

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ВЕБЗАСТОСУНОК ЗБОРУ АНАЛІТИКИ ЗАМОВЛЕНЬ
ІНТЕРНЕТ-МАГАЗИНУ НА ПЛАТФОРМІ SHOPIFY
WEB APPLICATION FOR COLLECTING ONLINE STORE
ORDER ANALYTICS ON THE SHOPIFY PLATFORM

Виконав: студент(ка) 4 курсу, групи КНТ-219сп

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Косенко Є.А.

(прізвище та ініціали)

Керівник Колпакова Т.О.

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

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

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2022 року

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

Косенко Єлизавети Андріївни
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Вебзастосунок збору аналітики замовлень
Інтернет-магазину на платформі Shopify. Web Application for Collecting Online
Store Order Analytics on the Shopify Platform.

керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 07 червня 2022 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. Аналіз та вибір засобів розробки програмного продукту. 3. Основні рішення щодо реалізації системи.
4. Керівництво програміста. 5. Керівництво користувача.

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
1-5 Основна частина	Колпакова Т.О., доцент		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання “04” квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис) Косенко Є.А.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Колпакова Т.О.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
97 с., 33 рис., 1 табл., 3 дод., 14 джерел.

E-COMMERCE, SHOPIFY, СТАТИСТИКА, ЗАМОВЛЕННЯ,
АДМІНІСТРАТИВНА ПАНЕЛЬ, ВЕБЗАСТОСУНОК, GRAPHQL, REST.

Об'єкт дослідження – процес накопичення замовлень від покупців в інтернет-магазині.

Предмет дослідження – програмна реалізація застосунку збору аналітики замовлень інтернет-магазину.

Мета роботи – розробка програмного забезпечення, що надає можливість відслідковувати аналітику та історію замовлень інтернет-магазину на базі платформи Shopify.

Матеріали, методи та технічні засоби: мова програмування JavaScript, GraphQL API та REST API для обміну даними між клієнтом та сервером.

Результати. Створено вебзастосунок, який допомагає збирати та відстежувати аналітику замовлень інтернет-магазину на платформі Shopify.

Висновки. Визначено основні завдання для виконання дипломної роботи, розроблено вебзастосунок, який надає можливість отримання аналітики інтернет-магазину на платформі Shopify.

Галузь використання. Подібну програму можна використовувати всім власникам або менеджерам інтернет-магазинів, побудованих на платформі Shopify.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 97 pages, 33 figures, 1 table, 3 appendixes, 14 sources.

E-COMMERCE, SHOPIFY, STATISCTIC, ORDERS, ADMIN PANEL, WEB-APPLICATION, GRAPHQL, REST.

The object of research is the process of adding orders from customers to e-commerce store.

The subject of research is the program implementation of the application of statistic of orders of online store.

The purpose of the work is the development of software that allows you to track statistics and order history of the online store based on the Shopify platform.

Materials, methods and technical means: JavaScript programming language for data exchange between client and server GraphQL API and REST API.

Results. A web application has been created to help collect and track online store order analytics on the Shopify platform.

Conclusions. The main tasks for the thesis have been identified, a web application has been developed, which provides an opportunity to obtain online store analytics on the Shopify platform.

The field of usage. This program can be used by all owners or managers of online stores built on the Shopify platform.

ЗМІСТ

	С.
Перелік скорочень та умовних познач 9	9
Вступ 10	10
1 Аналіз предметної області 11	11
1.1 Відслідковування аналітики замовлень засобами платформи Shopify 12	12
1.2 Аналіз існуючих програмних засобів для відстеження аналітики інтернет-магазину Shopify 15	15
1.2.1 Profit Analytics & Benchmark 15	15
1.2.1 Glew Analytics 16	16
1.2.3 Connector for Google Analytics..... 17	17
1.2.4 StoreView 18	18
1.3 Постановка завдання до роботи 20	20
1.4 Висновки за розділом 1 20	20
2 Аналіз та вибір засобів розробки програмного продукту 21	21
2.1 Принцип роботи клієнт-серверної архітектури застосунку 21	21
2.1.1 Архітектура передачі даних між клієнтом та сервером 22	22
2.1.2 Архітектура програмного продукту 22	22
2.2 Шаблон проєктування..... 24	24
2.3 Вибір мови програмування..... 25	25
2.4 Вибір JavaScript-фреймворків 28	28
2.5 Висновки за розділом 2 29	29
3 Основні рішення щодо реалізації системи..... 31	31
3.1 Розробка схеми отримання даних 31	31
3.2 Розробка вебінтерфейсу 34	34

3.3	Опис реалізованих функцій	36
3.3.1	Функції авторизації до власного інтернет-магазину	36
3.3.2	Функція отримання замовлень	37
3.3.3	Функції сортування, пошуку, фільтрації та пагінації	41
3.4	Висновки за розділом 3	45
4	Керівництво програміста	46
4.1	Призначення програми	46
4.2	Характеристики програми	46
4.3	Вимоги до інформативних структур	46
4.4	Звернення до програми	47
4.5	Вхідні та вихідні дані	47
4.5.1	Вхідні дані	47
4.5.2	Вихідні дані	47
4.6	Розгортання вебзастосунку на Shopify Platform	47
4.7	Висновки за розділом 4	49
5	Керівництво користувача	50
5.1	Призначення та умови застосування програми	50
5.1.1	Призначення програми	50
5.1.2	Умови виконання програми	50
5.2	Характеристики програми	51
5.3	Виконання програми	51
5.4	Структура програми	52
5.4.1	Сторінка авторизації	52
5.4.2	Сторінка “Home”	52
5.4.3	Сторінка “Orders”	54

5.5 Сторінка замовлення	55
5.6 Повідомлення користувачу.....	55
5.7 Висновки за розділом 5	56
Висновки.....	57
Перелік джерел посилання	58
Додаток А Технічне завдання.....	60
Додаток Б Текст програми.....	65
Додаток В Слайди презентації	91

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CLI	– Command Line Interface;
CSS	– Cascading Style Sheets;
HTML	– Hypertext Markup Language;
HTTP	– HyperText Transfer Protocol;
JSON	– JavaScript Object Notation, запис об'єктів JavaScript;
POS	– Point of sale;
PWA	– Progressive Web Application;
REST	– Representational State Transfer;
SSG	– Static Site Generator;
SSR	– Server Side Rendering;
URL	– Uniform Resource Locator.

ВСТУП

Сучасний світ та його діяльність з появою комп'ютеризованих систем спрямовано почали рухатись у бік віртуалізації та автоматизації. Багато сфер життя транслюються не тільки у реальному житті, а також у комп'ютерних застосунках та гаджетах. Торгівля, медицина, фінансова діяльність та інші сфери доступні у вільному використанні в Інтернеті та мобільних застосунках.

Дуже популярною автоматизованою системою продажу товарів під назвою "E-commerce" користується майже кожен житель нашої планети. Напевно, вже не зустріти людину, котра не робила б хоч раз замовлення в інтернет-магазинах.

Зважаючи на потік користувачів, які користуються інтернет-магазинами, засновникам, адміністраторам та продавцям товарів в інтернеті необхідно відстежувати аналітику замовлень та продажу товарів у своїх інтернет-магазинах з метою запобігання помилок у подальшій обробці замовлень та керуванням асортиментом товару.

Задачі, які було виконано в рамках даної дипломної роботи:

- аналіз існуючого програмного забезпечення;
- аналіз та вибір засобів розробки програмного продукту;
- реалізація системи;
- керівництво програміста;
- керівництво користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Науково-технічний прогрес досяг визначного розвитку у багатьох сферах життєвої діяльності суспільства. Однією з найбільш розвинених сфер в цифровому суспільстві є електронна комерція.

Електронна комерція (англ. “E-commerce”) – це сфера цифрової економіки, що включає всі фінансові та торгові транзакції, які проводяться за допомогою комп'ютерних мереж, та бізнес-процеси, пов'язані з проведенням цих транзакцій [1].

До електронної комерції відносять:

- електронний обмін інформацією – автоматизована, розрахована на багато користувачів система, що супроводжує процес управління роботою ієрархічної організації з метою забезпечення виконання цією організацією своїх функцій. При цьому передбачається, що управління спирається на читабельні документи, що містять інструкції для співробітників організації, необхідних для виконання;

- електронний рух капіталу;
- електронну торгівлю, яка є частиною комп'ютерного програмного забезпечення, яке дозволяє користувачам розміщувати замовлення на фінансові продукти через мережу з фінансовим посередником. Ці продукти включають такі продукти, як акції, облігації, валюти, товари та похідні фінансові інструменти. Першим широко поширеним електронним торговим майданчиком стала американська фондова біржа Nasdaq. Доступність таких торгових платформ для публіки сприяла зростанню роздрібних інвестицій;

- електронні гроші;
- електронний маркетинг;
- електронний банкінг;
- електронні страхові послуги.

Будь-який користувач мережі Інтернет з різних точок нашої планети може зробити замовлення в інтернет-магазині. Цей процес вимагає щоденного

покращення. Оскільки мільйони замовлень щодня обробляються у різних інтернет-магазинах, продавцям стало необхідне ефективне спостереження за замовленнями, транзакціями та відправками. Для цього були створені e-commerce платформи, призначені безпосередньо для зручного керування товарами, групами товарів, замовниками, замовленнями, відправленнями та іншим. Серед найпопулярніших e-commerce платформ можна виділити Shopify, WooCommerce [1], Magento, OpenCart та інше [2].

Проте, адміністративні панелі e-commerce платформ обмежені за функціоналом і не дозволяють надійним чином відстежувати аналітику замовлень в інтернет-магазині. Для цього необхідне встановлення додаткових плагінів або застосунків.

Найбільш використовуваною платформою для електронної комерції є Shopify. Вона містить зручний інтерфейс, налаштована під потреби продавця та має адміністративну панель з великою кількістю розділів для додавання, редагування та видалення товарів, колекцій, замовлень, покупців та інше.

1.1 Відслідковування аналітики замовлень засобами платформи Shopify

Керування сучасним інтернет-магазином потребує пильної уваги та програмного забезпечення із зручним інтерфейсом. Для цього e-commerce платформи мають вбудовані панелі адміністратора, які дозволяють керувати процесами інтернет-замовлень, асортиментом товару, колекціями, шаблонами сторінок, блогами та іншим.

Панель адміністратора інтернет-магазину – це програмне забезпечення, де відображається ключова інформація про нещодавні дії вашого магазину, подальші дії, які вам необхідно зробити з вашими замовленнями, та поради щодо розвитку вашого бізнесу.

Панель адміністратора платформи Shopify включає в себе наступне:

- Shopify Home – домашня сторінка, показує інформацію про щоденні завдання, нещодавню активність вашого магазину, візуалізаційну тему інтернет-магазину;
- бокова панель – навігаційна панель з усіма розділами панелі;
- розділ “Замовлення”, “Продукти” та “Клієнти” – включають панель з основними аспектами ведення бізнесу у Shopify;
- канали продажу – панель, яка включає налаштування каналів продажу товарів інтернет-магазин, точку продажу товарів та кнопку «Купити»;
- розділ “Додатки” або “Apps” – встановлені застосунки, які являються додатковим функціоналом інтернет-магазину;
- розділ “Налаштування” – розділ з налаштуваннями, застосованими до магазину Shopify та облікового запису;
- панель пошуку – створена для того, щоб швидко шукати продукти, сторінки чи іншу інформацію в адмінці Shopify;
- спадаючий список зі списком облікових записів – функціонал для переходу до магазинів з іншими обліковими записами.

Щоб використати Shopify, необхідно використати сучасний веббраузер. Якщо не використовується остання версія браузера, можливе виникнення помилок, або ж взагалі не вийде отримати доступ до панелі адміністратора Shopify або використовувати всі його функції. Для досягнення найкращих результатів використовувати Shopify рекомендовано з останньою версією Google Chrome. Shopify підтримує дві останні версії всіх браузерів, якщо не вказано інше [3].

Інтерфейс панелі адміністратора платформи Shopify наведено на рис. 1.1.

Основною метою адміністративної панелі інтернет-магазину Shopify є керування всіма аспектами свого магазину незалежно від пристрою яким є під рукою у користувача. Якщо увійти у Shopify з телефону, можна переглядати останні дії магазину, оновлювати каталог магазину та керувати замовленнями.

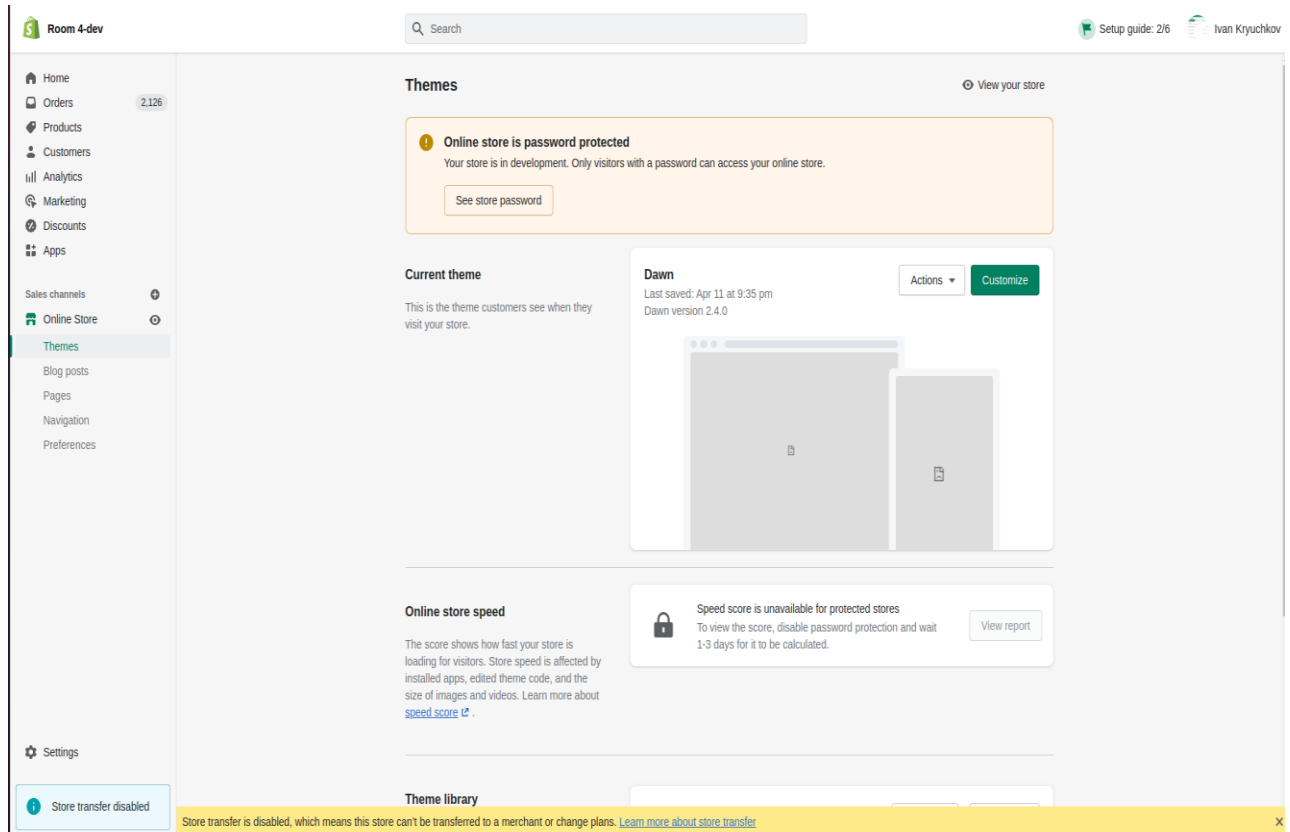


Рисунок 1.1 – Панель адміністратора платформи Shopify

Shopify Admin показує останню активність відвідувачів вашого магазину. Ваші дії в магазині записуються відповідно до місцевого часового поясу вашого магазину.

Проте деякий функціонал, такий як аналітика, у вбудованій панелі адміністратора обмежений. Щоб подолати ці обмеження, необхідне створення додаткового програмного продукту для відстеження аналітики замовлень.

Цей застосунок допоможе легко відстежити всі замовлення за певний період часу, кількість покупців на один товар за цим замовленням, дати замовлень та іншу корисну інформацію.

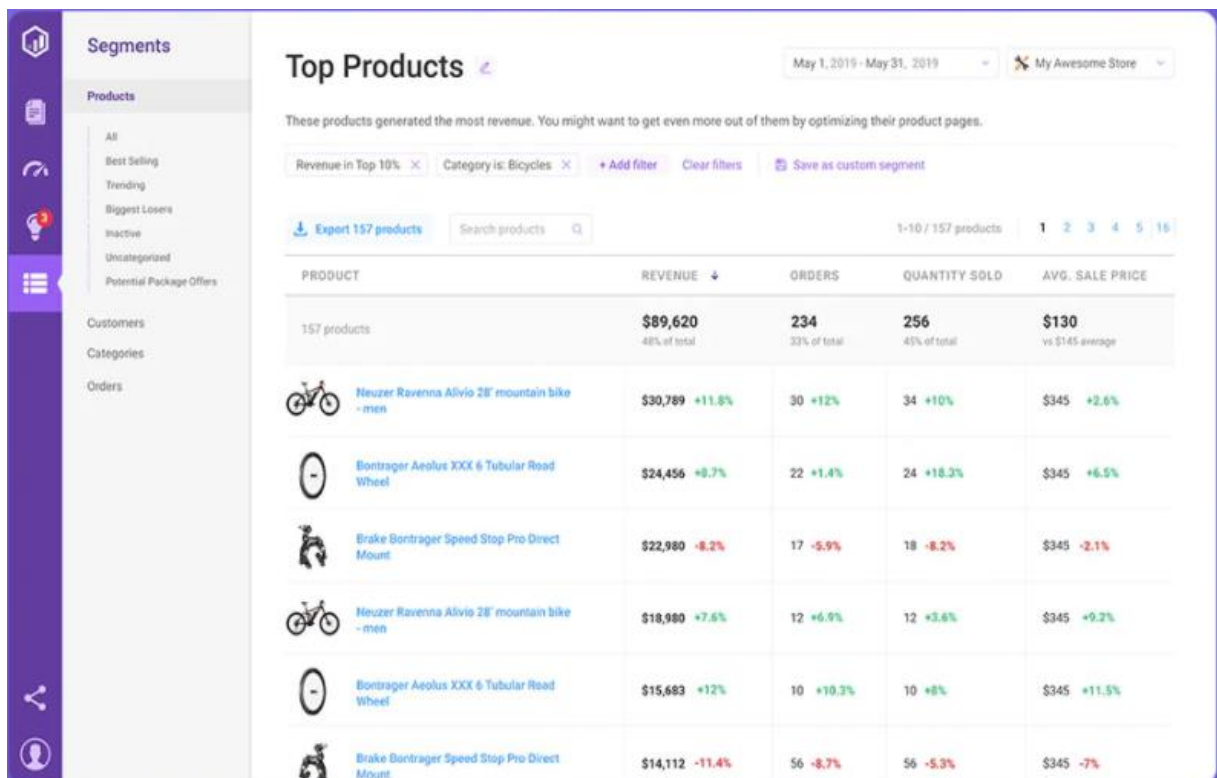
Метою застосунку з аналітикою є розширення вбудованого функціоналу спостереження показників інтернет-магазину у платформі Shopify, за допомогою якого можна робити висновки з покращення ведення бізнесу, логістики інтернет-замовлень та кращого обслуговування клієнтів з різних куточків планети.

