються автоматично, наприклад час запуску додатку. Для цього вже є готові рішення, писати код не потребується. Також є можливість використовувати користувацькі трасування коду. Це необхідно, коли треба зафіксувати продуктивність в окремих ситуаціях [15].

Firebase Test Lab – це хмарний сервіс, що дозволяє тестувати додатки на різних конфігураціях пристроїв. Це необхідно для того, щоб розуміти, як додаток буде себе вести при реальному користуванні. Тести можна робити на пристроях Android та Apple. Проходять вони на пристроях, що працюють у центрі обробки даних Google. Сервіс інтегрований у консоль Firebase, Android Studio, командний рядок gcloud [16].

Для поширення додатків серед тестувальників є сервіс Firebase App Distribution. Він спрощує поширення додатків, дозволяє швидко розміщати їх на пристроях тестувальників, отримувати відгуки раніше та частіше.

У парі з Crashlytics дозволяє отримувати показники стабільності для усіх збірок. В одному місці є можливість керувати версіями додатків під iOS та Android. Автоматизувати поширення можливо інтегрувавши інтерфейс командного рядка у завдання безперервної інтеграції. Команди тестування об’єднуються у групи для керування. Додаються нові тестувальники завдяки електронним запрошенням [17].

Firebase Remote Config – хмарний сервіс, який дає можливість змінювати поведінку та зовнішній вигляд вашого додатку, при цьому не публікуючи оновлення додатку. У сервісі створюються значення за замовчуванням у додатку.

Вони керують зовнішнім виглядом та поведінкою додатку. Надалі можна у консолі змінювати ці параметри для всіх користувачів, або для якоїсь окремої групи [18].

Для оптимізації взаємодії з додатком, спрощування запуску, аналізу та масштабування маркетингових експериментів використовується сервіс Firebase A/B Testing. Працює він на базі Google Optimize, дає розробникам можливість тестувати зміни в інтерфейсі користувача, функціях, компаніях взаємодії, щоб побачити як саме вони впливають на ключові показники до їх впровадження.

Працює сервіс з Firebase Cloud Messaging, що дозволяє тестувати різні маркетингові повідомлення, та з допомогою Firebase Remote Config тестувати зміни в додатку.

Створюючи різні варіанти експерименту, змінюючи зовнішній вигляд додатку, ми можемо перевірити та обрати більш ефективний результат. Не треба впроваджувати зміни одразу, є можливість впевнитись у необхідності нововведень за результатами тестування.

Використовуючи дані з Google Analytics, ми можемо налаштувати тестування для окремої версії додатку, платформи, мови або демографічних даних [19].

Залучення активних користувачів вашого додатку можливе з використанням сервісу Firebase In-App Messaging. Для цього залучаються контекстні повідомлення, що спонукають користувача використовувати ключові функції додатку. Воно може бути в різному вигляді, наприклад банеру, карточки, модального вікна або зображення.

Налаштовувати тригери, після яких будуть з’являтися повідомлення та спонукати користувача до певних дій: продивитися відео, купити предмет, підписатися тощо. Сервіс надсилає повідомлення, коли вони дійсно потрібні, поки користувач знаходиться у додатку. Firebase In-App Messaging надає можливість налаштовувати зовнішній вигляд, стиль, вміст повідомлень, пропонувати користувачам оновити додаток або залучати, наприклад, до розпродажі [20].

1.4 Порівняння з іншими аналогами

Слід зауважити, що платформа Firebase не є унікальною, та існують інші сервіси, що надають аналогічні можливості. Порівняємо її з Amazon Web Services. Це також хмарна платформа, власником якої є Amazon. Так само як і Firebase, вона надає широкий спектр хмарних послуг, і поставляються вони як онлайн сервіси, що дозволяє отримувати необхідні функції без витрати ресурсів на власну апаратуру та інфраструктуру.

Більшість основних сервісів є аналогічними до Firebase, але є ті, аналогів яких немає. Одним з таких є Amazon Elastic Compute Cloud, що надає віртуальні сервери у хмарі для масштабування обчислювальних ресурсів. Сервіс для керування, розгортання та масштабування реляційних баз даних – Amazon Relational Database Service та інші.

Перевагами сервісів AWS перед Firebase є більш широкий вибір послуг, тим самим роблячи його більш привабливим для розробників. Висока гнучкість налаштування та масштабованість, що дозволяє створювати інфраструктуру відповідно до вимог проєкту. Має високий рівень доступності та надійності завдяки розподіленим центрам обробки даних.

Але так само є й недоліки. AWS є більш складним для початківців, потребує більшого часу для налаштування та вивчення можливостей. Не усі сервіси підтримують роботу у реальному часі, аніж у Firebase.

Таким чином, можна зробити висновок, що вибір між цими платформами залежить від багатьох факторів та вимог до майбутнього продукту. Але Firebase є більш привабливим для початківців та невеликих проєктів через свою простоту та швидкість.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Постановка задачі

Необхідно спроєктувати та розробити додаток для сервісу доставки. Він повинен надавати можливість користувачам, кур’єрам, брати замовлення та виконувати доставку. Для адміністратора – надавати можливість слідкувати за виконанням замовлень.

Замовлення для виконання будуть поступати до бази даних з вже існуючою системи управління замовленнями або адміністратором через застосунок. Завдання, які будуть виконуватися користувачем при виконанні замовлення наступні: отримати посилку у точці видачі, дістатися точки доставки та передати посилку отримувачу.

Необхідно, щоб доступ до додатку надавався тільки авторизованим користувачам, відповідно необхідно буде розробити механізми входу та реєстрації у системі. Також відомо, що для роботи додатку не будуть надані обчислювальні ресурси у вигляді серверу, тому необхідно буде використовувати сервіси Firebase для зберігання даних про замовлення та авторизації користувачів. Адміністратор повинен мати можливість переглядати останнє місцеположення користувача на мапі, та його зміни у реальному часі.

2.2 Визначення варіантів використання

Для того, щоб описати систему на концептуальному рівні, використовується діаграма варіантів використання. Вона дозволяє відобразити відносини між акторами та прецедентами, які є складовими частинами моделі прецедентів. Для цього було зроблено діаграму варіантів використання для нашої системи. Більш детально можна ознайомитись з нею на рисунку 2.1.

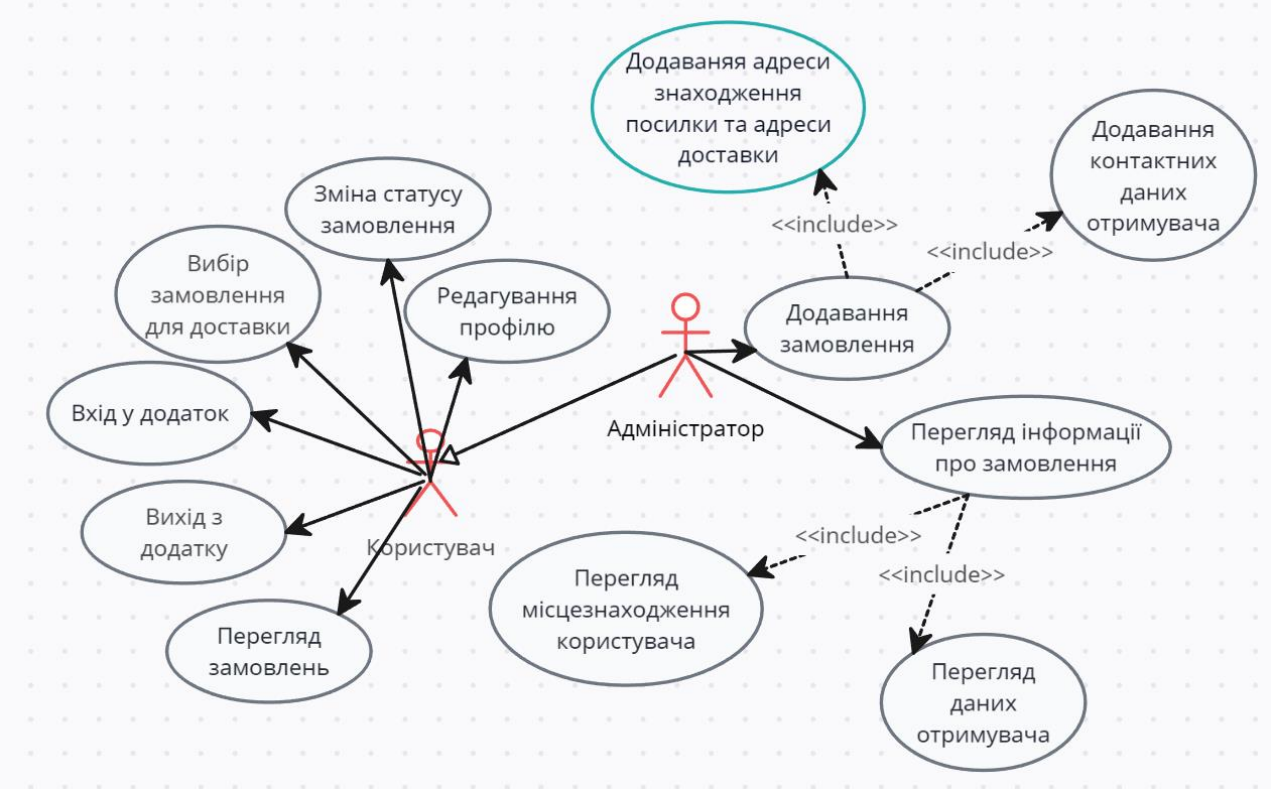


Рисунок 2.1 – Діаграма варіантів використання

На діаграмі можна побачити двох акторів, Адміністратор та Користувач. Перший, є користувачем з розширеними правами. Другий, сам користувач, який відображає спільний з адміністратором функціонал.

