

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ЕКОСИСТЕМИ
ДЛЯ ШВИДКОЇ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF THE ECOSYSTEM
FOR RAPID DEVELOPMENT OF MOBILE SOFTWARE

Виконав(ла): студент(ка) 2 курсу, групи КНТ-123м
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

РАКША Я.Д.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

РАКІШІ Ярослава Дмитровича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація екосистеми для швидкої розробки мобільних застосунків. Research and Software Implementation of the Ecosystem for Rapid Development of Mobile Software.

керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “07” жовтня 2024 року № 398

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

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Опис технологій розробки. 3. Розробка екосистеми. 4. Експлуатація та тестування.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтролер	ДЕЙНЕГА Л.Ю., ст.викладач		

7. Дата видачі завдання “ 08 ” вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 3
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис)

Ярослав РАКША

(Імя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Ольга ГЛАДКОВА

(Імя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
72 с., 3 табл., 40 рис., 2 дод., 13 джерел.

ЕКОСИСТЕМА, ШВИДКА РОЗРОБКА, РОЗРОБКА МОБІЛЬНИХ
ЗАСТОСУНКІВ, СИСТЕМА ПРОГРАМНИХ РІШЕНЬ ДЛЯ ОНЛАЙН
ТОРГІВЛІ.

Об'єкт дослідження – екосистеми та системи програмних рішень для розробки мобільних застосунків.

Предмет дослідження – комп'ютерні програми та технології для розробки онлайн-магазинів.

Мета роботи – створення програмної системи для швидкої та ефективної розробки мобільних застосунків для онлайн-торгівлі за допомогою Shopify та Elasticsearch.

Матеріали, методи та технічні засоби: мова програмування JavaScript, технології Shopify, Elasticsearch, React.js, Express.js, WebStorm, MySQL.

Результати. Створено програмну систему, яка дозволяє користувачам швидко розробляти мобільні застосунки для онлайн-торгівлі, інтегруючи платформу Shopify для отримання інформації про продукти та використовуючи Elasticsearch для збереження та пошуку даних.

Висновки. Розроблено програмну систему, яка спрощує та автоматизує процес створення мобільних онлайн-магазинів, що базуються на платформі Shopify.

Галузь використання – онлайн-торгівля, електронна комерція, розробка мобільних застосунків для бізнесу.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 72 pages, 2 table, 40 figures, 2 appendixes, 13 sources.

ECOSYSTEM, RAPID DEVELOPMENT, MOBILE APPLICATION DEVELOPMENT, SYSTEM OF SOFTWARE SOLUTIONS FOR ONLINE COMMERCE.

Object of study – ecosystems and systems of software solutions for mobile application development.

Subject of study – computer programs and technologies for developing online stores.

The purpose of the work is to create a software system for the fast and efficient development of mobile applications for online commerce using Shopify and Elasticsearch. Materials, methods, and tools: JavaScript programming language, Shopify technology, Elasticsearch, React.js, Express.js, WebStorm, MySQL.

Results: A software system has been developed that allows users to quickly create mobile applications for online commerce, integrating the Shopify platform to retrieve product information and using Elasticsearch for data storage and search.

Conclusions: A software system has been developed that simplifies and automates the process of creating mobile online stores based on the Shopify platform.

The field of use is online commerce, e-commerce, and mobile application development for businesses.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Актуальність проблеми	11
1.2 Огляд існуючих рішень	11
1.2.1 Tarcart.....	12
1.2.2 Shopney.....	13
1.2.3 AppMysite.....	15
1.3 Дослідження методологій швидкої розробки	18
1.4 Висновок за розділом 1.....	19
2 Опис технологій розробки.....	21
2.1 Технологічний стек	21
2.1.1 Express.js	21
2.1.2 MySQL та Redis	22
2.1.3 Next.js та React.js.....	22
2.1.4 React Native	23
2.1.5 Typescript.....	23
2.2. Допоміжні технології для розробки	24
2.2.1 Expo	25
2.2.2 Postman	25
2.2.3 Table Plus.....	26
2.2.4 Redis Insight.....	26
2.2.5 Shopify	27
2.3 Висновок за розділом 2.....	27
3 Розробка екосистеми.....	28
3.1 Розробка серверної частини	28
3.1.1 Структура серверної частини	28
3.1.2 Схема серверної частини.....	29

3.1.3 Маршрутизація на сервері.....	30
3.1.4 Функціонал серверної частини	33
3.2 Розробка клієнтської частини для браузеру	37
3.2.1 Структура клієнтської частини для браузеру.....	37
3.2.2 Допоміжні бібліотеки для клієнтської частини браузеру.....	39
3.2.3 Структура клієнтської частини для мобільного застосунку	40
3.3 Опис бази даних	42
3.3.1 Даталогічна модель.....	45
3.4 Опис роботи кешування даних	46
3.4.1 Кешування налаштувань для магазину.....	46
3.5 Архітектура системи.....	49
3.6 Висновок за розділом 3.....	51
4 Експлуатація та тестування.....	52
4.1. Функціонал клієнтської частини для браузеру	52
4.2 Функціонал клієнтської частини для мобільного застосунку	57
4.3 Висновок за розділом 4.....	61
Висновки	62
Перелік джерел посилання	63
Додаток А Текст програми.....	64
Додаток Б Слайди презентації	67

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

2FA – Two-Factor Authentication;

HTTP – Hypertext Transfer Protocol;

OTP – One Time Password;

QR – Quick Response;

UUID – Universally unique identifier;

ПЗ – програмне забезпечення.

ВСТУП

У сучасному світі технологічний прогрес і стрімкий розвиток цифрової економіки створюють умови для зростаючого попиту на інноваційні рішення в галузі розробки програмного забезпечення. Особливої актуальності набувають технології, орієнтовані на швидке створення продуктів, що дозволяють компаніям адаптуватися до динамічних змін ринку та залишатися конкурентоспроможними.

Мобільні пристрої сьогодні відіграють ключову роль у повсякденному житті, забезпечуючи доступ до цифрових сервісів будь-де й будь-коли. У зв'язку з цим мобільна розробка стає стратегічно важливою для бізнесу. В умовах зростаючої популярності смартфонів підприємства стикаються з необхідністю створювати продукти, які швидко адаптуються до потреб користувачів і відповідають високим вимогам щодо зручності та функціональності. Проекти, орієнтовані на швидку розробку мобільних додатків, пропонують значні переваги. Вони дають змогу скоротити час і витрати на створення цифрових продуктів, а також швидко реагувати на ринкові можливості. Це особливо важливо для сектору електронної комерції, де оперативність і мобільність є вирішальними факторами успіху.

Інструменти для прискореної розробки, інтегровані з сучасними платформами, такими як Shopify, дозволяють створювати функціональні та привабливі рішення, забезпечуючи безперебійну синхронізацію даних і підвищуючи ефективність бізнес-процесів.

Підсумовуючи, розвиток швидких технологій розробки, орієнтованих на мобільну сферу, відкриває нові можливості для бізнесу. Використання таких підходів забезпечує не лише економію ресурсів, але й створення сучасних цифрових продуктів, які відповідають потребам користувачів у зручності, швидкості та функціональності.

Таким чином, завданнями дипломної кваліфікаційної роботи є:

- провести аналіз існуючих підходів до розробки мобільних додатків з повною кастомізацією дизайну та функціоналу;
- визначити ключові технології та фреймворки, що оптимізують розробку;
- оцінити ефективність використання API для інтеграції з Shopify та іншими системами на прикладі конкретних запитів;
- реалізувати компоненти серверної частини, які підтримують високу швидкість обробки запитів та інтеграцію з базами даних;
- провести тестування на різних етапах розробки, зокрема тестування ефективності обробки запитів через маршрути;
- реалізувати клієнтську частину;
- виконати тестування розробленої екосистеми.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Збереження часу та підвищення ефективності процесів розробки є одними з найважливіших завдань сучасної індустрії програмного забезпечення. У світі, де технології швидко змінюються, компанії мають діяти оперативно, аби вчасно реагувати на запити споживачів і не втрачати конкурентну перевагу. Особливо це стосується мобільної сфери, яка стала ключовою платформою для надання послуг і взаємодії зі споживачами.

Технології швидкої розробки, такі як інтеграція готових рішень з платформами електронної комерції, дозволяють значно скоротити час на розробку продуктів і забезпечити їхню високу якість. Наприклад, використання хмарних платформ, автоматизація процесів тестування та готові конструктори додатків забезпечують суттєву оптимізацію витрат і спрощують реалізацію навіть складних проектів. Для мобільної сфери швидка розробка є особливо актуальною, оскільки мобільні пристрої стали не лише інструментом спілкування, але й основним засобом для здійснення покупок, споживання контенту та взаємодії з цифровими сервісами.

Створення мобільних додатків, орієнтованих на швидкий вихід на ринок, дозволяє компаніям швидко адаптуватися до змін у поведінці користувачів, покращувати клієнтський досвід і розширювати аудиторію.

Таким чином, впровадження сучасних підходів до швидкої розробки, особливо у мобільній сфері, є важливим кроком для бізнесу, який прагне забезпечити собі стійкий розвиток і успіх в умовах цифрової трансформації.

1.2 Огляд існуючих рішень

На даний час існує декілька платформ для швидкої розробки мобільного застосунку. Нижче проведений аналіз з виділенням основних переваг та недоліків цих сервісів.

1.2.1 Tapcart

Tapcart є платформою, яка забезпечує створення мобільних додатків для магазинів на Shopify без необхідності програмування. Використовуючи інтерфейс із функцією перетягування, користувачі можуть налаштовувати макет додатку, що відповідає потребам їхнього бізнесу, а також редагувати шаблони сторінок, що містять інформацію про продукти. Цей інструмент дозволяє безперешкодно інтегрувати дизайн інтернет-магазину в мобільний додаток, спрощуючи процес його розробки та налаштування [1].

Головна сторінка Tapcart зображена на рисунку 1.1.

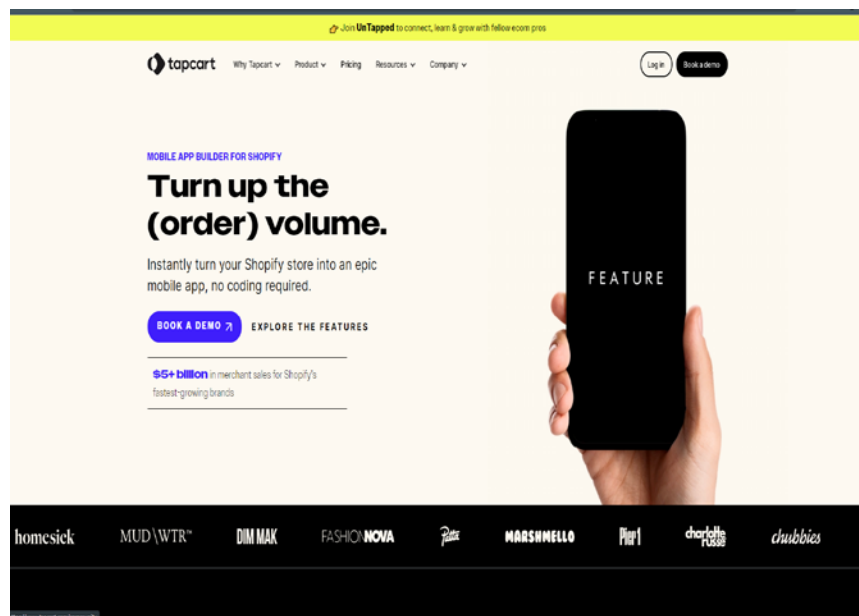


Рисунок 1.1 – Головна сторінка Tapcart [1]

Переваги:

- простота використання, Tapcart пропонує інтуїтивно зрозумілий інтерфейс, що дає змогу створювати мобільні додатки без необхідності знання програмування, тим самим спрощуючи процес розробки;
- широкий функціонал, платформа надає численні функції, серед яких: система оцінки товарів, можливість створення персональних кабінетів для користувачів, функція швидкого пошуку товарів тощо.

Недоліки:

- обмеження в налаштуваннях: використання заздалегідь визначених шаблонів та обмежених елементів дизайну може обмежити можливості для створення унікального вигляду додатку, ускладнюючи процес персоналізації;
- контроль над оновленнями: оскільки Tapcart є керованою платформою, користувачі можуть стикатися з обмеженим контролем над часом випуску оновлень та їх розгортанням;
- обмежені можливості кастомізації; для компаній, які потребують глибшої налаштування, виходячи за межі стандартних функцій Tapcart, відсутність доступу до коду або детальних параметрів налаштування може стати суттєвим недоліком.

Багато підприємств можуть віддати перевагу більшому контролю над кодом та інфраструктурою, чого неможливо досягти за допомогою цієї платформи. Tapcart дозволяє швидко створювати мобільний додаток без програмування, але обмеження в налаштуваннях шаблонів, контроль версій магазину і відсутність доступу до рівня розробника залишаються важливими аспектами, які не були враховані розробниками [1].

1.2.2 Shopney

Shopney – це платформа для розробки мобільних додатків, яка дозволяє компаніям у сфері електронної комерції створювати власні мобільні додатки для покупок. Оскільки основний акцент зроблений на зручності налаштування та простоті інтеграції з популярними платформами електронної комерції, Shopney надає комплексне рішення для бізнесів, які прагнуть вийти на ринок мобільних продажів [2].

Головна сторінка Shopney зображена на рисунку 1.2.

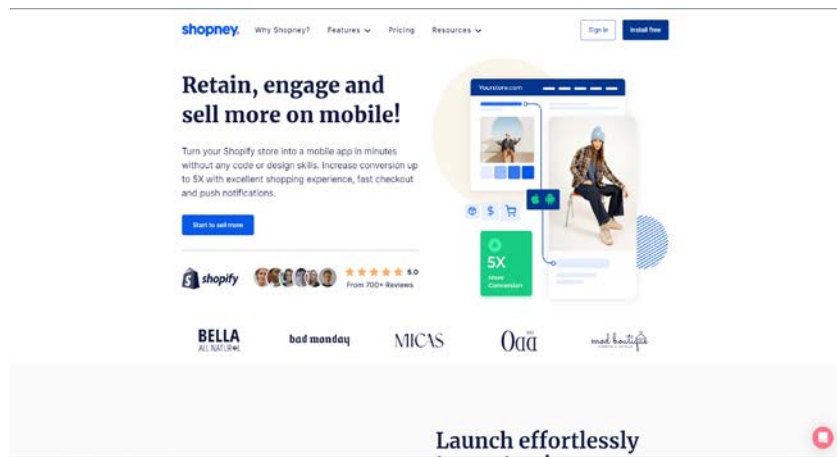


Рисунок 1.2 – Головна сторінка Shopney [2]

Переваги застосунку: інтеграція з платформами електронної комерції; персоналізація мобільного додатку; інтуїтивно зрозумілий інтерфейс.

Інтеграція з платформами електронної комерції. Shopney забезпечує легку інтеграцію з провідними платформами, такими як Shopify та WooCommerce, що дозволяє підприємствам безперешкодно синхронізувати свої каталоги продуктів, керувати інвентарем та обробляти замовлення. Це значно спрощує операційні процеси, оскільки компанії можуть уникати дублювання даних і ефективно працювати як через веб-сайт, так і через мобільний додаток [2].

Персоналізація мобільного додатку. Платформа надає розширені можливості налаштування, що дозволяє створювати фірмові мобільні програми, адаптовані під специфіку бренду. Змінюючи кольори, шрифти, піктограми та макети, компанії можуть забезпечити послідовний брендинг, що зміцнює довіру та впізнаваність серед клієнтів.

Інтуїтивно зрозумілий інтерфейс. За допомогою функції перетягування та попередньо створених шаблонів, компанії можуть швидко створювати візуально привабливі мобільні додатки без необхідності глибоких технічних знань. Це дозволяє зосередитися на користувацькому досвіді, а не на складних процесах розробки.

Недоліки: обмежена можливість налаштування; обмежений доступ до вихідного коду; потенційна схожість дизайну.

Обмежена можливість налаштування. Хоча платформа пропонує певний рівень персоналізації, для компаній, що мають специфічні вимоги до дизайну чи додаткової функціональності, це може бути обмеженням. Відсутність гнучкості в налаштуванні може стати проблемою для бізнесів, які потребують більш унікальних рішень.

Обмежений доступ до вихідного коду. Shopney не надає прямий доступ до основного коду додатка, що може бути недоліком для компаній, які потребують глибокої настройки або хочуть реалізувати складніші технічні рішення, що не підтримуються платформою.

