МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В.Н.Каразіна
Факультет математики і інформатики
Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

магістр

на тему: Застосування блокчейн-технологій у системах управління ланцюгами постачання

Виконав: студент 2 курсу, групи МФ-61
спеціальність 122 «Комп'ютерні науки»
освітньо-наукова програма
«Інформатика»

_____Цибулько В. О._____
(прізвище та ініціали)
Керівник ___Фролов В. В._____
(прізвище та ініціали)
Консультант ____Бережна Н. І._____
(прізвище та ініціали)

Рецензент_____
(прізвище та ініціали)

Харків – 2025 року

ЗМІСТ

ВСТУП

1.1 Мета і завдання роботи

Метою і завданням цієї дипломної роботи було розроблення веб-сервісу з застосуванням блокчейн-технологій, що дозволяють швидко та ефективно керувати ланцюгами постачання. Також блокчейн-технології забезпечують прозорість транзакцій, безпеку та автоматизацію процесів. За мету було обрано створення моделі, яка об’єднує продавців і покупців у ланцюзі постачання в єдину систему блокчейну. Це зменшить витрати на оперативні процеси та спростить взаємодію. Платформа надає можливість додавати, продавати та купувати товари, залишати на них відгуки та відстежувати процес відправки та доставки товару.

Завданням розробки є:

- Впровадження реєстрації та аутентифікації учасників ланцюга постачання, таких як компанії-виробники, дистриб’ютори, роздрібні продавці та покупці, з використанням криптогаманців для ідентифікації.
- Інтеграція блокчейну як розподіленого реєстру. Він відповідальний за фіксацію інформації угод, процесів транспортування, доставки та оплати; також блокчейн забезпечує прозорість та безпеку системи.
- Розробка API для здійснення фінансових транзакцій між учасниками, автоматизації та підтвердження угод.
- Розробка зручного веб-інтерфейсу для взаємодії із застосунком, що дозволить юзерам відстежувати замовлення, користуватися онлайн-магазином та залишати відгуки на товари.
- Надати можливість покупцям і продавцям писати відгуки про товари, що покращуватиме взаємодію між учасниками та підвищуватиме рівень довіри в системі.

- Запровадити функцію додавання та продажу товарів, що дозволить продавцям, компаніям-виробникам та дистриб’юторам продавати свою продукцію через платформу з використанням блокчейну.

Таким чином, метою цієї роботи є створення ефективної та безпечної системи для управління ланцюгами постачання, що сприятиме зменшенню бюрократії та запобіганню шахрайству. Система базується на новітніх блокчейн-технологіях і забезпечує підвищену прозорість, контроль на кожному етапі процесу та нові можливості для продажу товарів і взаємодії між учасниками.

1.2 Актуальність роботи

Актуальність даної дипломної роботи зумовлена сучасними проблемами у процесах управління ланцюгами постачання. Традиційні системи мають багато недоліків, які включають погану прозорість операцій, велику кількість помилок через роздуту систему бюрократії, а також ризики шахрайства. У таких умовах виникає потреба у вдосконаленні процесів та створенні ефективних рішень, здатних знизити витрати і підвищити рівень довіри між усіма учасниками ланцюга постачання.

Гарною альтернативою є використання блокчейн-технологій. Блокчейн-технології мають низку переваг перед традиційними реалізаціями ланцюгів постачання. А саме — надійне та безпечне зберігання даних без необхідності втручання посередників, що дозволяє знизити витрати на адміністративні процеси. Прозорість операцій, що робить систему більш надійною, сприятиме збільшенню довіри користувачів. Використання блокчейн-технологій як розподіленого реєстру для фіксації даних про товарні партії, етапи транспортування, доставки та оплату значно спрощує і прискорює процеси, роблячи їх більш ефективними та зручними для учасників ланцюга постачання.

Особливу увагу варто приділити питанням безпеки у цифрових транзакціях та ідентифікації учасників ланцюга постачання. Інтеграція гаманців для додаткової ідентифікації підвищує рівень безпеки, запобігаючи шахрайству, несанкціонованому доступу та дозволяючи здійснювати безпечні й анонімні транзакції. Блокчейн-технології гарантують, що всі дії в системі фіксуються в незмінному реєстрі, що значно зменшує можливість маніпуляцій із даними.

Також важливою складовою сучасних систем є зручний пошук товарів та інформації. Створення функціоналу для швидкого й ефективного пошуку значно підвищує комфорт кінцевих користувачів. Веб-сервіс має бути оптимізований для миттєвого доступу до інформації про товар, включно з історією переміщень, статусами доставок, цінами та наявністю на складі. Впровадження фільтрів та категоризації дозволяє покупцям швидко знаходити потрібні товари, що скорочує час на прийняття рішення та покращує досвід користувачів.

Інтеграція децентралізованого сервісу із використанням блокчейн-технологій є актуальною на даний момент. Це оптимізує бізнес-процеси, збільшує ефективність, швидкість та надійність системи.

Також можна зазначити, що блокчейн-технології є частиною глобальної тенденції, спричиненої розповсюдженням криптовалют і їхнім впровадженням у життя дедалі більшої кількості людей. Це дозволить збільшити кількість користувачів платформи, що позитивно сприятиме просуванню сервісу для управління ланцюгами постачання, в який інтегровано блокчейн-технології для прозорості та зручності користування.

Таким чином, дослідження є надзвичайно актуальним у контексті розвитку сучасних бізнес-процесів, де постійно зростає потреба в ефективних і надійних рішеннях для управління ланцюгами постачання. Впровадження блокчейн-технологій дозволяє розв’язати існуючі проблеми та створити умови для прозорого, безпечного й ефективного обміну інформацією та фінансовими ресурсами між учасниками. Створення зручної платформи для продажу товарів, написання відгуків та забезпечення безпеки

всіх транзакцій робить систему більш ефективною, доступною та конкурентоспроможною на ринку.

РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

2.1. Постановка задачі

Дипломна робота спрямована на створення веб-сервісу для управління ланцюгами постачання із застосуванням технології блокчейн. Такий підхід має на меті забезпечити максимальну прозорість, безпеку та ефективність взаємодії між усіма учасниками процесу. Розроблена система повинна включати можливість реєстрації користувачів, додавання і продаж товарів, а також зберігання всіх транзакцій у блокчейні, що сприятиме формуванню довіри між сторонами та автоматизації процесів через API.

Для досягнення поставленої мети передбачається виконання таких основних завдань:

- Створення веб-сервісу для реєстрації учасників постачального ланцюга — реалізація функціоналу реєстрації для різних типів користувачів: виробників, дистриб’юторів, продавців і покупців.
- Інтеграція блокчейн-технологій для зберігання даних про товари — впровадження блокчейну як децентралізованого реєстру для запису інформації про переміщення товарів, їх доставку та оплату. Використання публічного блокчейну дозволить гарантувати прозорість та незмінність даних.
- Розробка фінансового модуля через API — створення механізму для автоматичних фінансових розрахунків між учасниками, підтвердження операцій і оптимізації розрахункових процесів.
- Побудова інтерфейсу для перегляду історії постачання — створення зручного візуального середовища для відстеження переміщення товарів, змін у власності та верифікації транзакцій через блокчейн.
- Реалізація механізму продажу товарів із використанням блокчейну — забезпечення можливості розміщення товарів на платформі,

оформлення угод купівлі-продажу, з подальшим оновленням інформації у системі.

- Проведення тестування та валідації системи — перевірка працездатності всіх складових системи, включаючи реєстрацію, транзакції та їх збереження у блокчейні, а також забезпечення безпечної взаємодії між користувачами.

Реалізація цього проекту дозволить вирішити основні проблеми, пов’язані з управлінням ланцюгами постачання: покращити прозорість, підвищити рівень довіри, знизити ризики шахрайства та мінімізувати помилки при обробці інформації. Завдяки блокчейну можна автоматизувати фінансові процеси й забезпечити надійне зберігання даних, що суттєво підвищить ефективність і безпеку всієї системи.

2.2 Огляд та обґрунтування використаних технологій

Для реалізації цієї дипломної роботи, яка включає в себе управління ланцюгами постачання з використанням блокчейн-технологій. Було використано сучасні технології та інструменти для реалізації цього проекту, що виконало важливу роль у написанні надійної та ефективної програми.

- PostgreSQL виступає в ролі основної реляційної бази даних, яка гарантує надійність та масштабованість системи. Вона підтримує складну логіку запитів і транзакцій, що дає змогу безпечно зберігати дані про користувачів, товари та їхні взаємодії, дотримуючись принципів ACID для збереження цілісності інформації[1].
- Node.js у поєднанні з Express забезпечує серверну частину застосунку. Завдяки асинхронній природі Node.js система здатна ефективно обробляти численні одночасні запити, що важливо для роботи у реальному часі. Express, у свою чергу, спрощує розробку REST API та інтеграцію з іншими модулями, включно з блокчейн-мережею[2].

- React разом із Vite використовуються для створення клієнтської частини застосунку. React дозволяє швидко і гнучко створювати інтерфейси, які миттєво реагують на зміни без повного перезавантаження сторінки. А Vite пришвидшує процес розробки та збирання проєкту, завдяки сучасним методам оптимізації[3].
- Docker застосовується для контейнеризації компонентів системи — сервера, клієнта та бази даних. Це забезпечує ізоляцію середовищ, що полегшує тестування, запуск і масштабування сервісу в різних умовах без втрати стабільності[4].
- ngrok надає можливість швидко з’єднати локальний сервер з Інтернетом через безпечний тунель. Такий підхід значно полегшує тестування взаємодії API із зовнішніми сервісами, зокрема блокчейн-мережею, без необхідності складної мережевої конфігурації.
- TypeScript додає до JavaScript статичну типізацію, що дозволяє виявляти помилки ще до запуску програми. Завдяки чіткій типізації змінних і функцій стає простіше підтримувати, тестувати та масштабувати проєкт, що підвищує його надійність[5].

У висновку можна відзначити що стек обраних технологій відповідає вимогам проекту, а саме допомагає в реалізації безпечної, швидкої та децентралізованої програми. Це дозволяє створити надійну систему управління ланцюгами постачання, адаптовану до сучасних технічних стандартів.

2.3 Опис функціональності системи

Функціональні можливості системи були реалізовані з урахуванням потреб як покупця, так і продавця. У процесі розробки інтерфейсу для покупця було прийнято рішення розширити функціонал, додавши наступні можливості:

1. Прив’язка гаманця:

Користувач має можливість під’єднати свій криптогаманець до облікового запису. Цей функціонал включає:

- Тестова транзакція — можливо виконати після прив’язки для перевірки коректності підключення.
- Оновлення гаманця — користувач може змінити або оновити дані гаманця при потребі.
- Відв’язка гаманця — у будь-який момент користувач може відключити гаманець від облікового запису.

2. Перегляд магазину:

Користувачам доступний перегляд товарів у магазині. Додаткові функції цього розділу включають:

- Сортування — можливість впорядковувати товари за ціною або категорією.
- Пошук за назвою — швидкий пошук потрібного товару за його назвою. Також є можливість переходити на окрему сторінку товару для перегляду детальної інформації.

3. Перегляд замовлень:

Покупець може відстежувати деталі власних замовлень. Додатково передбачена функція:

- Підтвердження отримання — підтвердження факту доставки замовлення.