Основними для користувача можна виділити: перегляд замовлень та вибір замовлення для доставки. Вони будуть доступні після авторизації користувача у системі. З ними буде проходити основна робота. Для адміністратора це додавання замовлення та перегляд інформації для замовлення. Розпишемо декілька прецедентів.

Прецедент «Вхід у додаток».

Призначення: даний варіант використання дозволяє користувачу отримати доступ до системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач відкриває застосунок.

Виняткова ситуація: якщо при виконанні входу була помилка у пошті або паролі – виведеться помилка.

Передумова: перед початком виконання даного варіанта використання користувач повинен бути зареєстрований у системі.

Прецедент «Вихід з додатку».

Призначення: даний варіант використання дозволяє користувачу вийти з системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку виходу з застосунку.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід у застосунок.

Прецедент «Перегляд замовлень».

Призначення: даний варіант використання дозволяє користувачу переглянути замовлення, що доступні для доставки.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач успішно виконав вхід у систему та знаходиться на головній сторінці, де відображається список з доступними для виконання замовленнями.

Альтернативний потік подій: якщо доступних до виконання замовлень немає – виводиться відповідний допис.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід у застосунок та знаходитись на головній сторінці.

Прецедент «Додавання замовлення».

Призначення: даний варіант використання дозволяє адміністратору додати до системи нове замовлення.

Основний потік подій: даний варіант використання починає виконуватися, коли адміністратор натискає кнопку «Додати». Відкривається форма, у яку він повинен ввести необхідні дані.

Виняткова ситуація 1: у формі присутні порожні поля та була натиснута кнопка для додавання – буде відображено повідомлення.

Виняткова ситуація 2: користувач натиснув кнопку скасування – система поверне користувача на головну.

Передумова: перед початком виконання даного варіанта використання користувач повинен виконати вхід у застосунок та знаходитись на головній сторінці.

2.2 Візуалізація архітектури розгортання системи

Так як у системі не передбачено використання власного серверу для реалізації автентифікації та збереження даних, будемо використовувати описані у першому розділі сервіси, а саме Firebase Authentication та Firebase Reatime Database. Взаємодія додатку з ними буде через інтернет.

Для реалізації відображення будемо використовувати сервіс Google Maps, який має необхідні інструменти для відображення мапи, маркерів на ній та побудові маршрутів.

На діаграмі розгортання (див. рис. 2.2) ми можемо побачити, як буде відбуватися взаємодія з додатком користувача, адміністратора, firebase та google maps.

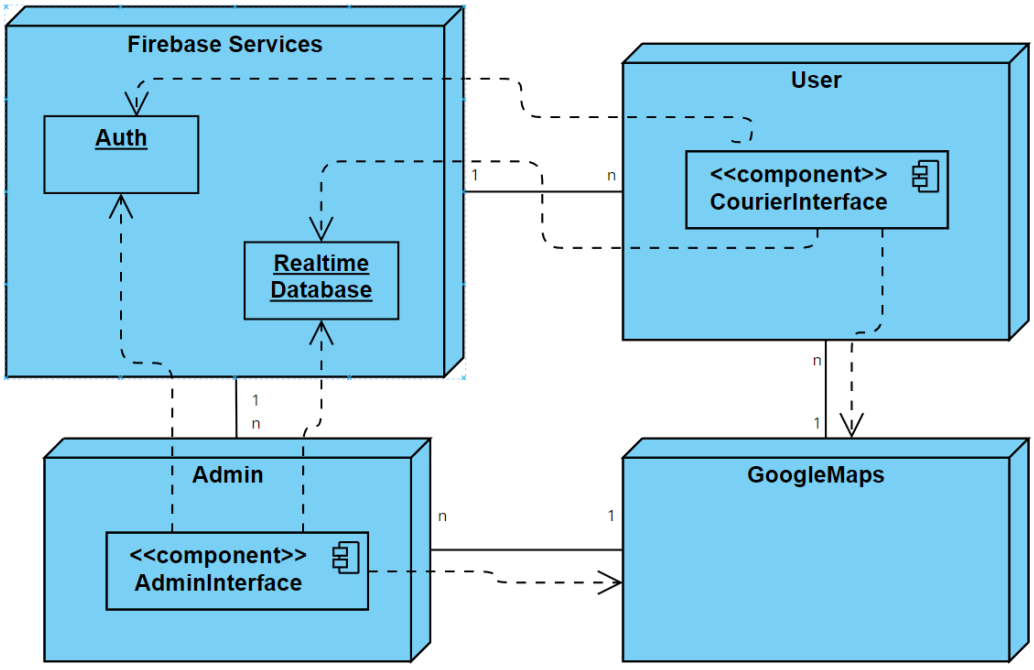


Рисунок 2.2 – Діаграма розгортання

2.3 Проєктування класів застосунку

Структура майбутнього додатку буде складатися з трьох частин:

– services;
– presentation;
– repository.

У частині services будуть знаходитися класи, які будуть надавати необхідний функціонал з використанням сервісів аутентифікації та роботи з мапою.

Необхідно буде створити наступні класи:

– FirebaseServices, який буде сінглтоном, у ньому буде описуватися використання сервісу Firebase Authentication для реалізації входу у застосунок;
– GoogleMap, що буде відповідати за відображення мапи, виставлення необхідних маркерів на мапі та будування рекомендованого маршруту від точки видачі до адреси доставки.

Дані класи можна побачити на діаграмі класів №1, що зображена на рисунку 2.3.

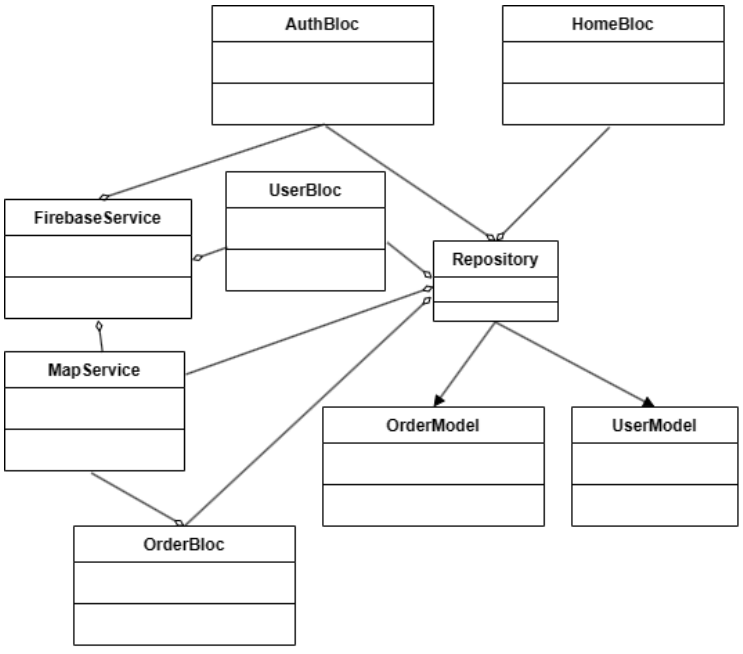


Рисунок 2.3 – Діаграма класів №1

Клас FirebaseServices повинен мати наступні методи:

- onLogin, який буде отримувати логін та пароль користувача та використовуватиме Firebase Authentication для виконання входу до системи, результатом виконання якого буде запис користувача, або повідомлення про помилку;
- onRegister, буде отримувати необхідні дані для реєстрації, та виконувати вхід у систему при успішній реєстрації, або помилку;
- getCurrentUserId, буде повертати значення унікального ідентифікатора користувача;
- logout, буде реалізовувати вихід користувача з додатку.

Клас GoogleMap повинен мати методи:

- getLocationUpdate, буде відслідковувати зміну позиції користувача та оновлювати її у випадку зміни;
- getUserLocation, буде оновлювати інформацію про переміщення користувача з бази даних;
- getData, який буде отримувати дані замовлення, які необхідні для малювання маркерів та рекомендованого маршруту доставки.