Потенційна схожість дизайну. Оскільки платформа використовує шаблони для створення додатків, існує ймовірність, що кілька компаній можуть мати схожі дизайни. Це може ускладнити виділення бренду серед конкурентів і обмежити можливості для створення унікальних мобільних додатків. Хоча Shopney пропонує зручні інструменти для створення мобільних додатків для покупок, важливо враховувати її обмеження, зокрема у налаштуваннях дизайну та технічних можливостях, що можуть бути критичними для підприємств із високими вимогами [2].

1.2.3 AppMysite

AppMySite – це інноваційна платформа, що дозволяє компаніям перетворювати свої існуючі веб-сайти на базі WooCommerce або WordPress на повнофункціональні мобільні додатки. Завдяки можливостям AppMySite підприємства можуть значно розширити свою онлайн-присутність, пропонуючи клієнтам безперебійний та зручний досвід мобільних покупок, оптимізуючи свій бізнес для мобільних пристроїв [3].

Головна сторінка Appmysite зображена на рисунку 1.3.

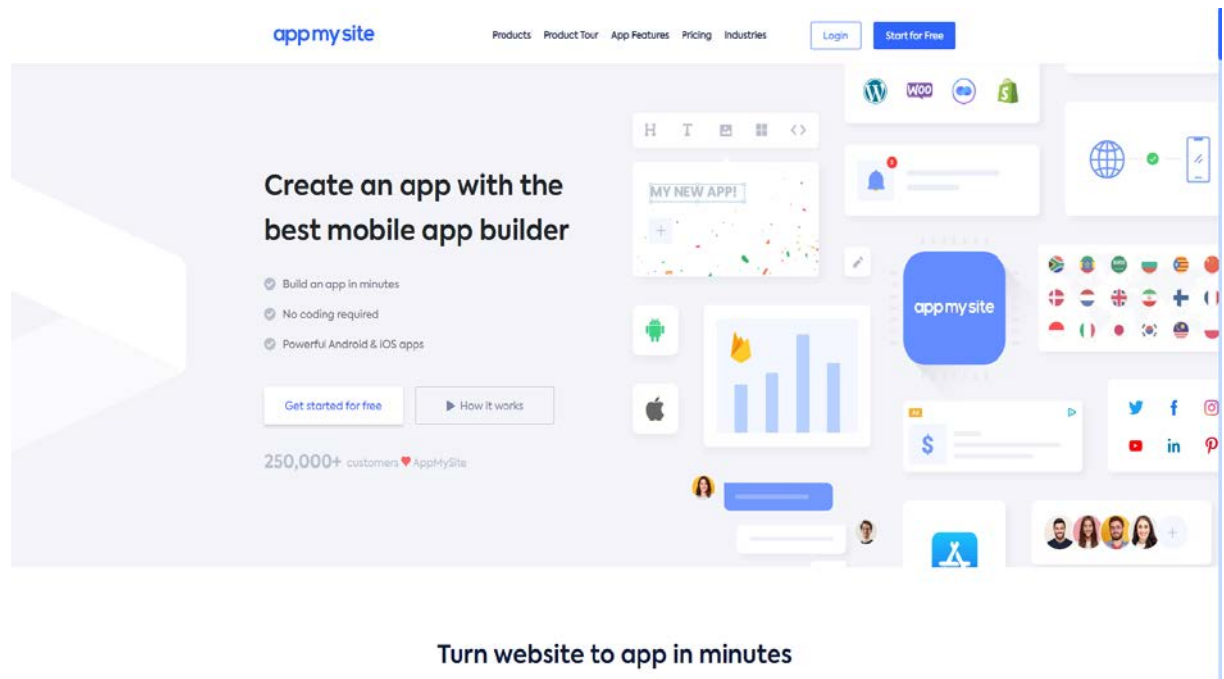


Рисунок 1.3 – Головна сторінка Appmysite [3]

Переваги системи перераховані нижче [3].

Легкість у перетворенні. Платформа AppMySite значно спрощує процес трансформації веб-сайтів на базі WooCommerce або WordPress у мобільні додатки. Завдяки інтуїтивно зрозумілому інтерфейсу, компанії можуть швидко створювати мобільні додатки без необхідності в програмуванні чи технічних знаннях. Це дозволяє підприємствам оперативно вийти на ринок мобільних додатків та залучити нову аудиторію.

Налаштування дизайну та брендинг. AppMySite пропонує широкі можливості для налаштування мобільного додатку, що дозволяє компаніям адаптувати його до власної бренд-ідентичності. Вибір кольорів, шрифтів, додавання логотипів і налаштування макетів дозволяють створити унікальний фірмовий мобільний додаток, який гармонійно доповнюватиме існуючий веб-сайт і підсилюватиме імідж бренду.

Push-сповіщення та залучення користувачів. Завдяки функції push-сповіщень компанії можуть надсилати персоналізовані повідомлення, рекламні акції та важливі оновлення безпосередньо своїм користувачам. Це дозволяє утримувати зв'язок із клієнтами, підвищувати їхню залученість і

сприяти збільшенню продажів, оскільки своєчасні сповіщення сприяють утриманню користувачів.

Недоліки перераховані нижче.

Обмежене налаштування. Хоча платформа пропонує деякі можливості для налаштування, компанії з дуже специфічними вимогами до дизайну або потребами у складних функціях можуть зустрітися з обмеженнями, що не дозволяють повною мірою реалізувати бажану кастомізацію додатку;

Проблеми з масштабованістю. Для компаній, що швидко зростають або мають великі клієнтські бази, AppMySite може бути обмеженою платформою. Проблеми з обробкою великих обсягів даних або високою кількістю одночасних користувачів можуть призвести до зниження продуктивності та швидкості роботи додатку на етапах масштабування;

Обмежена автономна функціональність. Платформа може не підтримувати розширені офлайн-функції, такі як перегляд контенту або управління кошиком без підключення до Інтернету. Для підприємств, які потребують функцій автономної роботи, це може бути важливим недоліком.

Загалом, AppMySite пропонує зручне рішення для перетворення веб сайтів WooCommerce або WordPress на мобільні додатки, однак перед вибором платформи варто врахувати її обмеження щодо налаштування, масштабованості та офлайн-функціональності.

В таблиці 1.1 наведено зведена інформація про переваги та недоліки платформ.

Таблиця 1.1 – Переваги та недоліки платформ

Платформа	Переваги	Недоліки
Tapcart	Інтуїтивний інтерфейс "drag-and-drop". Широкий функціонал (оцінка товарів, пошук, кабінети).	Обмеження персоналізації. Відсутність доступу до коду. Контроль оновлень платформою.

Продовження таблиці 1.1.

Платформа	Переваги	Недоліки
Shopney	Легка інтеграція з Shopify та WooCommerce. Персоналізація дизайну під бренд. Простий інтерфейс.	Відсутність доступу до коду. Можливе повторення дизайнів. Обмежена унікальність налаштувань.
AppMySite	Легкість перетворення сайтів WooCommerce/WordPress на додатки. Push-сповіщення. Налаштування брендингу.	Проблеми з масштабованістю. Обмежена автономна функціональність. Менше варіантів кастомізації.

1.3 Дослідження методологій швидкої розробки

Швидка розробка програмного забезпечення дозволяє компаніям швидко створювати продукти, зокрема мобільні додатки, за допомогою різних методів і технологій. Це дозволяє знизити витрати, прискорити вихід на ринок і адаптувати продукт до змінюваних вимог.

Однією з основних методологій є гнучкі методи, такі як Agile та Extreme Programming, які включають короткі цикли розробки та постійне вдосконалення продукту на основі зворотного зв'язку від користувачів. Це дозволяє оперативно реагувати на зміни та мінімізувати час розробки.

Крім того існує метод прототипування, який дозволяє швидко створити попередні версії продукту для тестування і отримання зворотного зв'язку.

Також використовуються готові рішення та шаблони, що особливо важливо для мобільних додатків, де можна швидко адаптувати функціонал без потреби в розробці інтерфейсу і архітектури з нуля. У застосунку застосовуються подібні підходи для значного скорочення часу на розробку і зниження витрат.

Підхід швидкої розробки є актуальним і необхідним, оскільки він дозволяє бізнесам швидко реагувати на ринкові зміни, знижувати витрати і перевіряти концепції без значних інвестицій. Хоча методологія має недоліки, більшість з них можна мінімізувати через обґрунтоване планування та поетапне вдосконалення продукту після запуску. Переваги та недоліки швидкої розробки зображені на таблиці 1.2.

Таблиця 1.2 – Переваги та недоліки підходу швидкої розробки

Переваги	Недоліки
Швидкість виходу на ринок: скорочення часу від початку розробки до релізу, що дозволяє швидше отримати результат і реакцію від користувачів.	Якість коду: при швидкій розробці можуть виникати компроміси, які впливають на довгострокову якість і підтримуваність коду.
Економія ресурсів: використання шаблонів, готових рішень і прототипів знижує витрати на розробку та залучення персоналу.	Часова вартість кастомізації: створення унікального дизайну та функціоналу вимагає більше часу й ресурсів, ніж використання готових рішень.
Адаптивність до змін: гнучкі методології дозволяють швидко змінювати продукт у відповідь на зворотний зв'язок.	Залежність від готових рішень: при зміні чи знятті з підтримки готових інструментів можуть виникнути проблеми з масштабуванням.
Тестування концепцій: завдяки прототипуванню можна швидко перевірити ідеї, зменшуючи ризики невдачі продукту.	Можливі технічні обмеження: готові компоненти чи шаблони можуть не підтримувати всі необхідні функції.

1.4 Висновок за розділом 1

У першому розділі було досліджено актуальність швидкої розробки мобільних додатків, зокрема для електронної комерції, де швидкість і адаптивність мають вирішальне значення. Проведено аналіз сучасних платформ, таких як Tapcart, Shopney та AppMySite, із визначенням їхніх

переваг і недоліків, що допомагає зрозуміти можливості інтеграції готових рішень. Також розглянуто методології швидкої розробки, включаючи Agile, прототипування та використання шаблонів, які дозволяють бізнесам мінімізувати витрати і ризики, забезпечуючи гнучкість та ефективність процесів. Усі ці аспекти підтверджують важливість впровадження сучасних підходів до розробки, що сприяють прискоренню виходу продуктів на ринок і підвищенню їхньої конкурентоспроможності.

2 ОПИС ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Технологічний стек

Для реалізації нашого проекту були обрані наступні технології:

- frontend: Next.js, React.js, React Native;
- backend: Express.js, MySQL, Redis.

Використання комплексного технологічного стека підвищує ефективність, масштабованість і зручність розробки програмного забезпечення.

Ці технології дозволяють створювати інтегровану систему, де кожен компонент доповнює інший, забезпечуючи швидкість і зручність обслуговування. Комбінація серверних та клієнтських технологій на основі JavaScript створює потужну і зручну екосистему для розробки додатків. JavaScript, як основна мова програмування, має ряд переваг [4]:

- асинхронне програмування: дозволяє не блокувати інтерфейс користувача;
- інтеграція з API: дає змогу взаємодіяти з різними зовнішніми сервісами;
- масштабованість: Next.js пропонує гнучкі рішення для великих програм.

Далі буде розглянуто окремо переваги кожної з основних технологій нашого стеку.

2.1.1 Express.js

Express.js є популярним фреймворком для розробки серверних додатків на Node.js. Він дозволяє швидко створювати масштабовані і ефективні веб-додатки, забезпечуючи зручний механізм маршрутизації, інтеграцію з API та підтримку різних проміжних компонентів [5].

Переваги фреймворку Express.js:

- простота маршрутизації та REST API: лаконічний і зручний механізм для створення API;
- middleware: дозволяє легко додавати нові функціональні можливості;
- масштабованість: ефективно обробляє велику кількість запитів завдяки неблокуючій архітектурі.

2.1.2 MySQL та Redis

Для зберігання даних використовуються дві технології: MySQL та Redis, що забезпечують надійне зберігання та швидке кешування [6].

Далі перераховані основні моменти щодо бази даних MySQL[7]:

- це реляційна база даних: відома своєю надійністю і підтримкою ACID;
- оптимізація запитів: використовує індексацію та кешування для покращення продуктивності;
- кешування в пам'яті: забезпечує дуже швидкий доступ до даних;
- гнучкість: підтримує різноманітні структури даних для зберігання.

2.1.3 Next.js та React.js

Next.js – це фреймворк для створення динамічних веб-додатків. Він забезпечує зручну маршрутизацію та підтримку серверного рендерингу (SSR), що покращує SEO та швидкість завантаження. React.js відповідає за створення інтерактивних компонентів інтерфейсу [8].

Переваги Next.js:

- автоматична маршрутизація: зручне створення сторінок без складних налаштувань;
- SSR і SSG: підтримка серверного рендерингу та статичної генерації;
- API маршрути: дозволяє створювати безсерверні кінцеві точки API.

Переваги React.js:

- компонентна архітектура: дозволяє створювати повторно використовувані компоненти;
- віртуальний DOM: забезпечує високу швидкість рендерингу та взаємодії з користувачем.

2.1.4 React Native

React Native дозволяє розробляти мобільні додатки для iOS і Android з однією кодовою базою. Це дозволяє скоротити час розробки і забезпечити узгодженість функціоналу на різних платформах. Переваги: кросплатформність та підтримка одночасної розробки для iOS та Android; гаряче перезавантаження, зміни відразу відображаються в додатку; швидкість розробки значно скорочує час на створення та тестування.

2.1.5 Typescript

TypeScript додає статичну типізацію до JavaScript, що покращує якість коду та робить його більш надійним і зручним для масштабованих проєктів.

Переваги:

- типізація: допомагає уникати помилок на етапі розробки;
- інтеграція з JavaScript: сумісний з популярними бібліотеками і фреймворками;
- масштабованість: зручний для великих проєктів завдяки своїй статичній типізації.

В таблиці 2.1 зведено опис та переваги технологічного стеку який буде використовуватись для розробки системи.

Таблиця 2.1 – Опис обраного технологічного стеку розробки

Технологія	Опис	Переваги
Express.js	Фреймворк для Node.js, який спрощує створення серверних додатків	Простота маршрутизації, REST API, масштабованість, middleware
MySQL	Реляційна база даних, використовується для зберігання даних	Надійність, підтримка ACID, оптимізація запитів
Redis	Система кешування даних у пам'яті	Швидкість доступу, підтримка різноманітних структур даних
Next.js	Фреймворк для створення динамічних веб-додатків	Автоматична маршрутизація, підтримка SSR, API маршрути
React.js	Бібліотека для створення користувацьких інтерфейсів	Компонентна архітектура, віртуальний DOM, висока швидкість рендерингу
React Native	Фреймворк для кросплатформенних мобільних додатків	Кросплатформність, швидкість розробки, гаряче перезавантаження
TypeScript	Мова програмування, яка додає типізацію до JavaScript	Типізація, сумісність з JavaScript, масштабованість

2.2. Допоміжні технології для розробки

У процесі реалізації застосунку було використано низку допоміжних технологій, які оптимально відповідають поставленим завданням. Їхнє впровадження обґрунтоване необхідністю забезпечення ефективності,

інтеграції та надійності всіх компонентів системи. До обраних інструментів належать:

- Expo;
- Postman;
- Open Server;
- phpMyAdmin;
- Upstash;
- Shopify.

2.2.1 Expo

Expo є платформою, що спрощує процес розробки мобільних додатків за рахунок надання інструментів для створення кросплатформного програмного забезпечення.

У межах цього проекту Expo використовується для забезпечення швидкої розробки функціоналу мобільного застосунку, створення єдиного базового коду для платформ iOS та Android, а також для оптимізації тестування та інтеграції. Це дозволяє скоротити час розробки та забезпечити якісну реалізацію користувацького інтерфейсу.

2.2.2 Postman

Postman — це популярний інструмент для тестування та проектування API. У нашому проекті він слугує ключовим засобом для перевірки взаємодії мобільного застосунку з API платформи Shopify.

Зокрема, Postman дозволяє тестувати запити для отримання даних про товари, оновлення інформації про замовлення та реалізацію функцій аутентифікації. Таким чином, цей інструмент сприяє стабільності та надійності інтеграції зовнішніх сервісів [9].

2.2.3 Table Plus

Table Plus — це сучасний інструмент для адміністрування баз даних, який надає зручний інтерфейс і багатий функціонал для ефективного управління базами даних. Він підтримує різноманітні типи баз даних, такі як MySQL, PostgreSQL, Redis, SQLite та інші, що робить його універсальним рішенням для розробників і адміністраторів.

Ключовою перевагою Table Plus є його інтуїтивно зрозумілий і добре організований інтерфейс. Завдяки цьому користувачі можуть легко виконувати основні операції, такі як створення, редагування, видалення таблиць і виконання запитів SQL, а також керувати складними структурами даних. Table Plus забезпечує швидкий доступ до функцій і дозволяє виконувати дії з мінімальними зусиллями.

