

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

## Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОНЛАЙН  
МАРКЕТПЛЕЙСУ  
SOFTWARE DEVELOPMENT OF A MOBILE APPLICATION FOR  
AN ONLINE MARKETPLACE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-210

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ГРИЩЕНКОВ М.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ФЕДОРОНЧАК Т.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЛИПОВЕЦЬ М.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
Національний університет «Запорізька політехніка»  
(повне найменування закладу вищої освіти)

Факультет КНТ  
Кафедра програмних засобів  
Ступінь вищої освіти бакалавр  
Спеціальність 122 Комп'ютерні науки  
(код і найменування)  
Освітня програма (спеціалізація) Комп'ютерні науки  
(назва освітньої програми (спеціалізації))

**ЗАТВЕРДЖУЮ**

Завідувач кафедри ПЗ, д.т.н, проф.  
Сергій СУББОТІН  
“ ” 2024 року

**З А В Д А Н Н Я**

**НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)**

ГРИЩЕНКОВА Максима Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного застосунку для онлайн маркетплейсу Software Development of a Mobile Application for an Online Marketplace

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,  
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

## 6. Консультанти розділів проєкту (роботи)

| Розділ              | ПРІЗВИЩЕ, ініціали та посада консультанта | Підпис, дата   |                           |
|---------------------|-------------------------------------------|----------------|---------------------------|
|                     |                                           | завдання видав | прийняв виконане завдання |
| 1-4 Основна частина | ФЕДОРОНЧАК Т.В., доцент                   |                |                           |
| Нормоконтроль       | ЛИПОВЕЦЬ М.В., асистент                   |                |                           |

7. Дата видачі завдання « 15 » квітня 2024 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів дипломного проєкту (роботи)                             | Строк виконання етапів проєкту ( роботи ) | Примітка     |
|-------|----------------------------------------------------------------------|-------------------------------------------|--------------|
| 1     | Постановка завдання роботи.                                          | 1 тиждень                                 | Завдання, ТЗ |
| 2     | Аналіз предметної області.                                           | 1 тиждень                                 | Розділ 1     |
| 3     | Вибір мови програмування та інших технологій розробки.               | 2 тиждень                                 | Розділ 2     |
| 4     | Розробка архітектури програми.                                       | 2 тиждень                                 | Розділ 3     |
| 5     | Розробка програми.                                                   | 3-4 тижні                                 | Розділ 3, 4  |
| 6     | Тестування та експериментальне дослідження програмного забезпечення. | 5 тиждень                                 | Розділ 4     |
| 7     | Оформлення пояснювальної записки та документів до неї.               | 6 тиждень                                 | Додатки      |
| 8     | Нормоконтроль та рецензування.                                       | 7 тиждень                                 |              |
| 9     | Захист роботи.                                                       | 8 тиждень                                 |              |

Студент(ка)

\_\_\_\_\_  
( підпис )      Максим ГРИЩЕНКОВ  
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

\_\_\_\_\_  
( підпис )      Тетяна ФЕДОРОНЧАК  
(Ім'я ПРІЗВИЩЕ)

## РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 122 с., 14 табл., 58 рис., 3 дод., 7 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, IOS, ANDROID, FLUTTER, DART, ОНЛАЙН МАРКЕТПЛЕЙС, FIREBASE.

Об'єкт дослідження – програмні засоби для онлайн маркетплейсу.

Предмет дослідження – мобільні застосунки для онлайн маркетплейсів.

Мета роботи – створення кросплатформного Android/iOS застосунку для онлайн маркетплейсу.

Матеріали, методи та технічні засоби: об'єктно орієнтоване програмування, мова програмування Dart, кросплатформний фреймворк Flutter, мобільний пристрій з операційною системою Android або iOS.

Результати. Створено кросплатформний Android/iOS застосунок для онлайн маркетплейсу.

Висновки. Розроблений програмний продукт дає можливість переглядати та створювати оголошення про продаж товарів, фільтрувати та шукати оголошення, створювати замовлення на покупку товару.

Галузь використання – користувачі смартфонів яким потрібно продавати або покупати товари різного призначення.

## **ABSTRACT**

Explanatory note for the bachelor's thesis: 122 pages, 14 tables, 58 figures, 3 appendices, 7 sources.

MOBILE APPLICATION, IOS, ANDROID, FLUTTER, DART, ONLINE MARKETPLACE, FIREBASE.

Research object – software tools for online marketplaces.

Research subject – mobile applications for online marketplaces.

Objective – to develop a cross-platform Android/iOS application for an online marketplace.

Materials, methods, and technical means: object-oriented programming, Dart programming language, cross-platform Flutter framework, mobile device with Android or iOS operating system.

Results. A cross-platform Android/iOS application for an online marketplace has been developed.

Conclusions. The developed software product allows users to view and create advertisements for the sale of goods, filter and search advertisements, and create orders to purchase goods.

Field of application – smartphone users who need to sell or buy goods of various purposes.

## ЗМІСТ

|                                                                                       |    |
|---------------------------------------------------------------------------------------|----|
|                                                                                       | С. |
| Перелік скорочень та умовних позначок .....                                           | 8  |
| Вступ.....                                                                            | 9  |
| 1 Аналіз предметної області .....                                                     | 10 |
| 1.1 Аналіз предметної області.....                                                    | 10 |
| 1.2 Приклади існуючих аналогів.....                                                   | 10 |
| 1.2.1 Amazon.....                                                                     | 10 |
| 1.2.2 OLX .....                                                                       | 12 |
| 1.2.3 AliExpress .....                                                                | 13 |
| 1.2.4 Порівняльна таблиця існуючих аналогів з розроблюваним програмним продуктом..... | 14 |
| 1.3 Висновки за першим розділом .....                                                 | 16 |
| 2 Розробка архітектури програми .....                                                 | 18 |
| 2.1 Функціональні вимоги до програми .....                                            | 18 |
| 2.2 Вибір програмних ресурсів.....                                                    | 19 |
| 2.3 Вибір СКБД.....                                                                   | 21 |
| 2.4 Вибір архітектури застосунку .....                                                | 23 |
| 2.5 Керування станом застосунку .....                                                 | 25 |
| 2.6 Висновки за другим розділом.....                                                  | 26 |
| 3 Основні рішення щодо реалізації компонентів системи .....                           | 27 |
| 3.1 Алгоритм функціонування програми .....                                            | 27 |
| 3.1.1 Алгоритм авторизації.....                                                       | 28 |
| 3.1.2 Алгоритм підбору рекомендованих оголошень .....                                 | 29 |
| 3.1.3 Алгоритм пошуку оголошень.....                                                  | 30 |
| 3.1.4 Алгоритм створення та редагування оголошень .....                               | 31 |
| 3.1.5 Алгоритм створення замовлення .....                                             | 32 |
| 3.1.6 Алгоритм керування замовленням.....                                             | 33 |
| 3.1.6 Алгоритм керування продажами.....                                               | 33 |
| 3.1.7 Алгоритм фільтрації оголошень .....                                             | 34 |

|                                                                           |     |
|---------------------------------------------------------------------------|-----|
| 3.1.8 Алгоритм надсилання повідомлення іншому користувачу .....           | 35  |
| 3.2 Опис бази даних.....                                                  | 36  |
| 3.3 Опис структури проєкту .....                                          | 42  |
| 3.3.1 Опис пакету data .....                                              | 43  |
| 3.3.2 Опис пакету dependencies .....                                      | 44  |
| 3.3.3 Опис пакету domain.....                                             | 45  |
| 3.3.4 Опис пакету presentation .....                                      | 46  |
| 3.4 Висновки за 3 розділом.....                                           | 48  |
| 4 Експлуатація, тестування та експериментальне дослідження програми ..... | 49  |
| 4.1 Призначення застосунку та умови його виконання .....                  | 49  |
| 4.2 Виконання програми .....                                              | 49  |
| 4.3 Тестування роботи додатку .....                                       | 64  |
| 4.4 Висновки за 4 розділом.....                                           | 66  |
| Висновки.....                                                             | 67  |
| Перелік джерел посилань.....                                              | 68  |
| Додаток А Технічне завдання.....                                          | 69  |
| Додаток Б Текст програми.....                                             | 73  |
| Додаток В Слайди презентації .....                                        | 113 |

## **ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК**

- UI – User Interface;
- URL – Uniform Resource Locator;
- SQL – Structured Query Language;
- БД – база даних.
- СКБД – система керування базами даних;

## ВСТУП

Сучасний розвиток інформаційних технологій суттєво змінив спосіб, яким ми взаємодіємо зі світом навколо нас. Одним з ключових аспектів цієї трансформації є широке використання мобільних пристроїв та застосунків для різних потреб. Онлайн маркетплейси стали важливою складовою цифрової екосистеми, надаючи можливість як покупцям, так і продавцям здійснювати торгівлю товарами в онлайн-середовищі.

Мета даної роботи полягає в розробці мобільного додатка для онлайн маркетплейса з метою полегшення процесу покупки та продажу товарів. Дослідження спрямоване на створення ефективного і зручного інструменту, який дозволить користувачам отримувати доступ до широкого асортименту товарів у будь-який час та з будь-якого місця.

У роботі будуть використані сучасні методи програмування та технології розробки мобільних додатків, зокрема мова програмування Dart та фреймворк Flutter. Основними функціональними можливостями розробленого додатка буде можливість перегляду товарів та оформлення замовлень на їх покупку, розміщення оголошень про їх продаж, а також функції фільтрації та пошуку.

Результати даного дослідження мають значний практичний потенціал у сфері електронної комерції та сприятимуть подальшому розвитку онлайн торгівлі.

## **1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

### **1.1 Аналіз предметної області**

Онлайн маркетплейс – складна багаторівнева платформа для реалізації товарів, яка об'єднує безліч продавців та надає їм інфраструктуру для ефективної онлайн-торгівлі [1].

Онлайн торгівля є дуже важливою частиною майже всіх сьогочасних бізнесів, а мобільні онлайн додатки та сайти це платформи для її здійснення. Вони дозволяють кожному виставляти на продаж свої товари чи послуги а також здійснювати покупки в інших продавців.

Переваги онлайн маркетплейсів очевидні для обох сторін угоди. Для покупців, це надає можливість здійснювати покупки в зручний для них час, вибирати з великого асортименту товарів та порівнювати ціни та характеристики. Для продавців, це відкриває доступ до глобального ринку та дозволяє легко розгортати та керувати своїми бізнесами.

Тому у світі, де електронна комерція стає все більш важливою складовою споживчого досвіду, онлайн маркетплейси залишаються ключовими гравцями, відкриваючи нові можливості для бізнесу та споживачів у всьому світі. Завдяки їм, торгівля переходить на новий рівень, де віртуальні майданчики стають основними місцями обміну товарами та послугами.

### **1.2 Приклади існуючих аналогів**

#### **1.2.1 Amazon**

За даними сайту [statista.com](https://www.statista.com) найбільш відвідуваним онлайн маркетплейсом являється [amazon.com](https://www.amazon.com) з 4.8 мільярдами переглядів в місяць [2]. Звичайно він має свій мобільний застосунок на платформах iOS та Android. Інтерфейс мобільного застосунку «Amazon» зображено на рис. 1.1 [3].

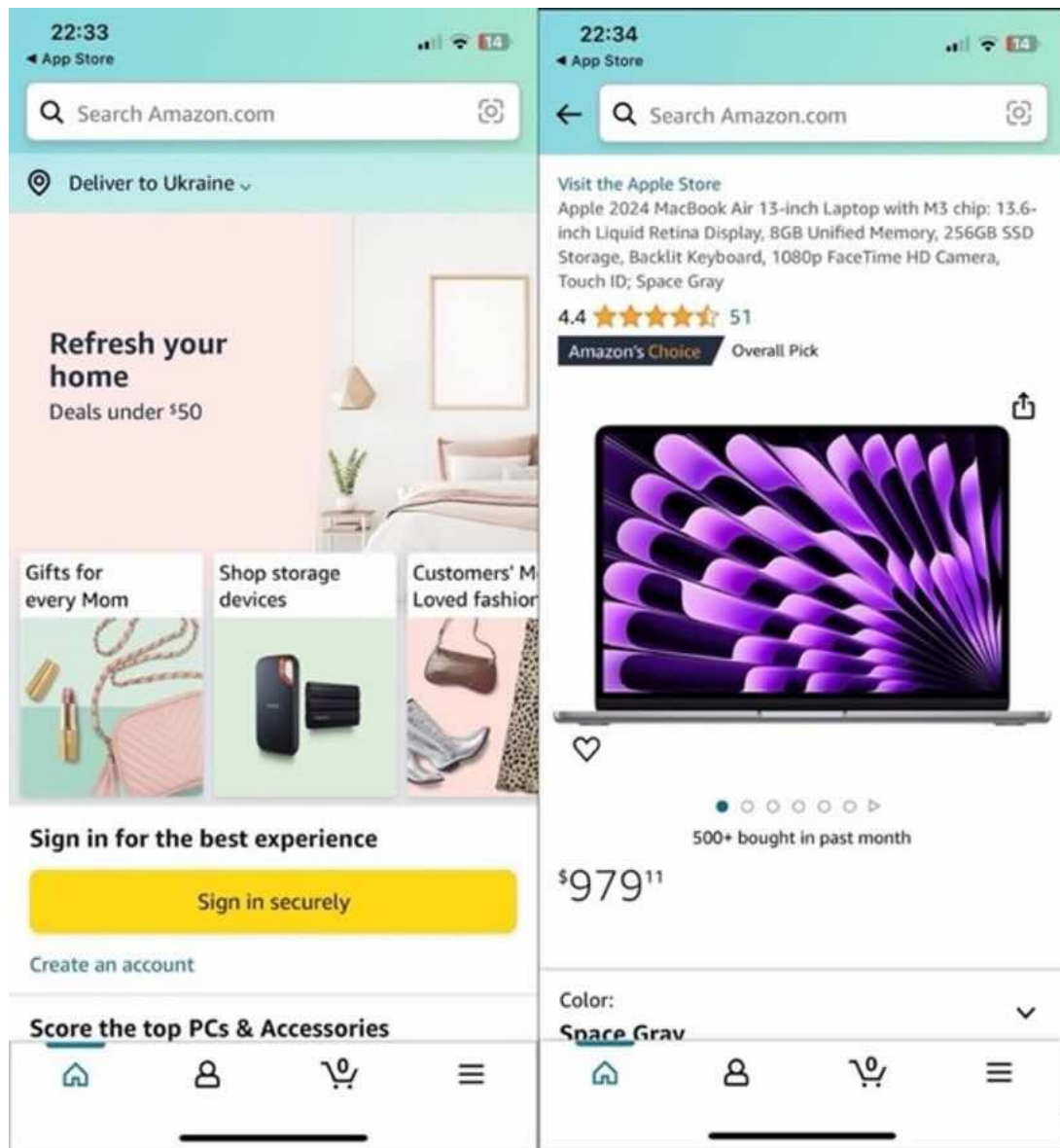


Рисунок 1.1 – Мобільний застосунок Amazon [3]

Основні переваги сервісу:

- гігантський вибір товарів;
- гарно підібрані персональні рекомендації;
- велика кількість категорій товарів та фільтрів;
- всі оголошення ретельно перевірені на якість.

Основні недоліки:

- дуже перенасичений інтерфейс;
- потрібно інший аккаунт для продажу товарів;
- немає української локалізації;

- немає вбудованого чату з продавцем;
- не можна додати товар в улюблені на попередньому перегляді.

### 1.2.2 OLX

Дуже популярна в Україні платформа онлайн оголошень, яка об'єднує людей для покупки, продажу або обміну товарами та послугами [4]. Має свій застосунок на iOS та Android. Інтерфейс мобільного застосунку «OLX» зображено на рис. 1.2 [5].

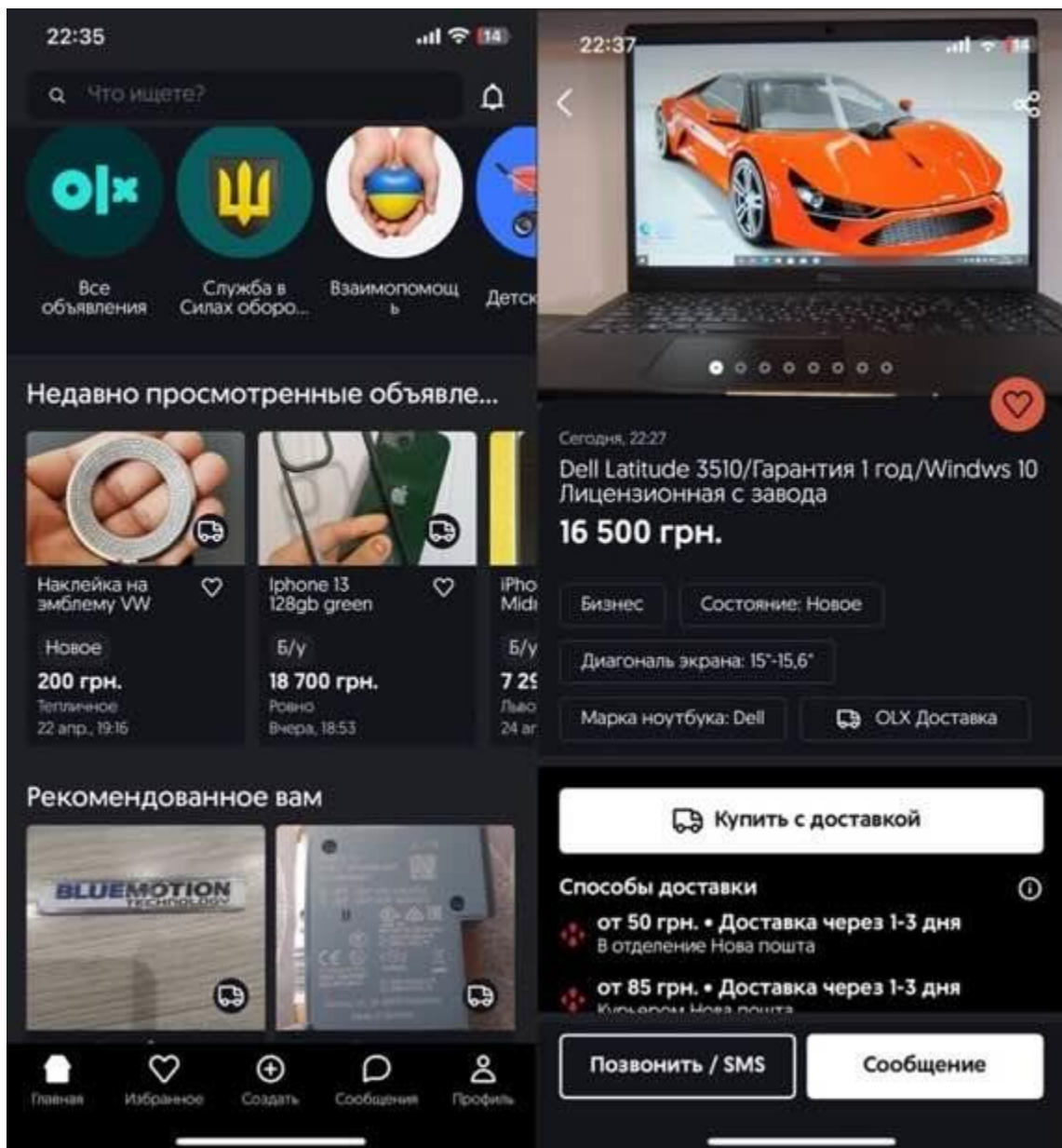


Рисунок 1.2 – Мобільний застосунок OLX [5]

Основні переваги сервісу:

- непоганий вибір товарів;
- містка карточка попереднього перегляду товару;
- вбудований чат;
- є варіант обміну товару;
- один аккаунт для покупки та продажу товарів.

Основні недоліки:

- якість оголошення переважно залежить від користувача;
- замало обов'язкових фільтрів до деяких категорій товарів;
- немає кошика товарів;
- немає можливості переглянути відгуки;
- немає пошуку по фото.

### 1.2.3 AliExpress

AliExpress – це інтернет-магазин роздрібної торгівлі малого бізнесу Китаю, що пропонує товари міжнародним покупцям в Інтернеті [6]. Це ще один досить популярний в Україні маркетплейс для купівлі товару, приклад зображено на рис. 1.3 [7].

Основні переваги сервісу:

- відносно широкий вибір товарів;
- вбудований чат;
- гарно підібрані рекомендації та можливість керувати ними;
- наявність категорій товарів.

Основні недоліки:

- окремий аккаунт для продавця;
- перенасичений інтерфейс екрану товару;
- замало обов'язкових фільтрів до деяких категорій товарів.

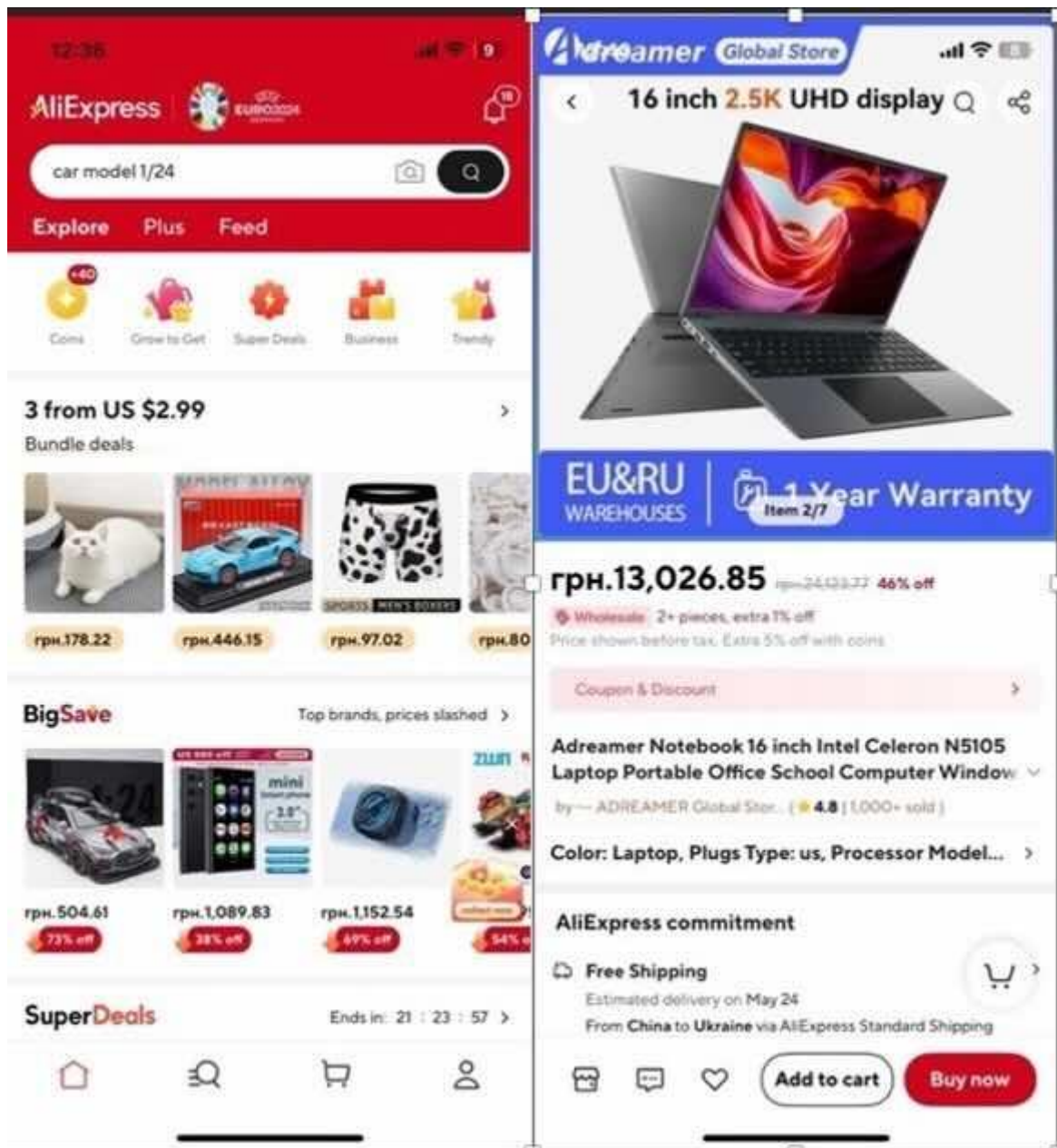


Рисунок 1.3 – Мобільний застосунок Aliexpress [7]

#### 1.2.4 Порівняльна таблиця існуючих аналогів з розроблюваним програмним продуктом

Уважно проаналізувавши існуючі аналоги, всі вони дають можливість робити покупки товарів та послуг, мають категорії, фільтрацію товарів, пошук.

Але OLX значно відрізняється від інших даючи унікальний корисний функціонал. У ньому можна продавати та купувати товари з єдиного аккаунта та варіант поставити договірну ціну чи обмін товарами, також він має

достатньо інтуїтивний та зручний інтерфейс. З нього було б зручно перейняти можливість мати єдиний аккаунт та зручність інтерфейсу.

Запланований для розробки застосунок буде враховувати наступні деталі:

- простий та зрозумілий інтерфейс;
- єдиний аккаунт продавця та покупця;
- фільтрацію товарів;
- сортування товарів;
- пошук товарів;
- категорії товарів;
- інформація про продавця та його рейтинг;
- кошик товарів;
- категорія нещодавно переглянутих товарів;
- категорія улюблених товарів;
- рекомендації товарів.

В табл. 1.1. проведено порівняння функціональних можливостей розглянутих програмних продуктів та програми «Sellers», яка буде створена в ході виконання роботи.

Таблиця 1.1 –Порівняльна характеристика застосунків

| Критерій                                      | Amazon | OLX | AliExpress | Sellers |
|-----------------------------------------------|--------|-----|------------|---------|
| 1                                             | 2      | 3   | 4          | 5       |
| Кросплатформність (iOS/Android)               | +      | +   | +          | +       |
| Єдиний аккаунт для продажу та покупок товарів | -      | +   | -          | +       |
| Сортування товарів                            | -      | +   | +          | +       |
| Фільтрація товарів                            | +      | +   | +          | +       |

Продовження таблиці 1.1

| 1                                                             | 2 | 3 | 4 | 5 |
|---------------------------------------------------------------|---|---|---|---|
| Інтуїтивний простий інтерфейс                                 | - | + | - | + |
| Кошик товарів                                                 | + | - | + | + |
| Категорії товарів                                             | + | + | + | + |
| Категорія улюблених товарів                                   | + | + | + | + |
| Категорії недавно відвіданих товарів                          | + | + | + | + |
| Персональні рекомендації товарів                              | + | + | + | + |
| Вбудований чат                                                | - | + | + | + |
| Інформація про продавця та його рейтинг                       | + | + | + | + |
| Посилання на зв'язок через месенджер замість вбудованого чату | - | - | - | + |

### 1.3 Висновки за першим розділом

У першому розділі було розглянуто область застосування, аналізуючи існуючі аналоги програм, такі як "Amazon", "OLX", "AliExpress".

