Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: **Розроблення мобільного застосунку для інтернет магазину, засобами фреймворку Flutter**

Виконав: студент 2 курсу, групи ІСТС-21
спеціальності: 126 – "Інформаційні системи та технології"
(шифр і назва напряму підготовки, спеціальності)
Блакита Р. В.
(прізвище та ініціали)

Керівник: Прусак Ю. В.
(прізвище та ініціали)

Рецензент: Мокрицька О. В.
(прізвище та ініціали)

Львів – 2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра інформаційних систем та комп'ютерного моделювання

Рівень вищої освіти перший (бакалаврський)

Спеціальність 126 "Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сторожук О. Л.
"07" 02 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Блакита Роман Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи: **Розроблення мобільного застосунку для інтернет магазину, засобами фреймворку Flutter**

керівник роботи: Прусак Ю. В., канд. техн. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 06.02.2024 року № С-87

2. Термін подання студентом роботи 10.06.2024 р.

3. Вихідні дані до роботи:
- вивчити предметну область, проаналізувати існуючі засоби та технології;
- розглянути і використати підходящі алгоритми для роботи компонентів та системи;
- розробити мобільний застосунок для інтернет магазину, з допомогою фреймворку Flutter;
- представити результати роботи програмного застосунку.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):
Розділ 1. Стан проблемної області
Розділ 2. Інформаційне та математичне забезпечення
Розділ 3. Програмне та технічне забезпечення

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

6. Дата видачі завдання 07 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел	10.02-10.03.2024	виконано
2	Розділ 1. Стан проблемної області	11.03-30.03.2024	виконано
3	Розділ 2. Інформаційне та математичне забезпечення	05.04-25.04.2024	виконано
4	Розділ 3. Програмне та технічне забезпечення	27.04-30.05.2024	виконано
5	Аналіз отриманих результатів та написання висновків. Оформлення дипломної роботи.	01.06-10.06.2024	виконано
6	Здача пояснювальної записки на перевірку керівнику, виправлення помилок та здача роботи рецензенту.	10.06-14.06.2024	виконано

Студент _____ Блакита Р.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Прусак Ю.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 43 сторінок пояснювальної записки, 27 рисунків, 19 літературних джерел, додатки.

В дипломній роботі реалізовано мобільний застосунок з продажу товарів мовою програмування Dart, засобами фреймворку Flutter. Проведено аналіз існуючих додатків та їх вплив на продажі за допомогою зручного і простого інтерфейсу. За допомогою інструментів онлайн сервісу Figma, розроблено графічний інтерфейс застосунку. Результати виконаної роботи представленні ілюстративним матеріалом, а сам застосунок готовий до подальшого використання.

Ключові слова: Flutter, Figma, Android, Firebase, база даних, програмний продукт, мобільний застосунок, інтерфейс користувача.

ABSTRACT

The qualification work contains 43 pages, 27 illustrations, 19 literary sources, appendices.

In the work, a mobile application was implemented for the sale of goods in the programming language – Dart, with using the framework Flutter. An analysis of existing applications and their impact on sales was carried out using a convenient and simple interface. With the help of the tools of the online service Figma, the graphical interface of the application was developed. The results are presented of the work performed in illustrative material, and the application itself is ready for use.

Keywords: Flutter, Figma, Android, Firebase, database, software product, mobile application, user interface.

ТЕХНІЧНЕ ЗАВДАННЯ

В дипломній роботі потрібно вирішити наступні завдання:

- проаналізувати методи, алгоритми та засоби розв’язання задачі;
- підібрати засоби та інструменти для розробки застосунку;
- створити структурну схему системи з врахуванням інформаційних потоків;
- розробити графічний інтерфейс застосунку;
- розробити програмне і технічне рішення;
- представити результати роботи мобільного застосунку.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

SDK – Software Development Kit

API – Application Programming Interface

UI – User Interface

UX – User Experience

UML – Unified Modeling Language

APP – Application

ОС – Операційна система

БД – База даних

ВСТУП

Завдяки розвитку цифрових технологій, на даний час Україна пришвидшилась у розвитку цифровій трансформації. З кожним днем державні та приватні установи нашої країни переходять на системи цифровізованого способу ведення бізнесу. Відсутність достатньої кількості ресурсів, на жаль, сповільнюють наші дії, які допомагають використати такі необхідні технології. У даній роботі, прикладом такої галузі буде система автоматизації продажів.

Крім того, ще є значна кількість проблем, через які стикаються і покупці, наприклад, при покупці взуття. Певним чином це пов’язане з тим, що для покупки товару клієнти змушені щоразу повторно шукати підходяще взуття, не тільки за розміром чи сезоном, а й за цільовим застосуванням, наприклад для зайнять спортом, хайкінгу тощо. Це веде до збільшення витрати часу з пошуку потрібного взуття.

З кожним роком, клієнт-серверні програми використовуються все частіше. Їх застосовують майже у галузях для того, щоб спростити та оптимізувати роботу. Наприклад, у Play Market на даний момент майже 3 млн. різноманітних застосунків, від розважальних до соціальних або бізнес. І цей ринок стрімко розвивається. Мобільні застосунки – це зручно, швидко і завжди під рукою. Фактично, кожний популярний портал, маркетплейс чи програма мають свою мобільну версію.

Саме тому, ми й поставили собі завдання, створити мобільний застосунок для пошуку і покупки товарів. Але, звісно, це дуже великий масив даних, тому ми обмежились взуттям. Застосунок покликаний допомогти вирішити вище наведену проблему. Також, дозволить продавцям ефективніше реалізувати взуття цільовим клієнтам. У свою чергу, клієнт має змогу віддалено вибрати необхідну пару взуття, що дозволить значно заощадити час та сили під час пошуку, а також отримати доступ до всього асортименту товарів не виходячи з дому.

Завдяки розробленому застосунку, я маю намір полегшити комунікацію між покупцем та продавцем, приблизити галузь продажі з фізичного до цифровізованого магазину.

Метою роботи є розроблення мобільного застосунку онлайн магазину з продажу взуття.

Об'єктом дослідження є середовище для управління товаром і автоматизації процесу продажів.

Предметом дослідження є проєктування інформаційної системи автоматизації роботи продавця.

Практичне значення роботи полягає у забезпеченні простоти взаємодії і комфорту роботи для кінцевого користувача.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

Розробка мобільних додатків – одна з найбільш розповсюджених та актуальних задач в IT-галузі на сьогодні і стрімко зростає. Згідно з дослідженнями [11], у 2024 році більше 62% населення землі активно використовують мобільні пристрої. Завдяки значному ривку у розвитку інтернету, його доступність і висока швидкість дають можливість величезній кількості людей його використовувати у власних інтересах, переглядати фото і відео, здійснювати онлайн покупки.

1.1. Огляд засобів для розробки мобільного застосунку.

За статистикою [12], на перший квартал 2024 року більше 99% користувачів використовують ОС iOS та Android на своїх смартфонах, менше 1% – інші операційні системи. Але щоб не витрачати надмірний ресурс, необхідно розробляти програми, які можна завантажити на кожен пристрій.

Кросплатформова розробка мобільних програм – реалізація програм, де пишуть один програмний код і він буде використовувати зразу на декількох операційних системах, наприклад iOS і Android. Для такої задачі ще використовують якийсь фреймворк – Flutter, Xamarin або React Native.

Даний принцип побудови є у кожній операційні системі і має свій SDK. SDK – це комплект, з допомогою якого буде розроблятися програмне забезпечення. Завдяки ньому можна реалізувати нативну програму. Він має компілятор та інші інструменти. Кожен має якусь свою рекомендовану мову програмування. Для Android використовують Kotlin або Java, для iOS – найкраще підійдуть Objective-C та Swift.

Для розробки застосовують API – це інтерфейс прикладного програмування. Він з’єднує програмний код та ОС.

Для самої розробки застосовується API – інтерфейс прикладного програмування. Він є сполучною складовою для коду та ОС. Ще, використовують IDE – це інтегроване середовище розробки.

Програму буде неможливо реалізувати без фреймворків (англ. Framework). Наприклад, Flutter, Xamarin або React Native – полегшують процес кросплатформної розробки. Усі вони мають свої можливості та переваги. перед тим як вибрати конкретний фреймворк, потрібно оцінити властивості майбутньої програми, але часто розробники беруть один найоптимальніший для них інструмент та використовують його у всіх проєктах.

Проаналізувавши низку популярних фреймворків ми зупинились на Flutter. Зокрема тому, що це фреймворк з відкритим кодом, для створення кросплатформних додатків, розроблений компанією Google в 2014 році. Він дозволяє розробникам створювати високоякісні нативні інтерфейси для мобільних (iOS і Android), веб- та настільних застосунків з використанням єдиної бази коду.

Рисунок 1.1 – Логотип програмного каркасу (фреймворку)
із відкритим кодом для створення додатків – Flutter

Flutter використовує мову програмування – Dart. Flutter SDK – набір інструментів, включаючи компілятор Dart, інструменти для розробки та бібліотеки віджетів. В цьому фреймвокру всі компоненти інтерфейсу користувача є віджетами, що дозволяє легко налаштовувати та створювати складні UI.

Порівнюючи Flutter з іншими мовами та фреймворками можемо виділити наступні сильні сторони [13]:

- Єдина база коду. Один код для мобільних, веб та настільних додатків;
- Глибока інтеграція з Android. Всі API Android доступні, що безпосередньо, дозволяє повністю використовувати можливості платформи;
- Велика спільнота. Багато ресурсів, бібліотек і підтримки від Google та спільноти розробників;

- Стабільність і продуктивність. Висока продуктивність і стабільність завдяки нативному виконанню коду;
- Крос-платформність: Можливість створювати веб та настільні додатки разом з мобільними;
- Готові віджети. Величезна бібліотека готових до використання віджетів, що дозволяє швидко створювати інтерфейси.

Також, можемо виділити певні недоліки:

- Підтримка різних платформ. Необхідність писати і підтримувати окремий код для Android і iOS;
- Тривала розробка. Більше часу на розробку та тестування, особливо для мультиплатформних застосунків.

Отже, можемо констатувати, що Flutter є сучасним і потужним інструментом для розробки кросплатформних додатків завдяки своїй продуктивності, зручності у використанні та єдиній базі коду для різних платформ. Він забезпечує швидший цикл розробки, високоякісну візуалізацію і можливість використовувати одну кодову базу для кількох платформ, що робить його привабливим вибором для сучасних розробників.

1.2. Огляд засобів для розробки дизайну застосунку.

У сучасному світі дизайн відіграє величезну роль. Це стосується й створення застосунків. Дизайн є важливим учасником процесу розробки, і для успішного результату розробники зобов’язані створити привабливий вигляд продукту і забезпечити вдалу користувацьку взаємодію.

Для розробки дизайну мобільних і веб-застосунків є велика кількість інструментів. Наведемо кілька популярних, які дозволяють створювати високоякісні UI/UX макети та інтерактивні прототипи: InVision Studio, Figma, Adobe XD, Sketch, Zeplin. Усі вони мають свої недоліки та переваги. Наприклад, у Sketch є велика кількість інтеграцій та плагінів із сторонніми інструментами,

Adobe XD надає можливість працювати з макетами у режимі кооперативної роботи, а у InVision Studio є можливості створити анімації та

інтерактивні зразки. Однак, Figma є лідером не тільки для кооперативної роботи, а і у доступності, завдяки чому її використовують команди, які працюють віддалено.