У частині presentation будуть знаходитися класи, що будуть реалізовувати графічний інтерфейс користувача та логіку його роботи. Наш додаток матиме чотири екрани:

- екран для автентифікації, що буде відображати форму для входу, за замовченням, та реєстрації, у випадку, якщо користувач не буде зареєстрований у системі;
- головний екран, який буде відображати список замовлень, які доступні для виконання;
- екран замовлення, який буде відображати повну інформацію про замовлення та відображатиме на мапі рекомендований маршрут доставки;
- екран користувача, що буде містити інформацію про нього, необхідну для адміністратора.

Для екрану автентифікації необхідно буде створити класи, опишемо їх

далі. На діаграмі послідовностей, що зображена на рисунку 2.4, можна детальніше оглянути взаємодію об’єктів, при сценарію входу у систему

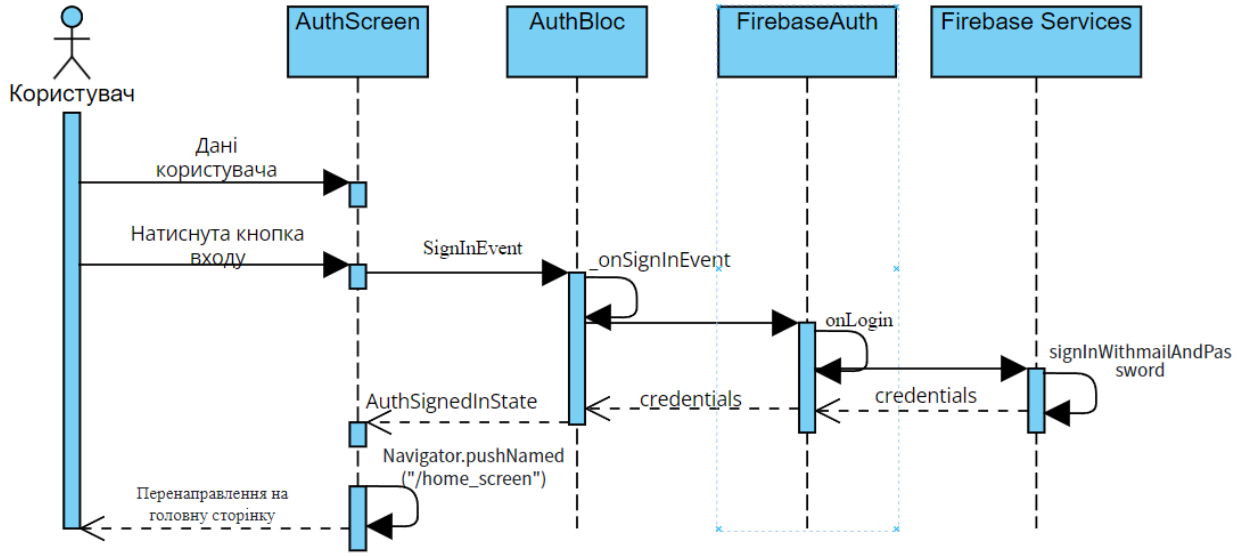


Рисунок 2.4 – Діаграма послідовностей

AuthBloc, що успадковується від класу Bloc. Він буде містити необхідну логіку для реалізації представлення. Діаграма класу зображена на рисунку 2.5.

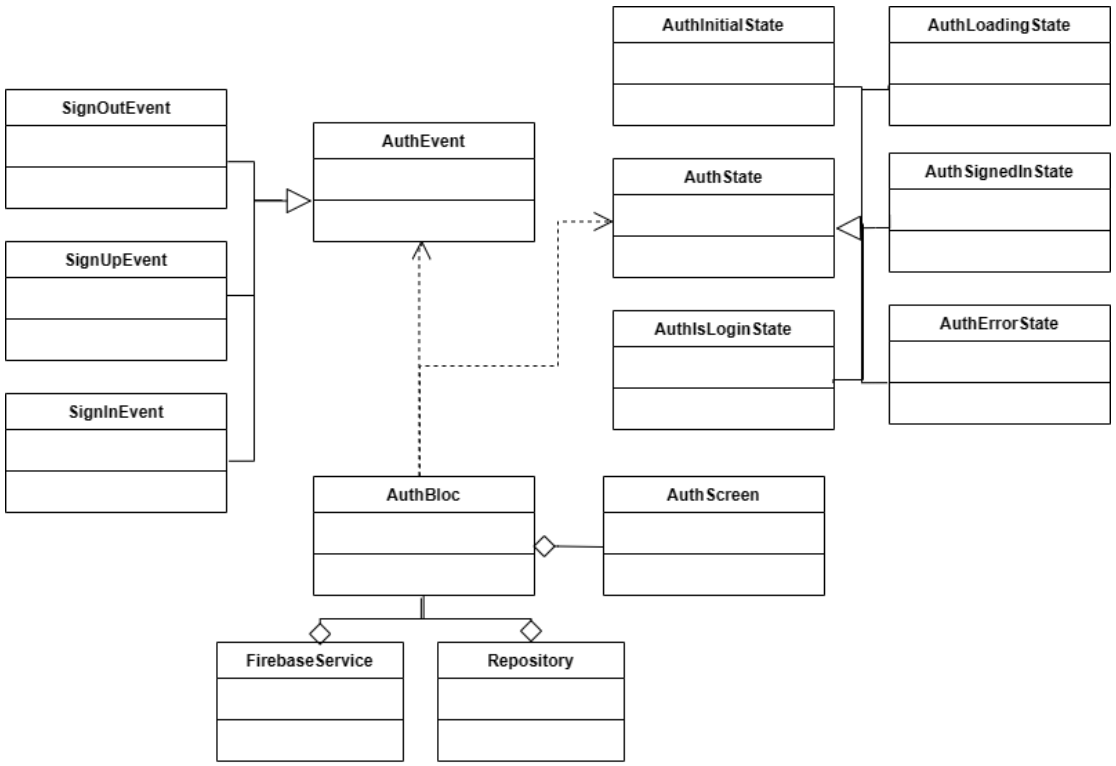


Рисунок 2.5 – Діаграма класів №2

Для класів подій необхідно створити абстрактний клас AuthEvent, від якого вони будуть успадковуватися, а саме:

- SignInEvent, клас, який буде подією, яка буде ініціювати вхід у додаток (прийматиме пошту та пароль користувача);
- SignUpEvent, клас, який являє подію, що буде сповіщати про необхідність запуску процесу реєстрації у додатку (буде приймати пошту, пароль, ім’я та номер телефону користувача);
- SignOutEvent, подія, яка сигналізує про необхідність запуску процесу виходу користувача з додатку;
- UpdateIsLoginEvent, подія, яка буде ініціювати зміну форми, що буде відображатися користувачу.

Для класів, що будуть реалізовувати стани, необхідно створити абстрактний клас AuthState. Для нашого екрану автентифікації знадобляться наступні класи:

- AuthInitialState, він буде являти початковий стан;
- AuthLoadingState, клас, що зберігатиме стан при процесі автентифікації у додатку, якщо вона ще не завершилася;
- AuthSignedInState, буде зберігати стан користувача при успішному проходження автентифікації у системі;
- AuthErrorState, клас, що зберігатиме помилку, яка виникла при проходженні автентифікації у системі;
- AuthIsLoginState, клас, який зберігатиме значення, що буде сигналізувати про необхідність у зміні форми.

Для головного екрану, необхідно створити клас HomeBloc, у якому буде описана логіка представлення.

За аналогією з попереднім класом, необхідно буде створити абстрактний клас HomeEvent, його будуть реалізовувати наступні класи:

- HomeOrdersLoadingEvent, подія, що буде сигналізувати про завантаження даних про замовлення;
- HomeOrdersLoadedEvent, сигналізуватиме, що необхідні дані були

завантажені та готові до відображення.

Так само, як для AuthBloc, необхідно буде створити абстрактний клас HomeState, який будуть реалізовувати наступні класи:

– HomeOrdersState, який зберігатиме завантажені замовлення при успішному отриманні;
– HomeOrdersLoadingState, який буде зберігати стан при завантажені даних з бази;
– HomeOrdersErrorState, що буде зберігати помилку, яка виникне при завантажені.

Для екрану відображення детальної інформації про замовлення також необхідно буде створити клас, OrderBloc. Сценарій роботи з замовленнями показаний на діаграмі діяльності, що відображена на рисунку 2.6.

Як з класами вище, необхідно створити абстрактний клас OrderEvent, який будуть реалізовувати наступні класи:

– OrderLoadingEvent, що буде сигналізувати про загрузку детальної інформації про замовлення з бази даних;
– OrderLoadedEvent, подія, що буде виконуватися після завантаження замовлення;
– OrderTakededEvent, подія, що буде виконуватися того, як користувач візьме замовлення у роботу;
– OrderDeliverededEvent, подія, що буде виконуватися того, як користувач завершить доставку замовлення.