Загалом, існуючі аналоги мають добре продумані системи створення, пошуку та обміну товарів. Також вони пропонують систему рекомендацій, розділення товарів на категорії та їх фільтрацію.

Під час аналізу цих застосунків було виявлено деякі недоліки, такі як перенасичений інтерфейс, що може призвести до плутанини серед

користувачів, а також відсутність можливості створення та покупки товарів в одному обліковому записі.

Порівнявши аналоги з застосунком "Sellers", було зроблено висновок, що розробка цього мобільного застосунку є актуальною та доцільною. Це обумовлено рядом функцій, які передбачає "Sellers", у відмінну від існуючих аналогів.

## 2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

### 2.1 Функціональні вимоги до програми

Розроблюваний мобільний застосунок має наступні функціональні вимоги:

- реєстрація та авторизація через електронну адресу, також можливість авторизації через смс;
- створення оголошення про товар який повинен мати наступні параметри: фотографія товару, назва, опис, теги, місце знаходження, ціна, категорія товару, фільтри для обраної категорії;
- зберігання оголошень у віддаленій БД;
- можливість видаляти та редагувати створені оголошення;
- можливість перегляду оголошення інших продавців, додавати їх в улюблене та до кошику або видаляти звідти;
- можливість перегляду інформації про продавців: рейтинг, інші товари на продаж, контактну інформацію;
- можливість сортування оголошень за популярністю, ціною, датою;
- перегляд оголошень за категоріями та фільтрування їх;
- перегляд нещодавно відвіданих оголошень;
- перегляд рекомендованих товарів;
- можливість переглядати кошику;
- можливість створювати замовлення на покупку товару вказавши кількість товару та адресу доставки;
- можливість приймати або відхиляти замовлення покупців;
- мати вбудований чат та можливість переписуватись з продавцями в чат.

На основі цих функціональних вимог було створено діаграму прецедентів нижче (рис. 2.1).

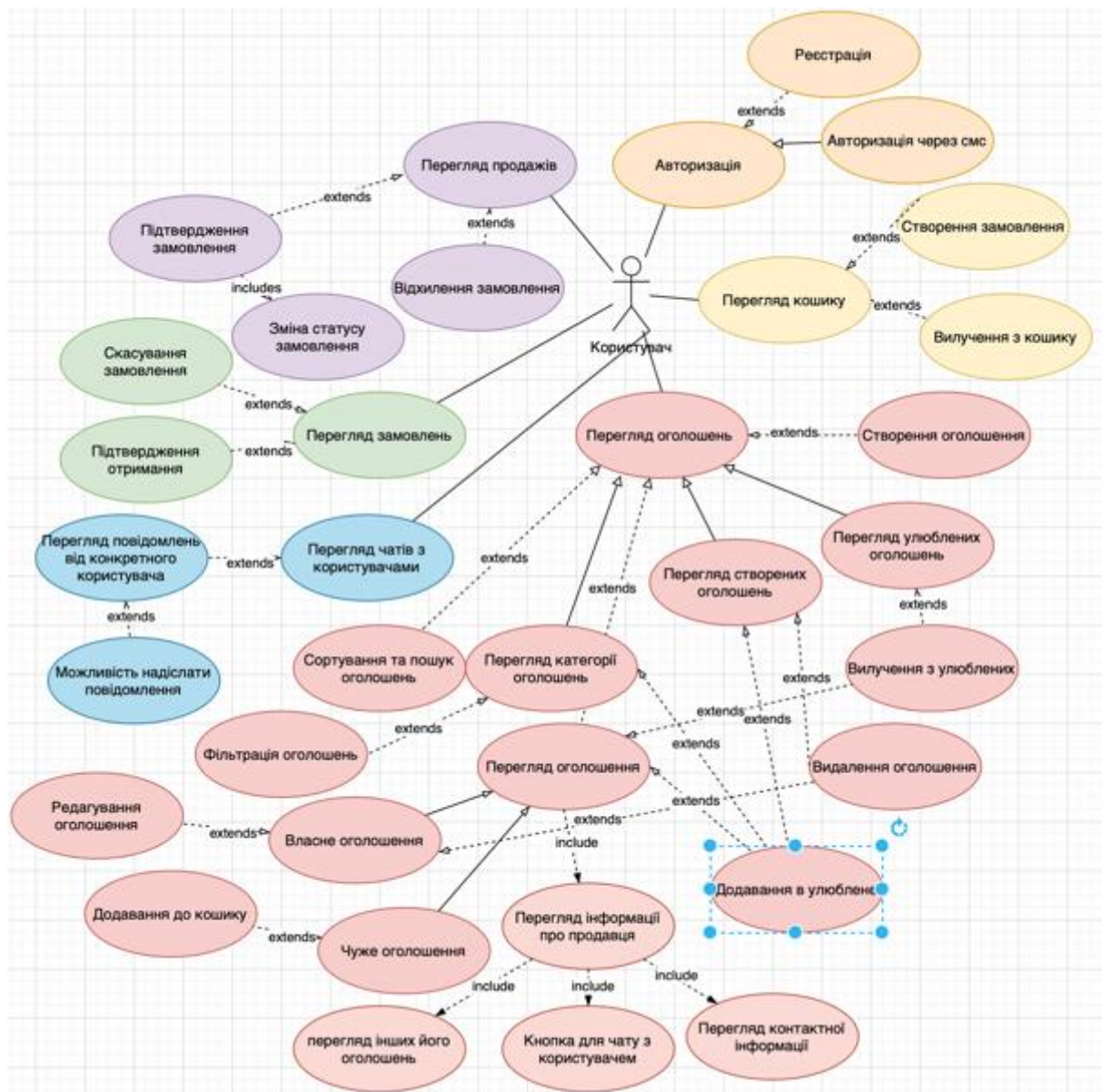


Рисунок 2.1 – Діаграма прецедентів

## 2.2 Вибір програмних ресурсів

Для виконання вказаного завдання було проведено аналіз існуючих фреймворків (див. табл. 2.1) та обрано універсальний фреймворк Flutter.

Обраний фреймворк є оптимальним вибором для створення мобільних застосунків як для платформ Android, так і для iOS, а також для веб та десктопу з операційних систем Windows або MacOS.

Також Flutter підтримує технології Hot Restart та Hot Reload, що дозволяють мінімізувати час на оновлення додатку згідно внесених змін у код.

Таблиця 2.1 – Порівняння фреймворків

| Критерій                               | Flutter      | React Native | Xamarin                  |
|----------------------------------------|--------------|--------------|--------------------------|
| Підтримка Android та iOS платформ      | +            | +            | +                        |
| Швидкість роботи додатку               | Найвища      | Висока       | Непогана                 |
| Вартість                               | Безкоштовний | Безкоштовний | Підписка                 |
| Велика спільнота та підтримка оновлень | +            | +            | +                        |
| Споживання ресурсів                    | Мінімальне   | Ефективне    | Залежить від оптимізації |

Найбільш важливими для кросплатформного додатку є відображення користувацького інтерфейсу однаковим на будь-якому пристрої та висока продуктивність цього додатку. Обидва цих критерії притаманні обраному фреймворку.

Основною мовою програмування для Flutter є Dart, яка і була обрана як основна для написання з наступних причин:

- зручне використання;
- кросплатформність;
- всі інструменти Flutter взаємодіють лише з мовою Dart як і сторонні бібліотеки для нього;
- підтримка асинхронності та багатопоточності;
- автоматичний збір сміття;
- Підтримка двох типів компіляції Ahead Of Time(AOT) для випускної програми та Just In Time (JIT) для розробки.

В якості середовища розробки було обрано Android Studio опираючись на результати порівняльної табл. 2.2.

Таблиця 2.2 – Порівняння середовищ розробки

| Критерій                                       | Android Studio | Visual Studio Code | IntelliJ Idea |
|------------------------------------------------|----------------|--------------------|---------------|
| Підтримка системи контролю версій              | +              | +                  | +             |
| Підтримка автоматизованого виправлення помилок | +              | -                  | +             |
| Безкоштовна                                    | +              | +                  | -             |
| Вбудований Android емулятор                    | +              | -                  | -             |
| Підтримка сніппетів                            | +              | +                  | +             |
| Вбудований відлагоджувач                       | +              | +                  | -             |
| Ресурсоемкість                                 | Висока         | Мінімальна         | Середня       |
| Оновлення версій                               | +              | +                  | +             |

Після аналізу середовищ розробки лідером виявилася Android Studio, оскільки вона надає інструменти, спеціально призначені для мобільної розробки.

## 2.3 Вибір СКБД

Для онлайн маркетплейсу просто необхідно зберігати дані віддалено. Для цього потрібно створити власний сервер або налаштувати хмарну СКБД.

Вирішено було взяти вже готовий варіант хмарної СКБД, тож треба було ще обрати тип БД (Реляційні та нереляційні).

Реляційні бази даних (RDBMS) використовують модель даних, що базується на таблицях, де кожен рядок представляє запис, а кожний стовпець – атрибут.

Нереляційні бази даних (NoSQL) використовують різноманітні моделі даних такі як колонки, документи, ключ-значення або графи.

Порівняльна характеристика реляційної та нереляційної баз даних наведена в табл. 2.3.

Таблиця 2.3 – Порівняння реляційної та нереляційної баз даних

| Критерії           | Реляційні бази даних                              | Нереляційні бази даних                                                |
|--------------------|---------------------------------------------------|-----------------------------------------------------------------------|
| Модель даних       | Таблична                                          | Різнорізнотна (документи, ключ-значення, колонки, графи)              |
| Мова запитів       | SQL                                               | Відрізняється залежно від конкретної бази даних                       |
| Зв'язки між даними | За допомогою ключів                               | Відсутні або мають інший механізм (наприклад, посилання на документи) |
| Схема даних        | Суворі, строго визначені                          | Гнучкі, може змінюватися з часом без перестановки бази даних          |
| Масштабованість    | Менш гнучкі, складніше масштабувати горизонтально | Більш гнучкі, краще масштабуються горизонтально                       |
| Швидкість доступу  | Зазвичай менша, особливо на великих обсягах даних | Зазвичай швидша, особливо для операцій з ключем                       |
| Вартість           | Зазвичай дорожче, особливо для комерційних систем | Зазвичай дешевші, багато відкритих джерел                             |
| Приклади           | MySQL, PostgreSQL, Oracle                         | MongoDB, Firebase, Redis                                              |

Було віддано перевагу сервісу NoSQL бази даних Firebase від компанії Google з наступних причин:

- гнучкість;

- масштабованість;
- безкоштовність.

Firebase у свою чергу має два типи сховищ даних – це Firestore та Realtime Database.

У табл. 2.4 зображена порівняльна характеристика цих двох сервісів.

Таблиця 2.4 – Порівняння баз даних Firestore та Realtime Database

| Критерії                     | Firestore                                                          | Realtime Database                              |
|------------------------------|--------------------------------------------------------------------|------------------------------------------------|
| Модель даних                 | Документоорієнтована (колекції та документи)                       | На основі дерева JSON                          |
| Структура даних              | Гнучка структура документів, підтримка вкладених об'єктів          | Одна велика JSON-структура, доступ через шляхи |
| Синхронізація                | Підтримка режиму офлайн, синхронізація в реальному часі            | Синхронізація в реальному часі                 |
| Функціональність             | Розширені можливості фільтрації та запитів, підтримка прав доступу | Простота використання, підтримка режиму офлайн |
| Підтримка мобільних додатків | +                                                                  | +                                              |

За результатами порівняльної таблиці маємо що Realtime Database більше підходить для задач де потрібна миттєва синхронізація та постійне відстеження статусу клієнта, тобто вона буде використана для вбудованого чату додатку. Firestore в свою чергу має більш розширені можливості фільтрації та запитів і має більш гнучку структуру документів з підтримкою вкладених об'єктів, тобто вона буде використана для роботи з оголошеннями та замовленнями.

## 2.4 Вибір архітектури застосунку

Для даного застосунку було обрано методологію Clean Architecture яка передбачає розділення додатку на декілька шарів:

- presentation (відповідає за відображення об'єктів на екрані користувача);
- domain (відповідає за бізнес-логіку додатку);
- data (відповідає за роботу з даними додатку).

Розбиття на шари значно спрощує масштабованість та зрозумілість написаного коду. Також додає зручності при тестуванні додатку та пошуку помилок.

Ще одним важливим аспектом даної методології є взаємодія між цими шарами відповідальності, вона має бути завжди лінійна і без пропусків шарів. Приклад зображено на рис. 2.2. Приклад показує що при взаємодії користувача з презентаційним рівнем(натиснення кнопки, закриття екрану, свайпі тощо) створюється подія яка переходить до рівню бізнес-логіки, далі створюється запит до рівня даних який в свою чергу створює запит на самі дані в БД. Потім БД дає відповідь яка оброблюється рівнем даних, переходить до рівня бізнес-логіки яка в свою чергу створює потік з оновленими даними відправлений до презентаційного рівня який відображає оновлені дані на екрані користувача.

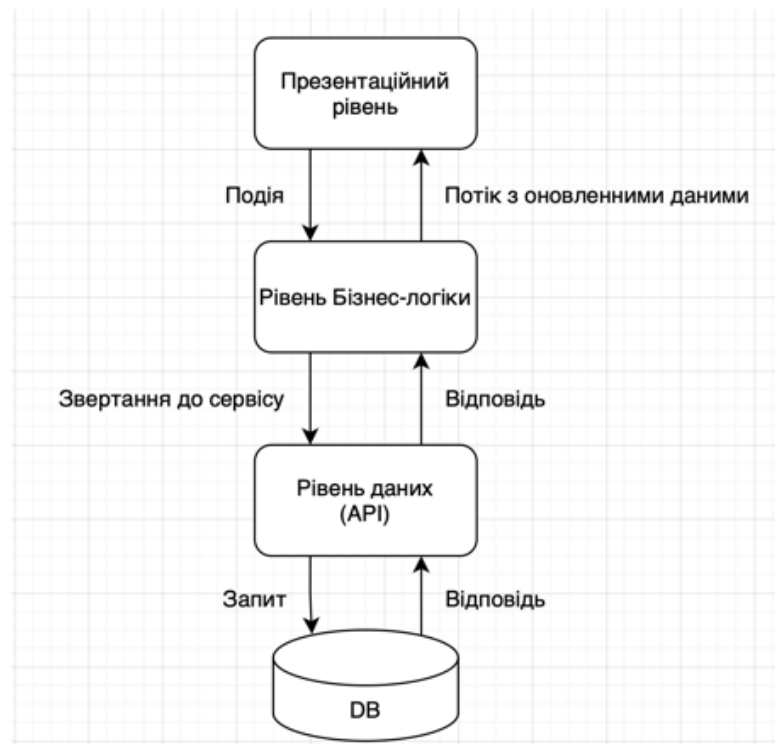


Рисунок 2.2 – Схема роботи шарів Clean Architecture

## 2.5 Керування станом застосунку

Спосіб керування станом потрібні для розділення шару бізнес-логіки від презентаційного шару додатку. Тому до вибору об'єкту керування станом потрібно відноситись дуже серйозно.

У Flutter є вбудовані рішення стейт-менеджменту, наприклад `StateElement` у `StatefulWidget` та `InheritedWidget`. Але `StatefulWidget` може впливати лише на віджети нижче в ієрархії, тому цей варіант підходить лише для змін конкретного віджета і не може бути використаний як універсальний варіант. `InheritedWidget` в свою чергу не має такої вади але не є достатньо низькорівневим рішенням яке займає багато зайвого коду.

Також є велика кількість сторонніх пакетів призначених для стейт-менеджменту, найбільш популярні це `Redux`, `Cubit` та `Provider`.

`Cubit` та `Provider` використовують принципи реактивного програмування коли `Redux` додержується функціональної парадигми.

Порівняння цих пакетів наведено у табл. 2.5.

Таблиця 2.5 – Порівняння стейт-менеджерів

| Критерії                                  | Cubit  | Redux    | Provider | GetX   |
|-------------------------------------------|--------|----------|----------|--------|
| Простота використання                     | Висока | Середня  | Непогана | Висока |
| Розділення бізнес-логіки та представлення | Висока | Середня  | Немає    | Висока |
| Швидкодія                                 | Висока | Непогана | Непогана | Висока |
| Масштабованість                           | Висока | Середня  | Непогана | Висока |
| Можливість робити локальні оновлення      | +      | -        | -        | +      |

Зробивши аналіз порівняльної табл. 2.5 було обрано Cubit з наступних причин:

- простота та зручність використання;
- можливість робити локальні оновлення;
- масштабованість.

## **2.6 Висновки за другим розділом**

У другому розділі було виконано опис програми з визначенням функціональних вимог, фреймворку, середовища розробки, бази даних, архітектури та стейт менеджера.

Ретельно проаналізувавши можливі варіанти фреймворку для мобільних кросплатформних додатків було обрано Flutter через значні переваги у швидкодії та зручності у використанні.

Для використання Flutter було обрано IDE Android Studio як найліпшу під мобільну розробку через її вбудований емулятор та необхідні інструменти.

Базу даних було обрано безкоштовні, швидкі та надійні Firebase Firestore для зберігання оголошень та Realtime Database для чату між користувачами.

Архітектурою застосунку буде Clean Architecture через найліпшу масштабованість та зручність тестування що є дуже важливим для достатньо великого проєкту як «Sellers».

Стейт-менеджером застосунку став Cubit який має гнучку структуру та простий інтерфейс.

## 3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

### 3.1 Алгоритм функціонування програми

На рис. 3.1 зображено схему функціонування розробленого застосунку.

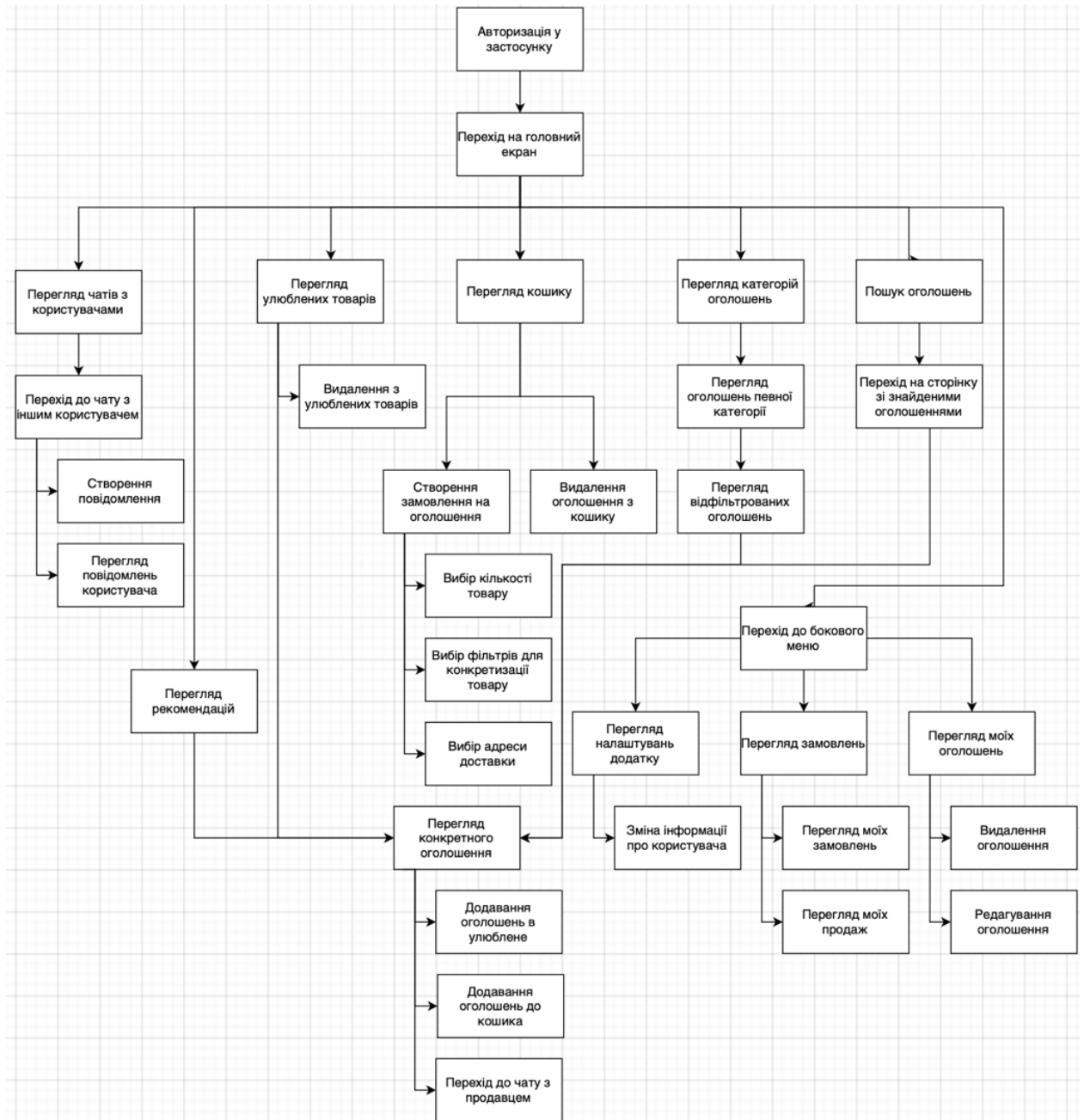


Рисунок 3.1 – Схема функціональних можливостей розробленого застосунку

Проаналізувавши схему вище можна зазначити наступні розроблені алгоритми:

- авторизації;
- підбору рекомендованих оголошень;
- пошуку оголошень;
- створення та редагування оголошень;
- створення замовлення;
- керування замовленням;
- керування продажами;
- фільтрація оголошень;
- надсилання повідомлення іншому користувачу.

### **3.1.1 Алгоритм авторизації**

Нижче наведені кроки для алгоритму авторизації

Крок 1. Користувач вводить дані та обирає створити чи увійти в обліковий запис(перехід до кроку 5)

Крок 2. Дані введені користувачем передаються в подію onSignUp та перевіряється валідність даних, при невалідних значеннях перехід на крок 5.

Крок 3. Дані передаються в подію onRegister де виконується запит до Firebase для створення користувача.

Крок 4. Якщо запит повертає токен, то крок 7, якщо є то крок 9.

Крок 5. Якщо користувач не пам'ятає пароль та має зв'язаний номер телефону з аккаунтом то перехід до наступного кроку, інакше до кроку 8.

Крок 6. Користувач вводить номер телефону та отримує повідомлення з кодом, інакше перехід до кроку 11

Крок 7. Користувач вводить код та здійснюється запит на авторизацію, якщо запит повертає токен то перехід до кроку 12, інакше до кроку 11.

Крок 8. Дані внесені користувачем передаються в подію onSignIn та перевіряється валідність даних, при невалідних значеннях перехід на крок 9.

Крок 9. Дані передаються в подію onLogin де виконується запит Firebase для авторизації користувача.

Крок 10. Якщо запит повертає токен то перехід на крок 12.

Крок 11. Повідомлення про помилку та перехід до кроку 13

Крок 12. Перехід на головний екран.

Крок 13. Кінець.

### **3.1.2 Алгоритм підбору рекомендованих оголошень**

Для отримання рекомендованих користувачу оголошень методу `getRecommendedProducts` потрібен список тегів. Нижче наведені кроки для алгоритму підбору тегів для пошуку рекомендованих товарів.

Крок 1. Додати перші три теги кожного нещодавно відвіданого товару до списку тегів `recentWatchedTags`.

Крок 2. Викликається подія `computeRecommendations` до якої передаються дані `recentWatchedTags`.

Крок 3. Створюється словник повторів з ключем типу `int` який відображає кількість повторень тегів та значенням кортеж рядків який відображає самі теги.

Крок 4. Для кожного переданого тегу виконується перевірка кількості повторень, та він додається у словник повторів до списку тегів з цією кількістю повторів.

Крок 5. Далі створюється список `sortedKeys` з відсортованих по зростанню ключів словника повторів.

Крок 6. Створюється список `significantKeys` з першими трьома значеннями.

Крок 7. Створюється список зі списків тегів з словника повторів для кожного значення `significantKey`.

Крок 8. Створений список зі списків об'єднується в один та записується у змінну `recentRecommendationList`.

Крок 9. Кінець.

Після цього при виклику методу `getRecommendedProducts` в нього передається список тегів, та метод вертає товари в яких співпадає хоч один з тегів.

### 3.1.3 Алгоритм пошуку оголошень

Нижче представлені кроки для алгоритму пошуку оголошень.

Крок 1. Запит який вводить користувач передається у подію `onSubmitted`.

Крок 2. У події `onSubmitted` для введеного рядка викликається метод `tokenize`.

Крок 3. У події `tokenize` рядок розбивається на слова, та вертає список слів у змінну `searchTokens`.

Крок 4. Список передається в подію `fetchNextItems` зі значенням `refreshList` встановленим в `true`.

Крок 5. У події `fetchNextItems` викликається метод `getProducts`.

Крок 7. В події `getProducts` ініціалізується змінна `searchedProducts` яка має тип списку об'єкту `Product`.

Крок 6. В `getProducts` викликається подія `getSearchedProducts` яка приймає список токенів.

Крок 7. `getSearchProducts` вертає тип `QueryDocumentSnapshot` з документами де співпадає хоч один з токенів у назві оголошень у змінну `snap`.

Крок 8. Подія `snapToProduct` вертає перетворені документи на об'єкти `Product` в змінну `products`.

Крок 9. Кожен токен зі змінної `searchTokens` перевіряється на наявність в кожному доданому оголошенні в `products` змінні, та при ненааявності видаляється продукт.

Крок 10. До змінної `searchedProducts` додаються всі елементи списку `products`.

Крок 11. Якщо кількість елементів `searchedProducts` менша за кількість потрібних то перехід до кроку 6.

Крок 12. `getProducts` вертає `searchedProducts` разом з іншими потрібними даними та `ProductCubit` оновлює оголошення на екрані.

### **3.1.4 Алгоритм створення та редагування оголошень**

Нижче представлені кроки для алгоритму створення та редагування оголошень.

Крок 1. Створюється `ProductFormState` в який передається переданий об'єкт типу `Product` або новостворений об'єкт типу `Product`.

Крок 2. Створюється об'єкт `BLoC( Business Logic Object Controller)` `ProductFormCubit productCubit` та передається туди `ActionMode` та `ProductFormState`.

Крок 3. Вводяться необхідні дані: назва, опис, ціна, фото, місцезнаходження.

Крок 4. При натиснутій кнопці `Next` викликається подія `verifyData`.

Крок 5. При введенні дані валідні, то перехід до кроку 7, якщо ні то до кроку 6.

Крок 6. З'являється повідомлення про помилку та перехід до кроку 3.

Крок 7. Перехід до екрану `AddFiltersScreen` та передається тоді `productCubit`.

Крок 8. Створюється об'єкт `filterCubit` типу `FiltersFormCubit`.

Крок 9. Обираються всі необхідні фільтри.