Рисунок 1.2 – Логотип онлайн-сервісу розробки інтерфейсів та прототипування – Figma

Figma – це онлайн-редактор, який використовують у створенні дизайну макетів, прототипів інтерфейсу користувача (UI) та його досвіду (UX). Це класний інструмент, з допомогою якого дизайнери працюють у режимі реального часу, де спільно редагують та діляться своїми проєктами у хмарному середовищі [14]. Вона стала важливим інструментом для великої частини фахівців у сфері графічного дизайну, а також для команд, які розробляють веб-інтерфейси і мобільні додатки. Ця програма отримала велику кількість прихильників завдяки особливим можливостям які у ній є:

- Кооперативна робота у реальному часі. У Figma можна дизайнерам спільно працювати над роботою у онлайн. Це дуже добре для команд, які працюють у різних містах або віддалено. Всі зміни миттєво застосовуються, завдяки чому значно ефективніше виконується спільна робота;
- Доступність. У Figma не треба використовувати спеціальне програмне забезпечення і працює у веб-браузері. Завдяки цьому він є легкодоступним для багатьох користувачів з різних пристроїв та ОС. Хоча, інколи це може бути недоліком, бо без доступу до Інтернет додаток не працюватиме;
- Створення прототипів та макетів. Figma дає можливості створити макет і прототип у одному інструменті, що полегшує процес розробки та

тестування інтерфейсу для користувача, що відіграє важливу роль у етапі розробки графічного дизайну;

- Робота з векторною та растровою графікою. Figma надає підтримку у роботі з растровою графікою, що дозволяє йому бути універсальним і корисним для робітників, які працюють зі різнотипними зображеннями;
- Створення бібліотек та компонентів. Figma надає можливість для створення дизайнерам власних бібліотек та компонентів, які у майбутньому можна повторно застосувати у різних проєктах. Це значно збільшує ефективність розробки дизайну та сприяє створенню одного стилю у всіх проєктах бренда або компанії.

Також, беззаперечною перевагою і необхідною умовою взаємодії Figma та Flutter є можливість експортувати дизайн у Flutter-код за допомогою плагіна конвертації «Figma to Flutter».

Отже, для розробки дизайну додатків на Flutter можна використовувати різні інструменти, які пропонують потужні можливості для створення високоякісних UI/UX макетів та інтерактивних прототипів, ми обрали Figma. Власне, вибір інструменту залежить від конкретних потреб та задач, а також від можливостей середовища.

1.3. Огляд засобів для розробки бази даних застосунку.

Розробка бази даних для застосунків є важливим етапом, який забезпечує збереження та управління даними. Існує низка популярних інструментів і бібліотек, які дозволяють розробникам легко інтегрувати базу даних у застосунки на Flutter. Наведемо деякі із них: SQLite, Hive, Moor, Firebase, Realm та ін.

Розглянувши ці інструменти, можна сказати, що у кожного свої переваги та недоліки, і вибір конкретного рішення залежить від вимог вашого проекту. SQLite (sqflite) та Moor (Drift) добре підходять для додатків з реляційними даними, тоді як Hive та Firebase можуть бути кращими для простих, легких додатків або для тих, що потребують хмарної синхронізації. Realm пропонує

сучасну базу даних з високою продуктивністю та реактивністю, що може бути корисним для складних додатків з великою кількістю даних.

Наша база даних не є складною. Тому, було вирішено для її створення використовувати інструмент Firebase.

Рисунок 1.3 – Логотип хмарної системи керування базами даних – Firebase

Firebase – це платформа від Google для створення мобільних та веб-додатків, яка включає базу даних Realtime Database та Firestore [15].

Основні можливості, якими наділений цей інструмент є:

- Хмарна база даних. Дані зберігаються в хмарі, що забезпечує синхронізацію між клієнтами;
- Реактивність. Автоматичне оновлення даних в реальному часі;
- Підтримка офлайн. Автоматичне збереження даних в офлайн-режимі з подальшою синхронізацією;
- Наявність Flutter-плагінів. Плагін для інтеграції Cloud Firestore та Realtime Database;
- Простота використання. Легкий API та інтеграція з іншими сервісами Firebase.

Також, не обійшлось і без незначних недоліків, а саме:

- Залежність від інтернету. Потрібен інтернет для синхронізації даних;
- Платні сервіси. Деякі функції можуть потребувати платної підписки.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз процесів та засобів для вирішення поставленого завдання.

Мобільні додатки мають значний вплив у теперішньому світі. Розвиток смартфонів та планшетів призвело до значного змінення невід'ємної частини нашого звичайного життя. Мобільні програми спрощують наше життя у великій кількості сферах: завдяки нам ми можемо спілкуватися, купляти, розважатися, переглядати новини, управляти фінансами та великою кількістю іншої побутової справи.

У загальній складності, процес створення мобільних додатків має в собі кілька етапів [1, 8]:

- Проектування. На цьому етапі втілюють ідеї для концепції застосунку: основні його функції визначаються, узгодження ідеї для інтерфейсу, структури та взаємодії для користувача;
- Кодування. Після процесу проєктування розробники програмують код, завдяки якому забезпечується робота всіх елементів програми, разом із взаємодією з базами даних, обробкою даних і візуальними ефектами. Програмісти застосовують обрану мову програмування, наприклад Kotlin, Swift або Dart чи інші, та з ідеї створюють робочий застосунок;
- Тестування. На цьому етапі необхідно переконатися, що функціонал додатку працює коректно, наприклад що всі кнопки, переходи на інші сторінки та взаємодії між ними працюють без багів;
- Підтримка та оновлення. Після завершення розробки застосунку, його розробники продовжують підтримувати оновленнями, завдяки отриманим відгукам користувачів вони можуть, провести аналіз на різні помилки та збої у програмі, що виникнули під час експлуатації, завдяки чому можуть надати оновлення, які допоможуть виправити ці помилки, щоб покращити функціональність додатку.

Після аналізу існуючих інструментів, для виконання даного мобільного застосунку нами було використано наступні засоби розробки: створення

мобільного застосунку мовою програмування Dart фреймворку Flutter за допомогою Android Studio у поєднанні баз даних Firebase. завдяки цій технології можна створити веб-програму, передачу двонаправлених даних між сервером і клієнтом, дозволяє створювати різні мікросервіси і т. д. Firebase має конкретний план розвитку, завдяки якому вона завжди вдосконалює свої можливості. для розроблення мобільних додатків де є великий фунціонал. Головна перевага платформи є у тому, ща завдяки ній створювачу не треба відволікатися на створення бекенду, тобто програмної частини проекту, яка прихована від звичайного користувача, наприклад, коду для сервера. Це полегшує і прискорює розробку мобільних додатків, дозволяючи зосередитися на UI/UX, тобто на користувацькому інтерфейсі його досвіді.

Firebase дозволяє розробникам швидко створювати додатки для Android і iOS, що вирішують широкий спектр завдань. Firebase включає такі функції, як Firebase Analytics, безкоштовне рішення для аналізу додатків, яке дозволяє оцінювати використання додатків та залучення користувачів. Firebase Cloud Messaging (FCM) є кросплатформовим рішенням для відправки повідомлень і сповіщень для Android, iOS і веб-додатків, яке наразі доступне безкоштовно [13].

Firebase Auth дозволяє виконувати аутентифікацію користувачів через клієнтську частину коду. Realtime Database забезпечує можливість працювати з базою даних в реальному часі та функції бекенду для додатків. Firebase Storage дозволяє безпечно завантажувати та вивантажувати файли для додатків Firebase, незалежно від якості мережі. Ці можливості підтримуються Google Cloud Storage.

Firebase підтримується сервісом Google Cloud Storage. Firebase Hosting, запущений у 2014 році, дозволяє хостити як статичні, так і динамічні веб-сайти, включаючи файли CSS, HTML, JavaScript і підтримку Node.js через Cloud Functions. ML Kit – це інструментарій для мобільних розробників, який включає різноманітні функції машинного навчання, такі як розпізнавання тексту, обличь, баркодів та інші, для роботи із зображеннями та наземними об’єктами.

Завдяки Figma разом з експлуатацією мов програмування розробки цього середовища ми можемо спроєктувати інтерфейс майбутнього додатку і потім загрузити його код в Android Studio. Додатково, щоб збільшити швидкість та покращити оформлення використовуються UI-фреймворки. Figma доступний у веб-браузері і доступна на багатьох платформах. Немає потреби в жодних встановлюваних програмах або додатках. Раніше дизайнери вважали, що робота через браузер – це лише перехідний етап у процесі, а не основне й серйозне середовище для роботи. Тому, треба трошки пристосуватися до цього. Усі файли для роботи зберігаються у хмарних середовищах та структуровані у простому дереві «команда → проєкт → файл». Концепція спільної роботи поширюється від одного продукту до іншого, показуючи, наскільки це захоплююче, динамічне і корисне для процесів [16].

У Figma є такі можливості: Фрейми (артборди) можуть бути вже готові, але також можна створювати власні, щоб краще відповідати конкретним завданням; модульна сітка допомагає організувати елементи дизайну в фреймах для більшої зручності; векторні форми використовуються для відтворення різних елементів інтерфейсу, а криві створюють криві та прості векторні форми; Text сумісний із Google Fonts і пропонує інструменти для інтеграції додаткових шрифтів за допомогою програми встановлення шрифтів; багатокористувацький режим редагування; ефективне зберігання файлів, анотування всередині макета, створення прототипів і виконання кількох завдань одночасно [14].

Особливості Android Studio: лайв-макети (layout): редактор WYSIWYG – Демонстрація кодування в реальному часі, представлення програм у процесі їх створення; інструменти розробника для оптимізації коду, перекладу, відстеження прогресу та аналізу даних Google Analytics; керування бета-версією та покроковими випусками через Gradle; покращення коду Android і швидке вирішення проблем; використання Lint для перевірки продуктивності, зручності та сумісності; використання ProGuard і сигнатур програм; використання шаблонів для стандартних дизайнів і компонентів Android; інтуїтивно

зрозумілий редактор макета з функцією перетягування; інтерфейс для перегляду макетів на різних конфігураціях екрана одночасно [8, 17].

2.2. Побудова дерева проблем та дерева цілей.

Дерево проблем – ієрархічна структура, яка являє собою модель проблемної ситуації і складається з кореня, стовбура і крони. Корінь – це причина, через яку виникла проблема. Якщо усунути корінь, тоді і ніякої проблеми відповідно не буде. Стовбур – це власне характеристика проблеми, а крона – наслідки, до яких призводить існування самої проблеми. Якщо усунути крону, то зникнуть і наслідки, але сама проблема не зникне, а лише створить інші наслідки. Отже, для побудови дерева проблем необхідно чітко визначити основну проблему, так якщо сформулювати її невірно, то правильна характеристика стане неможливою, адже проблема, яку ми вважали основною, може бути лише наслідком однієї з більш глобальних проблем [2, 6].

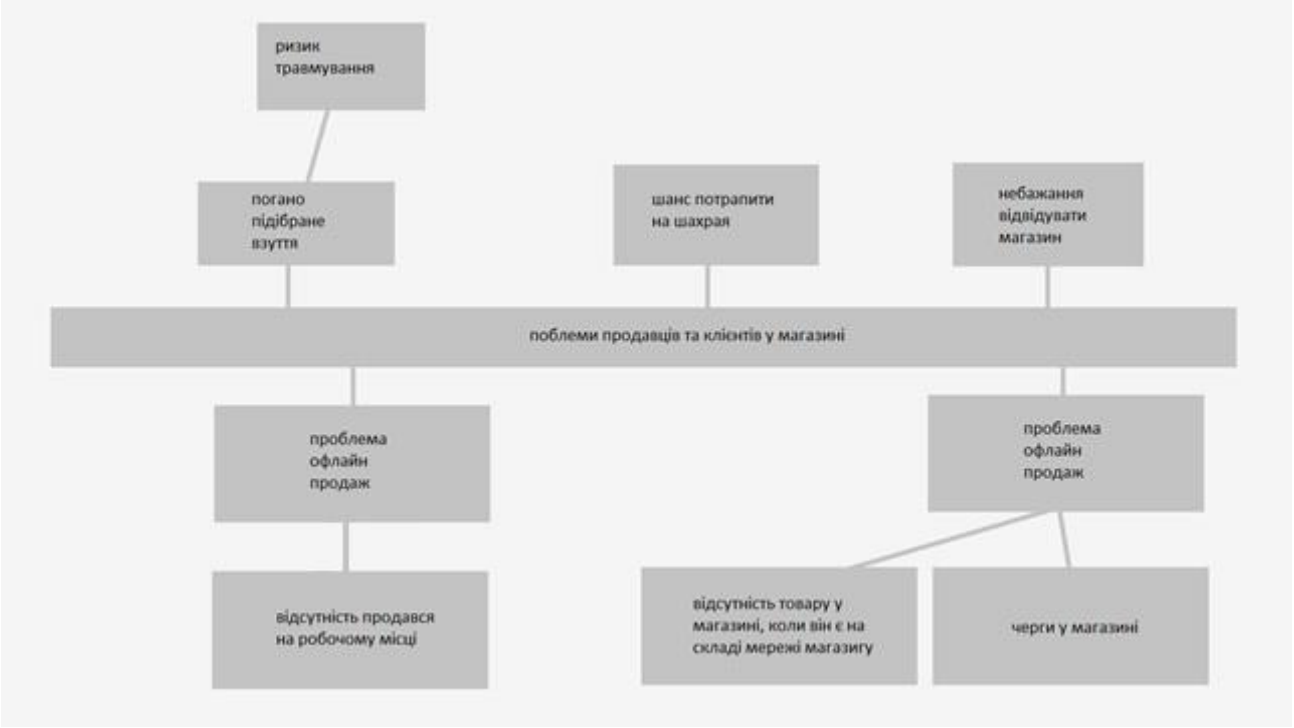

Рисунок 2.1 – Дерево проблем

На рисунку 2.1 можемо бачити глобальну проблему продавців та клієнтів у магазинах, яку породжують такі причини, як черги, відсутність продавця на робочому місці, неможливість купити бажаний товар. Ці причини створюють такі наслідки: небажання відвідувати магазин, шанс потрапити на шахрая, погано підібране взуття, що може призвести до травм під час подорожі у непідходящому взутті.

