МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПОРІВНЯННЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ ТА МОБІЛЬНОГО ДОДАТКУ ЗАСОБАМИ ФРЕЙМВОРКУ FLUTTER»**

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Н.А. Бойченко
(ініціали та прізвище)

Керівник викладач кафедри програмної інженерії, PhD Климчук А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Матвіїшина Н.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бойченку Нікіті Андрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Порівняння розробки вебзастосунку та мобільного додатку засобами фреймворку Flutter

керівник роботи Климчук Анастасія Олександрівна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Теоретичні основи кросплатформеної розробки.
2. Аналіз та проєктування програмного застосунку.
3. Програмна реалізація вебзастосунку та мобільного додатку.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 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.	19.12.2025	
2.	Збір вихідних даних.	30.01.2026	
3.	Обробка методичних та теоретичних	20.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2026	

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

Керівник роботи _____ А.О. Климчук
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Порівняння розробки вебзастосунку та мобільного додатку засобами фреймворку Flutter»: 47 с., 16 рис., 10 табл., 11 джерел.

ВЕБЗАСТОСУНОК, КРОСПЛАТФОРМЕНА РОЗРОБКА, МОБІЛЬНИЙ ДОДАТОК, РЕПОЗИТОРІЙ, ТРЕКІНГ ПОСИЛОК, ANDROID, CLOUD FIRESTORE, DART, FIREBASE AUTHENTICATION, FLUTTER, GO BOX, IOS, SMS-АВТЕНТИФІКАЦІЯ.

Об’єкт дослідження – процес розробки кросплатформених програмних застосунків.

Мета роботи: зменшення часу на пошук та відстеження посилок користувачами шляхом програмної реалізації застосунку «GO BOX» (з використанням Firebase Authentication та Cloud Firestore), на основі якого проводиться порівняння розробки вебзастосунку та мобільного додатку засобами фреймворку Flutter.

Методи дослідження – аналіз предметної області, порівняльний аналіз, об’єктно-орієнтоване проєктування, моделювання структури даних, програмна реалізація, функціональне тестування та аналіз результатів розгортання на різних платформах.

У роботі розглянуто основи кросплатформеної розробки, особливості Flutter для створення мобільних і вебзастосунків, а також структуру проєкту nova_post: моделі, сервіси, репозиторії та основні сторінки. Практична частина присвячена застосунку «GO BOX», який забезпечує авторизацію за номером телефону, перегляд і пошук посилок, відображення історії трекінгу та розмежування доступу між користувачем і гостем.

Результатом роботи є опис готового кросплатформеного застосунку, порівняння веб і мобільної реалізації на Flutter.

SUMMARY

Bachelor's qualifying paper "Comparison of the Web Application and Mobile Application Development Using Flutter Framework": 47 pages, 16 figures, 10 tables, 11 references.

WEB APPLICATION, CROSS-PLATFORM DEVELOPMENT, MOBILE APPLICATION, REPOSITORY, PARCEL TRACKING, ANDROID, CLOUD FIRESTORE, DART, FIREBASE AUTHENTICATION, FLUTTER, GO BOX, IOS, SMS AUTHENTICATION.

The object of the study is the process of developing cross-platform software applications.

The aim of the study is to reduce the time spent by users on searching and tracking parcels through the software implementation of the «GO BOX» application (using Firebase Authentication and Cloud Firestore), on the basis of which a comparison of web and mobile application development using the Flutter framework is conducted.

The methods of research include domain analysis, comparative analysis, object-oriented design, data structure modeling, software implementation, functional testing, and analysis of deployment results for different platforms.

The paper examines the basics of cross-platform development, the features of Flutter for creating mobile and web applications, and the structure of the nova_post project, including its models, services, repositories, and main pages. The practical part is devoted to the «GO BOX» application, which provides phone number authentication, parcel viewing and search, tracking history display, and access differentiation between an authorized user and a guest.

The result of the work is a description of the completed cross-platform application, a comparison of web and mobile implementation using Flutter.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Теоретичні основи кросплатформеної розробки 9
1.1 Поняття вебзастосунку та мобільного додатку 9
1.2 Фреймворк Flutter як інструмент кросплатформеної розробки 10
1.3 Мова Dart у структурі Flutter-проєкту 12
1.4 Firebase у клієнтському застосунку Flutter..... 14
1.5 Критерії порівняння веб та мобільної реалізації 14
2 Аналіз та проєктування програмного застосунку..... 17
2.1 Постановка задачі та вимоги до застосунку..... 17
2.2 Структура Flutter-проєкту 18
2.3 Проєктування моделі даних і бази Firestore 19
2.4 Проєктування інтерфейсу застосунку..... 21
2.5 Проєктування бази даних..... 22
2.6 Моделювання процесів та архітектурних рішень..... 26
2.7 Проєктування системи обробки помилок та виключних ситуацій.... 31
3 Програмна реалізація вебзастосунку та мобільного додатку..... 34
3.1 Реалізація інтерфейсу користувача та конфігурація навігації 34
3.2 Порівняння веб та мобільної реалізації Flutter 40
3.3 Порівняння поведінки Flutter-проєкту на різних платформах 41
3.4 Тестування та безпека застосунку..... 42
Висновки 45
Перелік посилань..... 47

ВСТУП

Сучасна розробка програмного забезпечення дедалі частіше орієнтується на одночасну підтримку кількох платформ. Користувачі очікують, що один і той самий сервіс буде доступний як у мобільному додатку, так і у вебзастосунку. Для розробника це створює потребу у виборі технології, яка дозволяє не дублювати логіку, зберігати єдиний стиль інтерфейсу та водночас враховувати особливості кожної платформи. Одним із таких інструментів є Flutter – фреймворк, який дає змогу створювати інтерфейси для різних платформ із єдиної кодової бази.

Актуальність теми полягає в тому, що традиційний підхід до створення окремого вебзастосунку та окремого мобільного додатку потребує більше часу, окремих команд розробників, дублювання бізнес-логіки та складнішого супроводу. Flutter пропонує інший підхід: основна логіка та значна частина інтерфейсу реалізуються один раз мовою Dart, а потім збираються для потрібних платформ. Разом з тим веб і мобільне середовище мають різні вимоги до навігації, розмірів екрана, авторизації, продуктивності та способу розгортання, тому порівняння цих підходів має практичне значення.

У межах кваліфікаційної роботи розглядається готовий Flutter-проєкт nova_post, який має брендове ім’я «GO BOX». Проєкт реалізує застосунок для відстеження посилок. Його функціональність включає авторизацію за номером телефону через Firebase Authentication, збереження даних у Cloud Firestore, перегляд списку посилок, пошук посилки за номером, відображення історії доставки та обмеження доступу до повної інформації залежно від ролі користувача.

Застосунок «GO BOX» є зручним прикладом для дослідження теми, оскільки він має типову для реальних сервісів структуру: окремі моделі даних, сервіси для роботи з зовнішнім backend, репозиторії для бізнес-логіки, сторінки інтерфейсу та віджети для повторного використання. Крім того, у

проєкті присутні конфігурації для мобільних і вебплатформ, що дозволяє оцінити відмінності між розробкою мобільного додатку та вебзастосунку в одному Flutter-проєкті.

Мета кваліфікаційної роботи – порівняти розробку вебзастосунку та мобільного додатку засобами фреймворку Flutter і на основі готового проєкту «GO BOX» описати архітектуру, структуру даних, реалізацію основних функцій та особливості використання Firebase.

Для досягнення мети необхідно виконати такі завдання:

– проаналізувати теоретичні засади кросплатформеної розробки;
– розглянути роль Flutter і Dart у створенні веб та мобільних застосунків;
– описати структуру проєкту «GO BOX», визначити моделі даних і структуру Firestore;
– описати реалізацію авторизації, списку посилок і трекінгу;
– провести порівняння веб і мобільної версій;
– сформулювати висновки щодо переваг і обмежень обраного підходу.

Об’єктом дослідження є процес розробки кросплатформених програмних застосунків. Предметом дослідження є особливості створення вебзастосунку та мобільного додатку засобами Flutter на прикладі застосунку «GO BOX».

Практичне значення роботи полягає в тому, що її матеріали можуть бути використані як технічний опис готового Flutter-проєкту, як основа для подальшого розвитку застосунку, а також як приклад організації кросплатформеної розробки з використанням Firebase Authentication і Cloud Firestore.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань. У першому розділі подано теоретичні основи розробки веб і мобільних застосунків засобами Flutter. У другому розділі описано постановку задачі, структуру проєкту, моделі даних та архітектуру застосунку. У третьому розділі наведено реалізацію основних функцій, тестування та порівняльний аналіз веб і мобільної версій.

1 ТЕОРЕТИЧНІ ОСНОВИ КРОСПЛАТФОРМЕНОЇ РОЗРОБКИ

1.1 Поняття вебзастосунку та мобільного додатку

Вебзастосунок – це програмний продукт, який запускається у браузері та взаємодіє з користувачем через вебінтерфейс. Його основною перевагою є доступність без обов'язкового встановлення на пристрій. Користувачу достатньо мати браузер і підключення до мережі. У випадку Flutter [1] вебзастосунок компілюється у набір файлів, які можуть бути розміщені на вебсервері та відкриті через браузер.

Мобільний додаток – це застосунок, який встановлюється на смартфон або планшет і працює в середовищі конкретної мобільної операційної системи. Для Flutter типовими цільовими платформами є Android та iOS. Мобільний додаток має доступ до функцій пристрою, може використовувати нативні механізми авторизації, локальні сховища, push-повідомлення та інші можливості платформи.

Порівняння веб і мобільної розробки важливе через різницю у способах використання. На мобільному пристрої користувач працює з сенсорним екраном, очікує швидкої реакції, зручної навігації одним пальцем та адаптації під вертикальну орієнтацію. У вебверсії користувач може працювати з клавіатурою, мишею, різними розмірами вікна браузера, горизонтальними макетами та іншою моделлю навігації.

У традиційній розробці вебзастосунок і мобільний додаток часто створюються різними технологіями. Наприклад, вебчастина може будуватися на JavaScript-фреймворку [11], а мобільна – окремо для Android та iOS [10]. Це збільшує кількість коду, ускладнює підтримку і потребує синхронізації логіки.

Веб та мобільна реалізація мають різні сильні сторони. Вебзастосунок простіше розповсюджувати, адже користувачеві не потрібно нічого встановлювати. Мобільний додаток краще підходить для постійного використання, тому що він ближчий до системних можливостей пристрою. У

Flutter ці відмінності не зникають повністю, але розробник отримує можливість реалізовувати їх в одному середовищі.

У межах проєкту основна функціональність однакова для обох варіантів: авторизація, перегляд посилок, пошук за номером і перегляд історії. Проте реалізація інтерфейсу та розгортання на платформі мають свої відмінності, що і стало предметом порівняння у цій роботі.

Ще одна важлива відмінність полягає у тестуванні. Вебзастосунок потрібно перевіряти у різних браузерах і при різній ширині вікна. Мобільний додаток потрібно перевіряти на різних розмірах екрана, з різними версіями Android або iOS, а також з урахуванням поведінки системної клавіатури. Відмінності базових характеристик обох платформ наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння вебзастосунку та мобільного додатку

Критерій	**Вебзастосунок**	**Мобільний додаток**
Спосіб запуску	Через браузер без обов’язкового встановлення	Після встановлення на пристрій користувача
Основне середовище	Браузер і вебсервер	Операційна система Android або iOS
Інтерфейс	Адаптація до різних розмірів вікна	Адаптація до сенсорного екрана смартфона
Розповсюдження	Посилання на сайт або PWA	APK/App Bundle або публікація в магазині застосунків
Доступ до можливостей пристрою	Обмежений можливостями браузера	Ширший доступ через platform-specific APIs
Оновлення	Оновлення на сервері	Оновлення збірки застосунку

1.2 Фреймворк Flutter як інструмент кросплатформеної розробки

Flutter – це фреймворк для створення інтерфейсів користувача, який дозволяє розробляти застосунки для різних платформ на основі єдиної кодової бази. У роботі Flutter розглядається не як абстрактна технологія, а як

практичний інструмент, на якому реалізовано застосунок «GO BOX». Код проєкту написано мовою Dart [2], а інтерфейс побудовано на системі віджетів.

