Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

СИСТЕМА АВТОМАТИЗАЦІЇ ПЕРІОДИЧНИХ ТРАНЗАКЦІЙ ДЛЯ ОТРИМАННЯ ПОСЛУГ

Виконала магістрантка 2 курсу, групи 1КІ-23м
спеціальності 123 — Комп'ютерна інженерія
_____ Дзюба Д.А.
Керівник к.т.н., доц. каф. ОТ
_____ Томчук М.А.
"12" грудня 2024р.
Опонент к.т.н. доцент кафедри ПЗ
_____ Черноволик Г.О.
"12" грудня 2024р.

Допущено до захисту

Завідувач кафедри ОТ
_____ д.т.н., проф. Азаров О.Д.
«___» _____ 2024 р.

ВНТУ 2024

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

Галузь знань — Інформаційні технології

Освітній рівень — магістр

Спеціальність — 123 Комп'ютерна інженерія

Освітньо-професійна програма — Комп'ютерна інженерія

ЗАТВЕРДЖУЮ

Завідувач кафедри обчислювальної техніки

О.Д. Азаров

"18" вересня 2024 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

магістрантці **Дзюбі Дар'ї Анатоліївні**

1 Тема роботи «Система автоматизації періодичних транзакцій для отримання послуг» керівник роботи Томчук Микола Антонович к.т.н, доцент, затверджено наказом вищого навчального закладу від 17.09.2024 року № 310

2 Строк подання магістранткою роботи 25.12.2024 р.

3 Вихідні дані до роботи: призначення — автоматизація процесу оформлення повторних замовлень, засоби — середовище програмування VS code, мова програмування React Native, інструменти розробки Expo та Firebase.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналітичний огляд технології автоматизації періодичних транзакцій, дослідження концепцій створення системи автоматизації періодичних транзакцій, розробка системи автоматизації

періодичних транзакцій, верифікація отриманих результатів, економічна частина.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) — блок-схема модулю оформлення замовлення.

6 Консультанти розділів роботи приведені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Томчук Микола Антонович к.т.н., доц. каф. ОТ	10.09.24	12.12.24
4	Небава Микола Іванович проф., к.е.н		

7 Дата видачі завдання 18.09.2024р.

8 Календарний план виконання МКР приведений в таблиці 2.

Таблиця 2 — Календарний план

з/п	Назва етапів МКР	Строк виконання	Підпис
1	Постановка задачі	10.09.2024	
2	Огляд наявних рішень	25.09.2024	
3	Дослідження концепцій створення комерційних проєктів	11.10.2024	
4	Розробка системи автоматизації транзакцій	1.11.2024	
5	Розрахунок економічної частини	15.11.2024	
6	Попередній захист	18.11.2024	
7	Оформлення пояснювальної записки	6.12.2024	
8	Виконання магістерської кваліфікаційної роботи	10.12.2024	
9	Перевірка якості виконання магістерської кваліфікаційної роботи та усунення недоліків	11.12.2024	
10	Підписи супроводжувальних документів у керівника, опонента, нормо-контролера	12.12.2024	
11	Перевірка «Антиплагіат»	13.12.2024	

Магістрантка Дзюба Дар'я Анатоліївна

Керівник доц. Томчук Микола Антонович

АНОТАЦІЯ

УДК 004.4:374.3

Дзюба Д.А. Система автоматизації періодичних транзакцій для отримання послуг. Магістерська кваліфікаційна робота зі спеціальності 123 — Комп’ютерна Інженерія, Вінниця: ВНТУ, 2024 — 117 с.

На укр. мові. Бібліогр. назв.: 21; рис.: 32; табл. 7.

У даній роботі було розглянуто проблематику реалізації періодичних транзакцій для автоматизації процесу оформлення замовлень. Проведено аналіз сучасних методів автоматизації, досліджено концепції оптимізації замовлень та обрано найефективніші підходи. Розроблено технічну архітектуру мобільного додатку, який спрощує повторні замовлення через збереження даних користувачів і використання QR-кодів. Проведено перевірку працездатності системи та оцінено її ефективність з точки зору зручності для користувачів і доцільності для бізнесу.

Ключові слова: автоматизація, періодичні транзакції, QR-коди, мобільний додаток, комерційна платформа.

ABSTRACT

УДК 004.4:374.3

Dziuba D.A. System for automating periodic transactions for obtaining services. Master's Qualification Work in the field of 123 — Computer Engineering, Vinnytsia: VNTU, 2024 — 117 pages. In Ukrainian. Bibliography titles: 21; figures: 32; tables: 7.

This paper examines the issue of implementing periodic transactions to automate the order processing process. An analysis of modern automation methods was conducted, concepts for optimizing orders were investigated, and the most effective approaches were selected. The technical architecture of a mobile application was developed, which simplifies repeat orders by saving user data and using QR codes. The system's performance was tested and its effectiveness was assessed in terms of user convenience and business feasibility.

Keywords: automation, recurring transactions, QR codes, mobile application, commercial platform.

ВСТУП

Швидкість і комфорт дедалі частіше впливають на вибір послуг і продуктів користувачем, тому моделі автоматизованих періодичних транзакцій стрімко набирають популярності серед споживачів. Через це підписка є одним із найбільш затребуваних форматів реалізації даної системи. Такий підхід забезпечує регулярний доступ до необхідного без зайвих зусиль, звільняючи клієнтів від повторного пошуку чи замовлення. Завдяки передбачуваній системі транзакцій люди можуть ефективніше розпоряджатися своїми коштами, а компанії отримуватимуть стабільний дохід.

Популярність моделей підписок демонструє зміну ставлення до споживання — суспільство прагне спрощення рутинних процесів. Від цифрових платформ для трансляцій до доставки продуктів — ці сервіси стали універсальним рішенням для багатьох аспектів життя. Завдяки своїй універсальності цей формат продовжує розвиватися, стаючи ще більш адаптивним і зручним для сучасного користувача.

Попри вищеописані переваги, бізнес-модель підписки має певні недоліки, які можуть впливати на рівень задоволеності клієнтів. Однією із найбільш поширених проблем є нестача гнучкості, що призводить до ситуацій, коли користувачі отримують товари або послуги за графіком, який не враховує їхні реальні потреби. Це не лише ускладнює процес користування, але й може сприяти зниженню лояльності клієнтів до компанії.

Актуальність роботи зумовлена зростаючою потребою у автоматизації процесів періодичних замовлень, тому що сучасні системи систематичних транзакцій зазвичай працюють за моделлю підписки, яка не враховує реальну потребу користувача в товарах. Дослідження та розробка системи, що дозволяє мінімізувати час та зусилля під час повторних замовлень, створює значні переваги для малого та середнього бізнесу, які складають основу економіки України.

Метою даної роботи є удосконалення періодичних транзакцій за допомогою функції швидкого повторного оформлення замовлень для отримання послуг за допомогою сканування QR-кодів.

Для досягнення мети необхідно розв’язати такі задачі:

— проаналізувати сучасні методи, які реалізують функціонал періодичних транзакцій задля отримання послуг;

— дослідити різні концепції автоматизації замовлень та вибрати найоптимальніші для автоматизації процесу;

— розробити технічну архітектуру мобільного додатку, автоматизувавши систему періодичних транзакцій;

— провести перевірку працездатності системи та оцінити ефективність запропонованого рішення з точки зору користувачів і бізнесу.

Об’єктом є процес автоматизації регулярних транзакцій в електронній комерції.

Предметом є пришвидшення повторного замовлення товарів за допомогою швидкого сканування QR-коду, що зберігає в собі інформацію про попереднє замовлення.

Практична цінність полягає у створенні зручного інструменту для бізнесу та споживачів, який надає можливість оформлення покупки та спрощує процес повторного замовлення товарів до відсканування QR-коду.

Новизна роботи полягає в удосконаленні процесу періодичних транзакцій за допомогою автоматизації повторних замовлень через сканування QR-кодів. Запропонована технологія надає можливість користувачам пришвидшити процес регулярних покупок, що також дозволяє підвищити клієнтоорієнтованість зі сторони бізнесу.

Апробацію результатів наукової роботи було проведено на науковій конференції : «LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025)», доповідь на тему «Методи організації безпеки даних під час автоматичних платежів» [1]. Також подано заявку на реєстрацію авторського свідоцтва комп’ютерної програми на тему «Сервіс замовлень та їх повторень із застосуванням QR-кодів» від 20.11.2024.

1 АНАЛІТИЧНИЙ ОГЛЯД ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ ПЕРІОДИЧНИХ ТРАНЗАКЦІЙ

У сучасному світі багато людей стикається з необхідністю періодично поповнювати запаси певних товарів, таких як побутова хімія, продукти харчування, косметика чи парфумерія. Ці потреби є регулярними, але сам процес їх замовлення часто забирає багато часу і зусиль. Користувачам доводиться згадувати, де вони зазвичай купують ці продукти, шукати їх у доступних магазинах, оформлювати замовлення, вводити дані для доставки й чекати на підтвердження. Усе це створює відчуття зайвого клопоту в житті, яке й без того наповнене іншими турботами.

З огляду на це, автоматизація процесу поповнення запасів могла б значно полегшити життя багатьом людям. Ідеальним рішенням було б звести цей процес до мінімуму дій з боку користувача — наприклад, до одного натискання кнопки чи навіть повністю автоматизованого сценарію. Зручність і простота є ключовими факторами, які споживачі шукають у таких рішеннях. Одним із способів реалізації цієї ідеї є модель підписки, яка передбачає автоматичне постачання товарів через визначені інтервали часу.

1.1 Огляд ключових аспектів бізнес-моделі за підпискою

Дослідження показують, що, за прогнозами, до 2025 року обсяг електронної комерції за підпискою сягне понад 450 мільярдів доларів [2]. Однак деякі бренди електронної комерції ще не перейшли на модель підписки, а інші навіть не знають, як вона працює, щоб ефективно стимулювати збільшення продажів і зростання їхнього бізнесу.

Бізнес-модель підписки на електронну комерцію — це модель, за якої клієнти можуть підписатися на регулярне отримання послуг від бренду. Клієнт може вибрати графік платежів, який його влаштовує і який відповідає його бюджету та плану. Такий підхід дозволяє бізнесу отримувати стабільний і передбачуваний дохід, а користувачам — насолоджуватися зручністю автоматичних платежів і постійного доступу до потрібного товару.

Вперше модель підписки була впроваджена ще у 17 столітті, коли видавці газет і журналів почали пропонувати читачам регулярні випуски за передплатою. З часом цей підхід поширився на інші сфери, зокрема на медіа, розваги, програмне забезпечення, харчування, а згодом і на цифрові платформи.

Модель підписки в електронній комерції забезпечує стабільний потік доходів, оскільки покупці регулярно замовляють товари. Як свідчить звіт Statista, цей формат продажів згенерував понад 38 мільярдів доларів у США за останні роки (рисунок 1.1). Крім того, підписка спрощує управління запасами. Завдяки передбачуваним вподобанням клієнтів та знанню графіків замовлень, компанії можуть ефективніше планувати закупівлі й оптимізувати логістику.

Збоку здається, що передплати є універсальним рішенням для будь-якого бізнесу. Але така модель монетизації підходить не всім компаніям. Наприклад, підписки на онлайн-кінотеатри є релевантними, тому що люди дивляться фільми та серіали щодня. Але впроваджувати передплату за послуги по пошукам довгострокової оренди нерухомості недоцільно, оскільки користувачі зазвичай знаходять житло і не повертаються із таким запитом протягом тривалого часу. Така модель ускладнює досягнення економічної рентабельності та ефективного використання ресурсів.

Підписка також допомагає скоротити витрати на залучення нових клієнтів. Регулярні покупки забезпечують стабільність продажів, що зменшує необхідність постійного пошуку нових споживачів. Це дозволяє бізнесу зосередитися на утриманні існуючих клієнтів, які продовжують робити замовлення. Завдяки цьому формується база лояльних споживачів. Вони не лише залишаються відданими бренду, а й часто рекомендують його своїм знайомим, розширюючи клієнтську аудиторію природним шляхом.

Реалізація концепції підписки також позитивно впливає на пожиттєву цінність клієнта (LTV). Коли споживачі регулярно роблять покупки, їхні відносини з брендом зміцнюються. Чим більше користувачі використовують продукцію та отримують задоволення від її якості, тим вищий дохід бізнесу з кожною наступною транзакцією. Зростання LTV є показником успішності

підпискової моделі, яка сприяє як фінансовій стабільності компанії, так і підвищенню рівня задоволеності клієнтів.

Рисунок 1.1 — Порівняльна статистика прибутку від продажу підписки в США

Основними підходами, які використовуються в галузі підписки на електронну комерцію є:

— підписка на доступ;

— кураторська модель;

— система поповнення запасів;

— преміум-членство.

Кожна з цих моделей має свої особливості, переваги та сфери застосування, що дозволяє брендам адаптуватися до потреб різних категорій споживачів.

Модель підписки на доступ передбачає щомісячну або щорічну оплату клієнтами за ексклюзивні пропозиції, які надає бренд. Завдяки такій системі споживачі можуть користуватися знижками на товари або отримувати доступ до нових продуктів раніше за інших. Вільний вибір дає можливість замовляти товари, які відповідають їхнім потребам, у зручний для них час із доставкою на визначену дату. Така підписка сприяє утриманню клієнтів, підвищенню їхньої

відданості бренду та заохочує рекомендувати послуги іншим. Приклад такої моделі підписки зображено на рисунку 1.2.

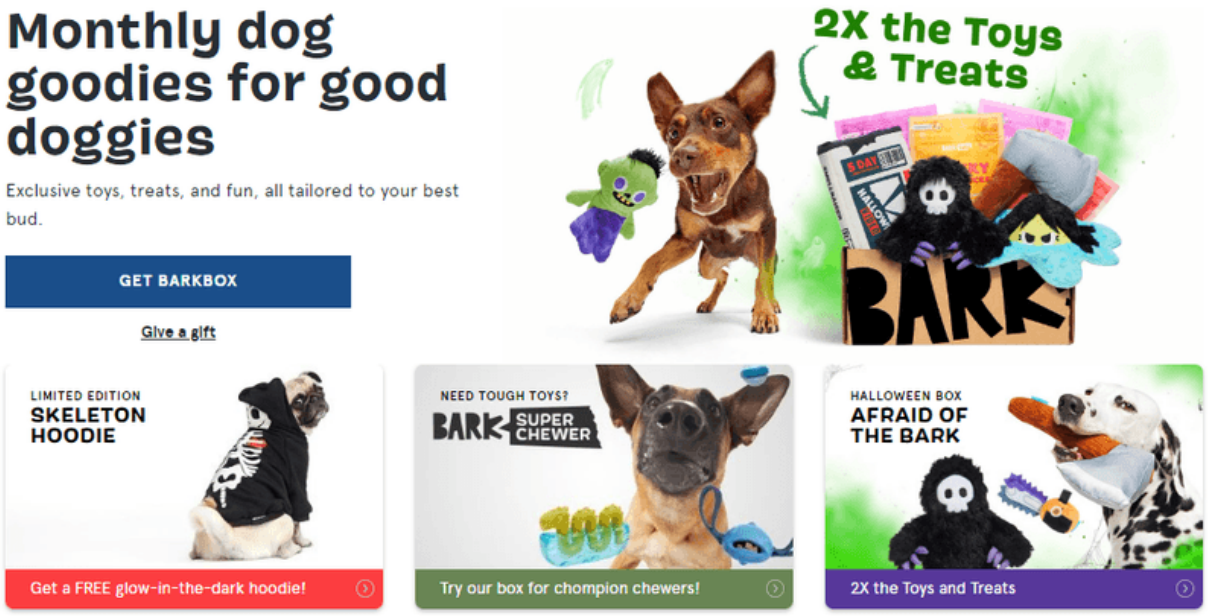

Рисунок 1.2 — Приклад моделі підписки на доступ від бренду BarkBox

Іншою популярною системою підписки є кураторська модель, яка пропонує споживачам персоналізовані товари відповідно до їхніх вподобань. Такий підхід особливо затребуваний у сферах косметики, їжі чи квітів. Клієнти отримують продукти, які точно відповідають їхнім смакам, що дозволяє брендам дивувати споживачів новими пропозиціями.

Магазини надсилають спеціально підібрані набори в заздалегідь визначені терміни, забезпечуючи своєчасну доставку. Для цього важливо дізнаватися про вподобання клієнтів, щоб максимально точно відповідати їхнім очікуванням. Така модель ідеально підходить для продажу шоколаду, алкоголю, книг, вина, сиру чи іграшок. Приклад такої моделі підписки зображено на рисунку 1.3.

Модель підписки на поповнення орієнтована на товари, які клієнти використовують регулярно, наприклад, продукти харчування або біологічно активні добавки. У такому випадку магазин пропонує доставку товарів без необхідності оформлення повторних замовлень.

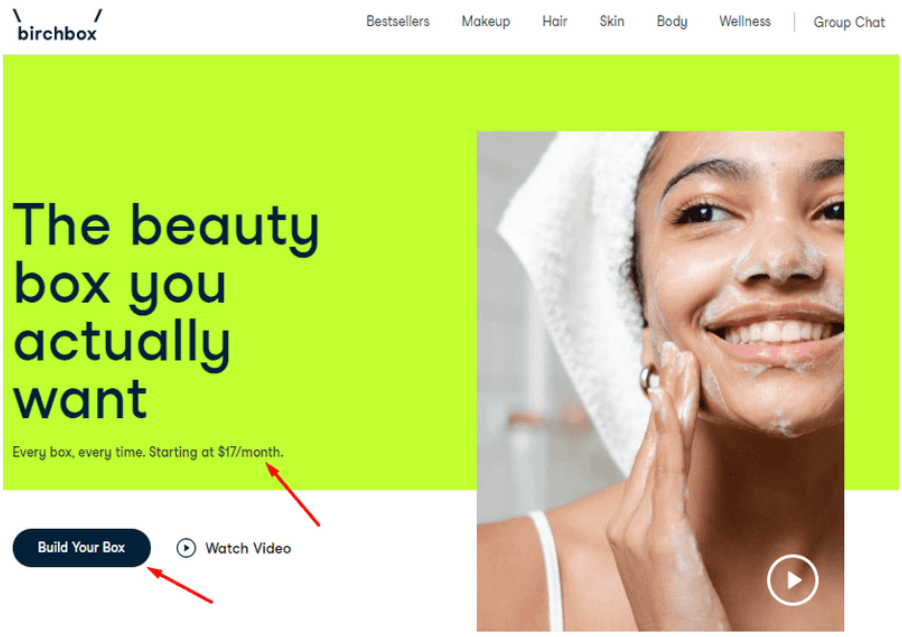

Рисунок 1.3 — Приклад підписки з кураторською моделлю від бренду BirchBox

Покупці надають графік, за яким їм потрібні поставки, та отримують потрібні товари у встановлені терміни. Це ідеальне рішення для тих, хто регулярно використовує певні продукти, та хочуть пришвидшити та автоматизувати процес оформлення періодичних замовлень. Приклад такої моделі підписки зображено на рисунку 1.4.

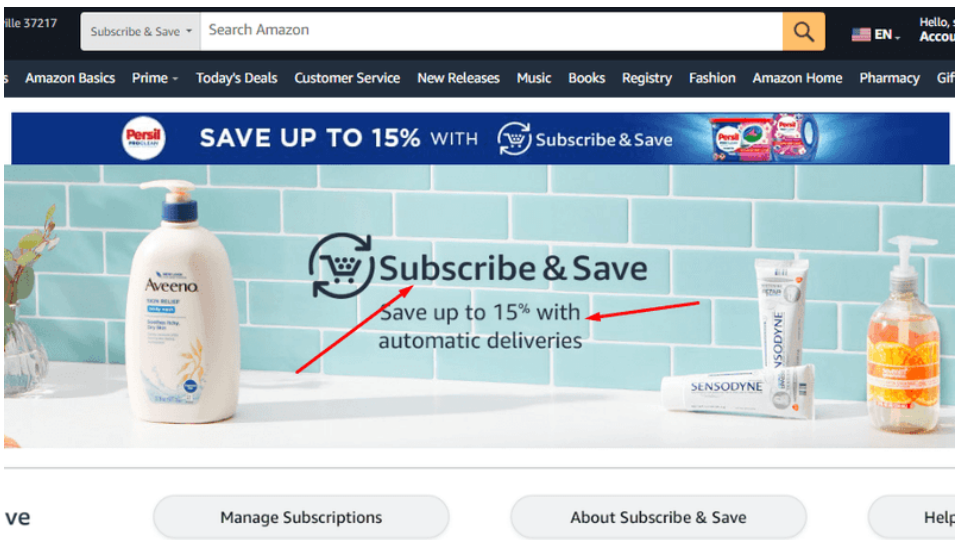

Рисунок 1.4 — Приклад підписки на поповнення товарів від компанії Amazon

Ще одним варіантом є модель підписки на преміум-членство, яка передбачає оплату за доступ до продуктів з лімітованої лінійки. Наприклад, сервіс Amazon Prime пропонує клієнтам доступ до ексклюзивних товарів для покупок і розваг. Під час оформлення підписки клієнти можуть протестувати послугу в рамках пробного періоду, а після його завершення обрати найбільш прийнятний пакет послуг. Такий підхід дозволяє брендам залучати клієнтів до більш активного використання преміум-пропозицій. Приклад такої моделі підписки зображено на рисунку 1.5.

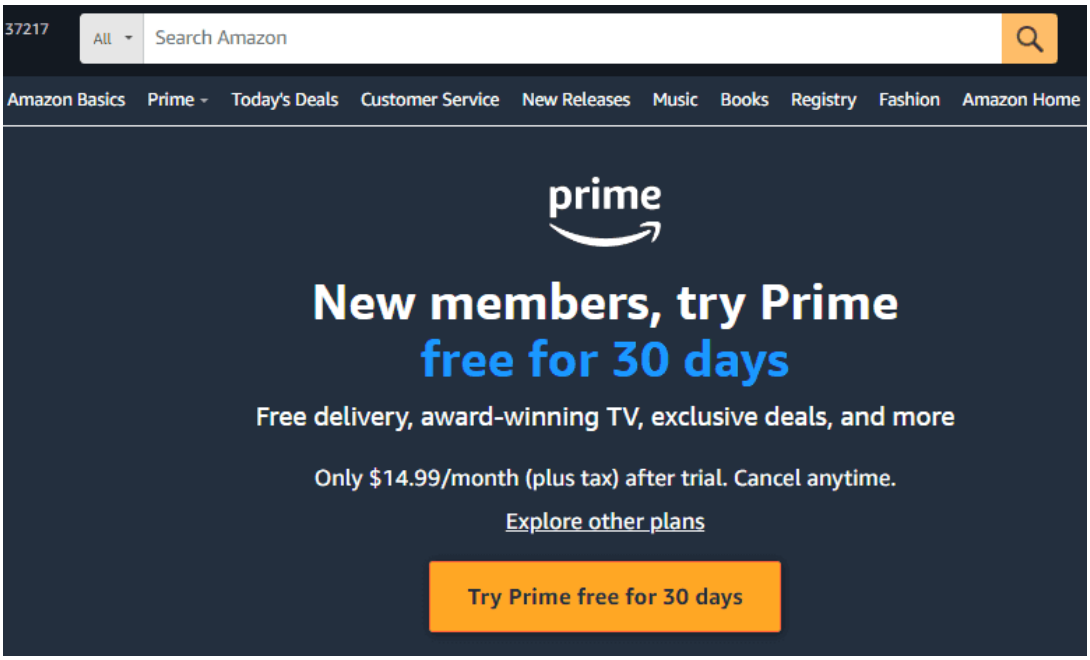

Рисунок 1.5 — Приклад моделі підписки на преміум членство від Amazon

1.2 Алгоритм роботи підписки

Регулярний платіж — це тип платіжної схеми, завдяки якій кошти автоматично знімаються з кредитної картки або банківського рахунку клієнта через регулярні проміжки часу. При такій схемі оплати використовується система «картка у файлі», коли інформація про клієнта зберігається за його згодою, а кошти знімаються в разі настання терміну платежу.

Хоча регулярні платежі не є чимось новим, за останні кілька років вони стали популярними в різних галузях. Це, ймовірно, пов’язане з їхньою зручністю, простотою використання та перевагами як для бізнесу, так і для клієнтів.

Існує два типи регулярних платежів: фіксовані та змінні. У фіксованих платежах з клієнтів стягується однакова сума за кожен платіжний цикл. Підходять для компаній, що працюють за підпискою, наприклад, спортзалів за абонементами або платформ, що транслюють контент. Змінні ж залежать від того, скільки продукту/послуги отримав клієнт. Застосовуються в комунальних компаніях або підписках SaaS, які використовують модель ціноутворення за принципом «сплачуй по мірі використання».

Схема роботи автоматичних регулярних платежів (рисунок 1.6) у моделі підписки забезпечує зручність для клієнтів та стабільність доходів для бізнесу. У процесі підписки клієнт спочатку обирає бажаний тарифний план і налаштовує автоматичний платіж через один із доступних платіжних сервісів, наприклад, PayPal або Braintree. Це дозволяє здійснювати регулярні списання коштів у визначений час, без необхідності вручну підтверджувати кожну оплату [3].