Додатково програма пропонує функції автоматичного збереження, вбудовану підтримку SSH для захищених з'єднань і потужні інструменти для редагування даних, що значно полегшує роботу з базами даних у різних середовищах.

2.2.4 Redis Insight

Redis Insight — це потужний інструмент для моніторингу, управління та візуалізації даних у базах даних Redis. Він забезпечує зручний інтерфейс, що дозволяє розробникам ефективно працювати зі структурами даних Redis, виконувати операції та аналізувати дані в реальному часі.

Однією з ключових переваг Redis Insight є його простота використання: розробники можуть легко переглядати ключі, структури даних, зберігати й отримувати інформацію, а також виконувати знайомі команди Redis через інтегровану консоль. Інструмент підтримує складні структури даних, такі як списки, множини, хеші та відсортовані множини, забезпечуючи повноцінне управління всіма можливостями Redis [10].

Зокрема, Redis Insight пропонує візуалізацію даних і метрик, що дозволяє швидко діагностувати продуктивність бази даних, аналізувати запити та оптимізувати використання Redis у додатках.

2.2.5 Shopify

Завдяки Shopify забезпечується інтеграція даних про товари, замовлення та інші компоненти електронної комерції з мобільним застосунком. Shopify використовується для отримання актуальної інформації про товари, а також для реалізації функцій управління замовленнями, що дозволяє дотримуватися принципу узгодженості між компонентами системи та підвищувати зручність для користувачів.

2.3 Висновок за розділом 2

У другому розділі було обрано та обґрунтовано технологічний стек для реалізації проекту, що включає як основні інструменти для фронтенд та бекенд-розробки, так і допоміжні засоби. Детально розглянуто переваги обраних технологій, які створюють потужну, гнучку екосистему для створення програмного забезпечення. Завдяки інтеграції цих технологій досягається висока продуктивність, зручність розробки та відповідність сучасним вимогам. Окремо було проаналізовано допоміжні інструменти, які забезпечують оптимізацію процесів розробки та інтеграцію зовнішніх. Ці технології значно спрощують розробку та тестування мобільного додатку, дозволяючи забезпечити якість та стабільність системи.

3 РОЗРОБКА ЕКОСИСТЕМИ

3.1 Розробка серверної частини

3.1.1 Структура серверної частини

На сервері Express.js упорядкування коду в різних папках допомагає підтримувати структуровану та модульну архітектуру. Кожна папка служить певній меті та містить пов'язані файли, які обробляють різні аспекти програми. Папки, з яких складається наш сервер: `routes`; `controllers`; `helpers`; `services`; `utils`; `сron`.

Routes (маршрути): Ця папка відповідає за визначення та обробку маршрутів для конкретних кінцевих точок або ресурсів. Кожен файл в цій папці містить визначення маршруту і логіку для обробки HTTP-запитів (GET, POST, DELETE тощо) для відповідного ресурсу. Наприклад, файл маршруту для користувачів може містити маршрути для реєстрації, входу та оновлення профілю користувача.

Controllers (контролери): Папка містить файли, які реалізують логіку для обробки запитів та генерації відповідей. Контролери слугують посередниками між маршрутами та сервісами. Вони приймають запити від користувачів, взаємодіють з відповідними сервісами для отримання даних і відправляють відповіді назад до клієнта. Наприклад, контролер для користувачів може містити методи для автентифікації та реєстрації.

Helpers (допоміжні функції): Ця папка містить загальні допоміжні функції, які використовуються в різних частинах програми. Наприклад, це можуть бути функції для валідації даних, обробки запитів або форматування дат. Вони забезпечують повторне використання коду та полегшують розробку.

Services (служби): Папка служить для інкапсуляції складної бізнес-логіки та операцій, пов'язаних з виконанням завдань програми. Служби надають високий рівень абстракції, що дозволяє зручно розподіляти навантаження та повторно використовувати код. Наприклад, служба автентифікації може відповідати за перевірку токенів та управління сесіями.

Utils (утиліти): Ця папка містить загальні функції або модулі, які надають часто використовувані функціональності для програми, наприклад, функції для роботи з датами, обробки помилок, або шифрування. Вони дозволяють уникнути дублювання коду та спрощують підтримку проекту.

Cron (планувальники завдань): Папка для зберігання файлів, що визначають завдання, які повинні виконуватися за розкладом. Це можуть бути завдання для періодичного виконання, такі як очищення старих записів у базі даних або відправка електронних листів.

Ці папки структурують проект і допомагають зберігати чистоту коду, забезпечуючи модульність та легкість підтримки [11].

3.1.2 Схема серверної частини

На рисунку 3.1 зображена схема серверної частини

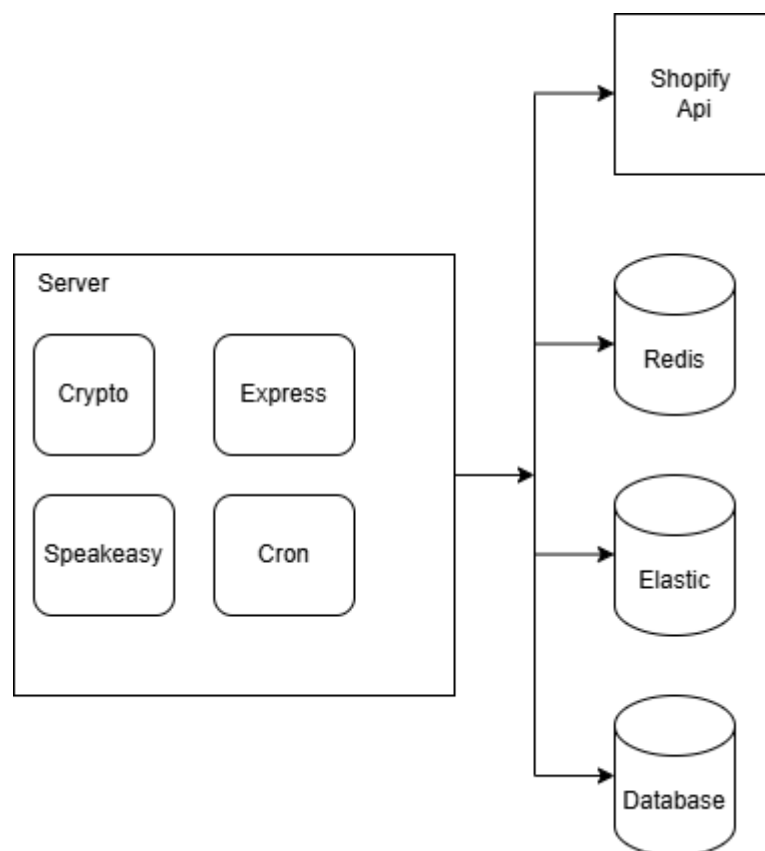


Рисунок 3.1 – Схема серверної частини

Елементи схеми перелічені нижче.

Server – центральний компонент серверної архітектури, який забезпечує обробку запитів і виконання ключових операцій. Він включає наступні модулі:

- Crypto – модуль для криптографічних операцій, таких як шифрування, хешування або генерація токенів безпеки;
- Express – веб-фреймворк для створення серверного застосунку, який обробляє HTTP-запити, маршрутизацію та надає API клієнтським запитам;
- Speakeasy – бібліотека для реалізації двофакторної автентифікації (2FA), що підвищує безпеку системи;
- Cron – модуль для виконання запланованих завдань на сервері, таких як регулярне оновлення даних або автоматизоване очищення.

Shopify API зовнішній інтерфейс для інтеграції з платформою електронної комерції Shopify. Сервер використовує цей API для отримання даних про продукти, замовлення, клієнтів та іншу інформацію, що синхронізується з додатком.

Redis – система зберігання даних у пам'яті, яка використовується для кешування запитів, зберігання тимчасових даних або прискорення доступу до часто використовуваних елементів.

Elastic – пошуковий рушій, що використовується для швидкого та ефективного виконання запитів до великих обсягів даних. Він забезпечує пошук і фільтрацію інформації, наприклад, продуктів у магазині.

Database. Основна база даних для довготривалого зберігання інформації, такої як користувачі, замовлення, налаштування системи та інші важливі дані.

3.1.3 Маршрутизація на сервері

Розглянемо всі обробники маршрутів, які присутні на сервері та їх функціонал. Список маршрутів, які оброблює сервер:

- analytics;
- products;
- store;
- storeSettings;
- user.

Analytics (аналітика):

а) призначення: надавання аналітичних даних;

б) методи:

1) GET /analytics/:id – повертає загальні аналітичні дані для магазину;

2) POST /analytics – надсилає на обробку нові аналітичні дані;

в) функціонал:

1) отримує та представляє дані про поведінку користувачів;

2) приймає нові дані для аналізу в реальному часі;

3) визначає джерела трафіку та показники залучення.

Products (продукти):

а) призначення: надавання інформації про продукти з Shopify за ідентифікатором;

б) методи:

1) GET /products/id – повертає відомості про продукт за ідентифікатором;

в) функціонал:

1) Отримує список продуктів, доступних у магазині;

2) Дозволяє клієнтам переглядати детальну інформацію про кожен продукт;

3) Інтегрується з Shopify API для отримання даних із магазину Shopify.

Store (магазин):

а) Призначення: створення магазину, повернення списку всіх магазинів;

б) Методи:

- 1) GET /store – повертає список всіх магазинів;
- 2) POST /store – отримує дані для створення магазину;

в) функціонал:

- 1) повернення списку магазинів для персонального кабінету;
- 2) додавання магазину до персонального кабінету;
- 3) запис магазину до бази даних та Redis.

StoreSettings (налаштування магазину):

а) призначення: надавання інформації щодо налаштувань магазину;

б) методи:

- 1) GET /storeSettings/:id – повертає інформацію щодо налаштувань магазину за ідентифікатором;
- 2) POST /storeSettings/:id – надсилає на обробку нові налаштування за ідентифікатором;

в) функціонал:

- 1) повернення загальних або окремих налаштувань для магазину;
- 2) запис нових налаштувань для магазину.

User (користувач):

а) призначення: керування операцій, пов'язаними з користувачами;

б) методи:

- 1) POST /user/register – запис нового користувача до бази даних;
- 2) POST /user/login – авторизація користувача;
- 3) POST /user/verify/:id – автентифікація користувача за ідентифікатором;
- 4) GET /session/:id – отримання сесії користувача у Redis.

3.1.4 Функціонал серверної частини

Розглянемо приклад одного з наших запитів. Рисунок 3.2 демонструє функціональну схему серверної частини.

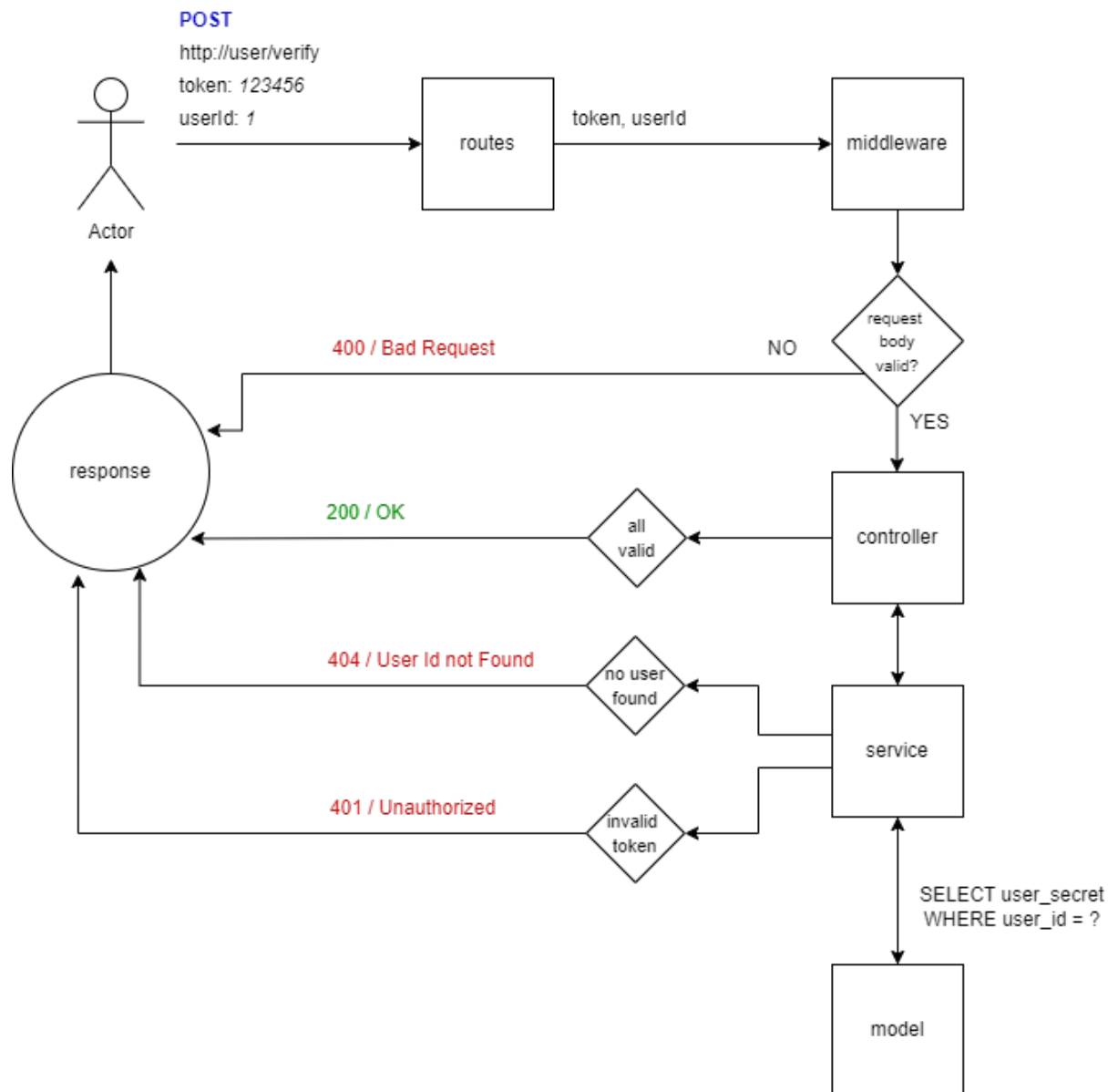


Рисунок 3.2 – Функціональна схема серверної частини

Спочатку користувач надсилає запит до сервера у форматі HTTP. Запит може використовувати різні методи. Розглянемо найбільш поширені з них:

- GET: використовується для отримання ресурсу з сервера;
- POST: надсилає дані до зазначеного ресурсу;

- DELETE: використовується для видалення ресурсу;
- PATCH: застосовується для часткового оновлення ресурсу на сервері.

У нашому випадку користувач надсилає запит методом POST, що означає, що запит містить певну інформацію.

Далі сервер приймає запит і направляє його до відповідного маршруту в папці "route", яка відповідає за визначення і обробку маршрутів. Однак перед тим, як обробити запит, необхідно перевірити його зміст. Для цього використовується "middleware".

У "middleware" є файл, який перевіряє коректність запиту. Наприклад, якщо користувач не вказав свій унікальний ідентифікатор або токен для автентифікації, сервер повертає помилку "400 Bad Request". Цей статус вказує на те, що сервер не може обробити запит через помилку з боку клієнта.

Якщо всі необхідні поля надано, сервер передає запит до "controller". Як зазначалося раніше, "controller" відповідає за логіку обробки запитів та генерацію відповідей. У нашому випадку, якщо "service" повертає позитивну відповідь, ми повертаємо статус "200 OK", що означає успішне виконання запиту.

Однак "service" також може повернути помилку. Наприклад, при зверненні до бази даних у папці "models", якщо унікальний ідентифікатор користувача не знайдений, сервер поверне статус "404 Not Found", що означає, що запитаний ресурс не знайдений.

Якщо при перевірці токена виявляється, що він неправильний, сервер повертає статус "401 Unauthorized", що вказує на відсутність дійсних облікових даних для автентифікації.

Нижче наведені приклади двох запитів для однієї і тієї ж URL-адреси. На рисунку 3.3 показано успішний запит зі статусом "200 OK", а на рисунку 3.4 — некоректний запит, який повертає статус "400 Bad Request", оскільки одне з полів вказано неправильно. Відповідь також містить поле "message", яке пояснює причину помилки.