Ключовою ідеєю Flutter є побудова інтерфейсу з віджетів. Віджет описує частину інтерфейсу або поведінки: текст, кнопку, контейнер, поле введення, список, сторінку тощо. З таких віджетів формується дерево інтерфейсу. У проєкті «GO BOX» сторінки PhoneLoginPage, ParcelListPage і ParcelTrackPage також реалізовані як Flutter-віджети, що відповідають за різні сценарії взаємодії з користувачем.

Для кросплатформеної розробки важливо, що Flutter не вимагає повного переписування логіки окремо під кожну платформу. Моделі даних, репозиторії та сервіси можуть залишатися спільними, а відмінності проявляються на рівні збірки, конфігурації платформи та окремих UI-адаптацій. Це добре видно у проєкті nova_post: структура lib містить спільний код, а платформи android, ios, web та інші мають власні конфігураційні файли.

У Flutter важливу роль відіграють StatelessWidget та StatefulWidget. Якщо віджет не має власного змінного стану, його можна реалізувати як StatelessWidget. Якщо ж екран повинен реагувати на введення користувача, завантаження даних або зміну режиму відображення, використовується StatefulWidget.

Однією з переваг Flutter є можливість швидко будувати повторно використовувані компоненти. У проєкті «GO BOX» окремими віджетами винесено ParcelTrackPage, ParcelListPage та PhoneLoginPage. Це робить код зрозумілішим і не змішує інтерфейс із усією логікою сторінки.

Flutter також підтримує Material Design [5], завдяки чому можна швидко створити інтерфейс із кнопками, полями введення, списками, AppBar і Drawer. У «GO BOX» використовується темний фон та помаранчевий акцент #FF8C0F. Така палітра повторюється у різних елементах і допомагає зробити інтерфейс цілісним.

Окремо слід згадати роботу з пакетами. У pubspec.yaml підключаються залежності, які розширюють можливості застосунку. У проєкті

використовуються firebase_core, firebase_auth [9], cloud_firestore, intl_phone_field [7], country_code_picker, intl [6], geocoding, cupertino_icons та flutter_lints, призначення яких описано в таблиці 1.2.

Таблиця 1.2 – Основні залежності проєкту

Залежність	**Призначення у проєкті**
firebase_core	Ініціалізація Firebase у Flutter-застосунку
firebase_auth	Авторизація користувача через номер телефону та SMS-код
cloud_firestore	Збереження користувачів, посилок і подій відстеження
intl_phone_field	Поле введення номера телефону з кодом країни
country_code_picker	Підтримка вибору коду країни
Intl	Форматування дат і часу
flutter_lints	Перевірка стилю коду та рекомендації аналізатора

1.3 Мова Dart у структурі Flutter-проєкту

Dart є мовою програмування, яка використовується у Flutter-проєктах. У застосунку «GO BOX» Dart застосовується для опису класів моделей, сервісів, репозиторіїв і сторінок інтерфейсу. Класи User, Parcel, Location, Trace та ParcelTrackResult демонструють об’єктно-орієнтований підхід до представлення даних.

Для теми роботи важливо, що Dart дозволяє описати спільну бізнес-логіку незалежно від того, де буде запущено застосунок – у браузері чи на мобільному пристрої. Наприклад, логіка пошуку посилки за номером або перевірки прав доступу користувача не повинна дублюватися окремо для web та Android. Вона винесена у репозиторій ParcelRepository, який звертається до сервісу FirebaseDbService.

У коді проєкту також використано асинхронні операції. Це необхідно, оскільки запити до Firebase Authentication і Cloud Firestore [4] виконуються

через мережу. Методи, що працюють із базою даних або авторизацією, повертають Future, а сторінки інтерфейсу оновлюють стан після отримання результатів. Такий підхід відповідає природі клієнтських застосунків, які активно взаємодіють із віддаленими сервісами.

Окреме значення має null-safety та робота з nullable-типами. У моделі ParcelTrackResult поле parcel може бути null, якщо користувач не має прав переглядати повну інформацію про посилку. Це дозволяє не приховувати всю історію трекінгу, але не показувати конфіденційні дані сторонньому користувачу, що відображено на рисунку 1.1.

```
class ParcelTrackResult {
  final Parcel? parcel;
  final List<Trace> traces;
  final bool canViewFullInfo;

  ParcelTrackResult({
required this.parcel,
    required this.traces,
    required this.canViewFullInfo,
  });
}
```

Рисунок 1.1 – Модель ParcelTrackResult

Наведений фрагмент демонструє, що результат пошуку посилки містить не лише дані, але й службовий прапорець доступу. Саме він використовується під час відображення інформації на сторінці трекінгу.

«GO BOX» не претендує на повну промислову систему доставки, але демонструє базові принципи: авторизацію, збереження даних, пошук, відображення історії та обмеження доступу. Саме ці принципи дають підставу виконувати порівняльний аналіз засобами Flutter.

Окремо слід враховувати, що кросплатформеність не означає повної відсутності перевірок. Навіть якщо код спільний, його потрібно запускати у

різних середовищах, перевіряти введення, переходи, роботу з Firebase і реакцію на помилки. Це особливо стосується сторінок авторизації та відстеження, де користувач взаємодіє з асинхронними операціями.

1.4 Firebase у клієнтському застосунку Flutter

Firebase [3] використовується у проєкті «GO BOX» як backend-платформа. У межах розглянутого застосунку застосовано два основні сервіси: Firebase Authentication для авторизації користувачів і Cloud Firestore [8] для збереження структурованих даних. Це дозволило реалізувати backend-функціональність без створення окремого серверного REST API.

Firebase Authentication у проєкті відповідає за вхід за номером телефону. Користувач вводить номер у міжнародному форматі, отримує SMS-код і підтверджує його у застосунку. Після успішної перевірки Firebase створює або повертає авторизованого користувача, а репозиторій UserRepository перевіряє наявність відповідного запису в колекції users.

Cloud Firestore використовується для збереження трьох основних груп даних: користувачів, посилок і подій відстеження. Така структура відповідає логіці предметної області. Користувач має ідентифікатор, номер телефону та дату створення. Посилка має номер, опис, вагу, початкову та кінцеву локацію, відправника, отримувача та ознаку оплати. Trace-фіксація містить статус, опис, водія, локацію та часову мітку.

1.5 Критерії порівняння веб та мобільної реалізації

Для коректного порівняння вебзастосунку та мобільного додатку потрібно визначити критерії. У цій роботі порівняння виконується за критеріями структури проєкту, інтерфейсу, авторизації, роботи з базою даних,

збірки, розгортання, доступу до можливостей платформи та подальшого супроводу.

У Flutter-проєкті більшість коду зосереджена в каталозі lib. Це означає, що сторінки, моделі, сервіси й репозиторії можуть використовуватися і для вебверсії, і для мобільного додатку. Для застосунків це важливо, бо основна логіка авторизації, пошуку посилки та відображення історії не потребує окремої реалізації.

Другим критерієм є інтерфейс. На мобільному пристрої користувач працює з сенсорним екраном, тому кнопки та поля введення мають бути достатньо великими. У вебверсії інтерфейс може відкриватися на широкому екрані, де надто вузьке розміщення елементів виглядатиме нераціонально.

Третім критерієм є авторизація. У «GO BOX» використовується Firebase Phone Auth, тобто вхід за номером телефону через SMS-код. На мобільному пристрої такий сценарій є звичним. У вебверсії він також можливий, але користувач може проходити його з комп’ютера, а SMS отримувати на телефон.

Четвертим критерієм є база даних. Firestore однаково доступний для мобільної і вебверсії, тому дані users, parcels та traces залишаються спільними. Це є сильним боком такого підходу, оскільки не потрібно створювати окреме сховище для кожної платформи.

П’ятим критерієм є навігація. У мобільному додатку користувач звик до переходів між екранами, кнопки назад і Drawer-меню. У вебзастосунку важливішим може бути відображення URL і можливість відкривати сторінки безпосередньо. У поточному проєкті використано named routes: /login, /parcelTrack та /parcelList.

Шостим критерієм є розгортання. Для Android застосунок збирається у APK або App Bundle. Для вебу результатом є набір файлів HTML, JavaScript та ресурсів, які можна розмістити на вебсервері. Код Flutter при цьому залишається спільним, але процес підготовки релізу різний.

Порівняння також має враховувати те, що деякі можливості Firebase можуть по-різному поводитися на різних платформах. Наприклад, phone

authentication для мобільних платформ і web має різні сценарії підтвердження, хоча в коді Flutter це може бути приховано за спільним API. Тому кросплатформеність не означає повної відсутності платформних особливостей.

Для застосунку «GO BOX» одним з найважливіших є те, що основна бізнес-логіка однакова для обох платформ. Пошук посилки, отримання списку traces, перевірка доступу та побудова результату відбуваються на спільному рівні репозиторію. Це є головною перевагою Flutter-підходу у порівнянні з повністю окремою розробкою веб та мобільних застосунків.

2 АНАЛІЗ ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

Практична частина роботи базується на проєкті nova_post, який у користувацькому інтерфейсі має назву «GO BOX». Це Flutter-застосунок, призначений для роботи з посилками. Його структура демонструє типовий підхід до навчального кросплатформеного проєкту: окремі сторінки відповідають за сценарії користувача, моделі описують дані, сервіси працюють з Firebase, репозиторії містять логіку доступу, а віджети повторно використовуються в інтерфейсі.

2.1 Постановка задачі та вимоги до застосунку

Застосунок «GO BOX» розроблено для спрощення роботи з посилками. Основний користувацький сценарій полягає в тому, що людина авторизується за номером телефону, після чого може переглянути список пов’язаних із нею посилок. Також користувач може відкрити сторінку відстеження та знайти посилку за її номером.

«GO BOX» передбачає два режими використання. Перший режим – авторизований користувач. Він проходить SMS-верифікацію і може бачити повну інформацію про посилки. Другий режим – гостьовий. Гість може відстежити посилку за номером, але не повинен отримувати персональні дані учасників доставки.

Перелік ключових функціональних вимог включає авторизацію користувачів через SMS-код, створення запису користувача у Firestore, а також відображення і пошук посилок за номером. Програма також передбачає завантаження історії подій, розмежування доступу, вихід із застосунку та підтримку гостьового режиму для перегляду статусів, а детальний опис цих процесів наведено в таблиці 2.1.

Таблиця 2.1 – Основні функціональні вимоги до «GO BOX»

№	Вимога	Опис реалізації
1	Авторизація за телефоном	Користувач вводить номер телефону та отримує SMS-код через Firebase Auth
2	Перевірка SMS-коду	Код вводиться у компоненті SmsCodeInput і перевіряється через Firebase
3	Створення користувача	Після успішного входу запис користувача створюється або знаходиться у Firestore
4	Перегляд списку посилок	ParcelListPage отримує посилки, пов’язані з поточним користувачем
5	Пошук посилки	ParcelTrackPage шукає посилку за номером
6	Історія доставки	З колекції traces завантажуються події відстеження
7	Гостьовий режим	Гість може переглядати тільки історію без персональних даних
8	Вихід із застосунку	FirebaseAuthService виконує signOut/logout

2.2 Структура Flutter-проєкту

Структура проєкту nova_post відповідає типовій структурі Flutter-застосунку. Головна частина коду розміщена у каталозі lib, де знаходяться файл main.dart, а також окремі підкаталоги для сторінок, моделей, сервісів, репозиторіїв, віджетів та тестових даних. Такий розподіл допомагає не змішувати в одному файлі інтерфейс, бізнес-логіку і доступ до бази даних.