Також необхідно буде створити абстрактний клас OrderState та класи, що будуть успадковуватися від нього, а саме:

– OrderLoadingState, клас, що буде зберігати стан при завантажені даних;
– OrderLoadedState, стан, який зберігатиме отриману інформацію про замовлення;
– OrderErrorState, зберігатиме помилку, що виникла при спробі завантаження даних з бази.

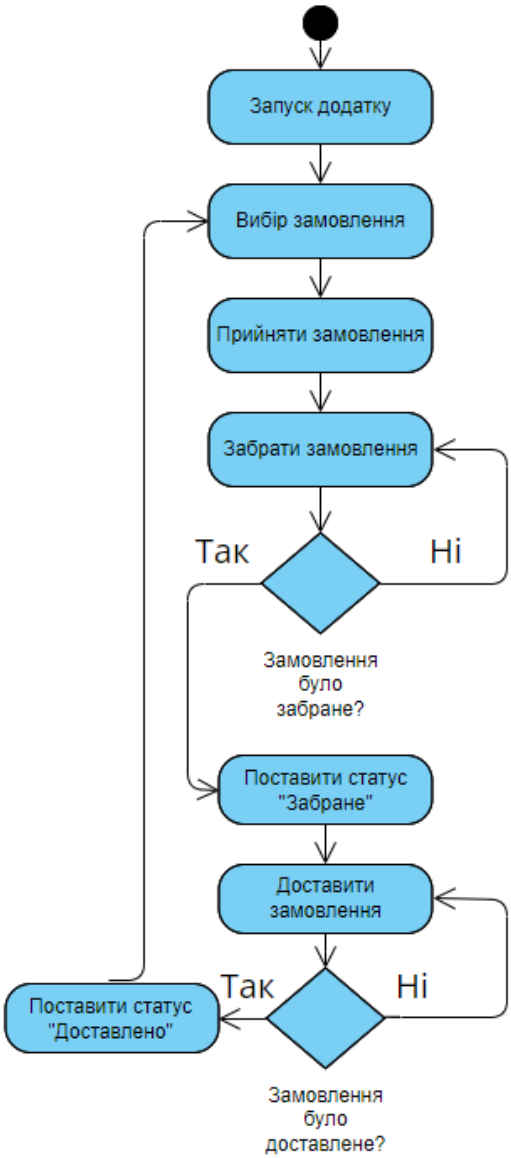


Рисунок 2.6 – Діаграма діяльностей

У частині repository необхідно буде реалізувати клас Repository, який буде використовуватися нашими bloc-класами для отримування необхідної інформації для відображення з бази даних та завантаження у базу та її оновлення. Також, для роботи цього класу необхідно буде розробити класи-моделі, у яких буде зберігатися інформація, а саме:

– OrderModel, клас, що буде зберігати усю детальну інформацію про одне замовлення;
– UserModel, буде зберігати інформацію про користувача системи, що буде необхідна адміністратору.

На діаграмі пакетів (див. рис. 2.7), ми можемо ознайомитися з тим, де

будуть знаходитися класи, що були описані вище.

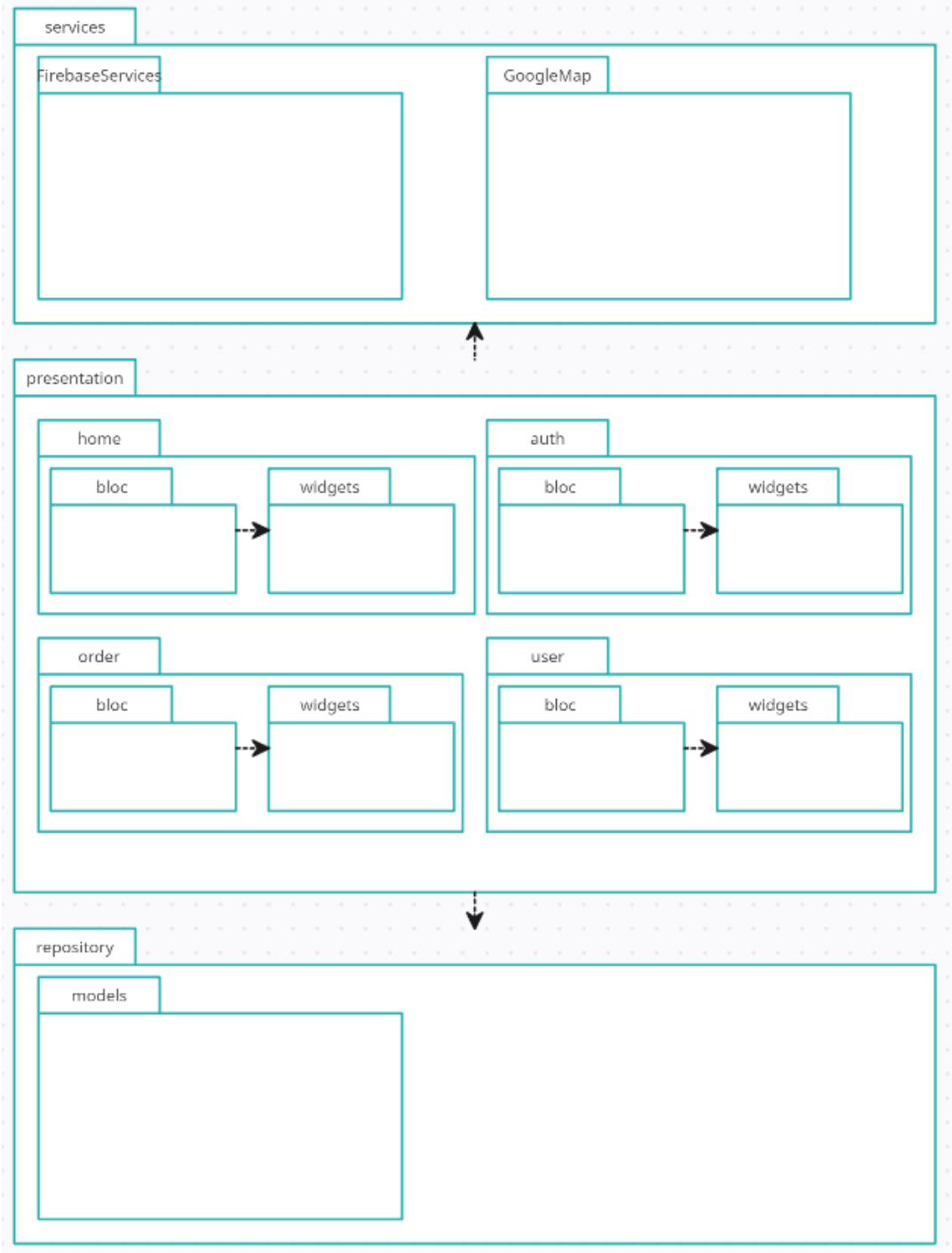


Рисунок 2.7 – Діаграма пакетів

Таким чином, було визначено варіанти використання системи та розписані головні прецеденти. Розроблена діаграма розгортання, спроєктовані необхідні класи для застосунку та його структура.

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Встановлення та налаштування робочого місця

На основі описаних у першому розділі даних, середовищем розробки було обрано AndroidStudio.

Перед початком розробки необхідно налаштувати середовище. Для цього було додано декілька плагінів (див. рис. 3.1), що зроблять розробку зручніше та швидше.

Плагін Dart надає підказки при кодуванні, включно з автоматичним завершенням коду, форматуванням та іншим. Завдяки ньому IDE знаходить помилки та надає засоби для їх автоматичного усунення.

Плагін Flutter Snippets додає у IDE шаблони для написання на Flutter, зменшуючи виникання помилок при написанні коду та економлячи час на його написання.

Плагін Bloc, інструменти для ефективного створення класів bloc для додатків на Flutter, та робить усі необхідні імпорти пакетів.

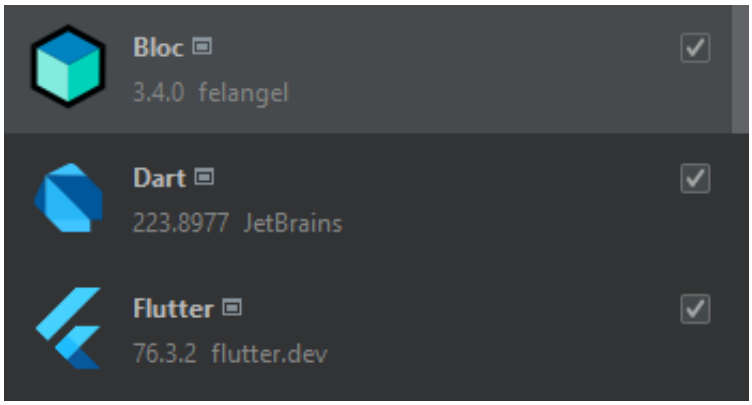


Рисунок 3.1 – Встановлені плагіни

Було підготовано теки, у яких будуть знаходитися класи, описані у діаграмі класів. Така структура дозволить спростити орієнтування у додатку та вносити зміни у коді.