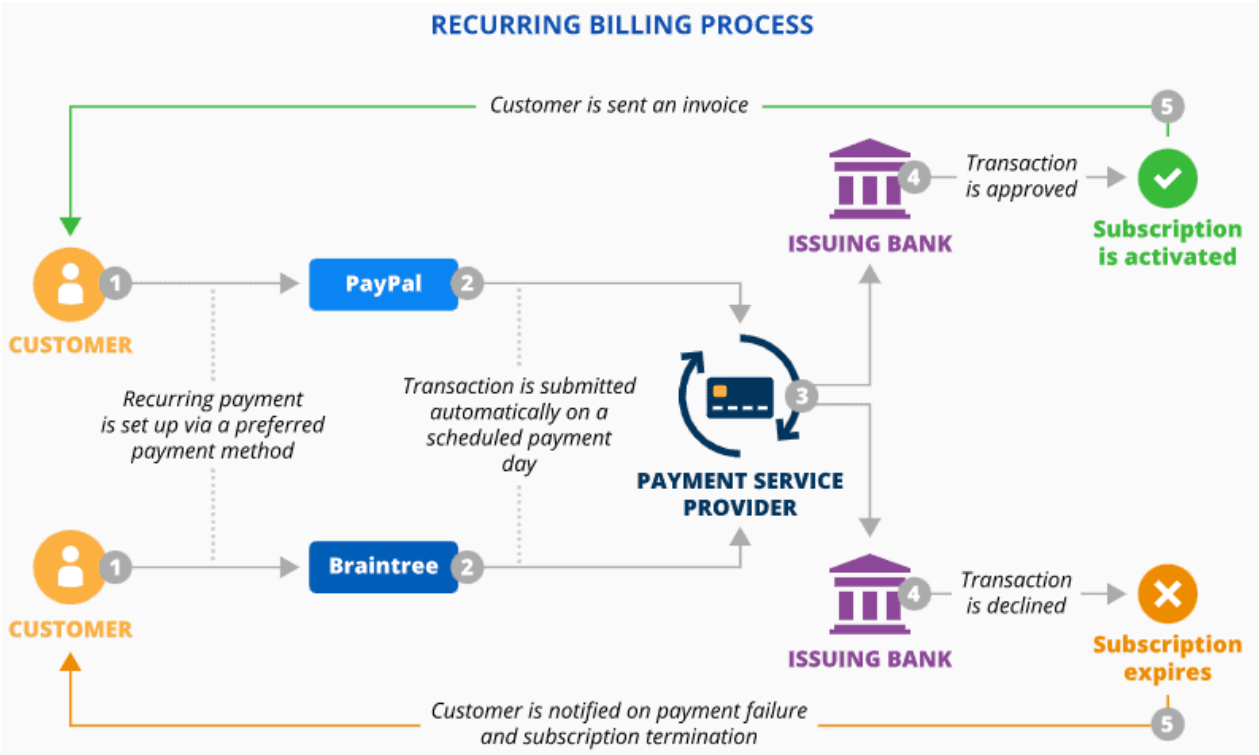

Рисунок 1.6 — Схема роботи автоматичних регулярних платежів

Після вибору способу оплати та оформлення підписки платіжна система автоматично ініціює транзакцію в зазначений день. Цей процес є ключовим елементом автоматизації, який позбавляє клієнтів необхідності повторно вводити дані картки або інші деталі для кожного списання. У той же час, це дозволяє компанії забезпечити регулярний потік доходів, зменшуючи кількість пропущених платежів.

Платіжний провайдер, наприклад, PayPal або Braintree, обробляє запит на транзакцію та передає його до банку-емітента, який видав картку клієнта. Якщо грошей достатньо і рахунок покупця не заблокований, емітент підтверджує операцію для еквайра. Останній зараховує списані на оплату товару гроші на розрахунковий рахунок магазину.

Щоб процедура онлайн-покупки товару для клієнта в електронній комерції була максимально комфортною, продавці можуть використовувати платіжні шлюзи. Це канал, який надається платіжним провайдером, по якому в зашифрованому вигляді передаються номер та інші дані платіжної картки покупця. Такий інструмент допомагає зробити процес покупки швидким і зручним для користувача, а продавцю — оперативно отримати гроші на свій рахунок.

Якщо відмовитися від інтеграції з платіжним шлюзом, то при оплаті замовлення покупцю доведеться перейти з сайту мерчанта на сайт банку, де відбуватиметься оплата. Це збільшує час операції, знижує конверсію та загалом негативно впливає на бізнес-результати.

У випадку, якщо банк відхиляє транзакцію через відсутність коштів, недійсність картки або інші проблеми, платіж не може бути оброблений. Це призводить до того, що підписка автоматично завершується, і клієнт втрачає доступ до послуг. У такій ситуації клієнт отримує повідомлення про невдалу спробу списання коштів із рекомендаціями щодо вирішення проблеми.

Автоматичні платежі значно спрощують управління фінансами для споживачів, оскільки усувають необхідність постійно стежити за термінами оплати. У той же час бізнес може оптимізувати свої процеси, зменшуючи витрати

на ручне управління підписками та забезпечуючи стабільний потік доходів. Впровадження такого механізму також сприяє підвищенню задоволеності клієнтів завдяки безперервності доступу до послуг.

Центральною частиною цієї моделі є надійна інтеграція платіжних систем, які забезпечують безпеку транзакцій та відповідність міжнародним стандартам, таким як PCI DSS. Це дозволяє клієнтам бути впевненими у збереженні їхніх фінансових даних, а компаніям — уникати ризиків, пов’язаних із витоком інформації. Автоматизована система також дозволяє бізнесу в реальному часі відстежувати статус платежів і оперативно реагувати на будь-які проблеми.

Ключовим елементом роботи є зворотний зв’язок із клієнтами. Успішна транзакція супроводжується автоматичним підтвердженням, яке може бути надіслане на електронну пошту або відображене у додатку. У випадку неуспішного платежу клієнт оперативно інформується про проблему з рекомендаціями, як її вирішити, наприклад, оновити платіжні дані або поповнити рахунок.

1.3 Аналіз переваг та недоліків бізнес-моделі за підпискою

Однією з ключових відмінностей між моделями підписки та традиційними бізнес-моделями є потік доходів. У той час як традиційний бізнес покладається на разові покупки, бізнес, що базується на підписці, отримує передбачуваний дохід завдяки постійному характеру надходжень. Крім того, моделі підписки сприяють розвитку довгострокових відносин з клієнтами, оскільки споживачі постійно взаємодіють з брендом, що призводить до підвищення їхньої лояльності.

Перевагами є те, що модель підписки створює стабільний потік доходів, що дозволяє компаніям ефективніше планувати майбутнє. Завдяки регулярним платежам бізнес отримує передбачуваність фінансових надходжень, яка полегшує довгострокове прогнозування. Також передплатники зазвичай демонструють більшу відданість бренду порівняно з одноразовими покупцями, оскільки їхній вибір передбачає тривале користування продуктом або послугою. Даний підхід сприяє формуванню глибших відносин між клієнтами та компанією. Через це

абоненти частіше згодні переходити на дорожчі плани або купувати додаткові функції, що створює потенціал для збільшення прибутків [4].

Залучення нових передплатників зазвичай коштує дешевше, ніж робота з постійним залученням нових покупців для одноразових продажів. Така модель дозволяє бізнесу фокусуватися на утриманні наявної клієнтської бази, знижуючи витрати на маркетинг. Крім того, передплата забезпечує гнучкість у ціноутворенні, що дає можливість пропонувати кілька рівнів або пакетів послуг. Це дозволяє компаніям задовольняти потреби різноманітних груп споживачів, враховуючи їхні бюджети та вподобання [5].

Окрім позитивних рис, модель підписки має певні недоліки, які можуть впливати на її ефективність. Хоч дана модель пропонує стабільний дохід, але ризик відтоку клієнтів все ще присутній, оскільки споживачі мають змогу скасувати підписку в будь-який момент. Тому залежність від утримання клієнтів може суттєво позначитися на доходах і фінансовій стабільності компанії. З часом споживачі також можуть втратити інтерес до послуги, відчуваючи певну втому від передплачених сервісів, що призводить до зниження залученості та зростання рівня відтоку.

Залучення та утримання передплатників вимагає значних зусиль у сфері маркетингу, що іноді може виявитися витратним моментом, особливо для компаній, які активно використовують платну рекламу для просування своїх послуг. Оскільки користувачі сплачують щомісячну постійну комісію, вони очікують більше, ніж за одноразовий платіж за програму. Вимагаючи від користувачів платити фіксовану плату за доступ до програми, бізнес зобов’язаний постійно надавати новий вміст і/або функції для користувача. Через це переконати споживачів оформити підписку стає непросто, особливо якщо вони вважають продукт чи послугу надмірно дорогими або непотрібними.

Вищеописаний момент створює додаткові труднощі для бізнесу, змушуючи шукати креативні підходи для залучення клієнтів. Окрім того, компанії, що працюють за підписковою моделлю, постійно стикаються з необхідністю створювати додаткову цінність для споживачів, аби виправдати регулярні витрати

на їхню послугу. Тому такий підхід вимагає від команд із розробки продуктів та обслуговування клієнтів значних зусиль для підтримання високого рівня якості та задоволення клієнтів.

Конкуренція в сегменті підпискових послуг також продовжує посилюватися. Розширення цього ринку у багатьох галузях змушує компанії шукати унікальні способи виділятися серед конкурентів і утримувати свою клієнтську базу. Усі ці ризики вимагають ретельного планування, адаптації та впровадження інновацій для підтримання успішності бізнесу.

Кожен із типів підписок пропонує різні підходи до задоволення потреб користувачів, однак підписка на поповнення товарів залишається найменш популярною. Основна причина цього полягає в тому, що модель передбачає автоматичне стягнення коштів і відправку продукції через певний проміжок часу, незалежно від фактичного використання запасів споживачем. У багатьох випадках це призводить до накопичення товарів, які клієнт ще не встиг використати, що створює незручності та розчарування.

Така ситуація часто викликає відчуття нав'язливості та непотрібності з боку користувача, особливо якщо товар не є критично необхідним або термін його споживання не збігається з графіком доставки. Наприклад, у випадку з косметикою, побутовою хімією або продуктами харчування, надмірна кількість замовленого може призвести до марнотратства або втрати придатності товарів. Це не лише негативно впливає на досвід взаємодії клієнта, але й підриває його довіру до бренду.

Крім того, автоматизація процесу, яка є основою таких підписок, інколи створює враження, що компанія більше дбає про регулярність отримання прибутку, ніж про реальні потреби своїх клієнтів. Це може погіршити відносини між бізнесом і споживачем, що є критично важливим у конкурентному середовищі.

1.4 Порівняльний огляд аналогів

Для вирішення повсякденних потреб українські компанії створили низку мобільних додатків, які орієнтовані на замовлення товарів із різних сфер життя. Наприклад, додаток "Сільпо" спеціалізується на замовленні продуктів харчування, "Розетка" пропонує широкий вибір техніки та товарів для дому, "Пром" дозволяє купувати майже будь-які товари, "OLX" надає можливість придбати речі за вигідними цінами, а "Шафа" орієнтована на продаж і купівлю одягу. Кожен із цих продуктів має унікальні функції та переваги, але в контексті автоматизації періодичних покупок виникають спільні проблеми, які потребують аналізу. Детальний аналіз основного функціоналу, переваг та недоліків зображений в таблиці 1.1.

Таблиця 1.1 — Порівняльний аналіз програм-аналогів

Назва додатку	Основний функціонал	Переваги	Недоліки
Сільпо	Замовлення продуктів харчування, послуги доставки, накопичення бонусів.	Зручність замовлення продуктів, можливість оформлення повторного замовлення	Для того, щоб повторно оформити замовлення потрібно виконати низку неоптимізованих дій
Розетка	Електронна комерція: продаж техніки, електроніки, одягу, товарів для дому	Великий асортимент, зручність пошуку, можливість повторного оформлення замовлень	Для того, щоб скористатись функцією повторного замовлення потрібно виконати низку неоптимізованих дій
Пром	Маркетплейс для торгівлі широким асортиментом товарів між продавцями і покупцями.	Можливість вибору серед багатьох продавців, конкурентні ціни, можливість повторного оформлення замовлення	Користувачеві для того, щоб скористатись функцією повторного замовлення потрібно виконати низку неоптимізованих дій
OLX	Онлайн-платформа для купівлі, продажу та обміну товарів і послуг між користувачами	Простий інтерфейс, великий вибір товарів і послуг, безкоштовне розміщення оголошень	Відсутність системи автоматичного повторного оформлення замовлень

Продовження таблиці 1.1

Shafa	Онлайн-платформа для купівлі та продажу одягу, взуття та аксесуарів між приватними особами	Можливість безпосереднього спілкування між покупцем і продавцем для уточнення деталей угоди; безпечна оплата через платформу з гарантіями для покупця	Відсутність системи автоматичного повторного оформлення

"Сільпо" — це додаток, який дозволяє користувачам замовляти продукти харчування з доставкою додому або для самовивозу. Застосунок включає в себе програму лояльності, яка заохочує клієнтів до повторних покупок, а також зручний інтерфейс для пошуку товарів. Проте недоліком є те, що для повторного замовлення потрібно відкрити додаток, увійти в обліковий запис, знайти попередні замовлення і після натиснення на кнопку «Повторити замовлення» всі товари переміщаються в кошик (рисунок 1.7). Даний процес автоматизовує лише частину із додаванням продуктів в кошик, але задля оформлення замовлення потрібно ще додатково вибрати адресу доставки та пройти весь процес створення покупки. На жаль, такий підхід спростовує лише маленьку частину, що може відвернути користувача від можливої покупки.

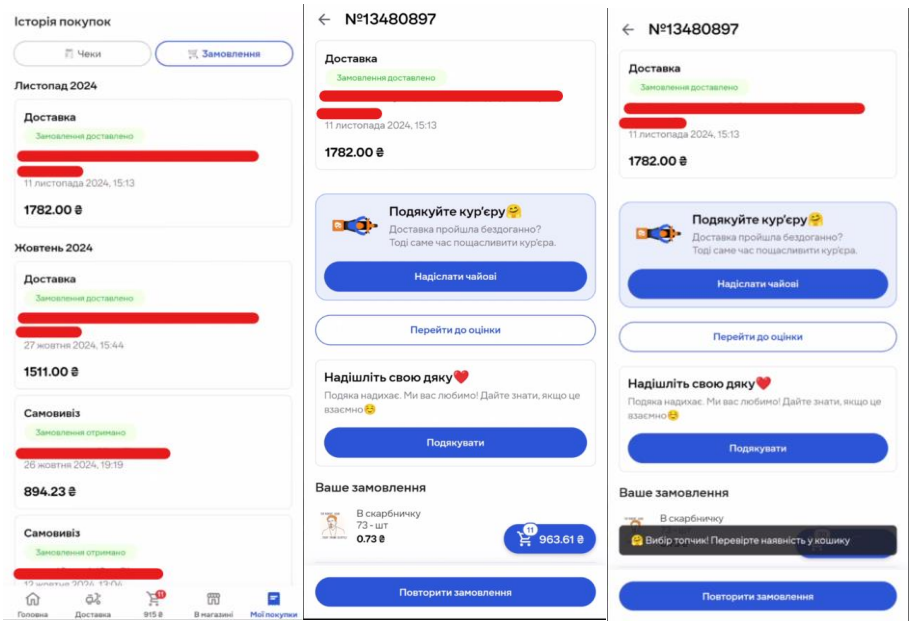

Рисунок 1.7 — Процес повторно оформлення замовлення в додатку «Сільпо»

"Розетка" забезпечує широкий асортимент товарів: від електроніки до побутової техніки та одягу. Вона пропонує інтуїтивний пошук, швидкі фільтри й багато способів оплати. Проте, як і в "Сільпо", повторне оформлення замовлення є складним процесом. Користувачам доводиться вручну відстежувати попередні покупки, а адреса замовлення автоматично не оновлюється до тієї, що була в замовленні (рисунок 1.8). Також одним із недоліків даного функціоналу є те, що можливість повторної купівлі продукту доступна лише для специфічних замовлень. Із зручностей можна виділите те, що при оформленні такого замовлення, в кошик можна також додати продукти, які раніше були обрані, але сам процес покупки не був завершений.

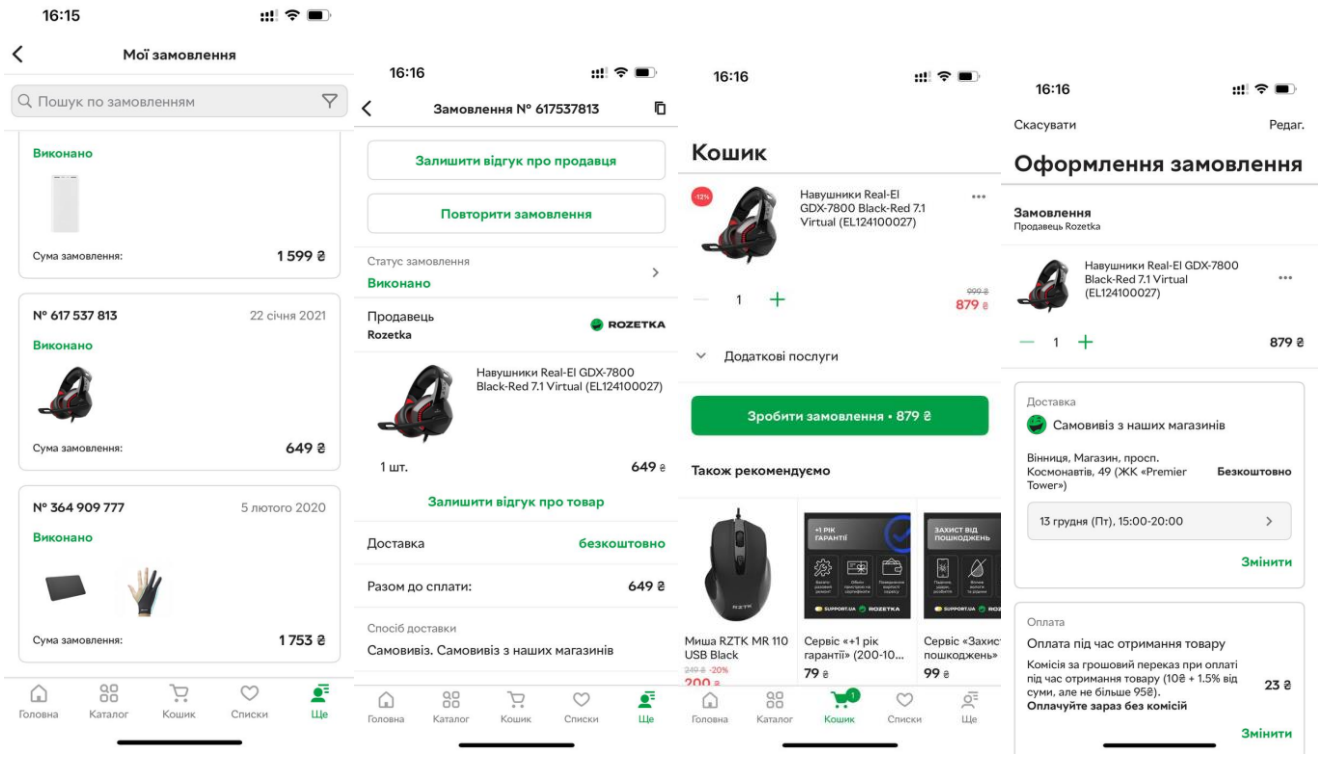

Рисунок 1.8 — Процес повторно оформлення замовлення в додатку «Rozetka»

"Пром" як маркетплейс дозволяє покупцям обирати серед численних продавців. Його сильна сторона — це доступ до багатьох пропозицій і можливість знайти товари за вигідними цінами. Водночас, повторне замовлення ускладнене через необхідність заново шукати покупку в історії, підтверджувати свій вибір на сторінці кошику (рисунок 1.9). Даний процес можна значно оптимізувати,

впровадивши функціонал, що дозволяє користувачам зберігати вибрані товари для швидкого повторного оформлення замовлення. Такий підхід не тільки спрощує процедуру замовлення, але й мінімізує кількість кроків, які потрібно виконати. Наприклад, користувачеві більше не доведеться щоразу заново вибирати товари чи вводити дані доставки. Система автоматично збереже адресу доставки, використану в попередньому замовленні, що особливо зручно для тих, хто здійснює замовлення на один і той самий адрес. Крім того, функція автоматичного вибору платіжної картки, яка використовувалася для попередньої транзакції, дозволить уникнути необхідності повторного введення платіжних даних.

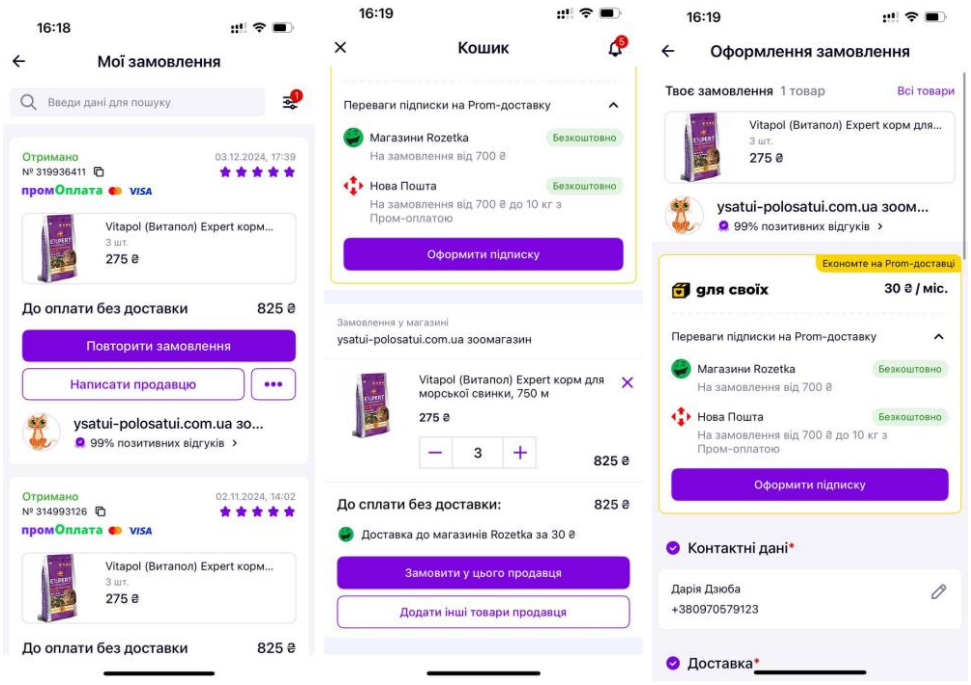

Рисунок 1.9 — Процес повторно оформлення замовлення в додатку «Prom»

OLX, у свою чергу, пропонує унікальну модель, орієнтовану на купівлю, продаж і обмін товарів між приватними особами. Простий інтерфейс і можливість безкоштовного розміщення оголошень приваблюють як покупців, так і продавців. Відсутність функції повторного оформлення замовлення на платформі OLX можна пояснити специфікою її діяльності та асортиментом товарів. Оскільки більшість товарів, представлених на платформі, є унікальними або доступними в

обмеженій кількості, то їхня повторна покупка, як правило, неможлива. Це стосується, наприклад, товарів ручної роботи, колекційних предметів, ексклюзивних пропозицій або продуктів із сезонними обмеженнями. У таких випадках реалізація механізму повторного замовлення може втратити свою доцільність, адже товар, придбаний раніше, просто більше не буде доступний у продажу.

Крім того, подібна платформа, ймовірно, орієнтована на те, щоб користувачі постійно відкривали для себе нові товари, а не купували одні й ті самі продукти. Такий підхід сприяє підтримці інтересу до платформи, адже клієнти повертаються не для повторення попереднього досвіду, а для пошуку чогось нового. Відсутність функціоналу для повторного оформлення замовлень може бути свідомим рішенням, яке спрямоване на заохочення користувачів до активного перегляду каталогу та відкриття унікальних пропозицій. Приклад вигляду замовлення на платформі OLX зображено на рисунку 1.10.

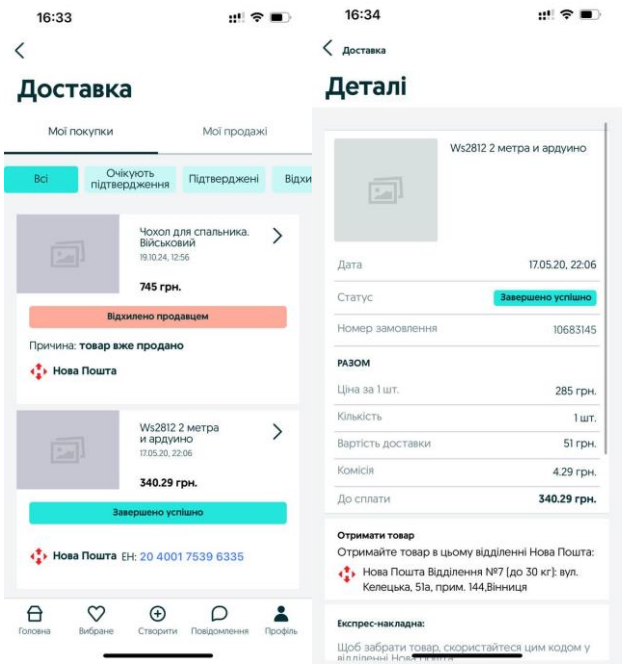

Рисунок 1.10 — Сторінки перегляду замовлень в додатку «OLX»

Shafa є платформою, орієнтованою на торгівлю одягом, взуттям та аксесуарами, що чітко виділяє її серед інших онлайн-майданчиків. Завдяки

спеціалізації на модній індустрії, Shafa створює зручний простір для поціновувачів стильного та доступного гардеробу. Покупці мають можливість знайти унікальні речі за вигідними цінами, що часто не зустрічаються в масовому рітейлі, а продавці, у свою чергу, можуть легко реалізовувати свій асортимент, отримуючи прибуток у стислі терміни. Це робить платформу привабливою як для тих, хто прагне оновити гардероб, так і для тих, хто бажає позбутися зайвого одягу чи аксесуарів.