1.2 Аналіз існуючих програмних засобів для відстеження аналітики інтернет-магазину Shopify

1.2.1 Profit Analytics & Benchmark

Одним з найпопулярніших програмних продуктів для відстеження аналітики замовлень інтернет-магазину Shopify є Profit Analytics & Benchmark.

Зі свого інтерфейсу (рис. 1.2) застосунок дозволяє отримувати аналітичні звіти для моніторингу інтернет-магазину, об'єднати Shopify та Google Analytics, щоб отримати всі необхідні дані в одному місці.



PRODUCT	REVENUE	ORDERS	QUANTITY SOLD	AVG. SALE PRICE
157 products	\$89,620 48% of total	234 33% of total	256 45% of total	\$130 vs \$145 average
Neuzer Ravenna Alivio 28" mountain bike - men	\$30,789 +11.8%	30 +12%	34 +10%	\$345 +2.6%
Bontrager Aeolus XXX 6 Tubular Road Wheel	\$24,456 +8.7%	22 +1.4%	24 +18.3%	\$345 +6.5%
Brake Bontrager Speed Stop Pro Direct Mount	\$22,980 -8.2%	17 -5.9%	18 -8.2%	\$345 -2.1%
Neuzer Ravenna Alivio 28" mountain bike - men	\$18,980 +7.6%	12 +6.9%	12 +3.6%	\$345 +9.2%
Bontrager Aeolus XXX 6 Tubular Road Wheel	\$15,683 +12%	10 +10.3%	10 +8%	\$345 +11.5%
Brake Bontrager Speed Stop Pro Direct Mount	\$14,112 -11.4%	56 -8.7%	56 -5.3%	\$345 -7%

Рисунок 1.2 – Інтерфейс застосунку “Profit Analytics & Benchmark”

Користувач завжди буде в курсі автоматичних щотижневих та щоденних звітів за найважливішими показниками (продаж, прибуток, собівартість), матиме змогу відстежувати свій прибуток за допомогою точних фінансових

звітів та розрахунків прибутку, слідкувати за показниками замовлення (витрати/продажу, прибуток тощо).

На відміну від своїх аналогів, застосунок інтегрується з Google Analytics, Google Console, Facebook Marketing, Google Marketing. Це легко дозволить за допомогою маркетингової аналітики та аналітики продажів відстежувати замовлення, що надходять з кожного каналу продажу та маркетингу, включаючи Amazon, Facebook Store, Instagram, eBay і т.д. [4].

1.2.1 Glew Analytics

Одним з найбільш функціональних застосунків є Glew Analytics. Glew підтримує більшість основних застосунків для електронної комерції та роздрібно́ї торгівлі.

Особливістю цього застосунку є зручний, наповнений графікою інтерфейс (рис. 1.3) з діаграмами, які візуально відображають аналітичні дані.

Також, корисним функціоналом цього застосунку є синхронізація з електронною поштою, тобто усі необхідні звіти з аналітикою надходять адміністратору на поштову скриньку за відповідним запитом.

Glew має функціоналбору аналітики найактивніших маркетингових каналів, це дозволяє підвищити рентабельність інвестицій у рекламу, залучати нових клієнтів та робити вірні висновки у розподілі витрат на рекламу.

Застосунок має можливість інвентарної аналітики. Він відображає інформацію про продукти, які найкраще продаються у різних групах клієнтів та каналах.

Кожна метрика і візуалізація в Glew можуть бути додані в звіти, що налаштовуються, які допоможуть вимірювати продуктивність, порівнювати категорії продуктів або групи клієнтів, оцінювати канали, деталізувати тенденції та багато іншого. Будь-який звіт може бути доставлений електронною поштою або запланований щоденно, щотижня або щомісяця [5].

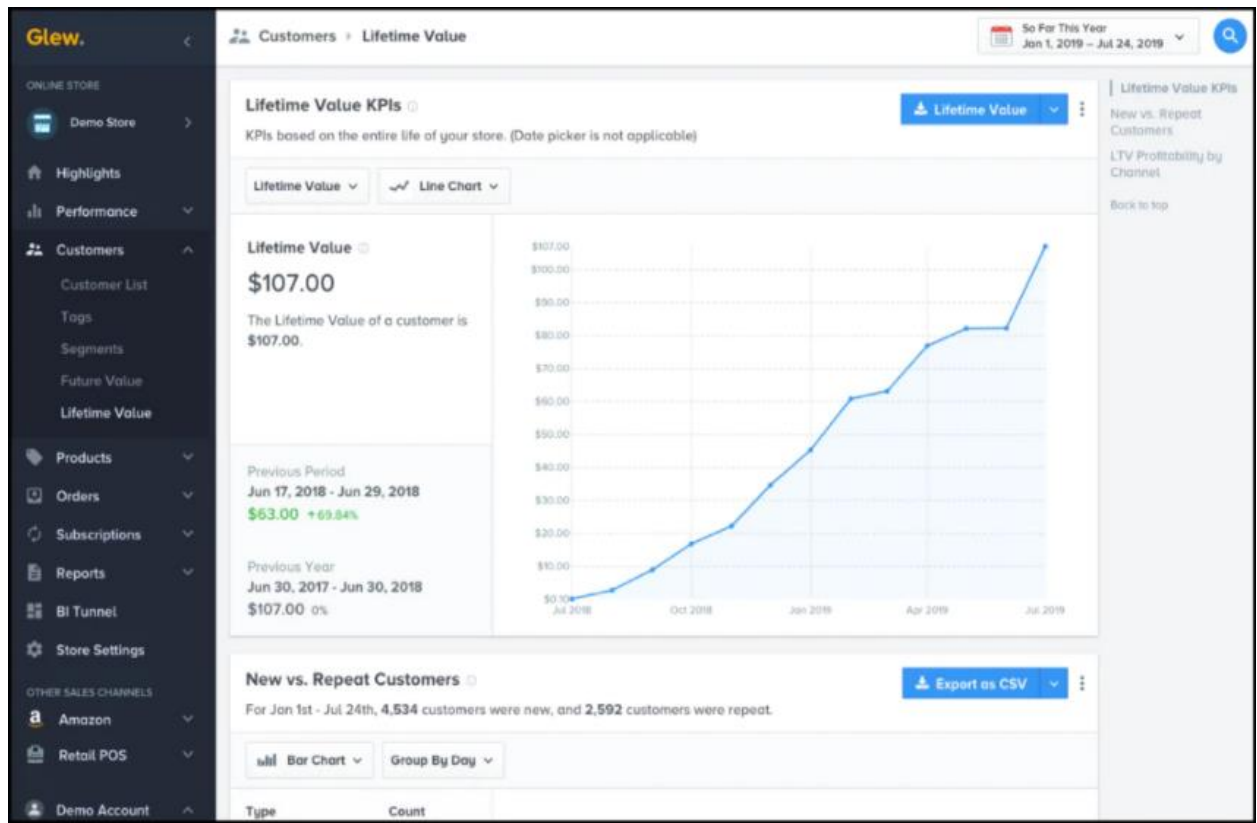


Рисунок 1.3 – Інтерфейс застосунку “Glew Analytics”

1.2.3 Connector for Google Analytics

Також, одним з програмних продуктів для відстежування аналітики магазинів Shopify є Connector for Google Analytics. Він пропонує повні дані про продаж, точну маркетингову атрибуцію та параметри, що настроюються для звітів про вартість життя (LTV) в Google Analytics. Однією з можливостей застосунку є відстеження аналітики продажу товарів на передплату. Потужні інтеграції для відстеження замовлень на передплату легко підключаються до ReCharge, Smartrr, Ordergroove або Bold Subscriptions.

Застосунок працює для будь-якої кількості магазинів, будь-якої тематики, будь-якого обсягу замовлення та має потужну синхронізацію з Google Analytics або будь-яким інструментом звітності (наприклад, Data Studio, Tableau, Power BI, Glew) [6].

Також, програмний продукт має міждоменне відстеження та підтримку кількох валют та магазинів.

У режимі реального часу застосунок показує рівень даних, що розширюється за допомогою диспетчера тегів Google (GTM).

Застосунок має достатньо простий, наближений до Google Analytics інтерфейс (рис. 1.4).

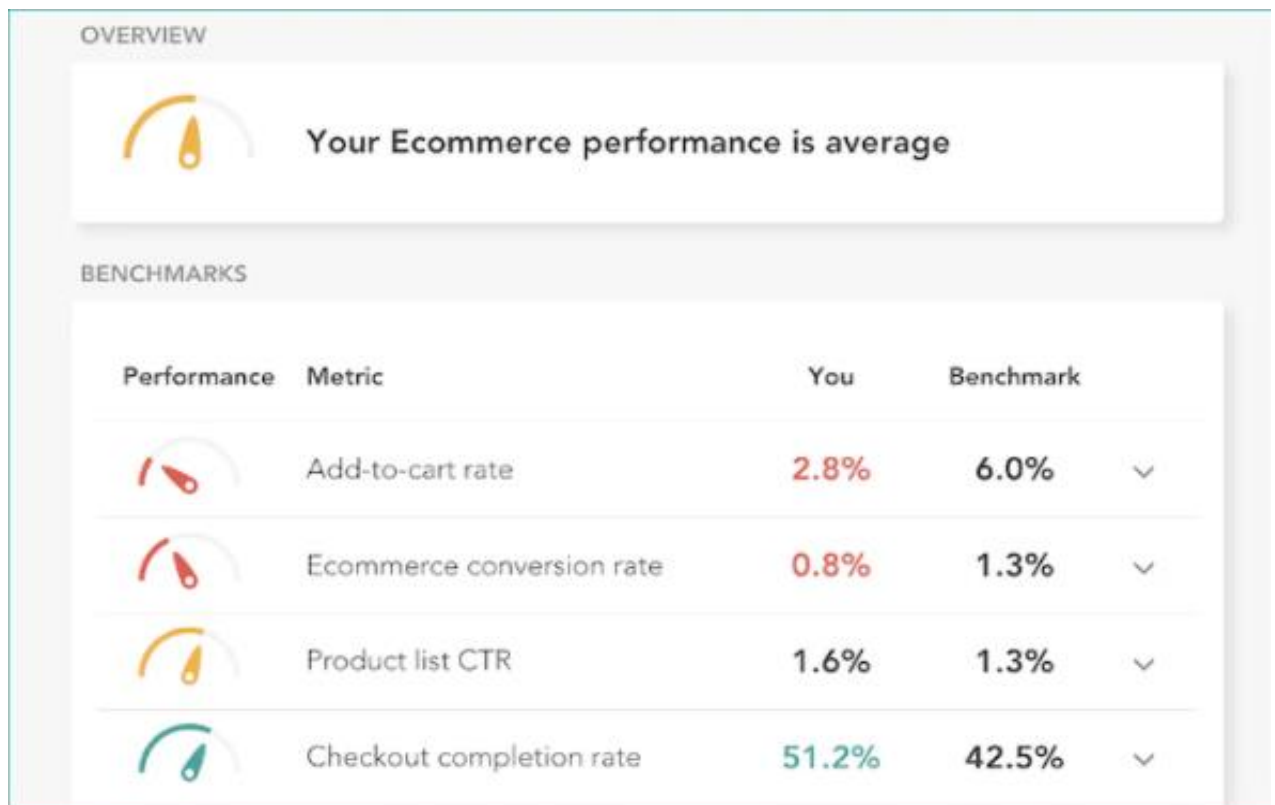


Рисунок 1.4 – Інтерфейс застосунку “Connector for Google Analytics”

1.2.4 StoreView

Ще одним прикладом програмного продукту відстежування аналітики у Shopify є StoreView. StoreView записує відвідування сторінок, зміни в кошику, перевірки та замовлення, які відбуваються у вашому магазині в режимі реального часу. Застосунок надає змогу побачити, які продукти додаються до кошика користувачів, що дозволить приймати розумні рішення щодо того, як збільшити кількість конверсій.

Застосунок надає такі можливості:

- детальне переглядання активності кошика магазину, коли користувачі додають чи видаляють товари;
- спостереження за активністю користувачів, коли вони оформляють замовлення та розміщують їх;
- щоденні звіти у поштовій скриньці, які показують хорошу розбивку всієї активності, що відбулася у магазині;
- сумісність із усіма пристроями;
- миттєве встановлення.

Інтерфейс застосунку (рис. 1.5) простий у використанні, має багато графічних елементів та діаграм, наповнений великою кількістю елементів вибору дат, користувацьких дій, пагінації, що збільшує взаємодію з користувачем [7].



Рисунок 1.5 – Інтерфейс застосунку “StoreView”

1.3 Постановка завдання до роботи

На основі аналізу існуючих застосунків збору аналітики у інтернет-магазинах платформи Shopify було вирішено створити застосунок який би містив основний та необхідний функціонал, а саме:

- авторизацію, синхронізовану з особистим інтернет-магазином;
- відображення діаграм;
- відображення товарів, що найкраще продаються;
- статистику замовлень;
- фільтрацію та сортування замовлень;
- перегляд сторінки одного замовлення при переході за посиланням.

1.4 Висновки за розділом 1

У розділі було розглянуто основні аспекти функціоналу для створення застосунку збору аналітики інтернет-магазину платформи Shopify. Також, було розглянуто існуючі програмні продукти з подібним функціоналом та виділено ключові функції для реалізації дипломної роботи.

На основі аналізу було зроблено висновки щодо розробки програмного продукту з необхідним функціоналом для відображення аналітичних даних інтернет-замовлень товарів в магазині платформи Shopify та поставлено завдання у реалізації вебзастосунку. У рамках дипломної роботи необхідно створити програмний продукт у вигляді адміністративної панелі, пов'язаний з відображенням статистичних даних у вигляді діаграм та таблиць, авторизацією, фільтрацією, сортуванням та окремою сторінкою одного замовлення при переході за посиланням.

2 АНАЛІЗ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Принцип роботи клієнт-серверної архітектури застосунку

Вебзастосунки являють собою клієнт-серверні застосунки. При цьому в якості клієнта виступає браузер. Вебзастосунок має три рівні: браузер, сервер і сервер бази даних. Спроектowana схема зображена на рис. 2.1.

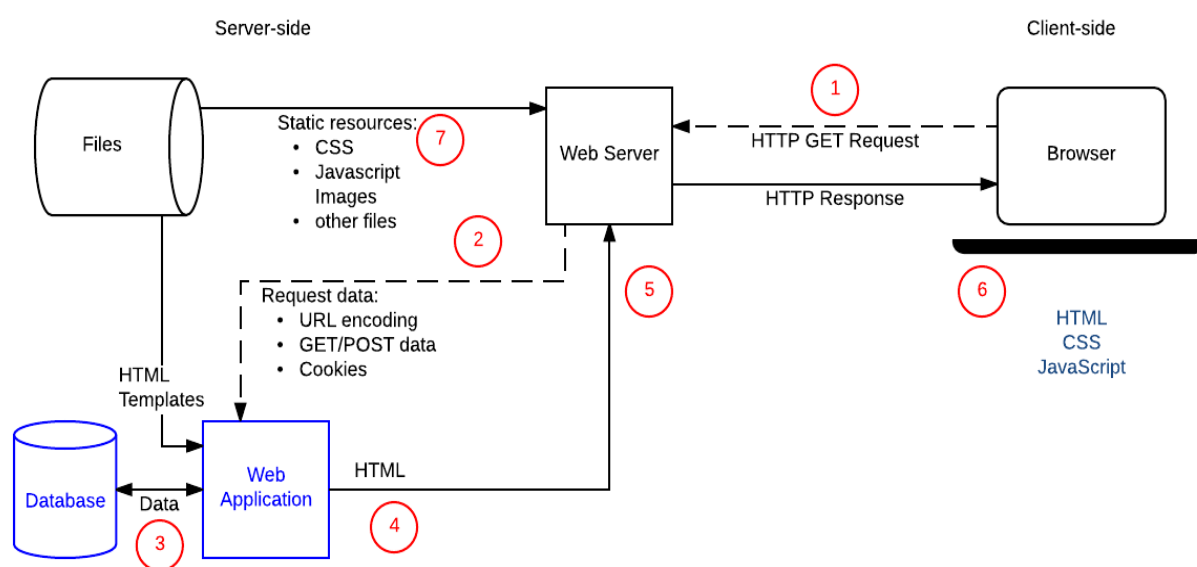


Рисунок 2.1 – Схема клієнт-серверної архітектури вебзастосунку

Вебсервер включає в себе декілька компонентів, які контролюють доступ користувачів до розміщених на сервері файлам, як мінімум це HTTP-сервер. HTTP-сервер – це частина програмного забезпечення, яка розуміє URL (Uniform Resource Locator) та HTTP (HyperText Transfer Protocol).

На самому базовому рівні, коли браузеру потрібен файл, розміщений на вебсервері, браузер запитує його через HTTP-протокол. Коли запит досягає потрібного вебсервера, сервер HTTP приймає запит, знаходить запитуваний документ (якщо документа немає, то повідомляє про помилку 404) і відправляє назад, також через HTTP.

2.1.1 Архітектура передачі даних між клієнтом та сервером

Для зв'язку між клієнтом та сервером використовується GraphQL та REST API.

GraphQL – це мова запитів даних та маніпулювання ними з відкритим вихідним кодом для API, а також середовище виконання для виконання запитів із існуючими даними.

GraphQL має три основні характеристики:

- дозволяє клієнту точно вказати, які дані йому потрібні;
- полегшує агрегацію даних із кількох джерел;
- використовує типову систему для опису даних.

REST (від англ. Representational State Transfer – "передача репрезентативного стану" або "передача "самоописуваного" стану") – архітектурний стиль взаємодії компонентів розподіленого застосунка в мережі. Іншими словами, REST – це набір правил того, як програмісту організувати написання коду серверного застосунка, щоб усі системи легко обмінювалися даними і програму можна було масштабувати. REST є узгодженим набором обмежень, що враховуються при проєктуванні розподіленої гіпермедіа-системи.