4. Налаштування облікового запису:

- Користувач має змогу переглядати й змінювати персональні дані, редагувати налаштування профілю та оновлювати інформацію про себе.

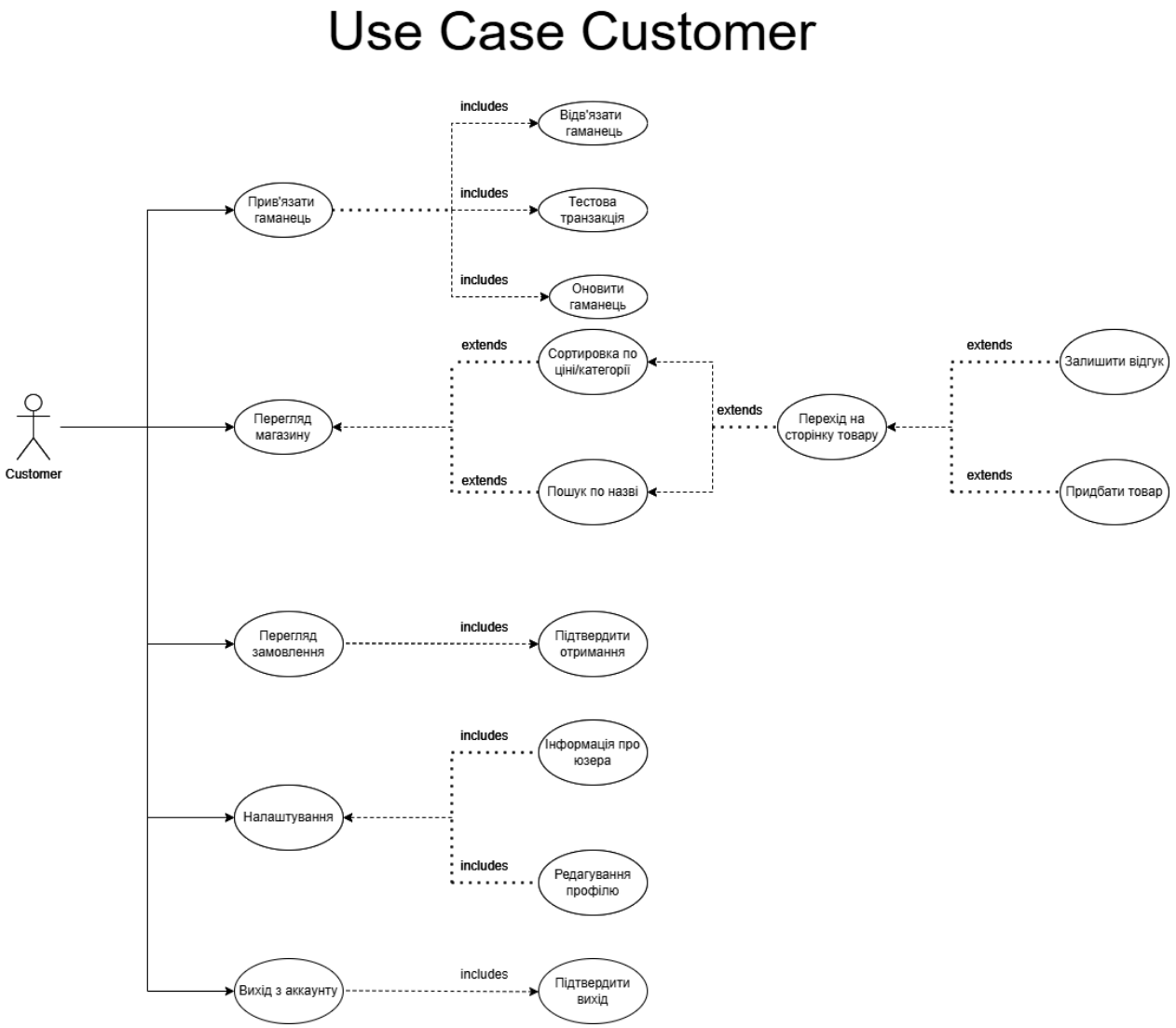


Рис. 1. Use Case Customer

Функціонал для продавця був розроблений з урахуванням можливостей покупця, а також з додатковими функціями, щоб забезпечити повний контроль над процесами продажу. Функціонал продавця був розширений наступними можливостями:

1. Перегляд замовлення:

- Підтвердження відправлення замовлення: Продавець може підтвердити відправлення замовлення.
- Підтвердження отримання коштів: Продавець може підтвердити, що платіж був успішно отриманий.

2. Історія товарів:

- Перегляд історії замовлень: Продавець може перевірити свої минулі замовлення, включаючи деталі завершених покупок.

3. Додавання товарів:

- Заповнення інформації про товар: Продавець вводить необхідні дані для кожного товару, такі як:
    - Назва товару
    - Опис товару
    - Ціна
    - Кількість в наявності
    - Категорія товару
    - Країна походження та країна знаходження товару
    - Фотографії товару: Продавець може додавати фотографії товарів для полегшення вибору покупцями.

