МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА E-COMMERCE СИСТЕМИ З ВИКОРИСТАННЯМ FIREBASE ПЛАТФОРМИ ТА БІБЛІОТЕКИ REACT»**

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

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

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Рудакову Владиславу Юрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка E-Commerce системи з використанням Firebase платформи та бібліотеки React

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

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

2. Строк подання студентом роботи 01.06.2026 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та обґрунтування вибору технологій.
2. Проєктування архітектури та бази даних застосунку.
3. Тестування та підготовка до впровадження програмного продукту.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	19.12.2025	
2.	Збір вихідних даних.	15.01.2026	
3.	Обробка методичних та теоретичних	23.01.2026	
	джерел.		
4.	Розробка першого та другого розділу.	09.03.2026	
5.	Розробка третього розділу.	10.04.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2026	

Студент _____ В.Ю. Рудаков
(підпис) (ініціали та прізвище)

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

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка E-Commerce системи з використанням Firebase платформи та бібліотеки React.»: 45 с., 7 рис., 1 табл., 11 джерел.

БАЗА ДАНИХ, ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН, КРОСПЛАТФОРМНА РОЗРОБКА, МОБІЛЬНИЙ ЗАСТОСУНОК, CLOUD FIRESTORE, FIREBASE, REACT NATIVE.

Об’єкт дослідження – процес проєктування та розробки кросплатформних мобільних застосунків для сфери електронної комерції.

Мета роботи – розробка швидкого та зручного кросплатформного мобільного застосунку для інтернет-магазину взуття «Sunjesus shop» з метою автоматизації процесів вибору товарів, оформлення замовлень та підвищення лояльності клієнтів.

Методи дослідження – методи системного та об’єктно-орієнтованого аналізу, моделювання нереляційних (NoSQL) баз даних, компонентно-орієнтований підхід до проєктування користувацьких інтерфейсів, методи функціонального та інтеграційного тестування програмного забезпечення.

Кваліфікаційну роботу присвячено розробці кросплатформного мобільного застосунку для роздрібної торгівлі взуттям на базі фреймворку React Native та хмарної платформи Firebase. Для управління даними використано денормалізовану базу Cloud Firestore.

У програмному продукті реалізовано повний цикл електронної комерції: захищену автентифікацію, клієнтську фільтрацію каталогу, управління кошиком та оформлення замовлень із автоматичною генерацією накладних (ТТН). Результати тестування підтверджують абсолютну стабільність застосунку та його готовність до комерційного впровадження.

SUMMARY

Bachelor’s qualifying paper “Development of the E-Commerce System Using Firebase Library and React Library”: 45 pages, 7 figures, 1 table, 11 references.

DATABASE, ELECTRONIC COMMERCE, ONLINE STORE, CROSS-PLATFORM DEVELOPMENT, MOBILE APPLICATION, CLOUD FIRESTORE, FIREBASE, REACT NATIVE.

The object of the study is the process of design and development of cross-platform mobile applications for the e-commerce sector.

The aim of the work is the development of a fast and user-friendly cross-platform mobile application for the online shoe store "Sunjesus shop" to automate the processes of selecting goods, processing orders, and increasing customer loyalty.

Research methods include methods of system and object-oriented analysis, non-relational (NoSQL) database modeling, a component-oriented approach to user interface design, and methods of functional and integration software testing.

The bachelor’s thesis is dedicated to the development of a cross-platform mobile application for retail shoe trading based on the React Native framework and the Firebase cloud platform. To manage the data, the denormalized Cloud Firestore database was used.

The software product implements a full e-commerce cycle: secure authentication, client-side catalog filtering, cart management, and order processing with automatic generation of waybills (TTN). The test results confirm the absolute stability of the application and its readiness for commercial implementation.

ЗМІСТ

Завдання на кваліфікаційну роботу……..…..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз предметної області та обґрунтування вибору технологій 9
1.1 Дослідження ринку електронної комерції та тенденцій розвитку мобільних застосунків 9
1.2 Порівняльний аналіз підходів до розробки мобільних застосунків .. 11
1.3 Аналіз існуючих аналогів та конкурентних рішень на ринку..... 13
1.4 Специфікація вимог до програмного забезпечення мобільного застосунку 16
2 Проєктування архітектури та бази даних застосунку 19
2.1 Проєктування архітектури навігації та маршрутизації застосунку ... 19
2.2 Організація передачі параметрів та управління станом маршрутів .. 22
2.3 Архітектурні патерни управління станом авторизації та захисту маршрутів..... 24
2.4 Реалізація потоку даних та візуальне моделювання процесів..... 25
2.5 Проєктування та програмна реалізація модуля кошика і процесу оформлення замовлення 28
2.6 Аналіз залежностей та конфігурація середовища розробки..... 32
2.7 Логічна та фізична модель даних інтернет-магазину 35
3 Тестування та підготовка до впровадження програмного продукту..... 37
3.1 Стратегія та види тестування мобільного застосунку 37
3.2 Розробка та виконання сценаріїв тестування (тест-кейсів)..... 39
Висновки 43
Перелік посилань..... 45

ВСТУП

Сучасна електронна комерція стрімко переходить на мобільні пристрої. Більшість споживачів віддають перевагу покупкам через смартфони завдяки їхній зручності та швидкості. Для роздрібної торгівлі взуттям наявність якісного мобільного застосунку стала критично важливою умовою для збереження конкурентоспроможності. Інтернет-магазин кросівок «Sunjesus shop» потребує сучасного мобільного клієнта із гнучкою фільтрацією товарів, зручним каталогом та кошиком для автоматизації продажів і підвищення лояльності клієнтів.

Розробка окремих нативних застосунків для iOS та Android вимагає значних ресурсів. Тому найбільш раціональним є використання кросплатформного фреймворку React Native та платформи Expo, які дозволяють створити швидкий застосунок з єдиною кодовою базою. Використання хмарної інфраструктури Firebase як готового бекенду вирішує проблему безпечної авторизації та збереження даних без необхідності розробки власних серверів, що робить обрану тему дослідження вкрай актуальною.

Об’єкт дослідження – процес проєктування та розробки мобільних застосунків для сфери електронної комерції.

Предмет дослідження – технології та інструментальні засоби створення кросплатформного застосунку на базі React Native та хмарної архітектури Firebase.

Мета роботи – розробка кросплатформного мобільного застосунку для інтернет-магазину взуття «Sunjesus shop» із зручним інтерфейсом для вибору товарів та оформлення замовлень.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

– проаналізувати вимоги до мобільного застосунку інтернет-магазину;
– обґрунтувати вибір стеку технологій (React Native, Expo, Firebase);

– спроєктувати архітектуру навігації та логіку взаємодії екранів;
– розробити структуру бази даних Cloud Firestore;
– створити графічний інтерфейс з урахуванням принципів UI/UX-дизайну;
– програмно реалізувати бізнес-логіку (каталог, фільтрація, кошик, імітація оплати);
– провести функціональне тестування застосунку.

Методи дослідження. У роботі використано методи системного та об’єктно-орієнтованого аналізу, моделювання NoSQL баз даних, компонентно-орієнтований підхід розробки інтерфейсів та методи функціонального тестування.

Практичне значення одержаних результатів полягає у створенні готового до використання мобільного застосунку для магазину «Sunjesus shop». Спроєктована база даних та архітектурні рішення можуть бути адаптовані для розробки інших проєктів у сфері мобільної комерції.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

1.1 Дослідження ринку електронної комерції та тенденцій розвитку мобільних застосунків

Розвиток глобального інформаційного суспільства та цифровізація економічних процесів призвели до радикальних змін у сферах роздрібної торгівлі та взаємодії між бізнесом і кінцевими споживачами. Електронна комерція (e-commerce), яка ще кілька десятиліть тому сприймалася як інноваційний та експериментальний канал дистриб'юції, сьогодні перетворилася на один із ключових драйверів світової торгівлі. Сучасний етап розвитку цього сектору характеризується появою та стрімким домінуванням мобільної комерції (m-commerce), що визначає нові стандарти розробки програмного забезпечення для комерційних структур.

Мобільна комерція є логічною еволюційною ланкою електронної комерції, яка передбачає проведення будь-яких фінансових та торговельних транзакцій за допомогою портативних бездротових пристроїв – смартфонів, планшетів, а також носимої електроніки.

Згідно зі статистичними дослідженнями аналітичних агенцій, понад дві третини загального обсягу глобального інтернет-трафіку припадає саме на мобільні пристрої. Це зумовлено повсюдним поширенням доступних смартфонів із високою обчислювальною здатністю, впровадженням високошвидкісних стандартів стільникового зв'язку (4G/LTE та 5G), а також зміною споживчої поведінки користувачів, які віддають перевагу концепції «купівлі на ходу» (on-the-go shopping).

Перехід від десктопних вебсайтів до мобільних рішень висуває нові вимоги до архітектури та функціональності програмного забезпечення. Адаптивні версії традиційних вебсторінок, попри свою універсальність, часто

не здатні забезпечити необхідний рівень швидкодії, плавності інтерфейсу та інтерактивності.