2.1.2 Архітектура програмного продукту

Вебзастосунок побудований за архітектурою PWA (Progressive Web Application) (рис. 2.2). PWA — це вебзастосунок, створений за допомогою певних технологій для досягнення конкретних результатів.

Браузер використовується як віртуальна машина, що зберігає та запускає в собі програму PWA. Як Android є віртуальною машиною для андроїд-застосунків, так і браузер стає віртуальною машиною для PWA. Як нативний застосунок звертається через файлову систему до своїх ресурсів, так і PWA звертається до своїх ресурсів по HTTP, але зберігається локально [9].

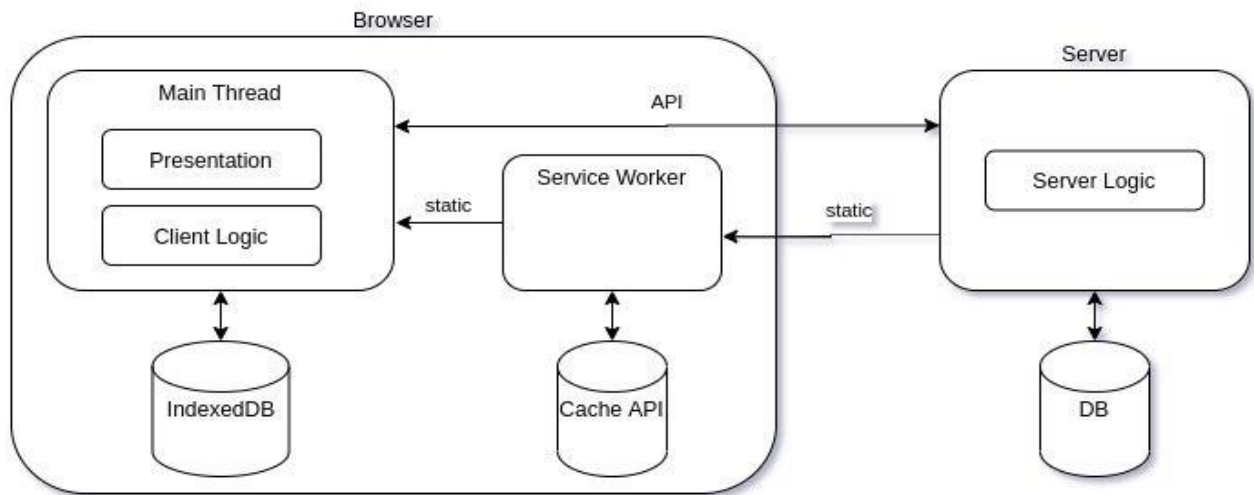


Рисунок 2.2 – Схема

Сердце PWA – Service Worker. Це шар-посередник між клієнтською та серверною частиною, що у браузері. Всі запити браузера проходять через нього. Даний поділ на два незалежні шари дозволив зробити перехід звичайного вебсайту в PWA максимально простим. Service Worker являє собою Javascript-файл, що підключається в HTML-коді сторінки. У ньому розробник визначає логіку роботи з запитами, що приходять з клієнту, та іншу функціональність.

PWA вимагає, щоб усі ресурси сайту передавалися за протоколом HTTPS. SSL сертифікат можна отримати безкоштовно. Але критично, щоб на сайті не було посилань на незахищені ресурси – деякі браузери просто не відображатимуть сайт у цьому випадку.

Шаблоном графічного інтерфейсу архітектури PWA є App Shell. App Shell зберігається на клієнті та завантажується при запуску програми, а потім уже в нього вантажиться з мережі динамічна інформація. І поки вона вантажиться, App Shell має виглядати у вигляді «лоадера», тощо.

PWA має обов’язково містити Web App manifest файл. Це файл формату JSON (JavaScript Object Notation), який декларативно визначає для браузера назву програми, іконку, як виглядатиме PWA (fullscreen, standalone та ін) та деякі інші параметри. Дозволяє встановити PWA як окрему програму на домашній екран смартфона.

Push Notifications – складова PWA яка поширює інформацію (контент) про застосунок в Інтернеті, коли дані надходять від постачальника до користувача на основі встановлених параметрів. Користувач же, у свою чергу, або відхиляє або приймає дані.

Простіше кажучи, Push сповіщення – це коротке спливаюче повідомлення у програмі або браузері. Його надсилають користувачам, щоб розповісти про оновлення, новини та акції. Головна мета push повідомлення – доставити клієнту релевантну інформацію для підтримки залучення.

2.2 Шаблон проєктування

Розробка вебзастосунків для платформи Shopify підтримує деякі особливості проєктування надані самою компанією Shopify. Одним з таких шаблонів є Shopify App Template (рис. 2.3) .

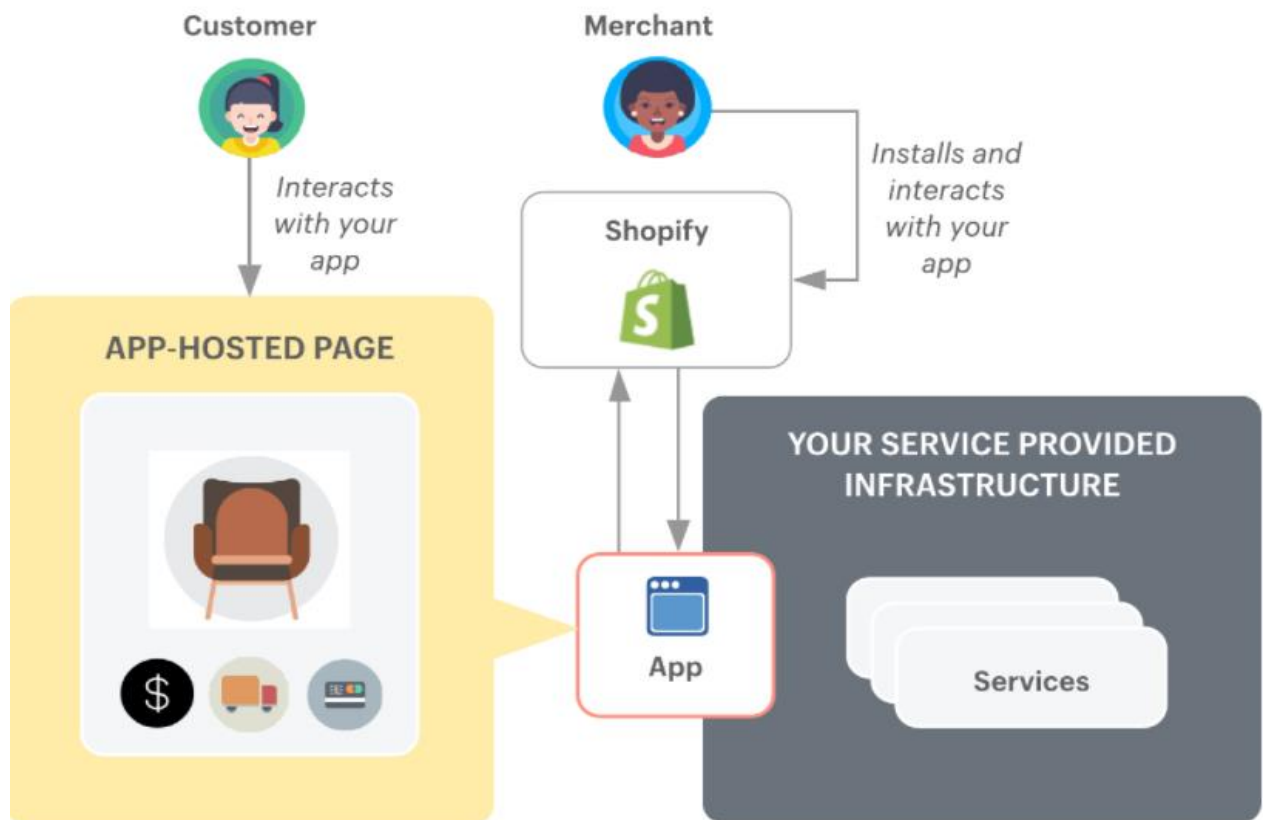


Рисунок 2.3 – Схема роботи Shopify App Template

За замовчуванням, цей спроектований паттерн поставляється з базовою комплектацією залежностей який можна завантажити з Shopify CLI (Command Line Interface).

Є три ключові області, в яких програми взаємодіють з платформою Shopify: адміністрація Shopify, інтернет-магазин та оформлення замовлення.

Цей шаблон створений для більш зручної інтеграції з ключовими областями інтернет-магазину та має базову структуру для зручної взаємодії застосунку с API.

Будь-які застосунки у Shopify завдяки цьому шаблону інтегруються таким чином:

- підключення до API-інтерфейсів Shopify для читання та запису даних магазину у відповідь на введення даних користувачем або повідомлення про події вебперехоплювача;
- розширення нових функцій усередині існуючих частин адміністратора Shopify або POS (Point of sale);
- покращення способу зберігання відображуваної інформації для клієнтів.

Використовуваний у розробці програмного продукту Shopify App Node Template допомагає дотримуватись цих парадигм за допомогою мови програмування JavaScript [10].

2.3 Вибір мови програмування

Середовище Shopify надає цілий ряд програмних інструментів, шаблонів застосунків, прикладів програм і бібліотек, щоб прискорити процес розробки.

Для створення PWA та зручного зв'язку між клієнтом та сервером було обрано Shopify Node App Template. Цей шаблон містить вбудовані залежності, які можна використати у створенні застосунку, а також вбудований функціонал для авторизації (OAuth) застосунку з системою Shopify Admin API, яка містить у собі дані адміністрування інтернет-магазином.

Shopify Node App Template пропонує будувати PWA за допомогою мови програмування JavaScript на стороні клієнту та сервера (рис. 2.4), також для зв'язку між клієнтом та сервером цей шаблон пропонує використовувати REST або GraphQL API.

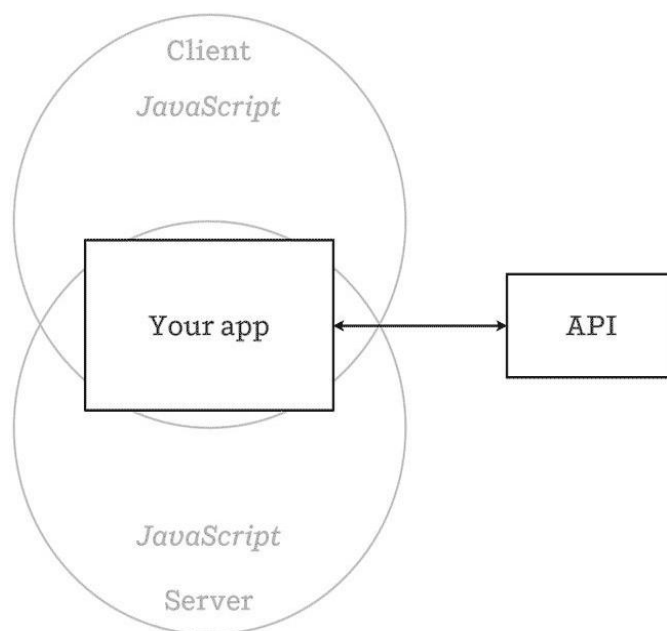


Рисунок 2.4 – Схема роботи клієнт-серверного JavaScript

JavaScript – це мультипарадигменна мова програмування, яка зазвичай застосовується як вбудований інструмент для програмного доступу до різних об'єктів застосунків. З погляду веброзробки без знань цієї технології неможливо займатися створенням сучасних інтерактивних сайтів. Мова JS – це те, що «оживляє» розмітку сторінок (HTML) та користувацький функціонал (CMS) сайтів. За допомогою цієї мови реалізується можливість реакції сторінки або її елементів на дії відвідувача. Наразі JavaScript є базовою мовою програмування для браузерів. Він повністю сумісний із операційними системами Windows, Linux, Mac OS, а також усіма популярними мобільними платформами.

Мова програмування JavaScript на стороні серверу дозволяє реалізовувати такі задачі:

- забезпечення захисту інформації;
- зберігання різноманітних електронних матеріалів;
- надання доступу до даних;
- безпосереднє оброблення клієнтських запитів;
- надсилання відповідей на ту чи іншу команду, надіслану від клієнтів.

Проте, на стороні клієнту JavaScript може дозволити зробити наступне:

- надання доступу до користувальницького графічного інтерфейсу;
- формування запиту для подальшої обробки та виконання тієї чи іншої команди;

- надсилання запитів на сервер;
- надання доступу до користувальницького графічного інтерфейсу;
- формування запиту для подальшої обробки та виконання тієї чи іншої команди;

- надсилання запитів на сервер;
- отримання результатів;
- направлення додаткових груп команд (додавання, оновлення, видалення відомостей);
- направлення додаткових груп команд (додавання, оновлення, видалення відомостей) [8].

Кожна мова програмування має свої недоліки і слабкі місця. Однією з причин виникнення проблем є популярність мови. Коли мова програмування стає такою популярною як JavaScript, вона стає об'єктом підвищеного інтересу для хакерів, шахраїв та інших шкідливих проявів третіх сторін, які намагаються знайти вразливості та слабкі місця у безпеці. Деякі слабкі місця:

- вразливий до експлойтів (шкідливий код, що використовує вразливості програмного продукту);
- може бути використаний для запуску шкідливого коду комп'ютера користувача;

- не завжди підтримується деякими браузерами чи пристроями;
- фрагменти JS коду можуть бути дуже великими;
- може по-різному відображатись на різних пристроях, що призводить до відсутності цілісності.

2.4 Вибір JavaScript-фреймворків

На стороні сервера використовується фреймворк серверного JavaScript – Koa.js, який являє собою мінімальний інтерфейс для створення програм. Це дуже маленька структура (600 LoC), яка надає необхідні інструменти для створення програм та досить гнучка. На npm є безліч модулів для Koa, які можна безпосередньо підключити до нього. Koa можна розглядати як ядро express.js [11].

Для клієнтської частини задля підтримки покращеної обробки сторінки на сервері буде використовуватись фреймворк React.js – Next.js. Він має покращений функціонал маршрутизації та обробки сторінок, а також дозволяє легко взаємодіяти з сервером у спрощеній формі.

Серед особливостей фреймворку Next.js можна виділити наступне:

- інтуїтивна система маршрутизації на основі сторінок (з підтримкою динамічних маршрутів);
- попередній рендеринг як статична генерація (SSG), так і рендеринг на стороні сервера (SSR) підтримуються окремо для кожної сторінки;
- автоматичне розділення коду для швидкого завантаження сторінок;
- маршрутизація на стороні клієнта з оптимізованою попередньою вибіркою;
- вбудована підтримка CSS та SASS, а також підтримка будь-якої бібліотеки CSS-in-JS;
- повністю розширюваний.

Next.js використовується на десятках тисяч робочих вебсайтів та вебзастосунків, у тому числі багатьох найбільших світових брендів [12].

Переваги та недоліки фреймворку Next.js наведені у таблиці 2.1.

Таблиця 2.1 – Переваги та недоліки фреймворку Next.js

Переваги	Недоліки
Популярні інструменти, такі як React, Material-UI, Builder, Sanity та DatoCMS, сумісні з Next.js.	Нижча продуктивність у порівнянні з іншими фреймворками.
Автоматичний рендеринг на стороні сервера та розбиття коду	Створений спеціально для роботи з React.js.
Дозволяє створювати динамічні метатеги у розділі <head> для оптимізації SEO. У Next.js є гаряче перезавантаження сторінки при виявленні будь-якої зміни.	Представлений набір певних алгоритмів та інструментів, які можуть відрізнятися від використовуваних у React. Розробнику доведеться адаптуватися до цього. Через те, що фреймворк достатньо молодий не дуже розвинений інформаційний простір.

У таблиці вказані лише основні переваги та недоліки, але насправді у фреймворку багато переваг, у зв'язку з чим можна зробити висновок, що це актуальний, прогресуючий і широко використовуваний в розробці фреймворк.

2.5 Висновки за розділом 2

В розділі 2 було проведено аналіз технологій, що використовуються для розробки подібних застосунків. На основі аналізу було прийняте рішення реалізувати вебзастосунок, а саме PWA за допомогою однієї з найпопулярніших мов програмування у світі JavaScript на стороні клієнта та сервера.

Застосунок буде використовувати Shopify App Node Template задля підтримки мови JavaScript та фреймворків відповідно. Для більш зручної

розробки PWA на стороні клієнту буде використано фреймворк Next.js, в основі якого закладений React.js, на стороні сервера буде використано Koa.js який буде містити усі достатні для серверної розробки застосунку модулі.

Для зв'язку між клієнтом та сервером буде використовуватися GraphQL та REST API. Завдяки схемам GraphQL можна швидко отримувати та обробляти необхідні дані з сервера, а саме з Shopify Admin API, за допомогою зручного синтаксису. Ті дані, які буде неможливо отримати за допомогою GraphQL з Shopify Admin, API застосунок буде отримувати за допомогою REST.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ СИСТЕМИ

3.1 Розробка схеми отримання даних

Створений вебзастосунок на стороні сервера виконує функціональну роль парсера (зчитувача даних з API, який у подальшому буде оброблений у зручному для виводу на клієнті вигляді). Джерелом для парсингу даних є Shopify Admin API, ці дані будуть отримані за допомогою GraphQL та REST. У випадку REST застосунок не має потреби конструювання схем, так як запит повертає JSON за відповідним URL, проте GraphQL працює лише за допомогою будови відповідних до синтаксису цієї технології схем.

Коли користувач виконує вхід до інтерфейсу застосунку, необхідно зчитати дані про магазин (рис. 3.1).

```
1 {  
2   shop {  
3     id  
4     myshopifyDomain  
5     name  
6     email  
7     url  
8   }  
9 }
```

Рисунок 3.1 – Схема даних для отримання інформації про магазин

Запит “shop” зберігає основні дані, які будуть відображені в адміністративній панелі застосунку.

Параметр “id” повертає унікальний ідентифікатор магазину який внесений в Shopify Admin API.

Параметр “myshopifyDomain” повертає назву домену, на якому існує магазин.

Параметр “name” повертає назву магазину з системи налаштувань, внесених адміністратором.

Параметр “email” повертає адресу поштової скриньки магазину з системи налаштувань внесених адміністратором.

Параметр “url” повертає адресу магазину у мережі.