Файл main.dart є точкою входу в застосунок. У ньому виконується запуск Flutter-програми, налаштовуються маршрути та підключаються основні сторінки.

Папка pages містить три основні сторінки. PhoneLoginPage відповідає за авторизацію користувача через номер телефону. ParcelListPage показує список посилок поточного користувача. ParcelTrackPage відповідає за пошук конкретної посилки і відображення її історії.

Папка models містить класи, які описують дані. Це User, Parcel, Location, Trace та ParcelTrackResult. Наявність моделей робить код більш зрозумілим, тому що дані не передаються як довільні Map-структури по всьому проєкту.

Папка services відповідає за роботу з Firebase. Firebase.auth.service.dart містить методи авторизації, перевірки SMS-коду та виходу з системи. Firebase.database.service.dart містить методи для створення і отримання користувачів, посилок та подій відстеження.

Папка repositories містить Parcel.repo.dart та User.repo.dart. Репозиторії не просто виконують запити, а об’єднують декілька дій у логіку вищого рівня. Наприклад, ParcelRepository.trackParcel шукає посилку, отримує історію та перевіряє права доступу.

Папка widgets. Це повторно використані компоненти інтерфейсу: AppBar з логотипом, поле введення телефону, компонент введення SMS-коду та картка посилки. Такий поділ зменшує дублювання коду та спрощує підтримку інтерфейсу.

2.3 Проєктування моделі даних і бази Firestore

Для збереження даних використовується Cloud Firestore. Це NoSQL-база даних, де інформація зберігається у колекціях і документах. У проєкті виділено три основні колекції: users, parcels та traces. Така структура відповідає предметній області застосунку.

Колекція users зберігає інформацію про користувачів. Основні поля: id, name, phone_number та created_at. Поле id відповідає UID користувача з Firebase Authentication. Номер телефону використовується як основна контактна інформація. Загальну структуру каталогів розробленого проєкту представлено на рисунку 2.1.

Колекція parcels містить дані про посилки. Кожна посилка має id документа, number, description, weight, destination, startLocation, senderId,

receiverId та paid. Поля senderId і receiverId пов’язують посилку з користувачами.

Колекція traces містить історію відстеження. Кожен документ Trace описує одну подію доставки: статус, водія, посилку, опис, поточну локацію і час. Саме ця колекція дозволяє показати користувачу, як змінювався стан посилки.

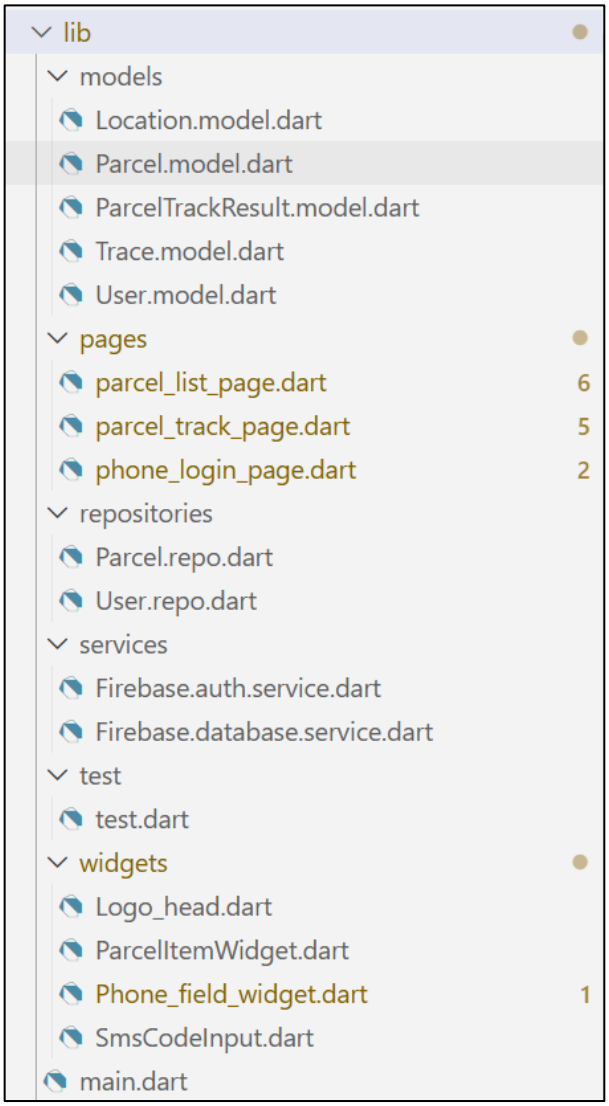


Рисунок 2.1 – Структура каталогів Flutter проєкту nova_post

Модель ParcelTrackResult є результатом пошуку посилки. Вона містить parcel, traces та canViewFullInfo. Ця модель особливо важлива для безпеки, оскільки дозволяє повернути історію навіть гостю, але приховати повну інформацію про посилку, якщо користувач не має прав (див. рис. 2.2).

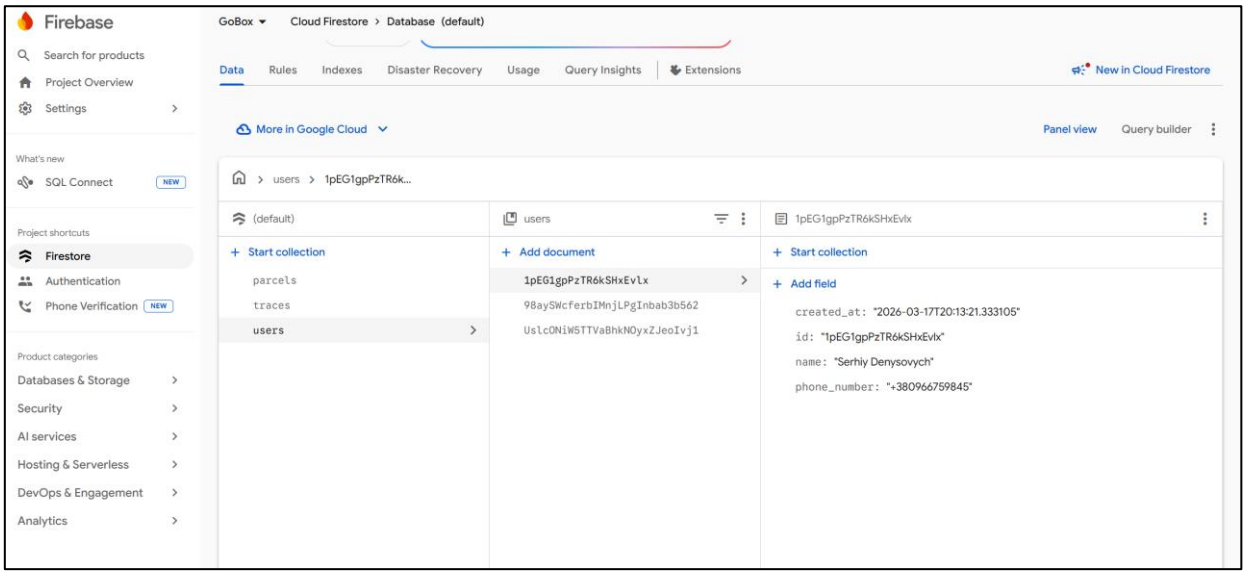


Рисунок 2.2 – Структура колекцій Firestore у застосунку

2.4 Проєктування інтерфейсу застосунку

Інтерфейс застосунку має темну кольорову схему з помаранчевим акцентом. Основний фон – темний, текст – білий або світло-сірий, акцентні елементи – помаранчеві. Така палітра створює контраст і виділяє ключові дії, зокрема кнопки пошуку та авторизації.

PhoneLoginPage є стартовим екраном. Він містить логотип «GO BOX», поле введення номера телефону, компонент введення SMS-коду після надсилання повідомлення, кнопку Send code або Verify code, а також можливість продовжити як гість. Цей екран реалізує весь сценарій входу користувача.

ParcelListPage призначений для перегляду списку посилок поточного користувача. Кожна посилка відображається через ParcelItemWidget. Користувач може натиснути кнопку Search і перейти до сторінки трекінгу. Також сторінка містить Drawer з кнопкою Logout і підтримує оновлення списку через RefreshIndicator.

ParcelTrackPage забезпечує пошук посилки за номером. Після введення номера та натискання кнопки пошуку застосунок звертається до репозиторію,

отримує результат і відображає або помилку, або історію трекінгу. Якщо користувач має права, додатково показується повна інформація про посилку та учасників доставки.

2.5 Проєктування бази даних

У межах розробки системи критично важливим етапом є проєктування шару збереження даних (Data Layer). Враховуючи специфіку архітектури фреймворку Flutter та орієнтацію на безсерверну інфраструктуру (Serverless), для збереження, обробки та синхронізації даних у застосунку використовується хмарна NoSQL-база даних Cloud Firestore від Google Firebase. На відміну від класичних реляційних СУБД (таких як PostgreSQL або MySQL), де дані нормалізуються і розподіляються по таблицях із жорсткими зв’язками (Foreign Keys), Cloud Firestore оперує концепціями колекцій (Collections) та документів (Documents), що містять пари «ключ-значення». Такий підхід забезпечує масштабування системи під високими навантаженнями та гарантує мінімальний час відгуку (Latency) під час виконання асинхронних запитів як у мобільних операційних системах (через нативні SDK), так і у веббраузерах (через компіляцію у JavaScript-запити).

Для мобільної версії застосунку, де логіка взаємодії передбачає глибоке розділення ролей, було спроєктовано нормалізовану NoSQL-структуру, яка імітує реляційні зв’язки типу один-до-багатьох. Дані розподілені між трьома взаємопов’язаними колекціями вищого рівня: users, parcels та traces (див. табл. 2.2 – 2.4).

Таблиця 2.2 – Структура документа колекції users

Назва поля	Тип даних	Опис поля
id	string	Унікальний UID сесії з сервісу Firebase Auth
phone_number	string	Верифікований номер телефону клієнта
name	string	Ім’я користувача для відображення в деталях доставки
created_at	string	Часова мітка реєстрації облікового запису

Таблиця 2.3 – Структура документа колекції parcels

Назва поля	Тип даних	Опис поля
id	string	Унікальний автоматичний ID документа
number	int	Числовий трекінг-номер посилки (наприклад, 10002)
description	string	Опис вмісту відправлення
weight	float	Вага вантажу в кілограмах
paid	boolean	Булевий статус фінансового розрахунку (сплачено)
startLocation_latitude	float	Координата широти початкової точки відправлення
startLocation_longitude	float	Координата довготи початкової точки відправлення
destination_latitude	float	Координата широти кінцевої точки доставки
destination_longitude	float	Координата довготи кінцевої точки доставки
senderId	string	Вказує на первинний ключ users.id відправника
receiverId	string	Вказує на первинний ключ users.id одержувача

Таблиця 2.4 – Структура документа колекції traces

Назва поля	Тип даних	Опис поля
id	string	Унікальний ідентифікатор події відстеження
status	string	Поточний стан посилки
description	string	Детальний текстовий опис поточного стану події
timestamp	string	Часова мітка фіксації події
currentLocation_latitude	float	GPS-координата широти поточного знаходження вантажу
currentLocation_longitude	float	GPS-координата довготи поточного знаходження вантажу
driverId	string	Вказує на users.id водія або оператора, який оновив статус
parcelId	string	Зв’язує подію з конкретним документом посилки

Схему бази даних мобільного застосунку наведено на рисунку 2.3.