У контексті роздрібної торгівлі одягом та взуттям, зокрема у вузькоспеціалізованому сегменті спортивного та повсякденного взуття (кросівок), вимоги до мобільного застосунку стають ще більш специфічними. Ринок кросівок має унікальні особливості, пов’язані з наявністю обмежених серій товарів (релізів), високою динамікою зміни асортименту та критичною важливістю точного підбору параметрів товару, таких як розмір, колір та бренд. Цільова аудиторія інтернет-магазину «Sunjesus shop» складається переважно з молодих людей, які активно користуються сучасними гаджетами та мають високі очікування щодо якості цифрових продуктів.

Для забезпечення високої конверсії та утримання клієнтів мобільний застосунок інтернет-магазину взуття повинен вирішувати низку взаємопов’язаних завдань:

– забезпечення високої швидкості рендерингу графічних елементів;
– реалізація ефективної багатопараметричної системи фільтрації та пошуку;
– оптимізація та спрощення процесу оформлення замовлення (чекауту);
– створення прозорого механізму соціального доказу (social proof) у вигляді системи відгуків та оцінок користувачів.

Таким чином, розробка мобільного застосунку для інтернет-магазину «Sunjesus shop» є необхідною відповіддю на сучасні виклики ринку електронної комерції. Створення такого програмного продукту дозволить компанії автоматизувати процеси продажів, побудувати прямий і незалежний канал комунікації із клієнтами, знизити витрати на залучення нових покупців та забезпечити стійкі конкурентні переваги в довгостроковій перспективі. Обґрунтований вибір архітектурних рішень та технологічного стеку для реалізації цього завдання є предметом подальшого дослідження у рамках цього розділу.

1.2 Порівняльний аналіз підходів до розробки мобільних застосунків

Вибір архітектурного підходу та технологічної стратегії є фундаментальним етапом проєктування будь-якого сучасного програмного забезпечення, особливо у сфері мобільної електронної комерції. Помилки, допущені на цьому етапі, можуть призвести до суттєвого збільшення вартості підтримки продукту, уповільнення його масштабування та погіршення користувацького досвіду.

У сучасній індустрії розробки програмного забезпечення для мобільних платформ виділилися три основні підходи: нативна розробка (Native Development), кросплатформна розробка (Cross-Platform Development) та створення гібридних застосунків або прогресивних вебзастосунків (Progressive Web Applications – PWA).

Нативна розробка являє собою класичний підхід, за якого для кожної цільової операційної системи створюється окремий програмний продукт із використанням спеціалізованих мов програмування та середовищ розробки. Для платформи iOS технологічний стек включає мови Swift або Objective-C та середовище розробки Xcode, а для платформи Android – мови Kotlin або Java та середовище Android Studio.

Основними перевагами нативного підходу є:

– максимальна продуктивність та чуйність інтерфейсу;
– повний та миттєвий доступ до можливостей операційної системи;
– високий рівень інтеграції з екосистемою.

Однак нативна розробка має серйозні економічні та організаційні недоліки. Головним із них є необхідність паралельного ведення двох незалежних проєктів. Це вимагає подвоєння штату кваліфікованих розробників, ускладнює процес синхронізації бізнес-логіки й тестування, а також значно збільшує бюджет проєкту та час його виходу на ринок (Time-to-Market).

Гібридний підхід та концепція PWA ґрунтуються на використанні стандартних вебтехнологій (HTML5, CSS3, JavaScript). Застосунок фактично

працює всередині спеціалізованого системного контейнера (WebView), який відображає вебсторінку під виглядом мобільного інтерфейсу.

Незважаючи на економію, гібридні застосунки мають критичні недоліки для сфери e-commerce:

– низька продуктивність;
– обмежений доступ до апаратних функцій пристрою та локального сховища;
– проблеми з безпекою даних та відсутність повноцінного нативного відгуку інтерфейсу.

Кросплатформна розробка стала компромісним і найбільш прогресивним рішенням, що об’єднало переваги двох попередніх підходів. Вона дозволяє розробникам писати єдиний вихідний код для бізнес-логіки та інтерфейсу, який потім компілюється або інтерпретується в елементи, що працюють на обох операційних системах (iOS та Android). Провідними технологіями в цьому сегменті є фреймворки Flutter та React Native.

Фреймворк Flutter (розробка компанії Google) використовує мову програмування Dart і власний графічний рушій Skia/Impeller. Flutter не використовує нативні компоненти операційної системи, а самостійно відмальовує кожен піксель інтерфейсу на екрані. Це гарантує абсолютну ідентичність візуального відображення на будь-яких пристроях, але іноді ускладнює інтеграцію специфічних системних елементів.

Фреймворк React Native (розробка компанії Meta) базується на мові JavaScript та бібліотеці React [9]. На відміну від Flutter, React Native використовує концепцію «мостів» (bridges) або сучасні архітектурні інтерфейси (JSI), які зв’язують код на JavaScript із реальними нативними компонентами UI операційної системи [2]. Під час рендерингу кнопка або текстове поле React Native перетворюються на справжню нативну кнопку iOS або Android.

Для інтернет-магазину «Sunjesus shop» кросплатформний підхід на базі React Native є найбільш обґрунтованим і ефективним рішенням з таких причин:

- *економічна доцільність*: створення єдиної кодової бази дозволяє скоротити витрати на розробку та подальшу підтримку застосунку майже вдвічі порівняно з нативним підходом;
- *скорочення часу виходу на ринок*: єдина логіка роботи кошика, каталогу та системи замовлень дозволяє одночасно запустити застосунок на обох платформах, що критично важливо для динамічного взуттєвого бізнесу;
- *нативний користувацький досвід*: завдяки використанню справжніх нативних компонентів UI, покупці отримують плавний, чуйний інтерфейс, який не поступається за швидкістю роботи нативним застосункам і кардинально перевершує гібридні вебрішення.

Таким чином, проведення детального аналізу існуючих підходів до мобільної розробки дозволяє зробити висновок, що кросплатформна стратегія є оптимальним вибором для реалізації проєкту «Sunjesus shop», забезпечуючи ідеальний баланс між високою продуктивністю програмного продукту та ефективним використанням ресурсів.

1.3 Аналіз існуючих аналогів та конкурентних рішень на ринку

Перед початком проєктування власного програмного продукту критично важливим етапом є проведення детального аналізу існуючих рішень на ринку мобільної електронної комерції. Це дозволяє виявити типові патерни проєктування користувацького інтерфейсу, зрозуміти очікування цільової аудиторії, а також уникнути поширених архітектурних та функціональних помилок, які допускають конкуренти у сегменті роздрібної торгівлі взуттям.

Для аналізу було обрано три популярні мобільні застосунки, які мають найбільшу базу активних користувачів та репрезентують різні підходи до електронної комерції: офіційний застосунок глобального бренду «Nike», застосунок найбільшої української мультибрендової мережі взуття «Intertop» та мобільний клієнт популярного ритейлера спортивного взуття «Megasport».

Мобільний застосунок Nike – є еталонним прикладом високобюджетної нативної розробки. Застосунок демонструє максимальні можливості сучасних мобільних інтерфейсів.

Переваги: надзвичайно висока швидкість роботи, плавні анімації переходів, високоякісний візуальний контент та глибока персоналізація стрічки товарів на основі попередніх покупок.

Недоліки: застосунок є надзвичайно ресурсомістким і займає багато місця у фізичній пам’яті смартфона. Крім того, інтерфейс сильно перевантажений супутнім контентом (маркетинговими статтями, відеороликами, тренуваннями), що відволікає користувача від головної мети – швидкого пошуку та купівлі конкретної пари взуття у класичному каталозі.

Мобільний застосунок Intertop – є один із найпопулярніших ритейлерів на локальному українському ринку, що пропонує величезний асортимент взуття різних брендів.

Переваги: наявність гнучкої бонусної системи, детальні розмірні сітки для кожного бренду, інтеграція з локальними службами доставки (зокрема Новою Поштою) та можливість самовивозу з офлайн-магазинів.

Недоліки: застосунок має ознаки гібридної розробки (використання технології WebView для рендерингу деяких сторінок), що негативно впливає на загальну швидкість роботи. Система багатокритеріальної фільтрації товарів працює із відчутними затримками через постійні звернення до сервера. Крім того, процес оформлення замовлення вимагає обов’язкового проходження складної багатоетапної валідації, що збільшує відсоток покинутих кошиків.

Мобільний застосунок Megasport – є класичний представник роздрібної мережі, орієнтований на масового споживача спортивного та повсякденного взуття.

Переваги: інтуїтивно зрозуміла та звична структура каталогу, яскраве виділення акційних пропозицій та знижок, проста система навігації нижньою панеллю вкладок.

Недоліки: застарілий дизайн інтерфейсу (UI), який не використовує сучасні гайдлайни операційних систем. Відсутність асинхронного

завантаження даних призводить до того, що під час застосування фільтрів або сортування екран повністю перезавантажується. Також відсутня зручна клієнтська валідація полів під час введення контактних даних, що призводить до помилок вже на етапі відправки замовлення на сервер.

Проаналізувавши наведені аналоги, можна сформувати чіткі вимоги до розроблюваного мобільного застосунку класичного інтернет-магазину взуття «Sunjesus shop».