Для використання необхідних сервісів Firebase, необхідно створити проєкт (див. рис. 3.2).

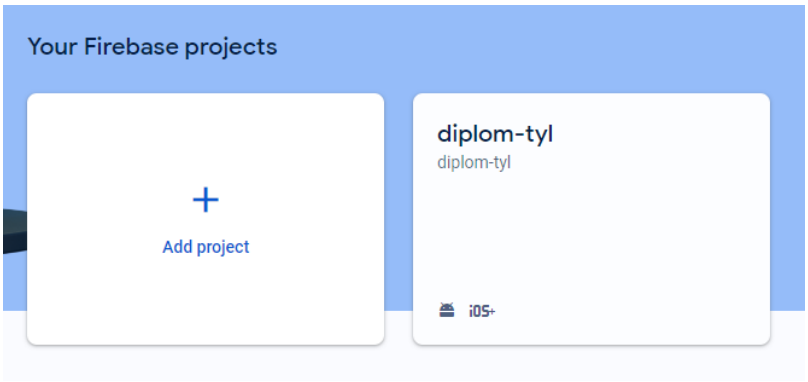


Рисунок 3.2 – Створений проєкт у Firebase

Для роботи з базою даних Realtime Database необхідно налаштувати правила, для доступу до неї (див. рис. 3.3). Це необхідно для того, щоб захистити дані від зловмисників, змін та їх втрати. На рисунку можна побачити правила, що були прописані у консолі. Завдяки цьому, доступ до бази дозволений тільки авторизованим користувачам.

```
{
  /* Visit https://firebase.google.com
  "rules": {
    "couriers": {
      "$uid": {
        ".read": "auth !== null",
        ".write": "$uid === auth.uid",
      }
    },
    "orders": {
      ".read": "auth !== null",
      ".write": "auth !== null",
    },
    ".read": false,
    ".write": false,
  }
}
```

Рисунок 3.3 – Правила бази даних

Авторизація за логіном та паролем у Firebase Authentication увімкнена за замовчування. У рамках нашого проєкту налаштувань не потребує.

3.2 Огляд розроблених класів

Відповідно до діаграми класів, було реалізовано клас FirebaseService, який реалізує механізми входу та реєстрації у застосунок. При його написанні було використано патерн singleton. Це надає можливість використовувати його у будь якій частині додатку. У якості параметрів він має auth, для того щоб звертатися до екземпляру класу Firebase Authentication та використовувати у своїх методах. Також він має параметр currentUser, що дозволяє використовувати дані користувача, такі як uid, email та інші.

Було реалізовано методи, описані у другому розділі, а саме: onLogin, onRegister, getCurrentUserId та logout. Методи onLogin та onRegister приймають пошту та пароль користувача, та роблять відповідні запити за допомоги Firebase Authentication. Фрагмент коду, відповідальний за вхід можна побачити на рисунку 3.4.

```
class FirebaseService {
  static final FirebaseService _singleton = FirebaseService._internal();
  factory FirebaseService() => _singleton;
  FirebaseService._internal();
  final auth = FirebaseAuth.instance;
  final currentUser = FirebaseAuth.instance.currentUser;
  late final String userId;
  onLogin(String email, String password) async {
    try {
      final credential = await auth.signInWithEmailAndPassword(
          email: email, password: password);
      return currentUser;
    } on FirebaseAuthException catch (e) {
      if (e.code == 'user-not-found') {
        print('No user found for that email.');
} else if (e.code == 'wrong-password') {
        print('Wrong password provided for that user.');}}}
```

Рисунок 3.4 – Фрагмент класу FirebaseService та методу onLogin

Згідно до спроєктованих раніше класів, було розроблено клас MapService, завдяки якому буде відбуватися відображення маркерів на мапі, місцезнаходження користувача, та рекомендованого маршруту.

Для відображення мапи використовується віджет GoogleMap з пакету google_maps_flutter. Для того, щоб відобразити маркер, або побудувати маршрут на мапі, використовуються координати, які зберігаються у екземплярах класу LatLng. Для позначення маркерів використовувався віджет Marker, з того ж пакету описаного вище.

Маркерам можна надавати ідентифікатори, іконки та інформаційне вікно, що буде відображатися при натисканні на нього та інше. Для того, щоб додати маркер на мапу, необхідно його до множини маркерів віджету GoogleMap.

Щодо побудови маршрутів, вони будуються при створенні замовлення та зберігатися у базі даних. Таким чином ми будемо будувати маршрут один раз, а далі використовувати його для відображення. Відображати маршрут на мапі будемо за допомоги ліній. Фрагмент коду з реалізованим віджетом можна побачити на рисунку 3.5.

Метод getLocationUpdates займається визначенням місця, де знаходиться користувач. Для визначення місцезнаходження користувача використовується пакет location. Екземпляр класу Location, має функцію, яка реагує на зміну положення, завдяки чому ми завжди будемо бачити актуальне місцезнаходження користувача.

Слід зауважити, що перед тим як використовувати місцезнаходження, слід впевнитися, що сервіс увімкнений та користувач надав необхідну згоду. Можемо побачити відповідні перевірки та запити у фрагменті коду на рисунку 3.6.

За отримування даних, необхідних для відображення маркерів та маршруту доставки був реалізований метод getData, що отримує дані з бази даних та оновлює необхідні значення.

Для отримування даних місцезнаходження користувача при перегляді деталей замовлення, використовується getUpdateLoc, що періодично отримує координати останнього місця розташування користувача з бази даних.

```
GoogleMap(
              initialCameraPosition: CameraPosition(
                target: _startP!,
                zoom: 15,),
              markers: {
                Marker(
                markerId: MarkerId("Courier"),
                icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueBlue),
                position: _courierP!,
                infoWindow: InfoWindow(
title: "Courier location",),),
                Marker(
                   markerId: MarkerId("_home"),
                   icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueGreen),
                   position: order.startAddressP!,
                   infoWindow: InfoWindow(
                     title: "Start",),),
                Marker(
                   markerId: MarkerId("Destination"),
                   icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueRed),
                   flat: true,
                   position: order.destAddressP!,
                   infoWindow: InfoWindow(
                     title: "Destination point",
                   )),
                polylines: Set<Polyline>.of(polylines.values),
},),
```

Рисунок 3.5 – Віджет мапи

```
_serviceEnabled = await _locationController.serviceEnabled();
   if(_serviceEnabled){
     _serviceEnabled = await _locationController.requestService();
   } else {
     return;
   }
   _permissionGranted = await _locationController.hasPermission();
   if(_permissionGranted == PermissionStatus.denied){
     _permissionGranted = await _locationController.requestPermission();
     if(_permissionGranted != PermissionStatus.granted){
       return;
     }
   }
```

Рисунок 3.6 – Перевірка дозволів для роботи з locationController

Для роботи з даними було розроблено, класи UserModel та OrderModel, описані у другому розділі.

Клас UserModel зберігає інформацію користувача, а саме:

– name, String, що зберігає і’мя користувача;
– email, String, що зберігає електрону пошту користувача;
– status, String, що зберігає статус користувача;
– phoneNumber, int, що зберігає номер телефону користувача.

Також було реалізовано конструктор dataSnapshot (див. рис. 3.7), для того, щоб виконати ініціалізацію приймаючи DataSnapshot.

```
factory UserModel.dataSnapshot(DataSnapshot snapshot) {
  return UserModel(
    name: snapshot.child('name').value.toString(),
    phoneNumber: int.parse(snapshot.child('phoneNumber').value.toString()),
    email: snapshot.child('email').value.toString(),
    status: snapshot.child('status').value.toString(),
    position: LatLng(double.parse(snapshot.child('position/Lat').value.toString()) ,
double.parse(snapshot.child('position/Lng').value.toString()))
  );}
```

Рисунок 3.7 – Конструктор класу UserModel

Клас OrderModel зберігає детальну інформацію про замовлення, а саме:

– name, String, що зберігає ім’я замовника;
– phoneNumber, int, що зберігає номер телефону замовника;
– status, String, що зберігає статус замовлення;
– homeAddress, String, що зберігає адресу знаходження замовлення;
– destAddress, String, що зберігає адресу доставки;
– homeAddressP, LatLng, що зберігає координати адреси знаходження замовлення;
– destAddressP, String, що зберігає координати адреси доставки;
– courier, String, що зберігає ідентифікатор користувача, який доставляє

замовленя.

Він також, як і UserModel, має іменований конструктор dataSnapshot.