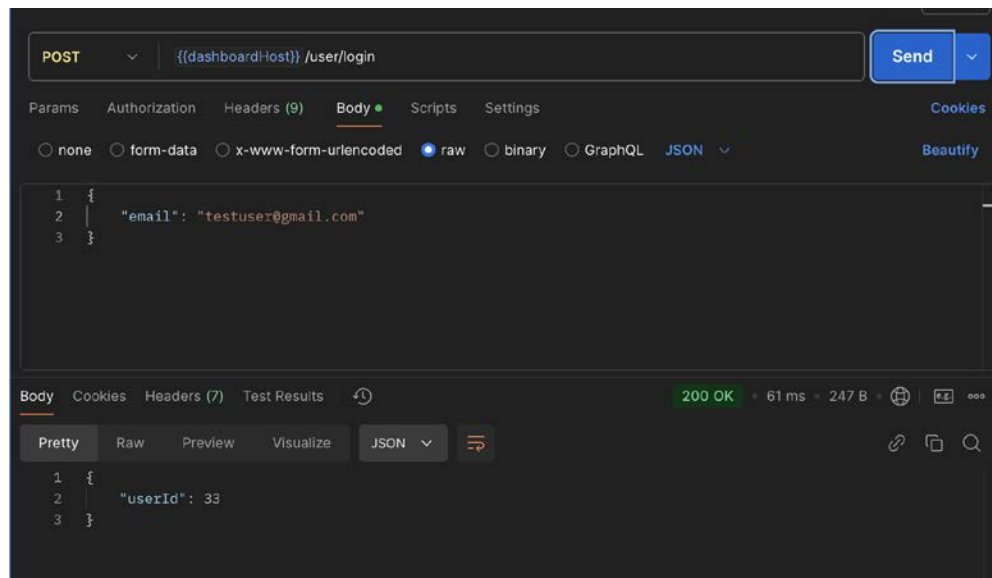


Рисунок 3.3 – Приклад успішного запиту

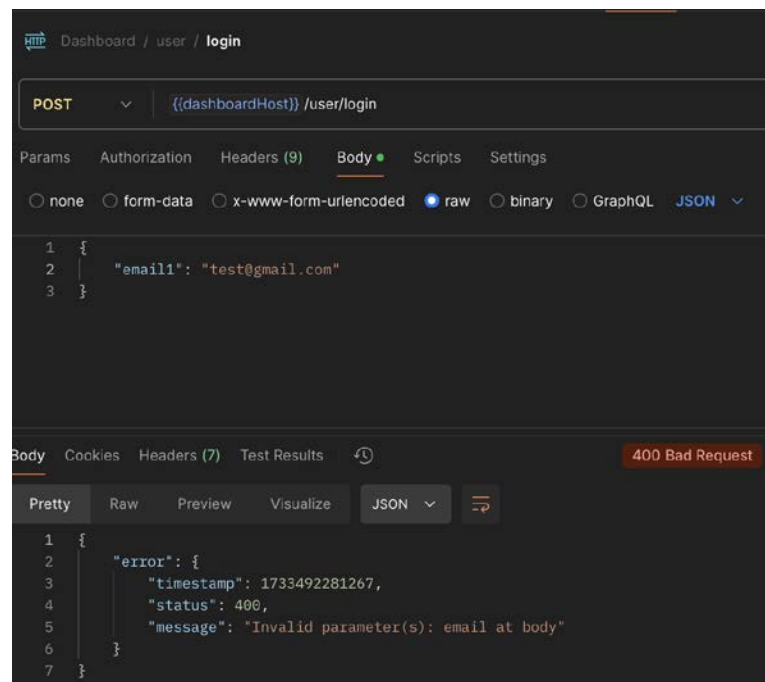


Рисунок 3.4 – Приклад некоректного запиту

Помилка може бути викликана практично у будь-якому модулі програми. Для цього, ми відтворили функцію `try catch`, яка містить в собі контролер. Код функції зображений на рисунку 3.5.

```
const tryCatch = (controller) => async (req, res, next) => {
  try {
    await controller(req, res);
  }
  catch (error) {
    return next(error);
  }
}

module.exports = tryCatch;
```

Рисунок 3.5 – Функція tryCatch

Розглянемо приклад повернення помилки сервісу автентифікації користувача на рисунку 3.6.

```
const verifyUserService = async ({ id, token, }) => {
  const result = await executeQuery(
    'SELECT secret FROM users WHERE id = $1',
    [id]
  );

  if(result.rowCount === 0)
    throw getError(404, 'User with such id was not found.').

  const { secret } = result.rows[0];

  const verified = speakeasy.totp.verify({
    encoding: 'base32',
    secret: decrypt(secret),
    token: token,
  });

  if(!verified)
    throw getError(401, 'Verification failed. Wrong token');

  const existingToken = await getRedisText(`authToken:${id}`);

  if (existingToken) {
    return { authToken: existingToken }
  } else {
    const createdToken = uuidv4();
    await setRedisText(`authToken:${id}`, createdToken, 36000);

    return { authToken: createdToken }
  }
}
```

Рисунок 3.6 – Сервіс автентифікації користувача

На рисунку ми бачимо, що спочатку запитується інформація для перевірки користувача. Якщо дані не знайдено, повертається помилка через метод throw з викликом функції getError. Подібним чином повертається

помилка, якщо користувач не проходить автентифікацію, тобто при неправильному токєні.

Функція `getError` повертає об'єкт помилки, який можна побачити на рисунку 3.7. Цей об'єкт містить наступні поля:

- `timestamp`: час повернення помилки;
- `type`: тип помилки;
- `status`: статус коду помилки;
- `message`: пояснювальний текст помилки.

```
const getError = (status, message, rest) => ({  
  error: {  
    timestamp: Date.now(),  
    status: status,  
    message: message,  
    ...rest,  
  }  
});  
  
module.exports = getError;
```

Рисунок 3.7 - Об'єкт повернення помилки

3.2 Розробка клієнтської частини для браузера

3.2.1 Структура клієнтської частини для браузера

На клієнтській стороні для браузера була розроблена модульна та структурована архітектура, яка полегшує керування та масштабування проекту. Ось список основних папок, які використовуються на клієнтській частині Next.js:

- `components`;
- `pages`;

- public;
- styles;
- helpers;
- utils;
- store.

Components. Дана папка містить багаторазові компоненти інтерфейсу користувача, написані за допомогою React.js. Організація компонентів в окремій папці сприяє повторному використанню коду та модульності.

Pages. Ця папка є ключовою папкою в Next.js і відповідає системі маршрутизації на основі файлів. Кожен файл у папці представляє певний маршрут або сторінку у вашій програмі. Наприклад, «pages/index.js» представляє домашню сторінку, а «pages/dashboard/index.js» представляє сторінку, яка містить адміністративну панель. Next.js автоматично обробляє маршрутизацію на основі файлів у цій папці.

Також дана папка містить в собі ще одну папку, яка називається “api”. Дана папка використовується для створення без-серверних кінцевих точок API. Файли в цій папці визначають маршрути API, які можна використовувати для обробки логіки на стороні сервера або служити серверною частиною для програми. Ці маршрути API можна викликати з інтерфейсних компонентів або із зовнішніх джерел.

Public. Дана папка використовується для зберігання статичних файлів.

Styles. Ця папка містить модулі CSS. Окреме зберігання файлів, пов’язаних із стилями, дозволяє краще організувати та керувати стилями в різних компонентах і на сторінках.

Utils. Папка "utils" зазвичай містить службові функції або модулі, які надають функціональні можливості. Ці утиліти можуть включати функції для обробки даних, виклики API, або будь-які інші спільні функції.

Helpers. Дана папка містить файли, які містять в собі допоміжні файли для розробки. Вони можуть забезпечити підтримку маніпулювання даними, обчислень, форматування, перевірки або будь-яких інших операцій.

Store. Дана папка відповідає за реалізацію Redux. Це є сховище, яке зберігає стан програми та полегшує керування станом у програмі.

3.2.2 Допоміжні бібліотеки для клієнтської частини браузеру

Розглянемо кожну бібліотеку окремо, які знаходяться на рис. 3.8:

```
"dependencies": {  
  "@dnd-kit/sortable": "^8.0.0",  
  "@dnd-kit/utilities": "^3.2.2",  
  "axios": "^1.7.3",  
  "chart.js": "^4.4.3",  
  "clsx": "^2.1.1",  
  "next": "14.2.5",  
  "nookies": "^2.5.2",  
  "react": "^18",  
  "react-chartjs-2": "^5.2.0",  
  "react-dom": "^18"  
},
```

Рисунок 3.8 – Допоміжні бібліотеки для клієнтської частини браузеру

@dnd-kit/sortable: бібліотека для реалізації перетягування (drag-and-drop) елементів у списках з можливістю їх сортування. Цей пакет є частиною набору інструментів @dnd-kit, призначеного для створення інтерактивних елементів на основі перетягування.

@dnd-kit/utilities: допоміжні утиліти для бібліотеки @dnd-kit, що забезпечують зручні функції для роботи з перетягуванням елементів у інтерфейсі користувача.

axios: бібліотека для виконання HTTP-запитів. Вона дозволяє зручно взаємодіяти з RESTful API, підтримує асинхронне виконання запитів і забезпечує можливість обробки помилок, а також автоматичну перетворення даних у формат JSON.

`chart.js`: бібліотека для побудови інтерактивних графіків. Вона дозволяє створювати різноманітні діаграми та графіки, зокрема лінійні, стовпчасті, кругові та інші типи, що використовуються для візуалізації даних.

`clsx`: легка бібліотека для умовного застосування CSS-класів. Вона дозволяє динамічно додавати класи до елементів на основі умов, що допомагає в управлінні стилями.

`next`: фреймворк для розробки додатків на React, який підтримує серверний рендеринг (SSR) та генерацію статичних сайтів (SSG). Це потужний інструмент для побудови високопродуктивних веб-додатків.

`cookies`: бібліотека для роботи з куками в додатках, побудованих на Next.js. Вона надає зручні методи для отримання, збереження та видалення куків на сервері та клієнті.

`react`: бібліотека для побудови інтерфейсів користувача. React дозволяє створювати компоненти, які можуть відображати дані та автоматично оновлюватися при їх зміні, забезпечуючи високу ефективність відображення.

`react-chartjs-2`: бібліотека для інтеграції Chart.js з React. Вона дозволяє зручно створювати інтерактивні графіки в додатках на базі React.

`react-dom`: пакет, який надає спеціальні методи для взаємодії з DOM у браузері в додатках на React. Він використовується для рендерингу компонентів React у DOM, а також забезпечує їх оновлення та керування їх життєвим циклом.

Ці залежності та `devDependencies` необхідні для розробки програми Next.js із React, Redux, TypeScript та іншими необхідними інструментами та бібліотеками для обробки HTTP-запитів, взаємодії з Redis та керування станом.

3.2.3 Структура клієнтської частини для мобільного застосунку

Структура мобільного застосунку майже не відрізняється від структури застосунку для браузера. Але, у даному випадку ми не використовуємо Next.js

для маршрутизації. Розглянемо перелік основних папок, з яких складається наш мобільний застосунок:

- components;
- helpers;
- store;
- utils;
- assets.

Як ми можемо побачити, в нас додалась нова папка assets. Вона відповідає за збереження статичних файлів, таких як: картинки, іконки та шрифти.

3.2.4 Схема клієнтської частини

На рисунку 3.9 зображена схема клієнтської частини.

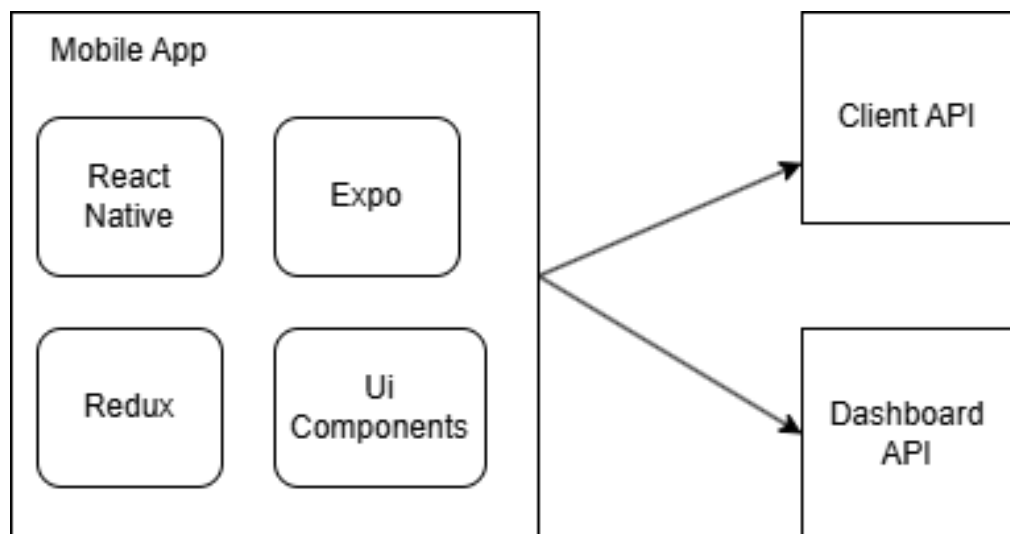


Рисунок 3.9 – Схема клієнтської частини

Mobile App. Центральний компонент, що представляє мобільний застосунок, створений із використанням наступних технологій.

React Native – основна платформа для кросплатформної розробки, що забезпечує роботу застосунку як на iOS, так і на Android.

Expo – інструмент для прискорення розробки, тестування та розгортання застосунків, інтегрований із React Native.

Redux – бібліотека для управління станом застосунку, що допомагає синхронізувати дані між різними компонентами.

UI Components – набір компонентів користувацького інтерфейсу, що забезпечують взаємодію користувача із застосунком.

Client API. Інтерфейс прикладного програмування, що відповідає за обробку запитів від мобільного застосунку, пов'язаних із даними клієнтів. Це включає отримання інформації про замовлення, аналітики тощо.

Dashboard API. Інший API, який взаємодіє з мобільним застосунком для доступу до функцій адміністративної панелі. Він використовується перегляду налаштувань магазину.

Взаємодія:

- мобільний застосунок через компоненти React Native та Expo забезпечує взаємодію користувача з інтерфейсом і даними;
- Redux забезпечує централізоване управління станом, підтримуючи синхронізацію між інтерфейсом і даними, які надходять від API;
- запити від застосунку надсилаються до Client API і Dashboard API залежно від необхідної функціональності;
- Client API обробляє дані, пов'язані з клієнтською частиною (замовлення, продукти);
- Dashboard API використовується для адміністративних задач, таких як керування магазином.

3.3 Опис бази даних

Для проекту була побудована база даних із декількох таблиць, що відображають ключові аспекти роботи системи. Нижче наведено опис кожної таблиці.

Таблиця "users" (користувачі). Ця таблиця зберігає інформацію про користувачів, які мають доступ до системи. Поля таблиці:

- id (первинний ключ) — унікальний ідентифікатор користувача;
- is_admin — прапорець, який визначає, чи є користувач адміністратором;
- email_hash — хеш електронної пошти користувача;
- email_encrypted — зашифрована електронна пошта користувача;
- secret — зашифрований секрет для додаткової безпеки.

Таблиця "stores" (магазини). Таблиця відповідає за зберігання інформації про магазини. Поля таблиці:

- id (первинний ключ) — унікальний ідентифікатор магазину;
- store_name — назва магазину;
- current_plan_id — ідентифікатор плану магазину (зв'язок із таблицею store_plans);
- api_key — зашифрований API-ключ;
- filters, collections, fields — JSON-поля, що містять налаштування магазину;
- access_token — токен для доступу до API (необов'язкове);
- account_name — ім'я облікового запису власника магазину.

Таблиця "store_plans" (плани магазинів). Ця таблиця зберігає інформацію про доступні плани магазинів. Поля таблиці:

- id (первинний ключ) — унікальний ідентифікатор плану;
- name — назва плану.

Таблиця "analytics" (аналітика). Ця таблиця зберігає інформацію про аналітику для кожного магазину. Поля таблиці:

- id (первинний ключ) — унікальний ідентифікатор аналітичного запису;
- store_id — ідентифікатор магазину (зв'язок із таблицею stores);
- action_id — ідентифікатор дії (зв'язок із таблицею analytics_actions);
- product_id — ідентифікатор продукту, на який стосується аналітика;

- platform_id — ідентифікатор платформи (зв'язок із таблицею platforms);

- timestamp — час створення запису.

Таблиця "analytics_actions" (дії аналітики). Таблиця відповідає за зберігання можливих дій користувачів. Поля таблиці:

- id (первинний ключ) — унікальний ідентифікатор дії;
- description — опис дії (наприклад, клік, перегляд).

Таблиця "platforms" (платформи). Ця таблиця зберігає інформацію про платформи, які взаємодіють із системою. Поля таблиці:

- id (первинний ключ) — унікальний ідентифікатор платформи;
- platform — назва платформи.

Таблиця "purchases" (покупки). Таблиця зберігає інформацію про здійснені покупки. Поля таблиці:

- id (первинний ключ) — унікальний ідентифікатор покупки;
- timestamp — час здійснення покупки;
- store_id — ідентифікатор магазину (зв'язок із таблицею stores).