Крок 10. При підтвердженні введених даних у `productCubit` викликається подія `addProduct` куди передається список фільтрів.

Крок 11. Параметр `filters` об'єкту `product` замінюється на переданий список фільтрів.

Крок 11. Якщо `mode` дорівнює `update` то перехід до кроку 14.

Крок 12. Викликається подія `addProduct` об'єкту `ProductUseCase` який створює нове оголошення.

Крок 13. Якщо оголошення не створене то показується повідомлення про помилку та перехід до кроку 16.

Крок 14. Викликається подія `editProduct` об'єкту `ProductUseCase` який редагує старе оголошення.

Крок 15. Якщо оголошення не було редаговане, то показується повідомлення про помилку та перехід до кроку 16.

Крок 16. Кінець алгоритму.

### **3.1.5 Алгоритм створення замовлення**

Нижче представлені кроки для алгоритму створення замовлення.

Крок 1. Вводяться необхідні фільтри, кількість товару та адресу доставки.

Крок 2. Користувач натискає кнопку `make order` в який передаються введені дані.

Крок 3. Якщо введені дані невалідні то перехід до кроку 9.

Крок 4. Створюється з отриманих даних об'єкт типу `ProductOrder`.

Крок 5. Викликається подія `newOrder` об'єкту класу `OrderUseCase` для створення нового замовлення.

Крок 6. Якщо замовлення не створене то перехід на крок 9.

Крок 7. Виклик події `toggleToCart` об'єкту класу `AccountUseCase` для вилучення товару з кошику.

Крок 8. Якщо товар не вилучено то перехід до кроку 9, інакше перехід до кроку 10.

Крок 9. Показується повідомлення про помилку.

Крок 10. Кінець алгоритму.

### 3.1.6 Алгоритм керування замовленням

Нижче наведені кроки для алгоритму керування замовленням.

Крок 1. Якщо замовлення має статус `accepted` то перехід до кроку 9, якщо `in the way` то перехід до кроку 4.

Крок 2. Якщо користувач натискає кнопку `cancel` то перехід до кроку 3, інакше до кроку 5.

Крок 3. Викликається подія `deleteOrder` класу `OrderUseCase` яка видаляє замовлення, якщо замовлення не видалено то перехід до кроку 8, до кроку 9.

Крок 4. Якщо користувач натискає кнопку `confirm` то перехід до кроку 5, інакше до кроку 9.

Крок 5. Викликається подія `deleteOrder` класу `OrderUseCase` яка видаляє замовлення, якщо замовлення не видалено то перехід до кроку 8.

Крок 6. Створюється діалог для оцінки продавця, при відхиленні від оцінки перехід до кроку .

Крок 7. Викликається подія `updateSeller` класу `OrderUseCase` яка оновлює рейтинг продавця додаючи одиницю до вказаної зірочки, якщо рейтинг оновлено то перехід до кроку 9.

Крок 8. Повідомлення про помилку.

Крок 9. Кінець алгоритму.

### 3.1.6 Алгоритм керування продажами

Нижче наведені кроки для алгоритму керування продажами.

Крок 1. Якщо замовлення має статус `waiting for accept` то перехід до кроку 2, якщо `accepted`, то до кроку 3.

Крок 2. Якщо користувач натискає кнопку `асерт` то перехід до кроку 3, якщо `decline` то до наступного кроку.

Крок 3. Викликається подія `updateOrderStatus` куди передається об'єкт `declined` типу `OrderStatus` яка змінює статус замовлення на переданий.

Крок 4. Якщо статус не оновлено то перехід до кроку `_`, інакше перехід до кроку 10.

Крок 5. Викликається подія `updateOrderStatus` куди передається об'єкт `accepted` типу `OrderStatus` яка змінює статус замовлення на переданий.

Крок 6. Якщо статус не оновлено то перехід до кроку 10.

Крок 7. Якщо користувач натискає кнопку `on the way` то перехід на крок вперед, інакше на крок 11.

Крок 8. Викликається подія `updateOrderStatus` куди передається об'єкт `onTheWay` типу `OrderStatus` яка змінює статус замовлення на переданий.

Крок 9. Якщо статус не змінено то перехід до кроку 10, інакше до кроку 11.

Крок 10. Показується повідомлення про помилку.

Крок 11. Кінець алгоритму.

### 3.1.7 Алгоритм фільтрації оголошень

Створений Cubit об'єкт `CategoryScreenCubit` має `state` який має всі додані фільтри представлені типом `List<FilterValue>` `addedFilters`, та `List<String>` `brands`. Нижче приведені кроки для алгоритму фільтрації оголошень.

Крок 1. Користувач натискає на фільтр де викликається подія `toggleFilter` в яку передається об'єкт `filter` типу `FilterValue`.

Крок 2. Якщо параметр `brand` типу `filter` не дорівнює `null` та дорівнює `value`, то перехід до кроку 7.

Крок 3. Якщо фільтр наявний в `addedFilters` то перехід до наступного кроку, інакше до кроку 5.

Крок 4. Фільтр видаляється з `addedFilters`, перехід до кроку 6.

Крок 5. Фільтр додається до списку `addedFilters`.

Крок 6. Об'єкт state оновлюються новим списком addedFilters, перехід до кроку 13.

Крок 7. Викликається подія toggleBrandFilter в яку передається filter.

Крок 8. Якщо даний параметр value об'єкту filter наявний в brands, перехід до наступного кроку, інакше до кроку 10.

Крок 9. Значення filter.value додається до списку brands, перехід до кроку 11.

Крок 10. Значення filter.value видаляється зі списку brands.

Крок 11. Викликається подія prepareBrandFilters яка оновлює список доступних фільтрів певного бренду.

Крок 12. Оновлюється об'єкт state новим списком brands.

Крок 13. Кінець алгоритму.

### **3.1.8 Алгоритм надсилання повідомлення іншому користувачу**

Нижче наведені кроки для алгоритму надсилання повідомлення іншому користувачу.

Крок 1. Якщо користувач обирає медіа повідомлення то перехід до кроку 3, якщо текстове то до наступного кроку.

Крок 2. Викликається подія onCreateMessageClick класу-кубіту PrivateChatCubit де створюється об'єкт message типу ChatMessage.

Крок 3. Викликається подія onAddMediaClick класу-кубіту PrivateChatCubit де створюється об'єкт медіа повідомлення message типу ChatMessage яке має тип ChatMessageType який переданий в аргументах.

Крок 4. Викликається подія createMessage класу ChatUseCase до якого передається створений об'єкт message.

Крок 5. Якщо ChatMessageType має тип video або audio то перехід до кроку \_, якщо тип text то до наступного.

Крок 6. Викликається подія saveFile яка збережує медіа файл у Firebase Storage та вертає url який присвоюється параметру text об'єкту message.

Крок 7. Викликається подія `sendMessage` класу `NotificationServices` яка передає до іншого користувача через `Firebase Cloud Messaging API (Application Programming Interface)` повідомлення про новостворене повідомлення.

Крок 8. Подія вертає об'єкт `message` зі створеним `id`.

Крок 9. Оновлюються об'єкт `state` з доданим до `List<ChatMessage>` `messages` об'єктом `message`.

Крок 10. Кінець алгоритму.

### 3.2 Опис бази даних

У розділі 2.3 було проналазовано бази даних `Firebase` та було обрано `Firestore` для збереження даних де потрібна фільтрація, та `Realtime Database` де потрібне оновлення в реальному часі. Обидва ці сховища відносяться до `NoSQL` систем управління базами даних, що означає що вони мають більш гнучку структуру та спосіб взаємодії з ними.

Щоб почати роботу з `Firebase` потрібно спочатку створити проєкт до якого додати цільові платформи, у даному випадку `Android` та `iOS`.

Було додано реєстрацію та авторизацію за електронною поштою та паролем, та можливість авторизуватися через смс повідомлення на вказаний номер телефону. Всі додані способи авторизації в застосунку наведені на рис. 3.2.

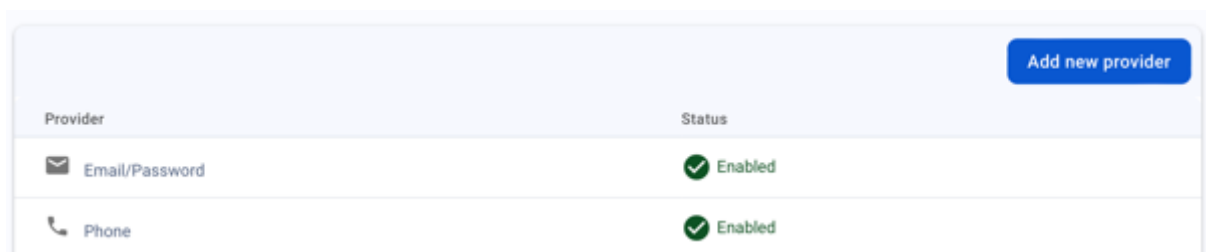


Рисунок 3.2 – Способи авторизації у додатку

Для взаємодії застосунку з базами даних використовуються бібліотеки «`Cloud Firestore`», «`Firebase Database`» та «`Firebase Auth`», де перші дві

надаютьс доступ до даних у Firestore Database та Realtime Database відповідно, а третя потрібна для авторизації користувача після якої надається доступ до баз даних. Нижче на рис. 3.3 представлена схема БД зі зв'язками між сутностями. Важливо зазначити що зв'язки побудовані на рівні застосунку для більш гнучкого контролю та меншої прив'язки до конкретної БД.

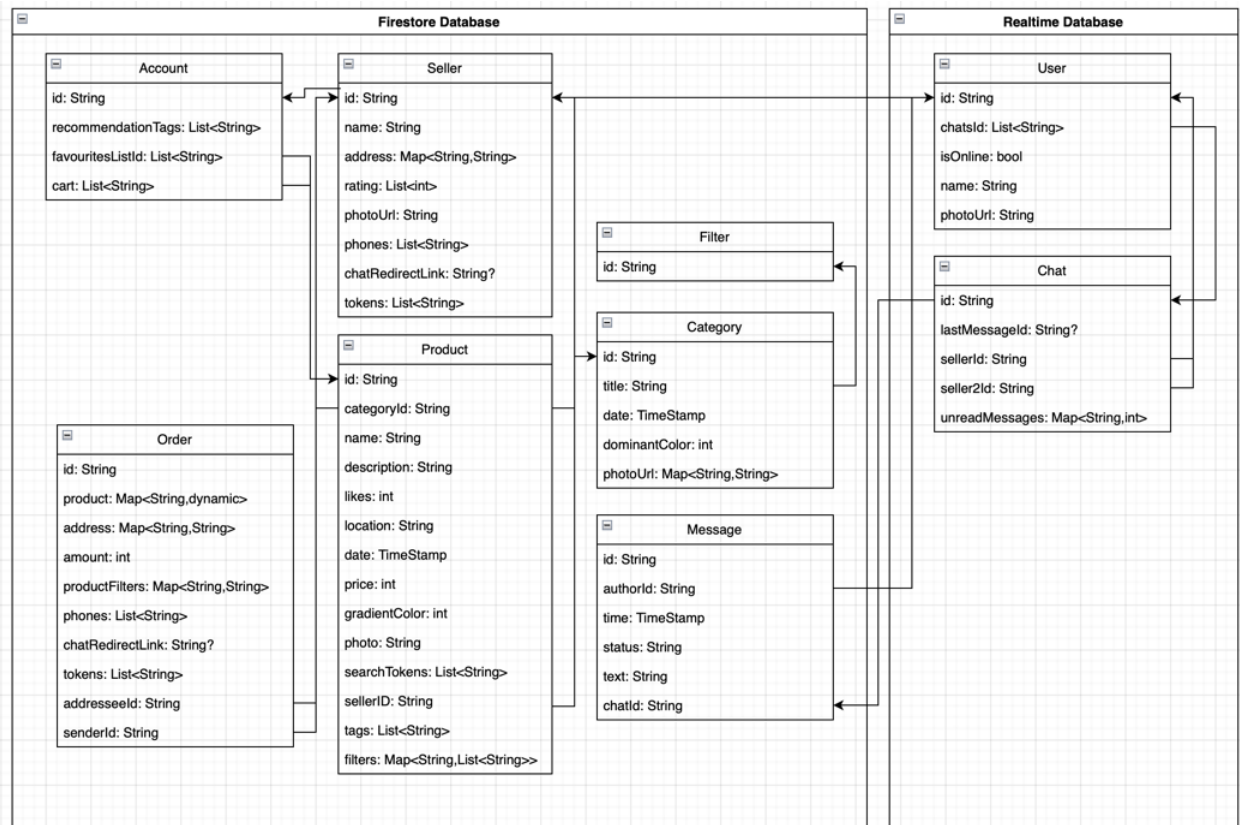


Рисунок 3.3 – Схема БД зі зв'язками між сутностями

Бази даних мають наступні колекції:

- products – зберігає оголошення створені користувачами;
- categories – зберігає категорії товарів;
- orders – зберігає всі замовлення;
- sellers – зберігає всю інформацію про продавців;
- accounts – зберігає аккаунти всіх користувачів;
- users – зберігає інформацію про користувачів для чатування;
- messages – зберігає всі повідомлення;

- filters – зберігає всі фільтри категорій;
- chats – зберігає всі чати користувачів.

Сутність product має наступні поля: id, categoryId, name, description, likes, location, date, price, gradientColor, photo, searchTokens, sellerID, tags, filters. Поля, їх тип та призначення цієї сутності наведено на табл 3.1.

Таблиця 3.1 – Сутність «Product»

| Назва         | Тип                       | Призначення                              |
|---------------|---------------------------|------------------------------------------|
| id            | String                    | Унікальний ідентифікатор товару          |
| categoryId    | String                    | Ідентифікатор категорії                  |
| name          | String                    | Назва товару                             |
| description   | String                    | Опис товару                              |
| likes         | int                       | Кількість вподобань                      |
| location      | String                    | Локація товару                           |
| date          | TimeStamp                 | Дата створення товару                    |
| price         | int                       | Ціна товару                              |
| gradientColor | int                       | Колір для градієнту на екрані оголошення |
| photo         | String                    | URL до фото товару                       |
| searchTokens  | List<String>              | Токени для пошуку                        |
| sellerID      | String                    | Ідентифікатор продавця                   |
| tags          | List<String>              | Теги товару                              |
| filters       | Map<String, List<String>> | Фільтри товару                           |

Сутність Seller має наступні поля: id, name, address, rating, photoUrl, phones, chatRedirectLink, tokens. Поля, їх тип та призначення цієї сутності наведено на табл 3.2.

Таблиця 3.2 – Сутність «Seller»

| Назва            | Тип                 | Призначення                                         |
|------------------|---------------------|-----------------------------------------------------|
| id               | String              | Унікальний ідентифікатор продавця                   |
| name             | String              | Ім'я продавця                                       |
| address          | Map<String, String> | Адреса продавця                                     |
| rating           | List<int>           | Рейтинг продавця                                    |
| photoUrl         | String              | URL до фото продавця                                |
| phones           | List<String>        | Контактні телефони продавця                         |
| chatRedirectLink | String?             | Посилання на альтернативний чат замість вбудованого |
| tokens           | List<String>        | Токени для сповіщень                                |

Сутність Order має наступні поля: id, product, address, amount, productFilters, phones, chatRedirectLink, tokens, addressId, senderId. Поля, їх тип та призначення цієї сутності наведено на табл 3.3.

Таблиця 3.3 – Сутність «Order»

| Назва            | Тип                  | Призначення                         |
|------------------|----------------------|-------------------------------------|
| 1                | 2                    | 3                                   |
| id               | String               | Унікальний ідентифікатор замовлення |
| product          | Map<String, dynamic> | Інформація про товар                |
| address          | Map<String, String>  | Адреса доставки                     |
| amount           | int                  | Кількість товарів                   |
| productFilters   | Map<String, String>  | Фільтри товару                      |
| phones           | List<String>         | Контактні телефони                  |
| chatRedirectLink | String?              | Посилання на чат                    |

Продовження таблиці 3.3

| 1         | 2            | 3                         |
|-----------|--------------|---------------------------|
| tokens    | List<String> | Токени для сповіщень      |
| addressId | String       | Ідентифікатор адреси      |
| senderId  | String       | Ідентифікатор відправника |

Сутність Filter має наступні поля: id. Також сутність Filter може мати динамічні поля різних типів різної кількості, тому відобразити їх неможливо. Гарантовані поля, їх тип та призначення цієї сутності наведено на табл 3.4.

Таблиця 3.4 – Сутність «Filter»

| Назва | Тип    | Призначення                      |
|-------|--------|----------------------------------|
| id    | String | Унікальний ідентифікатор фільтра |

Сутність Category має наступні поля: id, title, date, dominantColor, photoUrl. Поля, їх тип та призначення цієї сутності наведено на табл 3.5.

Таблиця 3.5 – Сутність «Category»

| Назва         | Тип                 | Призначення                                      |
|---------------|---------------------|--------------------------------------------------|
| id            | String              | Унікальний ідентифікатор категорії               |
| title         | String              | Назва категорії                                  |
| date          | TimeStamp           | Дата створення категорії                         |
| dominantColor | int                 | Колір для кольорового акценту сторінки категорії |
| photoUrl      | Map<String, String> | URL до фото категорії                            |

Сутність Message має наступні поля: id, authorId, time, status, text, chatId. Поля, їх тип та призначення цієї сутності наведено на табл 3.6.

Таблиця 3.6 – Сутність «Message»

| Назва    | Тип       | Призначення                           |
|----------|-----------|---------------------------------------|
| id       | String    | Унікальний ідентифікатор повідомлення |
| authorId | String    | Ідентифікатор автора повідомлення     |
| time     | TimeStamp | Час відправлення                      |
| status   | String    | Статус повідомлення                   |
| text     | String    | Текст повідомлення                    |
| chatId   | String    | Ідентифікатор чату                    |

Сутність User має наступні поля: id, chatsId, isOnline, name, photoUrl. Поля, їх тип та призначення цієї сутності наведено на табл 3.7.

Таблиця 3.7 – Сутність «User»

| Назва    | Тип          | Призначення                          |
|----------|--------------|--------------------------------------|
| id       | String       | Унікальний ідентифікатор користувача |
| chatsId  | List<String> | Список ідентифікаторів чатів         |
| isOnline | bool         | Статус онлайн                        |
| name     | String       | Ім'я користувача                     |
| photoUrl | String       | URL до фото користувача              |

Сутність Chat має наступні поля: id, lastMessageId, sellerId, seller2Id, unreadMessages. Поля, їх тип та призначення цієї сутності наведено на табл 3.8.

Таблиця 3.8 – Сутність «Chat»

| Назва         | Тип     | Призначення                           |
|---------------|---------|---------------------------------------|
| 1             | 2       | 3                                     |
| id            | String  | Унікальний ідентифікатор чату         |
| lastMessageId | String? | Ідентифікатор останнього повідомлення |

Продовження таблиці 3.8

| 1              | 2                | 3                                  |
|----------------|------------------|------------------------------------|
| sellerId       | String           | Ідентифікатор продавця             |
| seller2Id      | String           | Ідентифікатор другого продавця     |
| unreadMessages | Map<String, int> | Кількість непрочитаних повідомлень |

Для збереження зображень для сутностей Product, Message, Seller або інших медіа файлів використовується Firebase Storage який для доступу до збережених даних надає URL адресу. Тобто, всі сутності які використовують медіа файли зберігають в собі URL адресу до ресурсу у Firebase Storage. На рис. 3.4 зображення доданий до сховища медіа ресурс.

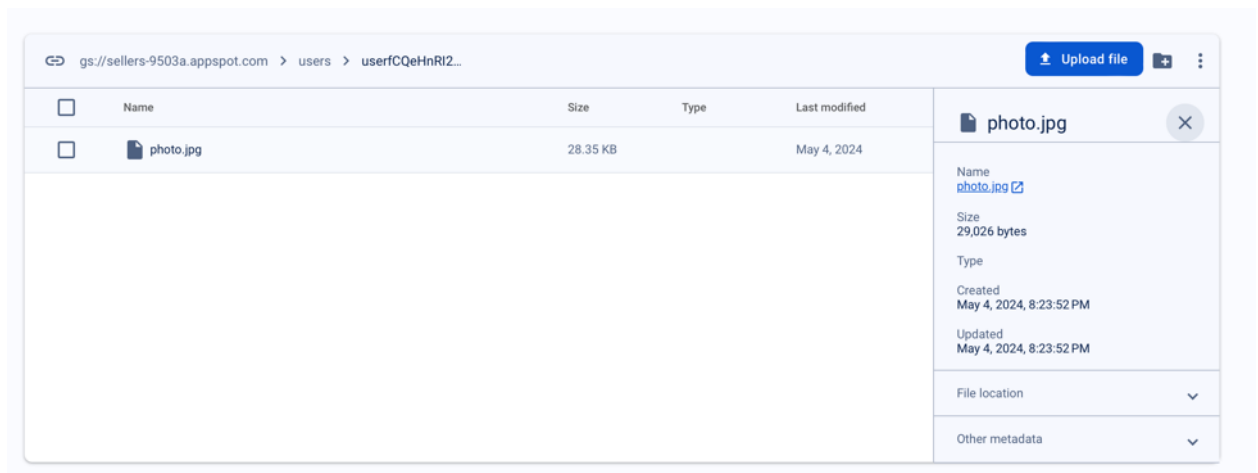


Рисунок 3.4 – Додане до сховища Firebase Storage зображення

Такий спосіб маніпулювання даних дуже зручний при повторному використанні ресурсів та дозволяє завантажувати дані з Firestore або Realtime Database паралельно з медіа для більш ефективного читання даних.

### 3.3 Опис структури проєкту

Проект був побудований на основі принципів архітектури Clean Architecture обраної у розділі 2.4. На рис 3.5 зображено структуру застосунку.

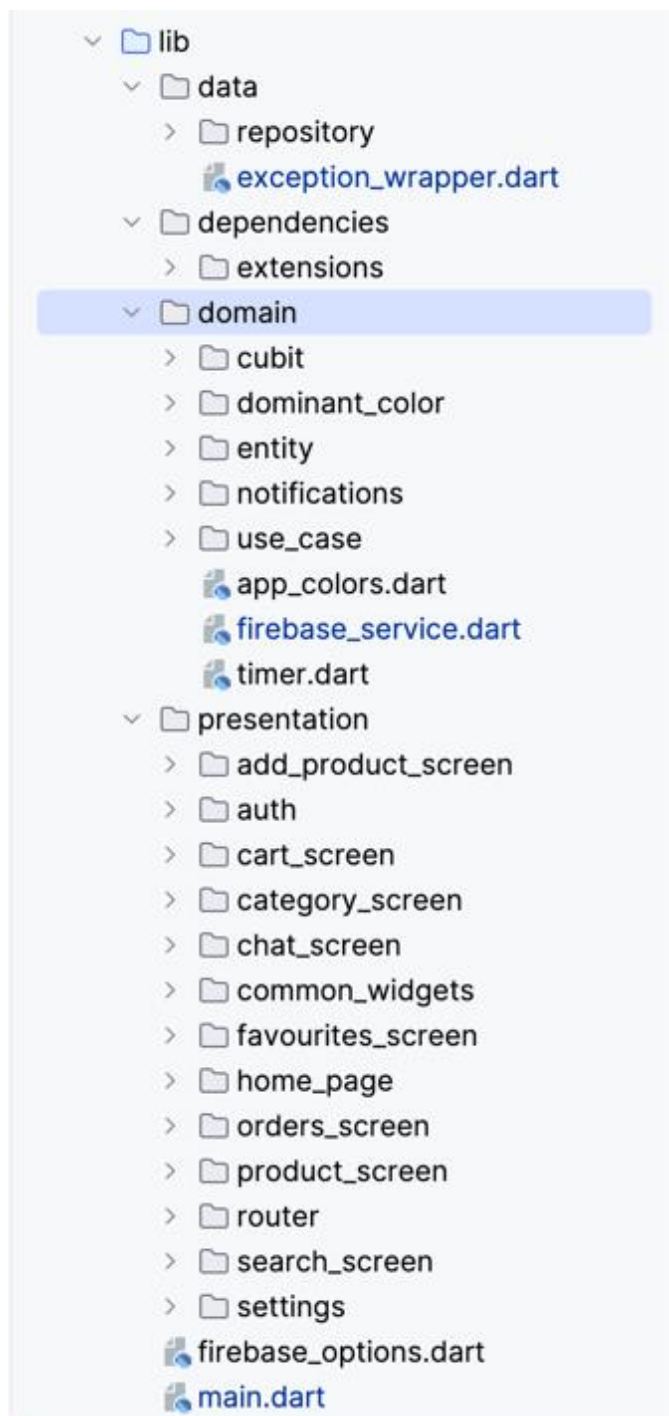


Рисунок 3.5 – Структура застосунку

### 3.3.1 Опис пакету data

В пакеті data (рис. 3.6) знаходяться класи відповідальні за роботу з даними.

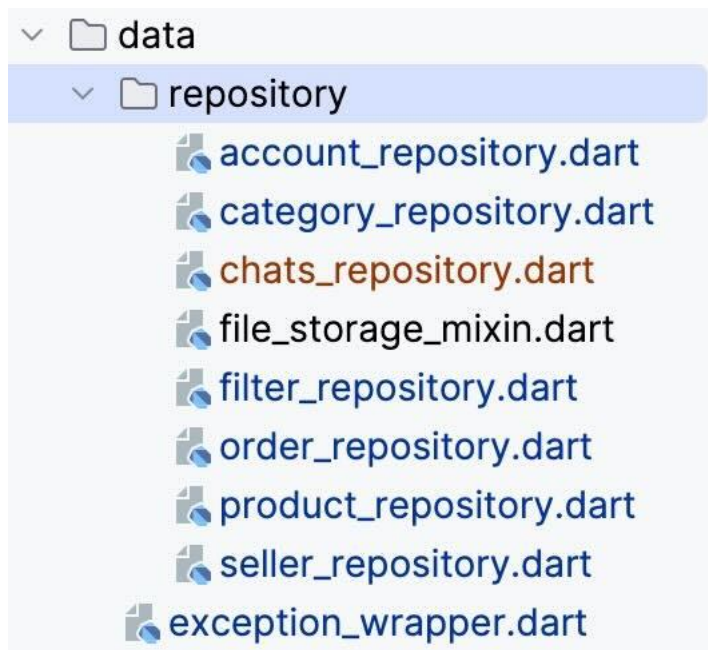


Рисунок 3.6 – Пакет data

У директорії repository зберігаються модулі які відповідають за зв'язок програми з базами даних, тобто створює запити до Firebase. Модуль exception\_wrapper.dart призначений для обробки помилок при запитах до Firebase.

### 3.3.2 Опис пакету dependencies

В пакеті dependencies (рис. 3.7) знаходяться допоміжні файли.