На відміну від мобільної версії, для вебреалізації була розроблена така структура колекцій, як представлено у таблиці 2.5.

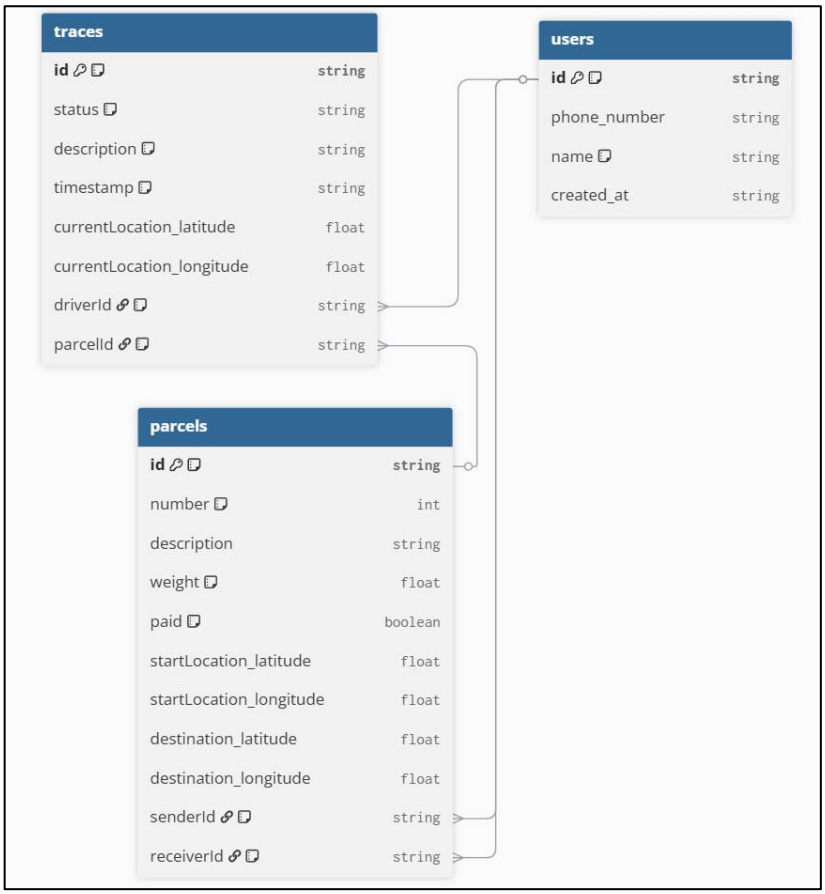


Рисунок 2.3 – Схема бази даних мобільного застосунку

Таблиця 2.5 – Структура документа єдиної колекції Parcel

Назва поля	Тип даних	Опис поля
parcelId	string	Первинний ключ, унікальний ідентифікатор документа
trackingNumber	string	Унікальний номер відправлення для пошуку через інтерфейс
title	string	Назва або короткий опис вмісту вантажу
status	string	Поточний статус логістичного ланцюга доставки
sender	string	Текстове ім’я або ідентифікатор особи-відправника
customer_name	string	Текстове ім’я одержувача (клієнта) посилки
phone	string	Контактний номер телефону для індексації та фільтрації
DateSHP	string	Фіксована дата відправлення у зручному рядковому форматі
DateARR	string	Фіксована дата прибуття вантажу у зручному рядковому форматі
deliveryAdress	string	Повна текстова адреса пункту призначення вантажу
isPayed	boolean	Булевий прапорець факту оплати доставки

На етапі проєктування вебреалізації було сформовано логічну модель колекції Parcels, схему та атрибути якої ілюструє рисунок 2.4.

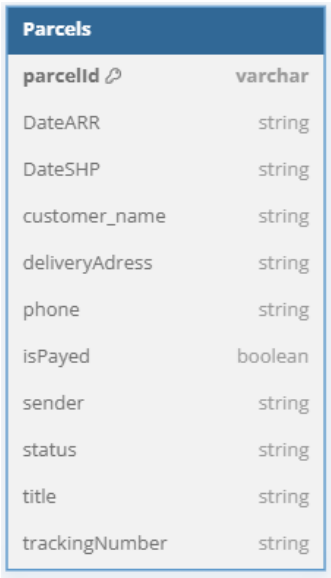


Рисунок 2.4 – Логічна схема бази даних вебзастосунку

Ефективність вебреалізації. Зберігання даних у єдиному документі Parcels виключає потребу у використанні віджетів FutureBuilder або додаткових циклів для завантаження імен користувачів за їхніми UID. Браузер отримує готовий текстовий масив за один мережевий запит через WebSocket, що дозволяє миттєво відобразити картку посилки на широкому Desktop-екрані та зменшує затримки в роботі інтерфейсу (UI blocking), забезпечуючи швидку роботу вебзастосунку.

Ефективність мобільної реалізації. Трикомпонентна структура (users, parcels, traces) оптимізована під нативні можливості мобільних SDK, зокрема під механізм автоматичного офлайн-кешування (Offline Persistence). Збереження історії переміщень в окремій колекції traces запобігає роздуванню документа посилки (який в Firestore обмежений розміром в 1 МБ), що дозволяє мобільному додатку ефективно синхронізувати лише нові події трекінгу при нестабільному мобільному інтернеті, заощаджуючи трафік пристрою.

Проєктування бази даних підтверджує, що кросплатформена розробка на Flutter дозволяє гнучко адаптувати шар даних під технічні вимоги

конкретного середовища виконання, поєднуючи надійність мобільної реляційної структури з високою швидкодією плоскої моделі вебзастосунку.

2.6 Моделювання процесів та архітектурних рішень

Основним елементом цього етапу є розробка діаграми варіантів використання (Use Case Diagram), яка дозволяє не залежати від конкретної програмної реалізації та визначити межі взаємодії між акторами (Actors) та прецедентами (Use Cases) системи.

У контексті дослідження порівняльної розробки мобільної та вебверсій побудова діаграми прецедентів демонструє, що цілі користувача та функціональні вимоги до системи є однаковими для обох платформ оскільки в обох випадках клієнт виконує однакові послідовні дії для досягнення результату. Ця діаграма відображає загальні функціональні вимоги до системи та визначає межі взаємодії між користувачем і програмою.

На рисунку 2.5 представлено розроблену діаграму варіантів використання системи для ключового актора предметної області.

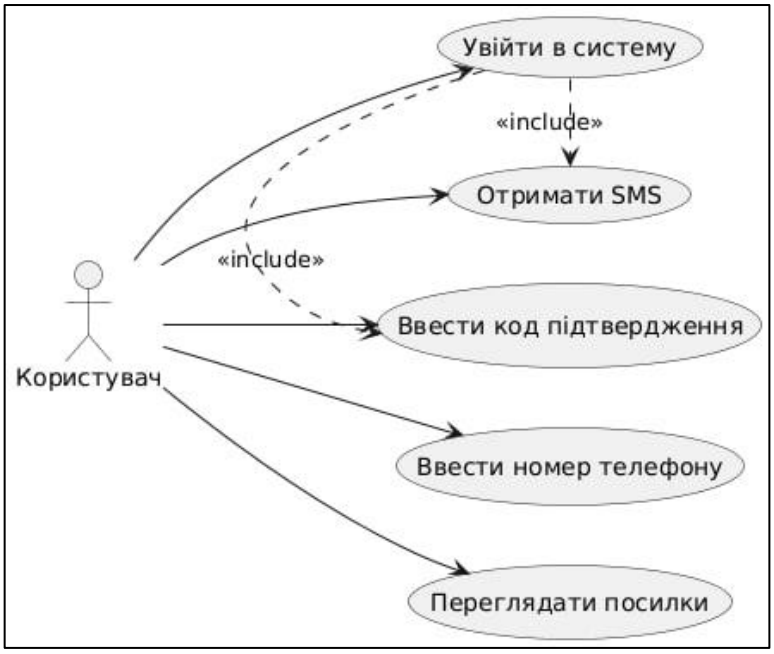


Рисунок 2.5 – Діаграма варіантів використання

Головним актором системи є «Користувач». Застосунок працює за таким принципом: користувач відкриває вебзастосунок, вводить свій номер телефону, отримує SMS з кодом підтвердження, вводить його у відповідне поле, після чого отримує доступ до списку активних та архівних посилок. Інформація про посилки зберігається в хмарній базі даних Firebase.

Розглянемо детальніше специфікацію одного з головних прецедентів системи – «Переглядати посилки». Його призначенням є забезпечення можливості перегляду поточного статусу посилок, пов’язаних з номером телефону користувача. Передумовою для початку взаємодії з даними про посилки є обов’язкове проходження автентифікації через введення номера телефону та коду підтвердження. В основному потоці подій після успішної автентифікації користувача система здійснює автоматичний запит до бази даних Firebase та відображає на екрані список відправлень з інформацією про поточний статус кожного з них. Альтернативний потік подій передбачає ситуацію, коли до номера телефону не прив’язані жодні посилки – у такому випадку система перериває побудову списку і виводить повідомлення «No parcels found»

Попри те, що загальна схема варіантів використання є спільною, її практичне втілення у мобільній та вебверсіях має суттєві відмінності у поведінці інтерфейсу. У мобільному додатку весь процес відбувається на одному пристрої, де екран орієнтований на сенсорний ввід, а системна клавіатура автоматично відкривається у числовому форматі. Оновлення списку посилок тут відбувається через нативний жест свайпу вниз, а на операційній системі Android можливе навіть автоматичне перехоплення та введення коду з SMS.

У вебверсії цей процес часто стає крос-девайсним, оскільки користувач може відкривати застосунок у браузері на комп’ютері, а повідомлення з кодом отримувати на мобільний телефон. Це вимагає ручного введення цифр клавіатурою з підтримкою фокусування на полях за допомогою клавіші Tab. Крім того, вебреалізація потребує особливого управління макетом сторінки на широких моніторах ПК. Щоб блоки не втрачали логіку розміщення і контент

не розтягувався, форму введення та списки посилок огорнуто у віджет Center, а максимальну ширину обмежено за допомогою Container. Це дозволяє зберегти компактний вигляд вебформи та забезпечує зручність використання сервісу на будь-якій платформі.

Таким чином, розробка UML-моделі варіантів використання доводить, що за повної ідентичності концептуальних бізнес-процесів, реалізація системи вимагає від кросплатформеного фреймворку Flutter гнучкого розділення логіки взаємодії, що було успішно досягнуто під час проєктування.

Статичне представлення структури застосунку зображено на діаграмі класів (див. рис. 2.6), де представлено основні класи, їх атрибути та методи.