Дерево цілей – ієрархічна структура, що відображає сукупність завдань в процесі створення проєкту. Ціль – бажаний результат, якого хоче досягти організація або проєкт при виконанні задач. Цілі повинні бути конкретними та мати спосіб досягнення, а також узгоджені між собою. Чим більше цілей прагнемо досягти, тим проєкт складніший за своєю структурою та управлінням.

Дерево цілей є одним з найефективніших способів планування задач. При правильному зображенні дерева ми можемо спостерігати проблеми, з якими доведеться зіткнутися та які додаткові ресурси будуть потрібні [2, 6].

При побудові дерева цілей дізнаємось:

- як організувати всі підструктури;
- як здійснити контроль над всіма підструктурами та встановити чіткі задачі та терміни їх виконання;
- як забезпечити необхідний контроль над бізнес-процесами;
- як досягти ефективного процесу розробки та прийняття управлінських рішень.

При побудові дерева цілей потрібно дотримуватись таких принципів:

- враховувати всі необхідні ресурси. Потрібно добре все спланувати, так як, швидше за все, відразу не вистачить ресурсів для реалізації поставлених завдань, або ж немає можливості для оцінки ресурсів через глобальність даної проблеми;
- конкретизувати всі задачі. Потрібно проаналізувати усі параметри, за якими буде вирішено, чи виконана задача, а також встановити певний часовий термін, за який вона має бути виконана;

- розбиття на етапи. Спочатку потрібно вибрати основну ціль, а потім проаналізувати ресурси, які необхідні для її реалізації. Далі потрібно вибрати підцілі, для яких також аналізуємо ресурси і виконуємо це доти, доки не буде продумана вся схема реалізації;
- сумісність. При виконанні усіх цілей нижнього рівня повинно бути забезпечено виконання цілі вищого рівня. Якщо ж це не так,– значить, дерево побудоване невірно;
- відповідність структурі підприємства. Структура дерева цілей повинна бути відповідною до структури підприємства, яке ці цілі буде виконувати.
- метод декомпозиції. Даний метод відповідає за розбиття основної цілі на підцілі.

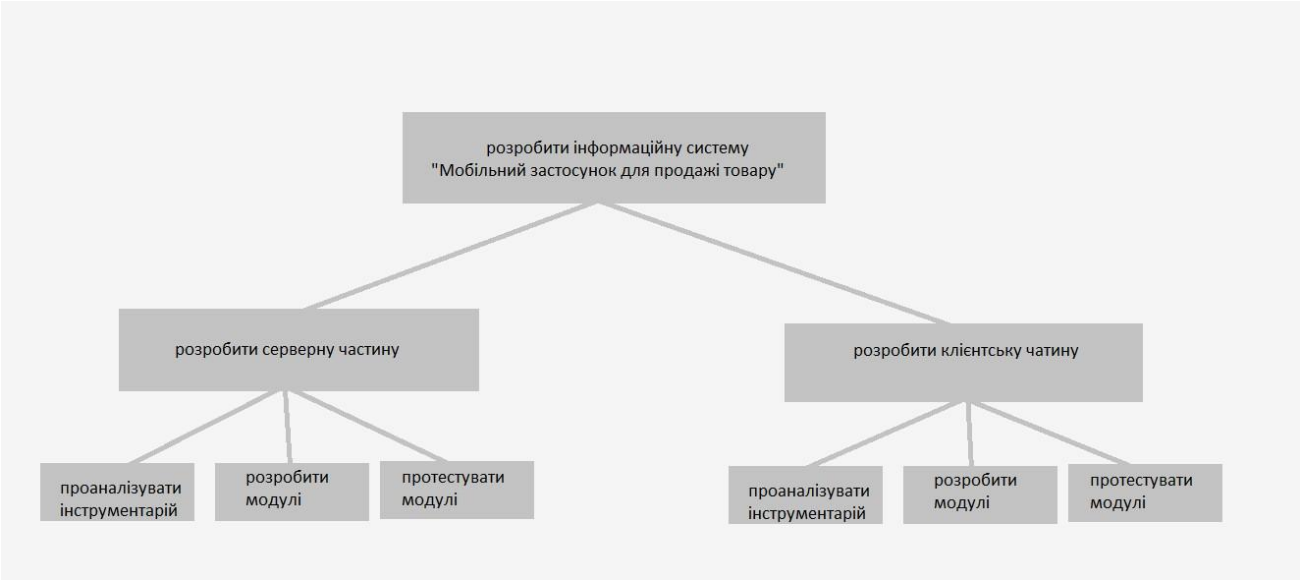

Рисунок 2.2 – Дерево цілей

Розглянувши дерево цілей (рис. 2.2), можемо побачити що основною ціллю є розробка програмного продукту, а підцілями – розробка підмодулів, які забезпечать реалізацію цілісної системи.

2.3. Схеми системи з врахуванням інформаційних потоків.

Структурна схема – сукупність функціональних частин продукту, зв’язків між ними та їх значення для системи в цілому. На схемі складові частини зображуються за допомогою прямокутників, а зв’язки – стрілками [7].

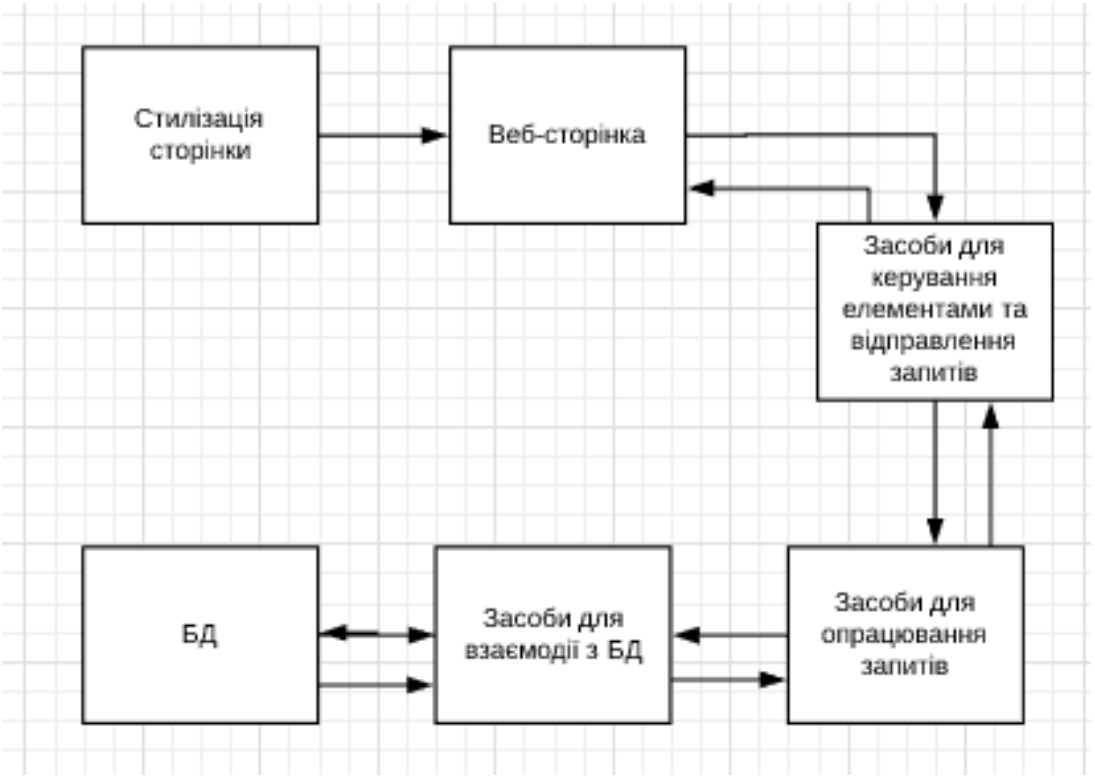

Рисунок 2.3 – Структурна схема

Розглянувши рисунок 2.3 можна сказати, що система складається з таких елементів, як стилізація сторінки, веб-сторінка, засоби для керування елементами та відправлення запитів, засоби для опрацювання запитів, засоби для взаємодії з БД, та сама БД.

Деякі елементи взаємодіють один між одним у зворотньому напрямку, тобто спочатку з сторінки відправляється запит на обробку і виконує певні дії з БД, а потім на сторінку відправляється відповідь. Це забезпечує певний зворотній зв’язок.

UML (уніфікована мова моделювання) – мова графічного опису для об’єктного моделювання розробки програмного забезпечення, бізнес-процесів,

системного проектування, і відображення організаційних структур. UML дозволяє розробникам програмного забезпечення досягнути одного стандарту в графічних позначеннях і більше концентрувати увагу на проектуванні та архітектурі систем [3].

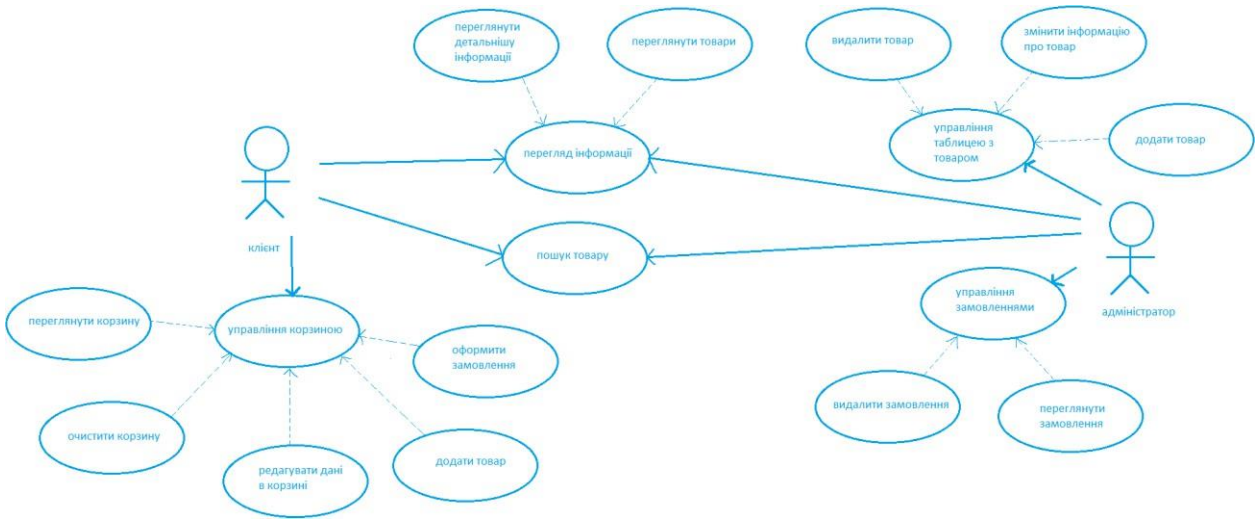

Рисунок 2.4 – Use-case діаграма

Якщо розглянути більш детально рисунок 2.4, то бачимо 2 групи користувачів: клієнт та адмін. Користувачі, які не зареєструвалися, все ще можуть переглядати інформацію про продукт, зокрема переглядати списки продуктів або натискати «докладніше», щоб отримати доступ до вичерпної інформації про певні продукти. Розглянемо функцію управління «кошиком». До функції управління «кошиком» входять ще декілька функцій, а саме: додати товар у «кошик», редагувати данні в «кошику», продивитись «кошик», очистити «кошик» та оформити замовлення. Додати товар у «кошик» можна тільки зі сторінки з детальною інформацією про товар, натиснувши на кнопку «додати до кошику».