Програмний продукт повинен поєднувати високу продуктивність та плавність інтерфейсів, але при цьому залишатися мінімалістичним, легким та сфокусованим виключно на процесі продажу, без зайвого інформаційного «шуму».

На відміну від застосунків Intertop та Megasport, система багатокритеріальної фільтрації у розроблюваному застосунку має виконуватися виключно на стороні мобільного клієнта (in-memory filtering) за допомогою заздалегідь завантаженого масиву даних. Це забезпечить миттєву реакцію інтерфейсу на зміну параметрів пошуку (бренду, розміру чи ціни). Процес оформлення замовлення (чекаут) має бути прозорим, інкапсульованим у зручні модальні вікна та містити сувору валідацію полів регулярними виразами безпосередньо під час введення тексту користувачем. Реалізація цих архітектурних концепцій за допомогою фреймворку React Native дозволить створити конкурентоспроможний інструмент електронної комерції, який перевершуватиме існуючі гібридні рішення за швидкістю та рівнем користувацького досвіду.

Таким чином, проведення детального аналізу існуючих підходів до мобільної розробки дозволяє зробити висновок, що кросплатформна стратегія є оптимальним вибором для реалізації проєкту «Sunjesus shop», забезпечуючи ідеальний баланс між високою продуктивністю програмного продукту та ефективним використанням ресурсів.

1.4 Специфікація вимог до програмного забезпечення мобільного застосунку

Успішна реалізація будь-якого програмного продукту вимагає чіткого визначення та формалізації вимог до нього ще на етапі проєктування. Специфікація вимог до програмного забезпечення (Software Requirements Specification, SRS) є фундаментальним документом, який описує очікувану поведінку мобільного застосунку інтернет-магазину «Sunjesus shop», його взаємодію з користувачами та хмарною інфраструктурою Firebase, а також визначає критерії якості кінцевого продукту. Вимоги до розроблюваної системи класифікуються на дві великі категорії: функціональні та нефункціональні.

Функціональні вимоги визначають безпосередню поведінку системи, тобто набір функцій та бізнес-процесів, які мобільний застосунок повинен виконувати у відповідь на дії користувача. Оскільки головною метою застосунку є забезпечення повного циклу роздрібного продажу взуття (кросівок), функціональні вимоги охоплюють кілька ключових модулів: систему автентифікації, управління каталогом, модуль кошика та систему оформлення замовлень.

Вимоги до підсистеми автентифікації та управління профілем передбачають, що застосунок повинен надавати можливість реєстрації нових клієнтів шляхом введення електронної пошти, номера телефону, імені та надійного пароля. Система повинна інтегруватися з сервісом Firebase Authentication для забезпечення безпечного входу, виходу та підтримки активної сесії користувача навіть після перезапуску застосунку. Крім того, авторизований користувач повинен мати доступ до особистого кабінету (профілю), де відображатиметься історія його попередніх замовлень.

Вимоги до підсистеми каталогу та фільтрації товарів є критично важливими для інтернет-магазину. Застосунок повинен асинхронно завантажувати з бази даних Cloud Firestore та відображати повний список

доступних моделей кросівок у вигляді адаптивної сітки карток. Кожна картка товару має містити фотографію, назву бренду, найменування моделі та актуальну ціну. Крім загального списку, система повинна надавати розширений функціонал клієнтської фільтрації: користувач повинен мати можливість сортувати товари за брендом, обирати потрібний розмір та встановлювати обмеження за ціною.

Важливою функціональною вимогою є миттєве оновлення списку товарів під час застосування фільтрів без необхідності повного перезавантаження екрана.

Вимоги до підсистеми кошика та чекауту (оформлення замовлення) визначають, що застосунок повинен дозволяти користувачеві додавати обрані кросівки певного розміру до віртуального кошика. Кошик повинен автоматично підраховувати загальну вартість обраних товарів. Під час переходу до оформлення замовлення система повинна вимагати введення контактних даних для доставки (ПІБ, місто, номер відділення логістичної компанії).

Також функціонал має передбачати вибір методу оплати (готівкою при отриманні або імітація оплати банківською карткою). Після підтвердження покупки застосунок повинен автоматично згенерувати номер товарно-транспортної накладної (ТТН), зберегти всі деталі транзакції у колекції замовлень бази даних та очистити поточний кошик клієнта.

Крім функціональних вимог, для забезпечення високої якості програмного продукту необхідно дотримуватися нефункціональних вимог. Ці вимоги описують атрибути якості системи, такі як продуктивність, масштабованість, безпека, кросплатформенність та зручність використання (юзабіліті).

Вимоги до кросплатформенності та інтерфейсу користувача (UI/UX) полягають у тому, що застосунок має компілюватися та стабільно працювати на обох провідних мобільних операційних системах — iOS та Android, маючи єдину кодову базу.

Використання фреймворку React Native повинно забезпечити нативний відгук інтерфейсу: плавні переходи між екранами за допомогою React Navigation, відсутність затримок під час прокручування довгих списків (завдяки використанню компонента FlatList) та коректне відображення контенту з урахуванням безпечних зон екрана (Safe Area) на сучасних смартфонах.

Вимоги до продуктивності та надійності передбачають оптимізацію мережевих запитів [11]. Застосунок не повинен завантажувати надлишкові дані; натомість має використовуватися механізм асинхронного читання документів з Cloud Firestore.

Вимоги до безпеки даних є пріоритетними при проєктуванні систем електронної комерції [7]. Мобільний клієнт не повинен локально зберігати у відкритому вигляді конфіденційні платіжні дані користувачів або їхні паролі. Усі процеси авторизації та управління токенами доступу повинні повністю делегуватися захищеним механізмам Firebase Authentication.

Таким чином, детальна специфікація функціональних та нефункціональних вимог створює чіткий вектор для подальшого проєктування архітектури системи та безпосереднього написання програмного коду. Відповідність розробленого мобільного інтернет-магазину «Sunjesus shop» усім переліченим вище вимогам гарантуватиме створення стабільного, безпечного та комерційно ефективного програмного продукту.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА БАЗИ ДАНИХ ЗАСТОСУНКУ

Будь-який процес розробки надійного програмного продукту починається зі стадії проєктування його архітектури. Для мобільного застосунку електронної комерції ключовими аспектами є організація системи навігації для забезпечення інтуїтивно зрозумілого користувацького досвіду (UX) та побудова ефективної структури збереження даних. Оскільки розробка ведеться з використанням Serverless-архітектури на базі Firebase, проєктування бази даних вимагає специфічного підходу, відмінного від класичних реляційних моделей. У цьому розділі детально розглядається логіка взаємодії екранів та структура колекцій NoSQL бази даних Firestore для інтернет-магазину «Sunjesus shop».

2.1 Проєктування архітектури навігації та маршрутизації застосунку

Маршрутизація (навігація) у мобільних застосунках є одним із найважливіших архітектурних елементів, що безпосередньо впливає на користувацький досвід (UX), продуктивність програмного продукту та загальну логіку взаємодії користувача з інтерфейсом.

На відміну від вебзастосунків, де навігація часто спирається на зміну URL-адрес у браузері, мобільні застосунки вимагають управління складними станами екранів, збереження контексту попередніх дій (стек викликів) та забезпечення плавних нативних анімацій переходу.

Для реалізації навігаційної архітектури у розроблюваному застосунку «Sunjesus shop» було обрано екосистему бібліотек React Navigation (версії 7.x) [8]. Цей вибір обґрунтований тим, що React Navigation є індустріальним стандартом для фреймворку React Native, забезпечуючи високу гнучкість

налаштувань та використання нативних оптимізацій операційних систем. У проєкті задіяно три ключові модулі цієї бібліотеки: базовий контейнер @react-navigation/native, модуль стекової навігації @react-navigation/native-stack та модуль вкладок @react-navigation/bottom-tabs. Важливим аспектом використання саме native-stack замість звичайного JS-стека є те, що він спирається на нативні примітиви (UINavigationController для iOS та Fragment для Android), що значно знижує навантаження на пам’ять пристрою та забезпечує ідеально плавні анімації.

Архітектура навігації застосунку побудована на принципі розподіленого доступу, який базується на глобальному стані автентифікації користувача. У кореневому компоненті застосунку (App.js) ініціалізується головний контейнер NavigationContainer.

Замість того, щоб дозволяти користувачеві вільно переміщуватися між усіма існуючими екранами, система використовує умовний рендеринг навігаторів. Логіка роботи контролюється слухачем подій onAuthStateChanged від Firebase Authentication. Під час запуску застосунку система перебуває у стані завантаження (loading = true), поки перевіряється наявність дійсного токена доступу. Після отримання відповіді від сервера рендериться один із двох незалежних навігаційних блоків: стек автентифікації (AuthStack) або головний робочий простір (MainTabs).

Стек автентифікації (AuthStack) ізолює неавторизованих користувачів від основного функціоналу магазину (каталогу, кошика, профілю). Цей стек містить два базових екрани:

– SignInScreen (екран авторизації) – виступає точкою входу за замовчуванням (дозволяє користувачеві ввести електронну пошту та пароль для входу);
– SignUpScreen (екран реєстрації) – доступний через перехід з екрана авторизації для створення нового облікового запису.