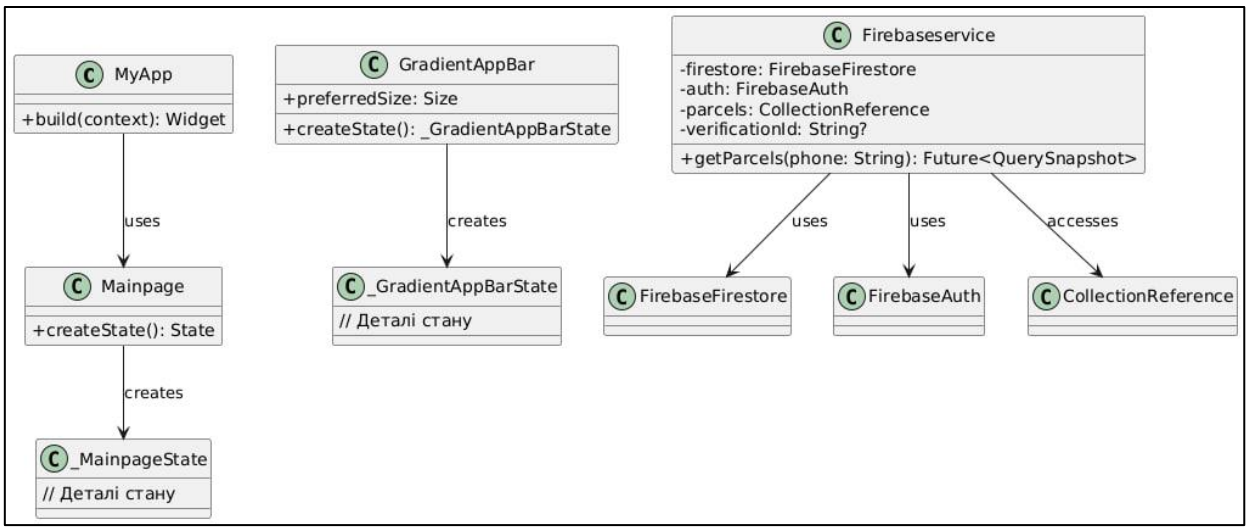


Рисунок 2.6 – Діаграма класів

Клас MyApp виступає точкою входу в програму. Він ініціалізує головну сторінку Mainpage, яка безпосередньо відповідає за рендеринг інтерфейсу. Візуальне оформлення верхньої частини екрана винесено в окремий клас GradientAppBar. Уся взаємодія з хмарною інфраструктурою винесена в окремий клас Firebaseservice, який містить посилання на об’єкти FirebaseFirestore та FirebaseAuth.

Ця структура класів є спільною для веб та мобільної реалізації, що є одною з переваг фреймворку Flutter. Розробнику не потрібно писати окремі класи сервісів для різних платформ. Проте на етапі виконання (Runtime) клас

Mainpage динамічно адаптує своє дерево віджетів: у браузері він активує обмежувачі ширини (ConstrainedBox), а на смартфоні дозволяє контенту займати весь екран.

Фізичну архітектуру та залежності між модулями системи відображає діаграма компонентів, які показані на рисунку 2.7.

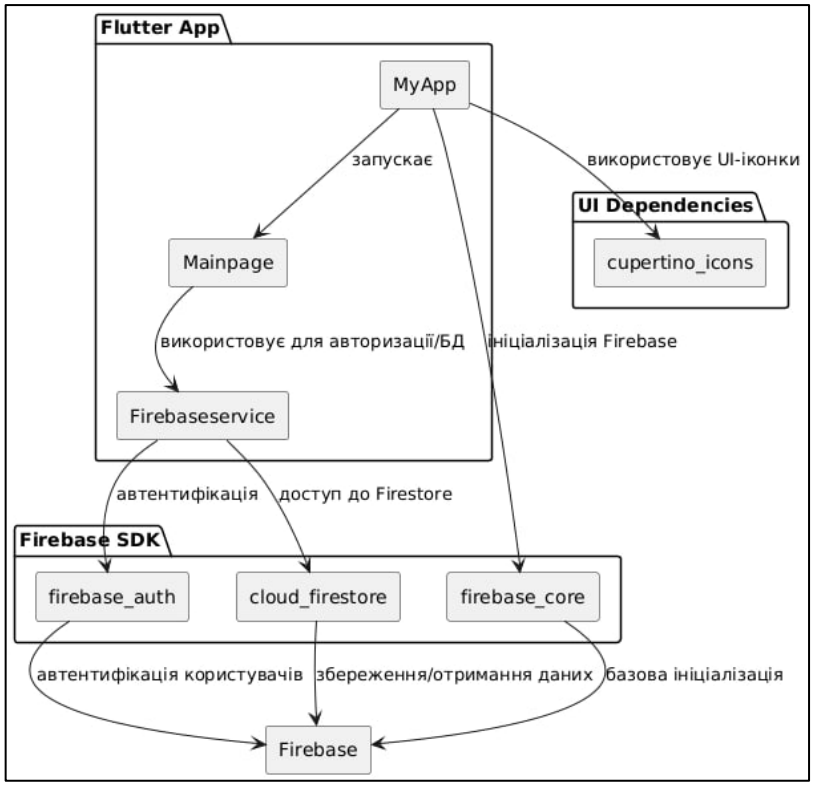


Рисунок 2.7 – Діаграма компонентів

Архітектура складається з трьох основних блоків: самого Flutter-додатка, UI-залежностей (наприклад, пакет cupertino_icons для відтворення нативних іконок) та модуля Firebase SDK. Останній містить основні бібліотеки: firebase_core (для базової ініціалізації), firebase_auth (для авторизації) та cloud_firestore (для збереження даних).

На діаграмі компонентів імпортуються одні й ті самі пакети, їхня фізична компіляція відрізняється. При збиранні мобільного додатка бібліотека cloud_firestore компілюється як міст до нативних SDK (Java/Kotlin для Android, Swift для iOS). Натомість при збиранні вебзастосунку цей самий компонент трансформується у JavaScript-код, який використовує веб-API Firebase. Це

забезпечує сумісність: єдиний компонент на діаграмі автоматично використовує найефективніший метод роботи під капотом цільової платформи.

Для візуалізації динаміки роботи системи та послідовності викликів під час авторизації розроблено діаграму послідовностей. Послідовність виконання кожного етапу продемонстровано на рисунку 2.8.

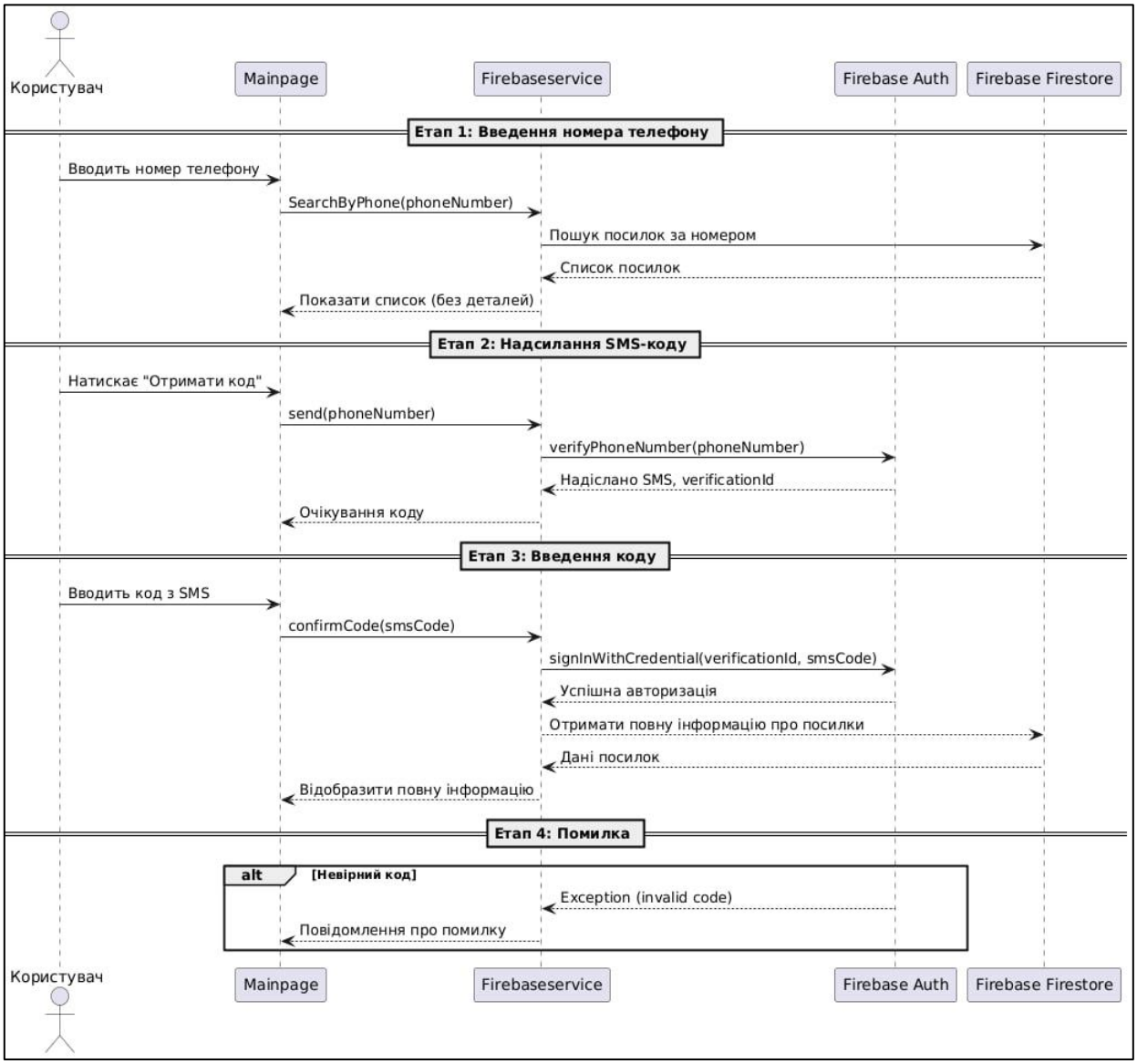


Рисунок 2.8 – Діаграма послідовностей

Процес розбито на чотири логічні етапи. На першому етапі користувач вводить номер телефону, а клас Mainpage передає його у Firebaseservice. Сервіс виконує запит до бази даних і повертає загальний список посилок без

конфіденційних деталей. На другому етапі ініціюється надсилання SMS: виклик функції verifyPhoneNumber передається до Firebase Auth, який повертає унікальний ідентифікатор сесії (verificationId). Третій етап передбачає введення коду: система формує облікові дані та викликає signInWithCredential. Після успішної авторизації система повторно звертається до Firestore, щоб завантажити повні, незнеособлені дані про посилки. Четвертий етап демонструє альтернативний сценарій: якщо код хибний, генерується виключення (Exception), і користувач бачить повідомлення про помилку.

Послідовність викликів API на діаграмі є абсолютно ідентичною для веб та мобільного середовища. Різниця полягає лише у тривалості життєвого циклу Етапу 2 (Очікування коду) та Етапу 4 (Помилка). У вебзастосунку користувач може закрити вкладку під час очікування SMS, що вимагає механізмів збереження стану сесії в LocalStorage браузера. У мобільному додатку цей процес відбувається безперервно в межах одного активного вікна операційної системи.

2.7 Проєктування системи обробки помилок та виключних ситуацій

Проєктування підсистеми обробки помилок та виключних ситуацій є одним із найважливіших етапів розробки кросплатформеного застосунку «GO BOX». Оскільки надійна робота логістичного сервісу трекінгу безпосередньо залежить від стабільності мережевого з’єднання та коректності дій користувача, виникає потреба у створенні відмовостійкої архітектури. Порівняльний аналіз середовищ виконання (браузера ПК та мобільної ОС) виявив, що за ідентичної бізнес-логіки архітектурні підходи до перехоплення та обробки збоїв мають глибокі платформні розбіжності.

На першому рівні – валідації користувацького введення – відмінності зумовлені різними фізичними пристроями введення даних. У мобільній версії

більшість помилок нівелюється на етапі проєктування інтерфейсу користувача: наприклад, при виклику поля введення номера телефону або SMS-коду застосунок жорстко обмежує віртуальну клавіатуру лише цифровим макетом, що фізично унеможливлює введення літерних символів.