Таблиця "purchase_products" (продукти покупки). Ця таблиця відображає зв'язок між покупками та продуктами. Поля таблиці:

- purchase_id — ідентифікатор покупки (зв'язок із таблицею purchases);
- product_id — ідентифікатор продукту.

Таблиця "terms" (пошукові терміни). Таблиця зберігає пошукові терміни користувачів. Поля таблиці:

- id (первинний ключ) — унікальний ідентифікатор терміну;
- store_id — ідентифікатор магазину (зв'язок із таблицею stores);
- term — пошуковий термін;
- platform_id — ідентифікатор платформи (зв'язок із таблицею platforms);

- timestamp — час пошуку.

Таблиця "user_store_access" (доступ користувачів до магазинів). Ця таблиця описує зв'язок між користувачами та магазинами. Поля таблиці:

- user_id — ідентифікатор користувача (зв'язок із таблицею users);
- store_id — ідентифікатор магазину (зв'язок із таблицею stores).

3.3.1 Даталогічна модель

Даталогічна модель даних бази даних описує структуру та організацію даних, а також взаємозв'язки між різними елементами даних. Ця модель також включає обмеження та правила, які визначають цілісність та поведінку даних, зокрема визначення первинних ключів, зовнішніх ключів, унікальних обмежень та інших правил для забезпечення точності й узгодженості даних [12].

Розглянемо даталогічну модель для нашої бази даних (рисунок 3.10).

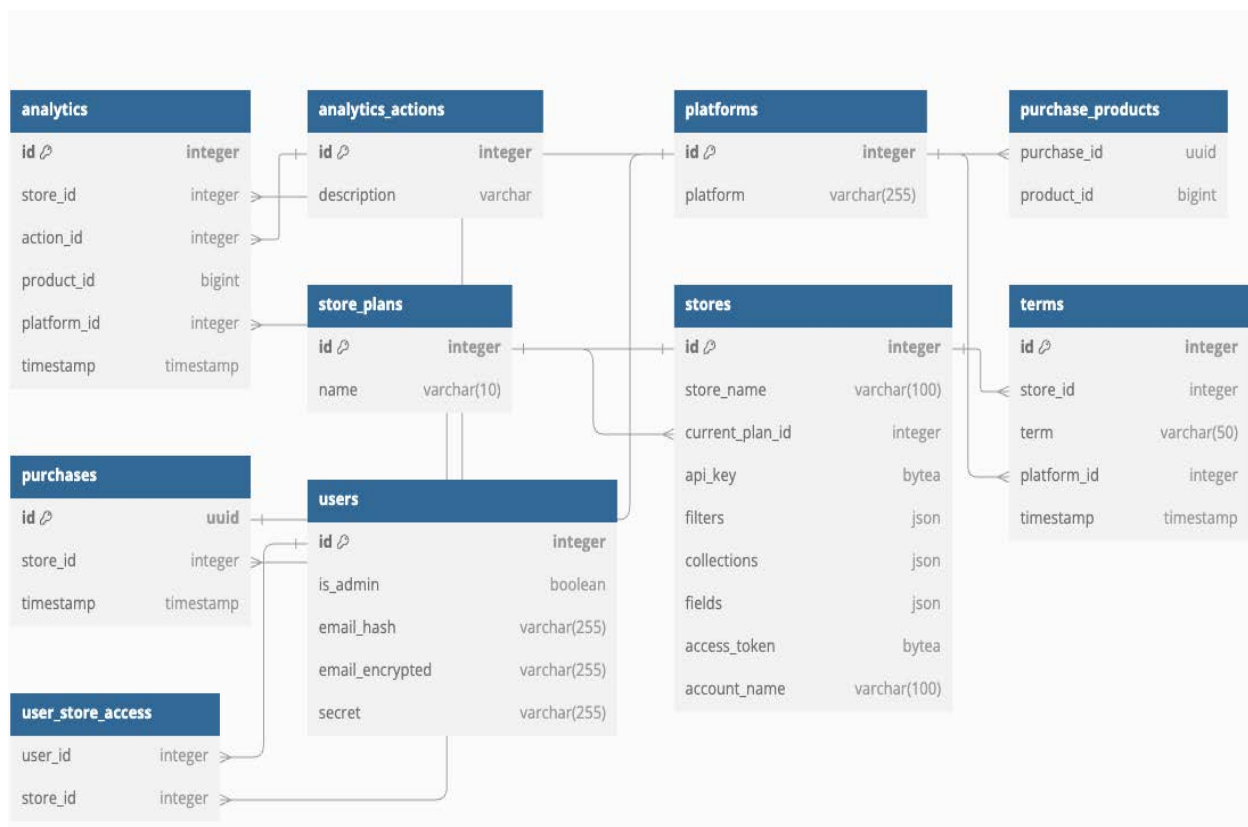


Рисунок 3.10 – Даталогічна модель бази даних

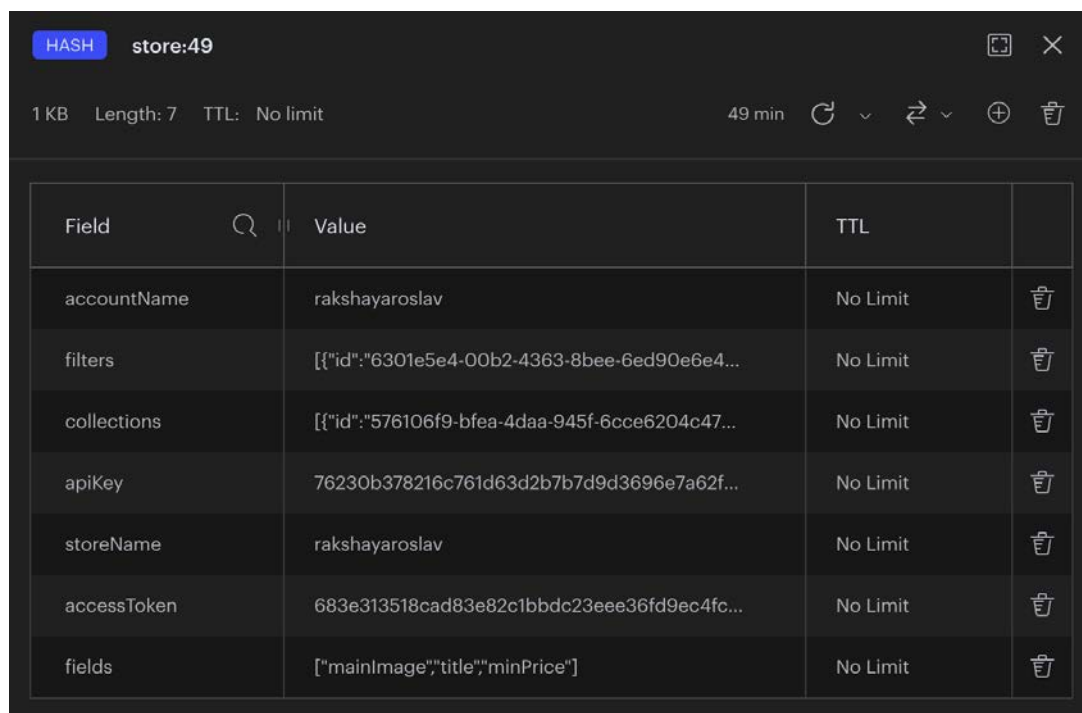
3.4 Опис роботи кешування даних

Як визначено раніше, для кешування даних ми використали Redis за допомогою інтерфейсу Redis Insight. Кешування дуже корисне для швидкого повернення даних. Тому було прийняте рішення використати кешування для зберігання сесій користувачів та налаштувань магазину. Розглянемо обидва випадки.

3.4.1 Кешування налаштувань для магазину

Кешування налаштувань для магазину грає дуже важливу роль у роботі застосунку. На сервер буде поступати дуже багато запитів щодо перегляду колекцій.

Наприклад, щоб переглянути список колекцій або каталог для колекції, ми повинні мати актуальну інформацію щодо навігації та ключу доступу для Shopify. Тому, така частота запитів негативно сприяла би на роботу нашої бази даних. Розглянемо структуру нашого кешування для магазину (рис. 3.11).



The screenshot shows the Redis Insight interface for a key named 'store:49'. The key is of type 'HASH' and has a size of 1 KB, a length of 7, and no TTL limit. The data is displayed in a table with four columns: Field, Value, TTL, and an edit icon. The table contains the following data:

Field	Value	TTL	
accountName	rakshayaroslav	No Limit	
filters	[{"id":"6301e5e4-00b2-4363-8bee-6ed90e6e4..."]	No Limit	
collections	[{"id":"576106f9-bfea-4daa-945f-6cce6204c47..."]	No Limit	
apiKey	76230b378216c761d63d2b7b7d9d3696e7a62f...	No Limit	
storeName	rakshayaroslav	No Limit	
accessToken	683e313518cad83e82c1bbdc23eee36fd9ec4fc...	No Limit	
fields	["mainImage","title","minPrice"]	No Limit	

Рисунок 3.11 – Кешування налаштувань для магазину

Структура кешування даних для магазину складається з таких полів.

AccountName. Зберігає назву облікового запису користувача, який пов'язаний із магазином.

Filters. Масив фільтрів, що використовуються для сортування чи відбору продуктів у каталозі магазину. Кожен об'єкт містить такі поля:

- id: унікальний ідентифікатор фільтра;
- title: назва фільтра, наприклад, "Brand";
- field: поле, до якого застосовується фільтр, наприклад, "vendor".

Collections. JSON-структура, яка зберігає інформацію про колекції магазину. Кожен об'єкт включає:

- id: унікальний ідентифікатор колекції;
- title: назва колекції, наприклад, "Men's";
- ref: посилання на ідентифікатор Shopify;
- children: масив дочірніх колекцій із такою самою структурою.

ApiKey. Унікальний ключ API, що використовується для доступу до Shopify. Він забезпечує автентифікацію запитів.

Fields. Масив полів, що визначає, які дані про продукти повертаються у запитах для відображення у мобільному застосунку. Наприклад:

- mainImage: основне зображення продукту;
- title: назва продукту;
- minPrice: мінімальна ціна продукту.

StoreName. Це поле яке зберігає назву інтернет-магазину для Shopify Api.

AccessToken. Ключ-токен для використання Shopify Api.

Схема прикладу використання кешування зображена на рисунку 3.12.

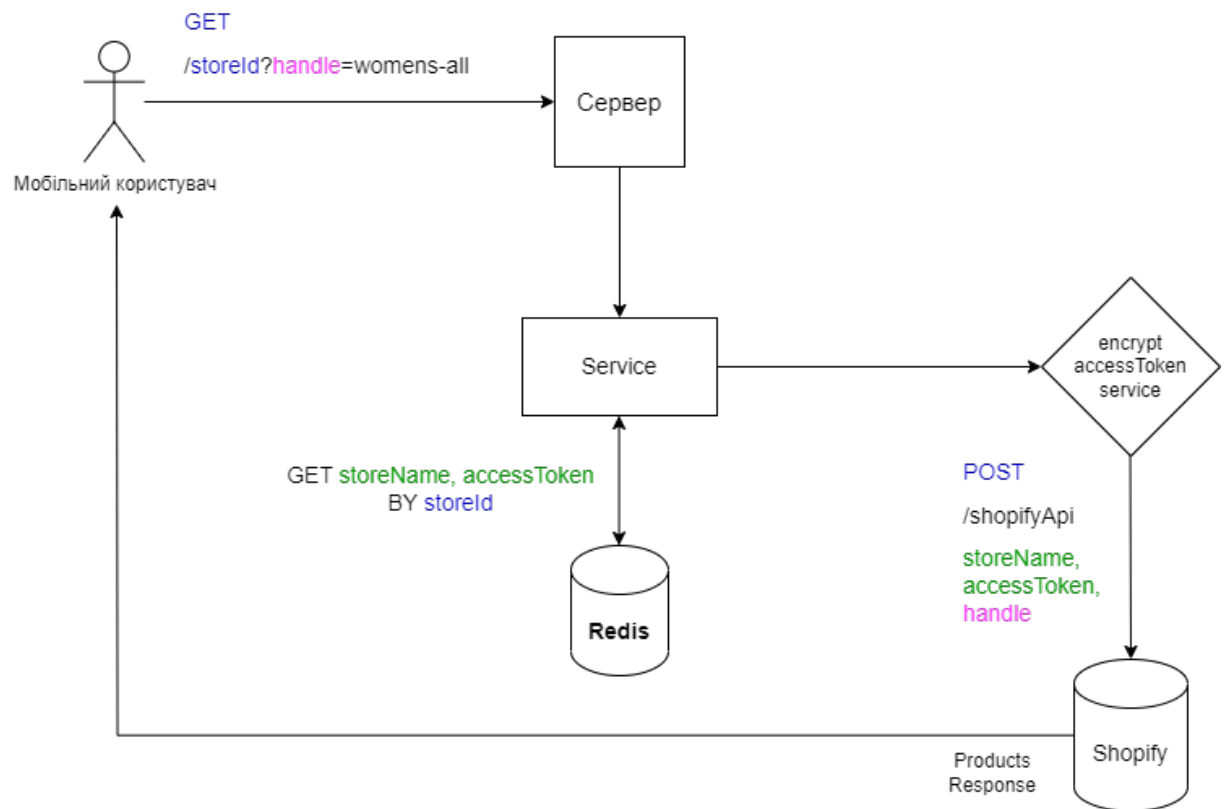


Рисунок 3.12 – Схема використання кешування

На рисунку зображено процес, у якому користувач надсилає запит на отримання інформації про продукти до серверу.

У запиті він зазначає унікальний ідентифікатор магазину та обрану колекцію для перегляду. Сервер, отримавши запит, обробляє його та викликає сервіс, відповідальний за надання інформації про продукти.

Цей сервіс отримує з кешу ключ і назву магазину за унікальним ідентифікатором. Далі ключ розшифровується, і виконується запит до Shopify із зазначенням необхідних полів.

Після успішної обробки запиту дані передаються клієнту.

3.5 Діаграма послідовності дій

Діаграма послідовності дій – це візуальне представлення, яке описує порядок виконання дій у системі. Вона демонструє, як окремі елементи системи взаємодіють між собою для досягнення конкретної мети чи виконання

завдання. За допомогою діаграми послідовності дій розробники отримують детальний огляд логіки процесів, що допомагає у проектуванні, аналізі та реалізації функціоналу програмного забезпечення (рис. 3.13).

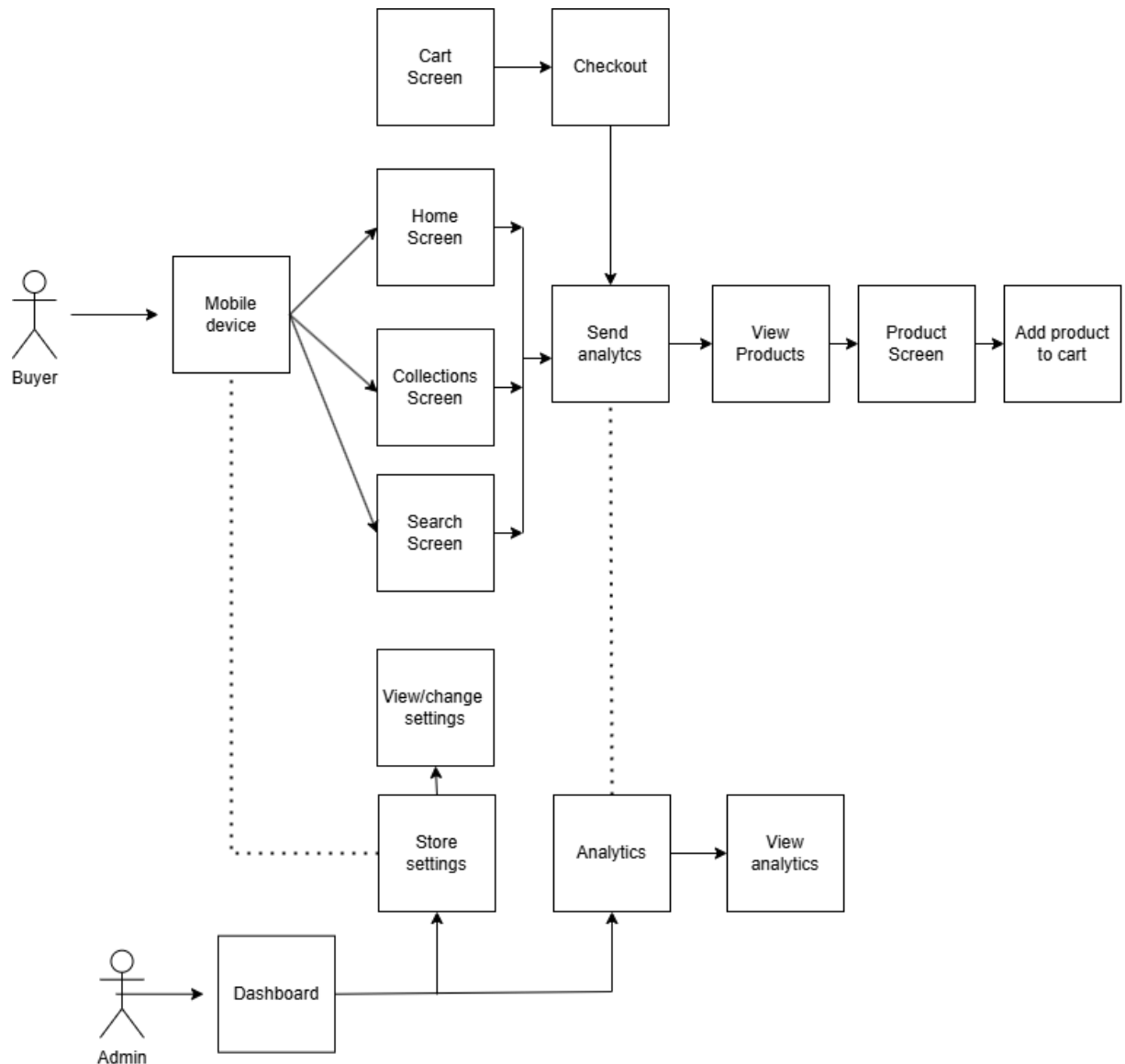


Рисунок 3.13 – Діаграма послідовності дій

3.5 Архітектура системи

Архітектура системи (рис. 3.14), також відома як архітектурна схема або структура системи, є візуальним представленням ключових компонентів, підсистем та взаємодій у рамках програмної або технічної системи. Вона надає чіткий і структурований огляд того, як різні модулі, сервіси або процеси

взаємодіють один з одним для досягнення визначених цілей або функціональних результатів.