Під редагуванням даних розуміється можливість змінити кількість одиниць товару, що замовляється, та можливість видалення непотрібного товару з «кошику». Переглянути «кошик» користувач може перейшовши на відповідне посилання у меню, яке розташовується у верхній частині застосунку, або ж натиснувши на посилання «перейти до кошика», яке виводиться після додавання

товару у «кошик». Щоб очистити «кошик» потрібно натиснути відповідну кнопку. Щоб оформити замовлення треба натиснути відповідну кнопку у «кошику», після чого відбудеться перехід на сторінку, де треба ввести адресу доставки та телефон.

Адміністратор має доступ до всіх функцій, які має зареєстрований користувач, за винятком функцій управління «кошиком». Крім того, з'явилися нові функції: можливість керувати таблицею запасів товарів і обробляти замовлення. Давайте заглибимося в функцію керування таблицею товарів. Ця функція включає кілька додаткових функцій, зокрема додавання продуктів, видалення продуктів і зміну інформації про продукт. На сторінці продукту кожне поле потрібно заповнити, перш ніж введені дані будуть збережені в базі даних. Щоб видалити продукт, перейдіть на сторінку продукту будь-якої категорії, зазначеної в меню в правій частині додатку, і клацніть відповідне посилання, що відображається біля кожного елемента.. Під керуванням замовленнями розуміється можливість видалення замовлення, якщо той виконаний чи некоректний, чи перегляду поточних замовлень.

IDEF0 – це метод, призначений для моделювання рішень, дій та активності організації, процесів або системи. Ефективні моделі IDEF0 допомагають організувати аналіз системи та сприяють хорошому зв'язку між аналітиком і замовником. IDEF0 корисний для встановлення сфери застосування, особливо для функціонального аналізу. Важливо представити потік прогресу або концепції моделі візуально, щоб полегшити її інтерпретацію для користувача. Наочні схеми допомагають уникнути неправильного тлумачення та забезпечити стандарт для різних користувачів [4].

Як інструмент комунікації, IDEF0 допомагає досягнути консенсусу запитів експертів у спрощеній графічній формі. Таким чином, IDEF0 моделі часто створюються як одне з перших завдань в процесі розробки системи. Обов'язковою умовою є те, що схема повинна бути максимально простою та зручною для користувача.

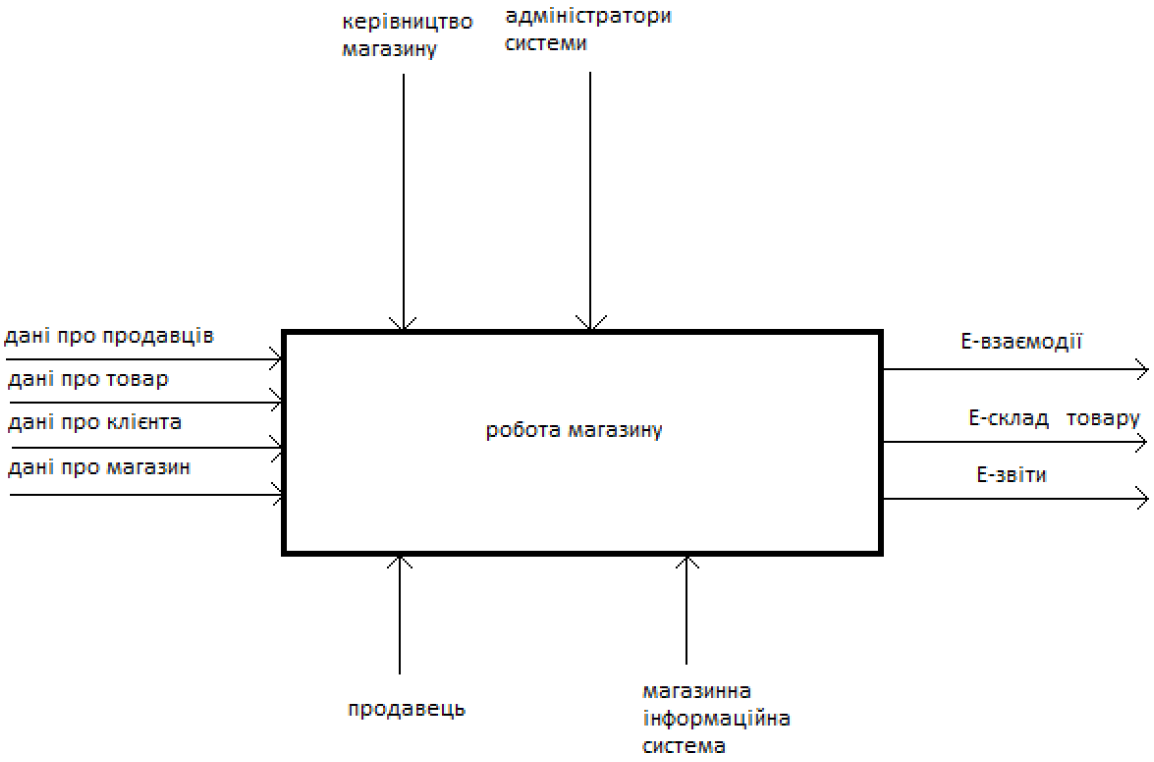

Рисунок 2.5 – Контекстна діаграма системи автоматизації роботи магазину

На даній діаграмі, блок «Робота магазину» отримує на вхід: дані про клієнта, дані про продавців, дані про товар, дані про магазин. Механізмами виступають продавці, магазинна інформаційна система. Процес роботи магазину контролюється керівництвом магазину та його адміністраторами. До виходів блоку належать: Е-взаємодії, Е-склад товару, Е-звіти.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Діаграми діяльності.

Діаграма діяльності – відображає діяльність системи, тобто показує певні стани системи, які взаємодіють між собою визначеним чином та кожен наступний стан спрацьовує після завершення роботи попереднього. Дана діаграма за своїм виглядом дуже схожа на блок-схему і показує, як система переходить від однієї дії до іншої. Вона застосовується для відображення послідовних та паралельних дій проєктованої системи. Якщо ми маємо певну ціль, то діаграма діяльності показує послідовність дій, які необхідно виконати для її досягнення [5, 7].

Розглянемо декілька прецедентів та побудуємо для них відповідні діаграми, які допоможуть зрозуміти, як функціонує створена система, та якою є послідовність дій:

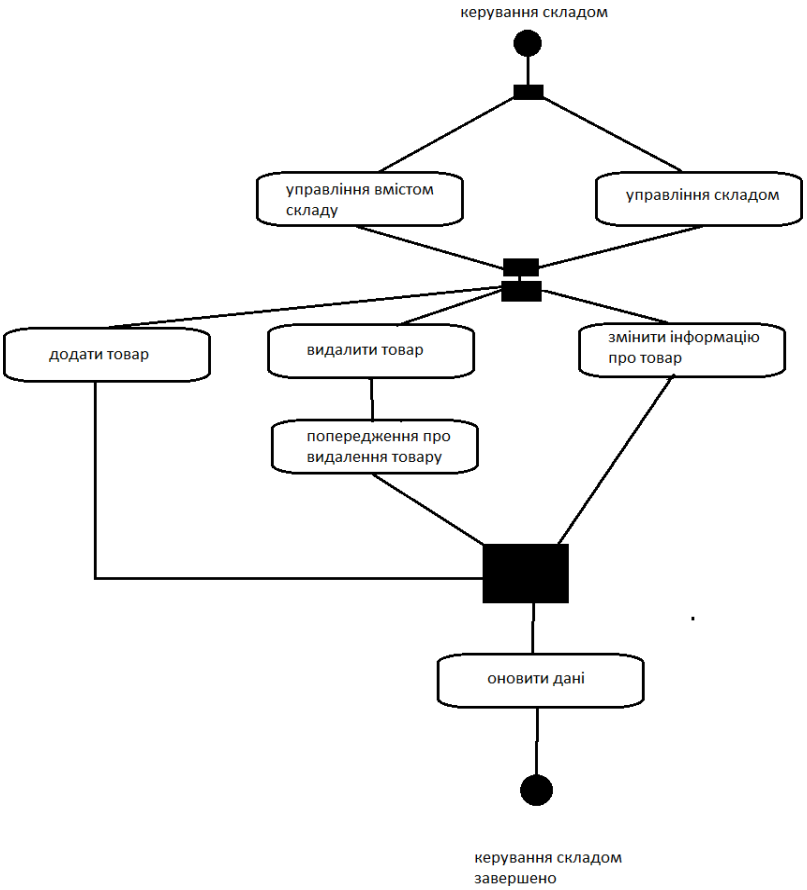

Рисунок 3.1 – Діаграма діяльності для прецеденту керування складом

Управління складом товару є невід’ємною частиною даної програми. При видаленні товару, передбачено вивід попереджуючого повідомлення про можливу його втрату, що допоможе запобігти випадковому видаленню. Крім того передбачено додавання на склад товару.

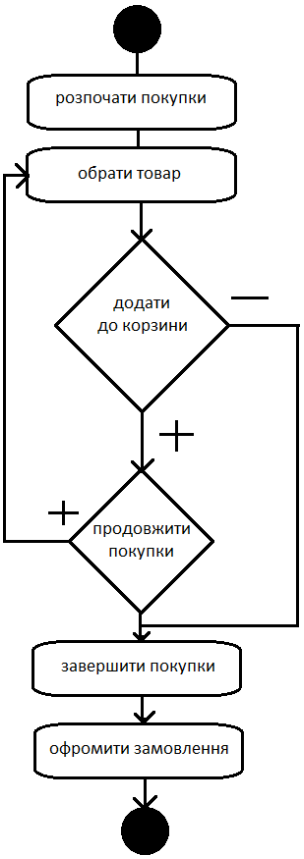

Рисунок 3.2 – Діаграма діяльності для прецеденту покупок

Оформлення покупок є достатньо формалізованим процесом. Тому, у даній діаграмі ми спростили цей процес з підтвердженням документів і т. д. Для початку покупок, користувачеві необхідно вибрати товар, після чого у новоствореному вікні, потрібно оглянути товар та додати у «кошик» (якщо він захоче його купити). Наступним кроком користувач може продовжити покупки або оформити замовлення. Після заповнення форми замовлення, користувач повинен підписати документ із подальшим надсиланням до центральної бази даних.

3.2. Структура бази даних.

База даних – це структурована компіляція даних, упорядкована на основі визначеної концепції, яка описує атрибути даних і те, як вони пов’язані між собою. Ця компіляція обслуговує принаймні один домен програми. Як правило, база даних складається зі схем, таблиць, представлень, збережених процедур та інших елементів. Дані в базі даних відповідають певній організаційній моделі. Отже, сучасна база даних не тільки містить дані, але також містить описи даних і може містити інструменти для маніпулювання ними [18].

В даній роботі для зберігання даних ми використали Firebase з такими полями:

- email
- idnumber
- Fname
- Lname
- phonenumber
- type

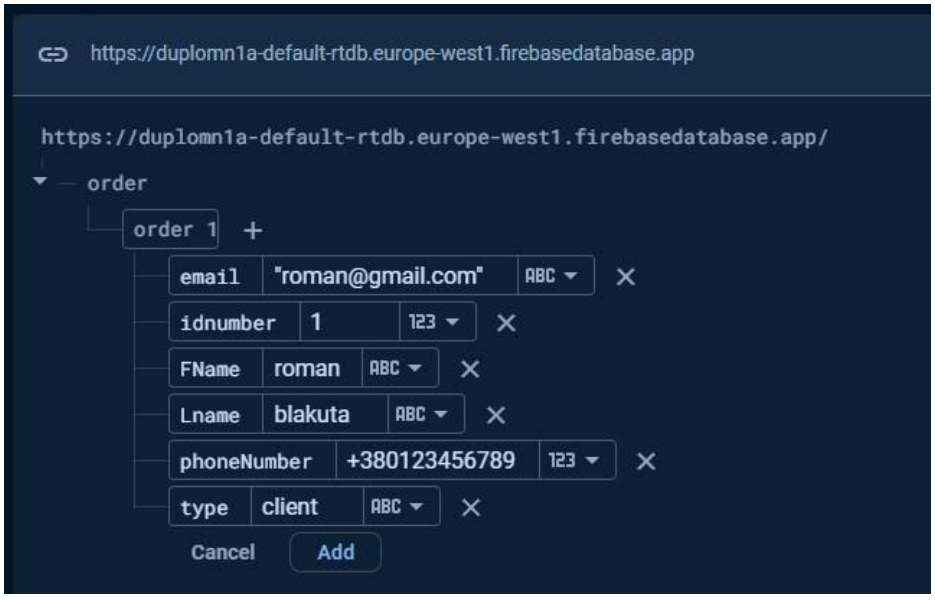

Рисунок 3.3 – База даних застосунку в Firebase

3.3. Представлення компонентів програми.