Після успішного проходження процедури реєстрації або входу, стан користувача у Firebase оновлюється, що автоматично викликає

перемальовування кореневого компонента. AuthStack знищується, і на його місце монтується MainTabs, що унеможливлює випадкове повернення на екран входу за допомогою системної кнопки «Назад».

Головний робочий простір (MainTabs) реалізований на базі компонента нижньої панелі вкладок (Bottom Tab Navigator). Такий патерн навігації є загальноприйнятим стандартом для застосунків електронної комерції, оскільки він дозволяє покупцеві миттєво перемикатися між ключовими розділами без втрати контексту. Панель містить п’ять основних вкладок:

- Home (головний екран) – містить інформацію про магазин, переваги та стрічку відгуків користувачів;
- Catalog (каталог) – основний розділ для пошуку та перегляду асортименту кросівок;
- Cart (кошик) – розділ для управління обраними для покупки товарами та переходу до оформлення замовлення;
- Favourite (обране) – список товарів, які користувач відмітив для майбутніх покупок;
- Profile (профіль) – особистий кабінет користувача з історією замовлень та налаштуваннями (Nested Navigation) [10].

Окремі вкладки нижньої панелі не є статичними екранами, а функціонують як самостійні стекові навігатори. Наприклад, вкладка каталогу (CatalogStack) об’єднує екран загального списку товарів (CatalogScreen) та екран детального перегляду конкретної моделі (SneakerDetailScreen). Коли користувач натискає на картку кросівок у каталозі, система викликає метод navigation.navigate(‘SneakerDetail’, { sneaker }), передаючи об’єкт вибраного товару як параметр маршруту (route.params). Новий екран накладається поверх списку товарів, утворюючи стек. При цьому нижня панель вкладок залишається активною, дозволяючи користувачеві здійснити перехід до кошика без необхідності попереднього повернення до загального каталогу.

Аналогічний підхід з вкладеним стеком реалізовано і для розділу профілю (ProfileStack), який забезпечує перехід від головного екрана

особистого кабінету (ProfileMain) до форми редагування персональних даних (EditProfile).

Важливим технічним рішенням при проєктуванні навігації стала інтеграція бібліотеки react-native-safe-area-context. Сучасні смартфони мають складну геометрію екранів, включаючи вирізи для камер (notch), заокруглені кути та нижні системні панелі жестів. Використання провайдера безпечних зон гарантує, що контент застосунку (кнопки, заголовки, списки товарів) ніколи не буде перекритий апаратними елементами пристрою, незалежно від моделі смартфона (як на платформі iOS, так і на Android).

Крім того, для оптимізації рендерингу використовується бібліотека react-native-screens, яка зв’язує екрани React Navigation безпосередньо з нативними контролерами представлення, звільняючи оперативну пам’ять від прихованих у стеку екранів.

2.2 Організація передачі параметрів та управління станом маршрутів

Важливим аспектом проєктування архітектури навігації є забезпечення надійного та ефективного механізму обміну даними між різними екранами застосунку.

У межах стекової навігації постійно виникає потреба передати контекст (дані) з одного екрана на інший для забезпечення безперервності бізнес-процесів. Бібліотека React Navigation реалізує цей механізм за допомогою спеціального об’єкта route, який містить властивість params.

У розробленому застосунку інтернет-магазину «Sunjesus shop» цей підхід найбільш яскраво демонструється при взаємодії користувача з каталогом товарів. Коли користувач перебуває на екрані CatalogScreen і натискає на компонент картки товару (SneakerCard), система не просто відкриває новий екран. Обробник події onPress викликає метод навігатора

Maps, якому передаються два аргументи: назва цільового маршруту (SneakerDetail) та об’єкт параметрів, що містить усі дані про обрану модель кросівок (бренд, назву, ціну, доступні розміри, опис та посилання на зображення).

Приймаючий екран (SneakerDetailScreen) вилучає переданий об’єкт безпосередньо з параметрів маршруту (route.params.sneaker). Такий архітектурний патерн має суттєву перевагу: він усуває необхідність робити додатковий мережевий запит до бази даних Firestore для отримання інформації про товар під час кожного переходу. Усі необхідні дані вже були завантажені на екрані каталогу і просто передаються в оперативній пам’яті до наступного екрана. Це критично важливо для забезпечення миттєвого рендерингу інтерфейсу та економії інтернет-трафіку користувача.

Крім об’єкта route, для управління маршрутизацією з глибоко вкладених компонентів (які не є безпосередньо екранами і не отримують пропси навігації за замовчуванням) активно використовується React-хук useNavigation. Він дозволяє будь-якому функціональному компоненту (наприклад, кнопці «Назад» у кастомному заголовку або модальному вікну CartOrderModal) отримати доступ до об’єкта навігації та викликати методи goBack() для повернення на попередній екран або Maps() для переходу до іншого розділу.

Окремої уваги заслуговує підхід до роботи з модальними вікнами у контексті навігації. Замість того, щоб реєструвати модальні вікна фільтрації (FilterModal) або оплати (PaymentModal) як повноцінні екрани у стеку MainTabs, вони реалізовані як компоненти, що монтуються поверх поточного екрана за допомогою локального стану (наприклад, const [isFilterVisible, setFilterVisible] = useState(false)).

Такий підхід запобігає перевантаженню історії навігації (навігаційного стека) зайвими кроками. Користувач сприймає відкриття фільтру як допоміжну дію на тому ж екрані, а закриття модального вікна не вимагає виклику системних функцій маршрутизації, що робить взаємодію більш природною та швидкою.

2.3 Архітектурні патерни управління станом авторизації та захисту маршрутів

У сучасних мобільних застосунках електронної комерції захист конфіденційних даних клієнта та запобігання несанкціонованому доступу до певних екранів (наприклад, профілю, історії замовлень або кошика) є критично важливим архітектурним завданням.

У класичних вебзастосунках ця проблема часто вирішується за допомогою серверних перенаправлень (редиректів), проте у середовищі React Native використовується патерн умовного рендерингу навігаційних стеків.

Процес маршрутизації, що базується на стані автентифікації користувача, програмно поділяється на такі послідовні етапи:

- ініціалізація базових компонентів застосунку та перевірка локального кешу пристрою на наявність збереженої сесії;
- виконання асинхронного запиту до сервісу Firebase Authentication для підтвердження валідності токена доступу [5];
- монтування стека авторизації (AuthStack) у разі відсутності активної сесії або закінчення терміну дії токена;
- рендеринг головного робочого простору (MainTabs) для верифікованих користувачів із наданням доступу до захищених екранів.

Розглянемо механізм асинхронної перевірки та ізоляції маршрутів детальніше. У кореневому компоненті App.js використовується хук useEffect, всередині якого ініціалізується метод auth.onAuthStateChanged. Цей слухач подій працює у фоновому режимі та миттєво реагує на будь-які зміни в об’єкті авторизації, отриманому від хмарного сервера. Поки триває перевірка стану, застосунок блокує будь-які переходи, відображаючи екран завантаження (loading state), що запобігає «блиманню» неавторизованих екранів перед користувачем.

Якщо користувач ініціює вихід із системи через екран профілю (викликаючи функцію signOut(auth)), слухач подій негайно фіксує зміну стану

користувача на null. Як наслідок, дерево компонентів React автоматично перемальовується: навігатор MainTabs повністю знищується, а на його місце монтується AuthStack. Цей підхід є найбільш безпечним, оскільки він повністю очищає історію навігації захищених екранів з оперативної пам’яті пристрою. Таким чином, після виходу з облікового запису користувач фізично не зможе скористатися системним жестом або кнопкою «Назад», щоб повернутися до екрана зі своїми персональними даними або кошиком.

2.4 Реалізація потоку даних та візуальне моделювання процесів

Ефективна архітектура мобільного застосунку вимагає не лише правильної організації переходів між екранами, але й оптимізованого потоку даних (Data Flow). Оскільки у розроблюваному застосунку «Sunjesus shop» не використовуються глобальні стейт-менеджери, управління станом базується на локальних станах компонентів (React хук useState) та прямих асинхронних запитах до хмарної бази даних Firebase Firestore.

Односпрямований потік даних є основоположним принципом React Native. Дані завжди передаються від батьківських компонентів до дочірніх через властивості (props). У контексті нашого інтернет-магазину життєвий цикл роботи з даними можна візуалізувати за допомогою блок-схеми. На рисунку 2.1 представлено загальну схему процесу взаємодії користувача з каталогом товарів.

Як видно з наведеної схеми, ключовим етапом є первинне завантаження колекції товарів. Щоб уникнути надмірної кількості мережевих запитів та знизити навантаження на базу даних, застосунок виконує читання каталогу лише один раз під час монтування компонента CatalogScreen. Отриманий масив документів зберігається у локальному стані.

Для реалізації цієї логіки в межах компонентно-орієнтованої архітектури фреймворку React Native використовується вбудований хук життєвого циклу useEffect.