Що стосується повторних покупок, Shafa має схожу ситуацію з платформою OLX. Товари, які викладають користувачі, зазвичай доступні в єдиному екземплярі. Це означає, що після покупки конкретного товару той самий продукт не буде доступний повторно. Такий підхід відповідає характеру платформи, яка орієнтована на унікальні пропозиції, а не на масовий продаж і поповнення одного й того ж асортименту. Як результат, Shafa стає місцем для пошуку оригінальних речей, але не для регулярного замовлення однакових товарів, що логічно випливає з її бізнес-моделі та формату. Приклад вигляду замовлення на платформі Shafa зображено на рисунку 1.11.

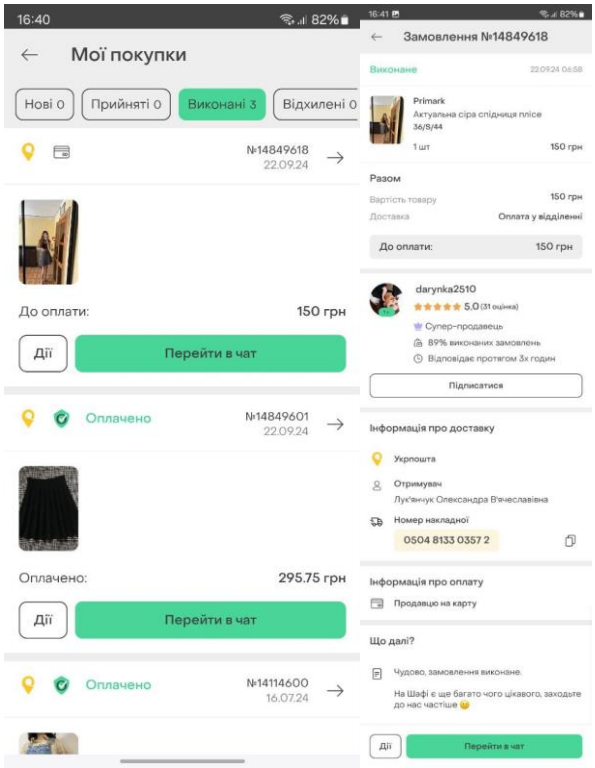

Рисунок 1.11 — Сторінки перегляду замовлень в додатку «Shafa»

Всі вищеописані платформи демонструють різні підходи до вирішення задач електронної комерції. Якщо "Сільпо" і Shafa орієнтовані на вузьку аудиторію з конкретними потребами, то "Розетка" і "Пром" пропонують універсальні рішення для масового ринку. OLX займає свою унікальну нішу, дозволяючи користувачам самостійно організовувати процеси купівлі та продажу.

Така система могла б бути побудована на основі інструментів, які дозволяють легко ініціювати повторне замовлення, або через механізми, які адаптуються до поведінки споживача, враховуючи його індивідуальні потреби. Це може включати автоматизовані алгоритми, що аналізують дані про попередні замовлення, або можливість зручно оформити повторний запит одним натисканням. У будь-якому випадку, мета полягає в тому, щоб створити персоналізований досвід, який би усував недоліки моделі підписки, забезпечуючи максимальну гнучкість і зручність для кінцевого користувача.

2 РОЗРОБКА КОНЦЕПЦІЙ СТВОРЕННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ПЕРІОДИЧНИХ ТРАНЗАКЦІЙ

Особливістю періодичних транзакцій є те, що вони спрощують взаємодію між постачальниками послуг і споживачами, забезпечуючи стабільність фінансових потоків та зручність для обох сторін. Періодичні платежі знаходять своє застосування в багатьох сферах, включаючи комунальні послуги, підписки на медіа та сервіси, але їхня найбільша роль стає очевидною саме в контексті електронної комерції, про що детально йшлося у попередніх розділах.

Автоматизація періодичних транзакцій відіграє ключову роль у побудові ефективної бізнес-моделі. Вона дозволяє мінімізувати людський фактор у процесах обробки платежів, що значно знижує ризик помилок і підвищує точність виконання фінансових операцій. Крім того, автоматизовані системи забезпечують швидку обробку транзакцій, що сприяє оптимізації операційних витрат і підвищенню загальної ефективності.

Електронна комерція охоплює широкий спектр бізнес-моделей, які забезпечують взаємодію між різними учасниками ринку через цифрові платформи. Залежно від характеру відносин між сторонами, можна виділити декілька основних типів:

— B2B (Business-to-Business);

— B2C (Business-to-Consumer);

— C2C (Consumer-to-Consumer);

— B2G (Business-to-Government).

Кожна з цих моделей має свої особливості, які впливають на структуру, цілі та методи ведення бізнесу. Проте серед них найбільш динамічною та масовою є B2C-комерція, що орієнтована на кінцевого споживача.

На відміну від B2B, де основна увага приділяється великим транзакціям між компаніями, і C2C, яка зосереджена на взаємодії між приватними особами, B2C-комерція характеризується прямим продажем товарів і послуг кінцевим

користувачам. Ця модель є найбільш поширеною завдяки своїй універсальності та доступності, що робить її ключовим елементом цифрової економіки.

Особливість B2C-комерції полягає в орієнтації на великі обсяги транзакцій за рахунок значної кількості користувачів. У цій моделі бізнес повинен забезпечувати зручність і швидкість для кінцевого споживача, що є основним чинником успіху. Клієнти очікують інтуїтивно зрозумілого інтерфейсу, мінімальної кількості кроків для оформлення замовлення та швидкої доставки, що створює високі вимоги до автоматизації бізнес-процесів.

2.1 Вибір платформи для реалізації системи автоматизації періодичних транзакцій

Електронна комерція реалізується на різних платформах, кожна з яких відповідає специфічним потребам бізнесу та клієнтів. Веб-сайти тривалий час залишалися основою комерції, пропонуючи багатофункціональні рішення для широкої аудиторії. Вони забезпечують доступність з будь-якого пристрою, інтеграцію з платіжними системами та аналітичними інструментами, але все частіше відступають на другий план через зростаючий попит на мобільність і персоналізацію.

Через це мобільні додатки набрали особливої популярності завдяки можливості надавати споживачам більш інтуїтивний та індивідуальний підхід. Мобільний додаток — це не просто платформа для покупок, а потужний інструмент взаємодії з клієнтами, який забезпечує постійний доступ до послуг, миттєві повідомлення та легкість використання.

Однією з ключових переваг мобільних додатків є їхня інтеграція з функціями смартфона, такими як біометрична автентифікація, push-сповіщення та збереження платіжних даних. На відміну від веб-платформ, мобільний додаток дозволяє бізнесу бути ближчим до клієнтів завдяки постійній присутності на їхніх пристроях. Тому споживачі докладають мінімум зусиль під час оформлення замовлень, і мобільний додаток забезпечує це завдяки простому доступу до історії платежів, швидкому повторному замовленню та безперервній взаємодії.

Мобільний додаток є релевантнішим для розробки завдяки своїй здатності забезпечувати постійний доступ до сервісів у будь-який час і з будь-якого місця, що відповідає сучасному стилю життя. Він дозволяє ефективно використовувати функції смартфонів, такі як біометрична автентифікація чи push-сповіщення, для зручності користувачів. Завдяки компактному дизайну, оптимізованому під мобільні пристрої, додаток краще підходить для зручного перегляду та управління транзакціями.

2.2 Розробка архітектури комерційної платформи

Розробка комерційної платформи вимагає ретельного вибору архітектурного підходу, який забезпечить стабільність, гнучкість і масштабованість системи. Архітектура програмного забезпечення може мати різні стилі та моделі: клієнт-сервер, багатокомпонентна, безсерверна, монолітна або мікросервісна архітектура. Три останні види на сьогодні вважаються найпопулярнішими, хоча відрізняються структурою, способом реалізації та вимогами до інфраструктури.

Монолітна архітектура характеризується єдиною кодовою базою, де всі компоненти системи, такі як авторизація, обробка замовлень і управління товарами, інтегровані в один додаток (рисунок 2.1).

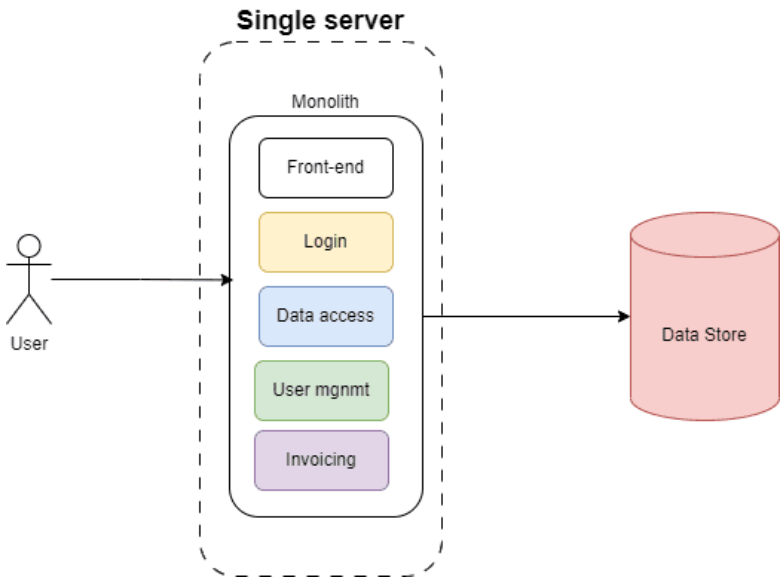

Рисунок 2.1 — Схематичне зображення монолітної архітектури

Такий підхід простий у реалізації на початкових етапах, оскільки всі модулі взаємодіють безпосередньо. Водночас, масштабування такого рішення може стати проблемою, оскільки для збільшення потужності потрібно розгортати повну копію програми, що збільшує використання ресурсів.

Мікросервісна архітектура розділяє функціональність системи на окремі, незалежні сервіси, кожен із яких відповідає за виконання конкретного завдання (рисунок 2.2). Наприклад, один сервіс може бути відповідальним за обробку платежів, інший — за управління товарами. Такий підхід дозволяє масштабувати окремі частини системи залежно від навантаження, що підвищує ефективність використання серверів [6].

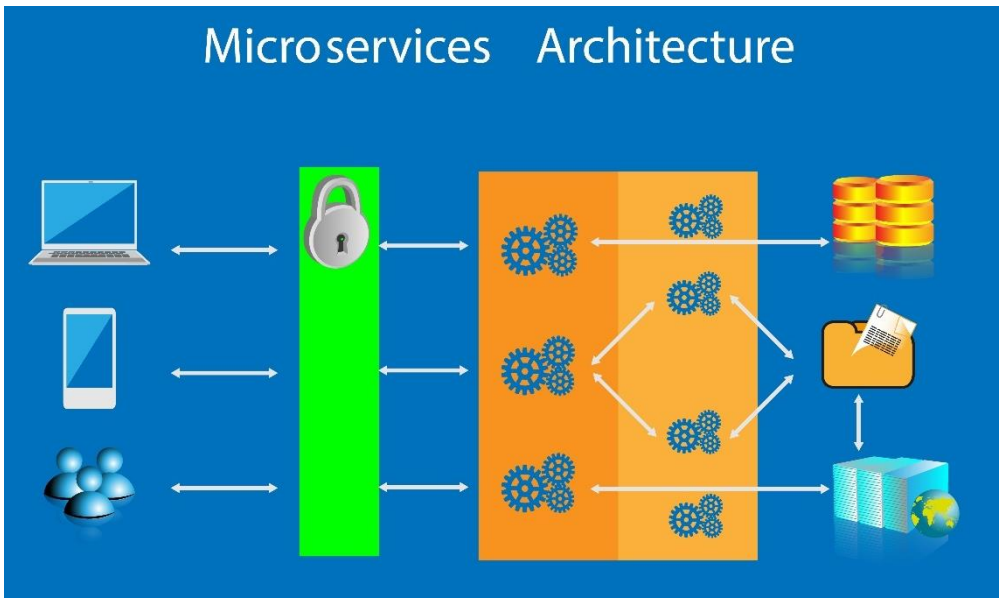

Рисунок 2.2 — Схематичне зображення мікросервісної архітектури

Безсерверна архітектура — це підхід до розробки програмного забезпечення з використанням безсерверних сервісів, що дозволяють розробникам не витрачати час на адміністрування серверів або ж для прикладу баз даних, а фокусуватися більше на написанні необхідних для роботи повноцінного застосунку функцій. Це означає, що адміністрування необхідних сервісів котрі використовуються для повноцінної роботи програмного продукту переходить до хмарних провайдерів даних сервісів. З основних пунктів адміністрування можна виділити такі як:

— обслуговування серверного обладнання;

— оновлення програмного забезпечення;

— оновлення систем безпеки;

— створення резервних копій.

Тому при використанні безсерверної архітектури у розробників звільняється більше часу на розробку програмного коду, що позитивно впливає на продуктивність роботи розробника або цілої команди. Також, при безсерверній архітектурі показники доступності, надійності, затримки та гарантії працездатності розробленого сервісу базуються на показниках зазначених самим хмарним провайдером [7].

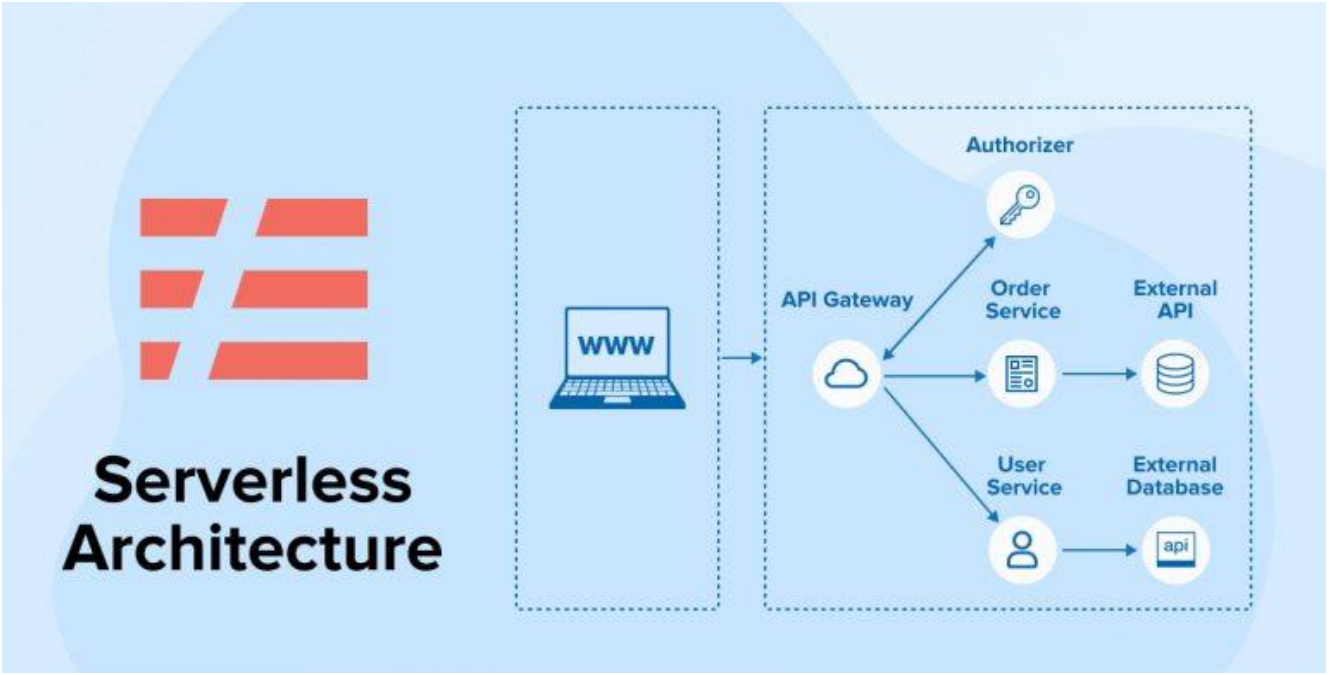

Рисунок 2.3 — Схематичне зображення безсерверної архітектури

Під час розгляду переваг архітектурних рішень, то для монолітного підходу є його простота. Для невеликих платформ із передбачуваним навантаженням це може бути найбільш економічно вигідним рішенням. Однак при зростанні кількості користувачів або функцій підтримка такої архітектури стає складнішою, оскільки будь-яка зміна може вплинути на всю систему. Мікросервіси, у свою чергу, дають змогу уникнути цього, адже кожен сервіс розвивається автономно.

Проте мікросервісна модель також має свої обмеження. Вона потребує складнішої інфраструктури для забезпечення зв’язку між сервісами, наприклад, за допомогою REST API або gRPC. Крім того, впровадження такої архітектури вимагає більш високого рівня компетенції команди розробників, оскільки включає оркестрацію контейнерів, таких як Docker і Kubernetes.

Порівнюючи безсерверну технологію із мікросервісами, можна зазначити, що безсерверна підходить для більш динамічних і малозатратних рішень, тоді як мікросервіси ефективні для великих платформ із постійним потоком користувачів. У той же час, монолітна архітектура все ще залишається актуальною для стартапів, які бажають швидко вийти на ринок із мінімальними витратами [8].

Для кращої оцінки переваг в таблиці 2.1 наведено порівняльну характеристику всіх трьох підходів.

Таблиця 2.1 — Порівняльний огляд монолітної, мікросервісної та безсерверної архітектур

Характеристика	Монолітна архітектура	Мікросервісна архітектура	Безсерверна архітектура
Гнучкість та масштабованість	Простота розробки та розуміння коду, легкість розгортання, але обмеження у гнучкості зі збільшенням функціоналу.	Висока гнучкість і масштабованість, але складності в управлінні мережею та розгортанні.	Автоматичне масштабування та адаптація до навантаження, але залежність від провайдера хмарних сервісів.
Управління даними	Простота процесу, оскільки інформація зберігається у базі, але керувати змінами у її структурі складно.	Гнучкість у виборі баз даних для кожного сервісу, зменшення залежності від єдиної бази, але потрібне уважне планування.	Дані можуть зберігатися в хмарних сервісах, що спрощує процес, але обмежує контроль.

Продовження таблиці 2.1

Складність розгортання та оновлення	Легко розгортати, оскільки всі компоненти інтегровані, але важко оновлювати, бо зміни можуть стосуватися всього кодового базису.	Легкість в оновленні окремих сервісів, стійкість до відмов у разі проблем з одним із них, але для управління версіями потрібно більше зусиль.	Мінімальні зусилля на розгортання, оскільки функції запускаються на вимогу, але складніше відстежувати версії та логіку.

Ключовим фактором у виборі архітектури є масштабованість. Мікросервісна модель дозволяє масштабувати окремі компоненти незалежно, тоді як моноліт вимагає масштабування всієї системи, що є менш ефективним. Безсерверна архітектура також забезпечує масштабованість, але для неї характерні більші затримки через запуск функцій.

Іншим важливим аспектом є витрати на розробку й підтримку. Моноліт дешевший на початкових етапах, оскільки потребує менше зусиль для налаштування. Мікросервіси та серверлес, навпаки, потребують більше часу й ресурсів для впровадження, але ці витрати окупаються за рахунок простоти масштабування й підтримки.

Безпека також відіграє важливу роль у виборі архітектури. Мікросервісна модель дозволяє ізолювати вразливі компоненти, наприклад, обробку платежів, тоді як у моноліті будь-яка вразливість може становити загрозу для всієї системи. Безмерверна покладається на хмарного провайдера, що забезпечує високий рівень безпеки, але обмежує контроль над інфраструктурою.

Для комерційної платформи з великою кількістю функцій і перспективою розширення мікросервісна архітектура є найбільш оптимальним вибором. Вона дозволяє інтегрувати нові сервіси без ризику порушити роботу існуючих, а також забезпечує стабільність під час збільшення навантаження. Якщо ж платформа має обмежений бюджет і розрахована на невелику кількість користувачів, моноліт може стати хорошим стартовим рішенням.

У підсумку вибір архітектурного підходу залежить від вимог бізнесу, технічних ресурсів і очікуваних масштабів платформи. Мікросервіси забезпечують гнучкість і адаптивність, монолітна архітектура підходить для швидкої розробки, а безсерверна архітектура мінімізує витрати на інфраструктуру.

Для реалізації комерційної платформи задля дослідження найкращим варіантом буде використання безсерверної архітектури, оскільки вона дозволяє швидко налаштувати й масштабувати систему, мінімізуючи час і ресурси, необхідні для управління інфраструктурою.

2.3 Вибір методу аутентифікації користувача

Аутентифікація є одним із важливих компонентів під час реалізації комерційної платформи, що забезпечує ідентифікацію користувачів перед наданням доступу до їхніх акаунтів і ресурсів. Існує кілька підходів до реалізації цього процесу, кожен із яких має свої особливості, переваги та обмеження.

Одним із найпоширеніших методів є аутентифікація через логін і пароль. Користувач надає ці дані, які система перевіряє, порівнюючи із записами в базі даних. Щоб забезпечити безпеку, паролі перед збереженням шифруються за допомогою хеш-функцій, таких як bcrypt або Argon2. Перевагою цього методу є його простота та універсальність, адже він підтримується практично всіма системами. Однак зловмисники можуть атакувати базу даних, що створює ризик компрометації паролів, навіть якщо вони зашифровані [9].

Біометричні методи аутентифікації, такі як розпізнавання відбитків пальців, обличчя або голосу, стають дедалі популярнішими завдяки зручності для користувачів. Вони використовують унікальні фізіологічні особливості, які складно підробити. Цими особливостями можуть виступати:

— сканування відбитків пальців;

— розпізнавання обличчя;

— розпізнавання райдужної оболонки.

Система зазвичай співставляє біометричні дані, отримані в реальному часі, з раніше збереженими зразками. Перевага цього методу полягає у високому рівні безпеки, але його застосування обмежене на пристроях, які не підтримують відповідне апаратне забезпечення. Крім того, у разі втрати або змін у біометричних даних користувач може втратити доступ до свого акаунту.

Одноразові паролі (OTP) є ще одним популярним методом, який часто використовується як додатковий рівень захисту. OTP — це пароль, дійсний лише для одного сеансу входу до системи або транзакції на комп'ютерній системі або іншому цифровому пристрої. Система надсилає унікальний код на номер телефону або електронну пошту користувача, який необхідно ввести для підтвердження доступу. Цей код генерується динамічно і дійсний лише протягом короткого проміжку часу [10]. OTP забезпечує захист навіть у випадках, коли пароль було скомпрометовано. Однак, для роботи цього методу потрібен доступ до зовнішніх ресурсів, таких як мобільна мережа або e-mail-сервер, що може створити незручності для користувачів у разі технічних проблем.

Соціальна аутентифікація дозволяє користувачам входити на платформу за допомогою облікових записів Google, Facebook, Apple або інших популярних сервісів. Цей метод реалізується через протокол OAuth 2.0, який забезпечує безпечний обмін інформацією між платформою і стороннім провайдером. Соціальна аутентифікація значно спрощує процес входу, оскільки користувачеві не потрібно створювати новий акаунт і запам'ятовувати ще один пароль. Проте залежність від зовнішніх сервісів може створювати ризики, пов’язані з доступністю цих платформ або змінами в їхніх політиках.

Комбінація кількох методів, відома як багатофакторна аутентифікація (MFA), забезпечує найвищий рівень безпеки. Наприклад, користувач може вводити пароль і підтверджувати доступ через OTP або біометричні дані. Хоча такий підхід створює додатковий захист, він може здаватися складним і незручним для деяких користувачів, що може зменшити рівень привабливості платформи в їхніх очах.

Залежно від особливостей комерційної платформи, різні методи аутентифікації мають свої сильні та слабкі сторони. Паролі підходять для базових реалізацій, але вимагають додаткових заходів безпеки. Біометрія забезпечує зручність і високий рівень захисту, але залежить від апаратного забезпечення. OTP є хорошим вибором для підвищення безпеки, але залежить від доступності зовнішніх каналів зв'язку. Соціальна аутентифікація спрощує процес входу, але залежність від сторонніх провайдерів може стати проблемою.

У висновку слід зазначити, що для ефективної реалізації системи аутентифікації доцільно використовувати готові інструменти, які вже інтегрують найкращі практики безпеки та зручності. Використання готових рішень, забезпечить надійний і гнучкий підхід до аутентифікації, який відповідає потребам сучасних комерційних платформ.

2.4 Розробка процесу оформлення замовлень

Правильне створення та управління замовленнями визначає ефективність взаємодії між клієнтом і компанією. Замовлення, яке оформлюється користувачем, проходить через кілька етапів, починаючи з ініціації і закінчуючи доставкою чи отриманням товару. Для забезпечення коректної обробки й відстеження замовлень платформи використовують різні методи та технічні підходи.