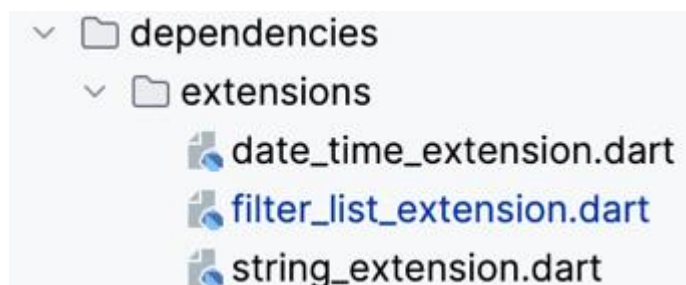


Рисунок 3.7 – Пакет dependencies

У директорії extensions зберігаються модулі які відповідають за розширення можливостей класів для зручності використання.

### 3.3.3 Опис пакету domain

В пакеті domain (рис. 3.8) знаходяться модулі які відповідають за бізнес-логіку програми.

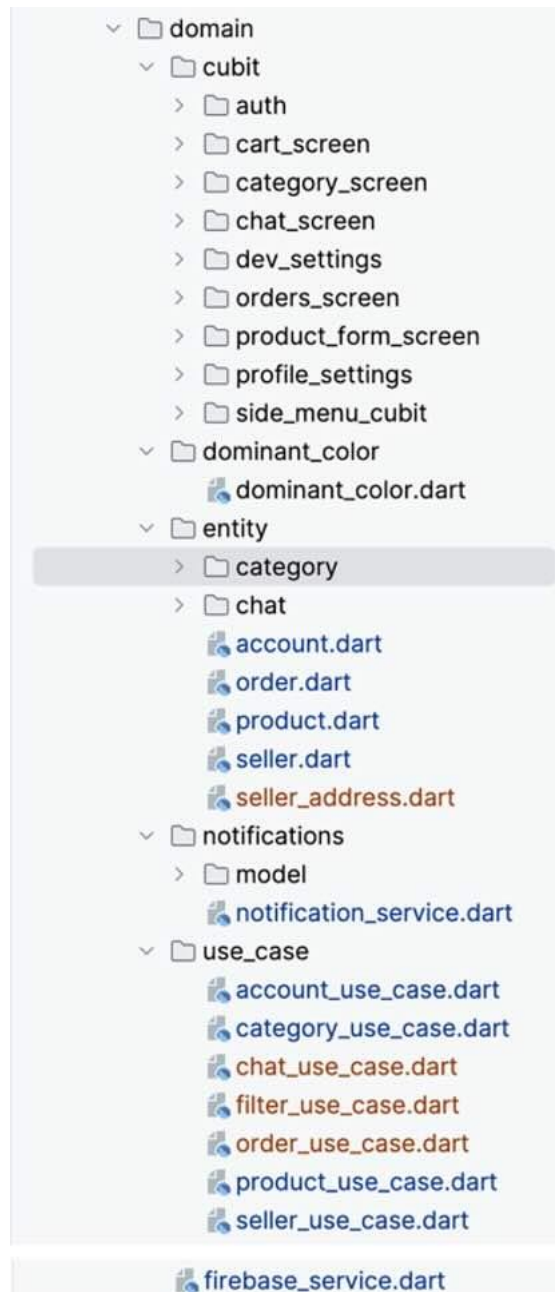


Рисунок 3.8 – Пакет domain

У директорії cubit зберігаються директорії в яких обов'язково є Cubit класи та класи їх стану. Вони використовуються у presentation модулі для обробки дій користувача та зв'язку з use\_case модулем.

У директорії `entity` знаходяться усі сутності з якими працює додаток. Вони зберігають дані отримані з баз даних Firebase у зручному форматі для використання в застосунку.

У директорії `dominant_color` знаходяться необхідні файли для визначення основного кольору градієнта для оголошення.

Модуль `notifications` має файли для роботи з push повідомленнями від Firebase.

Файл `firebase_service.dart` має методи для авторизації та реєстрації користувача.

У модулі `use_case` знаходяться усі класи які відтворюють сценарії використання, тобто вони є мостом між шарами `data` та `domain`.

### 3.3.4 Опис пакету `presentation`

В пакеті `presentation` (рис. 3.9) знаходяться файли для відображення користувацького інтерфейсу програми.

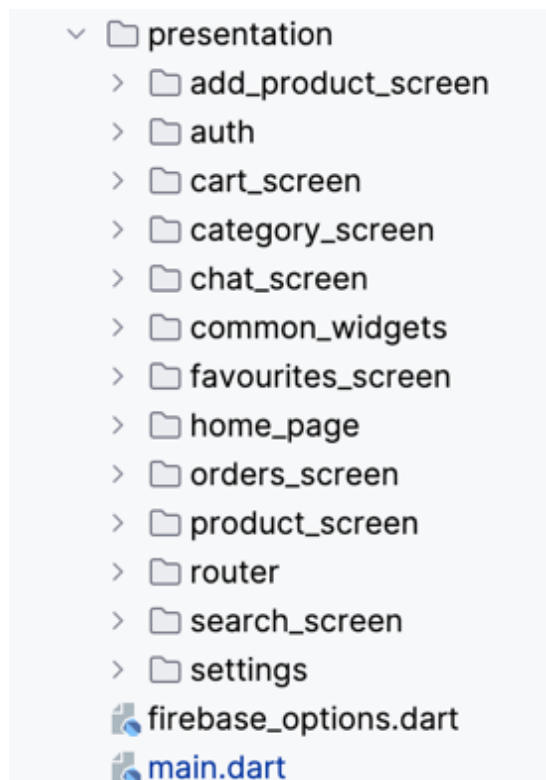


Рисунок 3.9 – Пакет `presentation`

Директорія `add_product_screen` містить в собі файли для відображення екрану створення чи редагування оголошення.

Директорія `auth` містить в собі файли для відображення екранів аутентифікації, таких як логін, реєстрація та відновлення паролю.

Директорія `cart_screen` містить в собі файли для відображення екрану кошика, де користувачі можуть переглядати, додавати або видаляти товари перед оформленням замовлення.

Директорія `category_screen` містить в собі файли для відображення екрану категорій, де користувачі можуть переглядати товари, згруповані за різними категоріями.

Директорія `chat_screen` містить в собі файли для відображення екрану чату, де користувачі можуть обмінюватися повідомленнями один з одним або з підтримкою.

Директорія `common_widgets` містить в собі загальні віджети, які використовуються на різних екранах додатка, такі як кнопки, поля введення та інші повторювані компоненти.

Директорія `favourites_screen` містить в собі файли для відображення екрану улюблених товарів, де користувачі можуть переглядати товари, які вони додали до списку обраного.

Директорія `home_page` містить в собі файли для відображення головної сторінки додатка, яка є центральним вузлом для навігації до інших екранів.

Директорія `orders_screen` містить в собі файли для відображення екрану замовлень та продажів, де користувачі можуть переглядати свої замовлення чи продажі та їх статус.

Директорія `product_screen` містить в собі файли для відображення екрану деталей товару, де користувачі можуть переглядати детальну інформацію про конкретний товар.

Директорія `router` містить в собі файли для налаштування маршрутизації в додатку, що відповідає за навігацію між різними екранами.

Директорія `search_screen` містить в собі файли для відображення екрану пошуку, де користувачі можуть шукати товари за ключовими словами або фільтрами.

Директорія `settings` містить в собі файли для відображення екрану налаштувань, де користувачі можуть змінювати параметри свого профілю та налаштування додатка.

Файл `firebase_options.dart` містить налаштування для інтеграції з Firebase, включаючи ключі API (Application Programming Interface) та інші конфігураційні параметри.

`main.dart` це головний файл додатка, який містить точку входу та ініціалізацію додатка, включаючи налаштування теми, маршрутизації та початковий екран.

### **3.4 Висновки за 3 розділом**

У даному розділі було розглянуто основні алгоритми функціонування програми, було розроблено схему функціональних можливостей програми та розібраний кожний алгоритм на кроки.

Зроблений розбір створених сутностей для БД та спроектовано схему зв'язків цих сутностей у сховищі.

Було проаналізовано структуру програми та розібраний кожний пакет додатку.

## **4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ**

### **4.1 Призначення застосунку та умови його виконання**

Ціллю створення даного застосунку була в тому, щоб у користувачів була можливість легко та зручно обмінюватись товарами у будь-який зручний час. Даний продукт дозволяє переглядати оголошення, створювати власні та шукати по ключовим словам, створювати замовлення та продавати власні товари, додавати товари в улюблені та кошик, фільтрувати оголошення за категоріями та фільтрами в них, чатуватись з іншими користувачами.

Для забезпечення нормального функціонування додатку потрібно мати такі мінімальні апаратні значення:

- мобільний телефон з системою iOS(версія 10+) чи Android (версія 5.0+);
- доступ до Інтернету.

### **4.2 Виконання програми**

Перед запуском програми потрібно встановити додаток на девайс. Для запуску програми на Android потрібно завантажити файл з розширенням .apk на девайс, для iOS додаток може бути лише завантажений через AppStore чи сертифіковані apple сторонні магазини Євросоюзу, для публікації в AppStore потрібно мати сертифікат розробника Apple, створити на офіційному порталі проєкт куди завантажити файл .ipa та документацію до нього. Для запуску програми потрібно натиснути на іконку з назвою «Sellers».

При першому відкритті застосунку у користувача запитується дозвіл на виправку повідомлень (рис. 4.1) а далі основним екраном неавторизованого користувача стає екран авторизації на рис. 4.2а, при натисканні на кнопку реєстрації з'являються додаткові поля реєстрації на рис. 4.2б. При помилці реєстрації чи авторизації виводиться повідомлення як на рис 4.3.

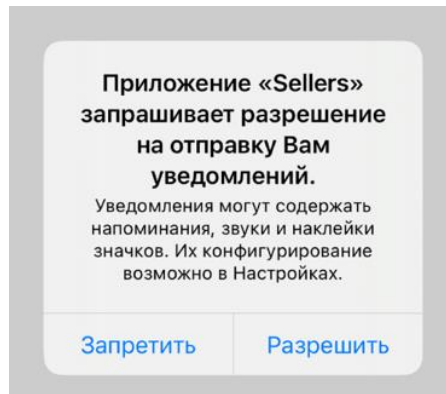
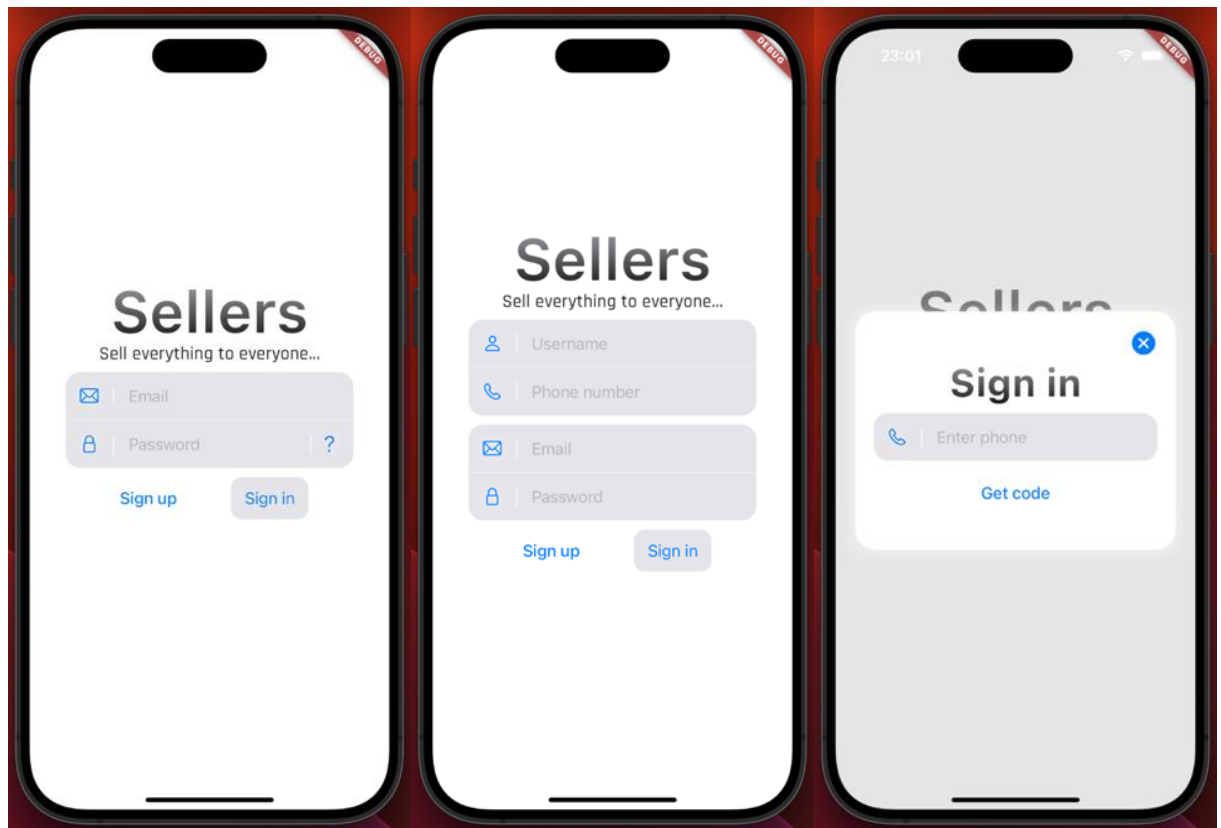


Рисунок 4.1 – Вікно запиту дозволу на повідомлення



*a*

*б*

*в*

а – сторінка авторизації; б – сторінка реєстрації; в – авторизація через смс.

Рисунок 4.2 – Екран реєстрації та авторизації користувача

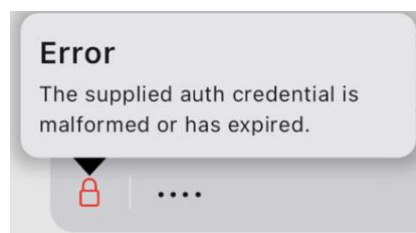
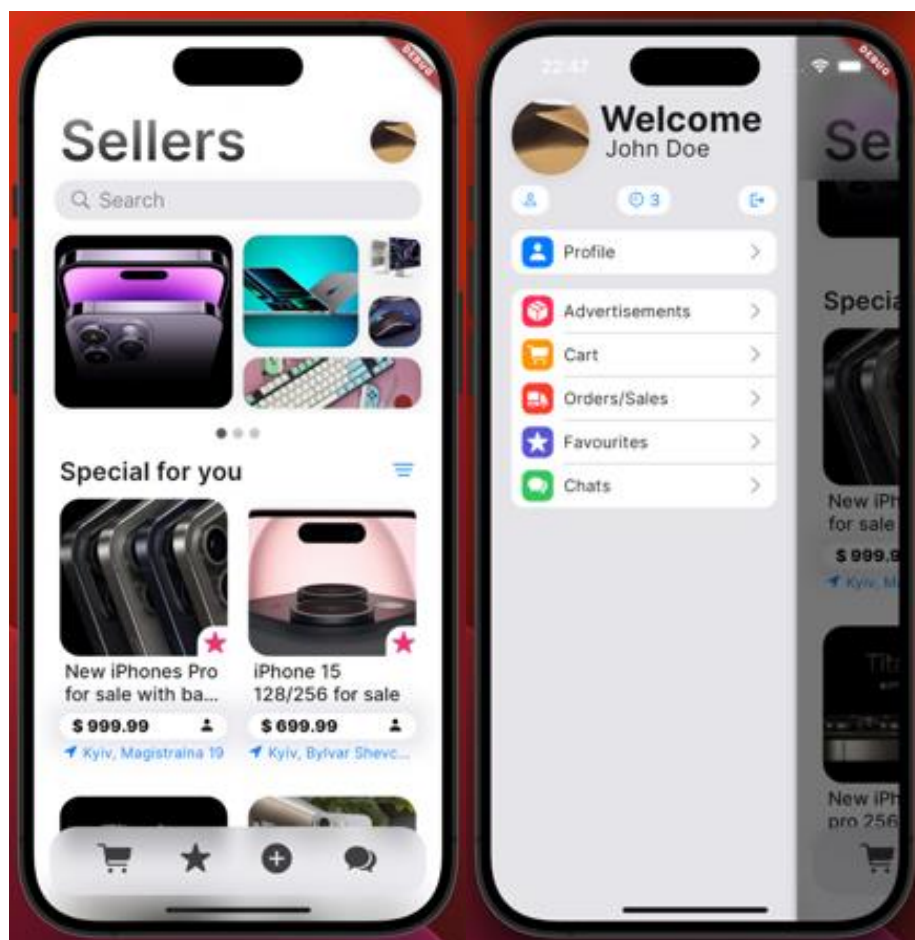


Рисунок 4.3 – Повідомлення про помилку вводу

Для переходу на екран авторизації через смс (рис. 4.2в) потрібно натиснути кнопку «?» зображену на рис. 4.2а.

Після того як користувач авторизувався він переходить на екран з головною сторінкою (рис 4.4а), також на ній присутнє бокове меню (рис 4.4б).

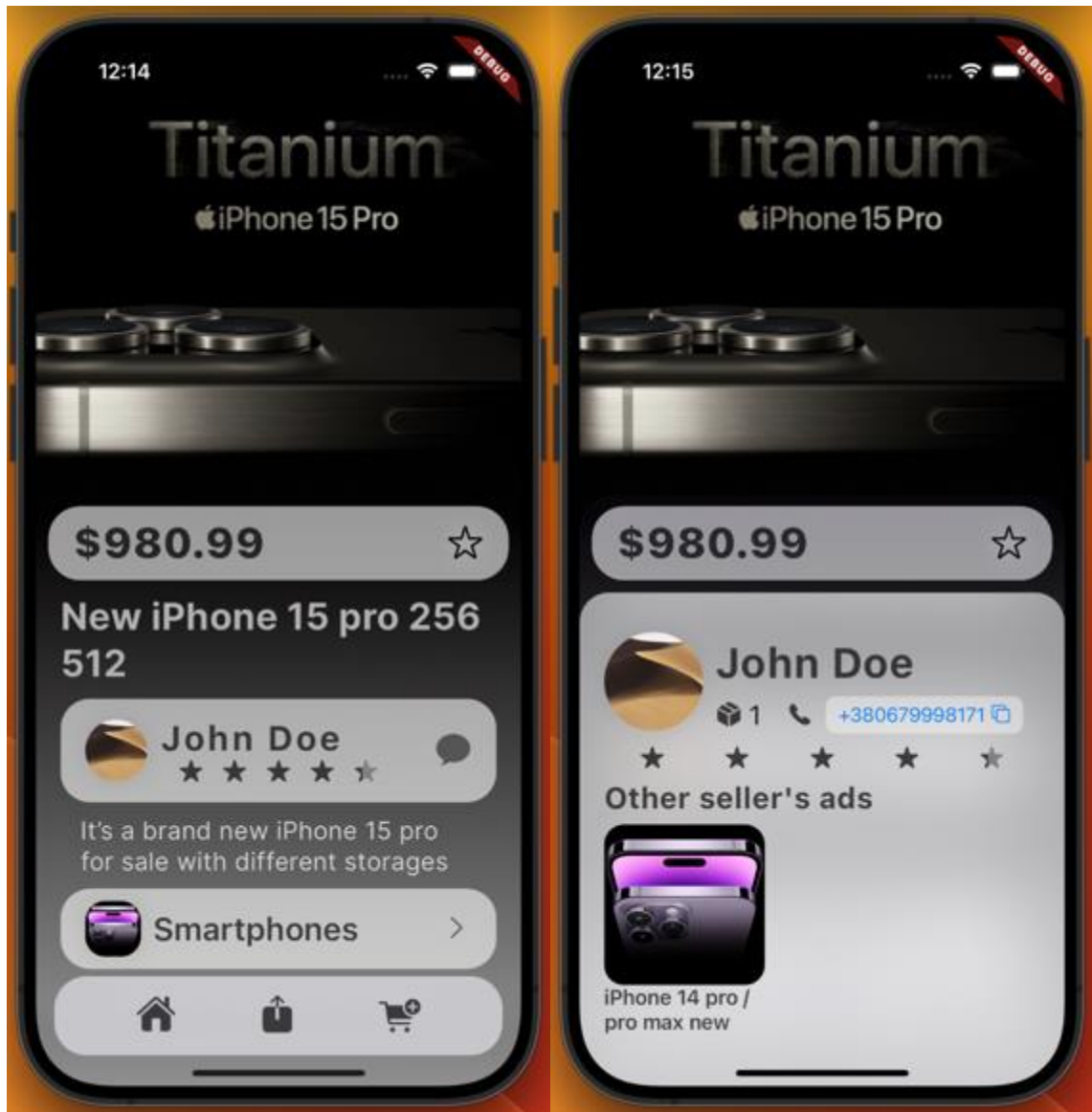
При натисканні на фотографію оголошення виконується перехід до екрану оголошення (рис. 4.5а) яке містить інформацію про нього, продавця та кнопки в навігаційному меню змінюються на повернутись до попереднього екрану, поділитись, додати чи вилучити з кошика. Також наявні кнопки переходу до категорії оголошення, занесення чи вилучення зі списку улюблених, детальної інформації про користувача та написання повідомлення до нього.

*а**б*

а – головна сторінка; б – бокове меню.

Рисунок 4.4 – Екран головної сторінки застосунку

Екран створення оголошень має два етапи. В першому (рис 4.6а) є кнопки для вибору фото, назви, опису, ціни, тегів, категорії оголошення та місцезнаходження продавця. У другому (рис 4.6б) в залежності від обраної категорії обираються необхідні фільтри оголошення. При невалідних даних (пропущене поле чи не введено коректних даних) виводиться повідомлення про помилку (рис. 4.7).

*a**б*

а – сторінка оголошення; б – поп-ап меню інформації про користувача.

Рисунок 4.5 – Екран оголошення

*a**б*

а – основна сторінка; б – сторінка вибору фільтрів.

Рисунок 4.6 – Екран створення оголошення

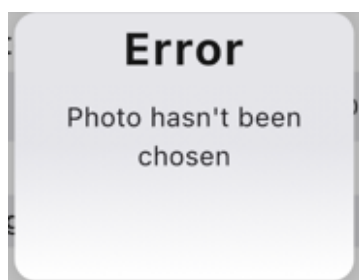


Рисунок 4.7 – Повідомлення про помилку при створенні оголошення

Для переходу до екрану з усіма чатами (рис. 4.8) потрібно на головній сторінці в доці натиснути кнопку з чатами, або через бокове меню натиснути на кнопку чатів. На екрані відображено список усіх чатів користувача.



Рисунок 4.8 – Экран з чатами користувача

Для переходу на екран чату з користувачем (рис. 4.9) потрібно на екрані оголошення натиснути на кнопку зображену на рис. 4.10 або на екрані чатів обрати потрібний чат.

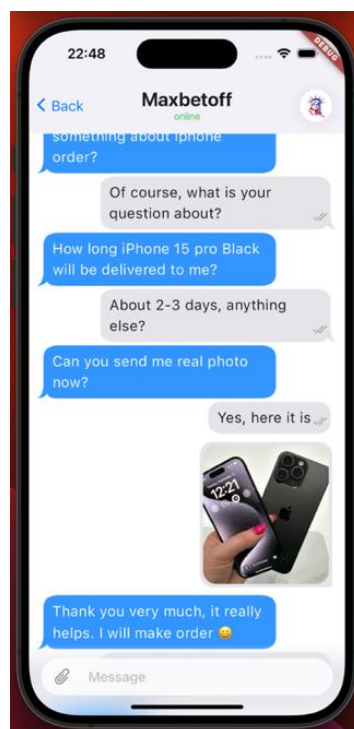


Рисунок 4.9 – Экран чату з користувачем

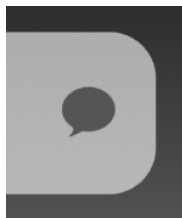


Рисунок 4.10 – Кнопка для переходу до чату з користувачем

На екрані чату є список всіх надісланих повідомлень, текстове поле для тексту, кнопка зображену на рис. 4.11 для вибору не текстового повідомлення. При натисканні на кнопку не текстового повідомлення з'являється меню з вибором бажаного повідомлення (рис. 4.12).

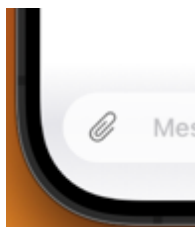


Рисунок 4.11 – Кнопка для вибору не текстового повідомлення

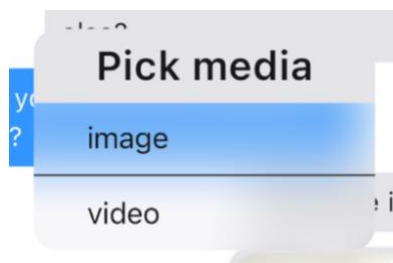


Рисунок 4.12 – Меню для вибору бажаного повідомлення

При натисканні на карточку продавця (рис. 4.13) з'являється поп-ап меню (рис. 4.56) де є більш детальна інформація про користувача, а саме ім'я, телефони, рейтинг, інші оголошення.

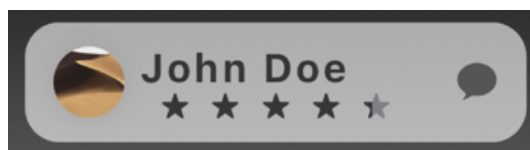
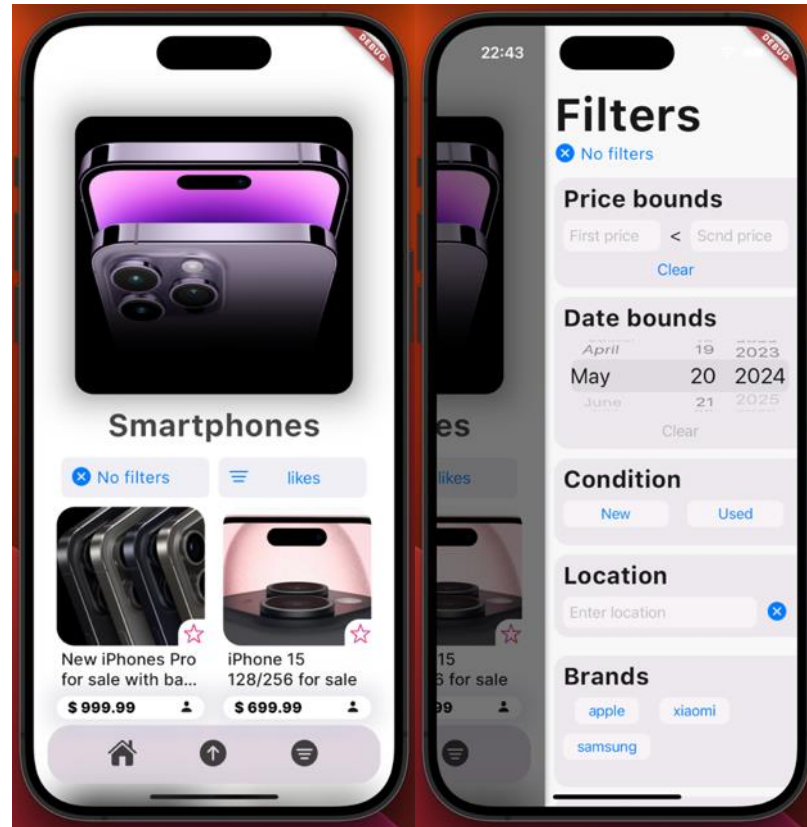


Рисунок 4.13 – Карточка продавця

Якщо натиснути на категорію оголошення то здійснюється перехід до екрану категорії товарів (рис 4.14а) де розташовані тільки оголошення цільової категорії, також є бокове меню з правого боку з фільтрами (рис. 4.14б).