Клас – це сукупність об’єктів із загальною структурою, поведінкою, зв’язками та семантикою. Структура класу представлена його атрибутами.

Для чіткого розуміння, як саме потрібно статично зобразити систему, на відміну від динамічних аспектів, застосовується діаграма класів (рис. 3.4). За допомогою даної діаграми, можна зобразити різні класи із різними зв’язками, на основі яких функціонує система.

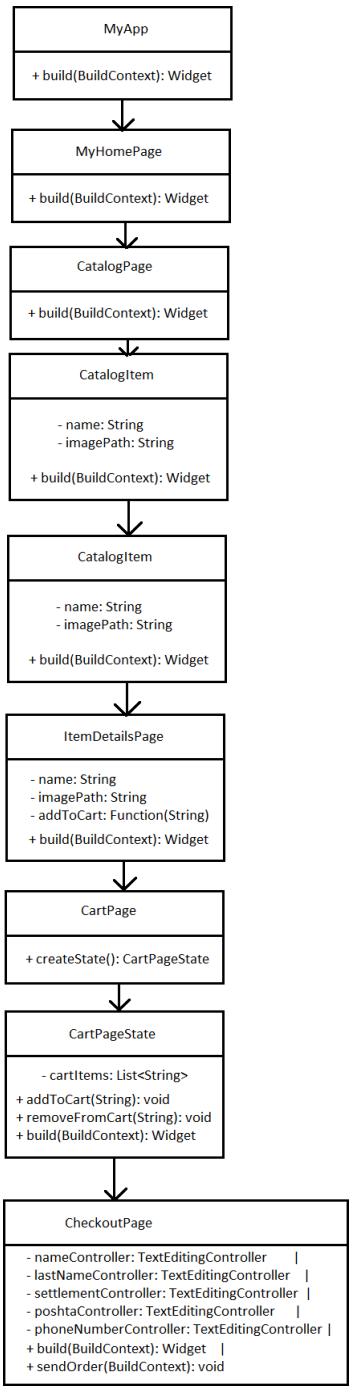

Рисунок 3.4 – Діаграма класів програми

Програмний продукт містить ряд модулів, які необхідні для його правильної роботи:

MyApp – початковий клас, який запускає, налаштовує тематику та головну сторінку застосунку.

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,  //виключає debug
      title: 'підкрадулі стор',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: MyHomePage(),
    ); // MaterialApp
  }
}
```

Рисунок 3.5 – Фрагмент коду класу MyApp

MyHomePage – головна сторінка застосунку. Використовує DefaultTabController для створення двох вкладок: «Каталог» та «Кошик».

```
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: 2,
      child: Scaffold(
        appBar: AppBar(
          title: Text('підкрадулі стор'),
          bottom: TabBar(
            tabs: [
              Tab(text: 'Каталог'),
              Tab(text: 'Кошик'),
            ],
          ), // TabBar
        ), // AppBar
        body: TabBarView(
          children: [
            CatalogPage(),
            CartPage(),
          ],
        ), // TabBarView
      ), // Scaffold
    ); // DefaultTabController
  }
}
```

Рисунок 3.6 – Фрагмент коду класу MyHomePage

CatalogPage – сторінка каталогу товарів. Використовує ListView для відображення списку товарів у вигляді елементів CatalogItem.

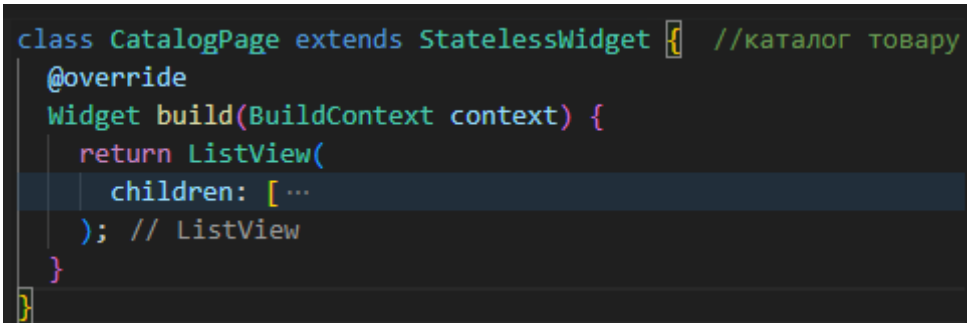

Рисунок 3.7 – Фрагмент коду класу CatalogPage

CatalogItem – компонент, що представляє окремий товар у каталозі. Відображає назву та зображення товару, переходить на сторінку з детальною інформацією про товар при натисканні.

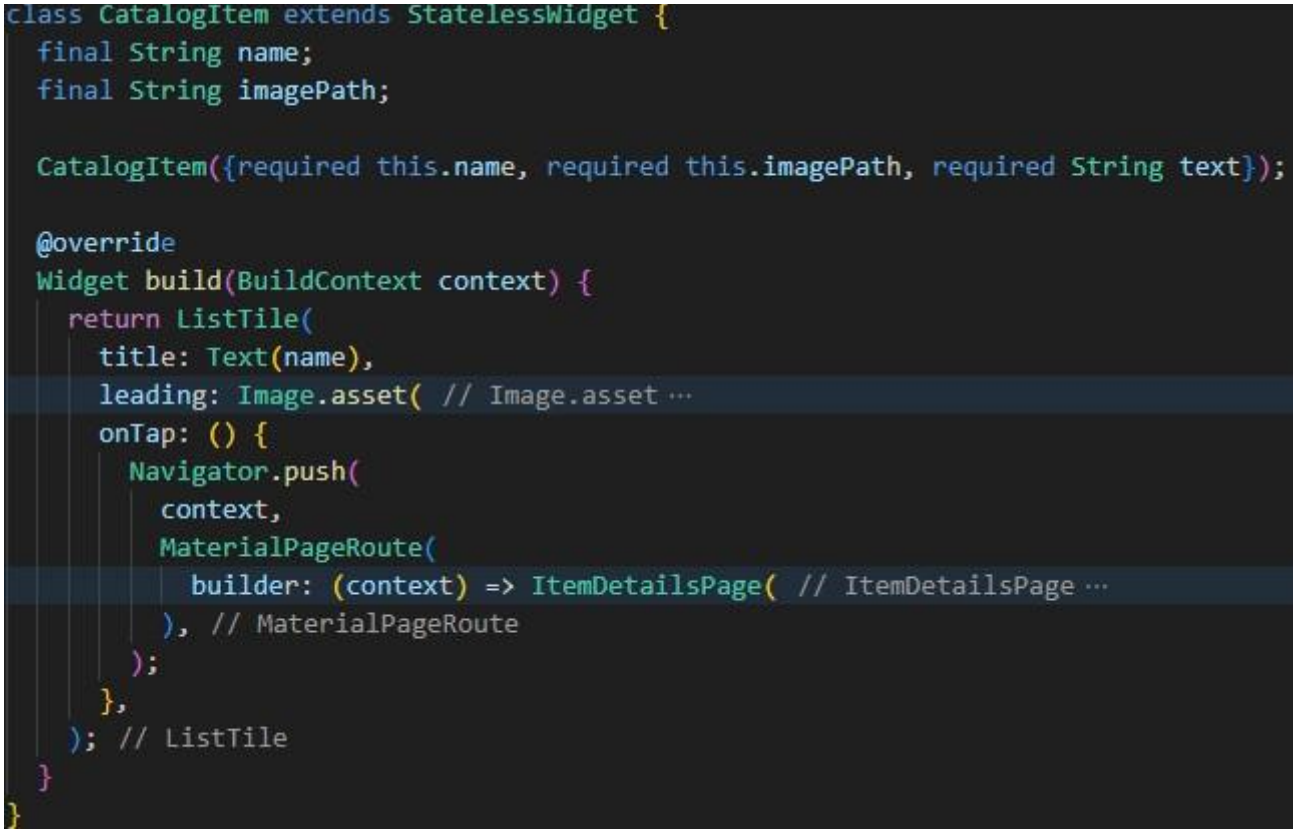

Рисунок 3.8 – Фрагмент коду класу CatalogItem

ItemDetailsPage – сторінка з детальною інформацією про товар, відображає велике зображення товару, назву, вартість. Дозволяє вибрати розмір товару через випадаюче меню. Додає товар у «кошик» при натисканні кнопки.

```
class ItemDetailsPage extends StatelessWidget {
  final String name;
  final String imagePath;
  final Function(String) addToCart;

  ItemDetailsPage({required this.name, required this.imagePath, required this.addToCart});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Деталі товара')),
      body: Column(
        mainAxisAlignment: MainAxisAlignment.start,
        children: [
          Image.asset( // Image.asset ···
          SizedBox(height: 20),
          Text( // Text ···
          SizedBox(height: 10),

ElevatedButton(     // вибір розміру
            onPressed: () {
              final RenderBox button = context.findRenderObject() as RenderBox;
              final RenderBox overlay = Overlay.of(context).context.findRenderObject() as RenderBox;
              final Offset position = button.localToGlobal(Offset.zero, ancestor: overlay);

              showMenu(
                context: context,
                position: RelativeRect.fromLTRB( // RelativeRect.fromLTRB ···
                items: <PopupMenuEntry<int>>[ // <PopupMenuEntry<int>>[] ···
              ).then((value) {
                // обробка вибраного елементу
                if (value != null) {···
                }
              });
            },
            child: Text('виберіть розмір EU'),
          ), // ElevatedButton

          Text( // Text ···
          SizedBox(height: 20),
          ElevatedButton( // ElevatedButton ···
        ],
      ), // Column
    ); // Scaffold
```

Рисунок 3.9 – Фрагмент коду класу ItemDetailsPage

CartPage – сторінка «кошика». Відображає список доданих товарів. Дозволяє видалити обраний товар. Містить кнопку для переходу до сторінки оформлення замовлення.

```dart
class CartPage extends StatefulWidget {
  @override
  CartPageState createState() => CartPageState();
}

class CartPageState extends State<CartPage> {
  static List<String> cartItems = [];

  static void addToCart(String item) {···

  static void removeFromCart(String item) {···

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        ElevatedButton( // ElevatedButton ···
        Expanded(
          child: ListView.builder(
            itemCount: cartItems.length,
            itemBuilder: (context, index) {
              return ListTile(
                title: Text(cartItems[index]),
                trailing: IconButton(
                  icon: Icon(Icons.delete),
                  onPressed: () {
                    setState(() {···
                  },
                ), // IconButton
              ); // ListTile
            },
          ), // ListView.builder
        ), // Expanded
      ],
    ); // Column
  }
}
```

Рисунок 3.10 – Фрагмент коду класу CartPage

CheckoutPage – цей блок програми відповідає за розробку функціоналу, який створює та відображає сторінку, на якій користувач може оформити своє замовлення.