Одним із базових способів управління є централізована модель, де вся інформація про замовлення зберігається у єдиній базі даних. У цій моделі кожне замовлення має унікальний ідентифікатор, що дозволяє легко відслідковувати його статус. Такий метод підходить для невеликих платформ із невеликою кількістю користувачів. Перевагою є простота реалізації та підтримки, однак цей метод не масштабується добре при великому обсязі замовлень.

Децентралізовані моделі, які використовуються на більших платформах, включають використання мікросервісів для обробки окремих аспектів замовлення. Наприклад, один сервіс може обробляти платіж, інший — перевіряти наявність товару на складі, а третій — забезпечувати логістику. Ця модель

забезпечує високу гнучкість і дозволяє масштабувати окремі частини системи. Проте недоліком є складність у налаштуванні та інтеграції всіх компонентів.

Іншим підходом є використання хмарних платформ для управління замовленнями. Сервіси, такі як Amazon Web Services (AWS) або Google Cloud, забезпечують готові рішення для створення й обробки замовлень. Вони інтегрують функції, такі як обробка платежів, оновлення статусів замовлень і управління доставкою. Головною перевагою цього методу є швидкість впровадження та висока доступність, але вартість використання хмарних ресурсів може бути значною.

Щодо станів замовлень, то вони зображені на рисунку 2.4. Першим статусом зазвичай є "Очікує обробки", що означає, що замовлення надійшло в систему, але ще не було перевірене. Далі замовлення може перейти в статус "Обробляється", що означає перевірку товару на складі та підтвердження платежу. Якщо всі етапи проходять успішно, замовлення отримує статус "Відправлено" або "Готується до доставки".

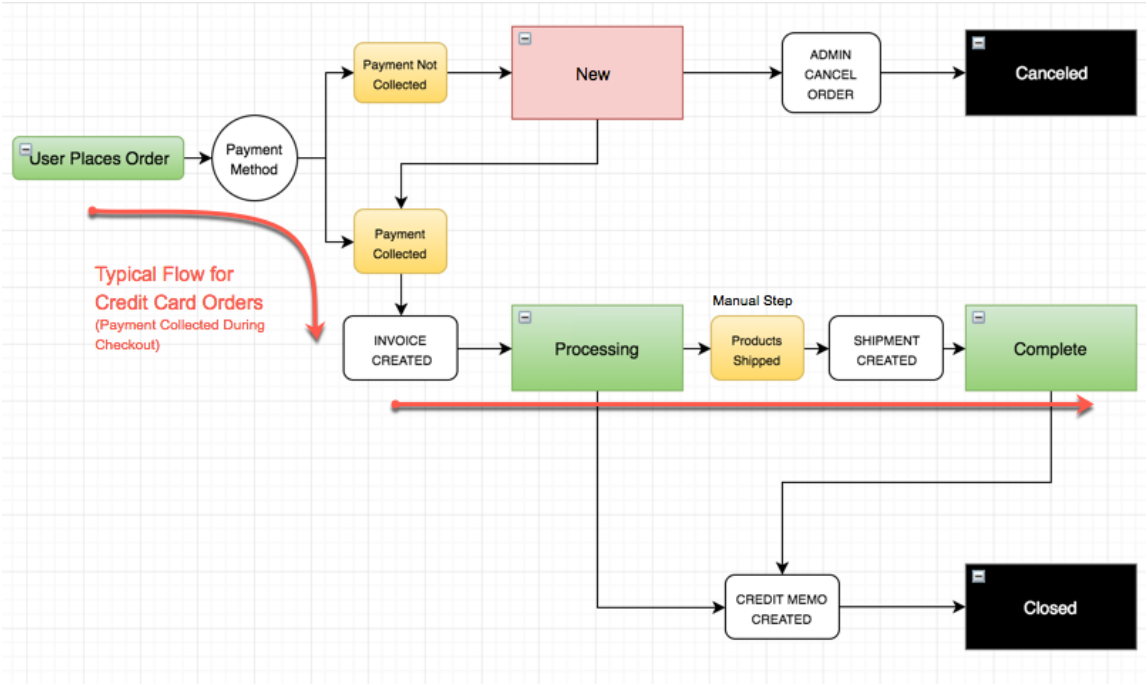

Рисунок 2.4 — Цикл стану замовлення

У разі проблем із товаром або оплатою замовлення може переходити в статус "Скасовано". Цей статус часто супроводжується коментарем для клієнта, який пояснює причину. Останнім етапом є "Доставлено", який сигналізує про успішне завершення всіх операцій.

Порівнюючи централізовану та децентралізовану моделі, можна зазначити, що перша є оптимальною для невеликих платформ із передбачуваним обсягом замовлень. Водночас мікросервісна архітектура підходить для великих компаній, які працюють із динамічним навантаженням. Хмарні сервіси пропонують високу надійність, але висока вартість та обмежений контроль можуть стати перешкодами для бізнесів, які прагнуть повної автономії та гнучкості в управлінні своїми системами.

Система сповіщень є невід’ємною частиною управління замовленнями. Push-сповіщення або SMS дозволяють клієнтам отримувати актуальну інформацію про статуси замовлень у режимі реального часу.

Порівнюючи підходи, варто зазначити, що централізовані системи є більш економічно доцільними в реалізації, але вони мають обмеження у масштабованості. Мікросервіси забезпечують гнучкість, але їхня розробка вимагає високої кваліфікації. Хмарні сервіси дозволяють швидко налаштувати інфраструктуру, однак обмежений контроль може бути недоліком.

Для більш ефективного управління замовленнями багато платформ використовують гібридні моделі. Наприклад, централізована база даних може поєднуватися з мікросервісами для обробки платежів або логістики. Такий підхід дозволяє балансувати між вартістю впровадження та функціональністю.Тому, важливо відзначити, що вибір методу створення й управління замовленнями залежить від масштабів платформи та її потреб. Для невеликих проєктів підходять централізовані моделі, тоді як великі компанії можуть використовувати мікросервісну архітектуру або хмарні сервіси.

2.5 Реалізація системи push-сповіщень

Push-повідомлення — це коротке спливаюче повідомлення в додатках або браузерах. Його відправляють користувачам, щоб розповісти про оновлення, новини та акції. Головна ціль push повідомлень — доставити клієнту релевантну інформацію для підримання залученості [11].

Мобільні push-сповіщення доступні для користувачів, які завантажили додаток і підтвердили отримання повідомлень. Мобільні повідомлення (in-app повідомлення) допомагають вчасно інформувати про оновлення, направляти підписників в потрібні розділи додатку та надавати короткі інструкції. Основними платформами для отримання push повідомлень в додатках є Android і iOS.

Згідно із RubyGarage, мобільні push повідомлення збільшують рівень взаємодії з додатком на 88% і в три рази підвищують показник утримання клієнтів. Крім того, користувачі, які вибрали отримання push повідомлень, запускають додаток в три рази частіше.

Довжина повідомлень залежить від мобільної операційної системи. Для iOS також грає роль тип повідомлення. Максимальна довжина варіюється від 62 символів для промо-повідомлень до 235 символів для сповіщень. На пристроях Android обмеження становить близько 84 символів в залежності від розміру екрана.

З технічного боку, наприклад, для IOS користувачів спочатку додаток відправляє запит на пристрій, а потім він передає його службі push повідомлень Apple (APNS), яка у відповідь відсилає токен пристрою. Девайс перенаправляє токен в додаток, який переміщує його в Backend. Після цього токен пристрою передається назад APNS разом з самим повідомленням. Тільки після цього повідомлення досягає девайса і користувач може його побачити [12]. Дана схема реалізації пуш-повідомлень зображена на рисунку 2.5.

З точки зору реалізації push-сповіщень в електронній комерції важливо відмітити, що вони є важливим інструментом для взаємодії з користувачами, оскільки забезпечують миттєву і пряму комунікацію. Цей механізм дозволяє інформувати клієнтів про статуси замовлень, спеціальні пропозиції, нагадування

чи інші важливі події в реальному часі, навіть коли додаток неактивний. Завдяки персоналізації та цілеспрямованості повідомлень бізнес може підвищити рівень залучення користувачів, зміцнити їхню лояльність і стимулювати повторні покупки.

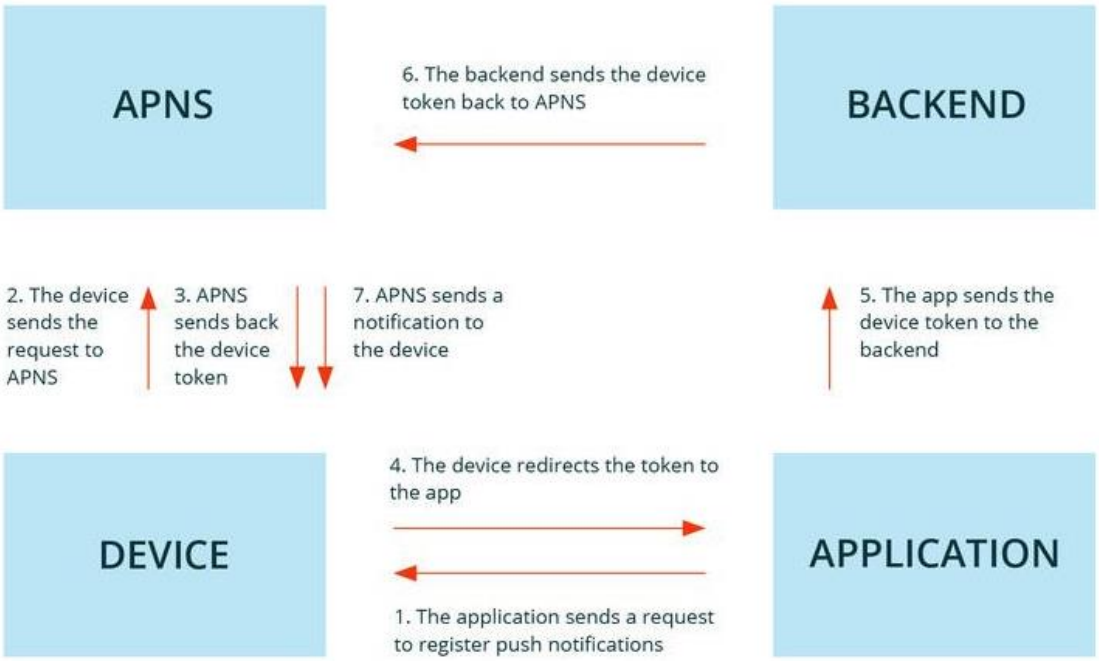

Рисунок 2.5 — Схема роботи push-повідомлень для ОС iOS

2.6 Інтеграція QR-коду в користувацький інтерфейс

QR-код, або код швидкого реагування, є двовимірним графічним зображенням, яке використовується для кодування інформації у форматі, що легко зчитується за допомогою камер смартфонів чи спеціальних сканерів. Ця технологія була вперше розроблена японською компанією Denso-Wave у 1994 році, як спосіб оптимізації облікових операцій у виробництві [13]. Сьогодні QR-коди є універсальним інструментом для обміну даними, що знаходить застосування у багатьох галузях, зокрема комерції.

На відміну від одномірних штрих-кодів, QR-коди мають можливість зберігати більше інформації завдяки своїй двовимірній структурі. Вони складаються з чорних квадратів та білих проміжків, організованих у матрицю. Алгоритми обробки забезпечують не лише високу щільність даних, але й

дозволяють виправляти помилки, що виникають під час зчитування, наприклад, через фізичні пошкодження коду.

Ключовими елементами структури QR-коду є три великі квадрати у верхніх і нижньому лівих кутах, які визначають орієнтацію коду. Вони працюють як маркери для сканерів, дозволяючи швидко виявити розташування і кут нахилу зображення. Інші частини QR-коду містять зашифровані дані, інформацію про версію, формат і корекцію помилок.

Система кодування QR дозволяє зберігати до 4296 символів тексту або 7089 цифр, залежно від типу використовуваного формату. Завдяки цьому QR-коди стали популярними у багатьох контекстах, включаючи розміщення URL-адрес, контактних даних, ідентифікаційних номерів, а також інтеграцію з платіжними системами.

Однією з важливих переваг цієї технології є простота її реалізації. QR-коди можуть бути створені та роздруковані або відображені на цифрових екранах для використання в широкому спектрі задач. Сканування можливе навіть за низької якості зображення чи за наявності перешкод, що робить цю систему надзвичайно надійною.

У комерції QR-коди використовуються для оптимізації обслуговування клієнтів. Наприклад, роздрібні магазини розміщують QR-коди на упаковці продуктів, які містять інформацію про товар, такі як його склад, інструкції з використання або сертифікацію. Це спрощує доступ до даних для споживачів і підвищує їхню обізнаність [14].

У сфері електронної комерції QR-коди використовуються для оптимізації процесів замовлення та доставки. Вони можуть містити інформацію про замовлення, що полегшує його обробку на складах і пришвидшує логістичні операції. Наприклад, сканування коду може автоматично створювати накладну або оновлювати статус доставки в реальному часі.

У сфері онлайн-платежів QR-коди стали стандартним інструментом. Вони використовуються для зручної ідентифікації платежів, особливо у системах мобільного банкінгу. Наприклад, скануючи код, клієнт автоматично отримує

заповнену форму з платіжними реквізитами, що знижує ризик помилок і пришвидшує процес.

Ще одним прикладом їхнього використання є інтеграція в програми лояльності. Наприклад, клієнти можуть сканувати QR-коди на касах, щоб автоматично нараховувати бали чи застосовувати знижки. Це підвищує рівень взаємодії між брендом і споживачем.

У маркетингу QR-коди використовуються як інструмент для залучення уваги до рекламних кампаній. Вони часто розміщуються на банерах, листівках чи упаковці товарів і можуть перенаправляти користувачів на сторінки із рекламними кампаніями, відео чи в мобільні додатки. Завдяки своїй інтерактивності QR-коди дозволяють брендам безпосередньо комунікувати зі споживачами.

У сфері післяпродажного обслуговування QR-коди застосовуються для спрощення доступу до технічної підтримки або гарантійного обслуговування. Сканування коду дозволяє клієнтам швидко знайти інструкції, подати заявку на ремонт або зв’язатися зі службою підтримки.

Розробники часто використовують QR-коди для інтеграції офлайн- і онлайн-активностей. Наприклад, у мобільних додатках сканування коду може перенаправляти користувача до певної функції додатка або запускати процес реєстрації.

Попри свої численні переваги, QR-коди мають і певні недоліки. Наприклад, для їх зчитування необхідний смартфон або сканер, що може обмежити доступ для користувачів без таких пристроїв. Крім того, погане освітлення чи пошкодження коду можуть ускладнити процес сканування.

Зважаючи на всі аспекти використання QR-кодів, за допомогою них можна вдосконалити систему періодичних транзакцій. Покращення можна створити за допомогою того, що після оформлення замовлення користувач перенаправляється на сторінку із QR-кодом, який містить собі номер даного замовлення та посилання на сторінку, яка автоматично оформлює покупку із тими самими даними про доставку та оплату базуючись на номері замовлення.

Таким чином можна уникнути надмірного споживання товарів, тому що підписка на продукції гарантує своєчасне надходження товару, не зважаючи на те чи попередній був використаний повністю. Тобто, користувач автоматично може сформувати повторне замовлення лише за допомогою сканування QR-коду і лише тоді, коли йому це буде необхідно.

З точки зору бізнесу, таке рішення може допомогти привабити користувачів оформити повторне замовлення ще раз, оскільки для того, щоб це зробити потрібно лише відкрити камеру і по можливості авторизуватись у застосунку.

3 РОЗРОБКА СИСТЕМИ АВТОМАТИЗАЦІЇ ПЕРІОДИЧНИХ ТРАНЗАКЦІЙ

3.1 Розробка схеми роботи додатку

Для створення комерційного додатку необхідно розробити декілька ключових компонентів, кожен з яких виконує важливу роль у забезпеченні функціональності платформи. Аутентифікацію потрібно розробити, тому що вона забезпечує доступ до індивідуального профілю користувача та зберігання персональних даних. Без цього компоненту неможливо гарантувати персоналізацію процесів, таких як збереження історії замовлень або налаштування повторних транзакцій. Аутентифікація повинна включати можливість входу за допомогою логіна та пароля, а також реєстрацію нових користувачів.

Оскільки додаток є комерційною платформою, повинна бути сторінка, що надає можливість перегляду каталогу товарів. Каталог необхідно організувати таким чином, щоб користувач міг швидко знайти потрібні товари. Для цього потрібно реалізувати функцію фільтрації та пошуку за категоріями, ціною та іншими параметрами. Каталог слід інтегрувати зі сторінками детального опису товарів, щоб користувач мав доступ до ключової інформації про продукт, включаючи його характеристики, фотографії та відгуки інших покупців. Це дозволить спростити процес вибору товару та підвищити рівень задоволеності користувача.

Процес оформлення замовлення потрібно організувати так, щоб він був максимально зручним для користувача. Першим етапом повинно бути введення даних для доставки. Наступним кроком - введення платіжних реквізитів. Перегляд підсумованих введених даних повинен бути останнім кроком для оформлення замовлення задля того, щоб користувач зміг все перевірити і у випадку помилки — виправити її. Лише після цього замовлення може бути оформлене.

Після самого процесу оформлення, потрібно проінформувати користувача про успішне завершення процесу, тому для цього потрібно розробити окрему

відображення. Дана сторінка повинна містити дані про створену покупку та унікальний QR-код, який в майбутньому повинен спростити процес повторного замовлення, дозволяючи швидко перейти до останнього етапу оформлення з попередньо заповненими даними.

Функціонал для реалізації періодичних транзакцій повинен включати кілька елементів. Необхідно додати можливість, щоб після сканування QR-коду із зашифрованими даними про замовлення, користувач був автоматично перенаправлений на сторінку вже підготованої покупки базуючись на інформації про попереднє замовлення. Також є можливість того, що на момент сканування QR-коду, користувач може бути неаутентифікованим. В такому випадку, потрібно запам’ятати дані, які були щойно відскановані та використати їх після успішного входу в систему.

Також важливо передбачити кнопку "Перезамовити" в акаунті користувача, що дозволить виконати аналогічну операцію без сканування. Для підвищення зручності користувачів доцільно впровадити функцію сповіщень, які нагадуватимуть про необхідність оновлення або підтвердження замовлення.

Базуючись на проаналізованих функціях, блок-схема роботи додатку буде виглядати так, як зображено на рисунку 3.1. Дана схема демонструє послідовність дій від авторизації до оформлення користувачем замовлення. Робота із системою починається з виконання обробки запиту на авторизацію або реєстрацію, створюючи чи перевіряючи облікові записи та зберігаючи відповідні дані. Після цього завантажується каталог, активуються модулі для пошуку та фільтрації товарів, які оптимізують доступ до асортименту. На етапі оформлення система приймає інформацію про доставку, платіжні реквізити та формує сторінку з підсумком замовлення. Після підтвердження створюється унікальний QR-код, що зберігає ключові параметри транзакції для майбутнього використання. Повторні запити обробляються через сканування цього коду або активацію функції "Перезамовити", завантажуючи збережені дані для швидкого завершення процесу. Закриття сесії відбувається після виходу, що забезпечує очищення тимчасових ресурсів і завершення взаємодії.

Рисунок 3.1 — Блок-схема роботи додатку

3.2 Налаштування інструментів для розробки

Для розробки додатку перш за все потрібно згенерувати правильний вибір середовища розробки, мови програмування та фреймворку. Ці компоненти формують технологічну базу проєкту, від якої залежатиме продуктивність процесу розробки, гнучкість у реалізації функціоналу та ефективність підтримки додатку в майбутньому.

Вибір мови програмування для мобільного додатку є ключовим рішенням, яке впливає на всі аспекти його розробки та експлуатації. Нативні мови, такі як Swift для iOS та Kotlin для Android, забезпечують найкращий рівень інтеграції з апаратними можливостями пристрою, дозволяючи використовувати всі переваги мобільної платформи. Вони надають повний контроль над процесом створення

додатку та ідеальну продуктивність. Проте така глибока інтеграція має свою ціну: розробка окремих версій додатку для кожної платформи значно збільшує витрати часу та ресурсів, що може бути критичним для багатьох проєктів.

Кросплатформені рішення пропонують альтернативу, що дозволяє створювати додатки для кількох платформ одночасно, використовуючи спільний код. Серед них Flutter є однією з популярних технологій, яка використовує мову Dart. Цей підхід забезпечує швидкий процес розробки завдяки миттєвому оновленню змін у коді без необхідності перезавантаження програми, а також багатий набір візуальних компонентів для створення інтерфейсу. Однак, мова Dart є менш відомою серед розробників, що може ускладнити залучення команди або збільшити час адаптації до нових технологій.

React Native вирізняється тим, що поєднує переваги кросплатформенного підходу з використанням однієї з найпоширеніших мов програмування — JavaScript. React Native, що побудований на базі React, дає змогу створювати динамічні інтерфейси з компонентною архітектурою [15]. Завдяки підтримці нативних модулів цей фреймворк забезпечує тісну інтеграції з платформами Android та iOS, дозволяючи використовувати функції пристрою, такі як камера, GPS або датчики руху.

Додатковою перевагою React Native є велика спільнота розробників, що активно підтримує технологію, створюючи численні бібліотеки та готові рішення, які прискорюють розробку. Гнучкість платформи дозволяє легко адаптувати додаток до змін у функціональних вимогах або специфіці цільової аудиторії.

Гібридні технології, такі як Ionic чи Apache Cordova, також дозволяють створювати мобільні додатки на основі веб-технологій, однак їх продуктивність і якість інтерфейсу часто поступаються нативним та кросплатформеним рішенням. Ці підходи можуть бути доцільними для проєктів із базовими вимогами, але вони навряд чи задовольнять складні технічні чи функціональні задачі.

Аналізуючи всі доступні варіанти, React Native виглядає найбільш оптимальним рішенням для сучасних мобільних додатків. Однак, щоб максимально спростити процес розробки та ефективно використовувати

можливості цієї технології, важливо обрати правильний набір інструментів. Серед інструментів, які доповнюють React Native, Expo вирізняється своєю зручністю та широкими можливостями, що роблять його ідеальним вибором для багатьох проєктів.

Expo надає інтегроване середовище розробки, яке усуває необхідність налаштовувати складні нативні середовища для роботи з React Native. Це дозволяє зосередитися на створенні функціоналу, замість витрачання часу на технічні деталі конфігурації.

Фреймворк містить численні готові модулі, такі як інтеграція з камерою, push-сповіщеннями та доступ до сенсорів пристрою, що значно пришвидшує впровадження ключових функцій додатку.

Крім того, Expo забезпечує швидкий запуск і тестування додатків за допомогою функції "live reload", що дозволяє миттєво перевіряти зміни в коді на реальному пристрої або емуляторі [16]. Через те, що задля перевірки результату відпрацювання коду розробленого для iOS на емуляторі потрібно розробляти на системі macOS, Expo своєю функцією «live reload» надає можливість запускати код на реальному девайсі із операційною системою iOS, що значно спрощує процес створення додатку. Також варто зазначити, що платформа пропонує спрощений процес публікації додатків у магазинах App Store і Google Play, що є важливою перевагою для швидкого виведення продукту на ринок.

За середовище розробки було обрано Visual Studio Code, оскільки воно надає зручні та потужні інструменти для роботи з React Native у поєднанні з Expo. Visual Studio Code пропонує інтеграцію з популярними розширеннями, такими як React Native Tools, які забезпечують налагодження коду, автоматичне завершення, і підсвічування синтаксису, оптимізуючи процес написання та тестування додатка [17]. Завдяки вбудованому терміналу та підтримці командної роботи через Git, це середовище дозволяє ефективно запускати команди Expo, такі як створення проекту, запуск симулятора чи налаштування збірок. Завдяки простоті використання та високій продуктивності, це середовище є ідеальним вибором для створення сучасних мобільних додатків.

Також для реалізації додатку потрібно визначитись із тим, які хмарні сервіси будуть найкращими для реалізації серверної частини застосунку. Firebase від Google є одним із найефективніших рішень у цій сфері, оскільки пропонує широкий набір інструментів і послуг.

Firebase легко інтегрується з мобільними платформами, написаним на React Native, що дозволяє легко використовувати всі функціональні можливості без необхідності розробки складної серверної логіки. Гнучкість платформи забезпечує швидке підключення сервісів, таких як автентифікація, хмарне зберігання даних, функції реального часу та управління аналітикою [18]. Це дозволяє мінімізувати час на створення серверної частини та сконцентруватися на вдосконаленні користувацького інтерфейсу.

Аутентифікація є першим і ключовим елементом взаємодії користувача з додатком. Використовуючи Firebase Authentication, можна впровадити безпечну та гнучку систему входу, яка підтримує різні методи ідентифікації. Крім традиційного логіна і пароля, цей інструмент дозволяє інтегрувати авторизацію через популярні сервіси, такі як Google, Facebook або Apple ID. Завдяки інтеграції цього сервісу, користувачі отримають зручний доступ до своїх облікових записів, а додаток забезпечить високий рівень безпеки даних.