а

б

а – відфільтровані оголошення; б – бокове меню фільтрів оголошень.

Рисунок 4.14 – Екран категорії оголошень

Кнопки в доку змінені на повернутись назад, повернутись до початку списку, відкрити бокове меню з фільтрами. Також наявна кнопка сортування оголошень.

В боковому меню стандартними для всіх категорій є фільтри за ціною (можна вказати мінімальну та максимальну ціну), датою (вказується дата до якої всі оголошення ігноруються), місцезнаходженням та станом товару (був у використанні чи новий). Всі інші фільтри різні для кожної категорії товарів. Для активації чи деактивації фільтрів які не є текстовими полями потрібно

натиснути на них. Після того як користувач встановив потрібні фільтри потрібно закрити бокове меню фільтрів та оновити сторінку потягнув її вниз.

Якщо ввести в пошуковий рядок (рис. 4.15) на головній сторінці назву бажаного товару та підтвердити, то здійсниться перехід до сторінки пошуку зображеного на рис. 4.16 де буде відображено шукані товари. Кнопки в доку замінені на перехід назад, перехід до початку списку та налаштувати сортування. Також рядок пошуку залишається та може бути використаний знову.

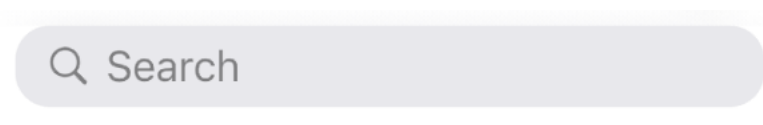


Рисунок 4.15 – Рядок пошуку

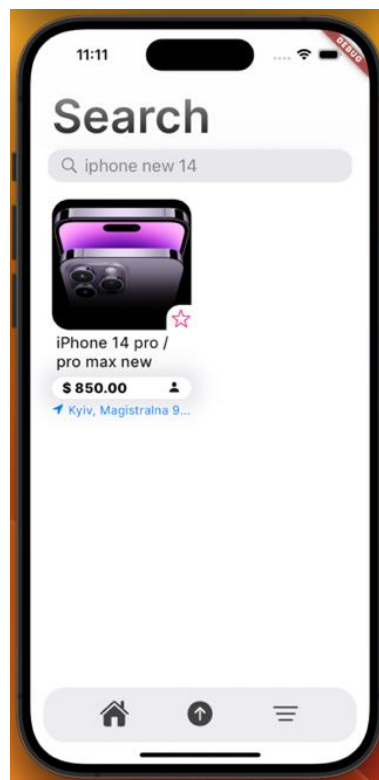


Рисунок 4.16 – Екран пошуку оголошень

Для налаштувань профілю потрібно на головній сторінці відкрити бокове меню та натиснути на кнопку «Profile» . Після цього виконується

перехід до екрану налаштувань профілю (рис 4.17), де наявні поля для зміни імені, електронної пошти, номеру телефону для входу, адреси доставки, альтернативного чату. Також є кнопки для зміни фотографії та підтвердження змін даних. Також є список телефонів для відображення в профілю користувача які можна додати чи видалити.

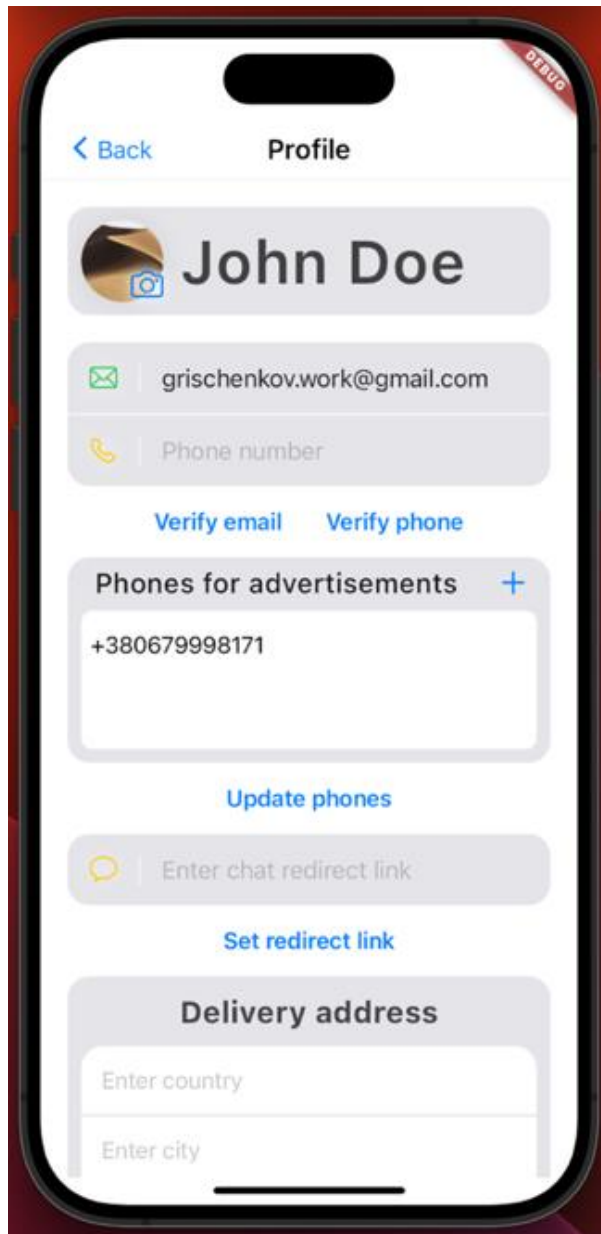


Рисунок 4.17 – Екран налаштувань профілю

Також є ще особливі категорії оголошень, наприклад нещодавно відвідані оголошення, улюблені оголошення, кошик оголошень, мої оголошення.

Усі вони відображають списки оголошень і назву категорії. Тобто нещодавно відвідані оголошення мають лише ті оголошення, які користувач відвідав, улюблені відображають лише ті оголошення які користувач заніс в улюблені, у кошику лише ті що користувач заніс до кошику, у моїх оголошеннях лише ті що створені безпосередньо користувачем. Екран нещодавно відвіданих зображено на рис. 4.18.

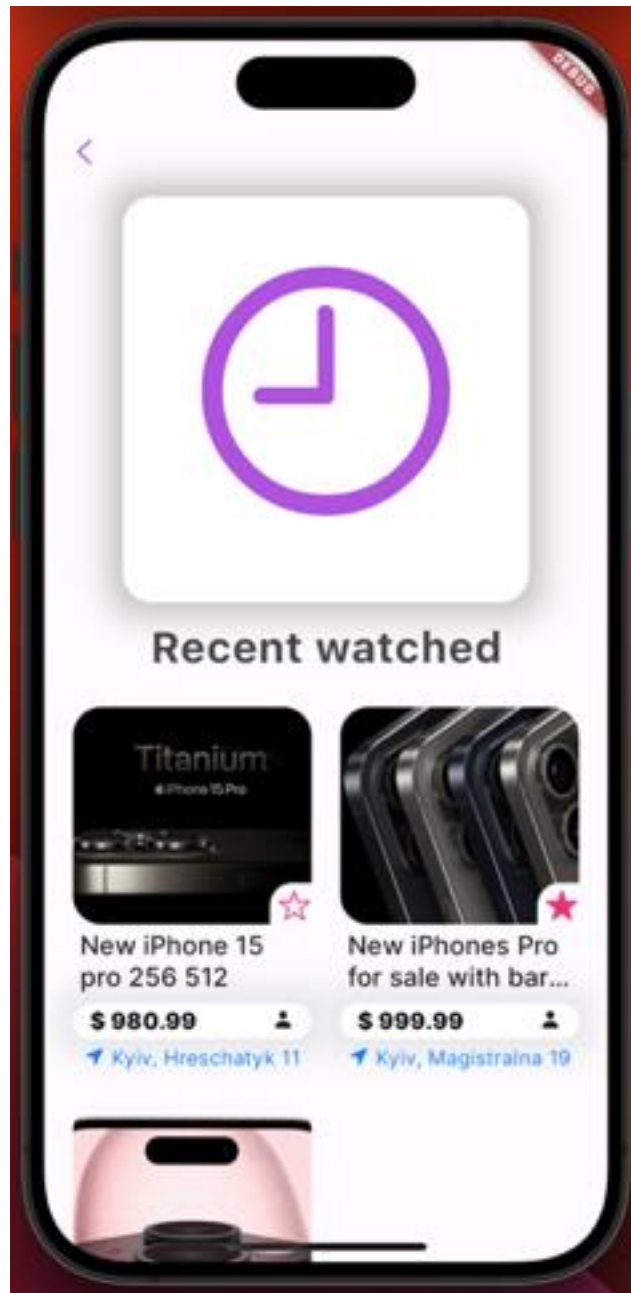


Рисунок 4.18 – Екран нещодавно відвіданих

На рис 4.19 зображено екран улюблених, екран кошику на рис 4.20, екран власних оголошень на рис. 4.21.

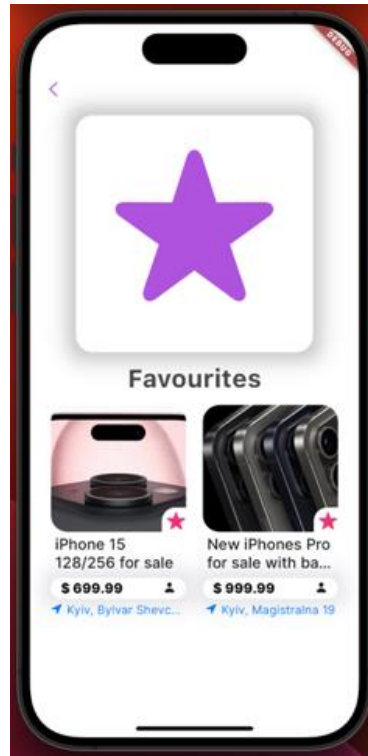


Рисунок 4.19 – Екран улюблених

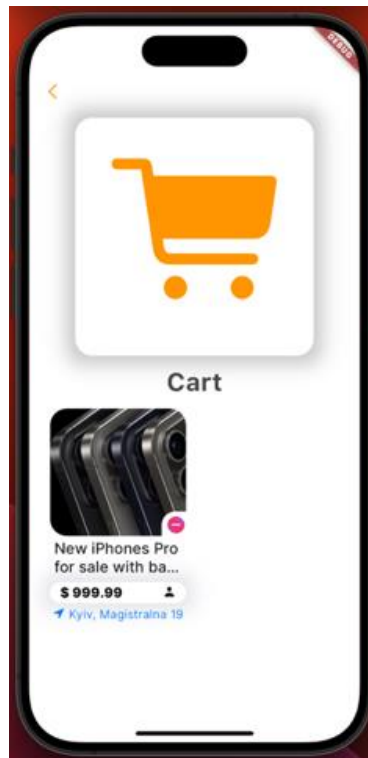


Рисунок 4.20 – Екран кошику

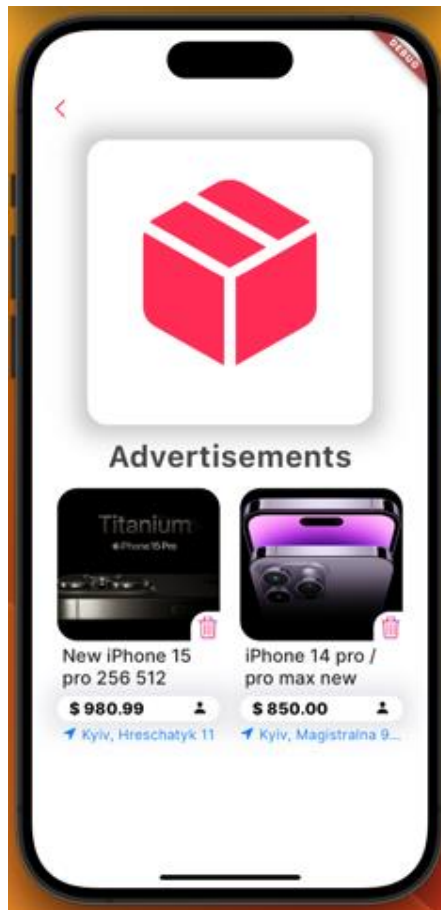


Рисунок 4.21 – Екран власних оголошень

Якщо користувач хоче подивитись улюблені чи кошик він може в боковому меню головної сторінки найти ці категорії, чи в доку головної сторінки.

Для переходу до нещодавно відвіданих треба натиснути кнопку зображену на рис. 4.22 на боковому меню головної сторінки, для переходу до створених оголошень потрібно в боковому меню головної сторінки натиснути пункт «Advertisements».

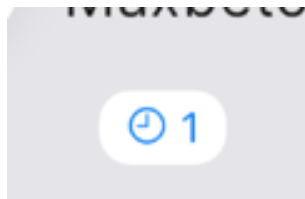


Рисунок 4.22 – Кнопка для переходу на екран нещодавно відвіданих

Для того щоб перейти на екран створення замовлення товару (рис. 4.23) потрібно перейти до кошику, а далі вибрати бажане замовлення. На екрані створення замовлення буде така сама інформація про товар з екрану оголошення, тільки в доку залишиться лише одна кнопка для створення замовлення.

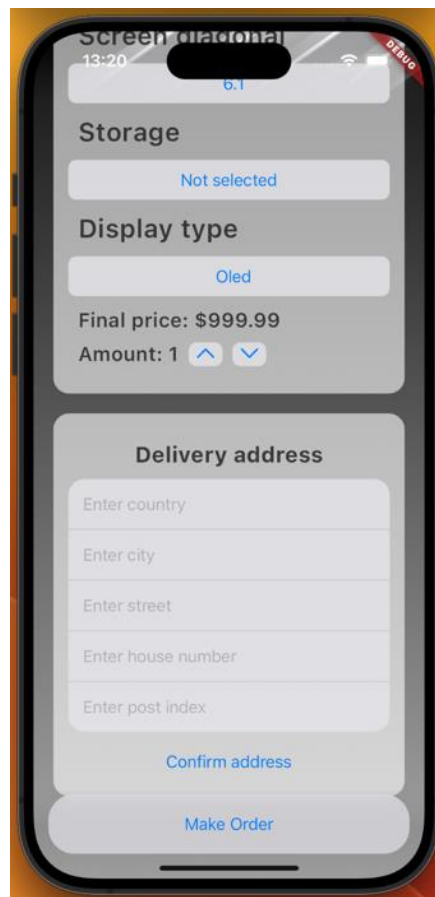


Рисунок 4.23 – Екран створення замовлення

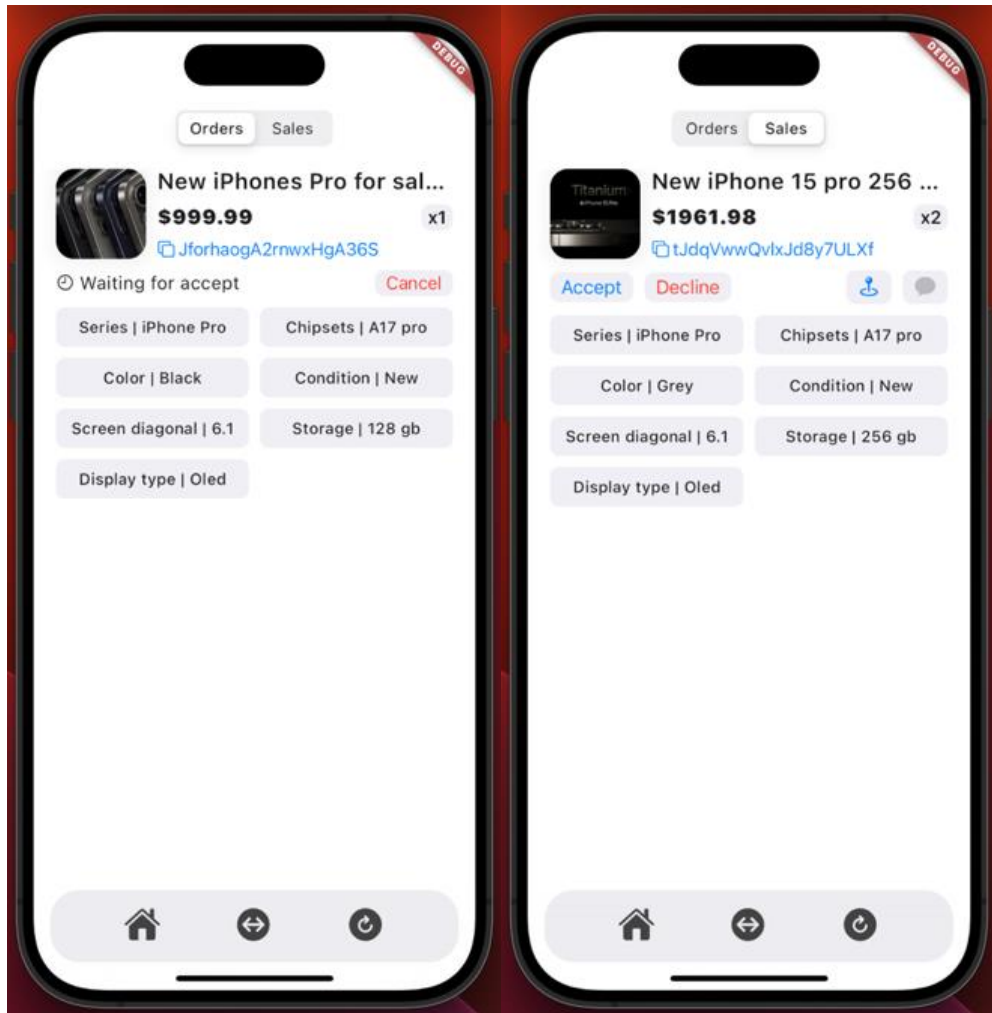
Також потрібно буде обрати конкретні фільтри товару, кількість товару та адресу доставки (якщо вона вказана у профілі користувача, то можна буде використати її). При невалідних даних, тобто не всіх обраних фільтрів чи без адреси буде виведено повідомлення про помилку зображене на рис 4.7.

Створене замовлення буде у списку оголошень на екрані замовлень та продажів щоб його подивитись потрібно на головній сторінці у боковому меню натиснути на «Orders and Sales». Всі свої замовлення можна переглянути на першій сторінці екрану, на другій розташовані продажі.

Екран замовлень (рис 4.24а) та продажів (рис 4.24б) об'єднаний в один та має дві вертикальні сторінки з замовленнями та продажами. На екрані як замовлення так і продажів можна побачити інформацію про товар: його назву, вартість, фото, кількість товару, вказані фільтри, ідентифікаційний номер товару.

Для замовлень ще є статус замовлення та кнопки в залежності від статусу замовлення.

Для продажів є кнопка для перегляду адреси, зв'язку з покупцем, зміни статусу замовлення.



а

б

а – сторінка замовлень; б – сторінка продажів.

Рисунок 4.24 – Екран замовлень та продаж користувача

На рис. 4.25 зображено навігаційну схему застосунку

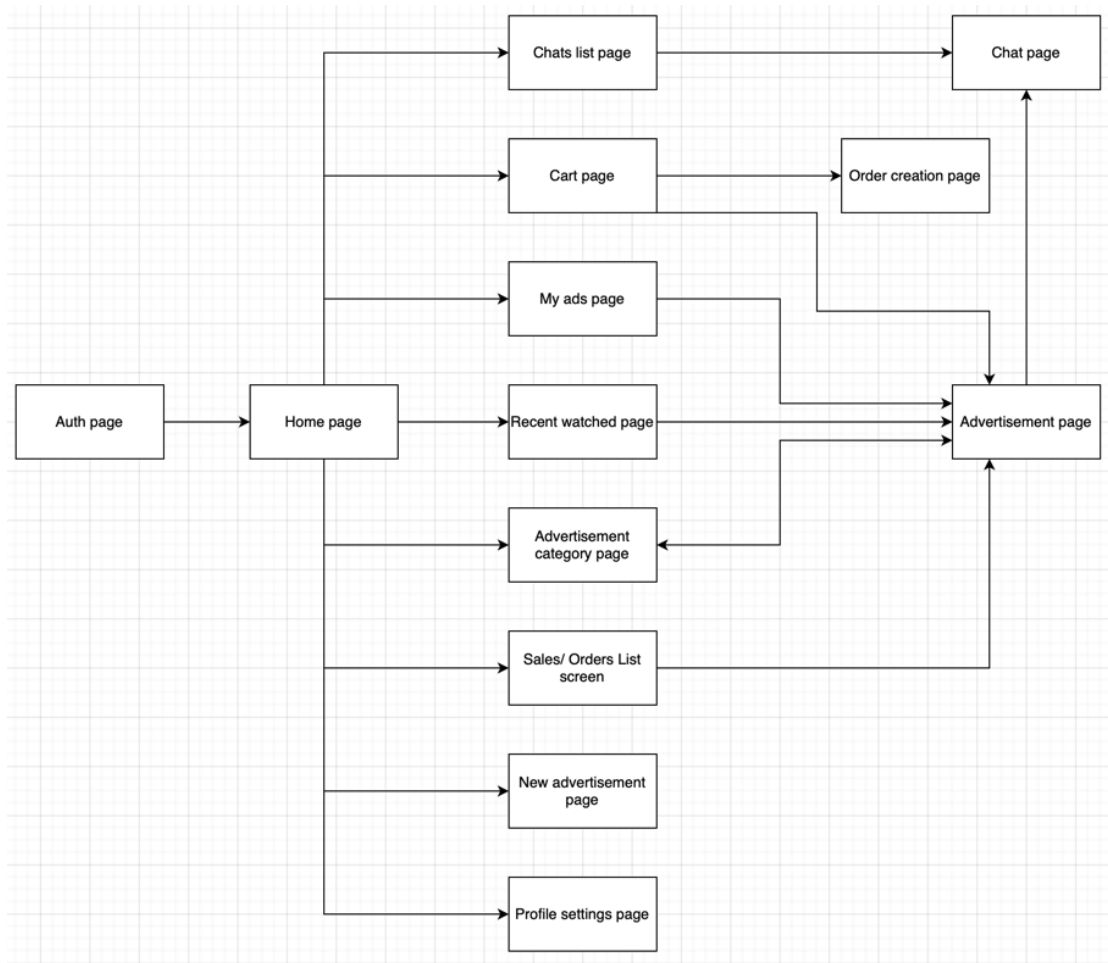


Рисунок 4.25 – Навігаційна схема застосунку

### 4.3 Тестування роботи додатку

Для перевірки якості створеного застосунку було проведено наступні тестування:

- тестування UI частини застосунку;
- функціональне тестування.

Для кожного тестування було створено чеклисти, які містять інформацію про пристрої, на яких проводилося тестування, перелік перевірок та результати цих перевірок.

Нижче на рис. 4.26 представлений чеклист функціонального тестування додатку.

| Критерій перевірки                                                                | iPhone 15 pro (iOS 17.4) | Pixel. 8 pro (Android 14) |
|-----------------------------------------------------------------------------------|--------------------------|---------------------------|
| Перевірити реєстрацію системи                                                     | +                        | +                         |
| Перевірити авторизацію системи                                                    | +                        | +                         |
| Перевірити авторизацію системи за смс                                             | +                        | +                         |
| Перевірити створення оголошення                                                   | +                        | +                         |
| Перевірити можливість вводу у поля конструктору оголошень                         | +                        | +                         |
| Перевірити можливість вибрати фільтри товару у конструкторі оголошень             | +                        | +                         |
| Перевірити повідомлення про помилку при невалідних даних у конструкторі оголошень | +                        | +                         |
| Перевірити видалення створеного оголошення                                        | +                        | +                         |
| Перевірити редагування створеного оголошення                                      | +                        | +                         |
| Перевірити занесення товарів в улюблені                                           | +                        | +                         |
| Перевірити вилучення товарів з улюблених                                          | +                        | +                         |
| Перевірити занесення товарів до кошика                                            | +                        | +                         |
| Перевірити вилучення товарів з кошика                                             | +                        | +                         |
| Перевірити створення замовлення товару                                            | +                        | +                         |
| Перевірити встановлення необхідних фільтрів товару у конструкторі замовлення      | +                        | +                         |
| Перевірити встановлення кількості товару у замовленні                             | +                        | +                         |
| Перевірити вводу адреси доставки у конструкторі повідомлення                      | +                        | +                         |
| Перевірити фільтрацію товарів за категорією                                       | +                        | +                         |
| Перевірити можливість додавання фільтрів у категорії товарів                      | +                        | +                         |
| Перевірити можливість вилучення фільтрів у категорії товарів                      | +                        | +                         |
| Перевірити роботу фільтрації товарів за встановленими фільтрами                   | +                        | +                         |
| Перевірити роботу пошуку товарів                                                  | +                        | +                         |
| Перевірити зміну даних профілю користувача                                        | +                        | +                         |
| Перевірити створення посилання на альтернативний чат                              | +                        | +                         |
| Перевірка створення адреси доставки за замовченням                                | +                        | +                         |
| Перевірити сортування товарів                                                     | +                        | +                         |
| Перевірити можливість створення чату з користувачем                               | +                        | +                         |
| Перевірити відправки повідомлення користувачу                                     | +                        | +                         |

Рисунок 4.26 – Чеклист функціонального тестування

На рис 4.27 зображено чеклист тестування UI частини застосунку.

| Критерій перевірки                                               | iPhone 15 pro (iOS 17.4) | Pixel. 8 pro (Android 14) |
|------------------------------------------------------------------|--------------------------|---------------------------|
| Перевірити коректне відображення сторінок згідно дизайну         | +                        | +                         |
| Перевірити наявність усіх сторінок                               | +                        | +                         |
| Перевірити коректне відображення шрифтів та стилів тексту        | +                        | +                         |
| Перевірити коректне відображення кнопок, блоків меню, тощо       | +                        | +                         |
| Перевірити коректність роботи анімацій та переходів              | +                        | +                         |
| Перевірити орфографію та граматику застосунку                    | +                        | +                         |
| Перевірити коректне відображення бокових меню                    | +                        | +                         |
| Перевірити коректне відображення застосованих фільтрів оголошень | +                        | +                         |
| Перевірити коректне відображення модальних вікон                 | +                        | +                         |

Рисунок 4.27 – Чеклист тестування UI частини застосунку

Проаналізувавши рис. 4.26 та 4.27 можна зробити висновок, що застосунок пройшов усі тест-кейси та його поведінка стабільна та передбачувана на обох платформах.

4.4 Висновки за 4 розділом

У четвертому розділі представлено призначення програми та мінімальні вимоги до апаратного та програмного забезпечення користувача. Описано процес налаштування середовища розробки проєкту.