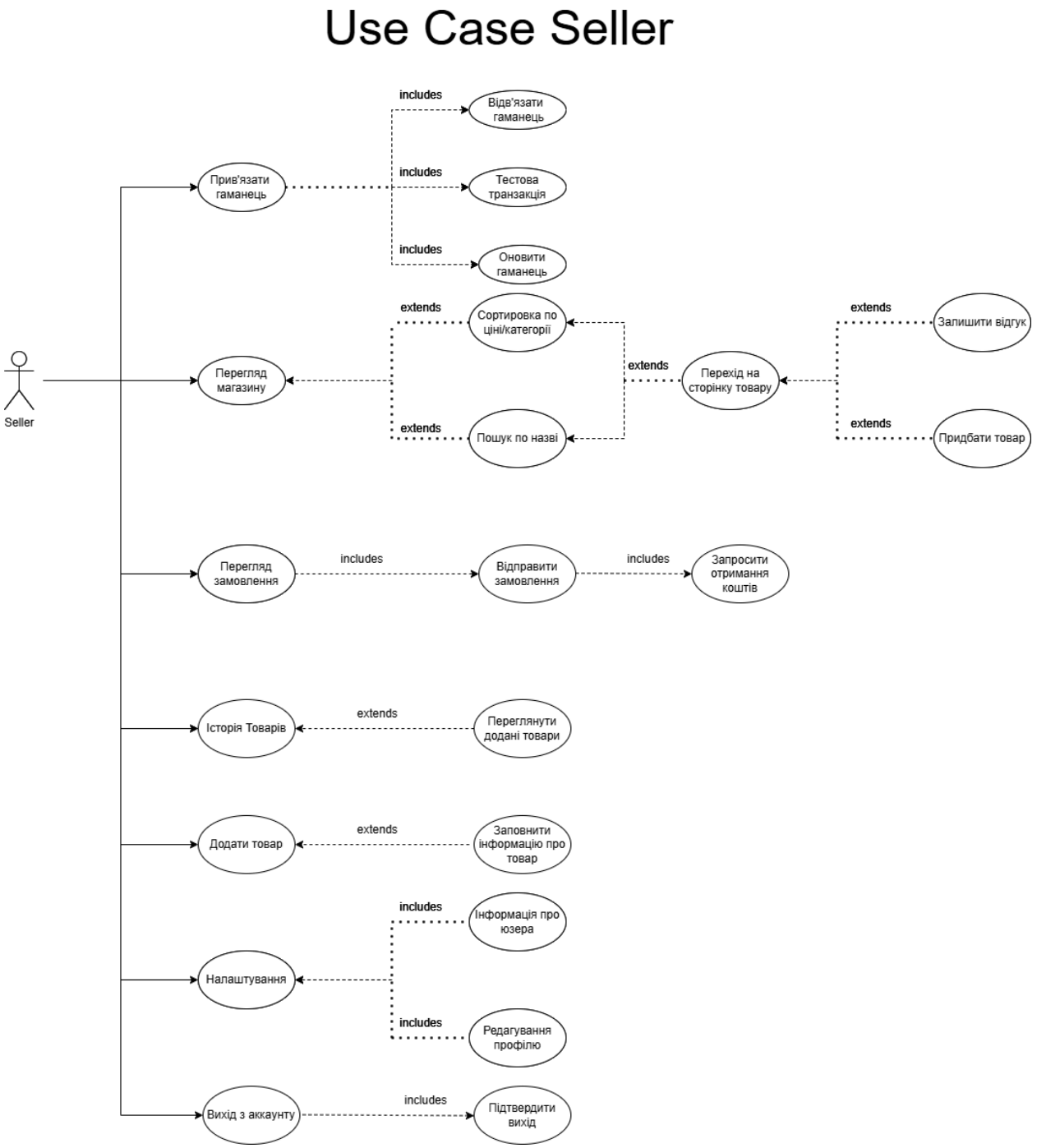


Рис. 2. Use Case Seller

2.4 Покроковий опис роботи застосунку

Посилаючись на пункт 2.3, було розроблено покроковий опис роботи застосунку. Знаючи всі функціональності системи, робота кожної з них була продумана для кожного користувача. Почнемо з покупця:

При вході у застосунок незареєстрований користувач має змогу перейти до вікна реєстрації та авторизації.

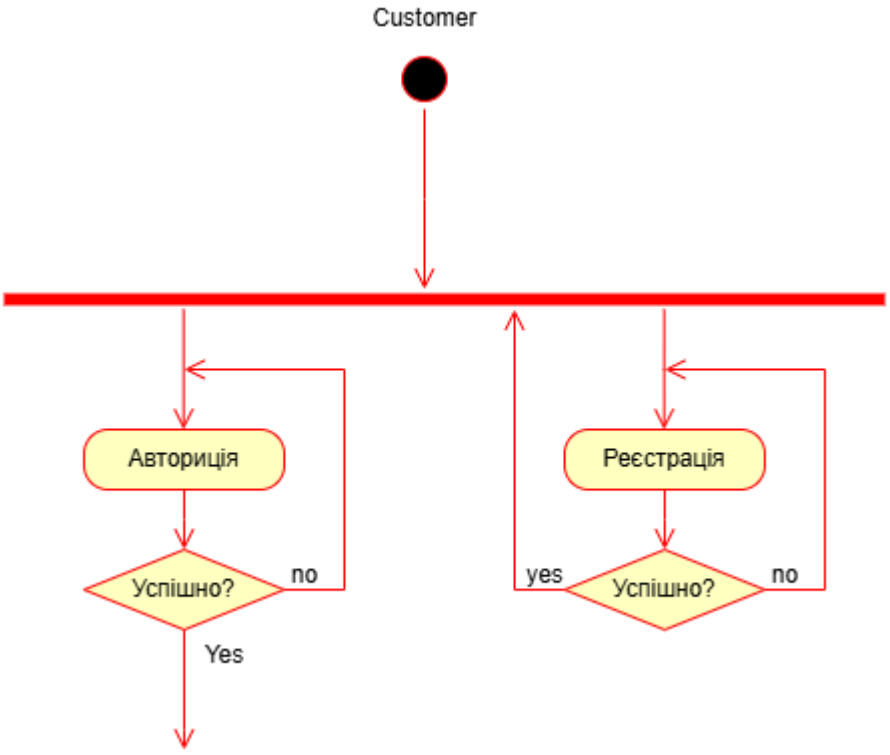


Рис. 3. Activity(для неавторизованного покупця)

Після цього розглянемо роботу кожної функціональності для зареєстрованого в системі покупця.