Після впровадження аутентифікації важливим етапом є організація зберігання та управління даними. Firebase пропонує Realtime Database і Cloud Firestore, які забезпечують зчитування та запис інформації у хмару в реальному часі. Це дозволяє створювати динамічні додатки, у яких зміни даних моментально синхронізуються між користувачами. Додатковою перевагою є підтримка офлайн-доступу, яка гарантує стабільну роботу навіть у випадках відсутності інтернет-з’єднання, що підвищує довіру до додатку з боку користувачів.

Також важливим компонентом є генерація пуш-сповіщень, яку можливо реалізувати через Firebase Cloud Messaging (FCM). Даний сервіс дозволить надсилати як індивідуальні, так і масові повідомлення, що зробить комунікацію з користувачами зручною та ефективною.

3.3 Розробка модулю аутентифікації

Аутентифікація — це процес перевірки особи користувача або пристрою, що дає змогу отримати авторизований доступ до конфіденційної інформації або системи. Сучасні технології пропонують різноманітні підходи до автентифікації, включаючи класичний логін і пароль, автентифікацію через сторонні сервіси (OAuth), а також біометричні методи, такі як розпізнавання обличчя або відбитків пальців.

Для швидкої та практичної реалізації аутентифікації в додатку було вирішено використати вже готове рішення від Firebase. Для того, щоб даним сервісом можна було скористатись, для початку потрібно зареєструвати свій проєкт в консолі Firebase. Після реєстрації, з'явилась можливість підключити сервіс аутентифікації та вибрати його методи (рис. 3.2).

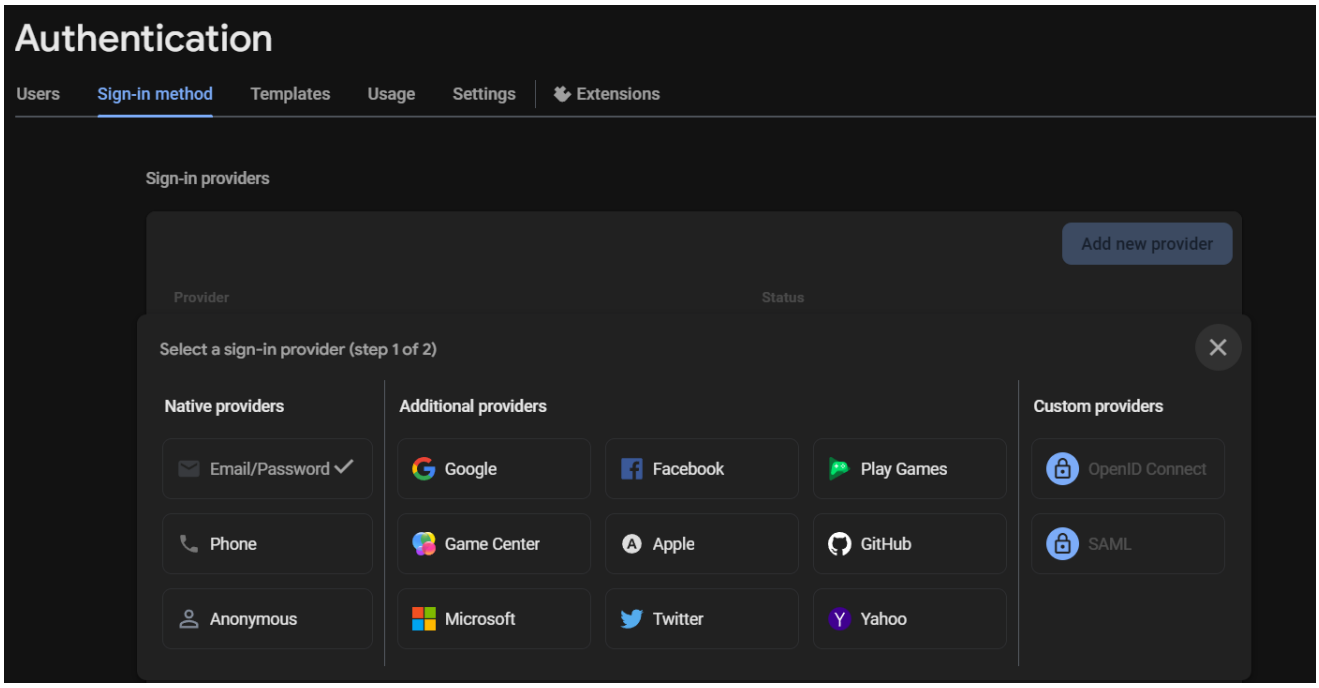

Рисунок 3.2 — Увімкнення сервісу автентифікації на платформі Firebase

Для даного додатку було розроблено стандартну аутентифікацію через логін та пароль. Метод, який реалізовує процес логіну в систему описано в лістингу 3.1.

Лістинг 3.1 — Лістинг функції, яка опрацьовує логін користувача в систему

```
const handleLogin = async () => {
        setIsLoading(true);
        setErrorMessage('');
        try {
            const response = await
signInWithEmailAndPassword(FIREBASE_AUTH, email, password);
            if (orderId) {
                navigation.navigate("checkout/[orderId]", { orderId:
orderId })
            } else {
                navigation.navigate('shop');
            }
        } catch (error) {
            console.log(error);
            if (error.code === 'auth/invalid-credential') {
                setErrorMessage('Перевірте правильність введених
даних');
            } else {
                setErrorMessage('Щось пішло не так. Спробуйте ще
раз.');
            }
        } finally {
            setIsLoading(false);
        }
    };
```

Функція handleLogin призначена для виконання авторизації користувача через Firebase із подальшим перенаправленням до відповідного розділу додатку. На початку виконується оновлення стану компонента: змінна isLoading встановлюється в true за допомогою методу setIsLoading, який дістається із глобального контексту, що дозволяє відобразити індикатор завантаження для користувача, а errorMessage очищується, забезпечуючи відсутність повідомлень про попередні помилки.

Далі функція виконує основну операцію входу за допомогою методу signInWithEmailAndPassword від бібліотеки «firebase/auth», передаючи у Firebase API дані для входу: пошту та пароль. Використання ключового слова await визначає, що виконання функції зупиняється до отримання відповіді від серверу. Якщо авторизація проходить успішно, функція перевіряє наявність змінної orderId, яка може бути передана для визначення контексту дій користувача. У разі наявності orderId, користувача перенаправляють до сторінки оформлення

замовлення через метож navigation.navigate від бібліотеки «@react-navigation/native», передаючи значення ідентифікатора як параметр. Якщо ж orderId відсутній, перенаправлення відбувається до розділу магазину.

У разі виникнення помилки, її обробка здійснюється у блоці catch. Помилки, пов’язані з некоректними даними для входу, ідентифікуються за кодом і користувачеві відображається відповідне повідомлення.

Завершує функцію блок finally, який виконується незалежно від того, чи авторизація була успішною, чи завершилася помилкою. У ньому стан isLoading встановлюється у false, сигналізуючи про завершення процесу і приховуючи індикатор завантаження. Повна реалізація компоненти логіну зображена в додатку Б.

Для процесу реєстрації застосовується схожа логіка, але в даному випадку використовується функція createUserWithEmailAndPassword від тієї ж бібліотеки «firebase/auth» (лістинг 3.2).

Лістинг 3.2 — Лістинг функції, яка опрацьовує реєстрацію користувача в системі

```
const handleRegister = async () => {
   setIsLoading(true);
   setErrorMessage('');
   try {
     const response = await
createUserWithEmailAndPassword(FIREBASE_AUTH, email, password);
     const user = response.user;
     await updateProfile(user, {displayName: `${firstName}
${lastName}`,});
            navigation.navigate('shop');
        } catch (error) {
            if (error.code === 'auth/email-already-in-use') {
setErrorMessage('Акаунт із такою електронною поштою вже існує');
            } else if (error.code === 'auth/invalid-email') {
setErrorMessage('Неправильний формат електронної пошти');
            } else if (error.code === 'auth/weak-password') {
setErrorMessage('Пароль має бути щонайменше 6 символів.');
            } else {
setErrorMessage('Щось пішло не так. Спробуйте ще раз.');
            }
        } finally {
            setIsLoading(false);
        }
    };
```

Даний метод використовується для реєстрації облікового запису на основі електронної пошти та пароля. Якщо операція завершиться успішно, змінна response міститиме об’єкт із деталями нового користувача, включаючи унікальний ідентифікатор (UID), який можна використовувати для подальшого управління обліковим записом. Також, для того, щоб запам’ятати ім’я користувача, після успішної реєстрації виконується метод updateProfile, за допомогою якого можна змінити атрибут користувача displayName для подальшого використання.

Повна реалізація компоненти реєстрації зображена в додатку В. На рисунку 3.3 зображено результат виконання компонент Login та Registration.

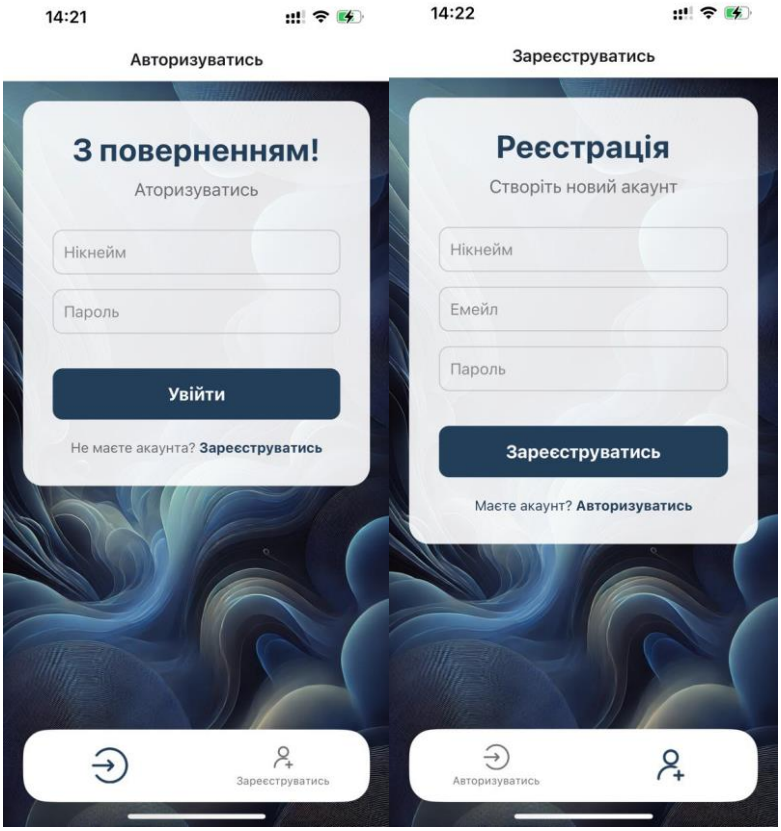

Рисунок 3.3 — Результат відображення компонент Login та Registration

3.4 Розробка сторінки каталогу

Сторінка каталогу повинна виконувати основні дві речі: показувати продукти та допомагати із пошуком потрібного товару. Тому на даній сторінці

було розроблено фільтрацію продуктів за продавцями, а також динамічне завантаження товарів із хмарного сховища.

Функція fetchProducts, що описана в лістингу 3.3 відповідає за завантаження даних про товари з колекції products, розташованої у базі даних Firestore. Ця асинхронна функція забезпечує отримання списку продуктів, їх обробку та збереження у локальному сховищі додатку, а також формує список унікальних категорій товарів.

Робота функції починається з блоку try, де виконується основна логіка завантаження. За допомогою методу getDocs від Firebase Firestore зчитується колекція products, розташована у базі даних, переданій через змінну FIRESTORE_DB. Результат зчитування зберігається у змінній querySnapshot, яка містить набір документів із бази.

Лістинг 3.3 — Лістинг функції, що забезпечує отримання списку продуктів із БД

```
const fetchProducts = async () => {
      try {
        const querySnapshot = await getDocs(collection(FIRESTORE_DB,
"products"));
        const fetchedProducts = querySnapshot.docs.map((doc) => ({
          id: doc.id,
          ...doc.data(),
        }));
        setProducts(fetchedProducts);

        const fetchedCategories = ["All", ...new
Set(fetchedProducts.map((item) => item.seller))];
        setCategories(fetchedCategories);
      } catch (error) {
        console.error("Помилка завантаження товарів:", error);
      } finally {
        setIsLoading(false);
      }
    };
```

Далі отримані дані обробляються за допомогою методу map. Кожен документ перетворюється у об'єкт із включенням унікального ідентифікатора документа (id) та всіх інших даних, зчитаних через метод doc.data(). Ці об'єкти зберігаються у змінній fetchedProducts, яка представляє собою список усіх

товарів. Цей список передається у функцію setProducts для збереження у стані додатку.

Після завантаження товарів відбувається створення масиву із доступних для фільтрації категорій. За допомогою методу map із кожного товару отримується значення поля seller, що позначає продавця, а далі використовується new Set, щоб видалити дублікати. Результат додається до масиву із загальною категорією "All" і передається у функцію setCategories для збереження у локальному сховищі.

Якщо під час виконання функції виникає помилка, вона обробляється у блоці catch, який виводить текст повідомлення про помилку у консоль. Незалежно від результату роботи, у блоці finally змінна стану setIsLoading встановлюється у false, сигналізуючи про завершення процесу завантаження.

Повний лістинг компоненти Shop додано до додатку Г, а результат відображення результату рендеру компоненти зображено на рисунку 3.4.

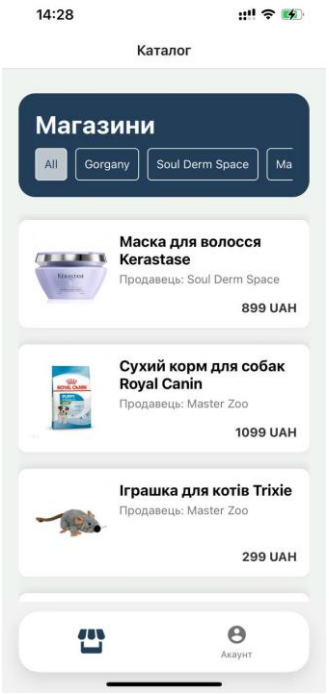

Рисунок 3.4 — Результат відображення компоненти Shop

Якщо ж користувач захоче більш детальніше ознайомитись із продуктом, то після натиснення на картку товару його буде перенаправлено на сторінку деталей. Для того, щоб відобразити повну інформацію про продукт, в компоненту сторінки

передається ідентифікатор продукту та робиться асинхронний запит в базу даних, який описано в лістингу 3.4.

Рисунок 3.4 — Лістинг функції отримання інформації про продукт із БД

```
const fetchProduct = async () => {
       try {
const docRef = doc(FIRESTORE_DB, "products", productId);
            const docSnap = await getDoc(docRef);
            if (docSnap.exists()) {
                setProduct(docSnap.data());
                setQuantity(docSnap.data().minOrderQuantity || 1);
                } else {
console.error("Продукт із заданим ID не знайдено!");
                }
            } catch (error) {
console.error("Помилка завантаження продукту:", error);
            } finally {
                setLoading(false);
            }
        };
```

На початку функції у блоці try створюється посилання на документ у колекції products бази даних Firestore за допомогою методу doc, де FIRESTORE_DB вказує на екземпляр бази даних, а productId є унікальним ідентифікатором потрібного продукту. Потім за допомогою методу getDoc здійснюється спроба отримати вміст цього документа, результат зберігається у змінній docSnap.

Після цього функція перевіряє, чи існує документ, використовуючи метод exists() на об'єкті docSnap. Якщо документ існує, викликається метод docSnap.data(), який повертає об'єкт із даними продукту. Цей об'єкт передається у функцію setProduct для оновлення стану продукту в додатку. Крім того, з отриманих даних зчитується значення мінімальної кількості замовлення (minOrderQuantity). Якщо це поле відсутнє, використовується значення за замовчуванням — 1, і відповідний стан оновлюється через setQuantity.

У випадку помилки під час виконання запиту, наприклад, через проблеми з мережею чи доступом до бази, вона обробляється у блоці catch, а текст помилки відображається у консолі. Незалежно від успішності або помилки виконання, у

блоці finally стан setLoading встановлюється у false, що сигналізує про завершення процесу завантаження для користувача.

Повний код компоненти Details зображено в додатку Д, а результат відображення результату рендеру компоненти зображено на рисунку 3.5.

Рисунок 3.5 — Результат відображення компоненти Details

3.5 Розробка сторінки для оформлення замовлення

Компонент для оформлення замовлення відповідає за реалізацію багатокрокового процесу, що включає внесення даних про доставку, оплату та підтвердження замовлення. Він забезпечує зручність для користувача, динамічно

відображаючи різні етапи замовлення залежно від поточного стану, а також інтегрується з базою даних для завантаження та збереження інформації.

На початковому етапі компонент перевіряє наявність ідентифікаторів продукту та замовлення, які передаються до нього. На основі цих параметрів виконуються запити до бази даних для отримання інформації про вибраний товар та існуюче замовлення. У випадку, якщо замовлення вже створено і користувач, який на даний момент аутентифікований, його оформив, додатково завантажуються деталі доставки та оплати. Це дозволяє автоматично заповнити відповідні форми та перенаправити користувача до фінального етапу — підтвердження коректності введених даних.

Для роботи з даними форми компонент використовує окремі етапи, кожен із яких відповідає за свій тип інформації. Наприклад, перший етап дозволяє ввести дані про доставку, другий — про оплату, а фінальний крок надає можливість переглянути та підтвердити замовлення. Дані, введені на кожному етапі, перевіряються на коректність і зберігаються для подальшого використання. Перехід між кроками можливий лише після завершення введення всіх необхідних даних, що мінімізує ризик помилок.

Процес створення замовлення завершується, коли користувач підтверджує усі введені дані. На даному етапі викликається функція handlePlaceOrder, яка описана в лістингу 3.5. Даний метод відповідає за створення нового замовлення в базі даних і обробку всіх необхідних дій для його успішного оформлення. Це включає збереження інформації про доставку, продукт і платіжні дані, генерацію унікального ідентифікатора замовлення, а також перенаправлення користувача до сторінки підтвердження.

Функція починає роботу з активації індикатора завантаження на сторінці, встановлюючи відповідний стан за допомогою хука useState від бібліотеки React. Це сигналізує користувачеві про початок процесу оформлення замовлення. Основна логіка реалізована у блоці try, який обробляє всі необхідні кроки для створення нового запису в базі даних.

Лістинг 3.5 — Лістинг функції, що опрацьовує створення замовлення

```
const handlePlaceOrder = async () => {
    setOrderLoading(true);
    const shippingAddressRef = await addDoc(
    collection(FIRESTORE_DB, "shippingAddresses"),
        {
          ...formData.delivery,
          userId: user.uid,
          status: "Опрацьовується",
        }
      );
    const newOrderId = await generateOrderId();
    const orderData = {
       id: newOrderId,
        userId: user.uid,
        product: {
          id: pid,
          name: product.name,
          price: product.price,
          quantity: qty,
          totalPrice: totalAmount,
          currency: product.currency,
        },
       shippingAddressId: shippingAddressRef.id,
        totalAmount,
        currency: product.currency,
        createdAt: Timestamp.now(),
        cardInfo: {
          cardNumber: formData.payment.cardNumber,
          expiryDate: formData.payment.expiryDate,
          cvv: formData.payment.cvv,
        }
      };
await setDoc(doc(FIRESTORE_DB, "orders", newOrderId), orderData);
navigation.navigate("myaccount/[orderId]", { orderId: newOrderId });
        resetFormData();
        setOrderLoading(false);
    };
```

Першим кроком є збереження даних про доставку. Використовуючи метод addDoc, функція додає новий запис до колекції shippingAddresses, передаючи туди всі дані, які користувач ввів на етапі доставки. Окрім стандартних полів, додається ідентифікатор користувача для зв’язування адреси з обліковим записом та початковий статус доставки зі значенням "Опрацьовується".

Далі функція викликає generateOrderId, яка генерує унікальний ідентифікатор замовлення. Виклик даної функції означає, що кожне замовлення матиме свій унікальний код, необхідний для його ідентифікації в системі.

Після цього формується об’єкт із даними замовлення, який включає інформацію про користувача, товар, адресу доставки, загальну суму замовлення та платіжні реквізити. Поля цього об’єкта наповнюються даними з форми, а також метаданими, такими як час створення замовлення.

Структурна схема роботи процесу оформлення замовлення зображена в додатку Е.

Коли дані замовлення сформовано, вони зберігаються у базі даних за допомогою методу setDoc. Запис додається до колекції orders з використанням щойно згенерованого ідентифікатора. Це гарантує, що замовлення буде доступне для подальшої обробки чи перегляду.

Після успішного збереження функція перенаправляє користувача до сторінки підтвердження, передаючи туди ідентифікатор замовлення. Наприкінці форми очищуються.

У разі виникнення помилки під час виконання будь-якого з кроків, блок catch перехоплює винятки та відображає відповідне повідомлення в консолі, допомагаючи діагностувати проблему. Незалежно від успішності виконання, у блоці finally індикатор завантаження вимикається, сигналізуючи про завершення операції.

Інтерфейс компонента побудований таким чином, щоб відображати кожен етап процесу у зручному для користувача вигляді. Прогресивна шкала візуалізує поточний стан, а кнопки для переходу між етапами дозволяють легко рухатися вперед або повертатися назад. Завдяки такій структурі, компонент інтегрує серверну логіку із користувацьким інтерфейсом, створюючи цілісну систему для зручного оформлення замовлень.

Повний код компоненти Checkout зображено в додатку Ж, а результат відображення результату рендеру компоненти зображено на рисунку 3.6.

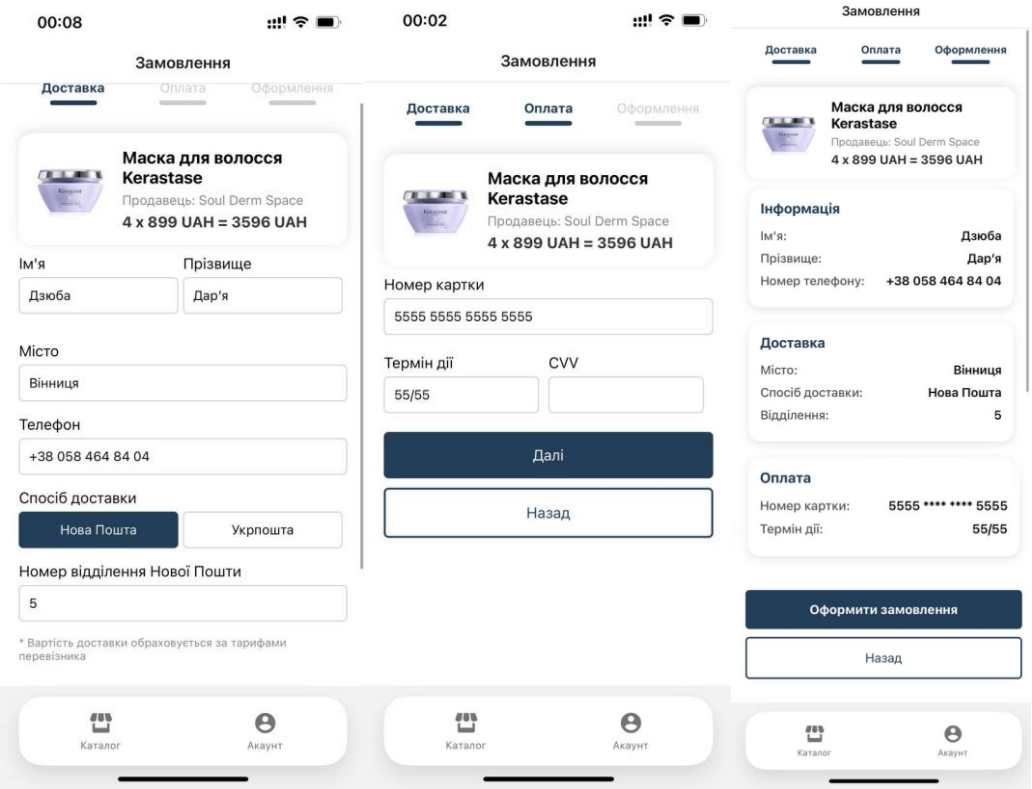

Рисунок 3.6 — Результат відображення компоненти Checkout

3.6 Розробка функціоналу із впровадження технології прямого посилання

Технологія deep linking у мобільних додатках є технікою, яка дозволяє напряму відкривати певні сторінки або функціонал додатку за допомогою спеціально створених URL-адрес. Цей підхід забезпечує користувачів зручним доступом до конкретних сторінок, обходячи необхідність навігації через головний екран або інші проміжні етапи. Основна ідея deep linking полягає у використанні унікальних посилань, які додаток розпізнає та обробляє відповідно до заданих параметрів [19].

Прямі посилання є особливо корисними для взаємодії між мобільними додатками та веб-середовищем, а також у маркетингових кампаніях, які спрямовують користувачів безпосередньо до бажаного контенту. Наприклад, посилання може привести користувача до конкретного продукту в інтернет-магазині, сторінки оформлення замовлення чи перегляду потрібної інформації. Це значно покращує користувацький досвід, скорочуючи шлях до необхідної інформації.

Deep linking працює завдяки інтеграції зі схемами URI, які задаються в конфігурації мобільного додатку. Ці схеми дозволяють операційній системі пристрою ідентифікувати додаток як той, що має обробити посилання. Наприклад, у конфігураційному файлі може бути задана схема myapp, яка слугує основою для всіх deep link посилань. Коли користувач натискає на посилання з цією схемою, пристрій запускає відповідний додаток і передає йому параметри, витягнуті з URL.