Натомість у вебверсії користувач взаємодіє з фізичною клавіатурою ПК і може вводити будь-які спецсимволи або копіювати некоректний текст мишею. Через це вебклієнт потребує складнішого проєктування регулярних виразів (RegEx) та масок введення на рівні контролерів, які динамічно блокують некоректні натискання клавіш та миттєво підсвічують форму введення червоним кольором.

На другому рівні – автентифікації через Firebase Auth – виникає серйозна архітектурна розбіжність у перехопленні виключних ситуацій безпеки. Для захисту від спаму та роботи бот-мереж хмарна інфраструктура Firebase вимагає обов'язкової верифікації клієнта. У мобільній версії це реалізується непомітно для користувача за допомогою нативних сервісів операційної системи (Google Play Services для Android або SafetyNet/DeviceCheck для iOS), які у фоновому режимі підтверджують легітимність пристрою. У вебсередовищі такий нативний механізм відсутній, тому Firebase ініціює запуск капчі (reCAPTCHA) безпосередньо у вікні браузера.

Під час проєктування вебреалізації доводиться закладати обробку специфічних браузерних помилок: відмови користувача від проходження капчі, блокування скриптів reCAPTCHA встановленими в браузері розширеннями (AdBlock, Privacy Badger) або помилок CORS при нестабільному з'єднанні. Усі ці винятки перехоплюються через конструкції try-catch та трансформуються у зрозумілі Snackbar-сповіщення у футері вебсторінки.

Найбільша розгалуження простежується на третьому рівні – обробці помилок мережевого з'єднання під час взаємодії з NoSQL-базою даних Cloud Firestore. Мобільні пристрої за своєю природою працюють у динамічному середовищі, де користувач може постійно втрачати сигнал зв'язку (наприклад, у ліфті чи транспорті).

Для мобільного клієнта «GO BOX» проєктне рішення базується на використанні нативного механізму Offline Persistence. Коли мобільний SDK Firestore перехоплює помилку відсутності мережі, він не зупиняє роботу програми, а прозоро перемикає запити на читання об’єктів parcels та traces в оперативній пам’яті пристрою.

Вебзастосунок (SPA) працює за абсолютно іншою логікою. Браузерна версія не має глибокої інтеграції з локальною пам’яттю системи. Будь-який критичний розрив мережі миттєво генерує виключення в потоці даних. Якщо в цей момент користувач оновить сторінку через клавішу F5, локальний стан оперативної пам’яті повністю знищиться.

Тому підсистема обробки помилок у вебверсії проєктується так, щоб негайно реагувати на втрату зв’язку: інтерфейс не просто підтягує старі дані, а блокує екран пошуку та відображає для користувача інтегроване візуальне повідомлення про необхідність перевірити підключення до Інтернету, забезпечуючи стабільність вебзастосунку, що продемонстровано на рисунку 2.9.

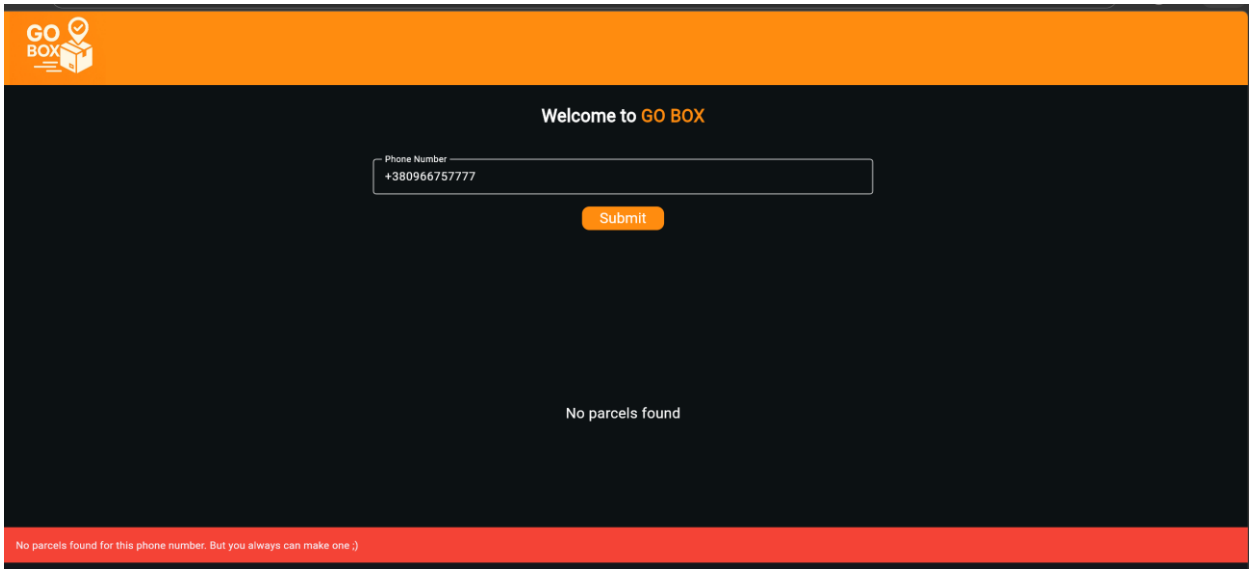


Рисунок 2.9 – Обробка помилки у вебверсії

Отже, реалізований механізм обробки помилок гарантує, що будь-які ситуації – від некоректного введення даних до втрати зв’язку з сервером – будуть коректно перехоплені системою.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ТА МОБІЛЬНОГО ДОДАТКУ

3.1 Реалізація інтерфейсу користувача та конфігурація навігації

Графічний інтерфейс користувача (GUI) системи «GO BOX» реалізовано за допомогою декларативного підходу Flutter, де стан додатка безпосередньо визначає його візуальне представлення. Відповідно до проєктних рішень, інтерфейс виконано в темній палітрі з використанням Material Design 3. Програмна реалізація кожного екрана та системи переходів має свої технічні особливості, зумовлені кросплатформеною специфікою.

Головною інженерною особливістю реалізації є те, що папка lib є абсолютно спільною, проте архітектура віджетів містить адаптивні механізми, які диференціюють поведінку інтерфейсу та навігації залежно від цільової платформи запуску: мобільних ОС (Android/iOS) або веббраузерів.

Реалізація інтерфейсу автентифікації та компонентів введення. При запуску проєкту на мобільних пристроях та у веббраузері інтерфейс адаптується до різних парадигм введення даних та архітектурних моделей управління екранами: мобільна реалізація та вебреалізація.

Мобільна реалізація: процес автентифікації винесено в окремий ізольований екран PhoneLoginPage (phone_login_page.dart), реалізований як StatefulWidget. Виклик віджета PhoneInputWidget (на базі плагіна intl_phone_field) автоматично ініціює підняття системної віртуальної цифрової клавіатури смартфона. Компонент SmsCodeInput оперує масивом із шести окремих текстових полів, де перехід фокуса через об’єкти FocusNode оптимізовано під швидкий сенсорний ввід (Touch input) – фокус автоматично переміщується на наступне поле при введенні однієї цифри.

Вебреалізація: функціонал автентифікації інтегровано безпосередньо в головний кореневий віджет Mainpage (MainPages.dart), який виконує роль

єдиної вебсторінки. Оскільки у браузері віртуальна клавіатура відсутня, обробку введення повністю переведено на фізичну клавіатуру ПК з підтримкою фокусування елементів за допомогою клавіші Tab. Для запобігання дефекту розтягування інтерфейсу на широких моніторах, форму введення огорнуто у віджет Center, а її максимальну ширину обмежено за допомогою контейнера з фіксованим параметром width. Введення номера телефону контролюється об'єктом phoneNumberController. Після виклику методу verifyPhoneNumber через Firebaseservice логіка перемикається на перевірку коду за допомогою об'єкта PhoneAuthProvider.credential. Візуальне представлення відповідних екранів автентифікації у веб та мобільному застосунку наведено на рисунках 3.1 і 3.2.

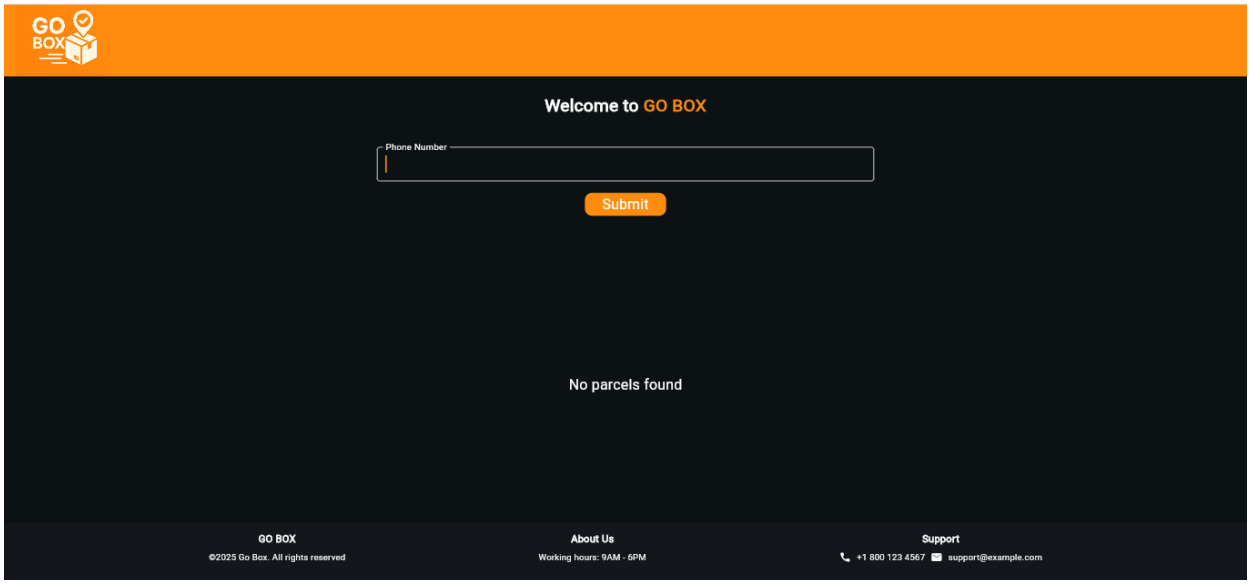


Рисунок 3.1 – Екран автентифікації системи «GO BOX» у вебінтерфейсі

Реалізація інтерфейсу списку відправлень та адаптивних списків. Відображення масиву посилок користувача на обох платформах покладається на віджет ListView.builder, який реалізує патерн лінивого завантаження контенту в межах Viewport, проте взаємодія та розмітка мають суттєві відмінності.

Мобільна реалізація: список відправлень винесено на окрему сторінку ParcelListPage (parcel_list_page.dart). Оновлення даних покладено на

мобільний жест свайпу вниз за допомогою віджета RefreshIndicator (патерн Pull-to-refresh), який викликає асинхронний метод fetchData(). Елементи списку рендеряться через кастомний ParcelItemWidget і займають 100% ширини екрана смартфона для зручності натискання пальцем. Додатково інтерфейс містить нативне бічне меню Drawer для виклику функції Logout.