Рисунок 2.1 – Блок-схема процесу взаємодії користувача з каталогом та кошиком

Завдяки передачі порожнього масиву залежностей як другого аргументу, цей хук функціонує як аналог методу componentDidMount у класичних класових компонентах, забезпечуючи виконання вкладеної асинхронної операції строго один раз під час первинного монтування та рендерингу екрана на пристрої користувача.

На рисунку 2.2 наведено уривок реального програмного коду застосунку, який відповідає за ініціалізацію з'єднання з хмарною інфраструктурою Firestore та асинхронне отримання повного списку кросівок під час завантаження екрана. Використання асинхронної конструкції async/await у поєднанні з блоком обробки винятків try...catch...finally дозволяє виконувати мережеві запити до NoSQL бази даних у фоновому потоці, абсолютно не навантажуючи головний графічний потік інтерфейсу (UI thread).

Розглянемо наведений фрагмент коду (див. рис. 2.2) детальніше. Асинхронна функція fetchSneakers обгорнута у блок try...catch для забезпечення надійного перехоплення та обробки можливих мережевих помилок. Метод getDocs виконує запит до серверів Firebase, звертаючись до колекції sneakers.

```
useEffect(() => {
   console.log('[CatalogScreen] Fetching sneakers on mount');
   fetchSneakers();
  }, []);

  const fetchSneakers = async () => {
   try {
    console.log('[CatalogScreen] Fetching sneakers from Firestore...');
    const querySnapshot = await getDocs(collection(db, 'sneakers'));
    const sneakersData = querySnapshot.docs.map(doc => ({
     id: doc.id,
     ...doc.data(),
    }));
    console.log('[CatalogScreen] Sneakers loaded:', sneakersData.length);

    const brands = [
     ...new Set(sneakersData.map(s => s.brand).filter(Boolean)),
    ];
    setAvailableBrands(brands);

    setSneakers(sneakersData);
    setFilteredSneakers(sneakersData);
   } catch (error) {
    console.error('[CatalogScreen] Error loading sneakers:', error);
   } finally {
    setLoading(false);
   }
  };
```

Рисунок 2.2 – Програмна реалізація завантаження каталогу товарів

Отриманий об'єкт querySnapshot трансформується за допомогою методу map: ідентифікатор документа (doc.id) об'єднується з його вмістом (doc.data()). Відмінною рисою даного фрагмента є динамічне формування списку доступних брендів. За допомогою об'єкта Set система автоматично вилучає лише унікальні значення брендів з масиву sneakersData та зберігає їх у стан availableBrands. Це дозволяє уникнути жорсткого прописування брендів у коді і робить систему фільтрів повністю гнучкою. Отриманий загальний масив об'єктів записується у два незалежні стани: sneakers (еталонний список) та

filteredSneakers (список для відображення). Блок finally гарантує, що індикатор завантаження (loading) буде вимкнено після завершення операції.

2.5 Проєктування та програмна реалізація модуля кошика і процесу оформлення замовлення

У будь-якому застосунку електронної комерції модуль кошика (Cart) та процес оформлення замовлення (Checkout) є критично важливими компонентами, від стабільності та зручності яких безпосередньо залежить рівень конверсії та загальний комерційний успіх інтернет-магазину. Під час проєктування цих модулів для застосунку «Sunjesus shop» основна увага приділялася оптимізації швидкодії, мінімізації кількості кроків для покупця та забезпеченню цілісності даних при асинхронній взаємодії з хмарною базою даних.

Оскільки як серверну інфраструктуру обрано NoSQL базу даних Firebase Cloud Firestore, архітектура збереження даних кошика кардинально відрізняється від класичних реляційних моделей (SQL) [1]. У реляційних базах даних для відображення кошика зазвичай використовується складний запит із застосуванням оператора JOIN, який об’єднує таблицю кошика з таблицею товарів.

У розробленому застосунку реалізовано підхід часткової нормалізації. Колекція cart не зберігає всю інформацію про товар (його назву, поточну ціну, зображення), оскільки ці параметри можуть змінитися на стороні сервера. Замість цього кожен документ у колекції cart виступає як зв’язуюча сутність. На рисунку 2.3 продемонстровано зовнішній вигляд екрана кошика, де відображаються ці агреговані дані.

Логіка роботи екрана кошика побудована на асинхронній гідратації (збагаченні) даних. Під час монтування компонента застосунок виконує первинний запит до колекції cart, фільтруючи документи за ідентифікатором

поточного авторизованого користувача (userId). Отримавши масив ідентифікаторів товарів, система ініціює цикл асинхронних запитів getDoc до колекції sneakers для кожного елемента кошика. Це гарантує, що під час перегляду кошика користувач завжди бачить найбільш актуальну ціну та наявність товару.

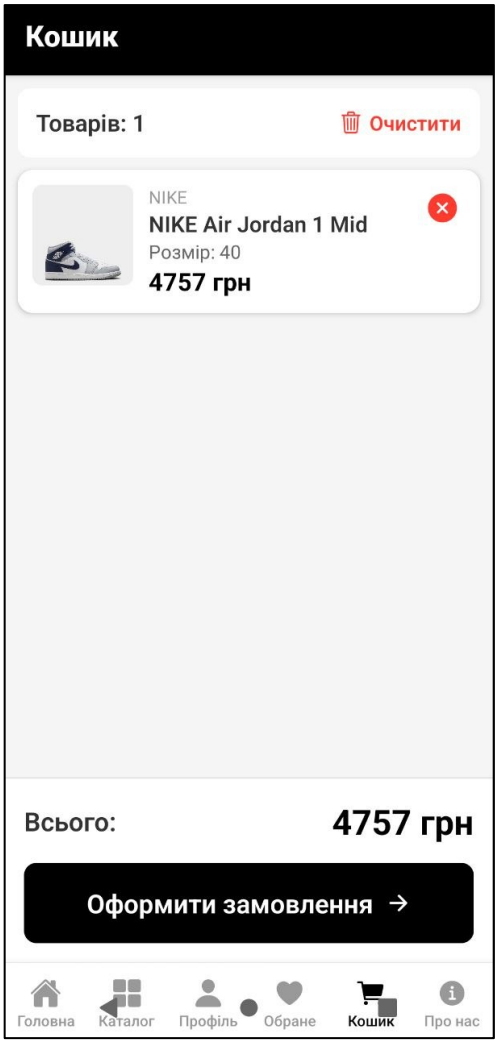


Рисунок 2.3 – Графічний інтерфейс екрана кошика користувача

Після перевірки вмісту кошика користувач переходить до найважливішого етапу – оформлення замовлення. У мобільному застосунку цей процес реалізовано за допомогою багатокрокового інтерфейсу, інкапсульованого у систему модальних вікон (CartOrderModal та PaymentModal). Такий підхід дозволяє утримувати фокус уваги покупця, не перевантажуючи стек навігації новими повноцінними екранами. Збір

контактних даних відбувається у першому модальному вікні, інтерфейс якого представлено на рисунку 2.4.

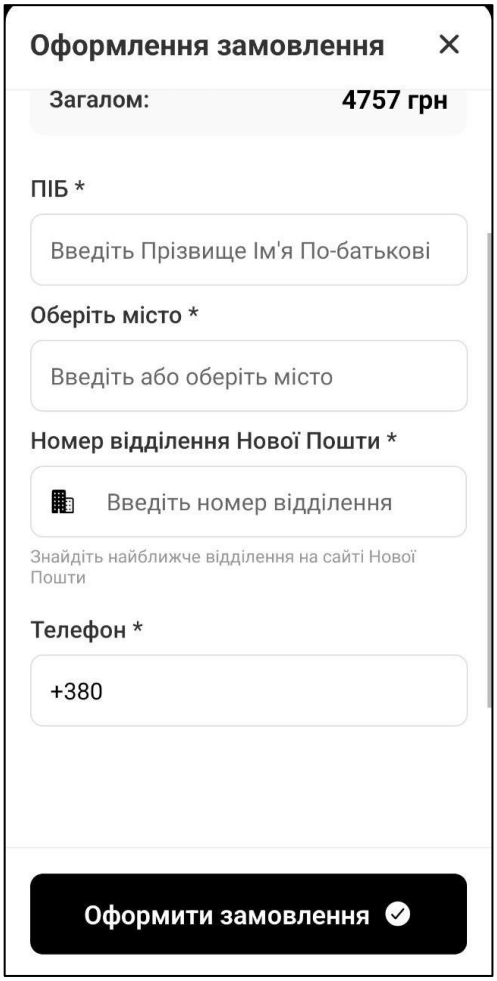


Рисунок 2.4 – Інтерфейс форми введення даних для доставки

Процес оформлення замовлення являє собою послідовний алгоритм взаємодії користувача з інтерфейсом, який починається зі збору контактних даних та детальної інформації про доставку (див. рис. 2.4). На цьому етапі всі текстові поля проходять сувору клієнтську валідацію, щоб унеможливити відправку некоректних запитів до бази даних.