Для роботи з базою даних було створено клас Repository, який використовується у класах bloc, для отримання та оновлення даних. Він не зберігає у собі дані, тільки надає необхідні методи для роботи з ними. Реалізовані методи класу Repository:

- onOrderListLoading, звертається до бази даних та отримує DataSnapshot, що містить інформацію про замовлення, що необхідна для відображення на головній сторінці, яку потім повертає у вигляді списку OrderModel;
- onOrderLoading, звертається до бази даних та отримує детальну інформацію про замовлення, яку необхідно відобразити на екрані інформації про замовлення, яку потім повертає вигляді у моделі OrderModel;
- onUserRegistered, він отримує дані реєстрації, у ньому створюється екземпляр класу UserModel, після чого відбувається звернення до бази даних та створюється документ, що буде містити інформацію про користувача;
- onUserLoading, що звертається до бази даних та отримує необхідну інформацію для відображення на сторінці користувача, яку повертає у UserModel;
- onOrderTaked, що вносить у замовлення ідентифікатор користувача, який доставляє замовлення, шляхом оновлення відповідного значення документу;
- onUserUpdate, який вносить зміни у профіль користувача.

За рекомендацією розробників flutter, що є у документації, клас репозиторій було винесено як окремий пакет, структуру якого можна побачити на рисунку 3.8.

У теці lib був створений файл з розширенням .dart, що зберігає у собі експорт необхідних класів. Сам клас Repository знаходиться у теці src, так само як і тека з розробленими моделями, необхідними для його роботи.

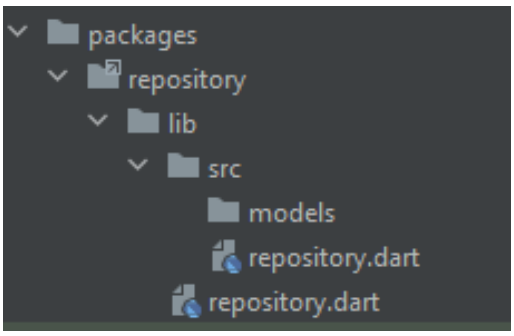


Рисунок 3.8 – Пакет repository

Також необхідно у теці repository створити файл pubspec.yalm та прописати у ньому наш файл для експорту.

```
export 'lib/src/repository.dart';
```

Рисунок 3.9

А у нашому основному файлі pubspec.yalm додаємо репозиторій.

```
repository:
  path: packages/repository
```

Рисунок 3.10

Для створення інтерфейсу у Flutter використовуються віджети. Вони є головними блоками при його створенні. У Flutter є дві категорії віджетів, це StatelessWidget та StatefullWidget. Головна відмінність в них в тім, що перші не змінюють свій стан протягом свого циклу життя, другі можуть його змінювати.

Для того, щоб використовувати спроєктовані у другому розділі класи bloc, необхідно використовувати віджет BlocProvider. Він надає можливість використовувати екземпляри розроблених класів bloc у дочірніх елементах, які у ньому огорнуті. Приклад коду з ним та класом AuthBloc можна побачити на рисунку 3.11.

```
class AuthScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
    child:
        BlocProvider(
        create: (context) => AuthBloc(),
        child: Scaffold(
          appBar: AppBar(
            title: Text('Authorisation'),
          ),
          body: Body()
        ),),
  );}
}
```

Рисунок 3.11 – Віджет AuthScreen

Точкою входу у додаток є файл main.dart. У ньому знаходиться void main, у якому ініціалізуються необхідні в нашому додатку сервіси Firebase. Також у ньому описаний клас MyApp.

MyApp містить віджет MaterialApp, у якому описується кольорова схема додатку, маршрути для перенаправлення користувача на інші екрани, поле назви та інше. В нашому випадку, поле home містить створений AuthScreen, який ми опишемо далі.

При розробці екранів застосунку, у теці відповідного екрану створювалися файли з однойменною назвою, де описується батьківський віджет, у якому буде відбуватися ініціалізація необхідних параметрів та використовуватися bloc провайдери, які прийматимуть відповідні до кожного екрану віджети Body. В них буде будуватися основний інтерфейс екрану та відбуватися взаємодія з розробленими класами bloc та окремими віджетами.

Для реалізації інтерфейсу користувача, було розроблено чотири екрани, описані у другому розділі. Для екрану авторизації було зроблено головний віджет AuthScreen, та для нього віджети SignIn та SignUp.

Вони будуть відображатися, у залежності від того, яка дія буде обрана користувачем, вхід або реєстрація. За замовчування відображається вікно входу у додаток, якщо користувачем буде обрано пункт «Реєстрація» – віджет з формою реєстрації.

Для реалізації роботи механізмів входу, реєстрації та виходу з додатку, були реалізовані описані у другому розділі класи. AuthBloc має параметр isLogin, що відповідальний за відображення форм реєстрації або входу, repository для роботи з базою даних та auth класу FirebaseService для входу та реєстрації.

Була розроблена необхідна логіка, для реагування на відповідні події, а саме методи: _onSignInEvent, _onSignUpEvent, _onSignOutEvent, та _onUpdateIsLoginEvent. Реагувати вони будуть на відповідні події.

При натисканні кнопки входу, спочатку перевіряються поля пошти та паролю, після чого через контекст до нашого класу bloc передається SignInEvent з даними для входу користувача. Далі відпрацьовує асинхронний метод _onSignInEvent, у якому використовується FirebaseService для виконання входу. Якщо він не отримує помилок, ми передаємо у emit стан AuthSignedInState. Після цього користувача направляє на головний екран.

При натисканні на кнопки реєстрації, до AuthBloc передається подія UpdateIsLoginEvent, яка змінює початкове значення змінної isLogin, що є флагом для зміни форми входу на реєстрацію.

При натисканні кнопки реєстрації виконується перевірка введених користувачем даних, якщо помилок немає, до нашого класу bloc передається SignUpEvent з даними для реєстрації користувача. У відповідь на подію, відпрацьовує асинхронний метод _onSignUpEvent, у якому спочатку виконується реєстрація у FirebaseService. Якщо помилок немає, виконується відповідний метод у класі репозиторію, що приймає отриманий унікальний ідентифікатор користувача, який генерується, та дані для збереження.

Після цього у emit передається стан AuthSignedInState та користувач перенаправляється на головний екран застосунку.

Для виходу з додатку, у верхньому меню застосунку є кнопка, при

натисканні якої до класу AuthBloc буде передаватися подія SignOutEvent, після якої виконається вихід з системи, та перенаправлення на сторінку входу.

Схожим чином побудований головний екран. У класі HomeScreen буде BlocProvider, що буде надавати екземпляр HomeBloc дочірньому віджету Body, у якому буде відбуватися відображення списку віджетів, які будуть містити необхідну інформацію про замовлення.

Список замовлень реалізований за допомоги ListView.builder, який має параметр itemCount, та itemBuilder, який будує список віджетів. Окрім інформації про замовлення, віджети мають подію при натисканні, при виникненні якої відбувається перенаправлення на сторінку детальної інформації про замовлення та передача ідентифікатора замовлення у якості параметру.

Методи, що були розроблені у класі HomeBloc: _onHomeOrdersLoading, та _onHomeOrdersUpdate. Перший відповідає за загрузку замовлень, а другий за оновлення. Для цього наш блок має екземпляр класу Repository, що повертає необхідні для цього дані.

Екран для відображення детальної інформації, OrderScreen, створений за аналогією з іншими, та надає дочірнім елементам екземпляр OrderBloc. Сам екран можна умовно поділити на дві частини.

У верхній частині екрану буде знаходитися віджет GoogleMap, на якому будуть відображені: маркери точки забору та доставки, побудований рекомендований маршрут.

У нижній частині буде відображатися вся інформація про замовлення у віджеті OrderCard. Також на екрані у нижній частині буде знаходитися кнопка для того, щоб взяти замовлення в роботу.

У класі OrderBloc було реалізовано методи: _onOrderLoading, _onOrderInWork, _onOrderTaked та _onOrderDelivered, що будуть виконуватися при відповідних подіях.

При події OrderLoadingEvent, отримуються дані з бази даних за допомоги класу Repository. Якщо дані успішно отримані, у emit передається відповідний стан, OrderLoadedState, що приймає OrderModel.

Якщо відбувається подія OrderInWorkEvent, статус замовлення змінюється на прийняте та у поле, що зберігає значення кур’єра, буде записано ідентифікатор користувача. Також відбудеться зміна статусу кур’єра, він прийме значення номеру замовлення.

Таким чином, якщо поле у заказі матиме ідентифікатор кур’єра, він не буде доступний для інших кур’єрів та якщо у кур’єра у статусі номер замовлення, він не зможе прийняти в роботу ще одне. Відповідні перевірки передбачені у коді.