Було детально розглянуто характеристики програми та спосіб її використання. Продемонстровано приклади основного функціоналу застосунку та способи взаємодії з користувачем.

Для перевірки надійності програми було проведено тестування UI частини та тестування функціональної частини розробленого додатку.

## ВИСНОВКИ

Під час виконання дипломної роботи було проаналізовано предметну область, тобто онлайн маркетплейси. Розібрано найпопулярніші аналоги програм, таких як «Amazon», «OLX» та «AliExpress», та порівняно їх функціональні можливості з функціональними можливостями «Sellers». Порівняльний аналіз показав що розроблюваний застосунок є актуальним та доцільним на фоні конкурентів.

При проєктуванні застосунку було обрано фреймворк Flutter як найбільш простий та ефективний спосіб зробити кросплатформний мобільний застосунок. Було обрано архітектуру застосунку Clean Architecture як найбільш зручний в масштабуванні варіант для такого великого застосунку. Також було порівняно середовища розробки, та за низкою переваг було обрано Android Studio.

Для збереження даних було обрано хмарне сховище Firebase від компанії Google. Було проаналізовано два можливі варіанти БД та обрано обидві для певних задач.

Було надано схему функціональних можливостей та наведено опис основних алгоритмів застосунку. Після цього визначені сутності для БД та також створено схему зв'язків сутностей між собою.

Розроблено користувацький інтерфейс, після цього розглянуто всі складові та їх призначення.

Після всіх етапів, було створено програмний продукт «Sellers» який дозволяє переглядати оголошення інших користувачів, створювати власні, сортувати та фільтрувати оголошення, створювати замовлення та чатуватись з користувачами за потреби.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Визначення маркетплейсу [Електрон. ресурс] – Режим доступу: <https://wezom.com.ua/ua/blog/yak-prosuvati-biznes-na-marketpleysah> - :~:text=складна багаторівнева платформа для реалізації товарів, яка об'єднує безліч продавців та надає їм інфраструктуру для ефективної онлайн-торгівлі.
2. Популярність маркетплейсів [Електрон. ресурс] – Режим доступу: <https://www.statista.com/statistics/1155246/leading-online-marketplaces-usa-average-monthly-visits/>.
3. Amazon додаток [Електрон. ресурс] – Режим доступу: <https://apps.apple.com/ua/app/amazon-shopping/id297606951?l=ru>.
4. OLX опис [Електрон. ресурс] – Режим доступу: <https://uk.wikipedia.org/wiki/OLX> - :~:text= платформа онлайн-оголошень, яка об'єднує людей для покупки, продажу або обміну товарами та послугами.
5. OLX додаток [Електрон. ресурс] – Режим доступу: <https://apps.apple.com/ua/app/olx-%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0-%D0%B0%D1%80%D0%B5%D0%BD%D0%B4%D0%B0-%D0%BA%D0%B2%D0%B0%D1%80%D1%82%D0%B8%D1%80/id663217552?l=ru>.
6. AliExpress опис [Електрон. ресурс] – Режим доступу: <https://uk.wikipedia.org/wiki/AliExpress> - :~:text= інтернет-магазин роздрібної торгівлі малого бізнесу Китаю, що пропонує товари міжнародним покупцям в Інтернеті.
7. AliExpress додаток [Електрон. ресурс] – Режим доступу: <https://apps.apple.com/ua/app/aliexpress-shopping-app/id436672029?l=ru>.

**ДОДАТОК А**  
**Технічне завдання**

## **А.1 Підстави для розробки**

Дипломна кваліфікаційна робота виконується за темою «Розробка мобільного застосунку для онлайн маркетплейсу» у відповідності з наказом №168 від 24 квітня 2024 р. за Національним університетом «Запорізька політехніка».

## **А.2 Призначення розробки**

Ціллю створення даного застосунку була в тому, щоб у користувачів була можливість легко та зручно обмінюватись товарами у будь-який зручний час. Даний продукт дозволяє переглядати оголошення, створювати власні та шукати по ключовим словам, створювати замовлення та продавати власні товари, додавати товари в улюблені та кошик, фільтрувати оголошення за категоріями та фільтрами в них, чатуватись з іншими користувачами.

## **А.3 Вимоги до програми**

### **А.3.1 Вимоги до функціональних характеристик**

Розроблюваний мобільний застосунок має наступні функціональні вимоги:

- реєстрація та авторизація через електронну адресу, також можливість авторизації через смс;
- створення оголошення про товар який повинен мати наступні параметри: фотографія товару, назва, опис, теги, місце знаходження, ціна, категорія товару, фільтри для обраної категорії;
- зберігання оголошень у віддаленій БД;
- можливість видаляти та редагувати створені оголошення;
- можливість перегляду оголошення інших продавців, додавати їх в улюблене та до кошику або видаляти звідти;

- можливість перегляду інформації про продавців: рейтинг, інші товари на продаж, контактну інформацію;
- можливість сортування оголошень за популярністю, ціною, датою;
- перегляд оголошень за категоріями та фільтрування їх;
- перегляд нещодавно відвіданих оголошень;
- перегляд рекомендованих товарів;
- можливість переглядати кошику;
- можливість створювати замовлення на покупку товару вказавши кількість товару та адресу доставки;
- можливість приймати або відхиляти замовлення покупців;
- мати вбудований чат та можливість переписуватись з продавцями в чат.

### **A.3.2 Вимоги до надійності**

Застосунок повинен розпізнавати ситуації, коли вводяться некоректні дані для реєстрації, авторизації та виводити відповідне повідомлення про помилку. Також при створенні чи редагуванні оголошення, коли користувач вводить або невалідні дані або не вводить їх зовсім. Також при відсутньому підключенні до мережі Інтернет повинно виводитись повідомлення про помилку.

### **A.3.3 Вимоги до технічних та програмних засобів**

Для забезпечення нормального функціонування додатку потрібно мати такі мінімальні апаратні значення:

- мобільний телефон з системою iOS(версія 10+) чи Android (версія 5.0+);
- доступ до Інтернету.

#### **А.4 Порядок контролю та приймання**

Дипломна робота допускається до захисту та захищається у присутності екзаменаційної комісії тільки після того, як вона була перевірена керівником, нормоконтролером, рецензентом та затверджена завідувачем кафедри.

**ДОДАТОК Б**  
**Текст програми**

## Б.1 Текст файла start\_page\_screen.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:icons_plus/icons_plus.dart';
import 'package:sellers/dependencies/firebase_service.dart';
import 'package:sellers/domain/cubit/auth/auth_cubit.dart';
import 'package:sellers/domain/cubit/auth/auth_state.dart';
import 'package:sellers/presentation/auth/widgets/auth_text_field.dart';
import
'package:sellers/presentation/auth/widgets/forgot_password_button.dart';
import 'package:sellers/presentation/auth/widgets/form_widget.dart';
import
'package:sellers/presentation/common_widgets/alert_dialog/alert_dialog.dart';
import
'package:sellers/presentation/common_widgets/animation/scale_animation.dart';
import
'package:sellers/presentation/common_widgets/load_indicator/load_indicator_wi
dget.dart';
import 'package:sellers/presentation/router/routes.dart';

class AuthPageWidget extends StatefulWidget {
  const AuthPageWidget({super.key});

  @override
  State<AuthPageWidget> createState() => _AuthPageWidgetState();
}

class _AuthPageWidgetState extends State<AuthPageWidget>
  with SingleTickerProviderStateMixin {
  final FirebaseService firebaseService = FirebaseService();
  late final TextEditingController mailController;
  late final TextEditingController passwordController;
  late final TextEditingController phoneNumberController;
  late final TextEditingController usernameController;
  late final Animation<double> animation;
  late final AnimationController controller;
  bool showSignUpForm = false;
  late final AuthCubit cubit;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      resizeToAvoidBottomInset: false,
      backgroundColor: CupertinoColors.white,
      body: Padding(
        padding: EdgeInsets.symmetric(
          horizontal: MediaQuery.of(context).size.width * 0.1),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ShaderMask(
              shaderCallback: (Rect bounds) {
                return LinearGradient(
                  begin: Alignment.topCenter,
                  end: Alignment.bottomCenter,
                  colors: [
                    CupertinoColors.systemGrey.withOpacity(0.8),
                    CupertinoColors.darkBackgroundGray,
                  ].createShader(
                    Rect.fromLTWH(0, 0, bounds.width, bounds.height));
              },
            ),
          ],
        ),
      ),
    );
  }
}
```

```

child: const Text(
  'Sellers',
  style: TextStyle(
    fontSize: 64,
    fontWeight: FontWeight.w600,
    color: CupertinoColors.lightBackgroundGray,
    height: 1,
  ),
),
),
Text(
  'Sell everything to everyone...',
  style: GoogleFonts.rajdhani(
    fontSize: 20, fontWeight: FontWeight.w500),
),
const SizedBox(height: 5),
ScaleAnimationWidget(
  isHided: !showSignUpForm,
  alignment: Alignment.bottomCenter,
  child: Column(
    children: [
      AuthFormWidget(
        children: [
          BlocBuilder<AuthCubit, AuthState>(
            bloc: cubit,
            buildWhen: (prev, current) {
              return current.usernameStatus !=
                prev.usernameStatus;
            },
            builder: (context, snapshot) {
              return AuthTextFieldWidget(
                controller: usernameController,
                placeholderText: 'Username',
                validStatus: cubit.state.usernameStatus,
                prefixIcon: CupertinoIcons.person);
            },
          ),
          const SizedBox(height: 0.5),
          BlocBuilder<AuthCubit, AuthState>(
            bloc: cubit,
            buildWhen: (prev, current) {
              return current.phoneStatus != prev.phoneStatus;
            },
            builder: (context, snapshot) {
              return AuthTextFieldWidget(
                controller: phoneNumberController,
                keyboardType: TextInputType.phone,
                validStatus: cubit.state.phoneStatus,
                placeholderText: 'Phone number',
                prefixIcon: CupertinoIcons.phone);
            },
          ),
        ],
      ),
      const SizedBox(height: 10),
    ],
  ),
  AuthFormWidget(
    children: [
      BlocBuilder<AuthCubit, AuthState>(
        bloc: cubit,
        buildWhen: (prev, current) {
          return current.mailStatus != prev.mailStatus;
        },
        builder: (context, snapshot) {

```

```

        return AuthTextFieldWidget(
          controller: mailController,
          placeholderText: 'Email',
          keyboardType: TextInputType.emailAddress,
          validStatus: cubit.state.mailStatus,
          prefixIcon: CupertinoIcons.mail);
      )),
    const SizedBox(height: 0.5),
    BlocBuilder<AuthCubit, AuthState>(
      bloc: cubit,
      buildWhen: (prev, current) {
        return current.passwordStatus != prev.passwordStatus;
      },
      builder: (context, snapshot) {
        return Row(
          mainAxisAlignment: MainAxisAlignment.start,
          children: [
            Expanded(
              child: AuthTextFieldWidget(
                controller: passwordController,
                validStatus: cubit.state.passwordStatus,
                placeholderText: 'Password',
                prefixIcon: CupertinoIcons.lock,
                obscureText: true),
            ),
            ForgotPasswordButton(
              showSignUpform,
              cubit: cubit,
            ),
          ],
        );
      })),
  ],
),
const SizedBox(height: 10),
SizedBox(
  height: 46,
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      CupertinoButton(
        onPressed: () async {
          if (!showSignUpform) {
            showSignUpform = true;
            setState(() {});
            return;
          }
          if (await cubit.onSignUp()) {
            Navigator.of(context).pushNamed(Routes.homePage);
          }
        },
        padding: const EdgeInsets.symmetric(horizontal: 10),
        child: const Text(
          'Sign up',
          style: TextStyle(
            color: CupertinoColors.link,
            fontWeight: FontWeight.w500,
            fontSize: 18,
          ),
        ),
      ),
      CupertinoButton(
        color: CupertinoColors.lightBackgroundGray,
        padding: const EdgeInsets.symmetric(horizontal: 15),

```

```

minSize: 46,
borderRadius: BorderRadius.circular(12),
onPressed: () async {
  if (showSignUpform) {
    showSignUpform = false;
    setState(() {});
    return;
  }
  if (await showLoadIndicator<bool>(
    cubit.onSignIn(), context)) {
    Navigator.of(context).pushNamed(Routes.homePage);
  }
},
child: const Text(
  'Sign in',
  style: TextStyle(
    color: CupertinoColors.link,
    fontWeight: FontWeight.w400,
    fontSize: 18,
  ),
),
),
/*CupertinoButton(
  onPressed: () {
    showAlertDialog(
      context, 'Error', 'Sorry, not works for now');
  },
padding: EdgeInsets.zero,
child: Container(
  height: double.infinity,
  width: 46,
  decoration: const BoxDecoration(
    color: CupertinoColors.lightBackgroundGray,
    borderRadius:
      BorderRadius.all(Radius.circular(12))),
  child: Padding(
    padding: const EdgeInsets.all(8.0),
    child: Brand(
      Brands.google,
      size: 24,
    ),
  ),
),
),
),
CupertinoButton(
  onPressed: () {
    showAlertDialog(context, 'Error',
      'Sorry, not works via mail for now. Try via phone
number instead');
  },
padding: EdgeInsets.zero,
child: Container(
  width: 46,
  alignment: Alignment.center,
  decoration: const BoxDecoration(
    color: CupertinoColors.lightBackgroundGray,
    borderRadius:
      BorderRadius.all(Radius.circular(12))),
  child: const Icon(
    Icons.apple,
    color: CupertinoColors.darkBackgroundGray,
    size: 36,
  ),
),
)*/
],

```

```

        ),
      ),
      const SizedBox(height: 40),
    ],
  ),
));
}

@override
void initState() {
  super.initState();
  mailController = TextEditingController();
  passwordController = TextEditingController();
  phoneNumberController = TextEditingController();
  usernameController = TextEditingController();
  controller = AnimationController(
    vsync: this, duration: const Duration(milliseconds: 250));
  animation = Tween<double>(begin: 0.0, end: 1.0).animate(controller);
  cubit = AuthCubit(AuthState.init(),
    mailController: mailController,
    passwordController: passwordController,
    phoneNumberController: phoneNumberController,
    usernameController: usernameController);
}

@override
void dispose() {
  mailController.dispose();
  passwordController.dispose();
  phoneNumberController.dispose();
  usernameController.dispose();
  controller.dispose();
  super.dispose();
}
}

```

## Б.2 Текст файлу home\_page\_screen.dart

```

import 'dart:ui';

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_inner_drawer/inner_drawer.dart';
import 'package:google_fonts/google_fonts.dart';
import 'package:sellers/data/repository/product_repository.dart';
import 'package:sellers/dependencies/firebase_service.dart';
import
'package:sellers/dependencies/notifications/notification_service.dart';
import 'package:sellers/domain/entity/product.dart';
import 'package:sellers/domain/use_case/chat_use_case.dart';
import 'package:sellers/domain/use_case/product_use_case.dart';
import
'package:sellers/presentation/common_widgets/dock_widget/dock_widget.dart';
import
'package:sellers/presentation/common_widgets/load_indicator/load_indicator_wi
dget.dart';
import
'package:sellers/presentation/common_widgets/product_widget/product_action.da
rt';

```

```

import
'package:sellers/presentation/common_widgets/product_widget/product_widget.dart';
import 'package:sellers/presentation/common_widgets/search_widget.dart';
import
'package:sellers/presentation/common_widgets/smart_list_view/infinite_listview.dart';
import
'package:sellers/presentation/common_widgets/smart_list_view/item_provider.dart';
import
'package:sellers/presentation/common_widgets/smart_list_view/product_provider.dart';
import
'package:sellers/presentation/home_page/categories/categories_widget.dart';
import
'package:sellers/presentation/home_page/side_menu/side_menu_widget.dart';
import 'package:sellers/presentation/router/routes.dart';

class HomePageScreenWidget extends StatefulWidget {
  const HomePageScreenWidget({super.key});

  static final GlobalKey<InnerDrawerState> drawerKey =
    GlobalKey<InnerDrawerState>();

  @override
  State<HomePageScreenWidget> createState() => _HomePageScreenWidgetState();
}

class _HomePageScreenWidgetState extends State<HomePageScreenWidget>
  with WidgetsBindingObserver {
  late final ProductProvider cubit;
  late final Future<void>? future;

  @override
  void initState() {
    super.initState();
    cubit = ProductProvider((docLimit, {startAfter}) async {
      final snap = await ProductUseCase.getRecommendedProducts(docLimit,
        startAfter: startAfter);
      final products = ProductUseCase.snapToProducts(snap.docs);

      return (products, snap.docs.lastOrNull);
    });
    future = FirebaseService().currentUser?.reload();
    WidgetsBinding.instance.addObserver(this);
  }

  @override
  void didChangeAppLifecycleState(AppLifecycleState state) {
    super.didChangeAppLifecycleState(state);
    var userId = FirebaseService().currentUser?.uid;

    if (userId == null) return;

    if (state == AppLifecycleState.resumed) {
      ChatUseCase.updateUser(userId, {'isOnline': true});
    } else {
      ChatUseCase.updateUser(userId, {'isOnline': false});
    }
  }

  @override
  void dispose() {

```

```

WidgetsBinding.instance.removeObserver(this);
super.dispose();
}

@override
Widget build(BuildContext context) {
  final deviceWidth = MediaQuery.of(context).size.width;
  return FutureBuilder(
    future: future,
    builder: (context, snapshot) {
      if (snapshot.connectionState != ConnectionState.done) {
        return const Scaffold(
          resizeToAvoidBottomInset: false,
          backgroundColor: CupertinoColors.white,
          body: LoadIndicatorWidget());
      }
      return InnerDrawer(
        key: HomePageScreenWidget.drawerKey,
        onTapClose: true,
        leftChild: const SideMenuWidget(),
        offset: (deviceWidth < 410)
          ? const IDOffset.horizontal(0.5)
          : const IDOffset.horizontal(0.4),
        scaffold: Scaffold(
          backgroundColor: CupertinoColors.white,
          resizeToAvoidBottomInset: false,
          body: Stack(children: [
            BlocBuilder<ItemProvider<Product, ProductState>,
ProductState>(
              bloc: cubit,
              builder: (context, state) {
                return InfiniteListView(
                  provider: cubit,
                  headerWidget: Column(
                    children: [
                      Hero(
                        tag: 'search_tag',
                        child: SearchWidget(
                          onSubmitted: (String searchString) {
                            Navigator.of(context).pushNamed(
                              Routes.searchScreen,
                              arguments: searchString);
                          },
                        ),
                      ),
                    ],
                  ),
                  const CategoriesWidget(),
                  Padding(
                    padding:
                      const EdgeInsets.symmetric(horizontal: 25),
                    child: Row(
                      children: [
                        Expanded(
                          child: Text(
                            'Special for you',
                            style: GoogleFonts.inter(
                              fontSize: 24,
                              fontWeight: FontWeight.w600),
                          ),
                        ),
                        const Icon(
                          CupertinoIcons.line_horizontal_3_decrease,
                          color: CupertinoColors.link,
                        )

```

```

        ],
      ),
    ),
    const SizedBox(height: 10),
  ],
),
builder: (context, index) {
  return ProductWidget(
    product: cubit.items[index],
    heroTag: '',
    action: ProductAction.actions['favs']!);
},
mainAxisExtent:
  (MediaQuery.of(context).size.width / 2 - 45) +
115,
);
  ));
// const AppBarWidget(),
DockWidget(backgroundcolor: CupertinoColors.white, buttons: [
  Hero(
    tag: 'cart_screen',
    child: FittedBox(
      child: DockButton(
        icon: CupertinoIcons.cart_fill,
        onPressed: () {
          Navigator.of(context).pushNamed(Routes.cartScreen);
        },
      ),
    ),
  ),
  Hero(
    tag: 'fav_screen',
    child: FittedBox(
      child: DockButton(
        icon: CupertinoIcons.star_fill,
        size: 31,
        onPressed: () {
          Navigator.of(context)
            .pushNamed(Routes.favouritesScreen);
        },
      ),
    ),
  ),
  DockButton(
    icon: CupertinoIcons.add_circled_solid,
    onPressed: () {
      Navigator.of(context).pushNamed(Routes.newProductScreen);
    },
  ),
  DockButton(
    icon: CupertinoIcons.chat_bubble_2_fill,
    size: 36,
    onPressed: () {
      Navigator.of(context).pushNamed(Routes.chatsScreen);
    },
  ),
]),
]),
);
});
}
}

```

### Б.3 Текст файла product\_screen.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:sellers/data/repository/account_repository.dart';
import 'package:sellers/domain/entity/product.dart';
import 'package:sellers/domain/use_case/account_use_case.dart';
import
'package:sellers/presentation/common_widgets/blurred_background_widget/blurred_background_widget.dart';
import
'package:sellers/presentation/common_widgets/dock_widget/dock_widget.dart';
import
'package:sellers/presentation/common_widgets/photo_widgets/photo_widget.dart'
;
import 'package:sellers/presentation/product_screen/add_to_cart_widget.dart';
import 'package:sellers/presentation/product_screen/body_widget.dart';
import 'package:sellers/presentation/product_screen/product_hero.dart';

class ProductScreenWidget extends StatefulWidget {
  const ProductScreenWidget({
    super.key,
    required this.product,
    required this.heroTag,
    this.dockButtons,
    this.widget,
  });

  final Product product;
  final String heroTag;
  final Widget? widget;
  final List<Widget>? dockButtons;

  @override
  State<ProductScreenWidget> createState() => _ProductScreenWidgetState();
}

class _ProductScreenWidgetState extends State<ProductScreenWidget>
  with SingleTickerProviderStateMixin {
  late final Animation<double> animation;
  late final AnimationController controller;
  late final ScrollController scrollController = ScrollController();

  @override
  void initState() {
    super.initState();
    controller = AnimationController(
      vsync: this, duration: const Duration(milliseconds: 450));
    animation =
      CurvedAnimation(parent: controller, curve: Curves.fastOutSlowIn);
    AccountRepository().recentWatchedList.add(widget.product);
    Future.delayed(const Duration(milliseconds: 300))
      .whenComplete(() => controller.forward());
  }

  @override
  void dispose() {
    controller.dispose();
    scrollController.dispose();
    super.dispose();
  }
}
```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    resizeToAvoidBottomInset: false,
    backgroundColor: CupertinoColors.white,
    body: Stack(
      children: [
        ConstrainedBox(
          constraints:
            BoxConstraints(maxHeight: MediaQuery.of(context).size.width),
          child: Stack(
            alignment: Alignment.bottomCenter,
            children: [
              ProductHero(
                tag: widget.product.id + widget.heroTag,
                child: PhotoWidget(
                  url: widget.product.photo,
                  width: double.infinity,
                  height: double.infinity,
                ),
              ),
            ],
          ),
        FadeTransition(
          opacity: animation,
          child: Stack(
            children: [
              BackgroundEffectWidget(
                product: widget.product,
                color: widget.product.gradientColor,
                controller: scrollController,
                maxHeight: MediaQuery.of(context).size.width - 40,
              ),
              ListView(
                controller: scrollController,
                physics: const BouncingScrollPhysics(),
                children: [
                  SizedBox(height: MediaQuery.of(context).size.width -
100),
                  Container(
                    clipBehavior: Clip.antiAlias,
                    margin: const EdgeInsets.symmetric(horizontal: 10),
                    height: 60,
                    decoration: BoxDecoration(
                      borderRadius: BorderRadius.circular(26)),
                    child: BlurredBackgroundWidget(
                      backgroundColor:
CupertinoColors.white.withOpacity(0.6),
                      child: Padding(
                        padding: const EdgeInsets.symmetric(horizontal:
20),
                        child: Align(
                          alignment: Alignment.center,
                          child: Row(
                            mainAxisAlignment: MainAxisAlignment.min,
                            children: [
                              Text(
                                '\${(widget.product.price /
100).toStringAsFixed(2)}',
                                style: TextStyle(
                                  fontSize: 34,
                                  fontWeight: FontWeight.w800,
                                  color: CupertinoColors.darkBackgroundGray

```



```

        size: 32,
        onPressed: () {},
      ),
      AddToCartWidget(product: widget.product),
    ],
    backgroundColor: Color.alphaBlend(
      widget.product.gradientColor.withOpacity(0.5),
      CupertinoColors.white,
    ),
  ),
),
],
),
);
}
}

class BackgroundEffectWidget extends StatefulWidget {
  const BackgroundEffectWidget({
    super.key,
    required this.product,
    required this.color,
    required this.controller,
    required this.maxHeight,
  });

  final Color color;
  final Product product;
  final ScrollController controller;
  final double maxHeight;

  @override
  State<BackgroundEffectWidget> createState() =>
    _BackgroundEffectWidgetState();
}

class _BackgroundEffectWidgetState extends State<BackgroundEffectWidget> {
  late double currentOffset;

  @override
  void initState() {
    super.initState();
    currentOffset = 0;
    widget.controller.addListener(changeOffset);
  }

  @override
  void dispose() {
    widget.controller.removeListener(changeOffset);
    super.dispose();
  }

  void changeOffset() {
    currentOffset = widget.controller.offset;
    setState(() {});
  }

  @override
  Widget build(BuildContext context) {
    var top = widget.maxHeight - currentOffset;
    if (top > widget.maxHeight) top = widget.maxHeight;
    return Padding(
      padding: EdgeInsets.only(top: top > 0 ? top : 0),
      child: Column(

```

```

children: [
  Container(
    width: double.infinity,
    height: 40,
    decoration: BoxDecoration(
      gradient: LinearGradient(
        begin: Alignment.bottomCenter,
        end: Alignment.topCenter,
        colors: [
          widget.color.withOpacity(1),
          widget.color.withOpacity(0.7),
          widget.color.withOpacity(0)
        ],
      ),
    ),
  ),
  Expanded(
    child: Container(
      color: CupertinoColors.white,
      child: Container(
        width: double.infinity,
        height: double.infinity,
        decoration: BoxDecoration(
          gradient: LinearGradient(
            begin: Alignment.topCenter,
            end: Alignment.bottomCenter,
            colors: [
              widget.color.withOpacity(1),
              widget.color.withOpacity(0.4),
            ]),
        ),
      ),
    ),
  ),
),
],
);
}

```

#### Б.4 Текст файла category\_screen\_widget.dart

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:flutter_inner_drawer/inner_drawer.dart';
import 'package:sellers/data/repository/product_repository.dart';
import
'package:sellers/domain/cubit/category_screen/category_screen_cubit.dart';
import
'package:sellers/domain/cubit/category_screen/category_screen_state.dart';
import 'package:sellers/domain/cubit/category_screen/date_filter.dart';
import 'package:sellers/domain/cubit/category_screen/location_filter.dart';
import 'package:sellers/domain/cubit/category_screen/price_filter.dart';
import 'package:sellers/domain/entity/category.dart';
import 'package:sellers/domain/entity/product.dart';
import 'package:sellers/domain/use_case/product_use_case.dart';
import 'package:sellers/presentation/category_screen/filters_drawer.dart';
import
'package:sellers/presentation/category_screen/widgets/selected_filters_widget
.dart';