Для отримання даних про замовлення, побудована схема GraphQL з необхідними параметрами для отримання з API масиву даних замовлень (рис. 3.2).

```

2  orders {
3    edges {
4      node {
5        id
6        name
7        currencyCode
8      totalPriceSet {
9        shopMoney {
10         amount
11       }
12     }
13    shippingAddress {
14      zip
15      city
16      country
17      name
18      address1
19    }
20    lineItems {
21      edges {
22        node {
23          quantity
24        }
25      }
26    }
27    createdAt
28    displayFinancialStatus

```

Рисунок 3.2 – Схема даних для отримання масиву замовлень

Запит “orders” зберігає дані які будуть відображені в таблиці замовлень.

На основі цього запиту буде побудований функціонал для пагінації, фільтрації та сортування даних.

Параметр “id” повертає унікальний ідентифікатор замовлення, який внесений в Shopify Admin API.

Параметр “name” повертає номер замовлення, який внесений в Shopify Admin API.

Параметр “currencyCode” повертає усі доступні коди валют, за якими було сплачено це замовлення.

Параметр “totalPriceSet” повертає повну грошову суму замовлення.

Параметр “shippingAdress” повертає адресу доставки, у яку включені такі поля як zip-код, країна, місто, ім’я замовника та його фактична адреса.

Параметр “lineItems” повертає деталі замовлення, у цьому випадку повертає кількість одиниць товару у замовленні.

Параметр “createdAt” повертає дату створення замовлення.

Параметр “displayFinancialStatus” повертає фінансовий статус оплати замовлення.

Для отримання даних про топ-5 товарів, які купують найчастіше, також використовується схема даних (рис. 3.3).

```
{
  orders(first: 5, sortKey: TOTAL_PRICE, query: $query) {
    edges {
      node {
        id
        name
        currencyCode
        totalPriceSet {
          shopMoney {
            amount
          }
        }
      }
    }
  }
}
```

Рисунок 3.3 – Схема даних отримання топ-5 замовлень

У дужках після назви запиту прописані параметри за якими потребується отримати топ-5 замовлень. Параметр “first” – вказує кількість отриманих даних у масиві, параметр “sortKey” – вказує, за яким критерієм ми отримуємо необхідну кількість даних під час сортування, у цьому випадку – за сумою оплати замовлень, та параметр “query” – містить формулу у змінній \$query, яка вказує, за який період часу необхідно отримати замовлення.

Інші параметри повертають значення описані у схемі отримання замовлень з рис. 3.2.

3.2 Розробка вебінтерфейсу

Інтерфейс програмного продукту розроблено за стандартом Shopify App Template. За цим стандартом створювані застосунки, які інтегруються з платформою Shopify, мають бути побудовані за дизайн-концепцією, яка розроблена компанією Shopify під назвою “Polaris”.

“Polaris” – це візуальна бібліотека з розробленими компонентами за стандартами Shopify Platform [13].

Створений застосунок побудований компонентно. Основний шаблон який містить шапку і навігаційну панель, а також фрейм для відображення контенту застосунку міститься у файлі index.js директорії DefaultLayout. Цей файл містить у собі підключення глобальних стилів застосунку, написані завдяки препроцесору SASS у форматі .scss та імпорти головних компонентів з бібліотеки “Polaris”.

```
import React, { useCallback, useState } from "react";
import TopBarMarkup from "../../components/TopBar";
import NavigationMarkup from "../../components/Navigation";
import { Frame, Page } from "@shopify/polaris";
import styles from "../styles.module.scss";
```

```

const DefaultLayout = ({ children }) => {
  const [mobileNavigationActive, setMobileNavigationActive] =
    useState(false);
  const toggleMobileNavigationActive = useCallback(
    () =>
      setMobileNavigationActive(
        (mobileNavigationActive) => !mobileNavigationActive
      ),
    []
  );
  const topBarMarkup = (<TopBarMarkup
toggleMobileNavigationActive={toggleMobileNavigationActive} />
  );
  return (
    <Frame
      topBar={topBarMarkup}
      navigation={<NavigationMarkup />}
      showMobileNavigation={mobileNavigationActive}
      onNavigationDismiss={toggleMobileNavigationActive}
    >
      <div className={styles.PageWrapper}>
        <Page>{children}</Page>
      </div>
    </Frame>
  );
};
export default DefaultLayout;

```

Стилі підключаються модульно за допомогою директиви `import`, таким чином клієнтська частина застосунку зчитує зміст файлу зі стилями з вказаної директорії у змінну `styles`, яка в браузері інтерпретується у тег `<link rel="stylesheet">`.

Для того, щоб в браузері відобразилися компоненти шапки та навігаційної панелі, з відповідних директорій за допомогою директиви `import` завантажуються модулі `TopBarMarkup` та `NavigationMarkup`, які в браузері інтерпретуються як HTML-розмітка, а у react-компоненті як `jsx` (спеціальний синтаксис написання компонентів у фреймворку `React.js`). Для коректного відображення навігаційної панелі на мобільних пристроях, написана функція `toggleMobileNavigationActive` яка приймає значення `true`, якщо мобільний вид навігаційної панелі не відображений, та значення `false` якщо мобільна панель вже показана на екрані.

Головні компоненти, огорнуті у батьківський компонент `Frame`, який буде відображати усі компоненти, які задіяні у виконанні коду вебзастосунку.

3.3 Опис реалізованих функцій

3.3.1 Функції авторизації до власного інтернет-магазину

Для того, щоб вхід у власний інтернет-магазин виконувався, необхідно зі сторони серверу визначити маршрут авторизації. Першочергова задача серверу отримати дані існуючого інтернет-магазину, щоб відправити інформацію для коректного відображення на клієнті. Для цього на сервері визначається маршрут `/login`.

```
router.get("/login", async (ctx) => {
  const session = await Shopify.Utils.loadCurrentSession(
    ctx.req,
    ctx.res,
    true
  );
  if (session && session.isOnline && session.expires) {
    return ctx.redirect(`/?shop=${session.shop}`);
  }
  await handleRequest(ctx);
});
```

```
});
```

Для того, щоб сервер отримав необхідні параметри входу і зміг відобразити дані інтернет-магазину, була створена функція `onSubmit()`, яка передає серверу параметр з адресою інтернет-магазину. У випадку, якщо адреси не існує або вона некоректно введена у форму, користувач отримує повідомлення про помилку.

```
const onSubmit = (value) => {
  if (event.target.value == "") {
    return setError("Shop url is required!");
  }
  if (!event.target.url.value.match(pattern)) {
    return setError("Shop url must be demo-
store.myshopify.com");
  }
  window.location.href = `/?shop=${event.target.url.value}`;
};
```

3.3.2 Функція отримання замовлень

Для того, щоб клієнт мав змогу отримувати дані за допомогою схем GraphQL, серверу необхідно визначити маршрут, за допомогою якого будуть надіслані запити у систему GraphQL Shopify Admin API. Для цих завдань визначено маршрут `/graphql`.

```
router.post(
  "/graphql",
  verifyRequest({ accessMode: "online", returnHeader: true }),
  async (ctx, next) => {
    await Shopify.Utills.graphqlProxy(ctx.req, ctx.res);
  }
);
```

);

Для того, щоб клієнт отримував дані за допомогою схем GraphQL, необхідно викликати функцію-хук (влаштовані у React.js функції для здійснення необхідних у компоненті дій) – `useQuery`. Необхідно створити об'єкт, який буде містити поля для запису даних отриманих з сервера у різних випадках(успіх, помилка, завантаження, дозаванатаження або перезавантаження), призначити об'єкту виклик функції-хук, у якості параметрів передати схему GraphQL для отримання замовлень яка знаходиться у змінній `GET_ORDERS`.

```
const { data, loading, fetchMore, error, refetch } =
useQuery(GET_ORDERS, {
  variables: {
    first: ORDERS_LIMIT,
    query: `created_at:>${formatISO(
      filters.date.start
    )} AND created_at:<${formatISO(addDays(filters.date.end,
1))}` ,
  },
});
```

Аналогічним чином, буде зроблено виклик функції для завантаження ТОП-5 замовлень. Для отримання цих даних також буде використовуватись функція-хук `useQuery`, та у якості параметру буде передаватися схема `GET_TOP_ORDERS`.

```
const { data, loading, error } = useQuery(GET_TOP_ORDERS, {
  variables: {
    query: `created_at:>${formatISO(
      currentDate.date.start
```

```

    }) AND
    created_at:<${formatISO(addDays(currentDate.date.end, 1))}` ,
    },
  });

```

Для того, щоб отримати дані для відображення одного замовлення схожим чином, необхідно викликати хук `useQuery` та передати у параметри схему `GET_ORDER` з відповідним унікальним ідентифікатором (`id`) замовлення який необхідно передати через `next`-маршрутизацію.

```

const { data, loading, error } = useQuery(GET_ORDER, {
  variables: {
    id: `gid://shopify/Order/${router.query.id}`,
  },
});

```

Для того, щоб отримати дані про замовлення для відображення у графіку, необхідно на сервері визначити маршрут для отримання даних з системи `Shopify Admin API` за допомогою архітектури `REST`.

```

router.get("/api/statistic", async (ctx) => {
  try {
    const session = await
Shopify.Utills.loadCurrentSession(ctx.req, ctx.res);
    const client = new Shopify.Clients.Rest(
      session.shop,
      session.accessToken
    );
    const data = await client.get({
      path: "orders/count",
      query: {
        status: "any",

```

```

        created_at_min: `${ctx.query.created_at_min}`,
        created_at_max: `${ctx.query.created_at_max}`,
    },
    });
    ctx.body = data;
    ctx.status = 200;
  } catch (err) {
    ctx.status = err.statusCode || err.status || 500;
    ctx.body = {
      message: err.message,
    };
  }
});

```

Для того, щоб отримати дані для відображення у графіку безпосередньо на клієнті, необхідно за допомогою аїах-запиту, який реалізовано у проєкті за допомогою бібліотеки `axios`, зробити запит на сервер за маршрутом `/api/statistic` та отримати ці дані для рендерингу у графік.

```

export const fetchAction = async (params) => {
  try {
    const data = await axios.get("/api/statistic", { params:
params });
    return data;
  } catch (error) {
    console.log(error);
  }
};

```

3.3.3 Функції сортування, пошуку, фільтрації та пагінації

Всі ці операції здійснюються за допомогою клієнтської частини застосунку. Сортування даних виконується за допомогою функції `sortData`, де використовується звичайний сортувальник масиву у JavaScript, влаштований у мову – функція `sort`.

```
export function sortData(rows, index, direction) {
  return rows.sort((rowA, rowB) => {
    const amountA =
      index == 4 ? new Date(rowA[index]) :
parseFloat(rowA[index].slice(1));
    const amountB =
      index == 4 ? new Date(rowB[index]) :
parseFloat(rowB[index].slice(1));
    return direction === "descending" ? amountB - amountA :
amountA - amountB;
  });
}
```

Операції фільтрації, пагінації і пошуку здійснюються за допомогою відповідних директив бібліотеки `apollo-client`, яка допомагає отримувати дані за допомогою схем GraphQL. Основною директивою фільтрації та пошуку є `refetch()` або `fetchMore()`. Вони допомагають перезаписувати дані отримані за допомогою схеми за відповідними параметрами.

Щоб `next.js` коректно відобразив ці дані, використовується функція-хук `useEffect`, який допомагає змінювати стан компоненту при зміні деяких даних у ньому, які прописані у масиві залежностей. Приклад написаного функціоналу фільтрації за фінансовим статусом замовлення:

```
useEffect(() => {
```

```

        let search = filters.search !== "" ? `name:${filters.search}`
: "";

        refetch({
            query: `financial_status:${filters.status.toLowerCase()},
${filters.search}, created_at:>${formatISO(
                filters.date.start
            )} AND created_at:<${formatISO(addDays(filters.date.end,
1))}` ,

            });
        }, [filters.status]);

```

У цьому випадку у якості параметру функції `refetch()` передається фінансовий статус, поточний статус пошуку та відповідна дата фільтрації. У якості залежності цього ефекту від даних передається властивість `status` об'єкту `filters`, яка зберігає статус замовлення.

Аналогічним чином здійснюється фільтрація за датою, проте у залежностях ефекту вже буде зберігатися властивість `date` об'єкту `filters`,

```

useEffect(() => {
    fetchMore({
        variables: {
            query: `financial_status:${filters.status.toLowerCase()},
created_at:>${formatISO(
                filters.date.start
            )} AND created_at:<${formatISO(addDays(filters.date.end,
1))}` ,

            },
        });
    }, [filters.date]);

```

У схожому порядку буде виконуватись пошук замовлення, проте у залежностях ефекту буде зберігатися властивість `search` об'єкту `filters`.

```
useEffect(() => {
  refetch({
    query:
      filters.search !== ""
        ? `financial_status:${filters.status.toLowerCase()},
name:${filters.search
      }, created_at:>${formatISO(
        filters.date.start
      )} AND
created_at:<${formatISO(addDays(filters.date.end, 1))}`
        : `financial_status:${filters.status.toLowerCase()},
created_at:>${formatISO(
        filters.date.start
      )} AND
created_at:<${formatISO(addDays(filters.date.end, 1))}`
    }, [filters.search]);
});
```

Пагінація на сторінку вперед та назад виконується за допомогою повного перезапису даних, який виконує директива `updateQuery`. Вона буде перезаписувати 10 замовлень на сторінці щоразу, коли користувач гортає таблицю замовлень вперед або назад.

```
const onPaginationNext = () => {
  setLoad((current) => ({
    ...current,
    right: true,
  }));
  fetchMore({
```

```

variables: {
  first: ORDERS_LIMIT,
  query: `financial_status:${filters.status.toLowerCase()}`,
created_at:>${formatISO(
  filters.date.start
)} AND created_at:<${formatISO(addDays(filters.date.end,
1))}` ,
  after: ordersData.edges[ordersData.edges.length -
1].cursor,
},
updateQuery: (previousResult, { fetchMoreResult }) => {
  if (!fetchMoreResult) return previousResult;
  const newOrders = fetchMoreResult.orders.edges;
  const pageInfo = fetchMoreResult.orders.pageInfo;
  const newCursor = newOrders[newOrders.length - 1].cursor;
  return {
    cursor: newCursor,
    orders: {
      pageInfo,
      edges: newOrders,
    },
  };
},
}).then((res) => {
  if (res) {
    setLoad((current) => ({
      ...current,
      right: false,
    }));
  }
});
setPage((current) => current + 1);
};

```

3.4 Висновки за розділом 3

В даному розділі описано основні компоненти розробленої системи.

Розділ містить опис схем даних, що отримуються за допомогою GraphQL, опис модульного шаблону інтерфейсу застосунку та його складових. Також наведено приклад реалізації основних функцій.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення програми

Вебзастосунок призначений для збору аналітичних даних замовлень інтернет-магазину на платформі Shopify. Він допомагає користувачу відслідковувати кількість отриманих замовлень за певний період часу, відображає графік для візуального розуміння аналітичних даних замовлень, виконує фільтрацію, пошук, сортування і пагінацію у таблиці замовлень, та показує історію та інформацію про замовлення на окремій сторінці застосунку.

4.2 Характеристики програми

Вебзастосунок розроблений за допомогою мови програмування JavaScript на стороні клієнта та сервера. Застосунок розроблений за архітектурою PWA, та працює за принципом Server Side Rendering, тобто відображає окремо кожну сторінку застосунку. На клієнті використовується фреймворк React.js – Next.js, на стороні сервера фреймворк Koa.js.

Застосунок виконує авторизацію інтернет-магазину користувача, зчитування та відображення даних про замовлення, відображає дані у вигляді графіку, виконує фільтрацію, сортування, пошук та пагінацію даних та надає можливість переглянути деталі окремого замовлення.

4.3 Вимоги до інформативних структур

Всі дані сайту зберігаються у системі Shopify Admin API. Доступ до цих даних виконується за допомогою технологій GraphQL та REST. Усі схеми та правила для отримання даних описані за синтаксисом GraphQL.

4.4 Звернення до програми

Звернення до вебзастосунку відбувається за допомогою запуску локального сервісу ngrok, який є тунелем між локальною машиною та платформою Shopify. Потім, у поле адреси браузера необхідно ввести адресу, яку навмання згенерував ngrok-тунель у командній консолі, щоб потрапити на сторінку авторизації інтернет-магазину користувача.

4.5 Вхідні та вихідні дані

4.5.1 Вхідні дані

Оскільки серверна частина застосунку виконує роль парсеру, застосунок не містить вхідних даних.

4.5.2 Вихідні дані

Вихідними даними програмного продукту є ті дані, які отримує клієнт з Shopify Admin API.

4.6 Розгортання вебзастосунку на Shopify Platform

Для того, щоб розмістити застосунок у системі Shopify Platform необхідно за допомогою терміналу операційної системи та консольного застосунку shopify cli, ввести команду авторизації до інтернет-магазину, де буде розміщений застосунок [14]:

```
shopify login [--store room-4-dev.myshopify.com]
```

Потім, у системі Shopify Partners, де зберігаються всі адміністровані інтернет-магазини, необхідно створити Shopify Custom App (рис. 4.1), дати йому назву та зберегти у системі.

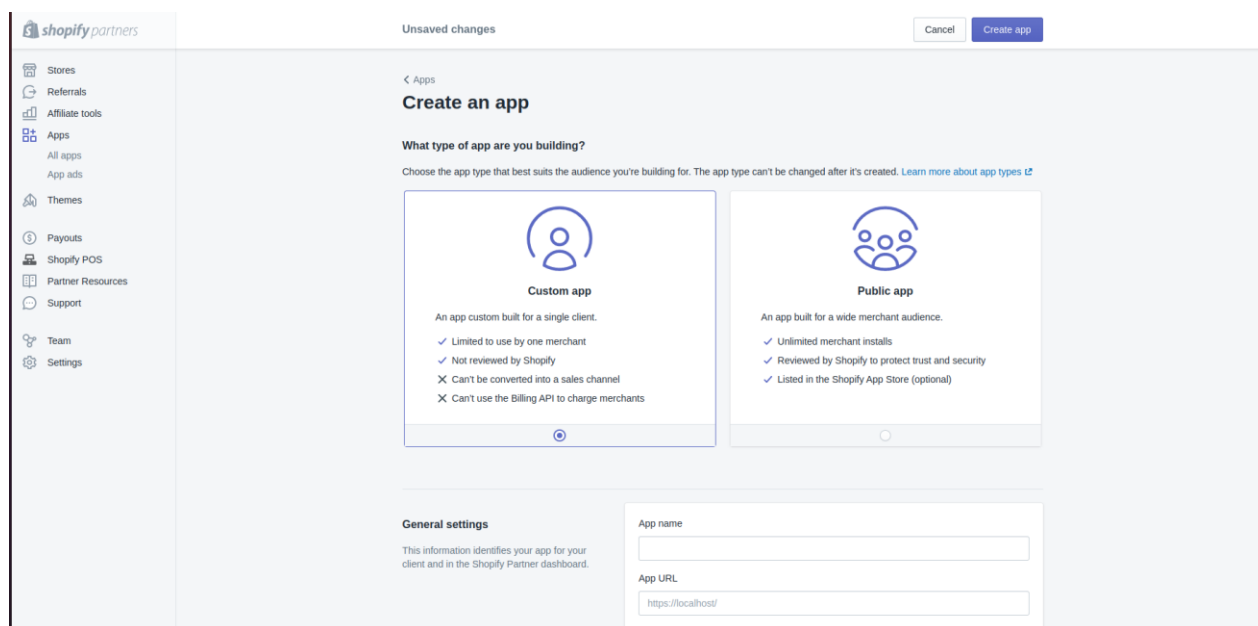


Рисунок 4.1 – Створення Shopify Custom App

Наступним кроком, необхідно ввести у консолі команду `shopify app connect`, яка допомагає зв'язати локально розгорнутий код вебзастосунку з фактично створеною програмою в системі Shopify Partners, у спадаючому списку з застосунками обрати необхідний.