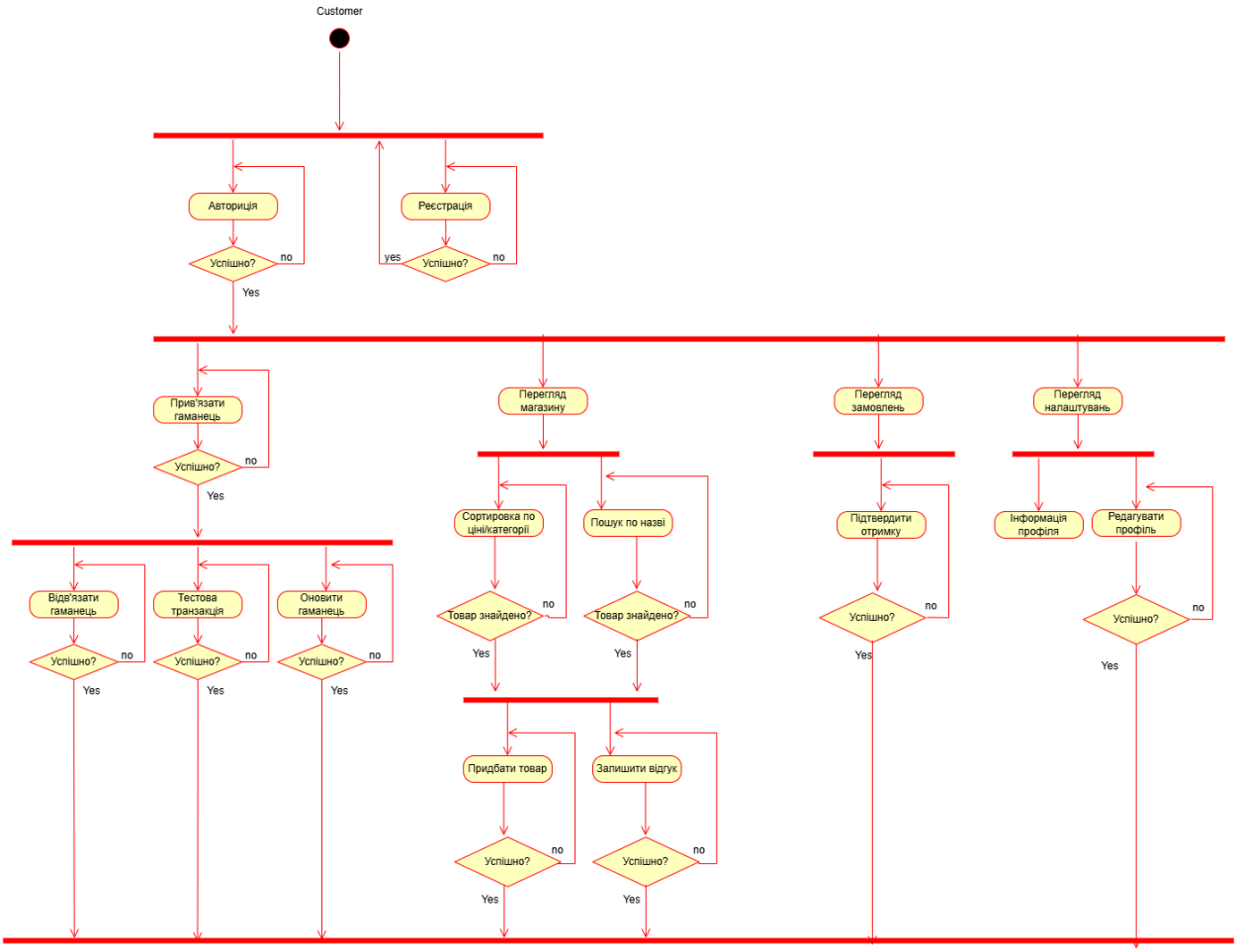


Рис. 4. Activity(для авторизованного покупця)

Також було продумано роботу кожної функціональності з боку продавця.