```

```

import
'package:sellers/presentation/common_widgets/dock_widget/dock_widget.dart';
import
'package:sellers/presentation/common_widgets/photo_widgets/photo_widget.dart'
;
import
'package:sellers/presentation/common_widgets/product_widget/product_action.da
rt';
import
'package:sellers/presentation/common_widgets/product_widget/product_widget.da
rt';import
'package:sellers/presentation/common_widgets/smart_list_view/infinite_listvie
w.dart';
import
'package:sellers/presentation/common_widgets/smart_list_view/item_provider.da
rt';
import
'package:sellers/presentation/common_widgets/smart_list_view/product_provider
.dart';

class CategoryScreenWidget extends StatefulWidget {
  const CategoryScreenWidget({super.key, required this.category});

  final ProductCategory category;

  @override
  State<CategoryScreenWidget> createState() => _CategoryScreenWidgetState();
}

class _CategoryScreenWidgetState extends State<CategoryScreenWidget> {
  late final ProductProvider productCubit;
  late final CategoryScreenCubit categoryScreenCubit;
  late final ScrollController controller;
  final GlobalKey<InnerDrawerState> drawerKey = GlobalKey();

  @override
  void initState() {
    super.initState();
    controller = ScrollController();
    categoryScreenCubit = CategoryScreenCubit(
      CategoryScreenState(
        addedFilters: {},
        brands: [],
        dateFilter: DateFilter(false),
        priceFilter: PriceFilter(false),
        locationFilter: LocationFilter(false)),
      widget.category);

    productCubit = ProductProvider(getProducts);
  }

  @override
  Widget build(BuildContext context) {
    return InnerDrawer(
      key: drawerKey,
      onTapClose: true,
      rightChild: FiltersDrawerWidget(
        cubit: categoryScreenCubit,
        backgroundAccent: widget.category.dominantColor,
        globalKey: drawerKey,
      ),
      scaffold: Scaffold(
        backgroundColor: CupertinoColors.white,
        resizeToAvoidBottomInset: false,

```

```

body: Stack(
  children: [
    BlocBuilder<ItemProvider<Product, ProductState>, ProductState>(
      bloc: productCubit,
      builder: (context, state) {
        return InfiniteListView(
          headerWidget: Column(
            children: [
              Hero(
                tag: widget.category.id,
                createRectTween: (begin, end) {
                  return MaterialRectCenterArcTween(
                    begin: begin,
                    end: end,
                  );
                },
              ),
              child: Container(
                decoration: BoxDecoration(boxShadow: [
                  BoxShadow(
                    color: CupertinoColors.darkBackgroundGray
                      .withOpacity(0.35),
                    spreadRadius: 10,
                    blurRadius: 30)
                ]),
              child: PhotoWidget(
                width: MediaQuery.of(context).size.width - 90,
                height: MediaQuery.of(context).size.width - 90,
                borderRadius: BorderRadius.circular(20),
                url: widget.category.photoUrl,
              ),
            ),
          ),
          const SizedBox(height: 10),
          Text(
            widget.category.title,
            style: TextStyle(
              color: CupertinoColors.darkBackgroundGray
                .withOpacity(0.8),
              fontWeight: FontWeight.w700,
              fontSize: 34),
          ),
          Padding(
            padding: const EdgeInsets.symmetric(
              horizontal: 30, vertical: 10),
            child: Row(
              children: [
                Expanded(
                  child: Container(
                    padding: const EdgeInsets.only(
                      top: 7, left: 10, bottom: 7),
                    decoration: BoxDecoration(
                      color: CupertinoColors
                        .extraLightBackgroundGray,
                      borderRadius: BorderRadius.circular(8),
                    ),
                    child: SelectedFiltersWidget(
                      cubit: categoryScreenCubit,
                      backgroundColor: CupertinoColors.link
                        .withOpacity(0.1)),
                  ),
                ),
                const SizedBox(width: 10),
                Expanded(
                  child: CupertinoButton(

```



```

        size: 34,
        onPressed: () {
          controller.position.animateTo(
            controller.initialScrollOffset,
            duration: const Duration(seconds: 1),
            curve: Curves.fastOutSlowIn);
        },
      ),
      DockButton(
        icon:
CupertinoIcons.line_horizontal_3_decrease_circle_fill,
        onPressed: () {
          drawerKey.currentState?.toggle();
        },
      )
    ],
    backgroundColor: CupertinoColors.white,
  )
),
),
);
}

Future<(List<Product>, QueryDocumentSnapshot<Map<String, dynamic>>?)>
  getProducts(docLimit, {startAfter}) async {
  final dateBound = categoryScreenCubit.state.dateFilter.isActive
    ? categoryScreenCubit.dateBound
    : null;
  final priceBound = categoryScreenCubit.state.priceFilter.isActive
    ? categoryScreenCubit.state.priceFilter.getPriceBounds(
        categoryScreenCubit.firstPriceBound.text,
        categoryScreenCubit.secondPriceBound.text)
    : null;
  final location = categoryScreenCubit.state.locationFilter.isActive
    ? categoryScreenCubit.locationController.text
    : null;
  final List<Product> filteredProducts = [];
  QueryDocumentSnapshot<Map<String, dynamic>>? last;
  while (true) {
    final snap = await ProductUseCase.getCategoryProducts(
      widget.category, docLimit,
      startAfter: startAfter);
    final products = ProductUseCase.snapToProducts(snap.docs);
    last = snap.docs.lastOrNull;
    final addedFilters = categoryScreenCubit.state.addedFilters
      .map((key, value) => MapEntry(key, value.map((e) => e.value)));
    for (var prod in products) {
      bool success = true;
      if (location != null && !prod.location.contains(location)) {
        continue;
      }
      if (priceBound != null) {
        if (priceBound.$1 != null && prod.price < priceBound.$1!) {
          continue;
        }
        if (priceBound.$2 != null &&
          priceBound.$2! > 0 &&
          prod.price > priceBound.$2!) {
          continue;
        }
      }
    }
    if (dateBound != null) {
      if (dateBound.isAfter(prod.date)) {

```

```

        continue;
    }
}
addedFilters.forEach((filterName, values) {
    for (final value in values) {
        if (!success) {
            break;
        }
        success = prod.filters[filterName]?.contains(value) ?? false;
    }
    if (!success) {
        return;
    }
});
if (success) filteredProducts.add(prod);
}
if (filteredProducts.length >= docLimit || products.length < docLimit)
{
    break;
}
}
return (filteredProducts, last);
}
}

```

## Б.5 Текст файла cart\_screen\_widget.dart

```

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:sellers/data/repository/account_repository.dart';
import 'package:sellers/domain/entity/order.dart';
import 'package:sellers/domain/entity/product.dart';
import 'package:sellers/domain/entity/seller_address.dart';
import 'package:sellers/domain/use_case/account_use_case.dart';
import 'package:sellers/domain/use_case/order_use_case.dart';
import 'package:sellers/domain/use_case/product_use_case.dart';
import 'package:sellers/domain/use_case/seller_use_case.dart';
import 'package:sellers/presentation/cart_screen/product_filter.dart';
import
'package:sellers/presentation/common_widgets/address_widget/address_widget.da
rt';
import
'package:sellers/presentation/common_widgets/alert_dialog/alert_dialog.dart';
import
'package:sellers/presentation/common_widgets/category_header_widget.dart';
import
'package:sellers/presentation/common_widgets/load_indicator/load_indicator_wi
dget.dart';
import
'package:sellers/presentation/common_widgets/product_widget/product_action.da
rt';
import
'package:sellers/presentation/common_widgets/product_widget/product_widget.da
rt';
import
'package:sellers/presentation/common_widgets/smart_list_view/infinite_listvie
w.dart';
import
'package:sellers/presentation/common_widgets/smart_list_view/product_provider

```

```

.dart';
import 'package:sellers/presentation/router/routes.dart';

class CartScreenWidget extends StatefulWidget {
  const CartScreenWidget({super.key});

  @override
  State<CartScreenWidget> createState() => _CartScreenWidgetState();
}

class _CartScreenWidgetState extends State<CartScreenWidget> {
  late final ProductProvider cubit;
  List<String> cartListId = [];
  DeliveryAddress? deliveryAddress;
  int startAfter = 0;

  @override
  void initState() {
    super.initState();
    cubit = ProductProvider((docLimit, {startAfter}) async {
      if (cartListId.isEmpty || cubit.items.isEmpty) {
        cartListId = await AccountUseCase.getCart();
        this.startAfter = 0;
      }
      final products = await ProductUseCase.getConcreteProducts(
        cartListId, docLimit,
        startAfter: this.startAfter);
      this.startAfter += docLimit;
      return (products, null);
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: CupertinoColors.white,
      body: FutureBuilder(
        future:
        SellerUseCase.getSeller(AccountRepository().currentUser!.uid),
        builder: (context, snapshot) {
          if (snapshot.connectionState != ConnectionState.done) {
            return const SizedBox();
          }
          deliveryAddress = snapshot.data?.sellerAddress;
          return BlocBuilder<ProductProvider, ProductState>(
            bloc: cubit,
            builder: (context, state) {
              return InfiniteListView(
                appBarWidget: SliverAppBar(
                  centerTitle: true,
                  collapsedHeight: 40,
                  pinned: true,
                  toolbarHeight: 40,
                  leading: CupertinoButton(
                    padding: EdgeInsets.zero,
                    minSize: 0,
                    child: const Icon(
                      CupertinoIcons.chevron_left,
                      color: CupertinoColors.systemOrange,
                    ),
                    onPressed: () {
                      Navigator.of(context).pop();
                    },
                  ),
                backgroundColor: Colors.transparent,

```

```

        surfaceTintColor: CupertinoColors.systemOrange,
        automaticallyImplyLeading: false,
      ),
      headerWidget: const CategoryHeaderWidget(
        title: 'Cart',
        child: Hero(
          tag: 'cart_screen',
          child: FittedBox(
            child: Icon(
              CupertinoIcons.cart_fill,
              color: CupertinoColors.systemOrange,
            ),
          ),
        ),
      ),
    ),
    provider: cubit,
    builder: (context, index) {
      return ProductWidget(
        product: cubit.items[index],
        heroTag: 'cart_screen',
        onPressed: (product) {
          Navigator.of(context)
            .pushNamed(Routes.productScreen, arguments: [
              product,
              'cart_screen',
              [
                CupertinoButton(
                  minSize: 0,
                  padding: const
EdgeInsets.symmetric(vertical: 5),
                  child: const Text(
                    'Make Order',
                    style: TextStyle(color:
CupertinoColors.link),
                  ),
                  onPressed: () {
                    makeOrder(product);
                  }
                ],
                Column(
                  children: [
                    SelFilterWidget(
                      key: SelFilterWidget.globalKey,
                      filters: product.filters,
                      product: product,
                    ),
                    const SizedBox(height: 20),
                    Container(
                      padding: const EdgeInsets.only(top:
15, left: 15, right: 15, bottom: 5),
                      decoration: BoxDecoration(
                        color:
CupertinoColors.white.withOpacity(0.6),
                        borderRadius:
BorderRadius.circular(16)),
                      child: DeliveryAddressWidget(
                        color:
CupertinoColors.lightBackgroundGray.withOpacity(0.8),
                        onEntered: (address) {
                          deliveryAddress = address;
                        },
                        deliveryAddress: deliveryAddress,
                      ))

```

```

        ],
      ),
    ));
  },
  action: ProductAction(
    isActive: (product) {
      return false;
    },
    iconStateRecord: (
      CupertinoIcons.minus_circle_fill,
      CupertinoIcons.minus_circle_fill
    ),
    onPressed: (product) async {
      await AccountUseCase.toggleToCart(product);
      await cubit.fetchNextItems(refreshList: true);
    });
  },
  mainAxisSize:
    (MediaQuery.of(context).size.width / 2 - 45) + 115,
);
});
}
),
);
}

```

```

Future<void> makeOrder(Product product) async {
  if(deliveryAddress == null){
    showAlertDialog(context, 'Error', 'Delivery address is empty');
    return;
  }
  final addedFilters =
    SelFilterWidget.globalKey.currentState?.cubit.state.addedFilters;
  if (addedFilters == null) return;
  bool isValid = true;
  addedFilters.forEach((key, value) {
    if (value.isEmpty) {
      isValid = false;
      return;
    }
  });
  if (!isValid) {
    showAlertDialog(context, 'Error', 'Some fields is empty');
    return;
  }
  final amount =
    SelFilterWidget.globalKey.currentState?.orderCubit.state.amount;
  try {
    await showLoadIndicator(
      OrderUseCase.newOrder(ProductOrder(
        id: '-1',
        product: product,
        productFilters: addedFilters
          .map((key, value) => MapEntry(key, value[0].value)),
        addresseeId: AccountRepository().currentUser!.uid,
        senderId: product.sellerID,
        status: OrderStatus.waitingForAccept,
        amount: amount ?? 1,
        sellerAddress: deliveryAddress!)),
      context);
    await AccountUseCase.toggleToCart(product);
    Navigator.of(context).pop();
    cubit.fetchNextItems(refreshList: true);
    showAlertDialog(context, 'Success',

```

```

        'New order created! You can check it on your orders!');
    } catch (_) {
        showAlertDialog(context, 'Error', 'Something went wrong');
    }
}
}

```

## Б.6 Текст файла add\_product\_screen.dart

```

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:sellers/dependencies/firebase_service.dart';
import 'package:sellers/domain/cubit/auth/valid_status.dart';
import
'package:sellers/domain/cubit/product_form_screen/product_form_cubit.dart';
import
'package:sellers/domain/cubit/product_form_screen/product_form_state.dart';
import 'package:sellers/domain/entity/product.dart';
import 'package:sellers/presentation/add_product_screen/action_mode.dart';
import
'package:sellers/presentation/add_product_screen/add_filters_screen.dart';
import 'package:sellers/presentation/auth/widgets/auth_text_field.dart';
import
'package:sellers/presentation/common_widgets/buttons/back_button/back_button.
dart';
import
'package:sellers/presentation/common_widgets/load_indicator/load_indicator_wi
dget.dart';
import
'package:sellers/presentation/common_widgets/photo_widgets/photo_widget.dart'
;
import
'package:sellers/presentation/common_widgets/picker_widget/picker_widget.dart
';
import
'package:sellers/presentation/common_widgets/table_view/table_view_widget.dar
t';

class AddProductScreen extends StatefulWidget {
    const AddProductScreen({super.key, this.product});

    final Product? product;

    @override
    State<AddProductScreen> createState() => _AddProductScreenState();
}

class _AddProductScreenState extends State<AddProductScreen> {
    late final ProductFormCubit cubit;

    @override
    void initState() {
        super.initState();
        cubit = ProductFormCubit(ProductFormState(
            product: widget.product ??
                Product(
                    id: '',
                    name: '',
                    description: '',

```



```

        ? null
        : state.product.photo,
width: double.infinity,
height: MediaQuery.of(context).size.width -
40,

bytes: state.photo,
shape: BoxShape.rectangle,
borderRadius: BorderRadius.circular(16),
),
const SizedBox(height: 10),
AuthTextFieldWidget(
  controller: cubit.titleController,
  borderRadius: BorderRadius.circular(16),
  maxLines: 2,
  placeholderText: 'Enter title',
  maxLength: 50,
  prefixIcon: CupertinoIcons.textformat,
  fontSize: 24,
  fontWeight: FontWeight.w600,
  validStatus: state.titleStatus,
),
const SizedBox(height: 10),
AuthTextFieldWidget(
  controller: cubit.descriptionController,
  placeholderText: 'Enter description',
  borderRadius: BorderRadius.circular(16),
  maxLength: 2000,
  fontSize: 20,
  fontWeight: FontWeight.w500,
  maxLines: 7,
  prefixIcon: CupertinoIcons.text_bubble,
  validStatus: state.descriptionStatus,
),
const Padding(
  padding: EdgeInsets.symmetric(
    horizontal: 10, vertical: 5),
  child: Text(
    'Enter price',
    style: TextStyle(
      color:
CupertinoColors.darkBackgroundGray,
      fontWeight: FontWeight.w500,
      fontSize: 24),
    ),
  ),
Row(
  children: [
    Expanded(
      flex: 6,
      child: AuthTextFieldWidget(
        maxLength: 9,
        controller:
cubit.dollarPriceController,
        borderRadius:
BorderRadius.circular(16),
        placeholderText: 'Dollars',
        keyboardType: TextInputType.number,
        prefixIcon:
CupertinoIcons.money_dollar,
        validStatus: state.dollarPriceStatus,
      ),
    ),
    const SizedBox(width: 10),
    Expanded(

```

```

flex: 4,
child: AuthTextFieldWidget(
  maxLength: 2,
  controller: cubit.centPriceController,
  borderRadius:
BorderRadius.circular(16),

  placeholderText: 'Cents',
  keyboardType: TextInputType.number,
  prefixIcon:
    CupertinoIcons.money_dollar_circle,
  validStatus: state.centPriceStatus,
),
),
],
),
if (widget.product == null)
  Padding(
    padding: const EdgeInsets.only(top: 10),
    child: PickerWidget(
      button: Text(
        state.selCategory?.title ??
        'Choose category',
        style: const TextStyle(
          color: CupertinoColors.link),
      ),
      onSelectedItemChanged:
cubit.onSelectData,

      children:
        cubit.categories.map<Widget>((e) {
          return Text(e.title);
        }).toList(),
    ),
  Builder(builder: (context) {
    int filterIterator = -1;
    return Padding(
      padding: const EdgeInsets.only(top: 10),
      child: TableViewWidget(
        title: 'Tags',
        itemHeight: 40,
        maxLength: 20,
        lines: 4,
        addItem: cubit.addFilter,
        deleteItem: cubit.deleteFilter,
        children: [
          ...state.product.tags
            .map<Widget>((text) {
              filterIterator += 1;
              final index = filterIterator;
              return Padding(
                padding: const EdgeInsets.only(
                  left: 20, right: 40),
                child: CupertinoTextField(
                  controller:
TextEditingController(

                    text: text),
                  onChanged: (newText) {
                    cubit.updateFilter(
                      index, newText);
                  },
                  decoration: const
BoxDecoration(),

                  placeholder: 'Enter filter',
                ),
              ),
            ],
        );

```

```

    ))
  ]),
);
)),
const SizedBox(height: 10),
AuthTextFieldWidget(
  maxLength: 100,
  controller: cubit.locationController,
  borderRadius: BorderRadius.circular(16),
  placeholderText: 'Enter location',
  prefixIcon: CupertinoIcons.location,
  validStatus: state.locationStatus),
const SizedBox(height: 10),
CupertinoButton(
  child: const Text(
    'Next',
    style: TextStyle(
      color: CupertinoColors.link,
      fontWeight: FontWeight.w500,
      fontSize: 20),
  ),
  onPressed: () {
    cubit.verifyData(context);
  },
const SizedBox(height: 50)
],
);
)),
),
AddFiltersScreen(productCubit: cubit)
],
),
);
)),
);
}
}

```

## Б.7 Текст файла orders\_screen\_widget.dart

```

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:sellers/domain/cubit/orders_screen/order_screen_cubit.dart';
import 'package:sellers/domain/cubit/orders_screen/order_screen_state.dart';
import
'package:sellers/presentation/common_widgets/dock_widget/dock_widget.dart';
import 'package:sellers/presentation/orders_screen/order_widget.dart';

class OrderStatusWidget extends StatelessWidget {
  const OrderStatusWidget({
    super.key,
    required this.icon,
    required this.title,
    required this.color,
  });

  final IconData icon;
  final String title;
  final Color color;

```

```

@override
Widget build(BuildContext context) {
  return Row(
    children: [
      Icon(
        icon,
        color: color,
        size: 16,
      ),
      const SizedBox(width: 5),
      Text(
        title,
        style: TextStyle(color: color, fontSize: 16),
      )
    ],
  );
}

class OrderScreenWidget extends StatefulWidget {
  const OrderScreenWidget({super.key});

  @override
  State<OrderScreenWidget> createState() => _OrderScreenWidgetState();
}

class _OrderScreenWidgetState extends State<OrderScreenWidget> {
  late final OrderScreenCubit cubit;

  @override
  void initState() {
    super.initState();
    cubit = OrderScreenCubit(OrderScreenState(orders: [], sales: []));
  }

  @override
  void dispose() {
    cubit.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: CupertinoColors.white,
      body: Padding(
        padding: const EdgeInsets.only(top: 70),
        child: SizedBox(
          width: double.infinity,
          child: FutureBuilder(
            future: cubit.init(),
            builder: (context, snapshot) {
              if (snapshot.connectionState != ConnectionState.done) {
                return const Center(child: CupertinoActivityIndicator());
              }
              return Stack(
                children: [
                  Column(
                    crossAxisAlignment: CrossAxisAlignment.center,
                    children: [
                      BlocBuilder<OrderScreenCubit, OrderScreenState>(
                        bloc: cubit,
                        buildWhen: (prev, current) {

```

```

        return prev.page != current.page;
      },
      builder: (context, state) {
        return CupertinoSlidingSegmentedControl(
          children: const {
            'Orders': Text('Orders'),
            'Sales': Text('Sales'),
          },
          groupValue: state.page,
          onValueChanged: cubit.changePage,
        );
      },
    )),
    const SizedBox(height: 20),
    Expanded(
      child: PageView(
        controller: cubit.controller,
        children: [
          BlocBuilder<OrderScreenCubit,
OrderScreenState>(
            bloc: cubit,
            buildWhen: (prev, current) {
              return prev.orders != current.orders;
            },
            builder: (context, state) {
              if (state.orders.isEmpty) {
                return const Center(
                  child: Text('No orders'));
              }
              return ListView.builder(
                padding:
                  const EdgeInsets.only(bottom: 100),
                itemBuilder: (context, index) {
                  return ProductOrderWidget(
                    order: state.orders[index],
                    isOrder: true,
                    cubit: cubit,
                  );
                },
                itemCount: state.orders.length,
              );
            },
          ),
          BlocBuilder<OrderScreenCubit,
OrderScreenState>(
            bloc: cubit,
            buildWhen: (prev, current) {
              return prev.sales != current.sales;
            },
            builder: (context, state) {
              if (snapshot.connectionState !=
                ConnectionState.done) {
                return const
CupertinoActivityIndicator();
              }
              if (state.sales.isEmpty) {
                return const Center(
                  child: Text('No sales'));
              }
              return ListView.builder(
                padding:
                  const EdgeInsets.only(bottom: 100),
                itemBuilder: (context, index) {
                  return ProductOrderWidget(
                    order: state.sales[index],
                    isOrder: false,

```

```

        cubit: cubit,
      );
    },
    itemCount: state.sales.length,
  );
}

],
),
)
],
),
DockWidget(buttons: [
  DockButton(
    icon: CupertinoIcons.house_fill,
    size: 32,
    onPressed: () {
      Navigator.of(context).pop();
    },
  ),
  DockButton(
    icon: CupertinoIcons.arrow_left_right_circle_fill,
    onPressed: () {
      cubit.changePage(null, auto: true);
    },
  ),
  DockButton(
    icon: CupertinoIcons.arrow_clockwise_circle_fill,
    onPressed: () {
      cubit.updateCurrentPage();
    },
  ),
], backgroundColor: CupertinoColors.white),
),
)),
),
),
);
}
}

```

## Б.8 Текст файла favourites\_screen.dart

```

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:sellers/data/repository/account_repository.dart';
import 'package:sellers/data/repository/product_repository.dart';
import 'package:sellers/domain/use_case/account_use_case.dart';
import 'package:sellers/domain/use_case/product_use_case.dart';
import
'package:sellers/presentation/common_widgets/category_header_widget.dart';
import
'package:sellers/presentation/common_widgets/product_widget/product_action.da
rt';
import
'package:sellers/presentation/common_widgets/product_widget/product_widget.da
rt';import
'package:sellers/presentation/common_widgets/smart_list_view/infinite_listvie
w.dart';
import
'package:sellers/presentation/common_widgets/smart_list_view/product_provider
.dart';

```



```

        ),
      ),
    ),
  ),
  provider: cubit,
  builder: (context, index) {
    return ProductWidget(
      product: cubit.items[index],
      heroTag: 'fav_screen',
      action: ProductAction.actions['favs']!);
  },
  mainAxisAlignment:
    (MediaQuery.of(context).size.width / 2 - 45) + 115,
);
  ));
);
}
}

```

## Б.9 Текст файла search\_screen\_widget.dart

```

import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:sellers/data/repository/product_repository.dart';
import 'package:sellers/dependencies/extensions/string_extension.dart';
import 'package:sellers/domain/entity/product.dart';
import 'package:sellers/domain/use_case/product_use_case.dart';
import 'package:sellers/presentation/common_widgets/appbar_widget.dart';
import
'package:sellers/presentation/common_widgets/dock_widget/dock_widget.dart';
import
'package:sellers/presentation/common_widgets/product_widget/product_action.da
rt';
import
'package:sellers/presentation/common_widgets/product_widget/product_widget.da
rt';import 'package:sellers/presentation/common_widgets/search_widget.dart';
import
'package:sellers/presentation/common_widgets/smart_list_view/infinite_listvie
w.dart';
import
'package:sellers/presentation/common_widgets/smart_list_view/item_provider.da
rt';
import
'package:sellers/presentation/common_widgets/smart_list_view/product_provider
.dart';

class SearchScreenWidget extends StatefulWidget {
  const SearchScreenWidget({super.key, required this.searchString});

  final String searchString;

  @override
  State<SearchScreenWidget> createState() => _SearchScreenWidgetState();
}

class _SearchScreenWidgetState extends State<SearchScreenWidget> {
  late final ProductProvider productCubit;
  late final ScrollController controller;

```

```

late final TextEditingController textController;
List<String> searchTokens = [];

@override
void initState() {
  super.initState();
  textController = TextEditingController(text: widget.searchString);
  controller = ScrollController();
  productCubit = ProductProvider(getProducts);
  searchTokens = widget.searchString.tokenize();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: CupertinoColors.white,
    body: Stack(
      children: [
        BlocBuilder<ItemProvider<Product, ProductState>, ProductState>(
          bloc: productCubit,
          builder: (context, state) {
            return InfiniteListView(
              provider: productCubit,
              controller: controller,
              appBarWidget: const SliverAppBar(
                automaticallyImplyLeading: false,
                pinned: true,
                collapsedHeight: 80,
                flexibleSpace: AppBarWidget(
                  title: 'Search',
                  action: SizedBox(),
                ),
              ),
              headerWidget: Padding(
                padding: const EdgeInsets.only(bottom: 20),
                child: Hero(
                  tag: 'search_tag',
                  child: SearchWidget(
                    onSubmitted: (str) {
                      searchTokens = str.tokenize();
                      productCubit.fetchNextItems(refreshList: true);
                    },
                    controller: textController,
                  ),
                ),
              ),
              builder: (context, index) {
                return ProductWidget(
                  product: productCubit.items[index],
                  heroTag: 'search',
                  action: ProductAction.actions['favs']!,
                );
              },
            ),
            mainAxisAlignment:
              MainAxisAlignment.spaceBetween,
            ),
          ),
        DockWidget(buttons: [
          DockButton(
            icon: CupertinoIcons.house_fill,
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
          DockButton(