```dart
import 'package:flutter/material.dart';
import 'package:cloud firestore/cloud firestore.dart';

class CheckoutPage extends StatelessWidget {
  final TextEditingController nameController = TextEditingController();
  final TextEditingController lastNameController = TextEditingController();
  final TextEditingController settlementController = TextEditingController();
  final TextEditingController poshtaController = TextEditingController();
  final TextEditingController phoneNumberController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar( // AppBar ···
      body: Padding(
        padding: EdgeInsets.all(16.0),
        child: Column( // Column ···
      ), // Padding
    ); // Scaffold
  }

  void sendOrder(BuildContext context) async {
    String name = nameController.text;
    String lastName = lastNameController.text;
    String settlement = settlementController.text;
    String poshta = poshtaController.text;
    String phoneNumber = phoneNumberController.text;

    //створюю документ у колекції "orders" у Firestore
    await FirebaseFirestore.instance.collection('orders').add({···

    //повертаюся назад після успішного замовлення
    Navigator.pop(context);
  }
}
```

Рисунок 3.11 – Фрагмент коду класу CheckoutPage

CartItem – представляє окремий товар у «кошику» з його атрибутами, такими як ідентифікатор, назва, кількість та вартість.

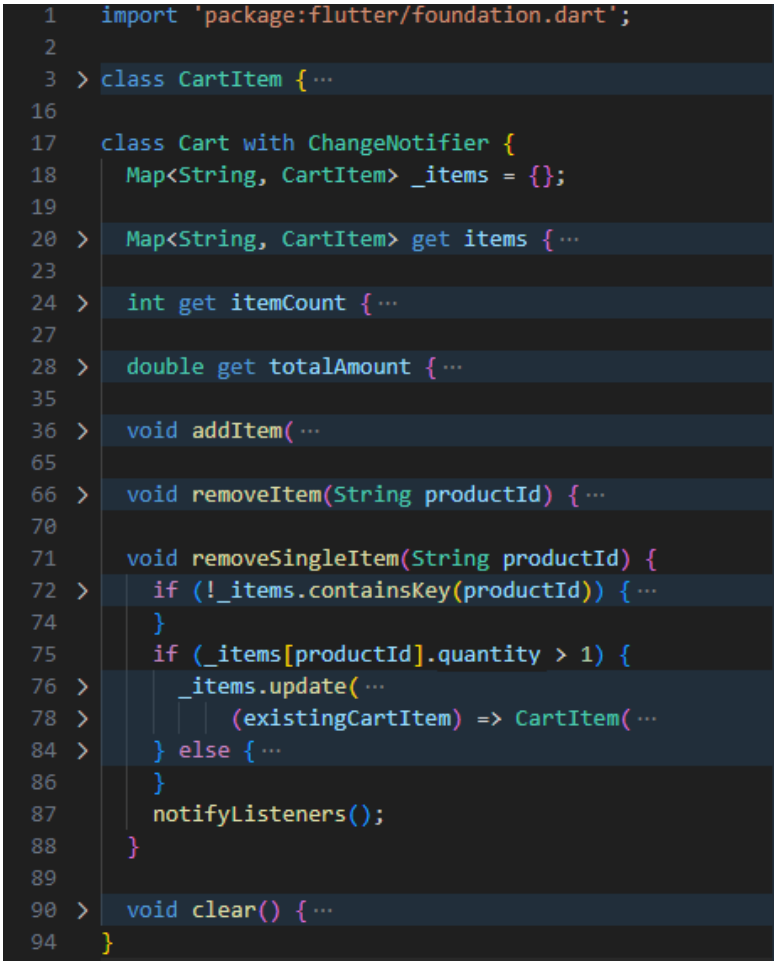

Рисунок 3.12 – Фрагмент коду класу CartItem

OrderItem – представляє окреме замовлення з атрибутами, такими як ідентифікатор, загальна вартість, список товарів у замовленні, дата та час замовлення.

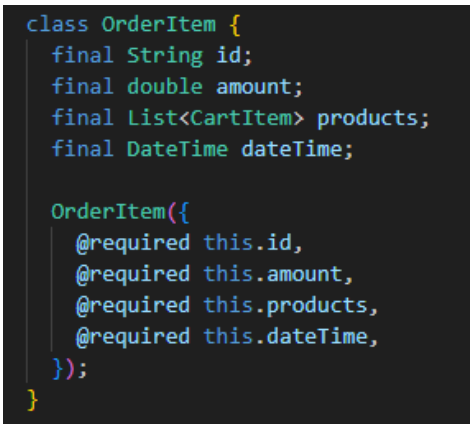

Рисунок 3.13 – Фрагмент коду класу OrderItem

Product – представляє окремий продукт з атрибутами: ідентифікатор, назва, опис, вартість, URL зображення та статус улюбленого.

```dart
class Product with ChangeNotifier {
  final String id;
  final String title;
  final String description;
  final double price;
  final String imageUrl;
  bool isFavorite;

  Product({
    @required this.id,
    @required this.title,
    @required this.description,
    @required this.price,
    @required this.imageUrl,
    this.isFavorite = false,
  });

  void _setFavValue(bool newValue) {
    isFavorite = newValue;
    notifyListeners();
  }

  Future<void> toggleFavoriteStatus(String token, String userId) async {
    final oldStatus = isFavorite;
    isFavorite = !isFavorite;
    notifyListeners();
    final url =
        'https://shop-49ff6-default-rtdb.firebaseio.com/userFavorites/$userId/$id.json?auth=$token';
    try {
      final response = await http.put(
        Uri.parse(url),
        body: json.encode(···
      );
      if (response.statusCode >= 400) {···
      }
    } catch (error) {
      _setFavValue(oldStatus);
    }
  }
}
```

Рисунок 3.14 – Фрагмент коду класу Product

Products – завдяки цьому компоненту можна управляти списком товарів.

```dart
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

import '../models/http exception.dart';
import './product.dart';

class Products with ChangeNotifier {
  List<Product> _items = [ ···
  final String authToken;
  final String userId;

  Products(this.authToken, this.userId, this._items);