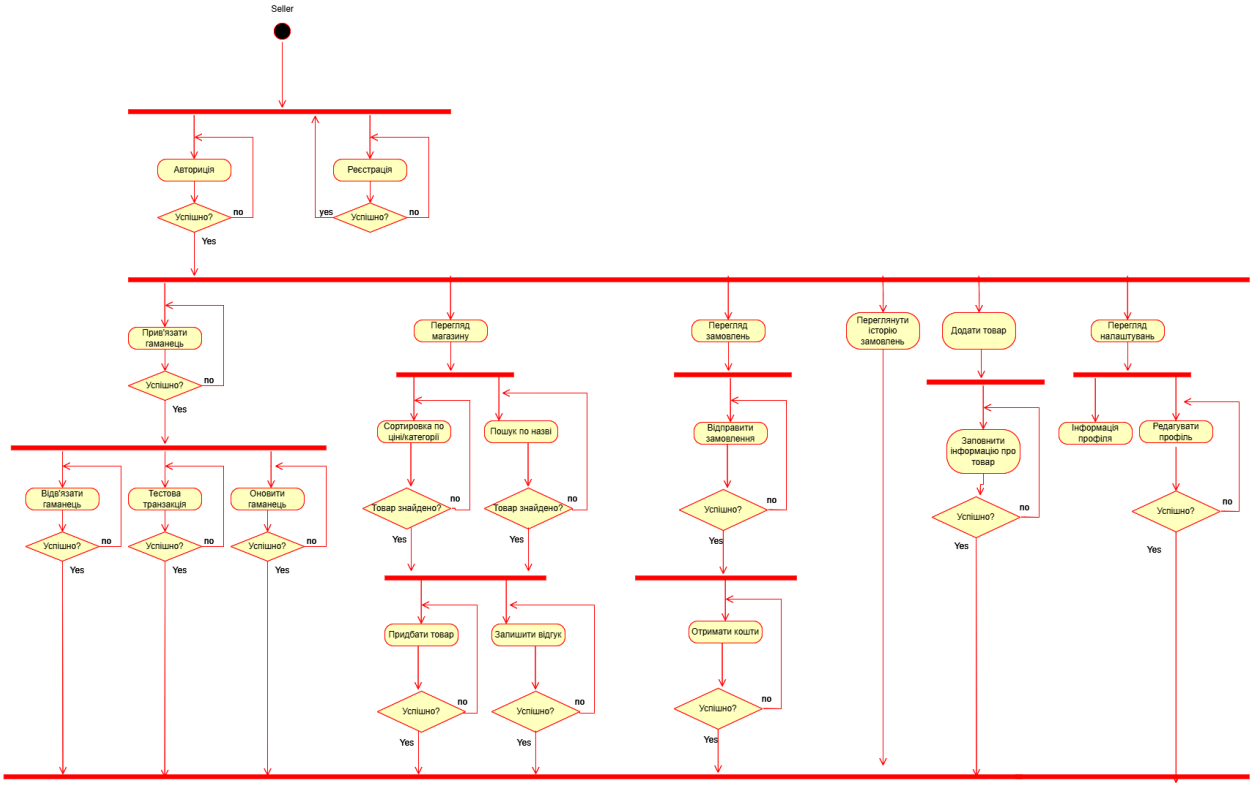


Рис. 5. Activity(для авторизованного продавця)

2.5 Основні налаштування проекту

При налаштуванні проекту для розробки застосунку були налаштовані наступні файли:

- .env — Це файл для зберігання змінних середовища, які використовуються в проекті. Зазвичай, сюди вносяться конфіденційні дані, такі як ключі доступу до баз даних, API ключі, паролі, порти серверів та інші важливі налаштування;

```
.env
DATABASE_URL=postgres://myuser:xxxxxxxx@xxxxxxxxx:xxxx/mydb
NODE_ENV=development
SESSION_SECRET=oxxxxx/>xxxxxx|xxxxxx
FRONTEND_URL=https://shiny-yeot-xxxxxx.netlify.app
TON_MNEMONIC="xxxxx xxxxxx xxxxxxx xxx xxxxxxxxx xxxxxxxxxx xxxxxx xxxxxxxxx xxxxxxxxxx xxxxxxxxxx xxxxxxxxxx xxxxxx xxxxx"
NETWORK=testnet
WALLET_ADDRESS=x:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
UNFORMATTED_WALLET_ADDRESS=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Рис. 6. Приклад файлу .env

У файлі .env зберігаються основні конфігураційні налаштування для проєкту. Зокрема, налаштування для підключення до бази даних PostgreSQL міститься в змінній DATABASE_URL. Для визначення середовища виконання проєкту використовується змінна NODE_ENV, яка вказує, чи працює застосунок у режимі розробки чи в продакшн-середовищі. Секретний ключ для шифрування сесій зберігається в SESSION_SECRET, що забезпечує безпеку користувацьких сесій. Змінна FRONTEND_URL містить URL-адресу фронтенду проєкту, що дозволяє налаштувати взаємодію між бекендом і клієнтською частиною.

Для інтеграції з криптовалютним гаманцем TON використовується змінна TON_MNEMONIC, яка містить мнемонічну фразу для доступу до гаманця. Крім того, змінна NETWORK вказує, що застосунок використовує тестову мережу для транзакцій. WALLET_ADDRESS і UNFORMATTED_WALLET_ADDRESS зберігають адреси гаманців, які використовуються для криптовалютних операцій.

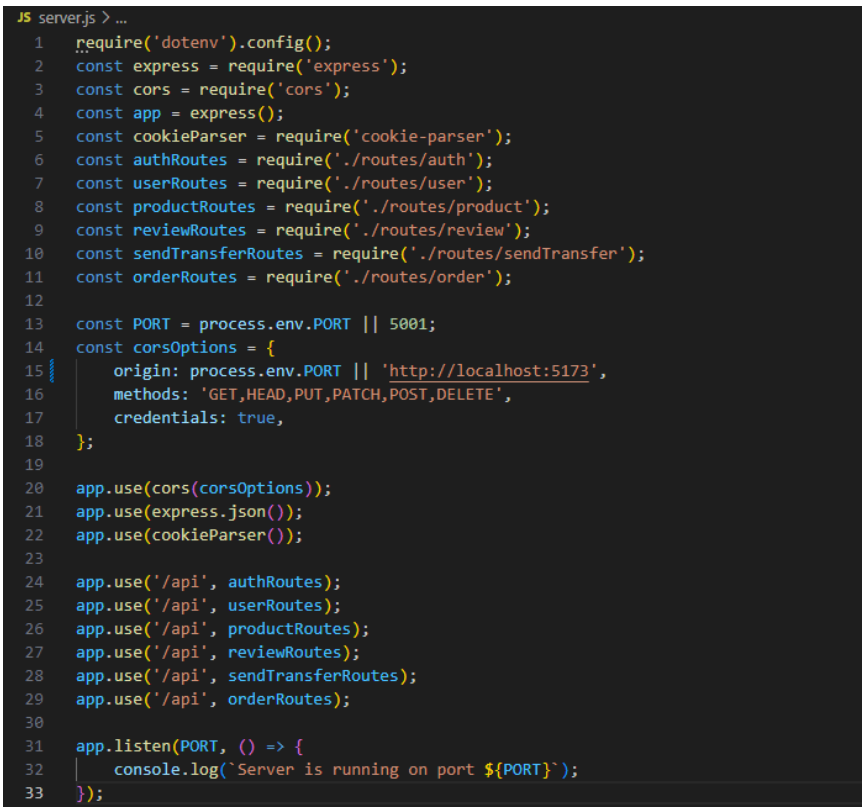
Цей файл є важливим інструментом для налаштування проєкту, оскільки він зберігає конфіденційну інформацію та забезпечує гнучкість у налаштуваннях середовища без необхідності змінювати код.

- package.json — Це основний конфігураційний файл для проєктів на Node.js. У ньому зберігаються залежності проєкту, скрипти для запуску програми, метадані про проєкт та інші налаштування.

Рис. 7. Файл package.json

Файл package.json є основним конфігураційним файлом для проєкту на Node.js. Він містить основну інформацію про проєкт, залежності та скрипти для запуску застосунку. Зокрема, у розділі scripts вказані команди для запуску серверу, тестів і збірки проєкту. У dependencies перелічені бібліотеки, необхідні для роботи серверу, взаємодії з базою даних та роботи з блокчейном. Розділ devDependencies включає залежності для розробки, такі як nodemon для автоматичного перезапуску серверу. Цей файл забезпечує налаштування проєкту та керує його залежностями.

- server.js — це основний файл для запуску серверної частини. У ньому визначаються налаштування серверу (наприклад, Express сервер для обробки запитів). Це також місце для налаштування маршрутизації та обробки HTTP запитів;