Далі необхідно ввести команду `shopify app serve`. Ця команда допомагає працюючий мережевий тунель між програмою та Shopify Custom App, вона допомагає відобразити у браузері працюючий функціонал та інтерфейс, відслідковувати всі зміни, внесені розробником програми, і відповідно відображає це у інтерфейсі, окрім того у випадку помилок команда відображає це у браузері та у консолі. Дана команда є основною для запуску вебзастосунку у платформі Shopify.

4.7 Висновки за розділом 4

У цьому розділі було надано основні характеристики, призначення та умови експлуатації вебзастосунку.

Було розглянуто вимоги до інформативних структур до яких відбувається звернення від вебзастосунку та вхідні і вихідні дані проєкту.

Описаний процес звернення до програми та процес розгортання вебзастосунку у системі Shopify Platform.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

5.1 Призначення та умови застосування програми

5.1.1 Призначення програми

Вебзастосунок для збору аналітичних даних замовлень інтернет-магазину на платформі Shopify, призначений для відстеження замовлень, перегляду топ-5 замовлень, перегляду лінійного графіку здійснення замовлень клієнтами, фільтрації, сортування, пагінації та перегляду сторінки історії окремого замовлення.

5.1.2 Умови виконання програми

Для коректної роботи вебзастосунку для збору аналітичних даних замовлень інтернет-магазину на платформі Shopify необхідно мати браузер який буде підтримувати сумісність останніх версій стандартів JavaScript, HTML, CSS та реєстрацію на платформі Shopify, з привласненим інтернет- магазином який буде містити адміністративну панель, до якої можна приєднати застосунок. Застосунок можна запустити на будь-якій з нижче перелічених операційних систем:

- ОС Microsoft Windows версії 7 та більш пізні версії;
- Mac OS X;
- Linux Ubuntu.

Окрім того, необхідно мати будь-який з цих браузерів актуальних на даний момент часу версій:

- Google Chrome;
- Opera;
- Mozilla Firefox;
- Safari.

5.2 Характеристики програми

Вебзастосунок має можливість відслідковувати кількість отриманих замовлень за певний період часу, відображає графік для візуального розуміння аналітичних даних замовлень, виконує фільтрацію, пошук, сортування і пагінацію у таблиці замовлень, та показує історію та інформацію про замовлення на окремій сторінці застосунку.

5.3 Виконання програми

Доступ до застосунку відбувається за допомогою переходу за посиланням “Orders Test App” у розділі “Apps” адміністративної панелі Shopify (рис. 5.1). При переході за посиланням користувач потрапляє на головну сторінку вебзастосунку.

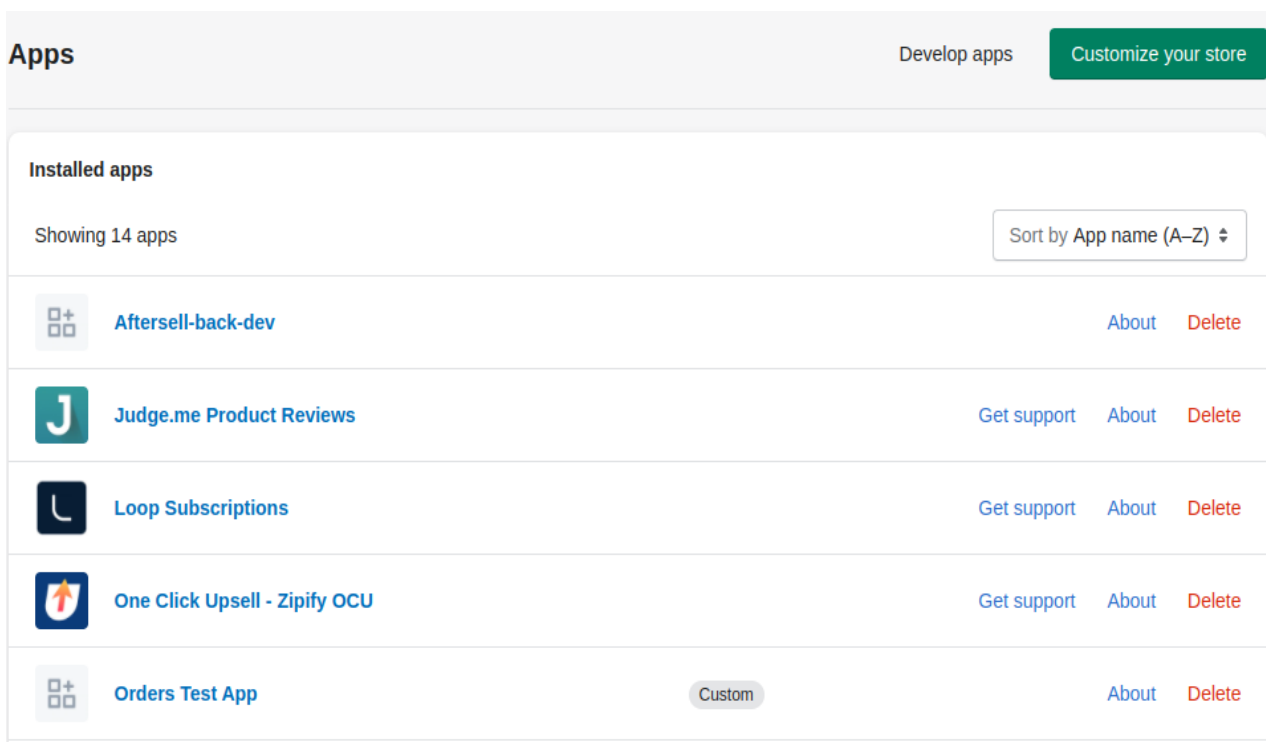


Рисунок 5.1 – Розділ “Apps” адміністративної панелі Shopify

5.4 Структура програми

5.4.1 Сторінка авторизації

Сторінка авторизації містить форму, в яку необхідно ввести адресу інтернет-магазину у текстове поле, щоб потрапити на головну сторінку вебзастосунку (рис.5.2).

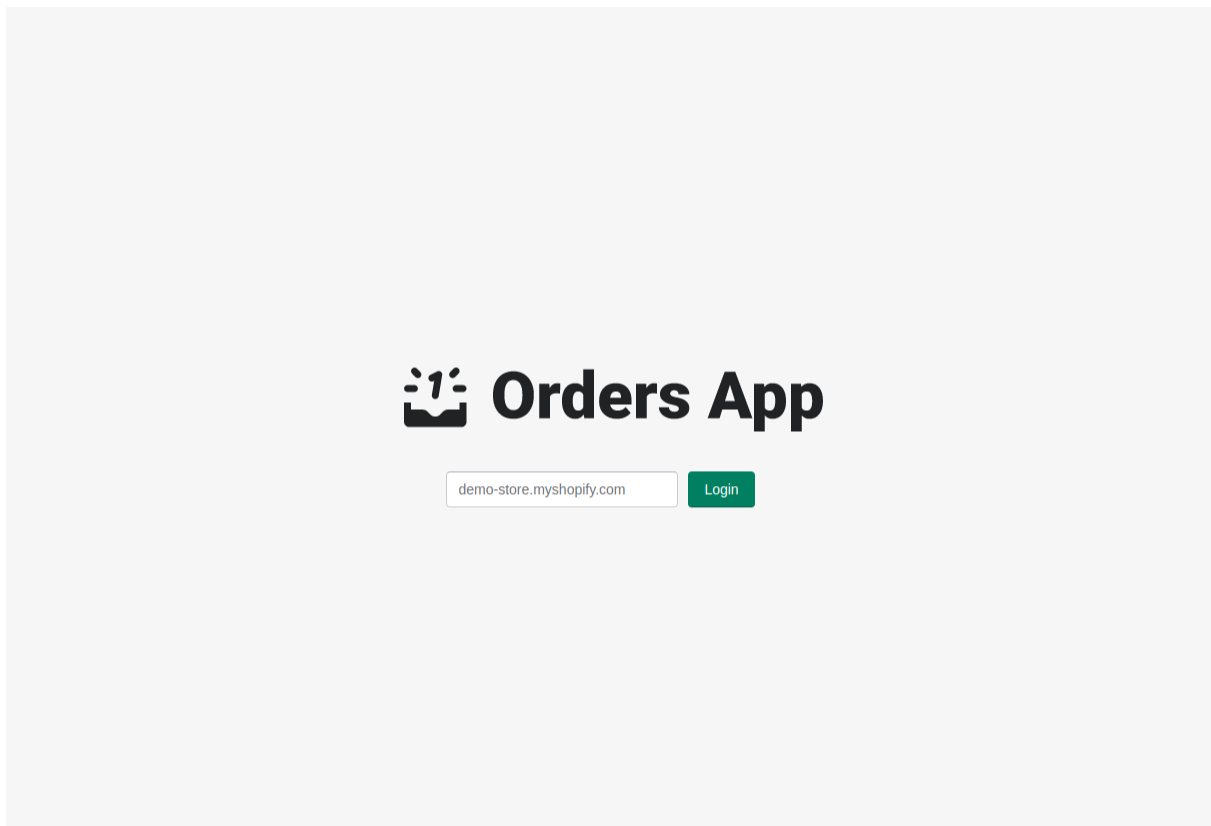


Рисунок 5.2 – Сторінка авторизації

5.4.2 Сторінка “Home”

Сторінка “Home” є головною сторінкою вебзастосунку збору аналітики інтернет-магазину на платформі Shopify. Сторінка містить два головні статичні елементи, які не змінюють свій вид протягом роботи всього застосунку: шапка та навігаційна панель.

Сторінка містить лінійний графік кількості отриманих магазином замовлень та таблицю зі списком п’яти товарів, що найчастіше купувались за місяць (рис 5.3).

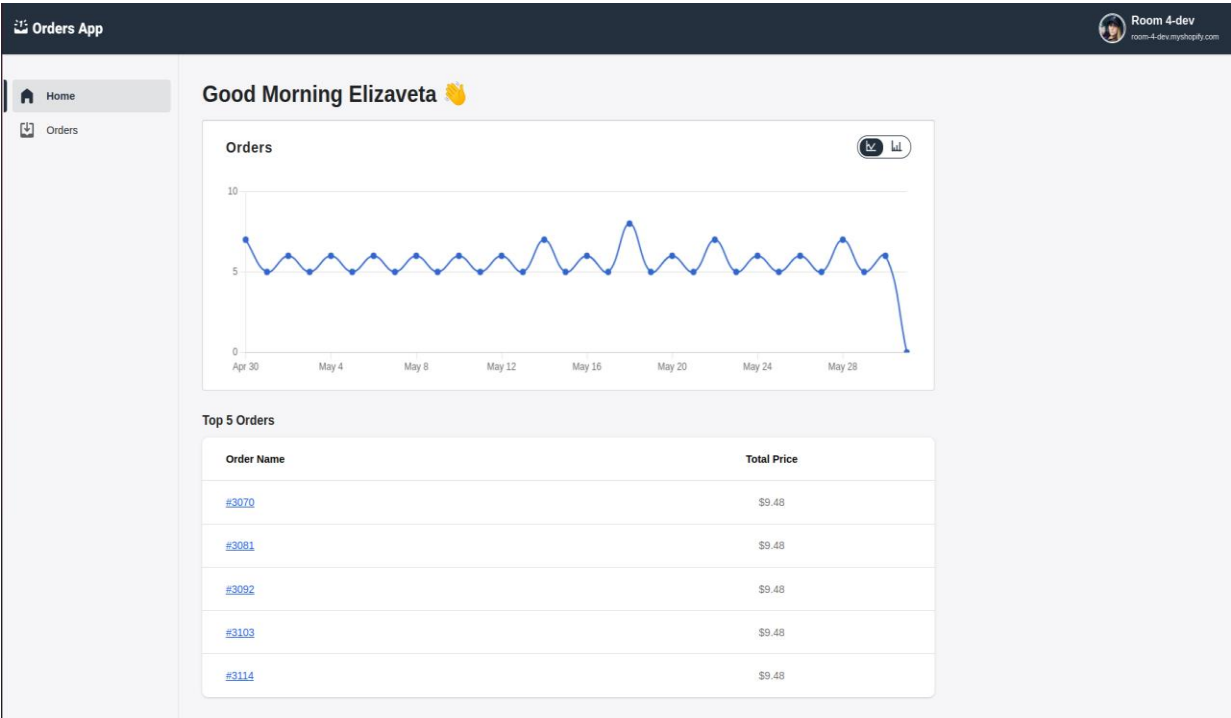


Рисунок 5.3 – Сторінка “Home”

Окрім того, користувач має змогу змінити вид графіку з лінійного графіку на стовпчасту діаграму, натиснувши на перемикач у правому верхньому куті відображення графіку (рис.5.4).

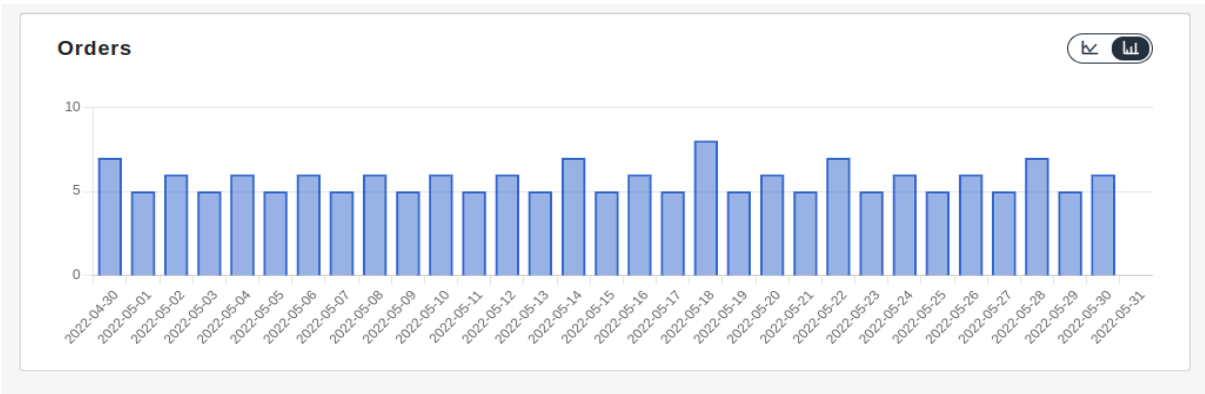


Рисунок 5.4 – Стовпчаста діаграма з даними кількості замовлень

5.4.3 Сторінка “Orders”

При переході за посиланням “Orders”, користувач потрапляє на сторінку (рис. 5.5), де можна переглянути таблицю замовлень, відфільтрувати за датою та за фінансовим статусом, відсортувати колонки за датою (від ранньої до пізньої та навпаки) та за кількістю (від найменшої до найбільшої та навпаки).

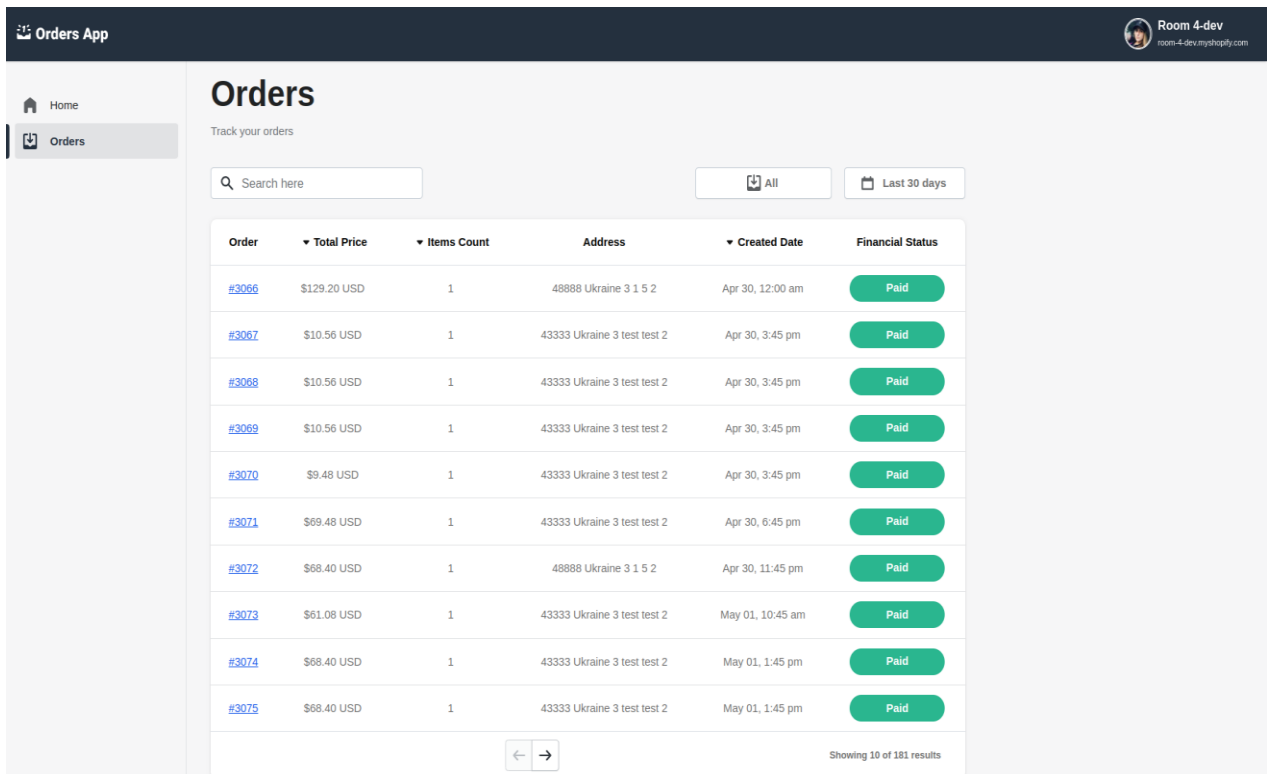


Рисунок 5.5 – Перегляд замовлень на сторінці “Orders”

У таблиці замовлення показуються по десять штук на сторінку, наступні десять замовлень можна подивитися за допомогою функціонала пагінації, натискаючи на стрілку вперед або назад.