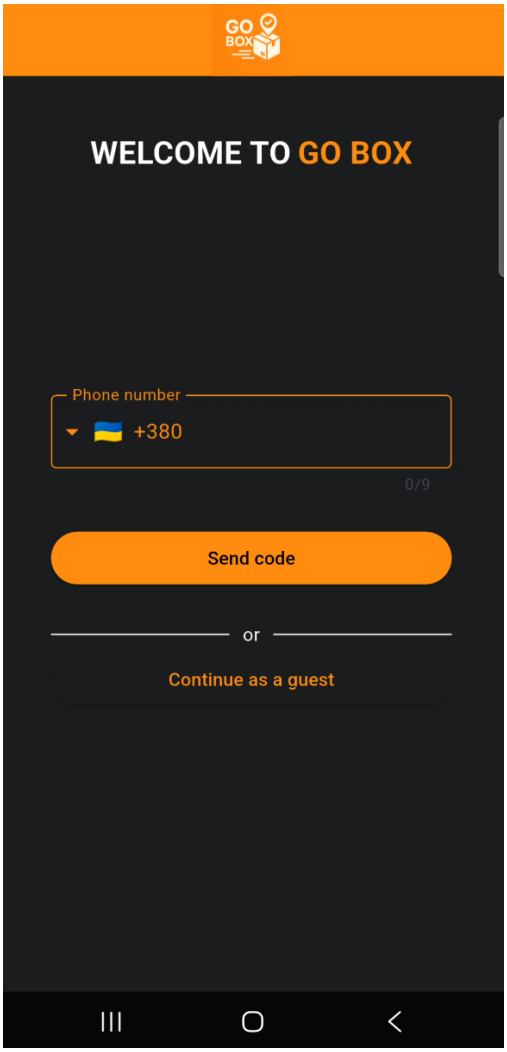


Рисунок 3.2 – Екран автентифікації системи «GO BOX»
у мобільному застосунку

Вебреалізація: список посилок відображається у межах тієї ж сторінки Mainpage за допомогою механізму умовного рендерингу (Conditional Rendering). Коли локальний булевий прапорець стану showParcelDetails набуває значення true після перевірки SMS, дерево віджетів динамічно перебудовується всередині базового Scaffold. Список розміщується у зв’язці

віджетів Expanded та Container і автоматично вирівнюється по центру екрана браузера. Якщо NoSQL-запит до колекції Parcels повертає порожній зріз даних, вебінтерфейс відразу відображає інтегрований стан No parcels found. Замість мобільного меню додано фіксований нижній елемент BottomAppBar, який виконує роль інформативного вебфутера. Зовнішній вигляд представлено на рисунках 3.3 і 3.4.

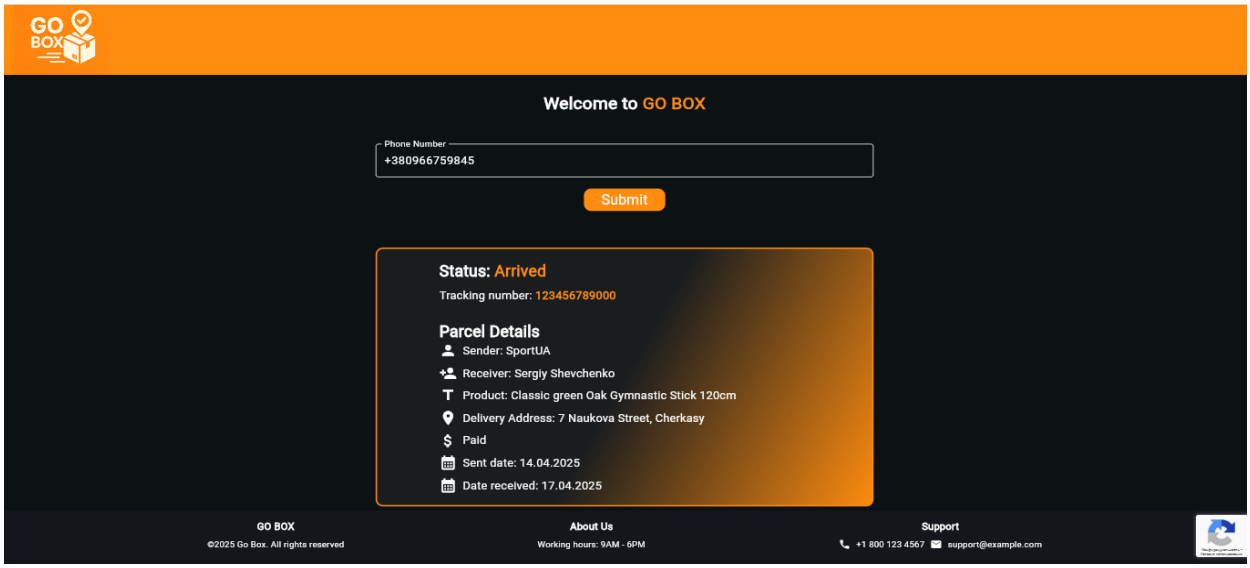


Рисунок 3.3 – Інтерфейс списку посилок у браузері
та на мобільному пристрої

Конфігурація кросплатформеної навігації та управління стеком. Найбільш критичний аспект порівняння платформ полягає в конфігурації логіки переходів та взаємодії із системними кнопками керування.

Мобільний аспект: навігація реалізована через систему іменованих маршрутів (Named Routes) у main.dart. Системний менеджер Navigator маніпулює стеком екранів додатка. При успішній автентифікації або при виході з системи додаток викликає оператор Navigator.pushReplacementNamed(context, ‘/login’). Це повністю видаляє попередній приватний екран із пам’яті навігаційного стеку, завдяки чому натискання користувачем апаратної чи системної кнопки «Назад» на Android-пристрої не повертає його до конфіденційних даних, а безпечно закриває застосунок.

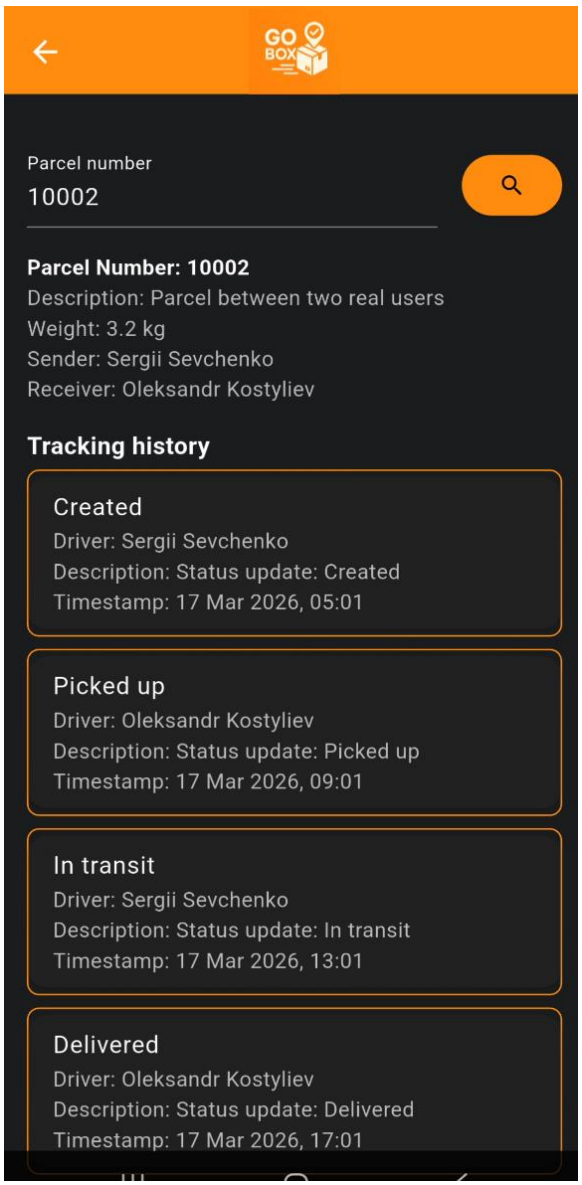


Рисунок 3.4 – Інтерфейс списку посилок на мобільному пристрої

Вебаспект: логіка навігації побудована на базі станів всередині єдиного маршруту home: const Mainpage(). Переходи між екраном пошуку, формою введення SMS та детальними картами посилок (sender, customer, title, deliveryAdress, isPayed, DateSHP, DateARR) імітуються реактивною зміною прапорців showSms та showParcelDetails. Для вирішення специфічної для вебу проблеми повного знищення стану додатка в оперативній пам’яті при натисканні клавіші оновлення сторінки (F5) – задіяно конфігурацію тривалого збереження сесії (Persistence) на рівні ядра Firebase Auth, яка автоматично зчитує токен із локального сховища браузера та відновлює актуальний стан інтерфейсу, що проілюстровано у порівняльній таблиці 3.1.

Таблиця 3.1 – Комплексне зіставлення реалізації UI та навігації у кросплатформеному середовищі

Назва критерію	**Реалізація у мобільному додатку (Android/iOS)**	**Реалізація у вебзастосунку (браузер ПК)**
Архітектурний каркас UI	Розподілена система класів-сторінок (PhoneLoginPage, ParcelListPage).	Односторінкова модель (монолітний StatefulWidget класу Mainpage).
Управління макетом (Layout)	Нативний повноекранний рендеринг під фіксовані габарити дисплея смартфона.	Використання віджетів Center та ConstrainedBox для оптимізації під Desktop-монітори.
Парадигма введення та фокус	Сенсорний ввід (Touch), автоматичний виклик системної цифрової клавіатури.	Фізична клавіатура та миша, перехід між елементами через клавішу Tab.
Механізм оновлення списків	Нативний жест прокручування екрана зверху вниз (RefreshIndicator).	Стандартний вебскролінг мишею, умовне відображення стану «No parcels found».
Модель навігації та переходи	Системний стек переходів Navigator на базі конфігурації Named Routes.	Модель умовного рендерингу (Conditional Rendering) за прапорцями стану (showSms, showParcelDetails).
Конфігурація Firebase інфраструктури	Компіляція статичних нативних файлів конфігурації у бінарний код додатка.	Динамічне зчитування параметрів firebaseConfig з контексту вікна через dart:js.
Статичні елементи навігації	Виклик нативного прихованого бічного меню Drawer для здійснення Logout.	Інтеграція фіксованого елемента BottomAppBar як стандартного вебфутера сторінки.

Таким чином, використання фреймворку Flutter дозволило реалізувати складну кросплатформену логіку інтерфейсу системи «GO BOX» без дублювання коду, забезпечивши при цьому нативний досвід взаємодії для мобільних користувачів та стандартну ергономіку для користувачів веббраузерів.

3.2 Порівняння веб та мобільної реалізації Flutter

Порівняння веб і мобільної реалізації доцільно почати з того, що основна кодова база є спільною. Папка lib містить сторінки, моделі, сервіси, репозиторії та віджети, які використовуються незалежно від того, чи запускається застосунок на телефоні, чи у браузері.

Для мобільного додатку важливим є відчуття нативності. Користувач очікує, що екран авторизації зручно відкриватиметься на смартфоні, SMS-код вводитиметься швидко, список посилок прокручуватиметься плавно, а кнопка назад або Drawer працюватимуть звично.

Для вебзастосунку важливо інше: сторінка повинна коректно відкриватися у браузері, не втрачати читабельність на широкому екрані та не виглядати надто розтягнутою. Якщо інтерфейс створений переважно для телефону, у вебверсії може знадобитися обмеження ширини основного контейнера.

Робота з базою даних також спільна. І мобільна, і вебверсія звертаються до Firestore. Це означає, що список посилок, історія доставки, користувачі та права доступу не дублюються.

Процес збірки різний. Для Android результатом може бути APK або App Bundle. Для веб результатом є файли у build/web, які публікуються на вебсервері. Для розробника це означає, що код спільний, але етапи релізу різні (див. рис. 3.5, 3.6).