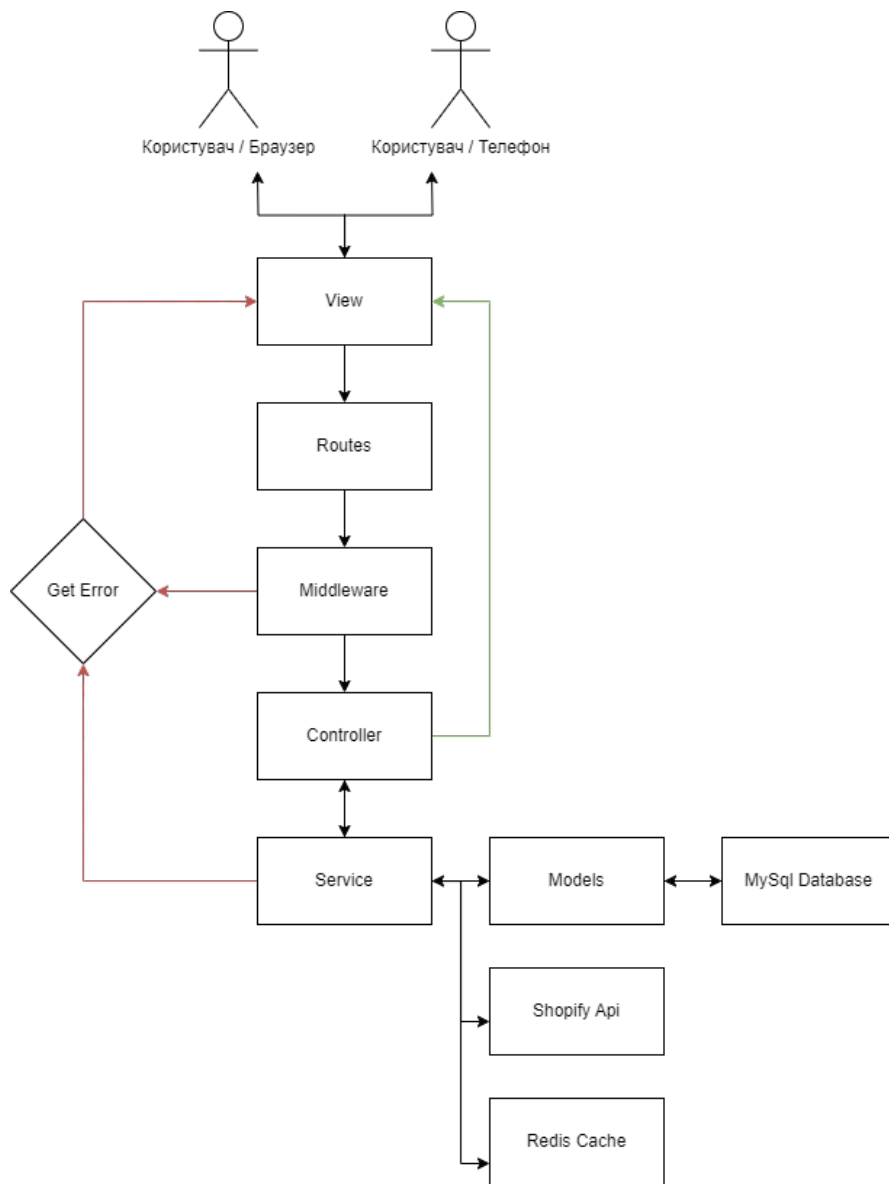


Рисунок 3.14 – Архітектура системи

На схемі можна побачити двох користувачів, які надсилають запити до нашого серверу. Дані надходять у HTTP форматі [13].

Після, запит передається до модуля маршрутизації. Перед тим як обробляти запит, сервер перевіряє інформацію від користувача у модулі проміжного програмного забезпечення. У випадку, якщо інформації недостатньо, або вона містить помилку, сервер відповідає помилкою.

Якщо все добре, дані з запиту переходять до модулю “controller”. Даний модуль викликає потрібний сервіс та повертає відповідь до клієнту.

Модуль сервіс містить складний функціонал. Це може бути операції з базами даних, сторонніми бібліотеками, запис кешу тощо. Якщо сталася помилка при операції, то сервіс може повернути також повернути помилку.

3.6 Висновок за розділом 3

У цьому розділі було розглянуто розробку клієнтської частини для браузера та мобільного застосунку, а також описано основні компоненти та структуру проекту. Важливу роль у клієнтській частині відіграють допоміжні бібліотеки, які спрощують реалізацію таких функцій, як виконання HTTP-запитів, побудова графіків тощо. Взаємодія між компонентами реалізована через API-запити, що забезпечує ефективну роботу як з даними клієнтів, так і з адміністративними функціями через відповідні API. Також було детально описано структуру бази даних, її таблиці та взаємозв'язки, що дозволяють зберігати інформацію про користувачів, магазини, покупки та інші важливі аспекти системи. Кешування даних за допомогою Redis дозволяє значно підвищити ефективність обробки запитів та швидкість роботи системи. У результаті, розроблена клієнтська частина для браузера та мобільного застосунку забезпечує надійну і зручну взаємодію користувача з системою, а також оптимізує роботу з даними завдяки використанню сучасних технологій і підходів.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ

4.1. Функціонал клієнтської частини для браузера

На головній сторінці можна побачити форму для заповнення даних. Вона складається з поля електронної пошти та кнопки входу (рис. 4.1).

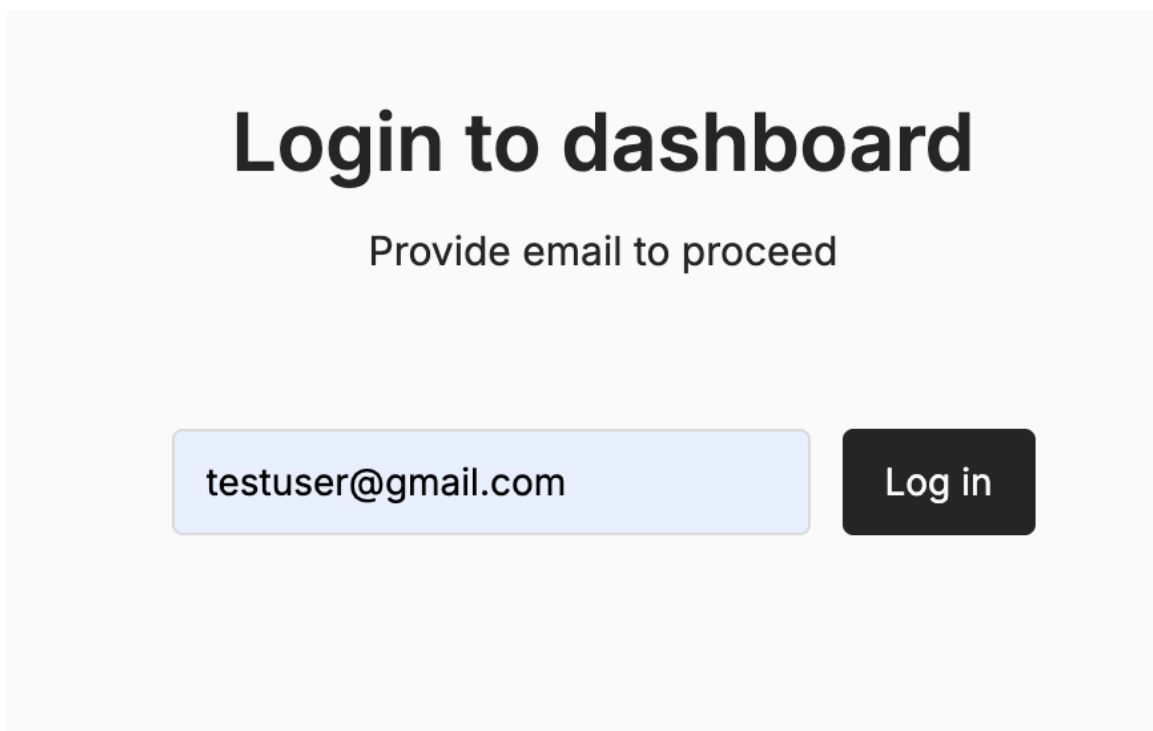
The image shows a login form on a light gray background. At the top, the text "Login to dashboard" is displayed in a large, bold, black font. Below it, the instruction "Provide email to proceed" is written in a smaller, regular black font. There are two input fields: a light blue rounded rectangle containing the text "testuser@gmail.com" and a dark gray rounded rectangle containing the text "Log in".

Рисунок 4.1 – Сторінка логіну застосунку у браузері

Також було розроблено обробник помилок. Наприклад, якщо відправити пусті поля через форму, то можна буде побачити “alert” з помилкою (рис. 4.2). Також, у консолі розробника бачимо коректну відповідь серверу.

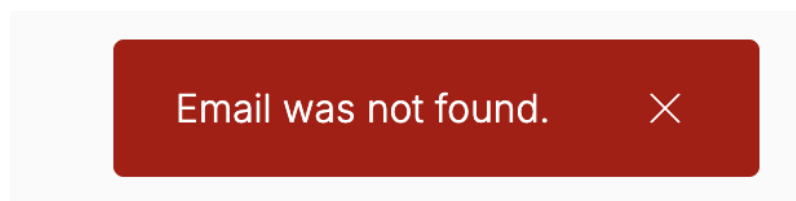
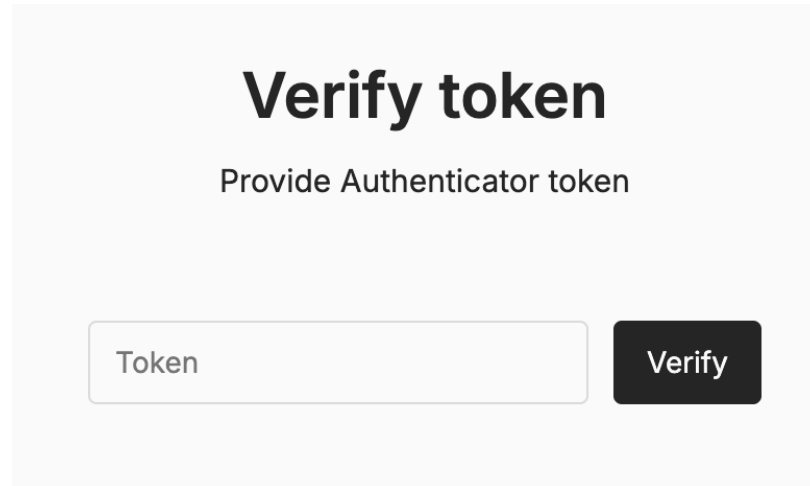


Рисунок 4.2 – Обробка помилки з клієнтської сторони

Якщо форма була заповнена правильною інформацією, то ми переходимо до наступної форми. Вона складається з одного поля та кнопки автентифікації (рис. 4.3).



The image shows a web form titled "Verify token" with the subtitle "Provide Authenticator token". It features a text input field labeled "Token" and a dark button labeled "Verify".

Рисунок 4.3 – Форма автентифікації

Для проходження автентифікації необхідно відкрити мобільний додаток Google Authenticator. Після успішного сканування в додатку з'явиться одноразовий токен для доступу, який оновлюється кожні 30 секунд (рис. 4.4).

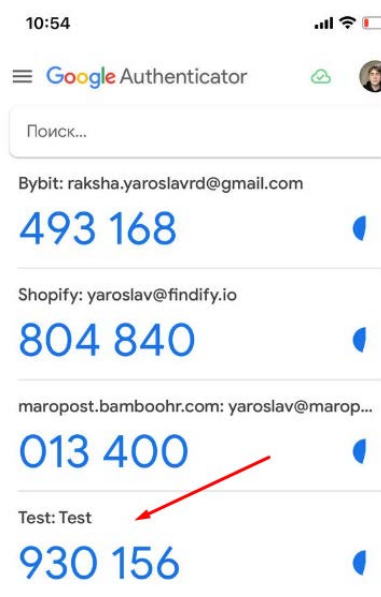


Рисунок 4.4 –Приклад перегляду токена в застосунку Authenticator

Якщо токен був вказаний правильно та співпадав з токеном, що знаходився у мобільному додатку, то ми переходимо до сторінки з персональним кабінетом (рис. 4.5). Вона складається з трьох частин.

Зліва ми можемо побачити навігацію по доступним сторінкам. Нам доступна роздільна аналітика та налаштування нашого магазину.

По центру ми можемо побачити сам контент для нашої сторінки. У цьому випадку, це повна аналітика.

Справа зверху ми можемо побачити вибраний магазин. Назва магазину прилежна до Shopify. У спадному списку ми можемо обрати любий інший магазин. При виборі магазину, контент сторінки оновлюється.

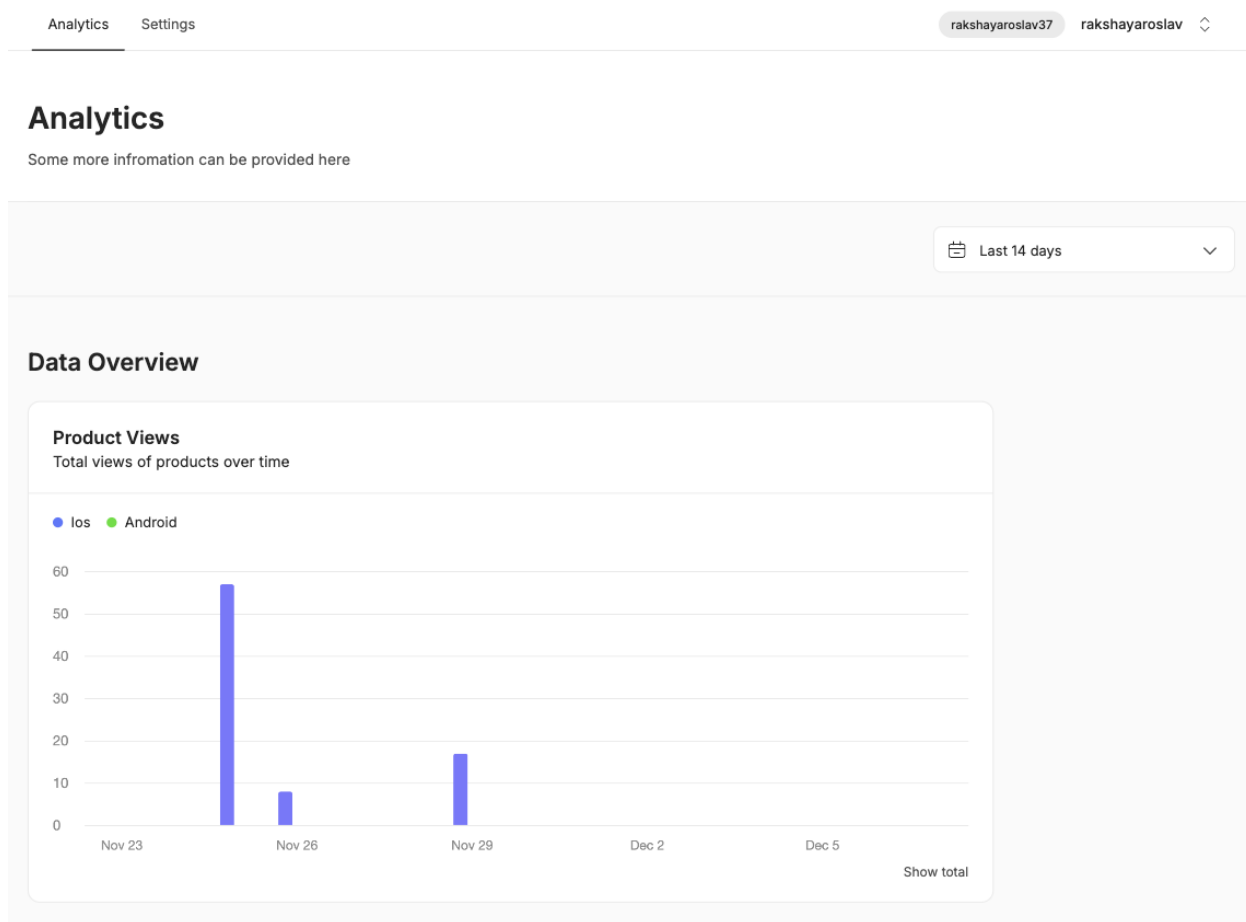


Рисунок 4.5 – Персональний кабінет

Перейдемо до перегляду налаштувань навігації нашого застосунку (рис. 4.6). Тут ми можемо побачити список з назвою сторінок для нашого

мобільного застосунку. Через персональний кабінет ми можемо змінювати навігацію у реальному часі.