Коли клієнт натискає на номер замовлення у колонці “Order”, маршрутизація застосунку переносить його на сторінку окремого замовлення.

5.5 Сторінка замовлення

Сторінка, зображена на рис. 5.6, дозволяє переглядати історію замовлення, кількість та вид замовлених товарів, адресу доставки, фінансовий статус, податковий збір, нотатки від покупця.

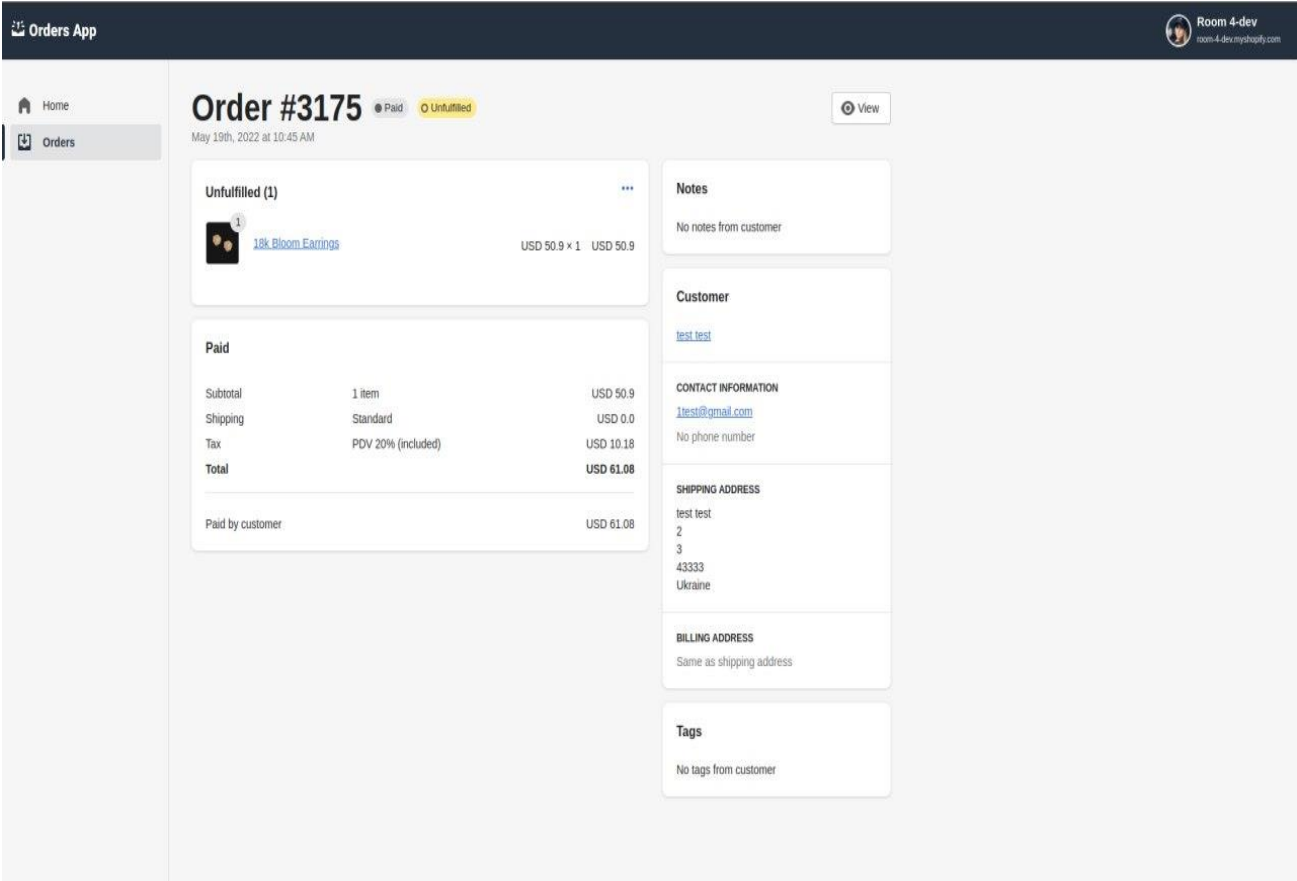


Рисунок 5.6 – Сторінка перегляду історії замовлення

5.6 Повідомлення користувачу

Інформаційне повідомлення для користувача виводиться лише у випадку невірної заповнення текстового поля на сторінці авторизації. Це поле має бути обов'язковим для заповнення та містити лише адресу магазину, у випадку некоректних даних користувач отримує повідомлення про помилку (рис. 5.7).

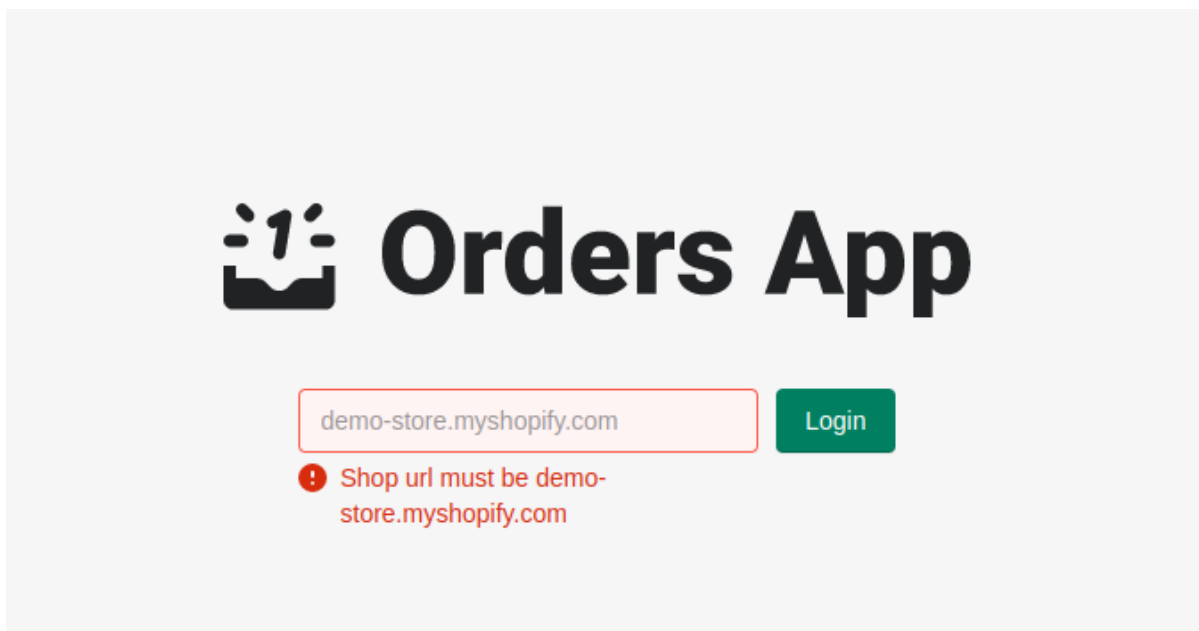


Рисунок 5.7– Повідомлення про некоректність введених даних

5.7 Висновки за розділом 5

В розділі було описано процес взаємодії користувача платформи Shopify з вебзастосунком, структуру сторінок та користувацькі дії з вебзастосунком.

Також було розглянуто процес отримання повідомлень для користувача під час експлуатації програмного продукту.

Інтерфейс розробленого вебдодатку є стандартизованим з користувацькими умовами розробки вебзастосунків для Shopify Platform та не викликає складнощів під час експлуатації користувачем.

ВИСНОВКИ

У рамках виконання дипломної кваліфікаційної роботи були вирішені наступні завдання:

- проведено аналіз предметної області, визначено недостатні можливості адміністративної панелі інтернет-магазину Shopify Platform пов'язані зі статистикою, проаналізовано існуючі вебзастосунки, створені для відслідковування статистики інтернет-магазинів, за результатами аналізу було обґрунтовано необхідність створити застосунок, який містить ключові властивості існуючих рішень;

- виконано проектування програмного продукту, обрано основні технології для розробки застосунку – мову програмування JavaScript на клієнтській та серверній частині вебзастосунку, зокрема фреймворк Next.js для написання клієнтської частини та Koa.js для серверної, розроблено схеми отримання даних за допомогою GraphQL, визначено архітектуру програмного продукту як PWA, яка безперешкодно підтримується платформою Shopify;

- виконано реалізацію та тестування програмного забезпечення.

Розроблений вебзастосунок дозволяє відслідковувати кількість отриманих замовлень за певний період часу, відображає графік для візуального розуміння аналітичних даних замовлень, виконує фільтрацію, пошук, сортування і пагінацію у таблиці замовлень, та показує історію та інформацію про замовлення на окремій сторінці застосунку.

Розробленою програмною реалізацією може користуватись будь-який власник або менеджер інтернет-магазину, створеного на платформі Shopify.

Проведені випробування показують, що створена програмна реалізація відповідає всім вимогам замовника та являється економічно доцільною для впровадження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Laudon, K.C. Traver E-commerce 2020-2021: Business, Technology and Society. 16th edition Global Edition / Kenneth C. Laudon Carol Guercio Traver. — Pearson, 2021. — 913 p.
2. Дем'янчук, К. Ф. Можливості Інтернету у формуванні, просуванні й реалізації продуктів та послуг / К. Ф. Дем'янчук, О. В. Орлик // Інформатика та інформаційні технології: студ. наук. конф., 20 квітня 2015 р. матер. конф. — Одеса, ОНЕУ. — С. 88-91.
3. Shopify admin overview [Electronic resource] – Access mode: <https://help.shopify.com/en/manual/shopify-admin/shopify-admin-overview>.
4. Profit Analytics & Benchmark [Electronic resource] – Access mode: <https://apps.shopify.com/conversific>.
5. Glew Analytics [Electronic resource] – Access mode: <https://apps.shopify.com/glew>.
6. Connector for Google Analytics [Electronic resource] – Access mode: <https://apps.shopify.com/littledata>.
7. StoreView [Electronic resource] – Access mode: <https://apps.shopify.com/store-view>.
8. Фримен, Э. Изучаем программирование на JavaScript: Учебник для вузов / Эрик Фримен, Элизабет Робсон. – М.:Питер Пресс, 2018. – 640 с.
9. Progressive Web Applications [Electronic resource] – Access mode: <https://habr.com/ru/post/418923/>.
10. Shopify App Overview [Electronic resource] – Access mode: <https://shopify.dev/apps/getting-started>.
11. Koa.js [Electronic resource] – Access mode: <https://koa.js.com>.
12. Next. [Electronic resource] – Access mode: <https://next.js.com>.
13. Shopify Polaris [Electronic resource] – Access mode: <https://polaris.shopify.com>.

14. Shopify CLI [Electronic resource] – Access mode:
<https://shopify.dev/apps/tools/cli/getting-started>.

ДОДАТОК А
Технічне завдання

A.1 Вступ

Вебзастосунок збору аналітики замовлень інтернет-магазину на платформі Shopify призначений для надання можливості відслідковувати статистику надходження замовлень адміністратору інтернет-магазину на базі даної платформи.

Застосунок надає можливість авторизації, синхронізованої з особистим інтернет-магазином.

A.2 Підстава для розроблення

Підставою для розробки є завдання на дипломний проєкт на тему «Вебзастосунок збору аналітики замовлень інтернет-магазину на платформі Shopify», затверджений наказом №102 від 27 квітня 2022 р.

A.3 Призначення розробки

Вебзастосунок збору аналітики замовлень на платформі Shopify призначений для надання можливості зручного відслідковування статистики замовлень власного інтернет-магазину. Вебзастосунок володіє необхідним функціоналом, що пов'язаний з фільтрацією замовлень за датою та фінансовим статусом, пошуком та сортуванням.

A.4 Основні вимоги до програмного забезпечення

A.4.1 Вимоги до функціональних характеристик

Розроблюваний вебзастосунок повинен виконувати такі функції:

- авторизацію синхронізовану з особистим інтернет-магазином;
- відображення діаграм;
- відображення п'яти найпродаваніших товарів;

- статистику замовлень;
- фільтрацію та сортування замовлень;
- сторінку одного замовлення при переході за посиланням.

A.4.2 Вимоги до надійності

Сервіс повинен розпізнавати ситуації нерозуміння сервісом користувача (користувачем програми), а також обробку збійних ситуацій.

A.4.3 Умови експлуатації

До експлуатаційних вимог на розроблювану програму входять:

- мережа Інтернет;
- зареєстрований інтернет-магазин у системі Shopify Platform;
- персональний комп'ютер;
- актуальний браузер.

A.4.4 Вимоги до складу та параметрів технічних засобів

Для експлуатації програмного продукту необхідні такі мінімальні програмно-технічні засоби як: ПК на апаратній платформі x86 під керуванням ОС сімейства WINDOWS (Windows 7/8/8.1/10), обсяг ОЗУ – не менш 256 Мб, обсяг вільного дискового простору – не менш 250 Мб, маніпулятор миша.

A.4.5 Вимоги до маркування та упакування

Вебзастосунок збору аналітики замовлень на платформі Shopify може бути упакований у віддалений git-репозиторій. На якій будуть знаходитись всі необхідні файли з кодом та документацією.

A.4.6 Вимоги до транспортування та збереження

Вимоги до транспортування й зберігання аналогічні вимогам, пропонуваним системою віддаленого git-репозиторія, на якому зберігається програма.

A.4.7 Вимоги до програмної документації

Вебзастосунок повинен бути з керівництвом користувача, керівництвом програміста.

A.5 Стадії та етапи розробки

A.5.1 Етап розробки аналіз вимог

Це етап на якому збирається вся необхідна інформація для розробки сервісу. Робиться на наліз існуючих програмних систем.

A.5.2 Етап розробки проєктування

На етапі проєктування відбувається розробка користувацького інтерфейсу та проєктування схем отримання даних.

A.5.3 Етап розробки програмування і верстка

Це основний етап розробки на якому відбувається розробка компонентів сторінок вебзастосунку, проходять перші тести на реальних даних, код може дороблятися, фіксуються помилки розроблюваного сервісу.

A.5.4 Етап розробки запуск і супровід

Це завершальний етап розробки на якому формується на відправлення замовникові вся документація, у вебзастосунок вноситься необхідний контент, вибирається певна платформа, та весь проєкт відсилається замовникові для тестування.

A.5.5 Етап розробки закриття проєкту

На цьому етапі замовник повідомляє після своїх тестів на своєму обладнанні, що все працює та він згоден підписати акт передачі проєкту та отримує від замовника ПЗ та документацію.

A.6 Порядок контролю та приймання системи

Замовник повинен провести тест на власному технічному забезпеченні. У разі знайдення помилок чи не відповідності ТЗ виконавець зобов'язується виправити всі помилки у певний термін. Якщо тестування було успішним замовник підписує акт передачі проєкту та отримує від замовника ПЗ та документацію.

ДОДАТОК Б
Текст програми

Б.1 Текст програми пов'язаний з обробкою даних на сервері

Б.1.1 Текст файлу server.js

```
import "@babel/polyfill";
import dotenv from "dotenv";
import Koa from "koa";
import cors from "@koa/cors";
import next from "next";
import Router from "koa-router";
import createShopifyAuth, { verifyRequest } from "@shopify/koa-shopify-auth";
import Shopify, { ApiVersion } from "@shopify/shopify-api";

dotenv.config();
const port = parseInt(process.env.PORT, 10) || 8081;
const dev = process.env.NODE_ENV !== "production";
const app = next({
  dev,
});
const handle = app.getRequestHandler();

Shopify.Context.initialize({
  API_KEY: process.env.SHOPIFY_API_KEY,
  API_SECRET_KEY: process.env.SHOPIFY_API_SECRET,
  SCOPES: process.env.SCOPE.split(","),
  HOST_NAME: process.env.HOST.replace(/https:\/\/|\/$/g, ""),
  API_VERSION: ApiVersion.Unstable,
  IS_EMBEDDED_APP: false,
  SESSION_STORAGE: new Shopify.Session.MemorySessionStorage(),
});

// Storing the currently active shops in memory will force them to re-
login when your server restarts. You should
```

```

// persist this object in your app.
export const ACTIVE_SHOPIFY_SHOPS = {};

app.prepare().then(async () => {
  const server = new Koa();
  const router = new Router();
  server.keys = [Shopify.Context.API_SECRET_KEY];

  const options = {
    origin: "*",
  };

  server.use(cors(options));

  server.use(
    createShopifyAuth({
      async afterAuth(ctx) {
        // Access token and shop available in ctx.state.shopify
        const { shop, accessToken, scope } = ctx.state.shopify;
        const host = ctx.query.host;
        ACTIVE_SHOPIFY_SHOPS[shop] = scope;

        const response = await Shopify.Webhooks.Registry.register({
          shop,
          accessToken,
          path: "/webhooks",
          topic: "APP_UNINSTALLED",
          webhookHandler: async (topic, shop, body) =>
            delete ACTIVE_SHOPIFY_SHOPS[shop],
        });

        if (!response.success) {
          console.log(

```

```

        `Failed to register APP_UNINSTALLED webhook:
        ${response.result}`
    );
}

    // Redirect to app with shop parameter upon auth
    ctx.redirect(`/?shop=${shop}&host=${host}`);
},
})
);

const handleRequest = async (ctx) => {
    await handle(ctx.req, ctx.res);
    ctx.respond = false;
    ctx.res.statusCode = 200;
};

router.get(
    "/",
    verifyRequest({ accessMode: "online", fallbackRoute: "/login" }),
    async (ctx) => {
        const session = await Shopify.Uutils.loadCurrentSession(
            ctx.req,
            ctx.res,
            true
        );
        const shop = ctx?.query?.shop?.toString() || session.shop;

        if (session && session.isOnline) {
            await handleRequest(ctx);
        } else {
            return ctx.redirect(`/auth?shop=${shop}`);
        }
    }
}