  List<Product> get items { ···

  List<Product> get favoriteItems { ···

  Product findById(String id) { ···

  Future<void> fetchAndSetProducts([bool filterByUser = false]) async { ···

  Future<void> addProduct(Product product) async { ···

  Future<void> updateProduct(String id, Product newProduct) async { ···

  Future<void> deleteProduct(String id) async { ···
}
```

Рисунок 3.15 – Фрагмент коду класу Products

При розробці програмних модулів деякі з них є дуже схожими за структурою і відрізняються лише полями, тому у застосунку винесено один модуль, який є найбільше розкритим і описує суть таких класів.

3.4. Розробка та опис інтерфейсу користувача.

Інтерфейс користувача має велике значення при розробці програмного продукту, адже кожен хоче, щоб його робота з додатком була якомога приємніша та легка. Він забезпечує передачу інформації між користувачем та сервером [10]. Тому, ми переосмислили деякі речі у існуючих застосунках і підлаштували їх під себе.

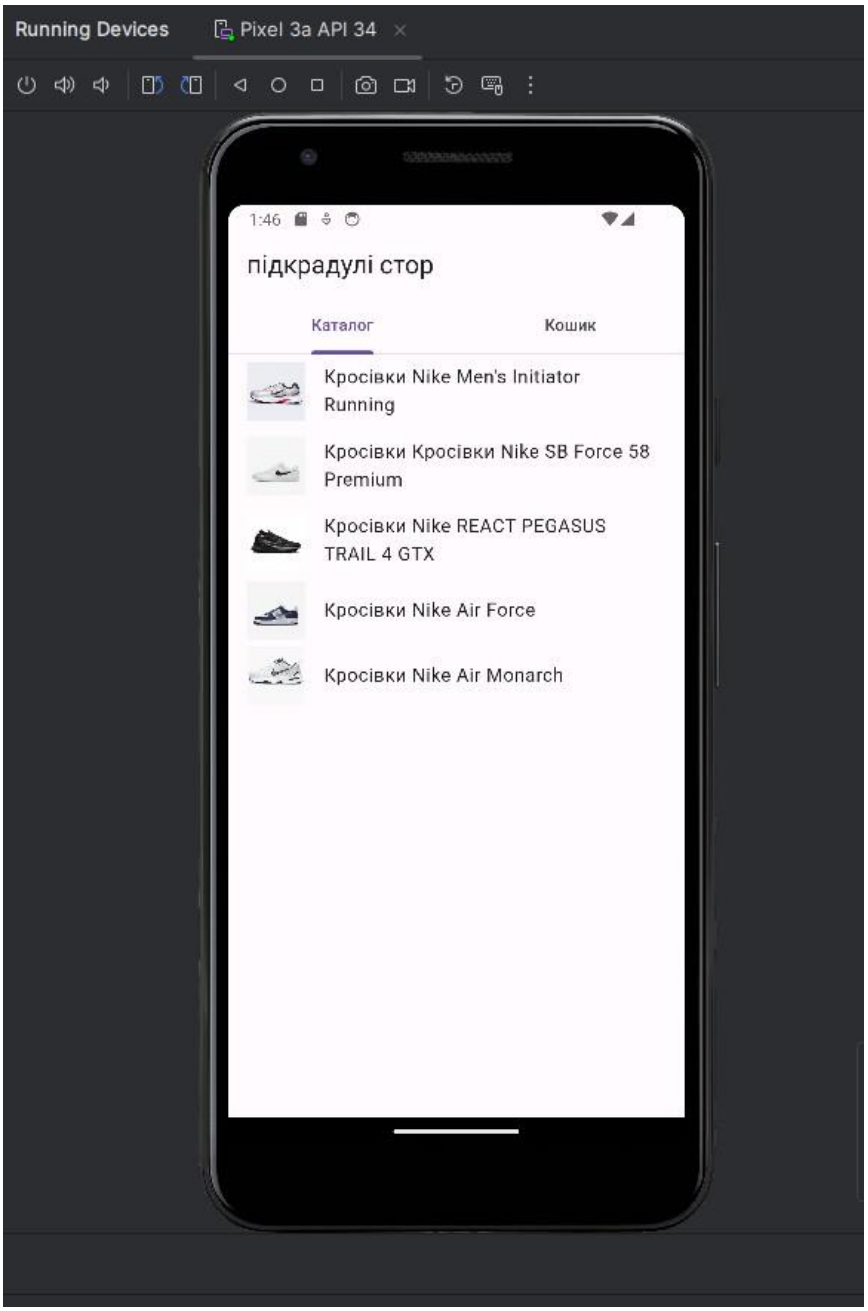

Рисунок 3.16 – Екран головного меню

У нашій програмі для зручності користувача було спрощено пошук товару, вивівши весь товар на початковий екран. При натисканні на якийсь рядок із товаром відкриється наступна сторінка, де можна буде подивитися характеристики товару та додати його у «кошик».

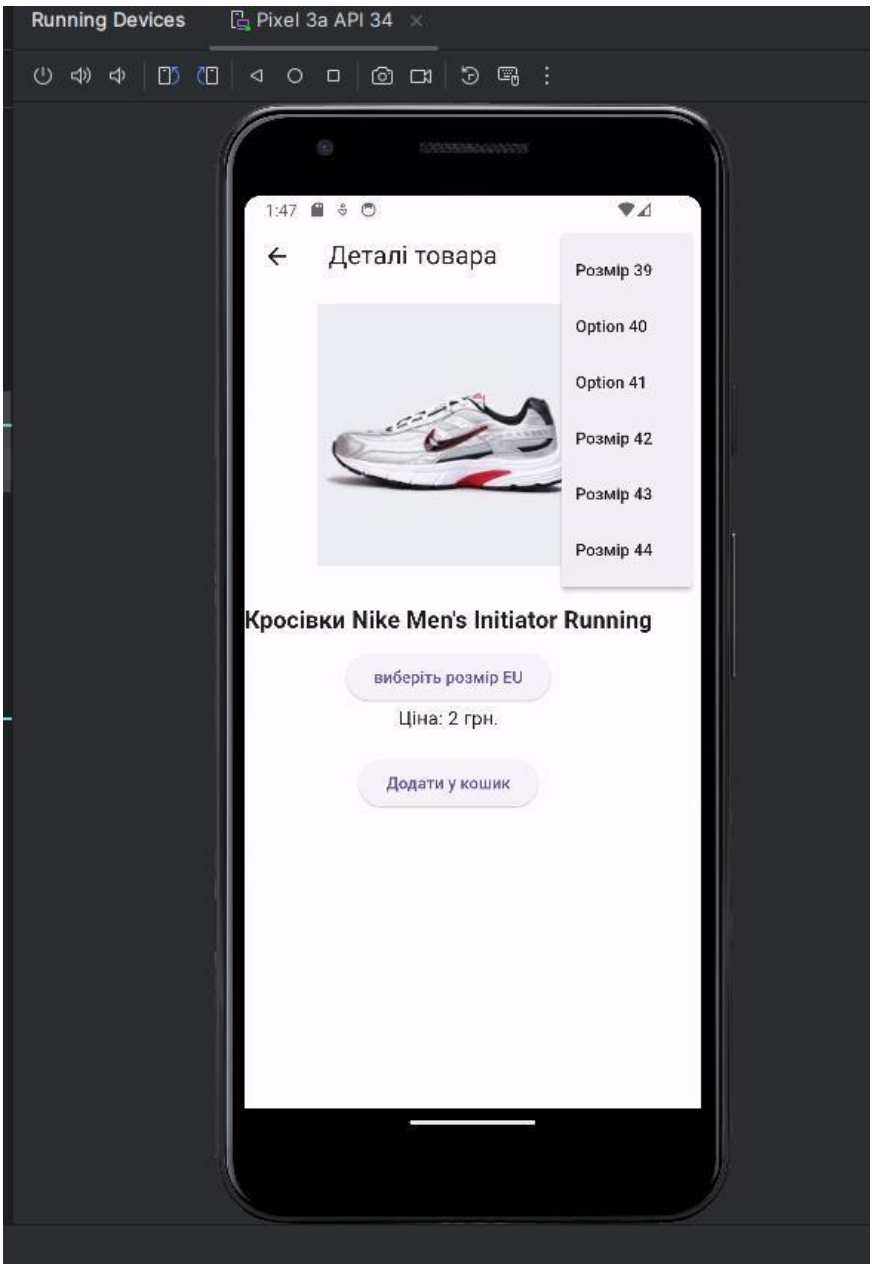

Рисунок 3.17 – Екран з інформацією про товар

На цьому екрані ми бачимо опис товару, та можемо вибрати розмір і додати товар у «кошик».

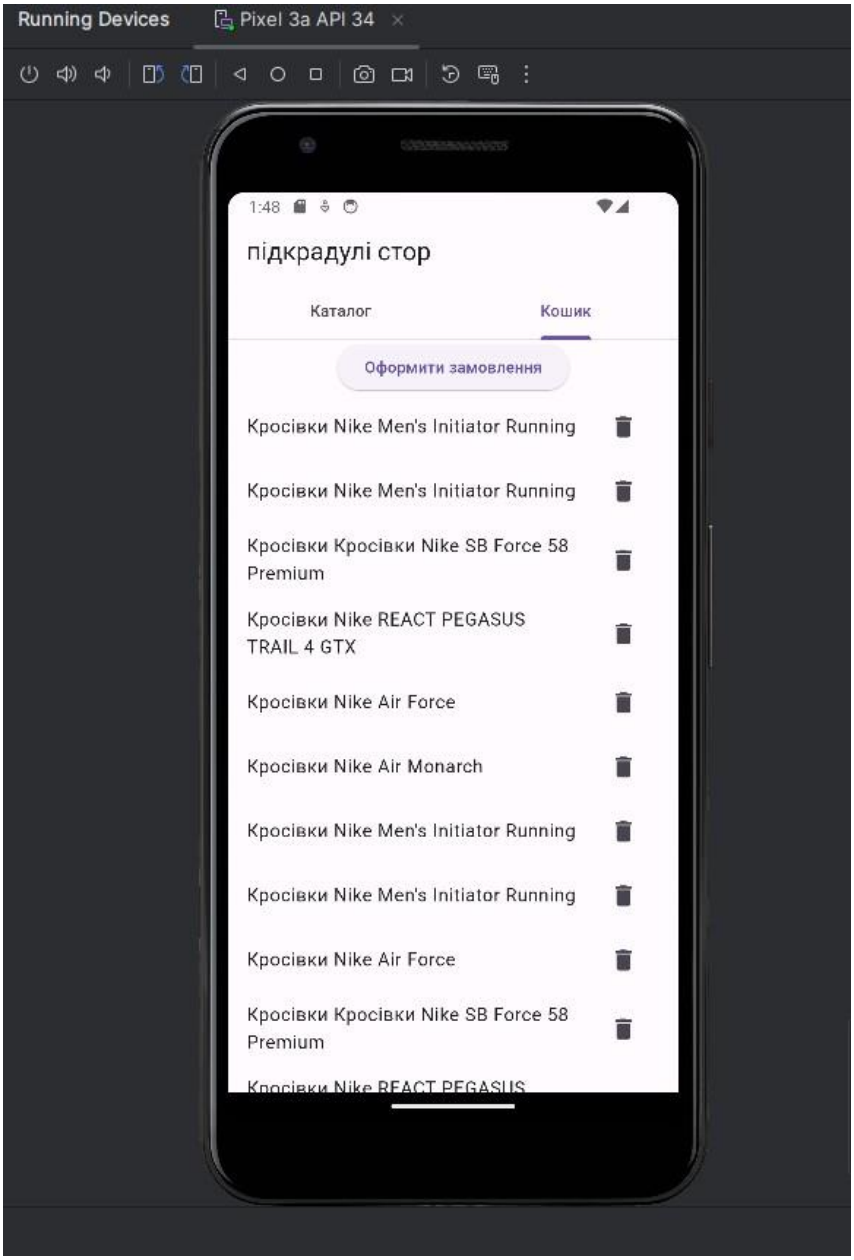

Рисунок 3.18 – Екран «кошик»

Тут можна переглянути весь обраний товар, видалити те, що зайве і перейти до оформлення замовлення.

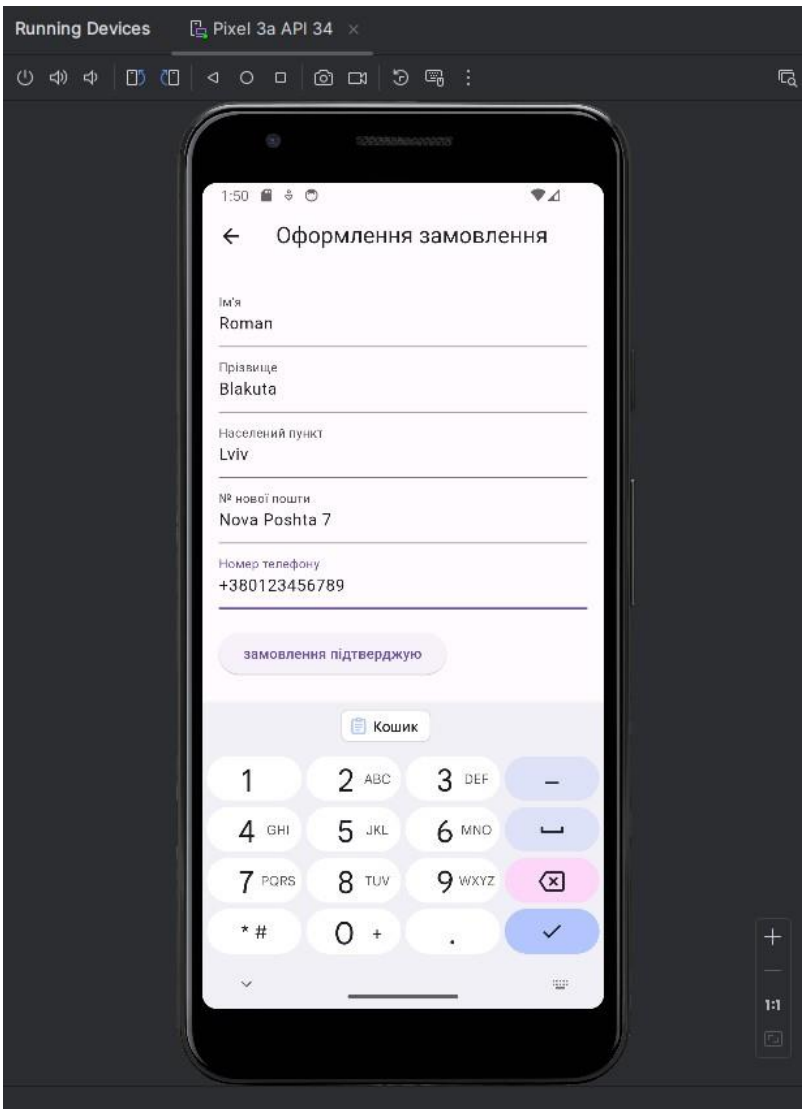

Рисунок 3.19 – Екран оформлення замовлення

На цьому етапі користувач може оформити замовлення (вказати особисті дані, адресу доставки та номер телефону).

3.5. Тестування продукту.

Тестування мобільних програм сильно відрізняється від тестування настільного програм, оскільки воно включає такі пристрої, як смартфони чи планшети, а не лише монітори, адже смартфони відрізняються як габаритами, так і за технічними характеристиками.

Зазвичай мобільні девайси, середньостатистичних користувачів мають широке ранжування версій ОС, від застарілих до найновітніших. Слід

враховувати розширення всіх екранів. Тому, при розробці мобільних додатків, і при їх тестуванні, слід зважати на ряд моментів [9, 19]:

- тестування на різних розширення телефонів, планшетів;
- монітор комп'ютера не може динамічно змінювати орієнтацію екрана;
- існує певний список обов'язкових функціональних параметрів мобільних додатків;
- смартфон завжди знаходиться у русі, тому з невеликою ймовірністю можна випадково викликати якась дію, як блокування, сповіщення та інші;
- пристрій завжди перебуває в стані пошуку мережі;
- під час тестування варто справність програми на неоднакових швидкостях передачі даних;
- під час розробки WEB/Mobile застосунків необхідно спрогнозувати перебування під час різних природніх умовах – при різному освітленні варто застосовувати контрастні кольори;
- не слід забувати, що основне завдання, наприклад смартфону, як і раніше є дзвінки, й додаток нічим не має заважати основній функціональності пристрою;
- сучасні мобільні девайси мають на додачу ще всілякі корисні функції. І наповнення вашого застосунку, повинно їм відповідати;
- для тестування краще використовувати мобільні емулятори з різним розширенням та версіями ОС.

Модульне тестування (Unit Testing) – Це стосується підходу до тестування програмного забезпечення, коли кожен окремий модуль коду тестується окремо. Для тестування нашого мобільного застосунку кожен раз ми використовували різні мобільні емулятори та писали Unit Tests.

При тестуванні нашого застосунку ми використовували наступні методи [9]:

- Manual testing – ручне тестування;
- Unit tests – модульне тестування;
- Security testing – тестування безпеки;

- Load testing – навантажувальне тестування;
- Interface tests – тестування інтерфейсу.

Всі тести відбувались на емуляціях реальних смартфонів, які можна завантажити в середовищі розробки Android Studio. Ці тестування показали хороший результат, на різних смартфонах, версіях Android та API.

ВИСНОВКИ

У ході виконання даної дипломної роботи розглянуто всі аспекти у розробці мобільного застосунку.

Було проведено порівняльний аналіз вже існуючих платформ і виявлено їх основні недоліки. На цьому етапі виникли деякі труднощі, адже, як виявилось, схожих застосунків наразі досить небагато.

Розглянуто підходи та необхідний інструментарій, на основі яких ґрунтувалася уся розробка програмного продукту. Також, важливо зазначити, що було продумано усі аспекти БД, сплановано, як система буде поводити себе із користувачами. Оскільки ми маємо користувачів з трьома різними ролями, то для кожної ролі система має надавати певний функціонал, та по різному реагувати на їхні дії.

При тестуванні застосунку, було виявлено, що ніякі компоненти програми не були «поламані» чи втрачені, всі тестування пройшли декілька етапів та закінчились успішно. Після розробки серверної частини вона, теж була протестована.

Даний програмний продукт є досить актуальним, через незначну кількість схожих на ринку. Він є зручним та легким, з простим інтуїтивно зрозумілим інтерфейсом. Завдяки цим перевагам, даний застосунок може з успіхом використовуватись продавцями товарів і їх клієнтами, як основна мобільна платформа для ведення бізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ткачук В.М. Алгоритми і структура даних: Навчальний посібник / В.М. Ткачук. – Івано-Франківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2016. – 286 с.;
2. Мартин Р. Чиста архітектура: Мистецтво створення програмного забезпечення / пер. з англ. І. Бондар-Терещенко. – Харків: вид-во «Фабула», 2019. – 368 с.;
3. Martin Fowler «UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition». Print2print, 2016. – 208 p.;
4. Funk M., Zhang Yu «Coding Art: A Guide to Unlocking Your Creativity with the Processing Language and p5.js in Four Simple Steps». Apress; 2nd edition, 2023. – 350 p.;
5. Давидов М.В. Програмне забезпечення мобільних пристроїв : навч. посібник / М.В. Давидов, А.Б. Демчук, О.В. Лозинська; М-во освіти і науки України, Нац. ун-т «Львівська політехніка». – Львів: Новий Світ-2000, 2021. – 217 с.;
6. Карпенко М.Ю. Технології створення програмних продуктів та інформаційних систем: навч. посібник / М. Ю. Карпенко, Н. О. Манакова, І. О. Гавриленко; Харків. Нац. ун-т міськ. госп-ва ім. О.М. Бекетова. – Харків: ХНУМГ ім. О.М. Бекетова, 2017. – 93 с.;
7. Дворецький М.Л. Розробка мобільних застосунків для OS Android : навч. посіб. / М.Л. Дворецький, Ю.О. Нездолій, С.В. Дворецька, І.О. Кандиба – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. – 140 с.;
8. Smyth N. Android Studio 4.0 Development Essentials – Java Edition / N. Smyth. – Payload Media, Inc., 2020. – 770 p.;
9. Panagiotis L. «Introduction to Software Testing. A Practical Guide to Testing, Design, Automation, and Execution». Apress, 2023. – 232 p.;
10. Yablonski J. «Laws of UX. Using Psychology to Design Better Products & Services». O'Reilly Media, Inc., 2020. – 150 p.;

11. Number of internet and social media users worldwide [Електронний ресурс] – Режим доступу: URL: https://www.statista.com/statistics/617136/digital-population-worldwide/ (дата звернення: 06.05.2024).
12. Mobile Operating System Market Share Worldwide [Електронний ресурс] – Режим доступу: URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата звернення: 03.06.2024).
13. Flutter for Android developers [Електронний ресурс] – Режим доступу: URL: https://docs.flutter.dev/get-started/learn-more (дата звернення: 20.05.2024).
14. Build great products [Електронний ресурс] – Режим доступу: URL: https://www.figma.com/design/ (дата звернення: 21.05.2024).
15. App Development Solutions [Електронний ресурс] – Режим доступу: URL: https://firebase.google.com/ (дата звернення: 15.05.2024).
16. Android Studio editor. Project structure [Електронний ресурс] – Режим доступу: URL: https://developer.android.com/studio/ (дата звернення: 12.05.2024).
17. Building Android Apps Using Android Studio [Електронний ресурс] – Режим доступу: URL: https://www.geeksforgeeks.org/android-studio-tutorial/ (дата звернення: 12.05.2024).
18. Визначення бази даних [Електронний ресурс] – Режим доступу: URL: https://uk.wikipedia.org/wiki/База_даних (дата звернення: 07.05.2024).
19. Особливості тестування додатків на мобільних пристроях [Електронний ресурс] – Режим доступу: URL: https://training.qatestlab.com/blog/technical-articles/testing-mobile-devices/ (дата звернення: 11.05.2024).

ДОДАТКИ

cart.dart – нижче наведений код реалізує управління кошиком покупок для програми