Зокрема, введений номер телефону програмно перевіряється на сувору відповідність регулярному виразу формату +380XXXXXXXXX, що гарантує стандартизацію контактної інформації. Відразу після успішного проходження перевірки даних доставки первинне модальне вікно CartOrderModal автоматично закривається, і система миттєво ініціює рендеринг наступного

інтерфейсного вікна – PaymentModal, яке відповідає за фінансову частину транзакції (рис. 2.5). У межах цього етапу архітектура застосунку підтримує два незалежні сценарії: класичну оплату готівкою при отриманні (післяплату) та імітацію онлайн-оплати банківською карткою.

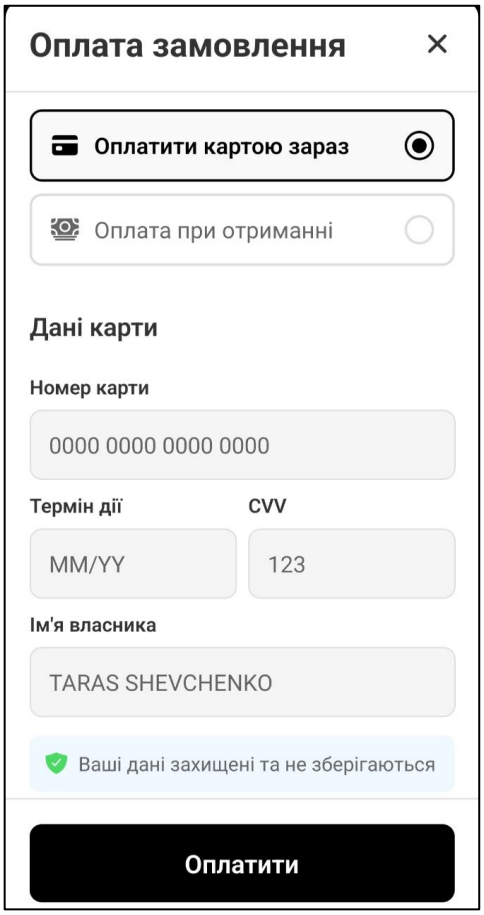


Рисунок 2.5 – Модальне вікно імітації платіжної системи

Етап фіналізації є найбільш складним з точки зору бізнес-логіки. Для забезпечення можливості подальшого відстеження статусу доставки застосунок містить спеціальну функцію generateTTN(), яка генерує випадковий 14-значний номер товарно-транспортної накладної, імітуючи інтеграцію з API логістичного оператора (Нової Пошти).

Після генерації ТТН формується комплексний об’єкт замовлення, який записується у колекцію orders. Цей об’єкт містить масив усіх товарів із кошика, загальну розраховану вартість, введені дані доставки та статус оплати. Статус оплати динамічно залежить від обраного методу: якщо користувач

імітував оплату карткою, статус встановлюється як paid (оплачено), якщо обрав готівку – pending (очікує на оплату).

Останнім кроком транзакції є виконання процедури очищення кошика. Застосунок ітерується по всіх ідентифікаторах документів, що належали поточному користувачеві у колекції cart, та викликає метод deleteDoc для кожного з них. Після успішного завершення користувач отримує візуальне підтвердження і автоматично повертається на екран каталогу.

2.6 Аналіз залежностей та конфігурація середовища розробки

Сучасна розробка мобільних застосунків на базі екосистеми JavaScript неможлива без використання менеджерів пакетів, які дозволяють керувати зовнішніми модулями, бібліотеками та конфігураціями проєкту. Центральним елементом архітектури розробленого застосунку (внутрішня системна назва проєкту footshop-mobile) є конфігураційний файл package.json. Він містить метадані проєкту, сценарії запуску (скрипти) та строго версіонований перелік усіх зовнішніх залежностей.

Для забезпечення модульності, масштабованості та надійності програмного продукту до проєкту було підключено низку спеціалізованих бібліотек. Згідно з маніфестом залежностей, інфраструктуру застосунку формують такі основні пакети:

- ядро системи складають бібліотеки react (версії 19.1.0) та react-native (версії 0.81.5), які забезпечують декларативний підхід до побудови інтерфейсів та рендеринг нативних компонентів для мобільних операційних систем;
- платформа expo (версії 54.0.31) та модуль expo-status-bar використовуються як основний інструментарій для спрощення процесу розробки, компіляції та візуального управління системним рядком стану (Status Bar) на пристрої [4];

- взаємодія з хмарним бекендом реалізована за допомогою офіційного SDK firebase (версії 12.8.0), що надає мобільному клієнту методи для прямого доступу до сервісів Authentication та Cloud Firestore;
- для управління маршрутизацією та станами екранів встановлено комплекс пакетів React Navigation 7-ї версії (@react-navigation/native, @react-navigation/native-stack та @react-navigation/bottom-tabs);
- оптимізація навігації та інтерфейсу на нативному рівні забезпечується модулями react-native-screens (версії 4.16.0) та react-native-safe-area-context (версії 5.6.2), які гарантують безпечне відображення контенту на пристроях із вирізами екрана та вивільняють оперативну пам’ять;
- розширення стандартного користувацького інтерфейсу реалізовано за допомогою бібліотеки @react-native-community/slider (версії 5.1.2), яка застосовується у компоненті фільтрації для плавного налаштування цінового діапазону;
- для локального збереження даних на пристрої користувача інтегровано модуль @react-native-async-storage/async-storage (версії 2.2.0), що забезпечує асинхронне кешування невеликих обсягів інформації.

Крім переліку залежностей, файл конфігурації містить набір командних скриптів (блок scripts), які автоматизують життєвий цикл розробки. Зокрема, базова команда expo start запускає локальний сервер розробки (Metro Bundler), який відстежує зміни у вихідному коді та оновлює інтерфейс.

Завдяки підтримці технології Fast Refresh будь-які модифікації в коді відображаються на екрані тестування миттєво й без втрати поточного стану сесії, що істотно оптимізує швидкість розробки компонентів.

Команди з розширеннями --android, --ios та --web дозволяють цілеспрямовано скомпілювати та розгорнути клієнт на відповідних платформах або в емуляторах, а команда --tunnel створює захищений тунель для доступу до локального сервера з будь-якої точки світу під час тестування

на реальних пристроях. Це дозволяє повністю абстрагуватися від складних низькорівневих конфігурацій нативного середовища кожної конкретної ОС та забезпечує уніфікований інтерфейс управління всіма етапами життєвого циклу проєкту. На рисунку 2.6 наведено повний лістинг конфігураційного файлу розробленого мобільного застосунку.

```
{
 "name": "footshop-mobile",
 "version": "1.0.0",
 "description": "Мобільний застосунок FootShop для iOS та Android",
 "main": "index.js",
 "scripts": {
  "start": "expo start",
  "android": "expo start --android",
  "ios": "expo start --ios",
  "web": "expo start --web",
  "tunnel": "expo start --tunnel"
 },
 "dependencies": {
  "@react-native-async-storage/async-storage": "^2.2.0",
  "@react-native-community/slider": "^5.1.2",
  "@react-navigation/bottom-tabs": "^7.9.1",
  "@react-navigation/native": "^7.1.27",
  "@react-navigation/native-stack": "^7.9.1",
  "expo": "~54.0.31",
  "expo-status-bar": "~3.0.9",
  "firebase": "^12.8.0",
  "react": "19.1.0",
```

Рисунок 2.6 – Конфігураційний файл package.json мобільного застосунку

Фіксація точних версій бібліотек (див. рис. 2.6) є критично важливою вимогою для проєктів електронної комерції. Завдяки строгому контролю версій (наприклад, використанню символу ^ для фіксації мажорних оновлень), гарантується, що застосунок збиратиметься ідентично як на локальному комп’ютері розробника, так і під час генерації фінальних дистрибутивів (APK для Android) у хмарних сервісах Expo Application Services (EAS) [3]. Це мінімізує ризик виникнення конфліктів між залежностями та підвищує загальну надійність програмного продукту перед релізом.

2.7 Логічна та фізична модель даних інтернет-магазину

Незважаючи на те, що базою даних проєкту виступає нереляційне (NoSQL) хмарне сховище Cloud Firestore, для забезпечення цілісності даних та коректної роботи бізнес-логіки застосунку була спроєктована строга логічна структура сутностей. Для візуалізації атрибутивного складу колекцій та кардинальності зв'язків між ними розроблено ER-діаграму (Entity-Relationship diagram) фізичної моделі даних.

На рисунку 2.7 наведено фінальну схему бази даних мобільного застосунку «Sunjesus shop», де чітко розмежовано ключові (Primary Key) та зовнішні (Foreign Key) атрибути зв'язку.