```
JS server.js > ...
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const app = express();
const cookieParser = require('cookie-parser');
const authRoutes = require('./routes/auth');
const userRoutes = require('./routes/user');
const productRoutes = require('./routes/product');
const reviewRoutes = require('./routes/review');
const sendTransferRoutes = require('./routes/sendTransfer');
const orderRoutes = require('./routes/order');

const PORT = process.env.PORT || 5001;
const corsOptions = {
    origin: process.env.PORT || 'http://localhost:5173',
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE',
    credentials: true,
};

app.use(cors(corsOptions));
app.use(express.json());
app.use(cookieParser());

app.use('/api', authRoutes);
app.use('/api', userRoutes);
app.use('/api', productRoutes);
app.use('/api', reviewRoutes);
app.use('/api', sendTransferRoutes);
app.use('/api', orderRoutes);

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

Рис. 8. Файл server.js

Цей файл є основним серверним файлом server.js, де налаштовується сервер на базі Express. Він використовує dotenv для завантаження змінних середовища з файлу .env, що дозволяє налаштувати конфіденційні параметри, такі як порти та інші налаштування. За допомогою express створюється сервер, який обробляє HTTP-запити, а cookie-parser дозволяє працювати з cookies.

Застосовується cors для налаштування політики доступу між доменами, дозволяючи клієнтській частині взаємодіяти з сервером, навіть якщо вони працюють на різних доменах. Далі налаштовуються маршрути для різних частин API: authRoutes, userRoutes, productRoutes, reviewRoutes, sendTransferRoutes та orderRoutes, що відповідають за функціональність авторизації, користувачів, продуктів, відгуків, переказів і замовлень відповідно. Сервер запускається на вказаному порті, за замовчуванням це 5001, і готовий обробляти запити від клієнтської частини.

- main.tsx — це головний файл для ініціалізації React-програми з використанням TypeScript. У ньому визначається рендеринг компонента App, який є точкою входу для відображення всієї програми.

src > main.tsx

```tsx
import { Buffer } from 'buffer';
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import './index.css';
import App from './App.tsx';
import store from './store.ts';
import { Provider } from 'react-redux';

(globalThis as any).Buffer = Buffer;

createRoot(document.getElementById('root')!).render(
  <StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </StrictMode>
);
```

Рис. 9. Файл main.tsx

- vite-env.d.ts — це TypeScript файл, який містить типи для змінних середовища та налаштувань, специфічних для Vite. Зазвичай він використовується для роботи з процесами компіляції та налаштуваннями змінних середовища.

src > TS vite-env.d.ts

```ts
/// <reference types="vite/client" />

```

Рис. 10. Файл vite-env.d.ts

2.6 Проектування макету

Перше, з чого почалася розробка макету - це навігаційна панель, яка буде присутньою на кожній сторінці застосунку. Головні вимоги, які були поставлені при розробці макету навігаційної панелі, це зручність та зрозумілість. Тому основна навігація мала включати у себе:

- Основну інформацію про користувачі, а саме ім'я та пошта;
- Лінк на основну сторінку сайту;
- Лінк на сторінку з магазином;
- Лінк на сторінку з історією доданих товарів;
- Лінк на сторінку для додавання товару;
- Лінк на сторінку з замовленнями;
- Лінк на сторінку з налаштуваннями;
- Лінк на сторінку виходу з системи.

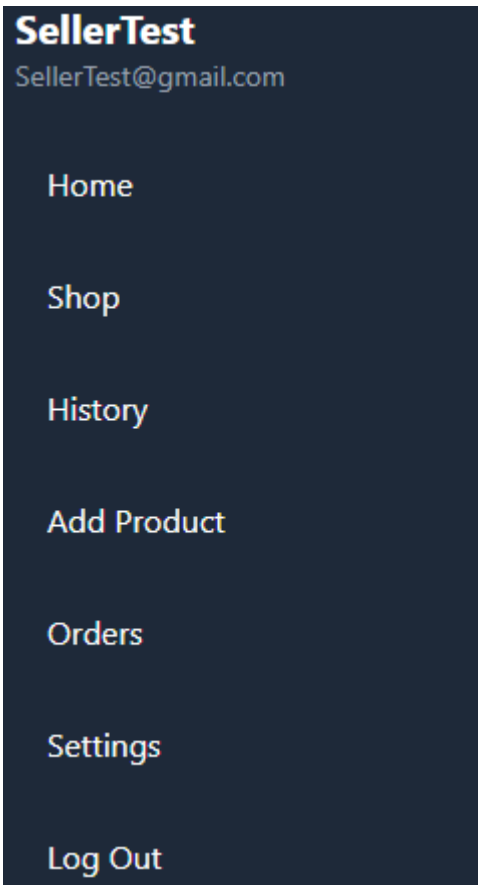


Рис. 11. Навігаційна панель

Після цього була розроблена панель для взаємодії з гаманцем, яка включала можливість в себе можливість під'єднання гаманеця.