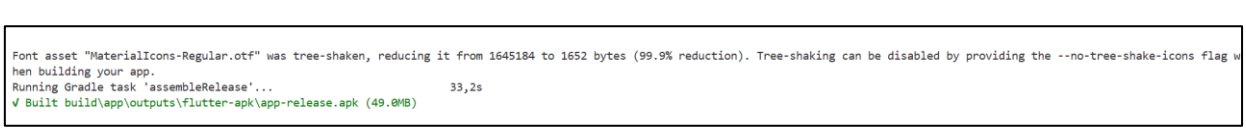


Рисунок 3.5 – Процес збірки мобільної версії

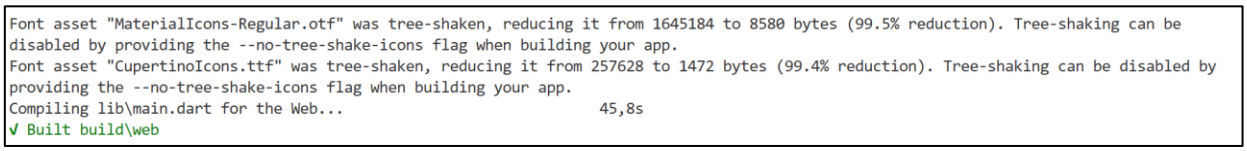


Рисунок 3.6 – Процес збірки вебверсії

У межах теми важливо підкреслити, що розробка «GO BOX» не зводиться тільки до створення екранів. Кожен елемент інтерфейсу пов’язаний із певною дією користувача: введенням номера телефону, переходом до списку посилок, пошуком номера відправлення або переглядом подій доставки. Через це опис функціональності в роботі подано разом із поясненням, які файли та класи відповідають за цю функціональність.

Для вебреалізації аспект «порівняння веб та мобільної реалізації» проявляється через потребу у коректному відображенні в браузері. Користувач може відкривати застосунок на широкому екрані, тому важливо, щоб блоки не втрачали логіку розміщення. У мобільному додатку той самий аспект оцінюється через зручність дотику, розмір полів, простоту введення та швидкість переходів між сторінками.

У проєкті «GO BOX» це рішення підтримується тим, що основна логіка не прив’язана до конкретної платформи. Сторінки, моделі, сервіси та репозиторії розміщені в папці lib і можуть використовуватися як для Android, так і для вебверсії. Саме тому у роботі порівняння веб та мобільного варіанта виконується не як два окремі проєкти, а як дві форми запуску однієї Flutter-кодової бази.

Практичний зміст цього аспекту видно через такі елементи: Android-збірка, web-збірка, Firestore, Firebase Auth, адаптивність, спільна папка lib. Вони не є випадковим набором файлів, а разом утворюють послідовний сценарій роботи застосунку. Якщо змінюється один з елементів, наприклад структура моделі Parcel або логіка ParcelRepository, це впливає і на мобільний, і на вебваріант застосунку.

3.3 Порівняння поведінки Flutter-проєкту на різних платформах

Поведінка Flutter-застосунку на мобільній платформі пов’язана з сенсорним керуванням. Кнопки повинні бути достатньо великими, поля

введення – зрозумілими, а навігація – короткою. Це особливо помітно на екрані SMS-коду та сторінці пошуку, де користувач взаємодіє з формами.

У вебверсії змінюється не логіка, а простір розміщення. Браузерне вікно може бути дуже широким, тому інтерфейс не повинен розтягуватися без контролю. Якщо поле телефону займає надто велику ширину, сторінка виглядає незбалансовано. Якщо ж контент залишити занадто вузьким, вебкористувач не отримає переваги від великого екрана.

Звідси випливає головний висновок порівняння: Flutter дає спільну основу, але не звільняє від адаптивного мислення. Розробник має перевіряти форми, списки, картки, drawer та повідомлення про помилки в кожному середовищі. Без цього формальна кросплатформеність не перетворюється на якісний користувацький досвід (див. табл. 3.2).

Таблиця 3.2 – Порівняння вебзапуску та мобільного запуску

Ознака	**Мобільний додаток**	**Вебзастосунок**
Основний пристрій	смартфон або планшет	браузер на ПК чи ноутбуці
Введення даних	сенсорна клавіатура	фізична клавіатура або екранна
Розмір інтерфейсу	обмежена ширина	гнучке вікно браузера
Сценарій використання	швидка перевірка статусу	демонстрація або розширений перегляд
Ризик UX	дрібні елементи	надмірно розтягнуті блоки

3.4 Тестування та безпека застосунку

Тестування має охоплювати основні сценарії: запуск застосунку, введення номера телефону, відправку SMS-коду, введення коду, перехід до списку посилок, пошук посилки за номером, перегляд історії доставки і вихід.

Окремо потрібно перевіряти механізми перехоплення та обробки помилок. Цей етап є критичним для забезпечення якісного користувацького

досвіду. Наприклад, якщо номер телефону введено неправильно або використано недопустимі символи, користувач має миттєво отримати зрозуміле локалізоване повідомлення про помилку валідації без відправки зайвих запитів на сервер. Якщо SMS-код неправильний, неповний або час дії коду (таймаут сесії) вже завершився, застосунок також повинен не просто відхилити доступ, а показати конкретну причину помилки, перехопивши відповідне виключення (Exception) від Firebase.

Безпека у проєкті пов’язана передусім з авторизацією та суворим контролем доступу до даних. Базовий рівень захисту реалізується за допомогою сервісу Firebase Authentication, який забезпечує надійну перевірку номера телефону через SMS. Проте сама авторизація лише підтверджує особу користувача, але не гарантує правильного та безпечного доступу до документів у базі даних Firestore.

Гостьовий режим (неавторизований доступ) також потребує особливої обережності. З одного боку, він є вкрай корисним інструментом для швидкого відстеження посилки сторонньою особою без необхідності проходити повну процедуру реєстрації. З іншого боку, гість за жодних обставин не повинен бачити імен, номерів телефонів сторін, точні адреси або інші особисті дані. Для цього у застосунку реалізовано механізм фільтрації, який повертає гостю виключно статуси доставки.

Окремої уваги під час тестування потребують асинхронні операції. Firebase-запити не виконуються миттєво і їхня тривалість сильно залежить від якості інтернет-з’єднання на пристрої клієнта. Тому графічний інтерфейс має обов’язково показувати стан очікування або хоча б не створювати враження зависання програми.

На сторінці відстеження це особливо важливо, оскільки користувач чекає на результат пошуку і, без індикатора, може багатократно повторно натиснути кнопку оновлення, що призведе до дублювання запитів і перевантаження бази даних. Приклад базових сценаріїв тестування та перевірок безпеки наведено у таблиці 3.3.

Таблиця 3.3 – Приклади перевірок для «GO BOX»

Сценарій	**Що перевіряється**	**Очікуваний результат**
Вхід	номер телефону і SMS-код	перехід до списку посилок
Пошук	існуючий номер посилки	відображення історії traces
Помилка	неправильний номер	зрозуміле повідомлення
Доступ	сторонній користувач	приховані приватні дані
Вихід	натискання logout	повернення на сторінку входу

Проходження цих тестових перевірок дозволяє мінімізувати ризик виникнення критичних помилок на етапі використання. Це забезпечує стабільну роботу кросплатформеного застосунку та високий рівень безпеки користувацьких даних.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра було успішно вирішено актуальне завдання сучасної програмної інженерії – проведено порівняльне дослідження процесів розробки вебзастосунку та мобільного додатку з використанням кросплатформеного фреймворку Flutter. Практичною площиною для дослідження стала розробка програмної системи логістичного трекінгу «GO BOX».

Результати проведеного аналізу, проєктування та програмної реалізації дали змогу сформулювати фінальні висновки роботи.

Аналіз теоретичних та технологічних засад: використання фреймворку Flutter та мови програмування Dart дозволяє ефективно вирішити проблему дублювання бізнес-логіки, характерну для нативної розробки. Використання єдиного репозиторію суттєво скорочує час випуску продукту. Проте, теоретичний аналіз виявив, що кросплатформеність не скасовує необхідності адаптації архітектурних рішень під парадигми взаємодії різних середовищ (сенсорні екрани мобільних ОС та широкі монітори персональних комп’ютерів із клавіатурою).

Моделювання бізнес-процесів: за допомогою інструментарію UML було доведено однаковість бізнес-логіки: цілі користувача (авторизація, пошук посилки, перегляд трекінгу) є абсолютно ідентичними для обох платформ. Водночас деталізація прецедентів показала, що фізичне втілення цих процесів вимагає різних підходів. Наприклад, сценарій SMS-автентифікації на смартфоні оптимізовано під автоматичне перехоплення коду операційною системою, тоді як у вебреалізації він вимагає врахування крос-девайсної взаємодії, триваліших таймаутів сесії та захисту від бот-мереж (reCAPTCHA).

Проєктування архітектури та баз даних: упровадження безсерверної (Serverless) інфраструктури Firebase дозволило реалізувати єдиний бекенд для

всіх платформ. Однак, порівняльне проєктування бази даних Cloud Firestore виявило необхідність застосування різних концептуальних моделей. Для мобільного додатка найбільш ефективною виявилася трикомпонентна нормалізована структура (колекції users, parcels, traces), яка оптимізує споживання трафіку та дозволяє використовувати нативне офлайн-кешування. Для вебзастосунку спроєктовано модель єдиної колекції Parcels, яка мінімізує кількість асинхронних запитів та забезпечує миттєвий рендеринг інтерфейсу без блокування основного потоку браузера.

Програмна реалізація інтерфейсу та навігації: практична реалізація системи «GO BOX» продемонструвала гнучкість Flutter в адаптації UI-шару. У мобільному додатку інтерфейс використовує повноекранний рендеринг, жести Pull-to-refresh та системний навігаційний стек маршрутів (Navigator.pushReplacementNamed), що блокує несанкціоноване повернення назад апаратними кнопками. У вебзастосунку застосовано умовний рендеринг (Conditional Rendering) на базі станів в межах одного маршруту, обмежувачі ширини макета (ConstrainedBox) та механізми персистентного збереження токенів для захисту від знищення сесії при оновленні сторінки браузера.

Мета кваліфікаційної роботи досягнута повністю. Розроблена система «GO BOX» функціонує стабільно. Дослідження доводить, що найкращих результатів у Flutter-розробці можна досягти, коли ядро бізнес-логіки залишається повністю спільним, а UI-шар та обмін даними проєктуються з урахуванням апаратних, мережевих та навігаційних обмежень конкретного середовища виконання. Отримані результати можуть бути використані як архітектурний шаблон для створення нових масштабованих кросплатформених систем.

ПЕРЕЛІК ПОСИЛАНЬ

1. Flutter documentation. *Flutter*. URL: https://docs.flutter.dev (дата звернення: 30.01.2026).

2. Dart language overview. *Dart*. URL: https://dart.dev/overview (дата звернення: 30.01.2026).

3. Firebase Authentication documentation. *Firebase*. URL: https://firebase.google.com/docs/auth (дата звернення: 04.02.2026).

4. Cloud Firestore documentation. *Firebase*. URL: https://firebase.google.com/docs/firestore (дата звернення: 04.02.2026).

5. Material Design 3. *Material Design*. URL: https://m3.material.io/ (дата звернення: 10.02.2026).

6. Intl package. *Pub.dev*. URL: https://pub.dev/packages/intl (дата звернення: 06.02.2026).

7. Intl_phone_field package. *Pub.dev*. URL: https://pub.dev/packages/intl_phone_field (дата звернення: 06.02.2026).

8. Cloud_firestore package. *Pub.dev*. URL: https://pub.dev/packages/cloud_firestore (дата звернення: 13.02.2026).

9. Firebase_auth package. *Pub.dev*. URL: https://pub.dev/packages/firebase_auth (дата звернення: 16.02.2026).

10. Android developers documentation. *Android Developers*. URL: https://developer.android.com (дата звернення: 18.02.2026).

11. Progressive web apps. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (дата звернення: 19.02.2026).