Крім базового deep linking, існує також універсальні посилання (universal linking) та відкладені посилання (deferred deep linking). Універсальне посилання (universal link) — це тип посилань, які працюють як у веб-браузері, так і безпосередньо в мобільному додатку. Основною метою універсальних посилань є забезпечення плавного переходу між веб- і мобільною версією сервісу. Якщо на пристрої користувача встановлений додаток, універсальне посилання автоматично відкриє потрібну сторінку всередині додатку. Якщо ж додаток відсутній, посилання відкриється у веб-браузері, надаючи доступ до тієї ж інформації через веб-інтерфейс [20].

Універсальні посилання працюють через систему асоціацій доменів, де сервер веб-сайту підтверджує, що додаток належить до певного домену. Це забезпечує безпеку й запобігає зловживанням. Вони особливо корисні для бізнесів, які мають як веб-версію, так і мобільний додаток, оскільки дозволяють користувачам взаємодіяти з платформою незалежно від їхніх уподобань або обставин.

Відкладене посилання (deferred deep link), у свою чергу, спрямоване на вирішення задачі перенаправлення користувачів до потрібного контенту після встановлення додатку. Це посилання працює так: якщо додаток уже встановлено, користувач одразу потрапляє на потрібну сторінку в додатку. Якщо ж додаток не встановлений, користувача перенаправляють до магазину додатків для його завантаження. Після встановлення додаток відкривається на тій самій сторінці, яка була зазначена у відкладеному посиланні [21].

Було вирішено використати технологію deep linking в цілях автоматизації системи оформлення періодичних покупок через те, що всередині посилання можна зберегти ідентифікатор сторінки для автоматичного відкриття додатку та передати специфічні параметри. Задля досягнення цілей додатку, що розробляється, потрібно створювати посилання на сторінку повторного оформлення замовлення разом із ідентифікатором покупки, яка повинно бути повторена.

Для того, щоб забезпечити зручність повторного використання даного посилання, можна використати шифрування посилання в QR-код. Завдяки тому, що користувач може роздрукувати QR-код та помістити його поруч із товарами, які він періодично замовляє, це сприятиме автоматизації процесу повторного оформлення замовлення, адже камера на телефоні завжди в швидкому доступі.

Щоб користувач відразу ознайомився із функціоналом автоматизованого процесу повторного оформлення замовлень, після успішного створення покупки система перенаправляє його на сторінку із деталями про доставку, оплату, продукт та сформованим QR-кодом, який доступний для збереження.

З технічної точки зору, при завантаженні сторінки здійснюється запит до бази даних для отримання інформації про вибране замовлення, а також пов’язані дані про доставку та продукт. Перш ніж виконати запит, перевіряється, чи користувач авторизований, і чи має він доступ до замовлення. У разі невідповідності доступу користувача або помилок у запиті, відображаються відповідні повідомлення.

Після успішного завантаження даних інформація про замовлення відображається на екрані у вигляді картки з деталями доставки, продукту та загальної суми. Для зручності користувача створюється динамічний QR-код, який містить посилання на сторінку повторного замовлення. Це дозволяє автоматизувати процес, зменшуючи необхідність введення даних вручну. QR-код можна зберегти у галереї пристрою для подальшого використання.

Генерація QR-коду реалізована за допомогою бібліотеки react-native-qrcode-svg. Ця бібліотека дозволяє динамічно створювати QR-коди з високою якістю

зображення, використовуючи текстові дані або URL як вхідні параметри. Також у компоненті реалізовано функцію, яка захоплює зображення QR-коду у вигляді знімка, а потім зберігає його у медіатеку. Для створення знімків QR-коду разом із текстовими даними використовується бібліотека react-native-view-shot. Вона дозволяє захоплювати зображення будь-якого елемента інтерфейсу у вигляді файлу зображення. А для збереження QR-коду в галерею пристрою використовується бібліотека expo-media-library. Вона надає доступ до медіатеки пристрою, дозволяючи зберігати зображення та інші файли.

Повний код компоненти OrderDetails зображено в додатку Ж, а результат відображення результату рендеру компоненти зображено на рисунку 3.7.

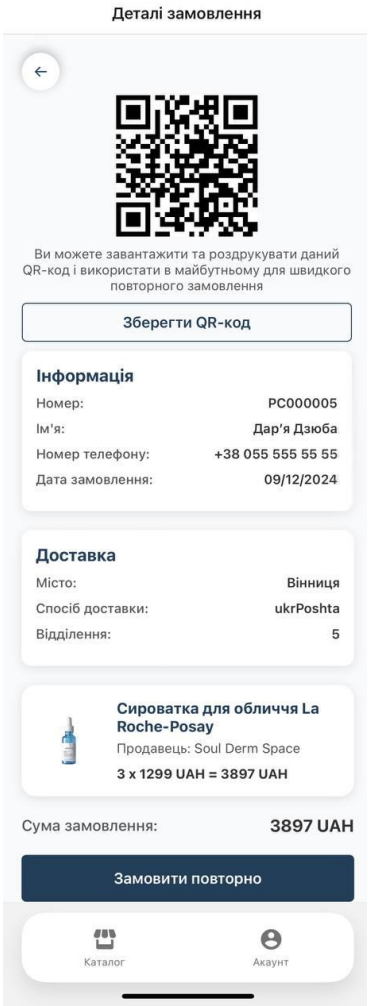

Рисунок 3.7 — Результат відображення компоненти Checkout

4 ВЕРИФІКАЦІЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Розділ має на меті дослідити та оцінити ефективність автоматизації системи періодичних транзакцій. Також в ньому буде проведено аналіз виконаної оптимізації.

4.1 Верифікація коректного виконання модулю аутентифікації

Після реєстрації користувача, його дані про логін, пароль та ім’я повинні зберігатись до системи Firebase. На рисунку 4.1 зображено сторінку форми реєстрації із заповненими даними про нового користувача.

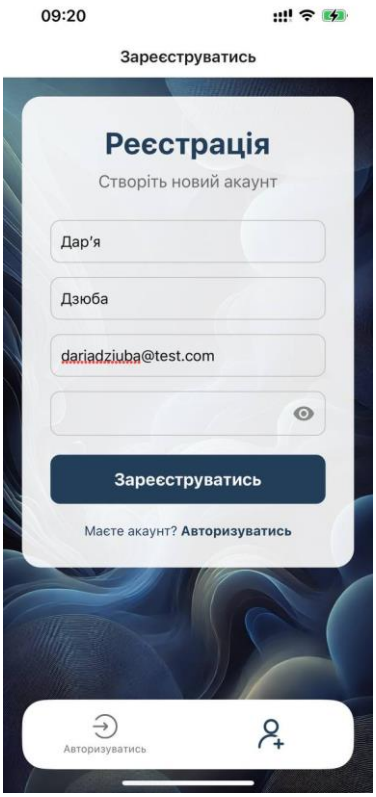

Рисунок 4.1 — Заповнена форма реєстрації

Після натиснення на кнопку «Зареєструватись» очікуваним результатом є те, що запис із новим користувачем буде додано всередині модуля Authorization від Firebase. На рисунку 4.2 можна зауважити, що функціонал відпрацював коректно і новий запис із даними, що було введено, було створено.

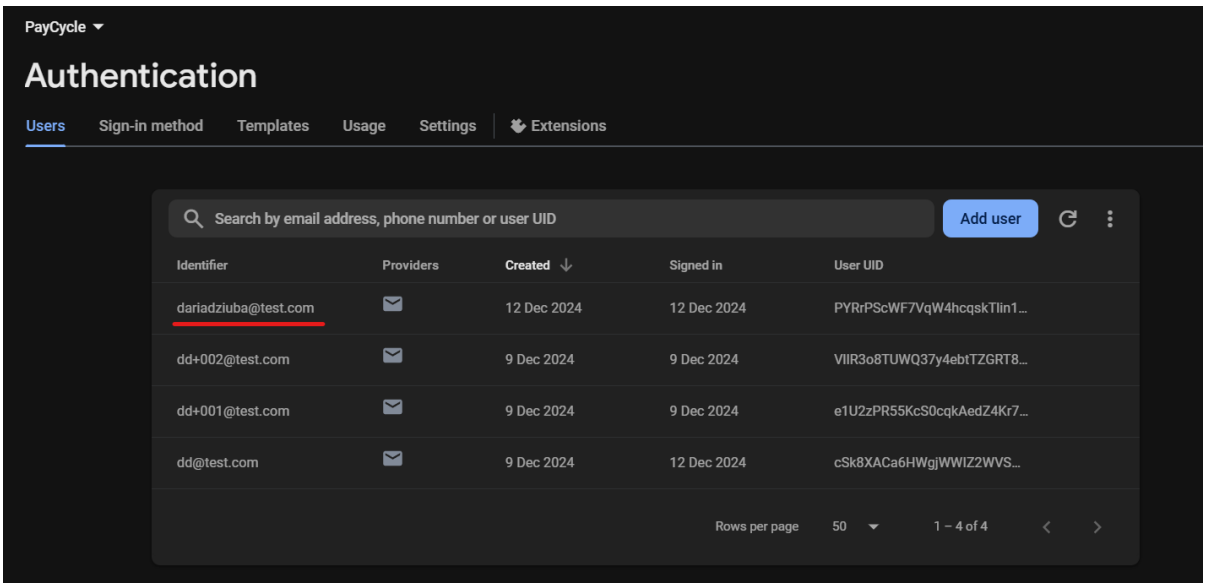

Рисунок 4.2 — Додавання нового запису до модуля Authorization від Firebase

Окрім цього, потрібно перевірити чи має змогу користувач авторизуватись із попередньо введеними даними в систему. На рисунку 4.3 зображена заповнена форма авторизації та вигляд акаунта після натиснення на кнопку «Авторизуватись».

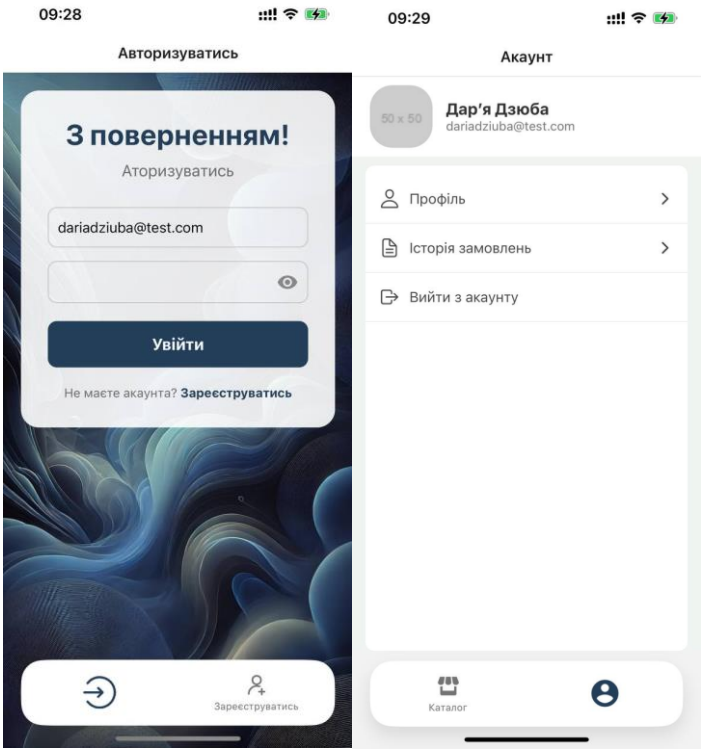

Рисунок 4.3 — Форма авторизації та сторінку результату дій після логіну

4.2 Верифікація створення замовлення на стороні Firebase

Після того як користувач обрав продукт для оформлення замовлення, він переходить до наступних етапів, які включають заповнення даних у форми доставки та оплати. На цьому етапі користувач вводить своє ім’я, контактні дані, адресу доставки, а також обирає спосіб доставки, наприклад, через Нову Пошту чи Укрпошту. Одночасно заповнюються платіжні дані, які необхідні для підтвердження оплати. Після того як усі обов’язкові поля заповнено, користувач натискає кнопку "Сформувати замовлення" (рисунок 4.4), що є фінальним кроком процесу оформлення.

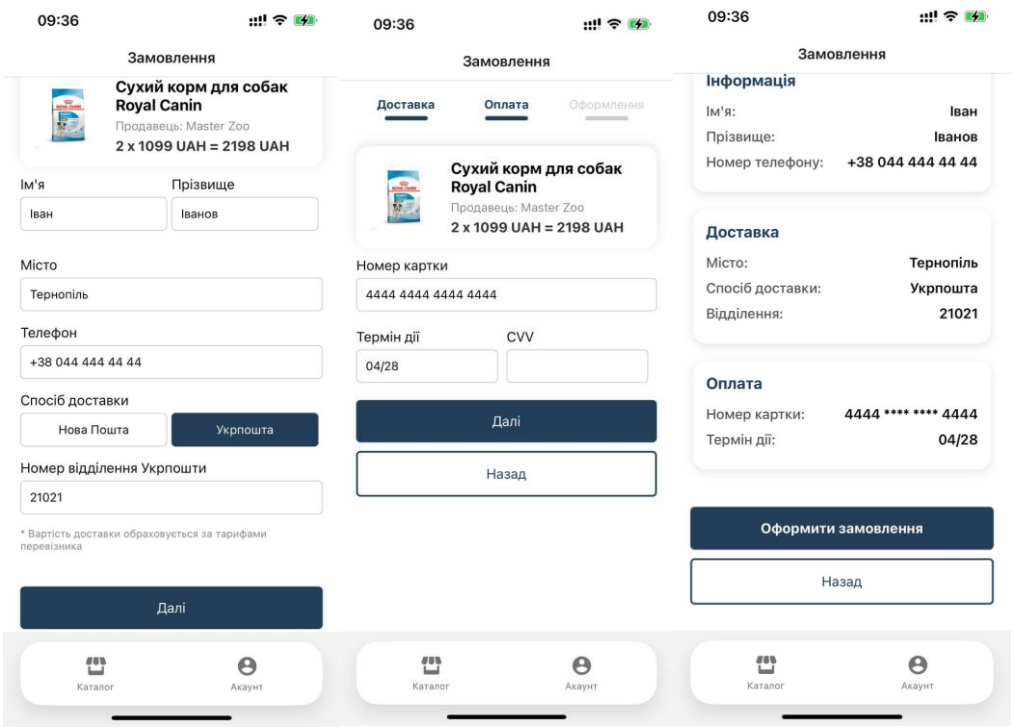

Рисунок 4.4 — Процес заповнення форм задля оформлення замовлення

На даному етапі система повинна автоматично створити запис у базі даних Firebase Cloud. Цей запис повинен включати всі ключові дані замовлення: ідентифікатор користувача, деталі обраного продукту (назву, кількість, ціну), загальну суму замовлення, адресу доставки, спосіб доставки, а також платіжну інформацію. Результат запису даних про замовлення в документ Orders в модулі Cloud Firestore зображено на рисунку 4.5.

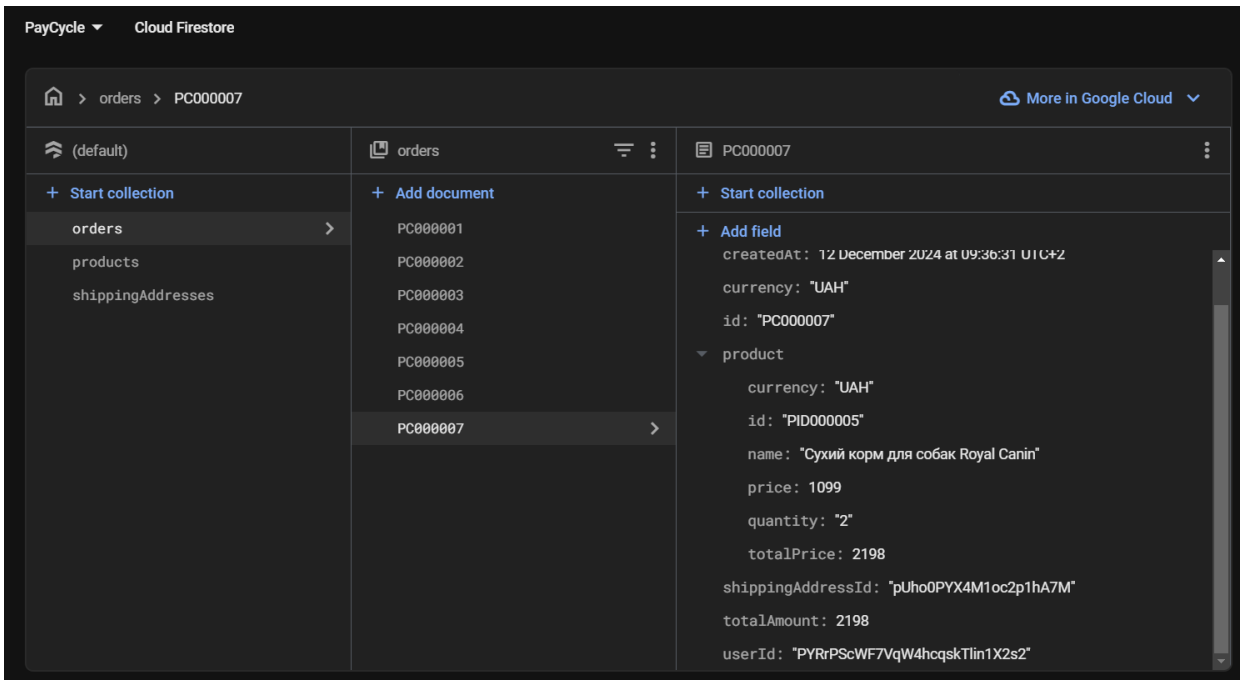

Рисунок 4.5 — Вигляд створеного запису замовлення

4.3 Верифікація функціоналу повторного замовлення

Після створення замовлення сторінка облікового запису повинна мати розділ, який зберігає історію попередніх покупок. У цьому розділі має відображатися список усіх замовлень користувача із зазначенням ключових даних, таких як дата оформлення, деталі продуктів, сума замовлення та статус його виконання. Це дозволить забезпечити користувачам зручний доступ до інформації про їхні покупки.

Кожне замовлення в історії повинно мати функціонал повторного оформлення. Така функція повинна надавати можливість повторного використання даних про доставку та оплату, збережених у попередньому замовленні. Повторне замовлення має ініціюватися або через прямий вхід у додаток, де користувач вручну вибирає потрібне замовлення, або через автоматичне сканування попередньо збереженого QR-коду (рисунок 4.6).

QR-код повинен мати всі необхідні дані для швидкого відновлення замовлення. Цей код має бути доступний для збереження у галереї пристрою або для друку в офлайн форматі. Сканування такого коду повинно автоматично

відкривати сторінку повторного замовлення з усіма попередньо заповненими даними, що значно спрощує процес оформлення.

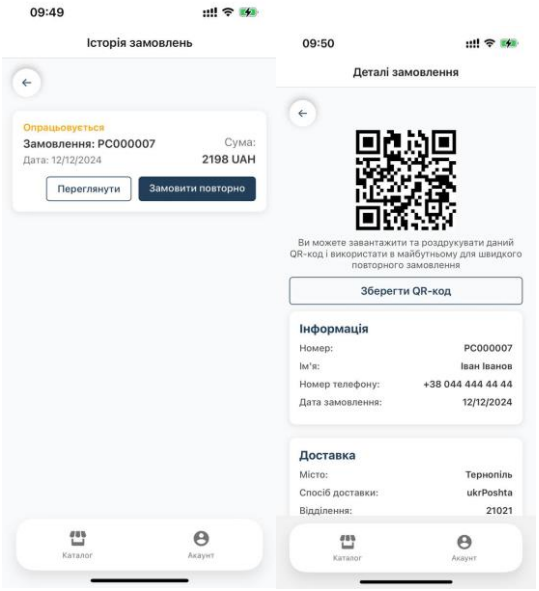

Рисунок 4.6 — Сторінки із можливістю розпочати процес повторного замовлення

Результат відображення додатку після натиснення на кнопку «Замовити повторно» та після сканування збереженого QR-коду зображено на рисунку 4.7.

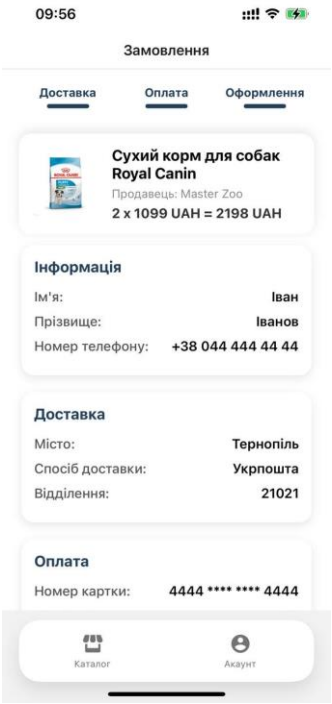

Рисунок 4.7 — Сторінка результату виконання повторно процесу замовлення

4.4 Верифікація кросплатформеності додатку

Оскільки перевагою інстурменту Expo є те, що він надає можливість створювати кросплатформенні додатки, тобто ті, які підтримуються різними платформами, то необхідно провести верифікацію коректності відображення одних і тих самих сторінок на системі Android та iOS.

На рисунку 4.8 зліва зображено як відображуються сторінки логіну та реєстрації на системі Android, справа — на системі iOS.

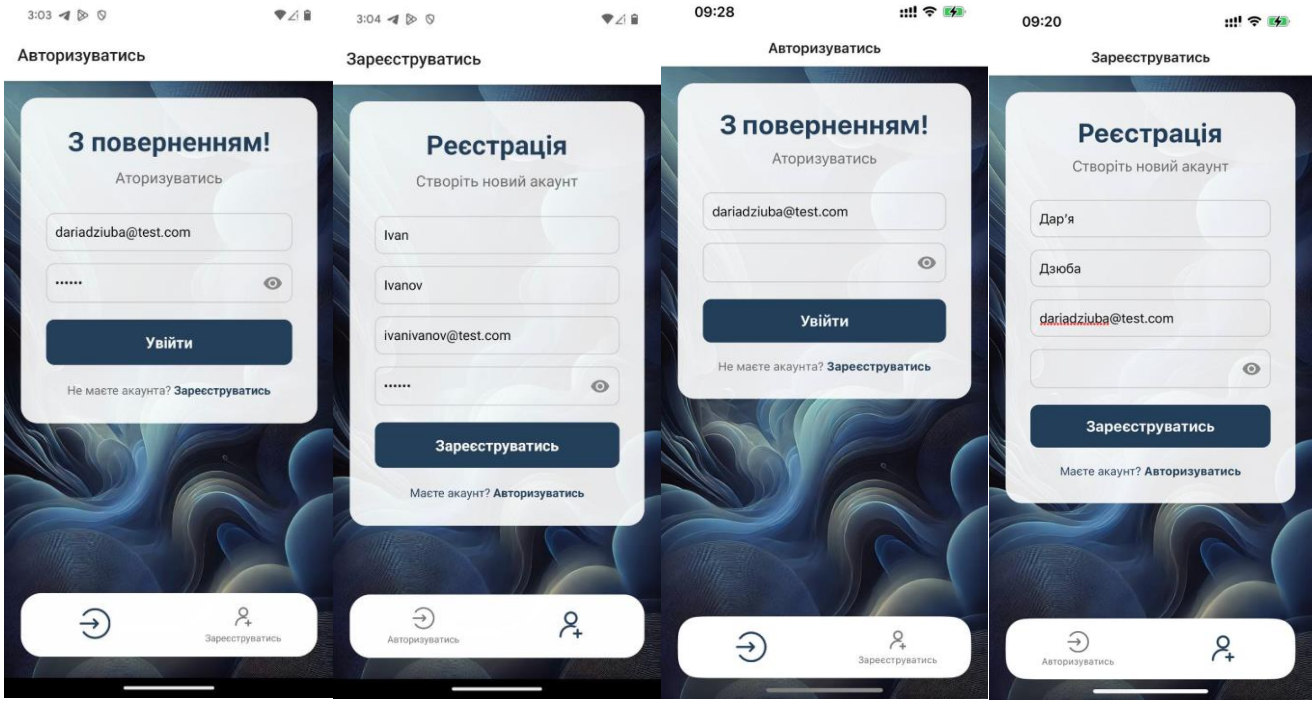

Рисунок 4.8 — Порівняння відображення сторінок для авторизації на ОС Android та iOS

Для наочного порівняння вигляду сторінок каталогу товарів та їх детального опису на різних платформах, рисунок 4.9 демонструє, як ці сторінки відображаються на Android та iOS. Ліва частина зображення ілюструє інтерфейс додатку на платформі Android, включаючи специфічні особливості стилю й компонування, характерні для цієї системи. Права частина показує аналогічні сторінки на iOS, де помітні відмінності у візуальних елементах, таких як шрифти, відступи та стиль компонентів, що відповідають дизайну системи.