```

```

);

router.get("/login", async (ctx) => {
  const session = await Shopify.Utils.loadCurrentSession(
    ctx.req,
    ctx.res,
    true
  );

  if (session && session.isOnline && session.expires) {
    return ctx.redirect(`/?shop=${session.shop}`);
  }
  await handleRequest(ctx);
});

router.post(
  "/graphql",
  verifyRequest({ accessMode: "online", returnHeader: true }),
  async (ctx, next) => {
    await Shopify.Utils.graphqlProxy(ctx.req, ctx.res);
  }
);

router.get("/api/statistic", async (ctx) => {
  try {
    const session = await Shopify.Utils.loadCurrentSession(ctx.req,
ctx.res);
    const client = new Shopify.Clients.Rest(
      session.shop,
      session.accessToken
    );
    const data = await client.get({
      path: "orders/count",
      query: {

```

```

        status: "any",
        created_at_min: `${ctx.query.created_at_min}`,
        created_at_max: `${ctx.query.created_at_max}`,
      },
    ));
    ctx.body = data;
    ctx.status = 200;
  } catch (err) {
    ctx.status = err.statusCode || err.status || 500;
    ctx.body = {
      message: err.message,
    };
  }
});

router.get("/_next/static/.*", handleRequest); // Static content is
clear
router.get("/_next/webpack-hmr", handleRequest); // Webpack content is
clear

router.get("/_next/image", handleRequest);

router.get(
  "(.*)",
  verifyRequest({ accessMode: "online", fallbackRoute: "/login" }),
  handleRequest
);

server.use(router.allowedMethods());
server.use(router.routes());
server.listen(port, () => {
  console.log(`> Ready on http://localhost:${port}`);
});
});

```

Б.2 Текст програми пов'язаний з обробкою даних на клієнті

Б.2.1 Текст файлу DefaultLayout.js.

```
import React, { useCallback, useState } from "react";
import TopBarMarkup from "../../components/TopBar";
import NavigationMarkup from "../../components/Navigation";
import { Frame, Page } from "@shopify/polaris";
import styles from "../styles.module.scss";

const DefaultLayout = ({ children }) => {
  const [mobileNavigationActive, setMobileNavigationActive] =
    useState(false);
  const toggleMobileNavigationActive = useCallback(
    () =>
      setMobileNavigationActive(
        (mobileNavigationActive) => !mobileNavigationActive
      ),
    []
  );
  const topBarMarkup = (
    <TopBarMarkup
      toggleMobileNavigationActive={toggleMobileNavigationActive} />
  );
  return (
    <Frame
      topBar={topBarMarkup}
      navigation={<NavigationMarkup />}
      showMobileNavigation={mobileNavigationActive}
      onNavigationDismiss={toggleMobileNavigationActive}
    >
      <div className={styles.PageWrapper}>
        <Page>{children}</Page>
      </div>
    </Frame>
  );
}
```

```

    </Frame>
  );
};

export default DefaultLayout;

```

Б.2.2 Текст файла Home.js

```

import React, { useState } from "react";
import DefaultLayout from "../../layouts/DefaultLayout";
import TopTable from "../../components/TopTable";
import MainTitle from "../../components/MainTitle";
import ChartView from "../../components/Chart";
import { Layout, Banner } from "@shopify/polaris";
import { useQuery } from "@apollo/client";
import { GET_TOP_ORDERS } from
"../../graphql/queries/getTopOrders.schema";
import Cut from "../../helpers/cut";
import { formatTopOrdersData } from "../../helpers/formatTopOrdersData";
import { addDays, formatISO, setMonth } from "date-fns";

const Home = () => {
  const [currentDate, setCurrentDate] = useState({
    date: {
      start: setMonth(new Date(), new Date().getMonth() - 1),
      end: new Date(),
    },
  });
  const { data, loading, error } = useQuery(GET_TOP_ORDERS, {
    variables: {
      query: `created_at:>${formatISO(
        currentDate.date.start
      )} AND created_at:<${formatISO(addDays(currentDate.date.end,
1))}` ,

```

```

    },
  ));
  const orders = Cut(data?.orders);
  const rows = formatTopOrdersData(orders);
  if (error) {
    return (
      <Banner title="Something went wrong" status="critical">
        <p>{error}</p>
      </Banner>
    );
  }
  return (
    <DefaultLayout>
      <Layout>
        <Layout.Section fullWidth>
          <MainTitle />
        </Layout.Section>
        <Layout.Section fullWidth>
          <ChartView title={"Orders"} type="line" />
        </Layout.Section>
        <Layout.Section fullWidth>
          <TopTable rows={rows} loading={loading} />
        </Layout.Section>
      </Layout>
    </DefaultLayout>
  );
};
export default Home;

```

Б.2.3 Текст файла Login.js

```

import React from "react";
import styles from "../styles.module.scss";

```

```

import Image from "next/image";
import LoginForm from "../../components/LoginForm";

const Login = () => {
  return (
    <div className={styles.LoginWrapper}>
      <Image
        src="/images/main-logo.svg"
        alt="Orders App"
        width={451}
        height={69}
      />
      <LoginForm />
    </div>
  );
};

export default Login;

```

Б.2.4 Текст файла Orders.js

```

import React, { useEffect, useState, useMemo } from "react";
import DefaultLayout from "../../layouts/DefaultLayout";
import {
  DisplayText,
  TextStyle,
  Layout,
  Spinner,
  Card,
  Banner,
} from "@shopify/polaris";
import SearchFilters from "../../components/Filters";
import OrdersTable from "../../components/OrdersTable";
import { useQuery } from "@apollo/client";

```

```

import { GET_ORDERS } from "../../graphql/queries/getOrders.schema";
import Cut from "../../helpers/cut";
import { format, formatISO, setMonth, addDays } from "date-fns";
import styles from "./styles.module.scss";
import { formatOrdersData } from "../../helpers/formatOrdersData";
import { ORDERS_LIMIT, status } from "../../constants";
import OrdersPagination from "../../components/OrdersTablePagination";

const Orders = () => {
  const [page, setPage] = useState(1);
  const [load, setLoad] = useState({
    left: false,
    right: false,
  });

  const [filters, setFilters] = useState({
    search: "",
    status: status.All,
    date: {
      start: setMonth(new Date(), new Date().getMonth() - 1),
      end: new Date(),
    },
  });

  const { data, loading, fetchMore, error, refetch } =
    useQuery(GET_ORDERS, {
      variables: {
        first: ORDERS_LIMIT,
        query: `created_at:>${formatISO(
          filters.date.start
        )} AND created_at:<${formatISO(addDays(filters.date.end, 1))}`,
      },
    });
};

```

```

useEffect(() => {
  setPage(1);
}, [filters]);

useEffect(() => {
  fetchMore({
    variables: {
      query: `financial_status:${filters.status.toLowerCase()},
created_at:>${formatISO(
  filters.date.start
)} AND created_at:<${formatISO(addDays(filters.date.end, 1))}` ,
    },
  });
}, [filters.date]);

useEffect(() => {
  let search = filters.search !== "" ? `name:${filters.search}` : "";
  refetch({
    query: `financial_status:${filters.status.toLowerCase()},
${search}, created_at:>${formatISO(
  filters.date.start
)} AND created_at:<${formatISO(addDays(filters.date.end, 1))}` ,
  });
}, [filters.status]);

useEffect(() => {
  refetch({
    query:
      filters.search !== ""
      ? `financial_status:${filters.status.toLowerCase()}, name:${
        filters.search
      }, created_at:>${formatISO(
        filters.date.start

```

```

        }) AND created_at:<${formatISO(addDays(filters.date.end,
1)))}`
        : `financial_status:${filters.status.toLowerCase()}`,
created_at:>${formatISO(
        filters.date.start
        }) AND created_at:<${formatISO(addDays(filters.date.end,
1)))}` ,
    });
    }, [filters.search]);

const memoizedCallback = useMemo(() => {
    if (data !== undefined) {
        return page !== 1
            ? (page - 1) * 10 + data.orders.edges.length
            : page * data.orders.edges.length;
    }
}, [data]);

const ordersData = data?.orders;
const orders = Cut(ordersData);
const rows = formatOrdersData(orders);

const onSearch = (value) => {
    setFilters((current) => ({
        ...current,
        search: value,
    }));
};

const onSortingStatus = (value) => {
    setFilters((current) => ({
        ...current,
        status: value,
    }));
};

```

```

};

const onDateFilter = (value) => {
  setFilters((current) => ({
    ...current,
    date: value,
  }));
};

const onPaginationNext = () => {
  setLoad((current) => ({
    ...current,
    right: true,
  }));
  fetchMore({
    variables: {
      first: ORDERS_LIMIT,
      query: `financial_status:${filters.status.toLowerCase()}`,
      created_at:>${formatISO(
        filters.date.start
      )} AND created_at:<${formatISO(addDays(filters.date.end, 1))}`,
      after: ordersData.edges[ordersData.edges.length - 1].cursor,
    },
    updateQuery: (previousResult, { fetchMoreResult }) => {
      if (!fetchMoreResult) return previousResult;
      const newOrders = fetchMoreResult.orders.edges;
      const pageInfo = fetchMoreResult.orders.pageInfo;
      const newCursor = newOrders[newOrders.length - 1].cursor;
      return {
        cursor: newCursor,
        orders: {
          pageInfo,
          edges: newOrders,
        },
      },
    },
  });
};

```

```

    };
  },
}).then((res) => {
  if (res) {
    setLoad((current) => ({
      ...current,
      right: false,
    }));
  }
});
setPage((current) => current + 1);
};

const onPaginationPrev = () => {
  setLoad((current) => ({
    ...current,
    left: true,
  }));
  fetchMore({
    query: GET_ORDERS,
    variables: {
      last: ORDERS_LIMIT,
      query: `financial_status:${filters.status.toLowerCase()}`,
      created_at:>${formatISO(
        filters.date.start
      )} AND created_at:<${formatISO(addDays(filters.date.end, 1))}`,
      before: ordersData.edges[0].cursor,
    },
  },
  updateQuery: (previousResult, { fetchMoreResult }) => {
    if (!fetchMoreResult) return previousResult;
    const newOrders = fetchMoreResult.orders.edges;
    const pageInfo = fetchMoreResult.orders.pageInfo;
    const newCursor = newOrders[0].cursor;
    return {

```

```

        cursor: newCursor,
        orders: {
            pageInfo,
            edges: newOrders,
        },
    };
},
}).then((res) => {
    if (res) {
        setLoad((current) => ({
            ...current,
            left: false,
        }));
    }
});
setPage((current) => (current > 1 ? current - 1 : current));
};

if (error) {
    return (
        <Banner title="Something went wrong" status="critical">
            <p>{error}</p>
        </Banner>
    );
}

return (
    <DefaultLayout>
        <Layout>
            <Layout.Section fullWidth>
                <div className={styles.TextWrapper}>
                    <DisplayText size={"extraLarge"}>Orders</DisplayText>
                    <TextStyle variation="subdued">Track your orders</TextStyle>
                </div>
            </Layout.Section>
        </Layout>
    </DefaultLayout>
);

```

```

</Layout.Section>
<Layout.Section fullWidth>
  <SearchFilters
    onSearch={onSearch}
    onSortingStatus={onSortingStatus}
    onDateFilter={onDateFilter}
    selectedDates={filters.date}
  />
  <div className={styles.cardWrapper}>
    <Card>
      {loading ? (
        <div className={styles.loading}>
          <Spinner size="large" />
        </div>
      ) : (
        <div className={styles.mainContent}>
          <OrdersTable initiallySortedRows={rows} />
          <OrdersPagination
            createdAtMin={format(filters.date.start, "yyyy-MM-
dd")}

            createdAtMax={format(
              addDays(filters.date.end, 1),
              "yyyy-MM-dd"
            )}
            hasNext={ordersData.pageInfo.hasNextPage}
            onPaginationNext={onPaginationNext}
            onPaginationPrev={onPaginationPrev}
            hasPrev={ordersData.pageInfo.hasPreviousPage}
            ordersCount={memoizedCallback}
            loading={load}
          />
        </div>
      )}
    </Card>
  </div>

```

```

        </div>
      </Layout.Section>
    </Layout>
  </DefaultLayout>
);
};

```

```
export default Orders;
```

Б.2.5 Текст файла Order.js

```

import React from "react";
import DefaultLayout from "../../layouts/DefaultLayout";
import {
  DisplayText,
  Layout,
  Button,
  TextStyle,
  Banner,
} from "@shopify/polaris";
import { ViewMajor } from "@shopify/polaris-icons";
import { useRouter } from "next/router";
import { useQuery } from "@apollo/client";
import { GET_ORDER } from "../../graphql/queries/getOrder.schema";
import PaymentBadge from "../../components/PaymentBadge";
import FulfillmentBadge from "../../components/FulfillmentBadge";
import styles from "./styles.module.scss";
import { format, parseISO } from "date-fns";
import FulStatusCard from "../../components/FulStatusCard";
import NoteCard from "../../components/NoteCard";
import TagsCard from "../../components/TagsCard";
import CustomerCard from "../../components/CustomerCard";
import PaidCard from "../../components/PaidCard";

```

```

const Order = () => {
  const router = useRouter();
  const { data, loading, error } = useQuery(GET_ORDER, {
    variables: {
      id: `gid:///shopify/Order/${router.query.id}`,
    },
  });
  if (error) {
    return (
      <Banner title="Something went wrong" status="critical">
        <p>{error}</p>
      </Banner>
    );
  }
  return (
    <DefaultLayout>
      {!loading && (
        <Layout>
          <Layout.Section fullWidth>
            <div className={styles.orderHeader}>
              <div className={styles.titleWrapper}>
                <DisplayText size={"extraLarge"}>
                  Order {data?.order?.name}
                </DisplayText>
                <PaymentBadge
status={data?.order?.displayFinancialStatus} />
                <FulfillmentBadge
                  status={data?.order?.displayFulfillmentStatus}
                />
              </div>
              <Button
                icon={ViewMajor}
                onClick={() =>
                  router.push(

```

```

`https://${router.query.shop}/admin/orders/${router.query.id}`
    )
  }
>
  View
</Button>
</div>
<TextStyle variation="subdued">
  {format(parseISO(data?.order?.createdAt), "PPPp")}
</TextStyle>
</Layout.Section>

<Layout.Section>
  <FullStatusCard order={data?.order} router={router} />
  <PaidCard order={data?.order} />
</Layout.Section>

<Layout.Section secondary>
  <NoteCard note={data?.order?.note} />
  <CustomerCard
    customer={data?.order?.customer}
    shippingAddress={data?.order?.shippingAddress}
    router={router}
  />
  <TagsCard tags={data?.order?.tags} />
</Layout.Section>
</Layout>
  )}
</DefaultLayout>
);
};
export default Order;

```

Б.2.6 Текст файла Chart.js

```
import React, { createElement, useEffect, useState } from "react";
import { OrdersCount } from "../FetchAction";
import styles from "../styles.module.scss";
import { SkeletonDisplayText, Spinner } from "@shopify/polaris";
import "chartkick/chart.js";
import { LineChart, ColumnChart } from "react-chartkick";
import ChartSwitcher from "../ChartSwitcher";

const chart = {
  line: LineChart,
  column: ColumnChart,
};

const ChartView = ({ title, type }) => {
  const [data, setData] = useState([]);
  const [isLoading, setIsLoading] = useState(true);
  const [activeToggleType, setActiveToggleType] = useState(type);
  useEffect(() => {
    if (!sessionStorage.getItem("orders-count")) {
      OrdersCount().then((v) => {
        sessionStorage.setItem("orders-count", JSON.stringify(v));
        setData(v);
        setIsLoading(false);
      });
    } else {
      setData(JSON.parse(sessionStorage.getItem("orders-count")));
      setIsLoading(false);
    }
  }, []);
  const typeChangeHandler = () =>
    setActiveToggleType((currentType) =>
      currentType === "line" ? "column" : "line"
    );

```

```

    );
    const sortByDate = (a, b) => {
        return new Date(a[0]) - new Date(b[0]);
    };
    return (
        <div className={styles.Chart}>
            <div className={styles.optionsWrapper}>
                <div className={styles.Chart__title}>
                    {isLoading ? <SkeletonDisplayText size="medium" /> : title}
                </div>
                <div className={styles.Chart__switcher}>
                    {!isLoading && (
                        <ChartSwitcher
                            name={activeToggleType}
                            onChange={typeChangeHandler}
                        />
                    )}
                </div>
            </div>
            {!isLoading ? (
                chart.hasOwnProperty(activeToggleType) &&
                createElement(chart[activeToggleType], {
                    data: activeToggleType === "line" ? data :
data.sort(sortByDate),
                    height: "235px",
                })
            ) : (
                <Spinner accessibilityLabel="Spinner example" size="large" />
            )}
        </div>
    );
};

export default ChartView;

```

Б.2.7 Текст файлу FetchAction.js

```
import { getDays } from "../../helpers/getDays";
import { fetchAction } from "../../helpers/fetchCount";

export const OrdersCount = async () => {
  const days = getDays();
  const OrdersDate = [];
  for (let i = 0; i < days.length; i++) {
    const params = {
      created_at_min: days[i],
      created_at_max: days[i + 1] || days[i],
    };
    const response = await fetchAction(params);
    OrdersDate.push([days[i], response.data.body.count]);
  }
  return OrdersDate;
};
```

Б.2.8 Текст файлу Filters.js

```
import React, { useEffect, useState } from "react";
import { Icon, TextField } from "@shopify/polaris";
import { SearchMinor } from "@shopify/polaris-icons";
import styles from "./styles.module.scss";
import DatePickerView from "../DatePicker";
import SortingPicker from "../SortingPicker";
import { useDebounce } from "../../helpers/useDebounce";

const SearchFilters = ({
  onSearch,
  onSortingStatus, onDateFilter,
  selectedDates,
```

```

})) => {
  const [search, setSearch] = useState("");
  const debouncedSearch = useDebounce(search, 700);
  useEffect(() => {
    onSearch(debouncedSearch.trim());
  }, [debouncedSearch]);
  return (
    <div className={styles.FiltersWrapper}>
      <div className={styles.FiltersWrapper__search}>
        <TextField
          type="search"
          value={search}
          onChange={(value) => setSearch(value)}
          prefix={<Icon source={SearchMinor} />}
          placeholder="Search here"
          autoComplete="off"
        />
      </div>
      <div className={styles.FiltersWrapper__sortingPicker}>
        <SortingPicker onChange={onSortingStatus} />
      </div>
      <div className={styles.FiltersWrapper__datePicker}>
        <DatePickerView date={selectedDates} onChange={onDateFilter} />
      </div>
    </div>
  );
};
export default SearchFilters;

```

Б.2.9 Текст файлу OrdersTable.js

```

import React, { useCallback, useState } from "react";
import { DataTable, EmptyState } from "@shopify/polaris";
import styles from "../styles.module.scss";