При події OrderTakedEvent, відбувається зміна статусу на «у кур’єра». Слід зауважити, щоб змінити цей статус, слід бути поруч з точкою забору замовлення. Так само з подією OrderDeliveredEvent, зміна статусу можлива тільки тоді, коли кур’єр знаходиться поруч з точкою доставки.

Дозволеною відстанню між точками було задано п’ятдесят метрів, так як під час доставки можуть виникнути різні випадки, наприклад, точка забору може знаходитися у великому торговому центрі. Відстань від місцем призначення до координат користувача буде вираховуватися без урахування кривизни земного шару, так як така точність не потрібна на такій малій відстані.

Для екрану користувача також був реалізований клас UserBloc. Він має методи _onUserLoading, _onUserChanged, _onUserChange, та виконуються при відповідних подіях. При виникненні події UserLoadingEvent, за допомоги класу Repository отримується інформація про користувача.

Якщо помилки не виникло, у emit передається UserLoadedState, що приймає UserModel. При виникненні події UserChangeEvent, форма користувача змінюється, що надає можливість редагувати профіль.

Також форма містить кнопки, що дозволяють відмінити зміну або зберегти зміни. Якщо користувач обирає зміну, тоді відбувається подія UserChangedEvent, та виконується метод _onUserChange, який завдяки класу репозиторію оновлює дані користувача.

Таким чином, було реалізовано усі спроєктовані у другому розділі класи, та був розроблений інтерфейс користувача, що дозволяє виконувати вхід до системи та реєстрацію, переглядати список з замовленнями та детальну

інформацію про них, брати замовлення у роботу та змінювати їх статус, переглядати профіль та редагувати його.

3.3 Тестування додатку

Невід'ємною частиною розробки є тестування. Завдяки ньому спеціалісти знаходять у коді помилки, що забезпечує створення надійного на стабільного застосунку.

На ранніх етапах розробки, воно знижує ризики, що можуть бути пов'язані з некоректною роботою коду.

Тестуючи код, розробники можуть бути впевнені, що зміни які були зроблені не будуть сприяти виникненню нових помилок.

Для нашого додатку було проведено юніт-тестування, перевірка роботи окремих модулів програми, та інтеграційне тестування, для перевірки роботи взаємодії модулів додатку.

При модульному та інтеграційному тестуванні були використані наступні пакети: test, flutter_driver, flutter_test, mockito,

Загалом було написано усі необхідні тести, які передбачали різні варіанти, які можуть виникнути на тому чи іншому етапі роботи застосунку, зокрема, тестувалася взаємодія додатку з сервісами Firebase Authentication та Realtime Database.

Фрагмент тестування класу AuthBloc з використанням mock-класів можна побачити на рисунку 3.12. Також було проведено тестування інтерфейсу користувача. Це дозволяє перевірити коректність роботи інтерфейсу після введення змін.

Виділяють декілька підходів до тестування інтерфейсу у Flutter: тестування віджетів, тестування інтерфейсу користувача та тестування зображень.

```
test('emits [AuthSignedInState] SignInEvent successfully', () async {
    when(mockRepository.onLogin(any, any)).thenAnswer((_) async => User(/* mock user data */));
    authBloc.add(SignInEvent(email: 'test@example.com', password: 'password'));
    await expectLater(
      authBloc,
      emitsInOrder([AuthSignedInState(user: isA<User>())]),
    );    });
   test('emits [AuthErrorState] SignInEvent error', () async {
    when(mockRepository.onLogin(any, any)).thenThrow(Exception('Mock error'));
    authBloc.add(SignInEvent(email: 'test@example.com', password: 'password'));
    await expectLater(
      authBloc,
      emitsInOrder([AuthErrorState(errorMessage: 'error')]),      );
```

Рисунок 3.12 – Фрагмент тестування AuthBloc

Було проведено тестування інтерфейсу користувача, для емуляції його дій, різних сценаріїв при роботі з додатком. Для цього був використаний пакет frutter_driver, шо дозволяє автоматизувати взаємодію з додатком та емулювати такі дії як: натискання, свайпи, набір тексту та інші. На рисунку 3.13 зображений фрагмент тесту, який заповнює форму входу у застосунок та виконує вхід, після чого очікує відображення кнопки виходу на головному екрані, що буде означати успішний вхід.

```
test('Login UI Test', () async {
    driver.tap(find.byValueKey('Email'));
    driver.enterText('yar.belgorod@gmail.com');
    driver.tap(find.byValueKey('Password'));
    driver.enterText('qweqwe');
    driver.tap(find.text('Login'));
    SerializableFinder iconButtonFinder = find.byValueKey('LogOut');
    await driver.waitFor(iconButtonFinder, timeout: Duration(seconds: 4));
    bool isPresent = true;
    expect(isPresent, isTrue);
  });
```

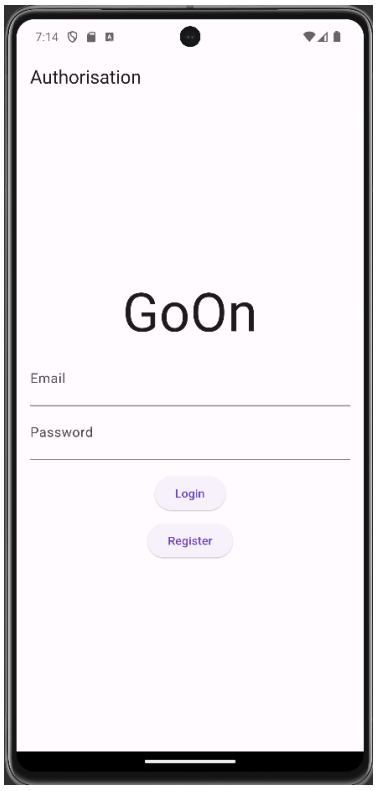
Рисунок 3.13 – Фрагмент тестування входу у застосунок

Таким чином було проведено тестування модулів додатку, їх взаємодію між собою та сервісами. Також було проведення автоматизоване тестування інтерфейсу користувача та різні сценарії роботи застосунку. Можемо перейти до огляду готового застосунку.

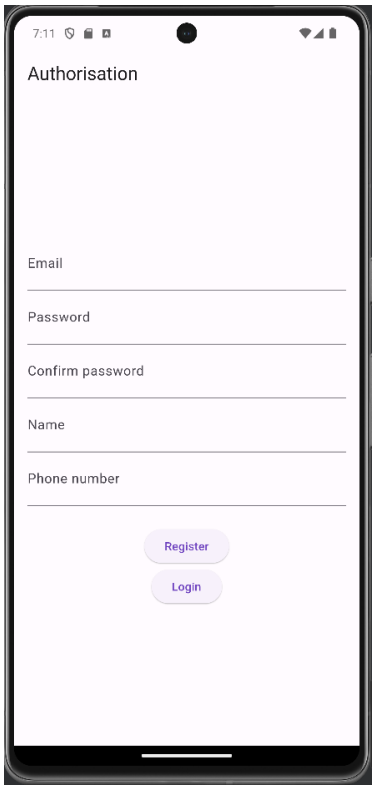
3.4 Огляд застосунку

При запуску додатку ми можемо побачити форму для входу у застосунок (див. рис 3.14), у ній зареєстрований користувач може заповнити свій логін та пароль у відповідні поля та виконати вхід до системи та попаде на головний екран.

При натисканні на кнопку реєстрації, він побачить форму реєстрації (див. рис 3.14). Вона має поля для пошти, паролю та його підтвердження, ім’я та номер телефону.