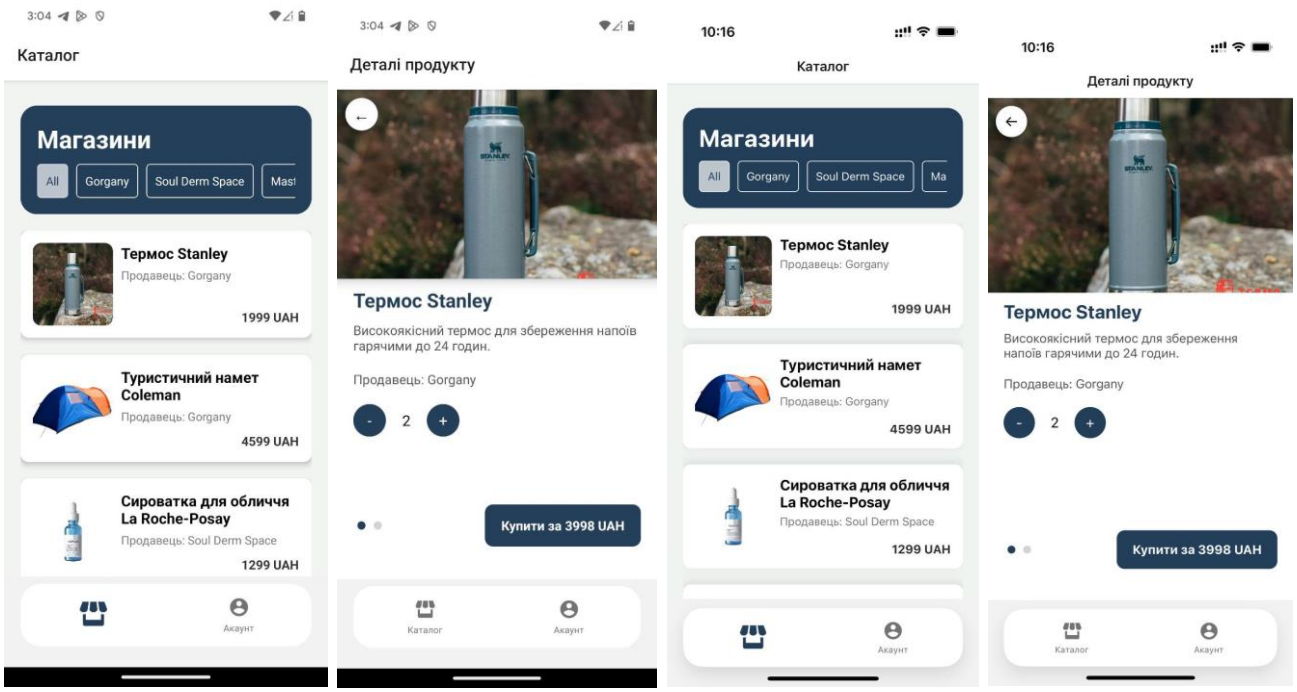

Рисунок 4.9 — Порівняльний огляд сторінок каталогу та деталей про продукт на різних мобільних ОС

Під час детального порівняння двох компонентів системи стає очевидним, що їхні основні функціональні можливості та принципи роботи майже ідентичні. Вони реалізують схожі задачі та забезпечують аналогічний результат для користувача, використовуючи подібні механізми обробки даних і взаємодії з інтерфейсом. Хоча невеликі відмінності у візуальному оформленні чи деталях реалізації присутні, вони не мають суттєвого впливу на загальну функціональність системи.

Таким чином у висновку верифікації кросплатформеності додатку стало наочно зрозуміло, що за допомогою інструментів Expo вдалося розробити максимально схожий додаток для обидвох платформ.

4.5 Аналіз оптимізації оформлення замовлення

Оскільки метою розробки даного додатку було доведення доцільності методу автоматизації процесу оформлення замовлення, то вартує провести аналіз на швидкість проходження кроків оформлення замовлення.

Стандартний шлях оформлення покупки в додатку займає в середньому 55 секунд. Це включає вибір товару, введення даних для доставки та оплати, а також підтвердження замовлення. Хоча цей час відповідає загальноприйнятим стандартам, він може бути значним у порівнянні з альтернативними сценаріями, особливо для користувачів, які регулярно оформлюють однакові замовлення.

Функція повторного замовлення через історію замовлень дозволяє скоротити час до 33 секунд. Це досягається за рахунок автоматичного завантаження раніше збережених даних про доставку та оплату, що зменшує кількість дій, необхідних для завершення процесу. Користувачу залишається лише вибрати потрібне замовлення з історії та підтвердити його, що значно спрощує оформлення для постійних клієнтів.

Ще більшу оптимізацію забезпечує функція повторного замовлення через QR-код. Вона скорочує час до 21 секунди завдяки тому, що всі дані, необхідні для оформлення, автоматично зчитуються після сканування коду. У цьому сценарії користувач повністю уникає етапів вибору замовлення та введення даних, отримуючи готову форму для підтвердження.

Різниця у часі між стандартним оформленням замовлення і повторним замовленням через історію складає 22 секунди, що становить скорочення приблизно на 40%. Це підтверджує, що навіть базова функція повторного замовлення є ефективним інструментом для зменшення навантаження на користувачів і спрощення рутинних дій.

Однак найбільшу ефективність демонструє використання QR-кодів. Скорочення часу на 34 секунди порівняно зі стандартним оформленням замовлення дозволяє знизити тривалість процесу майже на 62%. Це робить QR-коди не лише зручним, а й найбільш оптимізованим способом взаємодії з додатком, особливо для повторюваних покупок.

Економія часу на кожному етапі процесу оформлення замовлення сприяє зниженню відсотка відмов від покупки. Коротший шлях до завершення покупки зменшує ризик, що користувач залишить процес оформлення, не завершивши його.

Таким чином, оптимізація шляху оформлення замовлення через впровадження функцій повторного замовлення та використання QR-кодів дозволяє не тільки скоротити час на виконання задач, а й покращити загальний користувацький досвід використання додатку. Це підтверджує доцільність інвестицій у розвиток таких функцій для підвищення конкурентоспроможності додатку.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку системи автоматизації періодичних транзакцій для отримання послуг. Суть функціоналу полягає в тому, що користувач робить замовлення певної продукції і разом із нею отримує QR-код. Особливістю розробки є удосконалення методу підписки на товари та послуги за рахунок покращення системи підписок за допомогою QR кодів, які при скануванні автоматично оформлюють повторне замовлення.

Аналогами розробки можуть бути: Сільпо — приблизно 150000 гривень; Розетка — 400000 гривень; Пром — 120000 гривень; OLX — 200000; Shafa — 80000 гривень.

Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1

Таблиця 5.1 — Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження таблиці 5.1

3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 5.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3 -х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу розробки зведено в таблицю 5.2.

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	4	4	3
Наявність аналогів на ринку	3	3	3
Цінова політика	4	4	4
Технічні та споживчі властивості виробу	3	3	4
Експлуатаційні витрати	4	3	4
Ринок збуту	3	4	4
Конкурентоспроможність	4	3	3
Фахівці з технічної і комерційної реалізації	3	4	4
Фінансування	4	3	4
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	4
Супровідна документація	4	4	4
Сума	44	42	43
Середньоарифметична сума балів	(44+42+43) / 3 = 43		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0—10	Низький
11—20	Нижче середнього
21—30	Середній
31—40	Вище середнього
41—48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок покращення системи підписок за допомогою QR кодів, які при скануванні автоматично оформлюють повторне замовлення.

5.2 Прогнозування витрат на виконання науково-дослідної (дослідно-конструкторської) роботи

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$Т_р$ — число робочих днів за місяць, 21 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведено до таблиці 4.4.

Таблиця 4.4 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	38000	1809,52	33	59714,286
Програміст	35000	1666,67	33	55000,000
Всього				114714,29

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Розрахунок додаткової заробітної плати розробників, які брати участь в розробці обладнання/програмного продукту.

Додаткову заробітну плату прийнято розраховувати як 15 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 15\ \% / 100\ \% \quad (5.2)$$

$$З_д = (114714{,}29 \cdot 15\ \% / 100\ \%) = 17207{,}14 \text{ (грн.)}$$

5.2.3 Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \quad (5.3)$$

$$Н_з = (114714{,}29 + 17207{,}14) \cdot 22\ \% / 100\ \% = 29022{,}71 \text{ (грн.)}$$

5.2.4 Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$А=\frac{Ц}{Тв}\cdot\frac{t_{вик}}{12}\ [грн.]. \quad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років;

$t_{вик}$ — термін використання під час розробки, місяців.

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 58999 грн., термін його корисного використання згідно податкового законодавства — 2 роки, а термін його фактичного використання — 1,57 міс.

$$А_{обл}=\frac{58999}{2}\times\frac{1,57}{12}=3863,03\ грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.5.

Таблиця 5.5 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	58999	2	1,57	3863,030
Офісне обладнання (меблі)	33000	4	1,57	1080,357
Приміщення	2000000	20	1,57	13095,238
Всього				18038,63

Так як вартість ліцензійної ОС та спеціалізованих ліцензійних

нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, а також вартість використання Microsoft Windows 10 м — 3885 грн, Firebase та Firestore — 25 доларів/міс : 50*40*1,57 = 3140 грн., $В_{нем.ак.}$ = 7025 грн.

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,35 кВт;

Ф — фактична кількість годин роботи обладнання, годин;

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,35 \cdot 8 \cdot 33 \cdot 10,42 = 866,5272 \text{ (грн.)}$$

5.2.7 До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$І_в=(З_о+З_р)\cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$I_в = 114714,29 * 80\% / 100\% = 91771,43 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв}=(З_о+З_р)\cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 114714,29 * 150\ \% / 100\ \% = 172071 \text{ (грн.)}$$

5.2.9 Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$В_{заг} = 114714,29+17207,14+29022,71+18038,63+7025+866,53+91771,43+$$
$$+172071 = 450717,15 \text{ грн.}$$

5.2.8 Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η — коефіцієнт, який характеризує етап виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки

конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 450717,15 / 0,5 = 901434 \text{ грн}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об’єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 У випадку розробки чи суттєвого вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем, майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i=(\pm\Delta Ц_0\cdot N+Ц_0\cdot\Delta N)_i\cdot\lambda\cdot\rho\cdot(1-\frac{\vartheta}{100}), \qquad (5.9)$$

де $\pm\Delta Ц_o$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

Цб — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$;

р — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 2000 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на 100 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року — на 15000 шт., протягом другого року — на 14000 шт., протягом третього року на 13000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*100 + (2000 + 100)*15000)* 0{,}8333* 0{,}25) * (1 - 0{,}18) = 5124999{,}795 \text{ грн.}$$

$$\Delta П_2 = (0*100 + (2000 + 100)*(15000+14000)* 0{,}8333* 0{,}25) * (1 - 0{,}18) = 10403749{,}584 \text{ грн.}$$

$$\Delta П_3 = (0*100 + (2000 + 100)*(15000+14000+13000)* 0{,}8333* 0{,}25) * (1 - 0{,}18) = 15067499{,}397 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 30596248,78 грн.

5.3.2 Розраховуємо приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_1^T \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.10)$$

де $\Delta П_i$ —збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ —ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (5124999,795/(1+0,1)^1)+(10403749,584/(1+0,1)^2)+(15067499,397/(1+0,1)^3)$$
$$= 4659090,72 + 8598140,152 + 11320435,31 = 24577666,18 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$=2...5, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 901434 = 1802868,61 \text{ грн.}$$

Тоді абсолютний економічний ефект $Е_{абс}$ або чистий приведений дохід (NPV, Net Present Value) для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$Е_{абс} = ПП – PV, \quad (5.12)$$
$$Е_{абс} = 24577666,18 – 1802868,61 = 22774797,58 \text{ грн.}$$

Оскільки $E_{абс} > 0$то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності (IRR, Internal Rate of Return) вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій$E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.13}$$

де $T_ж$—життєвий цикл наукової розробки, роки.

$$E_в = 3\ \sqrt{(1 + 22774797{,}58/1802868{,}61} - 1 = 1{,}389$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0{,}14 + 0{,}05 = 0{,}19$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \quad (5.15)$$

$$T_{ок} = 1 / 1{,}389 = 0{,}72 \text{ р.}$$

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 0,72 роки, то фінансування даної наукової розробки є доцільним.

У висновку, економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 901434 гривень. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків розроблений програмний продукт за ціною дешевший за аналог і є високо конкурентоспроможним. Період окупності складе близько 0,72 роки.

ВИСНОВКИ

У даній роботі було успішно досягнуто мети та вирішено всі поставлені задачі, спрямовані на оптимізацію процесу періодичного оформлення замовлень через мобільний додаток. У першому розділі проведено аналіз сучасних методів, які реалізують функціонал періодичних транзакцій, зокрема моделі підписки. Було виявлено ключову проблематику: регулярне отримання продукції незалежно від фактичної потреби користувача, що може призводити до надмірного накопичення товарів.

У другому розділі досліджено концепції створення системи, яка б вирішувала вказану проблему. Зроблено висновок, що оптимальним підходом є надання користувачеві можливості самостійного оформлення замовлень із мінімізацією зусиль під час повторних покупок. Це дозволяє підвищити зручність і зменшити ймовірність нераціонального споживання.

Третій розділ був присвячений технічній реалізації системи у форматі мобільного додатку. У ньому детально описано архітектуру та функціонал, зокрема можливості автоматизації повторних замовлень через збереження даних у системі, а також використання QR-кодів для прискорення процесу. Реалізація була спрямована на забезпечення максимальної ефективності й зручності для користувачів.

Четвертий розділ зосереджувався на верифікації результатів, де підтверджено коректність роботи системи та її відповідність вимогам. Проведено аналіз оптимізації процесу оформлення замовлень, що продемонстрував значне скорочення часу виконання повторних покупок як через функцію історії замовлень, так і через QR-коди. Це підкреслює ефективність запропонованого рішення.

У п’ятому розділі оцінено економічну доцільність розробки. Встановлено, що реалізація мобільного додатку є доцільною як із точки зору бізнесу, так і з урахуванням користувацьких потреб, завдяки автоматизації та значному підвищенню швидкості процесів.

Апробацію результатів наукової роботи було проведено на науковій конференції : «LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025)», доповідь на тему «Методи організації безпеки даних під час автоматичних платежів» [1]. Також подано заявку на реєстрацію авторського свідоцтва комп’ютерної програми на тему «Сервіс замовлень та їх повторень із застосуванням QR-кодів» від 20.11.2024.

Під час виконання роботи було вирішено всі поставлені задачі. Проведено аналіз сучасних методів для реалізації періодичних транзакцій, досліджено та обрано оптимальні концепції автоматизації замовлень, розроблено технічну архітектуру мобільного додатку, а також проведено перевірку працездатності системи. Оцінка ефективності рішення підтвердила його позитивний вплив на користувацький досвід і доцільність з точки зору бізнесу. Робота досягла поставлених цілей і має перспективи для подальшого вдосконалення та впровадження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дзюба Д.А., Томчук М.А. «Методи організації безпеки даних під час автоматичних платежів». Матеріали конференції «LIV Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025)», Вінниця, 2024 [Електронний ресурс]. Режим доступу : https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/22829

2. Global Subscription E-Commerce Market is expected to reach US$ 478 Bn by 2025 [Електронний ресурс]. Режим доступу : https://www.prnewswire.com/news-releases/global-subscription-e-commerce-market-is-expected-to-reach-us-478-bn-by-2025cagr-68-univdatos-market-insights-301115593.html

3. Як працює модель підписки на електронну комерцію? [Електронний ресурс]. Режим доступу : https://www.ranktracker.com/uk/blog/how-does-an-ecommerce-subscription-model-work/

4. Subscription-Based Apps: Pros, Cons, and How to Make the Big Bucks [Електронний ресурс]. Режим доступу : https://uplandsoftware.com/localytics/resources/blog/subscription-based-apps-pros-cons-and-how-to-make-the-big-bucks/#:~:text=One%20VisionMobile%20study%20showed%20that,in-app%20purchases%20for%20revenue

5. Що таке регулярні платежі: переваги та рекомендації для успішного впровадження [Електронний ресурс]. Режим доступу : https://www.zen.com/uk/blog/personal-finance-uk/what-are-recurring-payments-benefits-and-recommendations-for-successful-implementation/

6. Що краще моноліт чи мікросервіси? Як обрати архітектуру проєкту? [Електронний ресурс]. Режим доступу : https://iampm.club/ua/blog/shho-krashhe-monolit-chi-mikroservisi-yak-obrati-arhitekturu-projektu/

7. Jason Katzer Learning Serverless: Design, Develop, and Deploy with Confidence : навч.-метод. посіб. O'Reilly Media; 1st edition, 2020. 232 с

8. Serverless Architecture Overview [Електронний ресурс]. Режим доступу : https://www.datadoghq.com/knowledge-center/serverless-architecture/

9. Аутентифікація, авторизація та ідентифікація: як не сплутати [Електронний ресурс]. Режим доступу : https://training.qatestlab.com/blog/technical-articles/authentication-authorization-and-identification/

10. Одноразовий пароль та інші алгоритми аутентифікації: як вони використовуються в цифровій безпеці [Електронний ресурс]. Режим доступу : https://introserv.com/ua/blog/digital-security-and-one-time-password-algorithms/

11. Why Are Push Notifications So Important? [Електронний ресурс]. Режим доступу : https://rubygarage.org/blog/benefits-of-push-notifications#targetText=Push%20notifications%20can%20considerably%20enhance,updates%2C%20promotions%2C%20and%20offers

12. Let's Discuss a Push Notifications Mechanism for iOS and Android [Електронний ресурс]. Режим доступу : https://agilie.com/blog/lets-discuss-a-push-notifications-mechanism-for-ios-and-android

13. Що таке QR-код та як його створити [Електронний ресурс]. Режим доступу : https://apix-drive.com/ua/blog/useful/schto-take-qr-kod

14. Створення та використання QR-коду для платежів у бізнесі [Електронний ресурс]. Режим доступу : https://tranzzo.com/uk-ua/blog/stvorennja-ta-vykorystannja-qr-kodu-dlja-platezhiv-u-biznesi

15. React Native [Електронний ресурс]. Режим доступу : https://reactnative.dev/

16. Expo Go [Електронний ресурс]. Режим доступу : https://expo.dev/go

17. Visual Studio Code [Електронний ресурс]. Режим доступу : https://code.visualstudio.com/

18. Firebase Overview [Електронний ресурс]. Режим доступу : https://firebase.google.com/docs

19. Deep Linking [Електронний ресурс]. Режим доступу : https://reactnavigation.org/docs/deep-linking/

20. Universal Links [Електронний ресурс]. Режим доступу : https://www.appsflyer.com/glossary/universal-links/

21. Deferred Deep Linking [Електронний ресурс]. Режим доступу : https://www.optimove.com/resources/learning-center/deferred-deep-linking

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
проф., д.т.н.. Азаров О.Д..
“29” вересня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

“Система автоматизації періодичних транзакцій для отримання послуг”

08-54.МКР.006.00.000 ПЗ

Науковий керівник: доц. каф.ОТ
_____ Томчук М.А.
Магістрантка групи 1КІ-23м
_____ Дзюба Д.А.

ВНТУ 2024

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність роботи полягає в автоматизації процесу повторного замовлення за допомогою відскановання QR-коду.

1.2 Наказ про затвердження теми МКР;

2 Мета МКР і призначення розробки

2.1 Мета роботи — дослідження та удосконалення періодичних транзакцій за допомогою функції швидкого повторного оформлення замовлень за допомогою сканування QR-кодів;

2.2 Призначення розробки — визначається необхідністю створення автоматизованої системи, яка буде пришвидшувати процес повторного замовлення товару користувачем на комерційній платформі;

3 Вихідні дані для виконання МКР

3.1 Проведення аналізу наявних методів та принципів.

3.2 Розробка алгоритму системи та мобільного застосунку.

3.4 Проведення верифікації та аналізу отриманих результатів.

3.5 Виконання розрахунків для доведення доцільності нової розробки з економічної точки зору.

4 Вимоги до виконання МКР

Головна вимога — що система повинна надавати можливість користувачеві оформити повторне замовлення після сканування специфічного QR-коду;

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1.

Таблиця А.1 — Етапи МКР

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналітичний огляд технології автоматизації періодичних транзакцій	19.09.2024	25.09.2024	Аналітичний огляд джерел, задачі досліджень, розділ 1
2	Дослідження концепцій створення системи автоматизації періодичних транзакцій	26.09.2024	11.10.2024	Розділ 2
3	Розробка системи автоматизації періодичних транзакцій	12.10.2024	1.11.2024	Розділ 3
4	Верифікація отриманих результатів	4.11.2024	8.11.2024	Розділ 4
5	Підготовка економічної частини	11.11.2024	15.11.2024	Розділ 5
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	18.11.2024	6.12.2024	ПЗ, графічний матеріал і презентація
7	Підготовка і підпис супроводжуючих документів, нормоконтроль та тест на плагіат	9.12.2024	13.12.2024	Оформленні документи

6 Матеріали, що подаються до захисту МКР

До захисту подаються: пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист

МКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання МКР

8.1 При оформлюванні МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104–2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документи на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21

ДОДАТОК Б

Лістинг компоненти Login

```
const Login = ({params}) => {
    const navigation = useNavigation();
    const { orderId } = params || {};

    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [isLoading, setIsLoading] = useState(false);
    const [errorMessage, setErrorMessage] = useState('');
    const [showPassword, setShowPassword] = useState(false);

    const handleLogin = async () => {
        setIsLoading(true);
        setErrorMessage('');
        try {
            const response = await
signInWithEmailAndPassword(FIREBASE_AUTH, email, password);
            if (orderId) {
                navigation.navigate("checkout/[orderId]", { orderId:
orderId })
            } else {
                navigation.navigate('shop');
            }
        } catch (error) {
            console.log(error);
            if (error.code === 'auth/invalid-credential') {
                setErrorMessage('Перевірте правильність введених
даних');
            } else {
                setErrorMessage('Щось пішло не так. Спробуйте ще
раз.');
            }
        } finally {
            setIsLoading(false);
        }
    };

    const handleEmailChange = (text) => {
        setEmail(text);
        setErrorMessage('');
    };

    const handlePasswordChange = (text) => {
        setPassword(text);
        setErrorMessage('');
    };
    return (
        <TouchableWithoutFeedback onPress={() =>
Keyboard.dismiss()}>
            <ImageBackground
```

```
                source={require('../assets/images/login.jpg')}
                style={styles.backgroundImage}
            >
                <TouchableWithoutFeedback onPress={() =>
Keyboard.dismiss()}>
                    <View style={styles.container}>
                        <Text style={styles.title}>З
поверненням!</Text>
                        <Text
style={styles.subtitle}>Аторизуватись</Text>

                        <View style={styles.inputContainer}>
                            <TextInput
                                style={styles.input}
                                placeholder="Електронна пошта"
                                placeholderTextColor="#888"
                                value={email}
                                onChangeText={handleEmailChange}
                                keyboardType="email-address"
                                autoCapitalize="none"
                            />
                            <View style={styles.passwordContainer}>
                                <TextInput
                                    style={styles.passwordInput}
                                    placeholder="Пароль"
                                    placeholderTextColor="#888"
                                    value={password}
                                    onChangeText={handlePasswordChan
ge}
                                    secureTextEntry={!showPassword}
                                />
                                <TouchableOpacity
                                    onPress={() =>
setShowPassword(!showPassword)}
                                    style={styles.showPasswordButton
}
                                >
                                    <Icon
                                        name={showPassword ? 'eye-
off' : 'eye'}
                                        size={24}
                                        color="#888"
                                    />
                                </TouchableOpacity>
                            </View>
                        </View>

                        {errorMessage ? <Text
style={styles.errorText}>{errorMessage}</Text> : null}
                        <TouchableOpacity
                            style={styles.loginButton}
                            onPress={handleLogin}
                            disabled={isLoading}
```

```
                        >
                            {isLoading ? (
                                <ActivityIndicator size="small"
color="#fff" />
                            ) : (
                                <Text
style={styles.loginText}>Увійти</Text>
                            )}
                        </TouchableOpacity>
                        <TouchableOpacity
style={styles.signUpContainer} onPress={() =>
navigation.navigate('register')}>
                            <Text style={styles.signUpText}>Не маєте
акаунта? <Text
style={styles.signUpLink}>Зареєструватись</Text></Text>
                        </TouchableOpacity>
                    </View>
                </TouchableWithoutFeedback>
            </ImageBackground>
        </TouchableWithoutFeedback>
    );
};
```