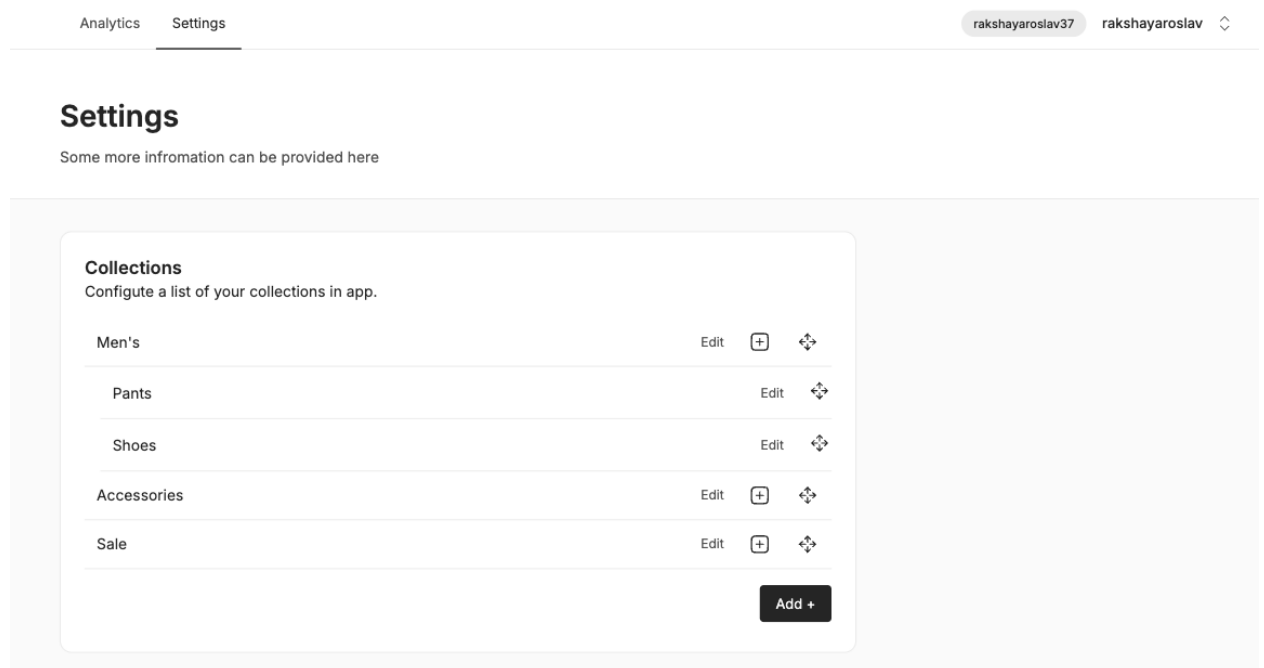
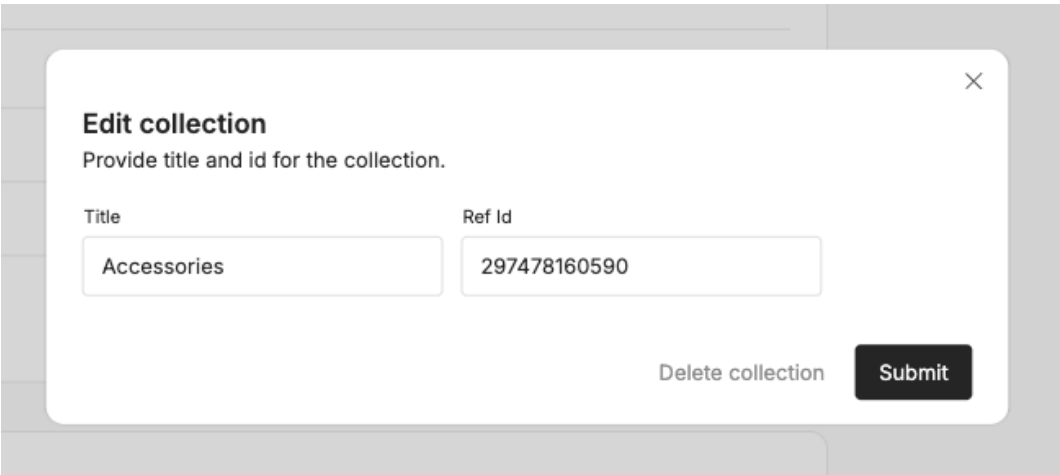


Рисунок 4.6 – Перегляд навігації у магазині

Розглянемо приклад роботи з налаштуваннями навігації (рис. 4.7). При натисканні кнопки "Edit" відкривається вікно з формою для редагування параметрів навігації. Форма містить два текстових поля для вводу та кнопку для підтвердження змін. Поля для вводу дозволяють зазначити назву та ключ колекції.

Назва використовується для відображення у мобільному додатку, тоді як ключ колекції можна отримати з панелі управління Shopify. Цей ключ відповідає конкретній колекції продуктів. Під час доступу до колекції через мобільний додаток здійснюється запит до сервера Shopify із зазначеним ключем для отримання відповідних даних.



Edit collection

Provide title and id for the collection.

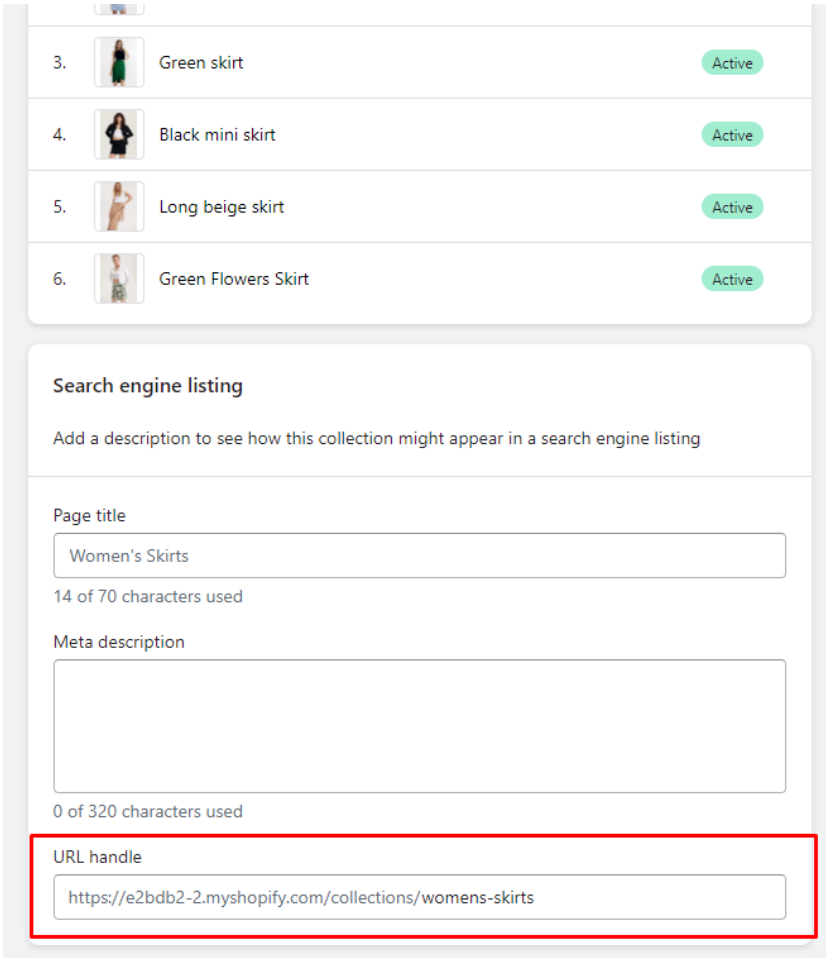
Title: Accessories

Ref Id: 297478160590

Delete collection Submit

Рисунок 4.7 – Налаштування навігації у магазині

Раніше, ми створили колекцію у Shopify та додали до неї продукти (рис. 4.8). Ми можемо побачити ключ у самому низу сторінки. По цьому ключу ми будемо робити запит та відображати продукти для цієї колекції.



3. Green skirt Active

4. Black mini skirt Active

5. Long beige skirt Active

6. Green Flowers Skirt Active

Search engine listing

Add a description to see how this collection might appear in a search engine listing

Page title: Women's Skirts
14 of 70 characters used

Meta description: 0 of 320 characters used

URL handle: https://e2bdb2-2.myshopify.com/collections/womens-skirts

Рисунок 4.8 – Ключ до колекції у Shopify

4.2 Функціонал клієнтської частини для мобільного застосунку

У мобільному застосунку у нас доступні такі екрани: головна сторінка, список колекцій, сторінка пошуку та кошик. Розглянемо поетапно кожен зі сторінок. Екран зі списком колекцій зображено на рисунку 4.9.

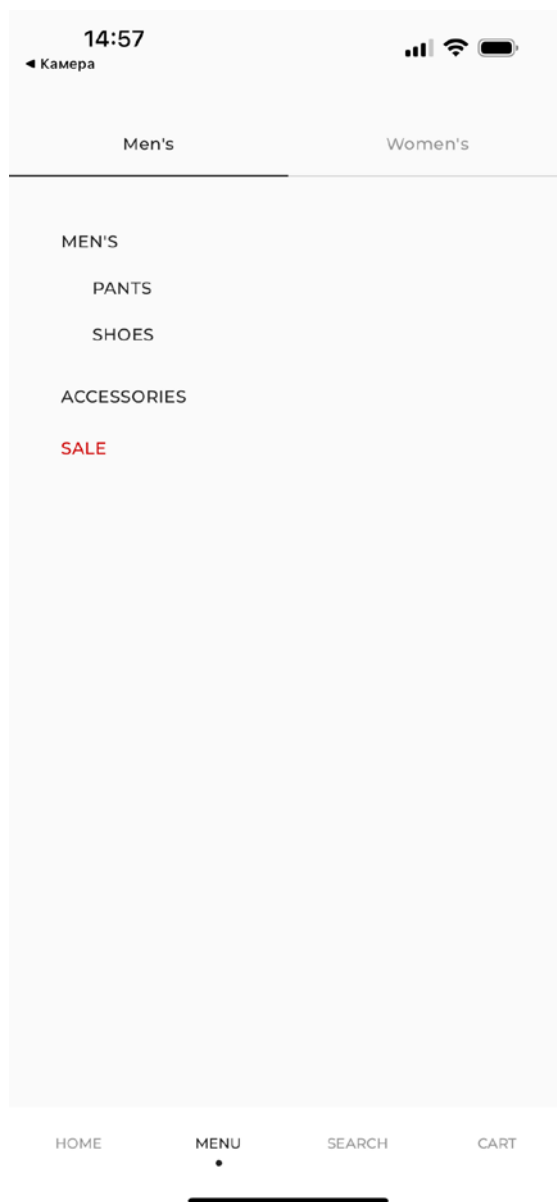


Рисунок 4.9 – Екран зі списком колекцій у магазині

Перший екран відображає список колекцій, який формується на основі налаштувань навігації магазину. Назви колекцій повністю узгоджуються з

тими, що зазначені у персональному кабінеті. Крім того, основні колекції можуть містити вкладені підколекції, створюючи ієрархічну структуру.

Під час вибору колекції відбувається перехід на екран із каталогом продуктів, які до неї належать (рис. 4.10). Сторінка каталогу відображає список товарів із такими деталями, як зображення, назва, ціна та валюта.

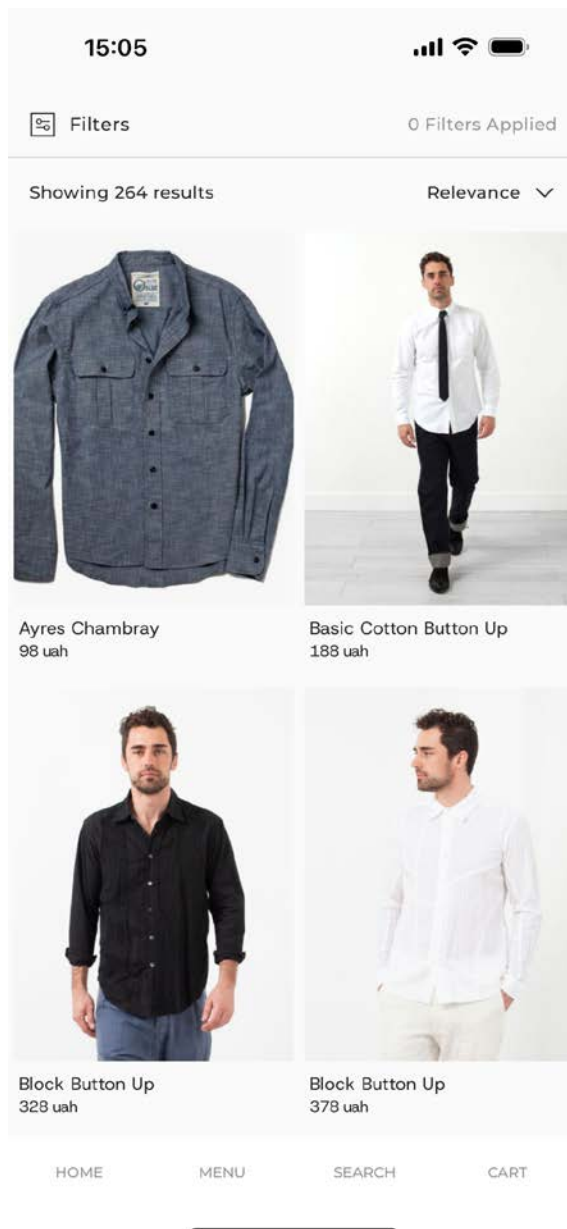


Рисунок 4.10 – Екран каталогу для колекції

Під час перегляду цієї сторінки надсилається запит на отримання аналітики щодо перегляд, якщо сторінка була відображена з сторінку пошуку.

Дані про товари повністю відповідають інформації, наданій у панелі Shopify та інформація береться з Elastic.

Натиснувши на зображення товару, користувач переходить на сторінку з детальною інформацією про нього (рис. 4.11). Під час цього переходу також здійснюється запит на запис аналітики про перегляд продукту.



Рисунок 4.11 – Екран показу продукту

Цей екран містить зображення товару, його основну інформацію та кнопку додавання до кошика. Після натискання на кнопку відображається повідомлення про успішне додавання товару. Щоб переглянути обраний

продукт у кошику та продовжити покупки, потрібно перейти на екран кошика (рис. 4.12).