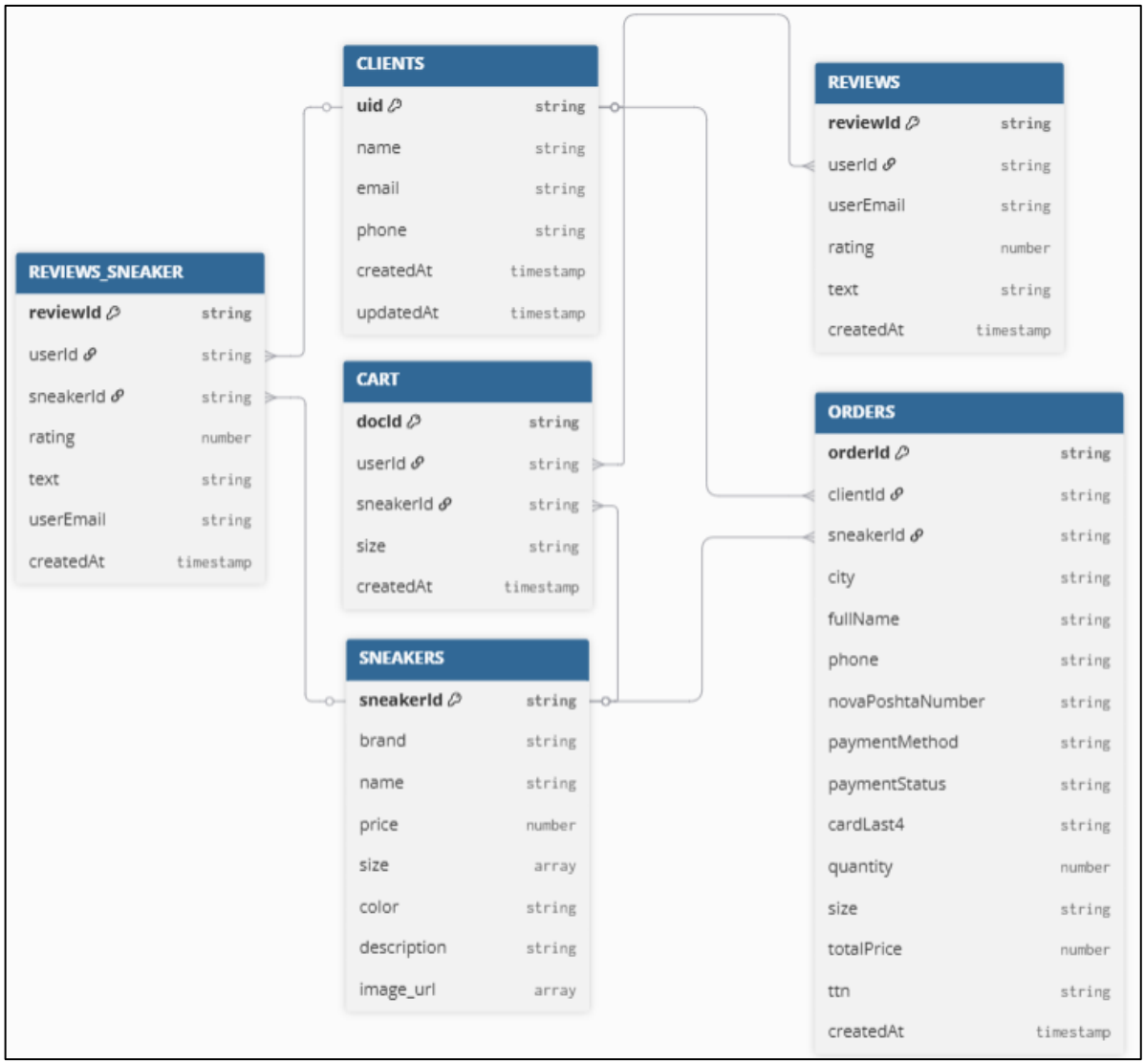


Рисунок 2.7 – ER-діаграма фізичної моделі даних та зв'язків застосунку

Аналіз наведеної діаграми (див. рис. 2.7) дозволяє виділити дві головні (батьківські) сутності, від яких залежать усі інші дані в системі:

– CLIENTS (клієнти);
– SNEAKERS (каталог кросівок).

Застосунок програмно підтримує цілісність наступних зв’язків:

– формування кошика (CART);
– управління замовленнями (ORDERS);
– система оцінювання.

Використання подібної денормалізованої моделі зі строгим контролем зовнішніх ідентифікаторів на стороні клієнтського коду (фронтенду) дозволяє ефективно використовувати можливості NoSQL-архітектури.

3 ТЕСТУВАННЯ ТА ПІДГОТОВКА ДО ВПРОВАДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

Розробка сучасного мобільного застосунку не обмежується лише етапами написання програмного коду та проєктування бази даних. Критично важливим кроком, який гарантує якість, стабільність та комерційну придатність кінцевого продукту, є етап комплексного тестування та підготовки до розгортання (впровадження). У третьому розділі кваліфікаційної роботи розглядаються стратегії забезпечення якості (Quality Assurance) для мобільного інтернет-магазину «Sunjesus shop», описуються методи функціонального тестування кросплатформного інтерфейсу, а також аналізуються процеси збірки проєкту для подальшої публікації на платформах цифрової дистрибуції.

3.1 Стратегія та види тестування мобільного застосунку

Тестування мобільного застосунку електронної комерції є багатовимірним та складним процесом, оскільки програмний продукт повинен коректно функціонувати на широкому спектрі пристроїв із різними розмірами екранів, версіями операційних систем та за умов нестабільного або повільного мережевого з’єднання. Для забезпечення найвищого рівня надійності інтернет-магазину «Sunjesus shop» було застосовано комплексний підхід до тестування, що включає кілька незалежних рівнів перевірки.

Наведемо основні види тестування, які було проведено в рамках проєкту.

Функціональне тестування (Functional Testing) – перевірка повної відповідності розробленого функціоналу початковим бізнес-вимогам. Зокрема, інтенсивно тестувалися процеси реєстрації та авторизації, логіка

додавання товарів до кошика, коректність роботи багатопараметричних фільтрів каталогу та алгоритм розрахунку фінальної вартості замовлення.

Фестування інтерфейсу користувача (UI/UX Testing) – аналіз коректності відображення візуальних елементів на різних роздільних здатностях екранів. Особлива увага приділялася перевірці безпечних зон (Safe Area) на сучасних смартфонах без фізичних кнопок навігації та коректному рендерингу компонента FlatList під час прокручування великого масиву карток кросівок [6].

Інтеграційне тестування (Integration Testing) – перевірка безперебійності взаємодії мобільного клієнта (фронтенду) з хмарною інфраструктурою Firebase. Оцінювалася швидкість та точність виконання базових CRUD-операцій (створення, читання, оновлення, видалення документів) у колекціях clients, cart та orders.

Тестування обробки виняткових ситуацій (Negative Testing) – штучне моделювання збоїв, таких як раптова втрата мережі, введення спеціальних символів чи некоректних даних у форми автентифікації та форми оформлення замовлення. Метою цього етапу була перевірка реакції системи та своєчасного виведення локалізованих повідомлень про помилки через компонент Toast.

Окремим і надзвичайно важливим етапом стало кросплатформне тестування (Cross-Platform Testing). Оскільки застосунок розроблено за допомогою фреймворку React Native, він має спільну кодову базу для операційних систем iOS та Android. Проте, через особливості рендерингу нативних компонентів, деякі елементи інтерфейсу можуть поводитися по-різному на цих платформах.

Тестування проводилося за допомогою офіційного клієнта Expo Go на фізичних мобільних пристроях (смартфонах під управлінням ОС Android 13+ та iOS 16+), а також із використанням програмних віртуальних середовищ (емуляторів Android Studio та симуляторів Xcode). Це дозволило виявити та своєчасно усунути дрібні візуальні дефекти (наприклад, специфічне обрізання тіней або неправильне позиціонування іконок на платформі Android),

забезпечивши повністю ідентичний користувацький досвід незалежно від обраного пристрою.

3.2 Розробка та виконання сценаріїв тестування (тест-кейсів)

Тестування програмного забезпечення у сфері електронної комерції вимагає підвищеної уваги до деталей, оскільки будь-яка помилка в логіці роботи кошика або системи оплати може призвести до прямих фінансових та репутаційних втрат для бізнесу. Саме тому процес перевірки якості базується на формалізованому підході з використанням заздалегідь підготовлених тестових сценаріїв.

Тест-кейс являє собою детальний, задокументований опис кроків, вихідних даних, передумов (preconditions) та очікуваного результату, необхідних для підтвердження того, що певна функція програмного продукту працює коректно і повністю відповідає початковим бізнес-вимогам.

Перед початком виконання тестових сценаріїв було підготовлено відповідне тестове середовище. Оскільки застосунок розроблено з використанням кросплатформного фреймворку React Native, тестування проводилося паралельно на двох апаратних платформах: фізичному смартфоні під управлінням операційної системи Android та пристрої на базі iOS, із використанням середовища Expo Go.

Крім того, для перевірки стійкості застосунку до різних умов експлуатації, тестові сценарії виконувалися як у стабільній локальній мережі Wi-Fi, так і в умовах імітації слабкого мобільного з’єднання (3G). Це дозволило оцінити ефективність роботи індикаторів завантаження (ActivityIndicator) та коректність обробки таймаутів при асинхронному зверненні до хмарної бази даних.

Важливим етапом підготовки до тестування стало формування валідних тестових даних (Test Data). Для уникнення конфліктів із реальними даними у базі Cloud Firestore було тимчасово створено окрему ізольовану колекцію з

тестовими товарами (кросівками), які містили як стандартні, так і граничні значення (наприклад, надзвичайно довгі назви брендів або нульову ціну для перевірки роботи повзунка фільтрації).