а) Форма входу

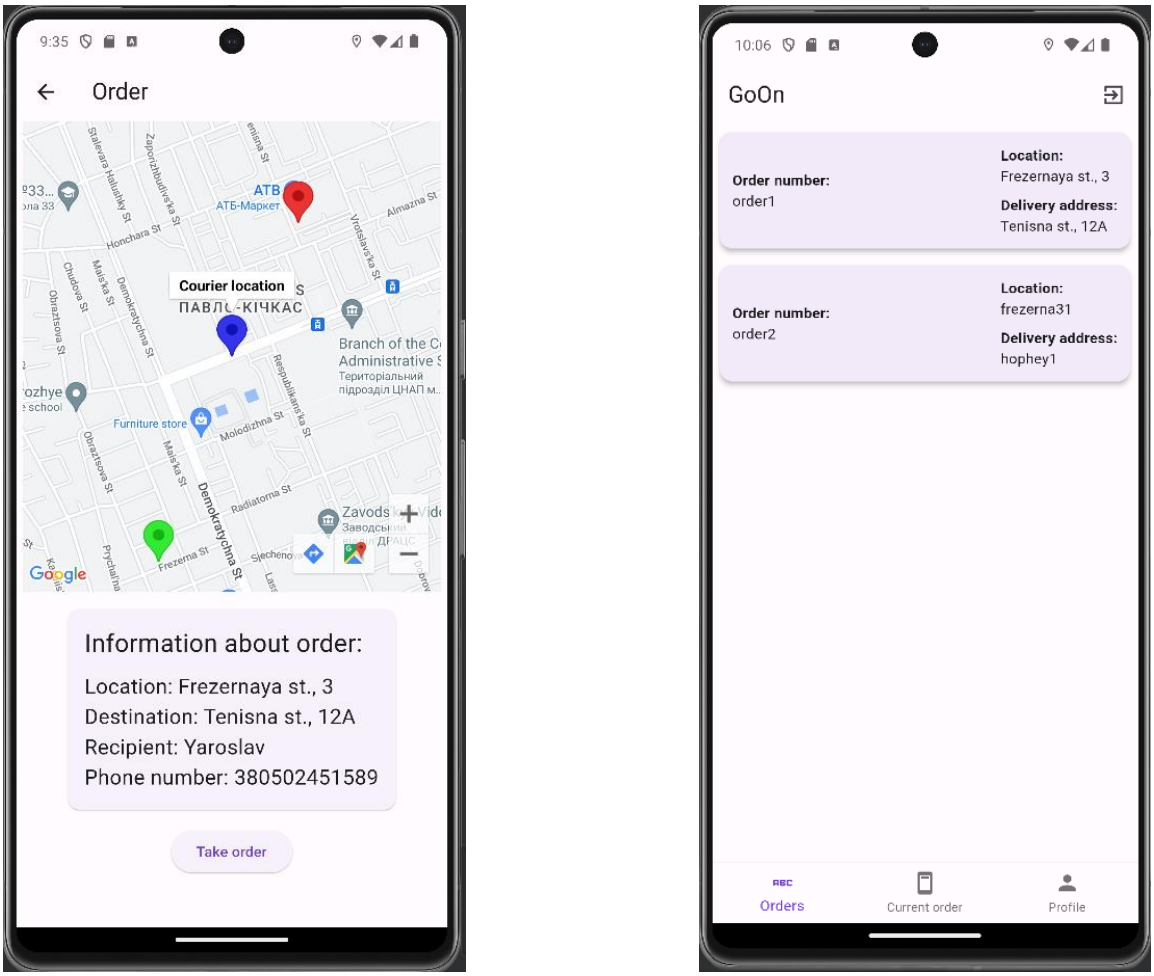


б) Форма реєстрації

Рисунок 3.14 – Екран авторизації

Після успішної реєстрації користувача буде направлено на головний екран. Внизу знаходиться навігаційна панель, яка містить 3 кнопки: головний екран, активне замовлення та профіль.

На головному екрані можемо побачити список вільних замовлень (див. рис 3.15). Після натискання на одне з замовлень, користувач буде направлений на сторінку з детальною інформацією про нього.



а) Екран замовлення б) Головний екран

Рисунок 3.15 – Відображення замовлень

Як ми можемо побачити, у верхній частині знаходиться мапа, яка вже має необхідні маркери точок знаходження замовлення та адресу доставки, місцезнаходження користувача та рекомендований маршрут доставки.

У нижній частині відображається картка з інформацією про замовлення та кнопка для прийняття замовлення у роботу. Після її натискання, користувач буде

назначений виконувачем замовлення.

При натисканні на кнопку активного замовлення, буде відображена детальна інформація про нього, або повідомлення, що активного замовлення немає.

При натисканні на кнопку профілю, користувач попаде на сторінку перегляду інформації про користувача (див. рис 3.16). Вона містить такі поля: пошта, ім’я, номер телефону, статус, список виконаних замовлень.

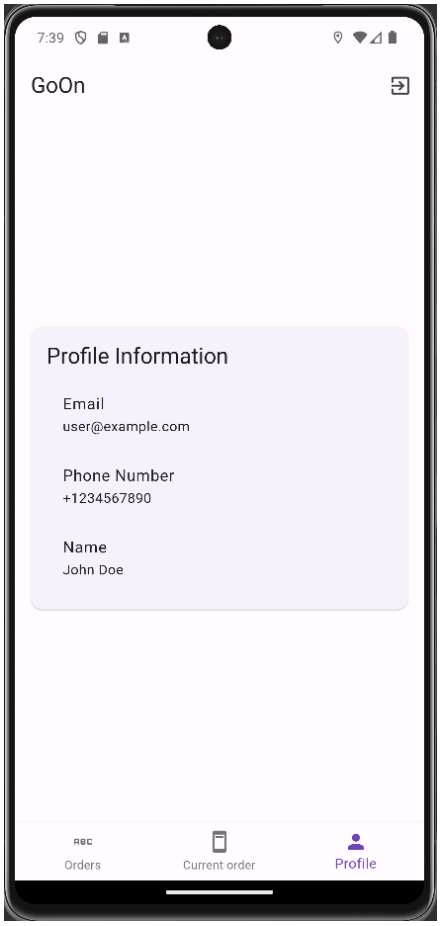


Рисунок 3.16 – Профіль користувача

Таким чином результатом роботи є розроблений додаток, який був розроблений за допомоги Flutter та Dart з використанням сервісів Firebase, за допомоги середовища для розробки Android Studio. Були розроблені усі спроєктовані класи та інтерфейс користувача. Усі розроблені класи та інтерфейс користувача були протестовані.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи проведено детальний аналіз технологій розробки Android застосунків та їх інтеграції з хмарними сервісами Firebase, спроєктовано, розроблено Android застосунок. Виконано тестування програмного засобу.

Особливістю реалізованої в кваліфікаційній роботі програмної системи є реалізація входу та реєстрації за допомоги сервісу Firebase Authentication, зберігання інформації у базі даних у реальному часі з використанням Firebase Realtime Database, відображення інформації про переміщення та побудова маршрутів з застосуванням Google Maps Platform.

Застосунок було реалізовано з використанням фреймворку Flutter. Для відстеження місцезнаходження використовувався пакет location. Також реалізовано методи для роботи з сервісом автентифікації, а саме, входу та реєстрації, механізм для додавання, оновлення та зчитування інформації для роботи з базою даних у реальному часі, методи для реагування на зміну місцезнаходження користувача та побудови маршруту між точками з використанням сервісу для роботи з географічними даними.

Перспективи подальших досліджень пов’язано з розширенням функціональності.

ПЕРЕЛІК ПОСИЛАНЬ

1. Sufyan M. Mastering Android Studio: A Beginner’s Guide (Mastering Computer Science). 2022. 248 p.
2. Allen G. Android for Absolute - Beginners Getting Started with Mobile Apps Development Using the Android Java SDK. 2021. 376 p.
3. Hagos T. Beginning Kotlin: Build Applications with Better Code, Productivity, and Performance. 2022. 241 p.
4. Sufyan M. Mastering Flutter: A Beginner’s Guide. 2022. 372 p.
5. Katz M., Moore K.D., Ngo V. Flutter Apprentice: Learn to Build Cross-Platform Apps. 2021. 615 p.
6. Firebase Realtime Database. Documentation. URL: https://firebase.google.com/docs/database (дата звернення: 27.06.2023).
7. Cloud Firestore. Documentation. URL: https://firebase.google.com/docs/firestore (дата звернення: 10.07.2023).
8. Firebase Machine Learning. Documentation. URL: https://firebase.google.com/docs/ml (дата звернення: 13.07.2023).
9. Cloud Functions for Firebase. Documentation. URL: https://firebase.google.com/docs/functions (дата звернення: 15.07.2023).
10. Firebase Authentication. Documentation. URL: https://firebase.google.com/docs/auth (дата звернення: 18.07.2023).
11. Firebase Cloud Messaging. Documentation. URL: https://firebase.google.com/docs/cloud-messaging (дата звернення: 22.07.2023).
12. Cloud Storage for Firebase. Documentation. URL: https://firebase.google.com/docs/storage (дата звернення: 24.07.2023).
13. Firebase Crashlytics. Documentation. URL: https://firebase.google.com/docs/crashlytics (дата звернення: 30.07.2023).
14. Google Analytics. Documentation. URL:

https://firebase.google.com/docs/analytics (дата звернення: 10.08.2023).

15. Firebase Performance Monitoring. Documentation. URL: https://firebase.google.com/docs/perf-mon (дата звернення: 13.08.2023).
16. Firebase Test Lab. Documentation. URL: https://firebase.google.com/docs/test-lab (дата звернення: 16.08.2023).
17. Firebase App Distribution. Documentation. URL: https://firebase.google.com/docs/app-distribution (дата звернення: 19.08.2023).
18. Firebase Remote Config. Documentation. URL: https://firebase.google.com/docs/remote-config (дата звернення: 23.08.2023).
19. Firebase A/B Testing. Documentation. URL: https://firebase.google.com/docs/ab-testing (дата звернення: 26.08.2023).
20. Firebase In-App Messaging. Documentation. URL: https://firebase.google.com/docs/in-app-messaging (дата звернення: 29.08.2023).