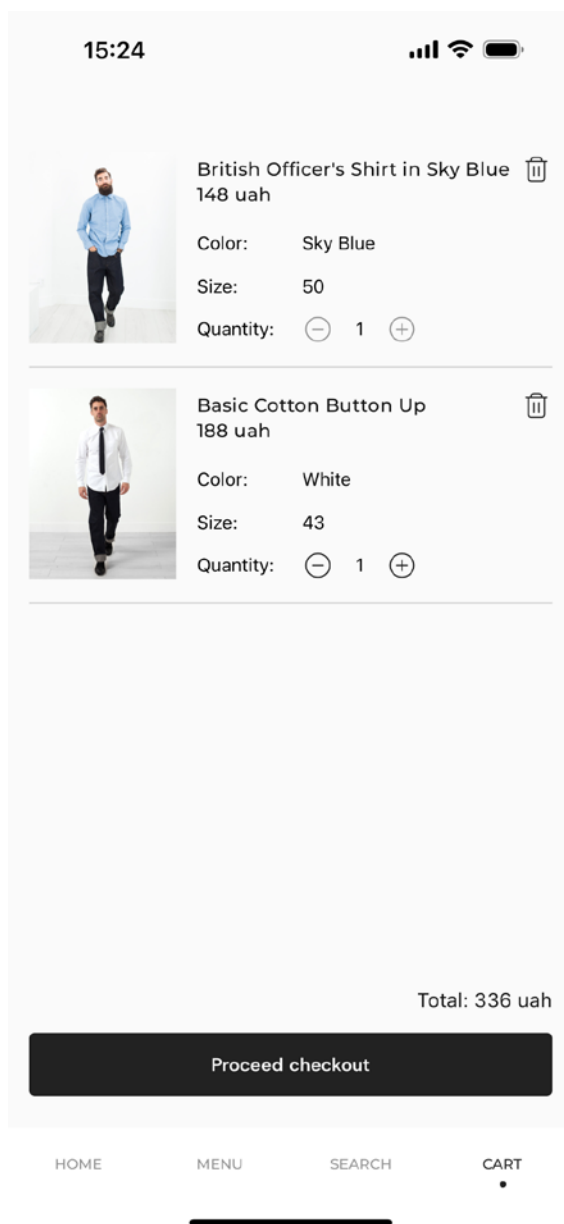


Рисунок 4.12 – Екран кошику

Якщо кошик порожній, на екрані відображається відповідне повідомлення. У разі наявності товарів кошик містить такі елементи:

- список продуктів доданих до кошика;
- загальна сума товарів у кошику;
- кнопка переходу до сплати.

Користувач може змінювати кількість товарів або видаляти їх із кошика. При цьому кількість обраних одиниць не може перевищувати доступний залишок товару, визначений у Shopify.

Натиснувши кнопку "Перейти до сплати", користувач переходить на екран оформлення замовлення.

4.3 Висновок за розділом 4

У цьому розділі детально розглянуто функціонал клієнтської частини програми для браузера та мобільного застосунку. Описано, як користувач взаємодіє з інтерфейсом на різних етапах використання. Визначено основні елементи інтерфейсу, такі як налаштування навігації, список колекцій, каталог продуктів та інші. Також описано процес обробки помилок та систему тестування взаємодій між клієнтською частиною та сервером, що дозволяє забезпечити коректне відображення даних. Використання мобільного додатка, інтеграція автентифікації, а також динамічні зміни навігації магазину забезпечують зручність і високий рівень безпеки при використанні системи.

ВИСНОВКИ

У підсумку, дана кваліфікаційна дипломна робота магістра була зосереджений на створенні комплексного рішення для інтеграції магазину Shopify з мобільним застосунком. Завдяки систематичному використанню Shopify API забезпечено повну синхронізацію між мобільним застосунком, персональним кабінетом користувача та платформою Shopify, що дозволяє здійснювати актуалізацію даних про каталоги й продукти в реальному часі. Розроблений персональний кабінет надає власникам магазинів централізовані засоби управління налаштуваннями мобільного додатка, а також функціонал для отримання аналітичних даних, що дає змогу відстежувати поведінкові метрики користувачів.

Мобільний застосунок оснащений інтуїтивно зрозумілим інтерфейсом, оптимізованим для роботи на смартфонах та планшетах. Він забезпечує користувачів доступом до якісних зображень продуктів, детальних описів і актуальної інформації про ціни, що сприяє прийняттю обґрунтованих рішень щодо покупок. Загалом, розроблена екосистема створює комплексний і клієнто-орієнтований досвід взаємодії з електронною комерцією. Інтеграція з платформою Shopify, персоналізовані функції та орієнтація на мобільну торгівлю дозволяють підприємствам не лише покращити комунікацію зі споживачами, але й сприяти збільшенню продажів і формуванню стійкої брендової присутності в середовищі мобільних технологій.

Наукова новизна роботи полягає у тому, що отримав подальший розвиток підхід швидкої розробки, який дозволяє прискорити, а також спростити, розробку та модифікацію мобільного застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Tapcart [Electronic resource]. – Access mode: <https://www.tapcart.com>.
2. Shopney [Electronic resource]. – Access mode: <https://shopney.co>.
3. App My Site [Electronic resource]. – Access mode: <https://www.appmysite.com>.
4. Myers, M. A Smarter Way to Learn JavaScript / M. Myers. – CreateSpace Independent Publishing Platform, 2014. – 254 с.
5. Express.js [Electronic resource]. – Access mode: <https://expressjs.com>.
6. MySQL Documentation [Electronic resource]. – Access mode: <https://dev.mysql.com/doc>.
7. Балик, Н. Р. Бази даних MySQL / Н. Р. Балик, В. І. Мандзюк. – Тернопіль: Навчальна книга – Богдан, 2010. – 160 с.
8. Next.js [Electronic resource]. – Access mode: <https://nextjs.org>.
9. Postman Responses [Electronic resource]. – Access mode: <https://learning.postman.com/docs/sending-requests/responses>.
10. Redis Documentation [Electronic resource]. – Access mode: <https://redis.io/docs/>.
11. Lim, G. Beginning Node.js, Express & MongoDB Development / G. Lim. – Independently published, 2019. – 154 с.
12. Доценко, С. І. Організація та системи керування БД / С. І. Доценко. – Харків: УкрДУЗТ, 2023. – 118 с.
13. HTTP Methods [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>.

ДОДАТОК А
Текст програми

А.1 Репозиторій з текстом програми

Вся програма збережена на ресурсі GitHub за посиланням <https://github.com/remido33/dyploma>, або можна відсканувати QR-код, який зображено на рисунку А.1.



Рисунок А.1 – QR код на проєкт у GitHub

Приклад запуску сервера для мобільного застосунку наведено нижче.

```
const express = require('express');  
const getError = require("../shared_utils/getError");  
const app = express();  
  
require('dotenv').config();  
  
app.use(express.json());  
app.use('/store', require('./routes/store'));  
app.use('/analytics', require('./routes/analytics'));
```

```
app.all('*', (req, res) =>
  res.status(404).json(getError(404, 'Resource not found')));

app.use(require('../../shared_utils/errorHandler'));

module.exports = app;
```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Дипломна кваліфікаційна робота магістра
на тему: Дослідження та програмна реалізація екосистеми для
швидкої розробки мобільних застосунків

Виконав: студент групи КНТ-123м

Ярослав РАКША

Керівник: к.т.н.

Ольга ГЛАДКОВА

Рисунок Б.1 - Слайд 1

Об'єкт дослідження – екосистеми та системи програмних рішень для розробки мобільних застосунків.

Предмет дослідження – комп'ютерні програми та технології для розробки онлайн-магазинів.

Мета роботи – створення програмної системи для швидкої та ефективної розробки мобільних застосунків для онлайн-торгівлі за допомогою Shopify та Elasticsearch.

Актуальність проблеми. У сучасному світі технологічний прогрес і стрімкий розвиток цифрової економіки створюють умови для зростаючого попиту на інноваційні рішення в галузі розробки програмного забезпечення. Особливої актуальності набувають технології, орієнтовані на швидке створення продуктів, що дозволяють компаніям адаптуватися до динамічних змін ринку та залишатися конкурентоспроможними.

Рисунок Б.1 - Слайд 2

Завдання роботи

- провести аналіз існуючих підходів до розробки мобільних додатків з повною кастомізацією дизайну та функціоналу;
- визначити ключові технології та фреймворки, що оптимізують розробку;
- оцінити ефективність використання API для інтеграції з Shopify та іншими системами на прикладі конкретних запитів;
- реалізувати компоненти серверної частини, які підтримують високу швидкість обробки запитів та інтеграцію з базами даних;
- провести тестування на різних етапах розробки, зокрема тестування ефективності обробки запитів через маршрути;
- реалізувати клієнтську частину;
- виконати тестування розробленої екосистеми.

Рисунок Б.3 - Слайд 3

Обрані технології для розробки

Технологія	Опис	Переваги
Express.js	Фреймворк для Node.js, який спрощує створення серверних додатків	Простота маршрутизації, REST API, масштабованість, middleware
MySQL	Реляційна база даних, використовується для зберігання даних	Надійність, підтримка ACID, оптимізація запитів
Redis	Система кешування даних у пам'яті	Швидкість доступу, підтримка різноманітних структур даних
Next.js	Фреймворк для створення динамічних веб-додатків	Автоматична маршрутизація, підтримка SSR, API маршрути
React.js	Бібліотека для створення користувацьких інтерфейсів	Компонентна архітектура, віртуальний DOM, висока швидкість рендерингу
React Native	Фреймворк для кросплатформених мобільних додатків	Кросплатформність, швидкість розробки, гаряче перезавантаження
TypeScript	Мова програмування, яка додає типізацію до JavaScript	Типізація, сумісність з JavaScript, масштабованість

Рисунок Б.4 - Слайд 4

Структура серверної частини

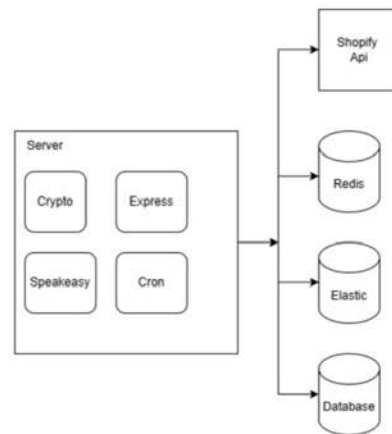


Рисунок Б.5 - Слайд 5

Приклад запиту

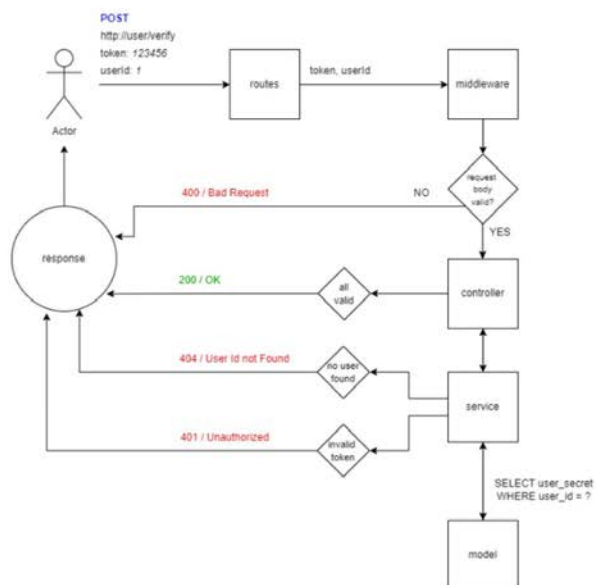


Рисунок Б.6 - Слайд 6

База даних

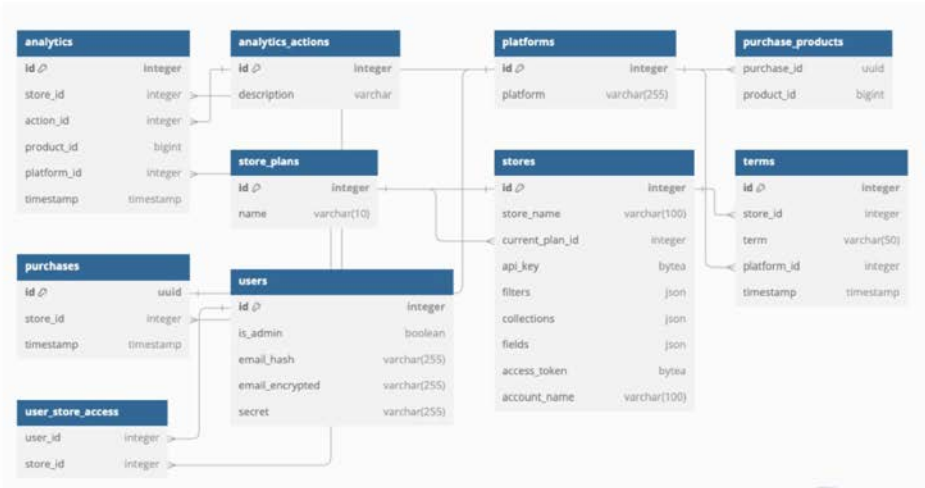


Рисунок Б.7 - Слайд 7

Інтерфейс мобільного застосунку

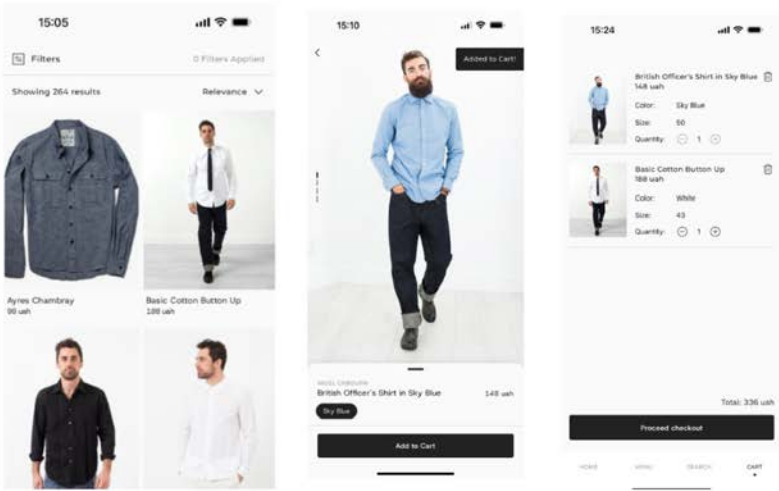


Рисунок Б.8 - Слайд 8

Інтерфейс адміністративної панелі

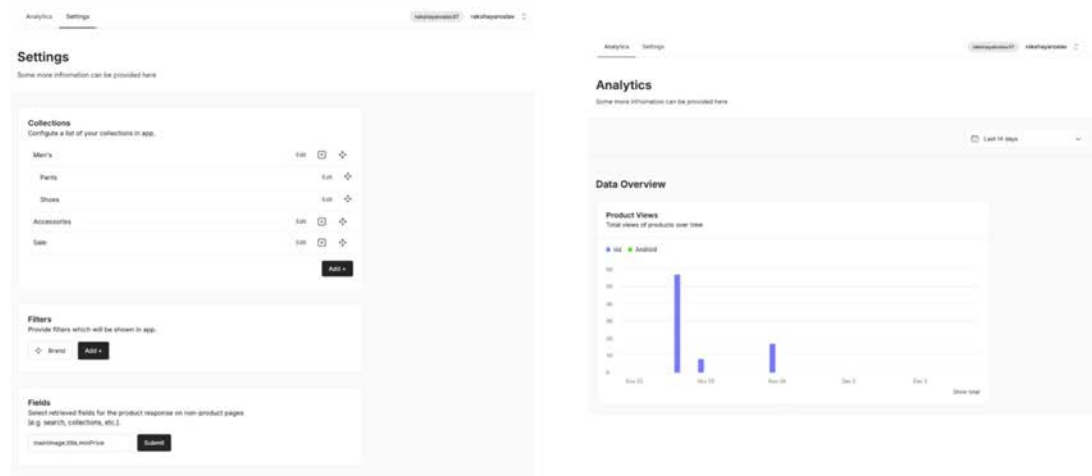


Рисунок Б.9 - Слайд 9

Висновки

Практична значущість роботи полягає у тому, що було створено систему комплексних рішень для інтеграції магазину Shopify з мобільним застосунком. Розроблений персональний кабінет надає власникам магазинів централізовані засоби управління налаштуваннями мобільного застосунку, а також функціонал для отримання аналітичних даних, що дає змогу відстежувати поведінкові метрики користувачів.

Мобільний застосунок оснащено інтуїтивно зрозумілим інтерфейсом, оптимізованим для роботи на смартфонах та планшетах. Він забезпечує користувачів доступом до якісних зображень продуктів, детальних описів і актуальної інформації про ціни, що сприяє прийняттю обґрунтованих рішень щодо покупок. Загалом, розроблена екосистема створює комплексний і клієнтоорієнтований досвід взаємодії з електронною комерцією.

Наукова новизна роботи полягає у тому, що отримав подальший розвиток підхід швидкої розробки, який дозволяє прискорити, а також спростити, розробку та модифікацію мобільного застосунку.

Рисунок Б.10 - Слайд 10