```

```

        icon: CupertinoIcons.arrow_up_circle_fill,
        onPressed: () {
          controller.animateTo(controller.initialScrollOffset,
            duration: const Duration(seconds: 1),
            curve: Curves.fastOutSlowIn);
        })),
      DockButton(
        icon: CupertinoIcons.line_horizontal_3_decrease,
        onPressed: () {}),
    ], backgroundColor: CupertinoColors.white)
  ],
),
);
}

Future<(List<Product>, QueryDocumentSnapshot<Map<String, dynamic>>?)>
  getProducts(docLimit, {startAfter}) async {
    final List<Product> searchedProducts = [];
    QueryDocumentSnapshot<Map<String, dynamic>>? last;
    while (true) {
      bool needToBreak = false;
      final snap = await ProductUseCase
        .getSearchedProducts(docLimit, searchTokens, startAfter:
startAfter);
      final products =
        ProductUseCase.snapToProducts(snap.docs, withSearchTokens: true);
      last = snap.docs.lastOrNull;
      if (products.length < docLimit) {
        needToBreak = true;
      }
      for (var searchToken in searchTokens) {
        products.removeWhere(
          (prod) => !(prod.searchTokens?.contains(searchToken) ?? false));
      }
      searchedProducts.addAll(products);
      if (searchedProducts.length >= docLimit || needToBreak) {
        break;
      }
    }
    return (searchedProducts, last);
  }
}

```

## Б.10 Текст файла chat\_screen.dart

```

import 'dart:async';

import 'package:chat_bubbles/chat_bubbles.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:image_picker/image_picker.dart';
import 'package:sellers/dependencies/firebase_service.dart';
import 'package:sellers/domain/cubit/chat_screen/private_chat_cubit.dart';
import 'package:sellers/domain/cubit/chat_screen/private_chat_state.dart';
import 'package:sellers/domain/entity/chat/chat.dart';
import 'package:sellers/domain/entity/chat/message.dart';
import 'package:sellers/domain/entity/chat/user.dart';
import
'package:sellers/presentation/chat_screen/message_widgets/image_message.dart'

```

```

;
import
'package:sellers/presentation/chat_screen/message_widgets/video_message.dart'
;
import
'package:sellers/presentation/common_widgets/app_picker/app_picker_widget.dar
t';
import
'package:sellers/presentation/common_widgets/buttons/back_button/back_button.
dart';
import
'package:sellers/presentation/common_widgets/dock_widget/dock_widget.dart';
import
'package:sellers/presentation/common_widgets/load_indicator/load_indicator_wi
dget.dart';
import
'package:sellers/presentation/common_widgets/photo_widgets/photo_widget.dart'
;

class ChatScreenWidget extends StatefulWidget {
  const ChatScreenWidget({super.key, required this.chat, required
this.user});

  final Chat chat;
  final ChatUser user;

  @override
  State<ChatScreenWidget> createState() => _ChatScreenWidgetState();
}

class _ChatScreenWidgetState extends State<ChatScreenWidget> {
  late final PrivateChatCubit cubit;
  late final ScrollController controller;

  @override
  void initState() {
    super.initState();
    controller = ScrollController()..addListener(scrollListener);
    cubit = PrivateChatCubit(
      PrivateChatState(chat: widget.chat, user: widget.user));
  }

  @override
  void dispose() {
    cubit.dispose();
    controller.dispose();
    super.dispose();
  }

  void scrollListener() {
    if (controller.offset >= controller.position.maxScrollExtent / 2 &&
        !controller.position.outOfRange) {
      cubit.fetchMessages();
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      extendBodyBehindAppBar: true,
      backgroundColor: CupertinoColors.white,
      appBar: AppBar(
        leading: const BackButtonWidget(),
        backgroundColor: CupertinoColors.white,

```

```

leadingWidth: 70,
toolbarHeight: 70,
centerTitle: true,
title: Row(
  children: [
    const Spacer(),
    BlocBuilder<PrivateChatCubit, PrivateChatState>(
      bloc: cubit,
      buildWhen: (prev, current) {
        if (prev.user != current.user) {
          return true;
        }
        return false;
      },
      builder: (context, state) {
        final sellerStatus =
          SellerStatus.getStatus(state.user, state.chat);
        return Column(
          children: [
            Text(
              state.user.name,
              style: const TextStyle(
                fontWeight: FontWeight.w700,
                fontSize: 24,
                height: 1.1),
            ),
            Text(
              sellerStatus.text,
              style: TextStyle(
                color: sellerStatus.color,
                fontSize: 13,
                fontWeight: FontWeight.w400),
            ),
          ],
        );
      },
    ),
    const Spacer(),
    SizedBox(
      width: 70,
      child: Row(
        children: [
          const Spacer(),
          PhotoWidget(
            width: 40,
            height: 40,
            shape: BoxShape.circle,
            url: widget.user.photoUrl,
          ),
        ],
      ),
    ),
  ],
),
),
body: Stack(
  alignment: Alignment.bottomCenter,
  fit: StackFit.expand,
  children: [
    FutureBuilder(
      future: cubit.initData(),
      builder: (context, snapshot) {
        if (ConnectionState.done != snapshot.connectionState) {
          return const LoadIndicatorWidget();
        }
      }
    )
  ]
)

```

```
return BlocBuilder<PrivateChatCubit, PrivateChatState>(
    bloc: cubit,
    builder: (context, state) {
      return ListView.builder(
        reverse: true,
        controller: controller,
        itemCount: state.chat.messages.length,
        itemBuilder: (context, index) {
          if (index == 0) {
            Future.delayed(
              const Duration(milliseconds: 200), () {});
            return Column(
              children: [
                buildMessage(
                  state.chat.messages.length - 1 - index),
                const SizedBox(
                  height: 100,
                ),
              ],
            );
          }
          return buildMessage(
            state.chat.messages.length - 1 - index);
        },
      );
    }),
Align(
  alignment: Alignment.bottomCenter,
  child: DockWidget(
    padding: const EdgeInsets.symmetric(vertical: 5),
    buttons: [
      Expanded(
        child: SizedBox(
          height: 36,
          child: Row(
            children: [
              CupertinoButton(
                onPressed: () async {
                  showAppPicker(
                    context: context,
                    title: 'Pick media',
                    maxSelectionLength: 1,
                    items: ['image', 'video'],
                    selectedItem: null,
                    onSelectItem: (item) async {
                      switch (item) {
                        case 'image':
                          {
                            final media = await ImagePicker()
                              .pickImage(
                                source:
                                  ImageSource.gallery);

                            if (media == null) {
                              return;
                            }
                            cubit.onAddMediaClick(
                              media, ChatMessageType.image);
                          }
                        case 'video':
                          {
                            final media = await ImagePicker()
                              .pickVideo(
                                source:
```



```

bool isSender =
    message.authorId == FirebaseService().currentUser!.uid;
if (!isSender) {
    cubit.onCheckMessage(message);
}
return BubbleSpecialThree(
    text: message.text,
    tail: tail,
    color: isSender
        ? CupertinoColors.lightBackgroundGray
        : CupertinoColors.link.withOpacity(0.8),
    delivered:
        isSender ? ChatMessageStatus.checked == message.status :
false,
    sent: isSender ? ChatMessageStatus.sent == message.status :
false,
    isSender: isSender,
    textStyle: TextStyle(
        color: isSender
            ? CupertinoColors.darkBackgroundGray
            : CupertinoColors.white,
        fontSize: 18),
);
}
case ChatMessageType.image:
{
    bool isSender =
        message.authorId == FirebaseService().currentUser!.uid;
    if (!isSender) {
        cubit.onCheckMessage(message);
    }
    return BubbleNormalImage(
        tail: tail,
        color: isSender
            ? CupertinoColors.lightBackgroundGray
            : CupertinoColors.link.withOpacity(0.8),
        delivered:
            isSender ? ChatMessageStatus.checked == message.status :
false,
        sent: isSender ? ChatMessageStatus.sent == message.status :
false,
        isSender: isSender,
        id: message.id,
        image: ImageMessageWidget(url: message.text),
    );
}
case ChatMessageType.video:
{
    bool isSender =
        message.authorId == FirebaseService().currentUser!.uid;
    if (!isSender) {
        cubit.onCheckMessage(message);
    }
    return BubbleNormalImage(
        tail: tail,
        color: isSender
            ? CupertinoColors.lightBackgroundGray
            : CupertinoColors.link.withOpacity(0.8),
        delivered:
            isSender ? ChatMessageStatus.checked == message.status :
false,
        sent: isSender ? ChatMessageStatus.sent == message.status :
false,
        isSender: isSender,

```

```
        id: message.id,  
        image: VideoMessageWidget(url: message.text),  
    );  
}  
}  
}
```

**ДОДАТОК В**  
**Слайди презентації**



Рисунок В.1 – Слайд 1



Рисунок В.2 – Слайд 2

## ПОРІВНЯННЯ ІСНУЮЧИХ АНАЛОГІВ

| Критерій                                      | Amazon | OLX | AliExpress | Sellers |
|-----------------------------------------------|--------|-----|------------|---------|
| Кросплатформність (iOS/Android)               | +      | +   | +          | +       |
| Єдиний аккаунт для продажу та покупок товарів | -      | +   | -          | +       |
| Сортування товарів                            | -      | +   | +          | +       |
| Фільтрація товарів                            | +      | +   | +          | +       |
| Інтуїтивний простий інтерфейс                 | -      | +   | -          | +       |
| Кошик товарів                                 | +      | -   | +          | +       |
| Категорії товарів                             | +      | +   | +          | +       |
| Категорія улюблених товарів                   | +      | +   | +          | +       |
| Категорії недавно відвіданих товарів          | +      | +   | +          | +       |
| Персональні рекомендації товарів              | +      | +   | +          | +       |
| Вбудований чат                                | -      | +   | +          | +       |

3

Рисунок В.3 – Слайд 3

## ПОСТАНОВКА ЗАВДАННЯ

- Створення облікового запису та авторизація;
- Авторизація за номером телефону;
- Редагувати свої особисті дані (ім'я, електронну пошту, телефон для входу, телефони для зв'язку, адрес доставки);
- Сортування товарів;
- Фільтрація товарів;
- Перегляд недавно відвіданих товарів;
- Додавання товарів до списку улюблених товарів;
- Додавання товарів у кошик;
- Перегляд товарів за категоріями;
- Можливість створення та видалення або редагування створених товарів
- Можливість переглядати інформацію про рейтинг продавця, його телефони, інші товари від нього;
- Можливість замовлення товару, підтвердження його доставки або відміни;
- Можливість приймати або відхиляти замовлення, та зміню його статусу;
- Чатування користувачів.

4

Рисунок В.4 – Слайд 4

## ПОРІВНЯННЯ ФРЕЙМВОРКІВ

| Критерій                               | Flutter      | React Native | Xamarin                  |
|----------------------------------------|--------------|--------------|--------------------------|
| Підтримка Android та iOS платформ      | +            | +            | +                        |
| Ефективність додатку                   | Найвища      | Висока       | Непогана                 |
| Вартість                               | Безкоштовний | Безкоштовний | Підписка                 |
| Велика спільнота та підтримка оновлень | +            | +            | +                        |
| Споживання ресурсів                    | Мінімальне   | Ефективне    | Залежить від оптимізації |

5

Рисунок В.5 – Слайд 5

## ПОРІВНЯННЯ СЕРЕДОВИЩ РОЗРОБКИ

| Критерій                                       | Android Studio | Visual Studio Code | IntelliJ Idea |
|------------------------------------------------|----------------|--------------------|---------------|
| Підтримка системи контролю версій              | +              | +                  | +             |
| Підтримка автоматизованого виправлення помилок | +              | -                  | +             |
| Безкоштовна                                    | +              | +                  | -             |
| Вбудований Android емулятор                    | +              | -                  | -             |
| Підтримка шніпетів                             | +              | +                  | +             |
| Вбудований відлагоджувач                       | +              | +                  | -             |
| Ресурсомікість                                 | Висока         | Мінімальна         | Середня       |
| Оновлення версій                               | +              | +                  | +             |

6

Рисунок В.6 – Слайд 6

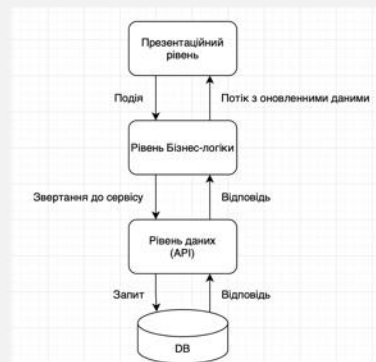
## ПОРІВНЯННЯ ТИПІВ СХОВИЩ

| Критерії           | Реляційні бази даних                              | Нереляційні бази даних                                                |
|--------------------|---------------------------------------------------|-----------------------------------------------------------------------|
| Модель даних       | Таблична                                          | Різноманітна (документи, ключ-значення, колонки, граfi)               |
| Мова запитів       | SQL                                               | Відрізняється залежно від конкретної бази даних                       |
| Зв'язки між даними | За допомогою ключів                               | Відсутні або мають інший механізм (наприклад, посилання на документи) |
| Схема даних        | Суворі, строго визначені                          | Гнучка, може змінюватися з часом без перестановки бази даних          |
| Масштабованість    | Менш гнучкі, складніше масштабувати горизонтально | Більш гнучкі, краще масштабуються горизонтально                       |
| Швидкість доступу  | Зазвичай менша, особливо на великих обсягах даних | Зазвичай швидша, особливо для операцій з ключем                       |
| Вартість           | Зазвичай дорожче, особливо для комерційних систем | Зазвичай дешевші, багато відкритих джерел                             |
| Приклади           | MySQL, PostgreSQL, Oracle                         | MongoDB, Firebase, Redis                                              |

7

Рисунок В.7 – Слайд 7

## АРХІТЕКТУРА ЗАСТОСУНКУ



8

Рисунок В.8 – Слайд 8

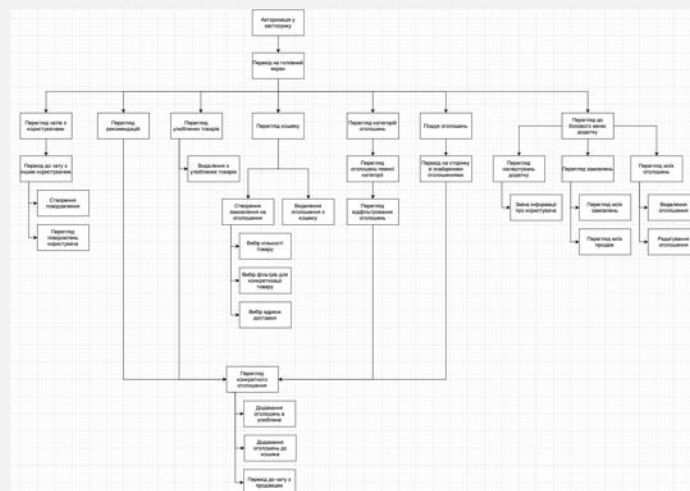
## ПОРІВНЯННЯ СТЕЙТ МЕНЕДЖЕРІВ

| Критерії                                  | Cubit  | Redux    | Provider | GetX   |
|-------------------------------------------|--------|----------|----------|--------|
| Простота використання                     | Висока | Середня  | Непогана | Висока |
| Розділення бізнес-логіки та представлення | Висока | Середня  | Немає    | Висока |
| Швидкодія                                 | Висока | Непогана | Непогана | Висока |
| Масштабованість                           | Висока | Середня  | Непогана | Висока |
| Можливість робити локальні оновлення      | +      | -        | -        | +      |

9

Рисунок В.9 – Слайд 9

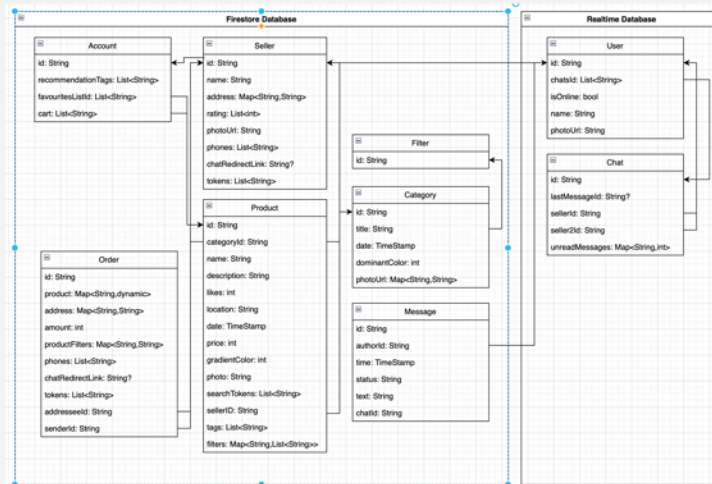
## ФУНКЦІОНАЛЬНА СХЕМА ПРОГРАМИ



10

Рисунок В.10 – Слайд 10

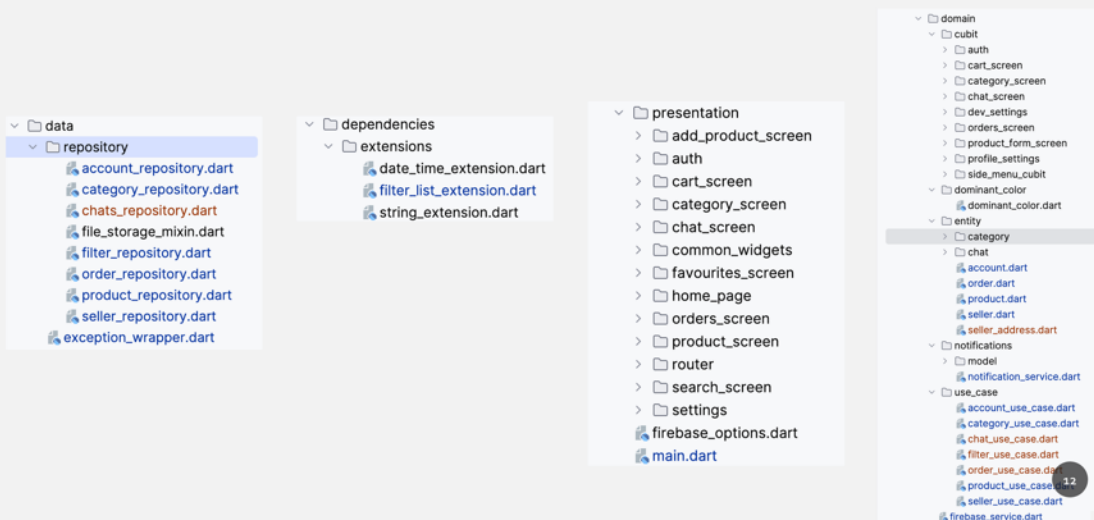
## СХЕМА БД ЗІ ЗВ'ЯЗКАМИ МІЖ СУТНОСТЯМИ



11

Рисунок В.11 – Слайд 11

## СТРУКТУРА ЗАСТОСУНКУ



12

Рисунок В.12 – Слайд 12

## ІНТЕРФЕЙСИ ЗАСТОСУНКУ



13

Рисунок В.13 – Слайд 13

## ІНТЕРФЕЙСИ ЗАСТОСУНКУ



14

Рисунок В.14 – Слайд 14

| ТЕСТУВАННЯ ЗАСТОСУНКУ                                            |                          |                           |
|------------------------------------------------------------------|--------------------------|---------------------------|
| Критерій перевірки                                               | iPhone 15 pro (iOS 17.4) | Pixel. 8 pro (Android 14) |
| Перевірити коректне відображення сторінок згідно дизайну         | +                        | +                         |
| Перевірити наявність усіх сторінок                               | +                        | +                         |
| Перевірити коректне відображення шрифтів та стилів тексту        | +                        | +                         |
| Перевірити коректне відображення кнопок, блоків меню, тощо       | +                        | +                         |
| Перевірити коректність роботи анімацій та переходів              | +                        | +                         |
| Перевірити орфографію та граматику застосунку                    | +                        | +                         |
| Перевірити коректне відображення бокових меню                    | +                        | +                         |
| Перевірити коректне відображення застосованих фільтрів оголошень | +                        | +                         |
| Перевірити коректне відображення модальних вікон                 | +                        | +                         |

| Критерій перевірки                                                                        | iPhone 15 pro (iOS 17.4) | Pixel. 8 pro (Android 14) |
|-------------------------------------------------------------------------------------------|--------------------------|---------------------------|
| Перевірити реєстрацію системи                                                             | +                        | +                         |
| Перевірити авторизацію системи                                                            | +                        | +                         |
| Перевірити авторизацію системи за смс                                                     | +                        | +                         |
| Перевірити створення оголошення                                                           | +                        | +                         |
| Перевірити можливість вводу у поля конструктору оголошень                                 | +                        | +                         |
| Перевірити можливість вибрати фільтри товару у конструкторі оголошень                     | +                        | +                         |
| Перевірити повідомлення про помилку при невірному введенні даних у конструкторі оголошень | +                        | +                         |
| Перевірити видалення створеного оголошення                                                | +                        | +                         |
| Перевірити редагування створеного оголошення                                              | +                        | +                         |
| Перевірити занесення товарів в улюблені                                                   | +                        | +                         |
| Перевірити вилучення товарів з улюблених                                                  | +                        | +                         |
| Перевірити занесення товарів до кошика                                                    | +                        | +                         |
| Перевірити вилучення товарів з кошика                                                     | +                        | +                         |
| Перевірити створення замовлення товару                                                    | +                        | +                         |
| Перевірити встановлення необхідних фільтрів товару у конструкторі замовлення              | +                        | +                         |
| Перевірити встановлення кількості товару у замовленні                                     | +                        | +                         |
| Перевірити вивід адреси доставки у конструкторі повідомлення                              | +                        | +                         |
| Перевірити фільтрацію товарів за категорією                                               | +                        | +                         |
| Перевірити можливість додавання фільтрів у категорії товарів                              | +                        | +                         |
| Перевірити можливість вилучення фільтрів у категорії товарів                              | +                        | +                         |
| Перевірити роботу фільтрації товарів за встановленими фільтрами                           | +                        | +                         |
| Перевірити роботу пошуку товарів                                                          | +                        | +                         |
| Перевірити зміну даних профілю користувача                                                | +                        | +                         |
| Перевірити створення посилання на айдентичний чат                                         | +                        | +                         |
| Перевірка створення адреси доставки за замовченням                                        | +                        | +                         |
| Перевірити сортування товарів                                                             | +                        | +                         |
| Перевірити можливість створення чату з користувачем                                       | +                        | +                         |
| Перевірити відправку повідомлення користувачу                                             | +                        | +                         |

Рисунок В.15 – Слайд 15

| ВИСНОВКИ                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <p>Під час виконання дипломної роботи було проаналізовано предметну область, тобто онлайн маркетплейси. Розібрано найпопулярніші аналоги програм, таких як «Amazon», «OLX» та «AliExpress», та порівняно їх функціональні можливості з функціональними можливостями «Sellers». Порівняльний аналіз показав що розроблюваний застосунок є актуальним та доцільним на фоні конкурентів.</p> <p>При проєктуванні застосунку було обрано фреймворк Flutter як найбільш простий та ефективний спосіб зробити кросплатформний мобільний застосунок. Було обрано архітектуру застосунку Clean Architecture як найбільш зручний в масштабуванні варіант для такого великого застосунку. Також було порівняно середовища розробки, та за низкою переваг було обрано Android Studio.</p> <p>Для збереження даних було обрано хмарне сховище Firebase від компанії Google. Було проаналізовано два можливі варіанти БД та обрано обидві для певних задач.</p> <p>Було надано схему функціональних можливостей та наведено опис основних алгоритмів застосунку. Після цього визначені сутності для БД та також створено схему зв'язків сутностей між собою.</p> <p>Розроблено користувацький інтерфейс, після цього розглянуто всі складові та їх призначення.</p> <p>Після всіх етапів, було створено програмний продукт «Sellers» який дозволяє переглядати оголошення інших користувачів, створювати власні, сортувати та фільтрувати оголошення, створювати замовлення та чатуватись з користувачами за потреби.</p> |  |

Рисунок В.16 – Слайд 16

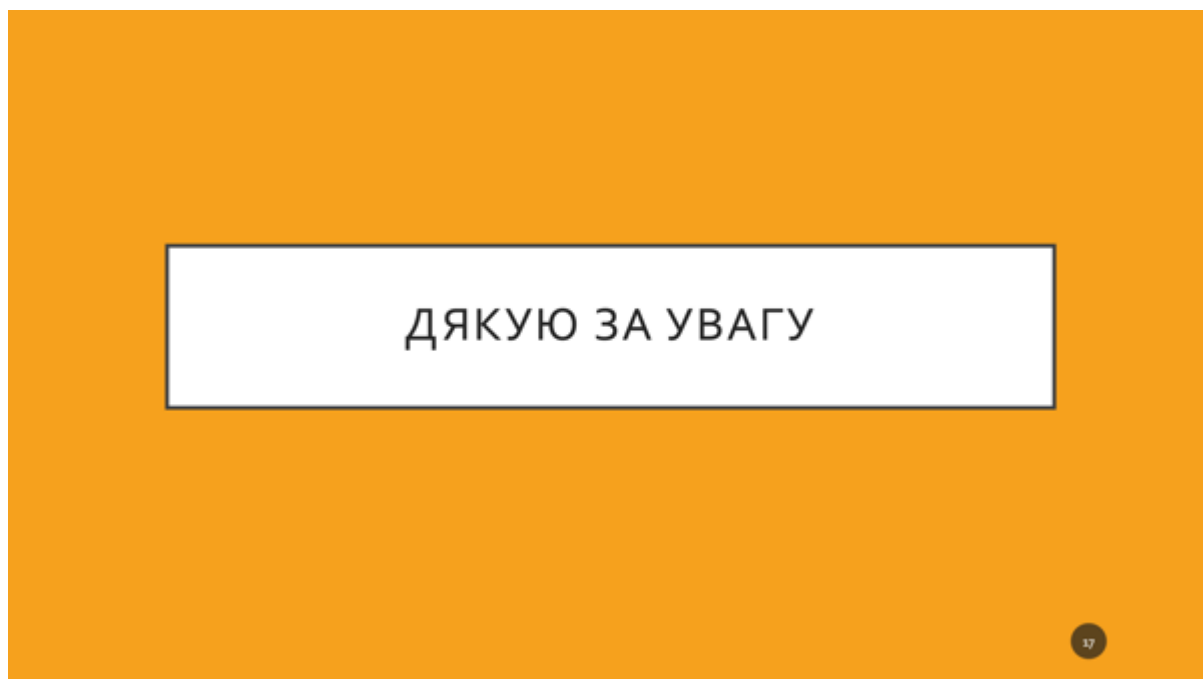


Рисунок В.17 – Слайд 17