```
import 'package:flutter/foundation.dart';

class CartItem {
  final String id;
  final String title;
  final int quantity;
  final double price;

  CartItem({
    @required this.id,
    @required this.title,
    @required this.quantity,
    @required this.price,
  });
}

class Cart with ChangeNotifier {
  Map<String, CartItem> _items = {};

  Map<String, CartItem> get items {
    return {..._items};
  }

  int get itemCount {
    return _items.length;
  }

  double get totalAmount {
    var total = 0.0;
    _items.forEach((key, cartItem) {
      total += cartItem.price * cartItem.quantity;
```

```
  });
  return total;
}

void addItem(
  String productId,
  double price,
  String title,
) {
  if (_items.containsKey(productId)) {
    // change quantity...
    _items.update(
      productId,
      (existingCartItem) => CartItem(
        id: existingCartItem.id,
        title: existingCartItem.title,
        price: existingCartItem.price,
        quantity: existingCartItem.quantity + 1,
      ),
    );
  } else {
    _items.putIfAbsent(
      productId,
      () => CartItem(
        id: DateTime.now().toString(),
        title: title,
        price: price,
        quantity: 1,
      ),
    );
  }
  notifyListeners();
}

void removeItem(String productId) {
```

```
    _items.remove(productId);
    notifyListeners();
  }

  void removeSingleItem(String productId) {
    if (!_items.containsKey(productId)) {
      return;
    }
    if (_items[productId].quantity > 1) {
      _items.update(
          productId,
          (existingCartItem) => CartItem(
                id: existingCartItem.id,
                title: existingCartItem.title,
                price: existingCartItem.price,
                quantity: existingCartItem.quantity - 1,
              ));
    } else {
      _items.remove(productId);
    }
    notifyListeners();
  }

  void clear() {
    _items = {};
    notifyListeners();
  }
}
```

orders.dart – нижче наведений код реалізує управління замовленнями для інтернет-магазину. Клас Orders дозволяє отримувати замовлення з віддаленого сервера, додавати нові замовлення та зберігати їх у локальному списку.

```
import 'dart:convert';
```

```
import 'package:flutter/foundation.dart';
import 'package:http/http.dart' as http;

import './cart.dart';

class OrderItem {
  final String id;
  final double amount;
  final List<CartItem> products;
  final DateTime dateTime;

  OrderItem({
    @required this.id,
    @required this.amount,
    @required this.products,
    @required this.dateTime,
  });
}

class Orders with ChangeNotifier {
  List<OrderItem> _orders = [];
  final String authToken;
  final String userId;

  Orders(this.authToken, this.userId, this._orders);

  List<OrderItem> get orders {
    return [..._orders];
  }

  Future<void> fetchAndSetOrders() async {
    final url =
        'https://shop-49ff6-default-
rtdb.firebaseio.com/orders/$userId.json?auth=$authToken';
```

```
    final response = await http.get(Uri.parse(url));
    final List<OrderItem> loadedOrders = [];
    final extractedData = json.decode(response.body) as
Map<String, dynamic>;
    if (extractedData == null) {
      return;
    }
    extractedData.forEach((orderId, orderData) {
      loadedOrders.add(
        OrderItem(
          id: orderId,
          amount: orderData['amount'],
          dateTime: DateTime.parse(orderData['dateTime']),
          products: (orderData['products'] as List<dynamic>)
              .map(
                (item) => CartItem(
                  id: item['id'],
                  price: item['price'],
                  quantity: item['quantity'],
                  title: item['title'],
                ),
              )
              .toList(),
        ),
      );
    });
    _orders = loadedOrders.reversed.toList();
    notifyListeners();
  }

  Future<void> addOrder(List<CartItem> cartProducts, double total)
async {
    final url =
        'https://shop-49ff6-default-
rtdb.firebaseio.com/orders/$userId.json?auth=$authToken';
```

```
    final timestamp = DateTime.now();
    final response = await http.post(
      Uri.parse(url),
      body: json.encode({
        'amount': total,
        'dateTime': timestamp.toIso8601String(),
        'products': cartProducts
            .map((cp) => {
                  'id': cp.id,
                  'title': cp.title,
                  'quantity': cp.quantity,
                  'price': cp.price,
                })
            .toList(),
      }),
    );
    _orders.insert(
      0,
      OrderItem(
        id: json.decode(response.body)['name'],
        amount: total,
        dateTime: timestamp,
        products: cartProducts,
      ),
    );
    notifyListeners();
  }
}
```

product.dart – нижче наведений код реалізує управління продуктами для інтернет-магазину

```
import 'dart:convert';

import 'package:flutter/foundation.dart';
```

```
import 'package:http/http.dart' as http;

class Product with ChangeNotifier {
  final String id;
  final String title;
  final String description;
  final double price;
  final String imageUrl;
  bool isFavorite;

  Product({
    @required this.id,
    @required this.title,
    @required this.description,
    @required this.price,
    @required this.imageUrl,
    this.isFavorite = false,
  });

  void _setFavValue(bool newValue) {
    isFavorite = newValue;
    notifyListeners();
  }

  Future<void> toggleFavoriteStatus(String token, String userId)
async {
    final oldStatus = isFavorite;
    isFavorite = !isFavorite;
    notifyListeners();
    final url =
        'https://shop-49ff6-default-
rtdb.firebaseio.com/userFavorites/$userId/$id.json?auth=$token';
    try {
      final response = await http.put(
        Uri.parse(url),
```

```
        body: json.encode(
          isFavorite,
        ),
      );
      if (response.statusCode >= 400) {
        _setFavValue(oldStatus);
      }
    } catch (error) {
      _setFavValue(oldStatus);
    }
  }
}
```

products.dart – нижче наведений код реалізує управління продуктами для інтернет-магазину, забезпечуючи функціонал для отримання, додавання, оновлення та видалення продуктів.

```
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

import '../models/http_exception.dart';
import './product.dart';

class Products with ChangeNotifier {
  List<Product> _items = [
  ];
  final String authToken;
  final String userId;

  Products(this.authToken, this.userId, this._items);

  List<Product> get items {
```

```
    return [..._items];
  }

  List<Product> get favoriteItems {
    return _items.where((prodItem) =>
prodItem.isFavorite).toList();
  }

  Product findById(String id) {
    return _items.firstWhere((prod) => prod.id == id);
  }

  Future<void> fetchAndSetProducts([bool filterByUser = false])
async {
    final filterString =
        filterByUser ? 'orderBy="creatorId"&equalTo="$userId"' :
'';
    var url =
        'https://shop-49ff6-default-
rtdb.firebaseio.com/products.json?auth=$authToken&$filterString';
    try {
      final response = await http.get(Uri.parse(url));
      final extractedData = json.decode(response.body) as
Map<String, dynamic>;
      if (extractedData == null) {
        return;
      }
      url =
          'https://shop-49ff6-default-
rtdb.firebaseio.com/userFavorites/$userId.json?auth=$authToken';
      final favoriteResponse = await http.get(Uri.parse(url));
      final favoriteData = json.decode(favoriteResponse.body);
      final List<Product> loadedProducts = [];
      extractedData.forEach((prodId, prodData) {
        loadedProducts.add(Product(
```

```
          id: prodId,
          title: prodData['title'],
          description: prodData['description'],
          price: prodData['price'],
          isFavorite:
              favoriteData == null ? false : favoriteData[prodId]
?? false,
          imageUrl: prodData['imageUrl'],
        ));
      });
      _items = loadedProducts;
      notifyListeners();
    } catch (error) {
      throw (error);
    }
  }

  Future<void> addProduct(Product product) async {
    final url =
        'https://shop-49ff6-default-
rtdb.firebaseio.com/products.json?auth=$authToken';
    try {
      final response = await http.post(
        Uri.parse(url),
        body: json.encode({
          'title': product.title,
          'description': product.description,
          'imageUrl': product.imageUrl,
          'price': product.price,
          'creatorId': userId,
        }),
      );
      final newProduct = Product(
        title: product.title,
        description: product.description,
```

```
        price: product.price,
        imageUrl: product.imageUrl,
        id: json.decode(response.body)['name'],
      );
      _items.add(newProduct);
      notifyListeners();
    } catch (error) {
      print(error);
      throw error;
    }
  }

  Future<void> updateProduct(String id, Product newProduct) async
{
    final prodIndex = _items.indexWhere((prod) => prod.id == id);
    if (prodIndex >= 0) {
      final url =
          'https://shop-49ff6-default-
rtdb.firebaseio.com/products/$id.json?auth=$authToken';
      await http.patch(Uri.parse(url),
          body: json.encode({
            'title': newProduct.title,
            'description': newProduct.description,
            'imageUrl': newProduct.imageUrl,
            'price': newProduct.price
          }));
      _items[prodIndex] = newProduct;
      notifyListeners();
    } else {
      print('...');
    }
  }

  Future<void> deleteProduct(String id) async {
    final url =
```

```
        'https://shop-49ff6-default-
rtdb.firebaseio.com/products/$id.json?auth=$authToken';
    final existingProductIndex = _items.indexWhere((prod) =>
prod.id == id);
    var existingProduct = _items[existingProductIndex];
    _items.removeAt(existingProductIndex);
    notifyListeners();
    final response = await http.delete(Uri.parse(url));
    if (response.statusCode >= 400) {
      _items.insert(existingProductIndex, existingProduct);
      notifyListeners();
      throw HttpException('Could not delete product.');
    }
    existingProduct = null;
  }
}
```