```

```

import { sortData } from "../../helpers/sortData";

const OrdersTable = ({ initiallySortedRows }) => {
  const [sortRows, setSortRows] = useState([]);
  const rows = initiallySortedRows;
  const handleSort = useCallback(
    (index, direction) => setSortRows(sortData(rows, index, direction)),
    [rows]
  );
  return (
    <div className={styles.OrdersTable}>
      {rows.length > 0 ? (
        <DataTable
          columnContentTypes={[
            "text",
            "numeric",
            "numeric",
            "text",
            "text",
            "text",
          ]}
          headings={[
            "Order",
            "Total Price",
            "Items Count",
            "Address",
            "Created Date",
            "Financial Status",
          ]}
          verticalAlign="middle"
          rows={rows}
          sortable={[false, true, true, false, true, false]}
          defaultSortDirection="descending"
          initialSortColumnIndex={4}

```

```
        onSort={handleSort}
      />
    ) : (
      <EmptyState heading="No orders" />
    )}
  </div>
);
};

export default OrdersTable;
```

ДОДАТОК В
Слайди презентації

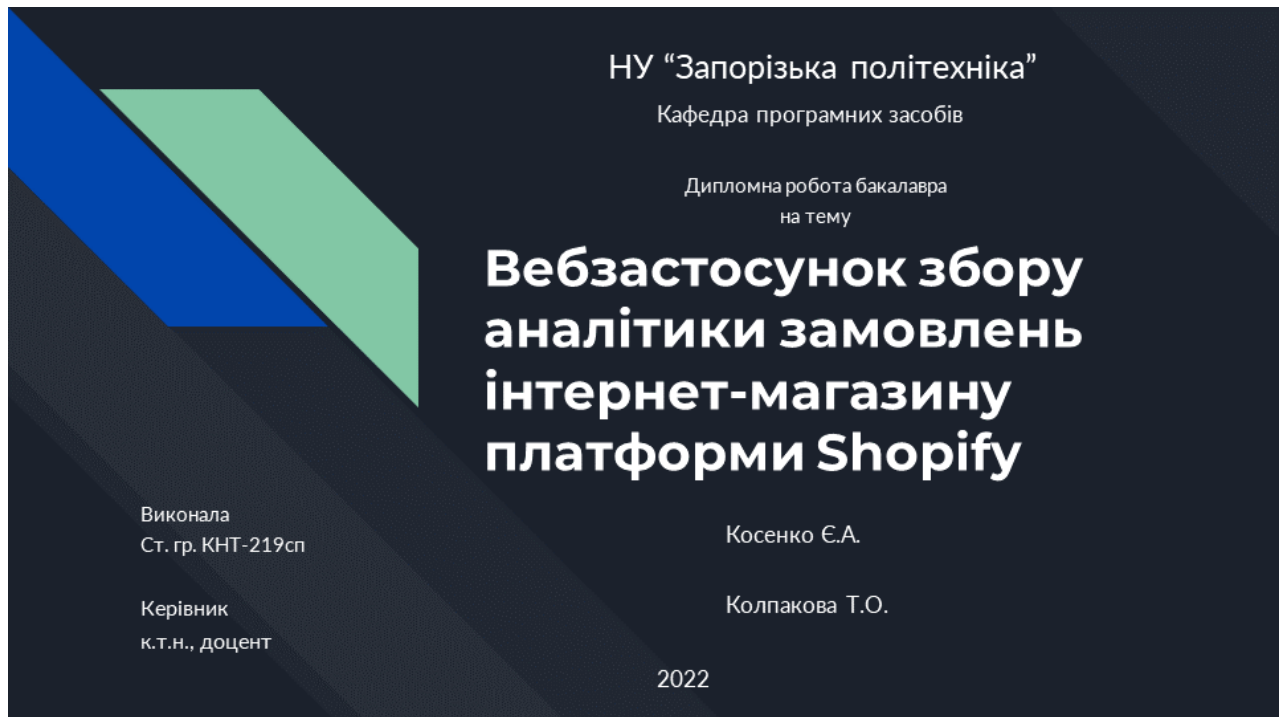


Рисунок В.1 – Слайд № 1

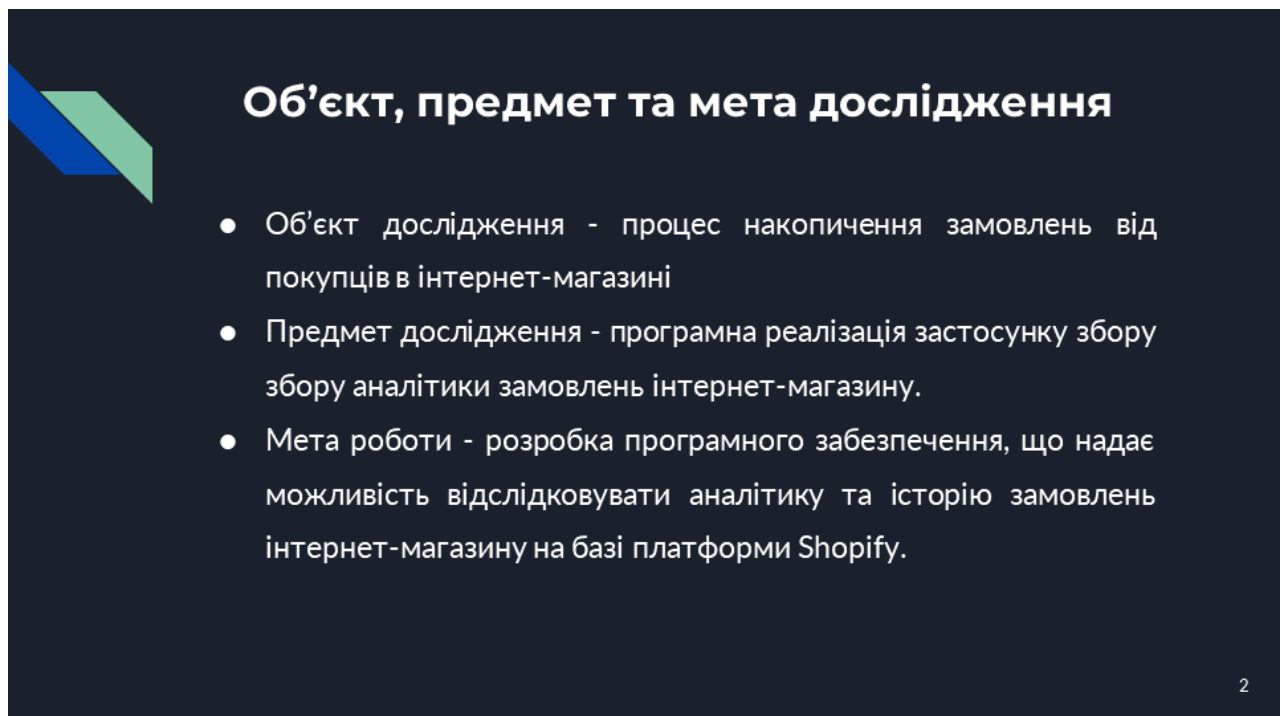
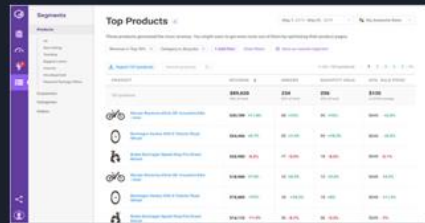


Рисунок В.2 – Слайд №2

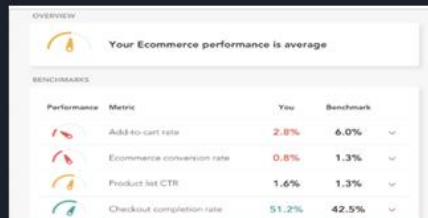
Огляд існуючих програмних засобів



Profit Analytics & Benchmark



Glew Analytics



Connector for Google Analytics

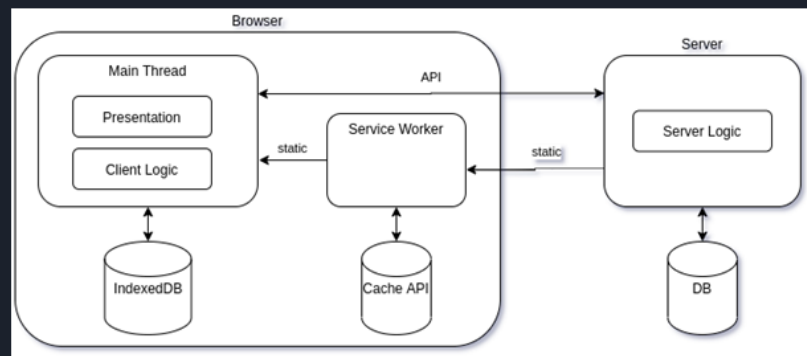


StoreView

3

Рисунок В.3 – Слайд №3

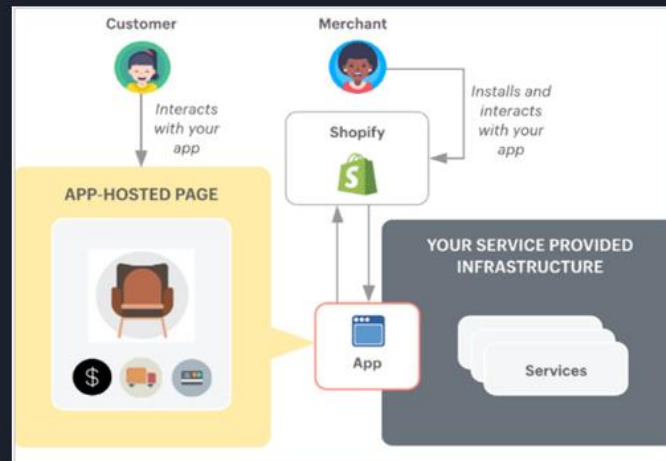
Архітектура PWA



4

Рисунок В.4 – Слайд №4

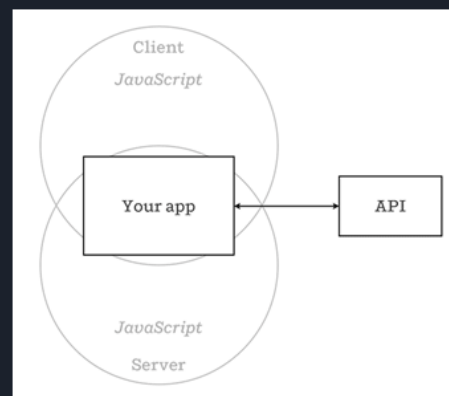
Схема роботи Shopify App Template



5

Рисунок В.5 – Слайд №5

Схема роботи клієнт-серверного JavaScript



6

Рисунок В.6 – Слайд №6



Переваги	Недоліки
Популярні інструменти, такі як React, Material-UI, Builder, Sanity та DatoCMS, сумісні з Next.js.	Нижча продуктивність у порівнянні з іншими фреймворками.
Автоматичний рендеринг на стороні сервера та розбиття коду	Створений спеціально для роботи з React.js.
Дозволяє створювати динамічні метатегу у розділі <head> для оптимізації SEO.	Представлений набір певних алгоритмів та інструментів, які можуть відрізнятися від використовуваних у React. Розробнику доведеться адаптуватися до цього.
У Next.js є гаряче перезавантаження сторінки при виявленні будь-якої зміни.	Через те, що фреймворк достатньо молодий не дуже розвинений інформаційний простір.

7

Рисунок В.7 – Слайд №7



Схеми отримання даних

Схема даних для отримання інформації про магазин

```
{
  shop {
    id
    myshopifyDomain
    name
    email
    url
  }
}
```

Схема даних для отримання масиву замовлень

```
orders {
  edges {
    node {
      id
      name
      currencyCode
      totalPriceSet {
        shopMoney {
          amount
        }
      }
      shippingAddress {
        zip
        city
        country
        name
        address1
      }
      lineItems {
        edges {
          node {
            quantity
          }
        }
      }
      createdAt
      displayFinancialStatus
    }
  }
}
```

Схема даних отримання топ-5 замовлень

```
{
  orders(first: 5, sortKey: TOTAL_PRICE, query: $query) {
    edges {
      node {
        id
        name
        currencyCode
        totalPriceSet {
          shopMoney {
            amount
          }
        }
      }
    }
  }
}
```

8

Рисунок В.8 – Слайд №8

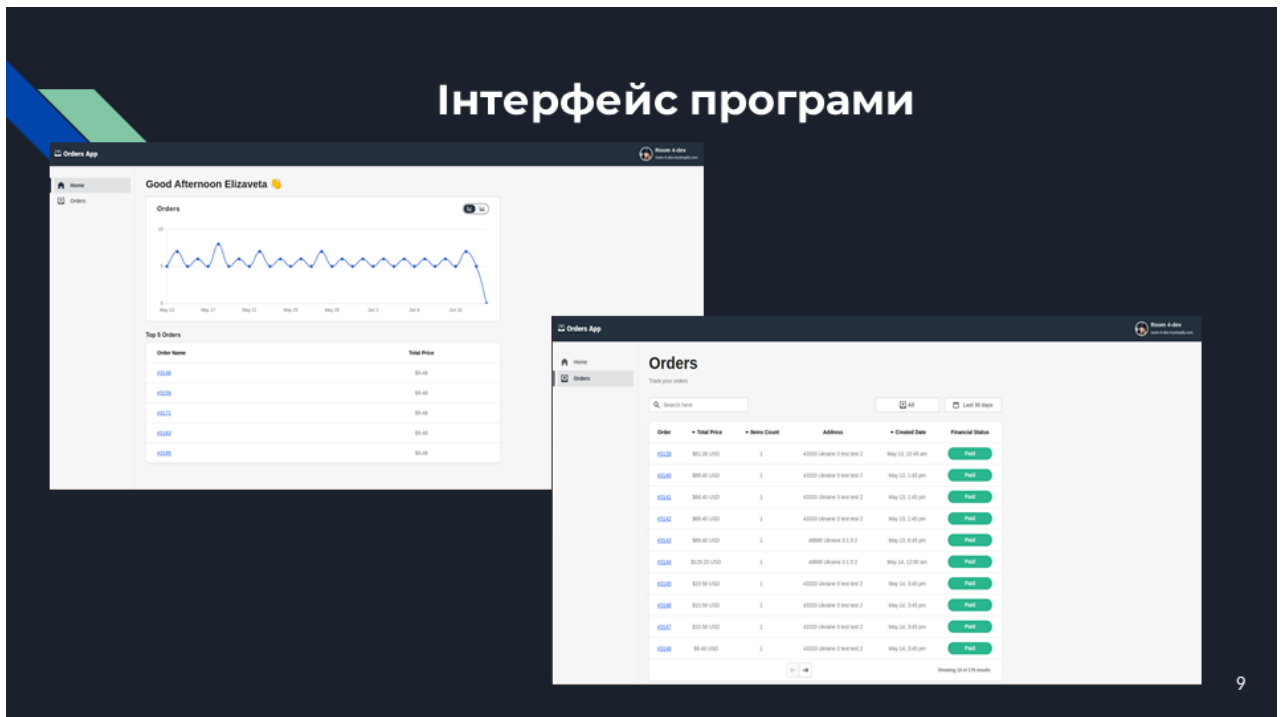


Рисунок В.9 – Слайд №9

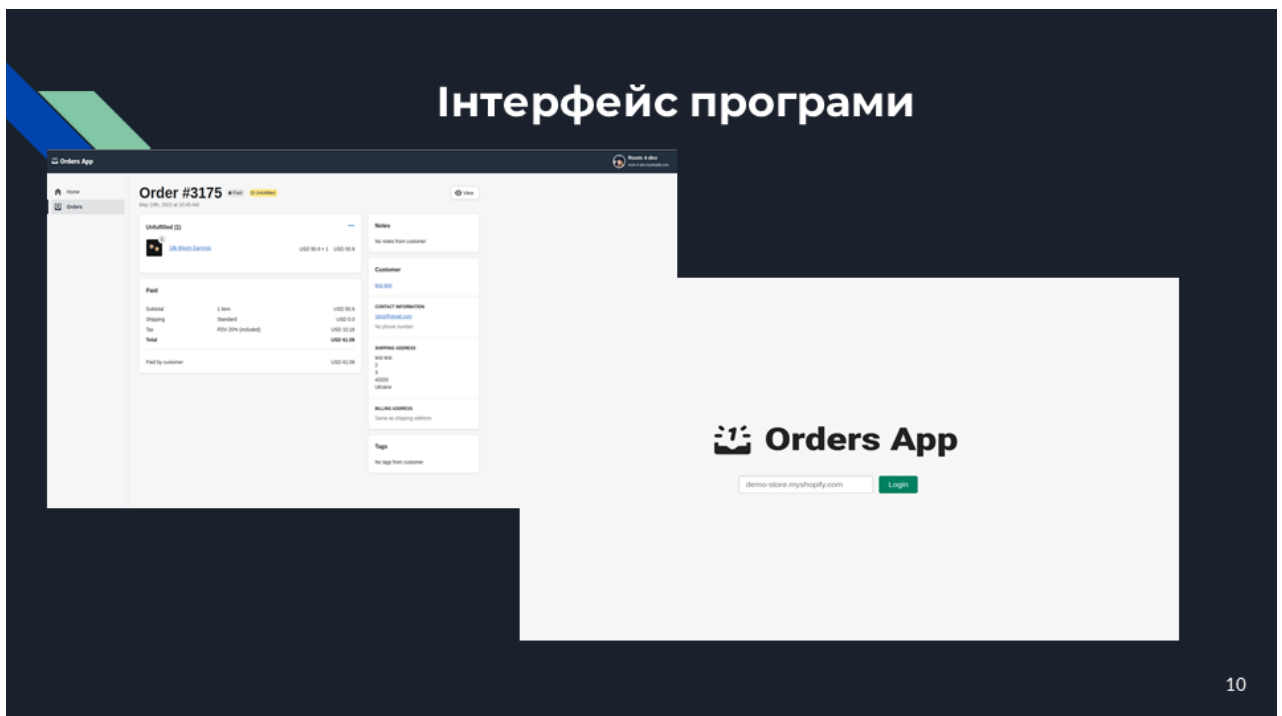


Рисунок В.10 – Слайд №10



Висновки

У рамках виконання дипломної кваліфікаційної роботи були вирішені наступні завдання:

- проведено аналіз предметної області, визначено недостатні можливості адміністративної панелі інтернет-магазину Shopify Platform пов'язані зі статистикою
- виконано проектування програмного продукту, обрано основні технології для розробки застосунку
- виконано реалізацію та тестування програмного забезпечення.

11

Рисунок В.11 – Слайд №11



Дякую за увагу!

12

Рисунок В.12 – Слайд №12