У сервісі Firebase Authentication було згенеровано декілька тестових облікових записів із різними станами: нові користувачі, користувачі з уже наповненим кошиком та клієнти з наявною історією замовлень. Такий підхід забезпечив ідемпотентність тестів – можливість їх багаторазового повторення з отриманням стабільних та ідентичних результатів.

Кожен розроблений тестовий сценарій має чітко регламентовану структуру. Він включає унікальний ідентифікатор (ID) для зручності відстеження, назву та мету тесту, послідовність кроків для відтворення (Steps to reproduce), а також очікуваний результат (Expected Result), який порівнюється з фактичним результатом (Actual Result) під час виконання. Якщо фактичний результат повністю збігається з очікуваним, тест-кейсу присвоюється статус «Успішно» (Passed); у разі виявлення розбіжностей фіксується дефект (Bug) із статусом «Провалено» (Failed), після чого він передається на етап зневадження (debugging) та подальшого регресійного тестування.

Окрему увагу під час тестування було приділено розмежуванню позитивних та негативних тестових сценаріїв. Позитивне тестування підтверджувало здатність системи виконувати свої основні функції за умови введення коректних даних (наприклад, успішне створення профілю або оформлення замовлення). Натомість негативне тестування було спрямоване на перевірку реакції застосунку на нетипові або помилкові дії користувача (введення невалідного формату пошти, спроба замовити товар без вибору розміру тощо). Це дозволило переконатися у надійності механізмів клієнтської валідації та відсутності критичних збоїв програми (крешів).

Процес ручного функціонального тестування охоплював усі ключові модулі застосунку «Sunjesus shop»: автентифікацію, роботу з каталогом, клієнтську фільтрацію, управління кошиком та імітацію платіжного шлюзу. У таблиці 3.1 наведено базовий набір функціональних тест-кейсів.

Таблиця 3.1 – Функціональні тестові сценарії (тест-кейси)

ID	Назва сценарію та кроки виконання	Очікуваний результат	Статус
TC-01	Реєстрація нового користувача 1. Відкрити застосунок, обрати екран SignUp. 2. Ввести валідні дані: ім’я, телефон, унікальний email та пароль. 3. Натиснути кнопку «Зареєструватися».	Створюється новий обліковий запис. Документ користувача записується у колекцію clients. Здійснюється перехід до MainTabs.	Успішно
TC-02	Обробка помилки входу (невірний пароль) 1. Відкрити екран SignIn. 2. Ввести існуючий email, але вказати навмисно хибний пароль. 3. Натиснути кнопку «Увійти».	Застосунок перехоплює помилку Firebase auth/wrong-password. На екрані з’являється Toast-повідомлення «Невірний пароль».	Успішно
TC-03	Робота багатокритеріального фільтра 1. Перейти на екран Catalog. 2. Відкрити FilterModal. 3. Обрати бренд «Nike», розмір «42», ціну від 2000 до 5000 грн. 4. Натиснути «Застосувати».	Каталог миттєво оновлюється. Відображаються лише ті картки товарів, які повністю відповідають усім трьом заданим умовам.	Успішно
TC-04	Додавання товару до кошика 1. Відкрити картку товару (SneakerDetail). 2. Обрати розмір зі списку доступних. 3. Натиснути «Додати в кошик».	Дані (userId, sneakerId, розмір) зберігаються у колекції cart. Кнопка змінює стан або з’являється підтвердження.	Успішно
TC-05	Оформлення замовлення (оплата при отриманні) 1. Перейти до CartScreen, натиснути «Оформити». 2. Ввести валідні ПІБ, телефон, місто, номер НП. 3. У PaymentModal обрати «Готівка».	Замовлення створюється в orders зі статусом оплати pending. Генерується ТТН. Кошик поточного користувача очищується.	Успішно
TC-06	Оформлення замовлення (картка) 1. Повторити кроки TC-05. 2. У PaymentModal обрати «Картка». 3. Ввести неповний номер картки (менше 16 цифр).	Клієнтська валідація блокує перехід. З’являється повідомлення про необхідність перевірити номер картки. Замовлення не створюється.	Успішно
TC-07	Вихід з облікового запису 1. Перейти на екран Profile. 2. Натиснути кнопку «Вийти». 3. Спробувати повернутися назад системним жестом.	Сесія руйнується. Користувач перенаправляється на AuthStack. Повернення до закритого профілю неможливе.	Успішно

Як видно з наведених результатів (див. табл. 3.1), усі критичні бізнес-процеси застосунку успішно пройшли перевірку та працюють згідно із заданими архітектурними вимогами. Виявлені під час попередніх ітерацій розробки незначні дефекти були оперативно виправлені до моменту проведення фінального тестування. Завдяки розділенню логіки на незалежні компоненти та використанню асинхронного перехоплення помилок, застосунок продемонстрував високу стабільність і готовність до впровадження.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи бакалавра успішно вирішено науково-прикладне завдання з розробки кросплатформного мобільного застосунку для інтернет-магазину взуття «Sunjesus shop». Поставлена мета досягнута повною мірою, а створений програмний продукт повністю готовий до комерційної експлуатації.

Під час дослідження предметної області обґрунтовано вибір кросплатформного підходу до розробки. Використання фреймворку React Native у поєднанні з інструментарієм Expo дозволило створити високопродуктивний клієнт із єдиною кодовою базою для iOS та Android. Для реалізації серверної інфраструктури інтегровано хмарну платформу Firebase (BaaS), що забезпечило надійну автентифікацію користувачів та безпечне збереження даних без необхідності розгортання власних серверів.

Спроєктовано безпечну навігаційну архітектуру мобільного клієнта на базі екосистеми React Navigation. Логічний розподіл інтерфейсу на незалежні стеки для неавторизованих (AuthStack) та авторизованих користувачів (MainTabs), підкріплений глобальним слухачем стану Firebase, унеможливив несанкціонований доступ до конфіденційних даних клієнтів та оптимізував використання оперативної пам’яті пристрою.

Розроблено оптимізовану інформаційну модель нереляційної (NoSQL) бази даних Cloud Firestore. Денормалізована структура колекцій (профілі, каталог, кошик, замовлення, відгуки) зі строгим клієнтським контролем зовнішніх ідентифікаторів дозволила мінімізувати кількість серверних запитів і забезпечила миттєве зчитування пов’язаних масивів даних.

Програмно реалізовано комплексну бізнес-логіку електронної комерції. Створено адаптивний інтерфейс із багатопараметричною системою фільтрації каталогу, розроблено модуль управління кошиком з асинхронною гідратацією даних, а також впроваджено багатокроковий процес оформлення замовлення

(чекаут) із суворою валідацією полів, імітацією платіжного шлюзу та автоматичною генерацією товарно-транспортних накладних.

Комплексне тестування (функціональне, UI/UX, інтеграційне та негативне) підтвердило надійність та стійкість застосунку до відмов. Реалізований механізм перехоплення виняткових ситуацій гарантує стабільну роботу системи та зрозумілий зворотний зв’язок навіть за умов нестабільного інтернет-з’єднання.

Практична значущість роботи полягає у готовності розробленого застосунку до інтеграції в реальні бізнес-процеси магазину «Sunjesus shop» для автоматизації продажів та підвищення лояльності клієнтів. Подальші перспективи розвитку проєкту включають підключення реальних платіжних систем, інтеграцію API логістичних операторів та впровадження системи push-сповіщень.

ПЕРЕЛІК ПОСИЛАНЬ

1. Cloud Firestore Data model. *Firebase Documentation*. URL: https://firebase.google.com/docs/firestore/data-model (дата звернення: 18.03.2026).

2. Core Components and APIs. *React Native Official Documentation*. URL: https://reactnative.dev/docs/components-and-apis (дата звернення: 09.03.2026).

3. EAS Build Introduction. *Expo Documentation*. URL: https://docs.expo.dev/build/introduction/ (дата звернення: 18.03.2026).

4. Expo Application Services Overview. *Expo Documentation*. URL: https://docs.expo.dev/ (дата звернення: 18.03.2026).

5. Firebase Authentication Overview. *Firebase Documentation*. URL: https://firebase.google.com/docs/auth (дата звернення: 18.03.2026).

6. FlatList Component API. *React Native Official Documentation*. URL: https://reactnative.dev/docs/flatlist (дата звернення: 10.04.2026).

7. Get started with Cloud Firestore Security Rules. *Firebase Documentation*. URL: https://firebase.google.com/docs/firestore/security/get-started (дата звернення: 18.03.2026).

8. Getting Started with React Navigation. *React Navigation*. URL: https://reactnavigation.org/docs/getting-started (дата звернення: 18.03.2026).

9. Introduction to React Native. *React Native Official Documentation*. URL: https://reactnative.dev/docs/getting-started (дата звернення: 09.03.2026).

10. Nested Navigation Guide. *React Navigation Documentation*. URL: https://reactnavigation.org/docs/nesting-navigators (дата звернення: 18.03.2026).

11. Performance Overview in React Native applications. *React Native Official Documentation*. URL: https://reactnative.dev/docs/performance (дата звернення: 09.03.2026).