ДОДАТОК В

Лістинг компоненти Login

```
const Register = () => {
    const navigation = useNavigation();
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [firstName, setFirstName] = useState('');
    const [lastName, setLastName] = useState('');
    const [showPassword, setShowPassword] = useState(false);
    const [isLoading, setIsLoading] = useState(false);
    const [errorMessage, setErrorMessage] = useState('');
    const goToLogin = () => {
        navigation.navigate('index');
    };
    const handleRegister = async () => {
        setIsLoading(true);
        setErrorMessage('');
        try {
            const response = await
createUserWithEmailAndPassword(FIREBASE_AUTH, email, password);
            const user = response.user;
            await updateProfile(user, {
                displayName: `${firstName} ${lastName}`,
            });
            navigation.navigate('shop');
        } catch (error) {
            console.log("Помилка реєстрації:", error);
            if (error.code === 'auth/email-already-in-use') {
                setErrorMessage('Акаунт із такою електронною поштою
вже існує');
            } else if (error.code === 'auth/invalid-email') {
                setErrorMessage('Неправильний формат електронної
пошти');
            } else if (error.code === 'auth/weak-password') {
                setErrorMessage('Пароль має бути щонайменше 6
символів.');
            } else {
                setErrorMessage('Щось пішло не так. Спробуйте ще
раз.');
            }
        } finally {
            setIsLoading(false);
        }
    };
    return (
        <ImageBackground
            source={require('../assets/images/login.jpg')}
            style={styles.backgroundImage}
        >
            <ScrollView
contentContainerStyle={styles.scrollContainer}>
```

```
                <View style={styles.container}>
                    <Text style={styles.title}>Реєстрація</Text>
                    <Text style={styles.subtitle}>Створіть новий
акаунт</Text>
                    <View style={styles.inputContainer}>
                        <TextInput
                            style={styles.input}
                            placeholder="Ім'я"
                            onChangeText={setFirstName}
                            placeholderTextColor="#888"
                            value={firstName}/>
                        <TextInput
                            style={styles.input}
                            placeholder="Прізвище"
                            onChangeText={setLastName}
                            placeholderTextColor="#888"
                            value={lastName} />
                        <TextInput
                            style={styles.input}
                            placeholder="Електронна пошта"
                            onChangeText={setEmail}
                            placeholderTextColor="#888"
                            value={email}
                            keyboardType="email-address"
                            autoCapitalize="none"/>
                        <View style={styles.passwordContainer}>
                            <TextInput
                                style={styles.passwordInput}
                                placeholder="Пароль"
                                onChangeText={setPassword}
                                placeholderTextColor="#888"
                                value={password}
                                secureTextEntry={!showPassword}/>
                            <TouchableOpacity
                                onPress={() =>
setShowPassword(!showPassword)}
                                style={styles.showPasswordButton}
                            >
                                <Icon
                                    name={showPassword ? 'eye-off' :
'eye'}
                                    size={24}
                                    color="#888"
                                />
                            </TouchableOpacity>
                        </View>
                    </View>
                    {errorMessage ? <Text
style={styles.errorText}>{errorMessage}</Text> : null}
                    <TouchableOpacity
                        style={styles.registerButton}
                        onPress={handleRegister}
                        disabled={isLoading}>
```

```
                        {isLoading ? (
                            <ActivityIndicator size="small"
color="#fff" />
                        ) : (
                            <Text
style={styles.registerText}>Зареєструватись</Text>
                        )}
                    </TouchableOpacity>
                    <TouchableOpacity style={styles.loginContainer}
onPress={goToLogin}>
                        <Text style={styles.loginText}>Маєте акаунт?
<Text style={styles.loginLink}>Авторизуватись</Text>
                        </Text>
                    </TouchableOpacity>
                </View>
            </ScrollView>
        </ImageBackground>
    );
};
```

ДОДАТОК Г

Лістинг компоненти Shop

```
const Shop = () => {
  const [products, setProducts] = useState([]);
  const [categories, setCategories] = useState(["All"]);
  const [selectedTag, setSelectedTag] = useState("All");
  const [isLoading, setIsLoading] = useState(true);
  useEffect(() => {
    const fetchProducts = async () => {
      try {
        const querySnapshot = await getDocs(collection(FIRESTORE_DB,
"products"));
        const fetchedProducts = querySnapshot.docs.map((doc) => ({
          id: doc.id,
          ...doc.data(),
        }));
        setProducts(fetchedProducts);
        const fetchedCategories = ["All", ...new
Set(fetchedProducts.map((item) => item.seller))];
        setCategories(fetchedCategories);
      } catch (error) {
        console.error("Помилка завантаження товарів:", error);
      } finally { setIsLoading(false); }
    };
    fetchProducts();
  }, []);
const filteredData =  selectedTag === "All" ? product :
products.filter((item) => item.seller === selectedTag);

  if (isLoading) {
    return (
      <View style={styles.loaderContainer}>
        <ActivityIndicator size="large" color="#233E59" />
      </View>
    );}
  return (
    <View style={styles.container}>
      <View style={styles.header}>
        <Text style={styles.title}>Магазини</Text>
        <ScrollView
          horizontal
          showsHorizontalScrollIndicator={false}
          style={styles.tags}>
          {categories.map((category) => (
            <TagButton
              key={category}
              title={category}
              isSelected={selectedTag === category}
              onPress={() => setSelectedTag(category)} />
          ))}
        </ScrollView>
```

```
      </View>
      <ScrollView>
        {filteredData.map((item) => (<Card key={item.id} data={item}
/>))}
      </ScrollView>
    </View>
  );};
const TagButton = ({ title, isSelected, onPress }) => {
  return (
    <TouchableOpacity
      style={[
        styles.tag,
        isSelected && {
          backgroundColor: "rgba(255, 255, 255, 0.7)",
          borderColor: "rgba(255, 255, 255, 0.7)",
          borderWidth: 1,
        },
      ]}
      onPress={onPress}>
      <Text style={[styles.tagText, isSelected && { color: "#233E59"
}]}>
        {title}
      </Text>
    </TouchableOpacity>
  );};
const Card = ({ data }) => {
  const navigation = useNavigation();
  const handleDetails = () => { navigation.navigate("details", {
productId: data.id });};
  return (
    <TouchableOpacity style={styles.card} onPress={handleDetails}>
      <Image source={{ uri: data.imagePath }}
style={styles.cardImage} />
      <View style={styles.textContainer}>
        <Text style={styles.cardTitle}>{data.name}</Text>
        <Text style={styles.cardSeller}>{`Продавець:
${data.seller}`}</Text>
        <View style={styles.priceContainer}>
          <Text style={styles.cardPrice}>{`${data.price}
${data.currency}`}</Text>
        </View>
      </View>
    </TouchableOpacity>
);};
```

ДОДАТОК Д

Лістинг компоненти Details

```
const Details = () => {
    const { productId } = useLocalSearchParams();
    const [product, setProduct] = useState(null);
    const [loading, setLoading] = useState(true);
    const [currentIndex, setCurrentIndex] = useState(0);
    const [quantity, setQuantity] = useState(1);
    const screenWidth = Dimensions.get('window').width;
    const navigation = useNavigation();
    useEffect(() => {
        const fetchProduct = async () => {
            try {
                const docRef = doc(FIRESTORE_DB, "products",
productId);
                const docSnap = await getDoc(docRef);
                if (docSnap.exists()) {
                    setProduct(docSnap.data());
                    setQuantity(docSnap.data().minOrderQuantity ||
1);
                } else {
                    console.error("Продукт із заданим ID не
знайдено!");
                }
            } catch (error) {
                console.error("Помилка завантаження продукту:",
error);
            } finally {
                setLoading(false);
            }
        };
        fetchProduct();
    }, [productId]);
    const handleScroll = (event) => {
        const contentOffsetX = event.nativeEvent.contentOffset.x;
        const index = Math.floor(contentOffsetX / screenWidth);
        setCurrentIndex(index);
    };
    const handleBack = () => { navigation.navigate('shop'); };
    const incrementQuantity = () => setQuantity(quantity + 1);
    const decrementQuantity = () => {
        if (quantity > 1) setQuantity(quantity - 1);
    };
    if (loading) {
        return (
            <View style={[styles.container, { justifyContent:
'center', alignItems: 'center' }]}>
                <ActivityIndicator size="large" color="#233E59" />
            </View>
        ); }
    if (!product) {
```

```
        return (
            <View style={styles.container}>
                <Text style={{ textAlign: 'center', marginTop: 20
}}>Продукт не знайдено.</Text>
                <TouchableOpacity style={styles.backButton}
onPress={handleBack}>
                    <Text style={styles.backButtonText}>←
Назад</Text>
                </TouchableOpacity>
            </View>
        ); }
    const totalAmount = product.price * quantity;
    return (
        <View style={styles.container}>
            <View style={styles.circle}>
                <Image
                    source={{ uri: product.imagePath }}
                    style={styles.image}
                    resizeMode="cover"/>
                <TouchableOpacity onPress={handleBack}
style={styles.backButton}>
                    <Text style={styles.backButtonText}>←</Text>
                </TouchableOpacity>
            </View>
            <ScrollView
                horizontal
                pagingEnabled
                onScroll={handleScroll}
                showsHorizontalScrollIndicator={false}
                style={styles.textScrollContainer}>
                <View style={[styles.textContainer, { width:
screenWidth }]}>
                    <Text style={styles.title}>{product.name}</Text>
                    <Text
style={styles.description}>{product.description}</Text>
                    <Text style={styles.description}>Продавець:
{product.seller}</Text>
                    <View style={styles.quantityContainer}>
                        <TouchableOpacity
style={styles.quantityButton} onPress={decrementQuantity}>
                            <Text
style={styles.quantityButtonText}>-</Text>
                        </TouchableOpacity>
                        <Text
style={styles.quantityText}>{quantity}</Text>
                        <TouchableOpacity
style={styles.quantityButton} onPress={incrementQuantity}>
                            <Text
style={styles.quantityButtonText}>+</Text>
                        </TouchableOpacity>
                    </View>
                </View>
```

```
                <View style={[styles.textContainer, { width:
screenWidth }]}>
                    <Text style={styles.title}>Характеристики</Text>
                    {product.characteristics.map((char, index) => (
                        <View key={index}
style={styles.characteristicRow}>
                            <Text
style={styles.characteristicName}>{char.name}:</Text>
                            <Text
style={styles.characteristicValue}>{char.value}</Text>
                        </View>
                    ))}
                </View>
            </ScrollView>
            <View style={styles.footerContainer}>
                <View style={styles.dotsContainer}>
                    {[...Array(2).keys()].map((_, i) => (
                        <View
                            key={i}
                            style={[styles.dot, currentIndex === i
&& styles.activeDot]} />
                    ))}
                </View>
                <TouchableOpacity
                    style={styles.buyButton}
                    onPress={() => navigation.navigate('checkout', {
productId: productId, quantity: quantity })}>
                    <Text style={styles.buyButtonText}>Купити за
{totalAmount} {product.currency}</Text>
                </TouchableOpacity>
            </View>
        </View>
    );};
```

ДОДАТОК Е

Блок-схема алгоритму процесу оформлення замовлення

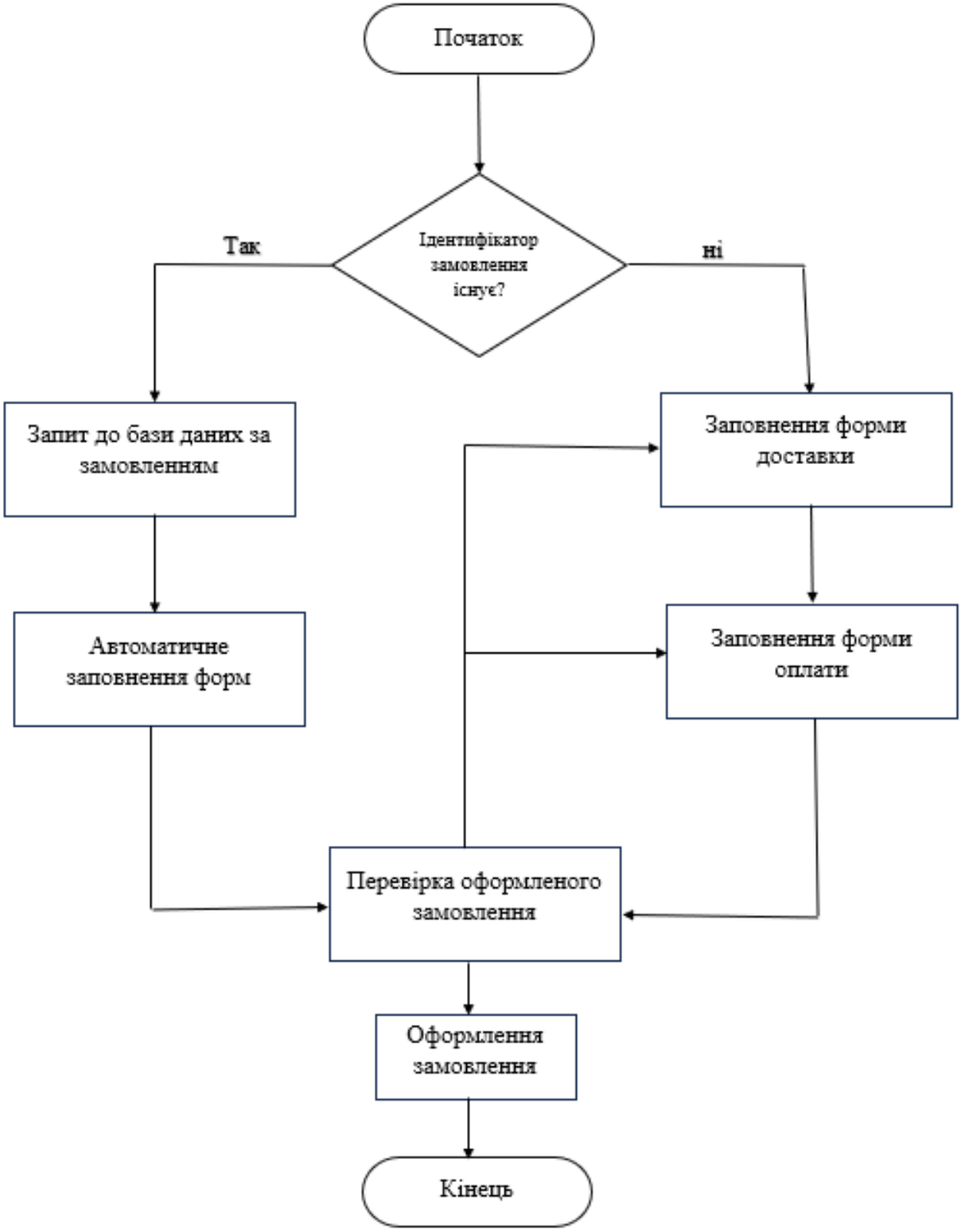

Рисунок Е.1 — Блок-схема алгоритму процесу оформлення замовлення

ДОДАТОК Ж

Лістинг компоненти Checkout

```
const Checkout = () => {
  const user = FIREBASE_AUTH.currentUser;
  const { orderId, productId, quantity } = useLocalSearchParams();
  const navigation = useNavigation();
  const [pid, setPid] = useState(productId);
  const [product, setProduct] = useState(null);
  const [qty, setQty] = useState(quantity || 1);
  const [currentStep, setCurrentStep] = useState(1);
  const [formData, setFormData] = useState({
    delivery: {
      name: "",
      surname: "",
      phone: "+38 0",
      city: "",
      deliveryOption: "novaPoshta",
      deliveryNumber: "",
    },
    payment: {
      cardNumber: "",
      expiryDate: "",
      cvv: "",
    },
  });
  const [orderLoading, setOrderLoading] = useState(false);
  const resetFormData = () => {
    setFormData({
      delivery: {
        name: "",
        surname: "",
        phone: "+38 0",
        city: "",
        deliveryOption: "novaPoshta",
        deliveryNumber: "",
      },
      payment: {
        cardNumber: "",
        expiryDate: "",
        cvv: "",
      },
    });
    setCurrentStep(1);
  };
  useEffect(() => {
    const unsubscribe = navigation.addListener("blur", () => {
resetFormData();     });
    return unsubscribe;
  }, [navigation]);
  useEffect(() => {
    const fetchProduct = async () => {
```

```
      if (!pid) return;
      const docRef = doc(FIRESTORE_DB, "products", pid);
      const docSnap = await getDoc(docRef);
      if (docSnap.exists()) {
        setProduct(docSnap.data());
      }};
fetchProduct();
  }, [pid]);
  useEffect(() => {
    const validateOrderId = async () => {
      if (!orderId || !user) return;
      try {
        const orderRef = doc(FIRESTORE_DB, "orders", orderId);
        const orderSnap = await getDoc(orderRef);
        if (orderSnap.exists()) {
          const orderData = orderSnap.data();
          if (orderData.userId !== user.uid) {
            Alert.alert("Помилка", "Це замовлення не належить
поточному користувачеві.");
            navigation.navigate("shop");
            return;
          }
          if (orderData.product) {
            const productRef = doc(FIRESTORE_DB, "products",
orderData.product.id);
            const productSnap = await getDoc(productRef);
            if (productSnap.exists()) {
              setProduct(productSnap.data());
              setPid(orderData.product.id);
              setQty(orderData.product.quantity);
            }}
          setFormData({
            delivery: {
              name: orderData.delivery?.name || "",
              surname: orderData.delivery?.surname || "",
              phone: orderData.delivery?.phone || "",
              city: orderData.delivery?.city || "",
              deliveryOption: orderData.delivery?.deliveryOption ||
"novaPoshta",
              deliveryNumber: orderData.delivery?.deliveryNumber ||
"",
            },
            payment: {
              cardNumber: orderData.cardInfo?.cardNumber || "",
              expiryDate: orderData.cardInfo?.expiryDate || "",
              cvv: orderData.cardInfo?.cvv || "",
            },
          });

          setCurrentStep(3);
        } else {
          Alert.alert("Помилка", "Замовлення не знайдено.");
          navigation.navigate("shop");
```

```
        }
      } catch (error) {
        Alert.alert("Помилка", "Не вдалося перевірити замовлення.");
        navigation.navigate("shop");
      }
    };
    validateOrderId();
  }, [orderId, user]);
  useEffect(() => {
    const fetchOrderDetails = async () => {
      if (!orderId) return;
      try {
        const orderRef = doc(FIRESTORE_DB, "orders", orderId);
        const orderSnap = await getDoc(orderRef);
        if (orderSnap.exists()) {
          const orderData = orderSnap.data();
          const shippingAddressRef = doc(
            FIRESTORE_DB,
            "shippingAddresses",
            orderData.shippingAddressId
          );
          const shippingSnap = await getDoc(shippingAddressRef);
          const shippingData = shippingSnap.exists() ?
shippingSnap.data() : {};
          const productRef = doc(FIRESTORE_DB, "products",
orderData.product.id);
          const productSnap = await getDoc(productRef);
          if (productSnap.exists()) {
            setPid(orderData.product.id);
            setQty(orderData.product.quantity);
            setProduct(productSnap.data());
          }
          setFormData({
            delivery: {
              name: shippingData.name || "",
              surname: shippingData.surname || "",
              phone: shippingData.phone || "",
              city: shippingData.city || "",
              deliveryOption: shippingData.deliveryOption ||
"novaPoshta",
              deliveryNumber: shippingData.deliveryNumber || "",
            },
            payment: {
              cardNumber: orderData.cardInfo?.cardNumber || "",
              expiryDate: orderData.cardInfo?.expiryDate || "",
              cvv: orderData.cardInfo?.cvv || "",
            },
          });
          setCurrentStep(3);
        } else {navigation.navigate("shop");}
      } catch (error) {console.error("Помилка завантаження даних
замовлення:", error);} };
    if (orderId) fetchOrderDetails();
```

```
  }, [orderId]);
  const totalAmount = product ? product.price * qty : 0;
  const isDeliveryComplete = useMemo(() => {
    const { name, surname, phone, city, deliveryOption,
deliveryNumber } =
      formData.delivery;
    return (name.trim() && surname.trim() && phone.trim() &&
city.trim() &&
      deliveryOption && deliveryNumber.trim()
    ); }, [formData.delivery]);
  const isPaymentComplete = useMemo(() => {
    const { cardNumber, expiryDate, cvv } = formData.payment;
    return cardNumber.trim() && expiryDate.trim() && cvv.trim();
  }, [formData.payment]);
  const canAccessStep = (step) => {
    if (step === 1) return true;
    if (step === 2) return isDeliveryComplete;
    if (step === 3) return isDeliveryComplete && isPaymentComplete;
    return false;
  };
  const handleStepChange = (step) => {
    if (canAccessStep(step)) {
      setCurrentStep(step);
    }
  };
  const generateOrderId = async () => {
    const ordersRef = collection(FIRESTORE_DB, "orders");
    const latestOrderQuery = query(ordersRef, orderBy("createdAt",
"desc"), limit(1));
    const latestOrderSnapshot = await getDocs(latestOrderQuery);
    const latestOrder = latestOrderSnapshot.docs[0].id;
    const latestNumber = parseInt(latestOrder.slice(2), 10) || 0;
    return `PC${String(latestNumber + 1).padStart(6, "0")}`;
  };
  const handlePlaceOrder = async () => {
    setOrderLoading(true);
    try {
      const shippingAddressRef = await addDoc(
        collection(FIRESTORE_DB, "shippingAddresses"),
        {
          ...formData.delivery,
          userId: user.uid,
          status: "Опрацьовується",
        }
      );
      const newOrderId = await generateOrderId();
      const orderData = {
        id: newOrderId,
        userId: user.uid,
        product: {
          id: pid,
          name: product.name,
          price: product.price,
```

```
          quantity: qty,
          totalPrice: totalAmount,
          currency: product.currency,
        },
        shippingAddressId: shippingAddressRef.id,
        totalAmount,
        currency: product.currency,
        createdAt: Timestamp.now(),
        cardInfo: {
          cardNumber: formData.payment.cardNumber,
          expiryDate: formData.payment.expiryDate,
          cvv: formData.payment.cvv,
        }
      };
      await setDoc(doc(FIRESTORE_DB, "orders", newOrderId),
orderData);
      navigation.navigate("myaccount/[orderId]", { orderId:
newOrderId });
      resetFormData();
    } catch (error) {console.error("Помилка оформлення замовлення:",
error);
    } finally { setOrderLoading(false); }
  }
  const handleNext = () => {
    if (currentStep === 1 && isDeliveryComplete) {
      setCurrentStep(2);
    } else if (currentStep === 2 && isPaymentComplete) {
      setCurrentStep(3);
    }};
  const handlePrevious = () => {if (currentStep > 1)
setCurrentStep(currentStep - 1);};
  const updateFormData = (step, data) => {
    setFormData((prev) => ({ ...prev, [step]: { …prev[step], ...data
},}));
  };
  return ( <CheckoutPage />)};
```

ДОДАТОК И

Лістинг компоненти OrderDetails

```
const OrderDetails = () => {
    const navigation = useNavigation();
    const { orderId } = useLocalSearchParams();
    const [order, setOrder] = useState(null);
    const [shippingAddress, setShippingAddress] = useState(null);
    const [product, setProduct] = useState(null);
    const [loading, setLoading] = useState(true);
    const viewShotRef = useRef();
    const user = FIREBASE_AUTH.currentUser;
    useEffect(() => {
        const fetchOrderData = async () => {
            try {
                if (!user) {
                    Alert.alert("Помилка", "Користувач не
залогінений.");
                    navigation.navigate("shop");
                    return;
                }
                const orderRef = doc(FIRESTORE_DB, "orders",
orderId);
                const orderSnap = await getDoc(orderRef);
                if (orderSnap.exists()) {
                    const orderData = orderSnap.data();
                    if (orderData.userId !== user.uid) {
                        Alert.alert("Доступ заборонено", "Це
замовлення належить іншому користувачеві.");
                        navigation.navigate("shop");
                        return;
                    }
                    setOrder(orderData);
                    const shippingAddressRef = doc(FIRESTORE_DB,
"shippingAddresses", orderData.shippingAddressId);
                    const shippingAddressSnap = await
getDoc(shippingAddressRef);
                    if (shippingAddressSnap.exists()) {
                        setShippingAddress(shippingAddressSnap.data(
));
                    } else {
                        Alert.alert("Помилка", "Адреса доставки не
знайдена.");
                    }
                    const productRef = doc(FIRESTORE_DB, "products",
orderData.product.id);
                    const productSnap = await getDoc(productRef);
                    if (productSnap.exists()) {
                        setProduct(productSnap.data());
                    } else {
                        Alert.alert("Помилка", "Дані продукту не
знайдені.");
```

```
                    }
                } else {
                    Alert.alert("Помилка", "Замовлення не
знайдено.");
                    navigation.navigate("shop");
                }
            } catch (error) {
                Alert.alert("Помилка", "Не вдалося завантажити
дані.");
                navigation.navigate("shop");
            } finally {setLoading(false); }
        };
        fetchOrderData();
    }, [orderId, user, navigation]);
    const saveQrCodeWithText = async () => {
        try {
            const uri = await viewShotRef.current.capture();
            const { status } = await
MediaLibrary.requestPermissionsAsync();
            if (status === "granted") {
                await MediaLibrary.saveToLibraryAsync(uri);
                Alert.alert('QR-код збережено');
            } else {
                Alert.alert("Помилка", "Дозвіл на доступ до
бібліотеки не надано.");
            }
        } catch (error) {
            Alert.alert("Помилка", "Не вдалося зберегти QR-код.");
        }
    };
    if (loading) {
        return (
            <View style={styles.loaderContainer}>
                <ActivityIndicator size="large" color="#233E59" />
            </View>
        );
    }
    if (!order || !shippingAddress || !product) return null;
    const qrCodeUrl = `${DOMAIN}/--/checkout/${orderId}`;
    const totalAmount = order.totalAmount;
    return ( <OrderdetailsView/>);
};
```

117

ДОДАТОК К

Протокол перевірки кваліфікаційної роботи

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Система автоматизації періодичних транзакцій для отримання послуг

Тип роботи: Магістерська кваліфікаційна робота

(кваліфікаційна робота, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ кафедра обчислювальної техніки

(кафедра, факультет (інститут), навчальна група)

Керівник Томчук М.А., доц. каф. ОТ

(прізвище, ініціали, посада)

Показники звіту подібності

Turnitin	
Оригінальність	92%
Загальна схожість	8%

Аналіз звіту подібності (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Дзюба Д.А.

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

_____

_____

_____

Особа, відповідальна за перевірку _____ Захарченко С.М

(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали, посада)