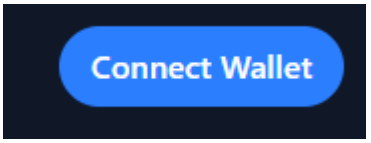


Рис. 12. Кнопка під'єднання гаманця

Після цього користувач міг взаємодіяти з гаманцем, а саме: оновити його, переглянути адресу, провести тестову транзакцію та від’єднати гаманець.

Рис. 13. Секція взаємодії з гаманцем.

Далі була розроблена домашня сторінка сайту, на якій є привітання користувача; секція для взаємодії з гаманцем, секція в якій показано нові надходження товарів, та нижня частина, якій можна підписатися на розсилку новин сайту на пошту.

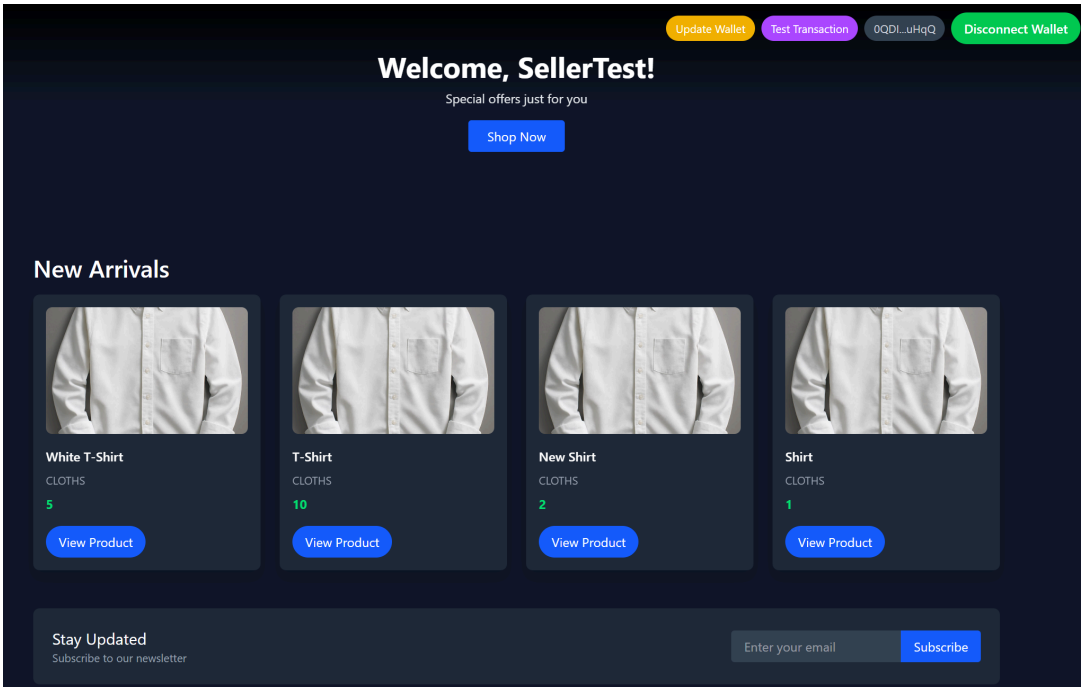


Рис. 14. Макет головної сторінки застосунку.

Також була розроблена сторінка магазину з товарами, на якій є можливість пошуку товарів за назвою та сортування їх за ціною й категорією.

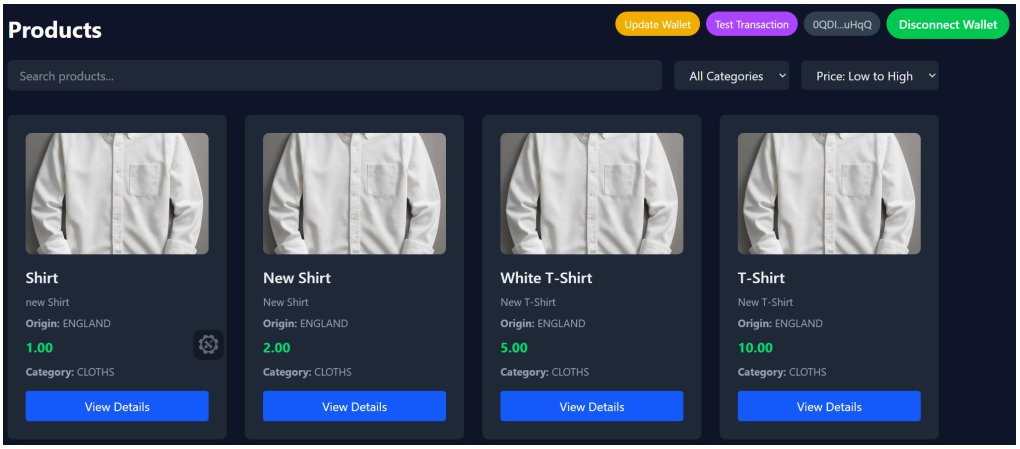


Рис. 15. Макет сторінки магазину.

Після цього було додано сторінку, на якій продавець може додавати товар і вводити його характеристики.

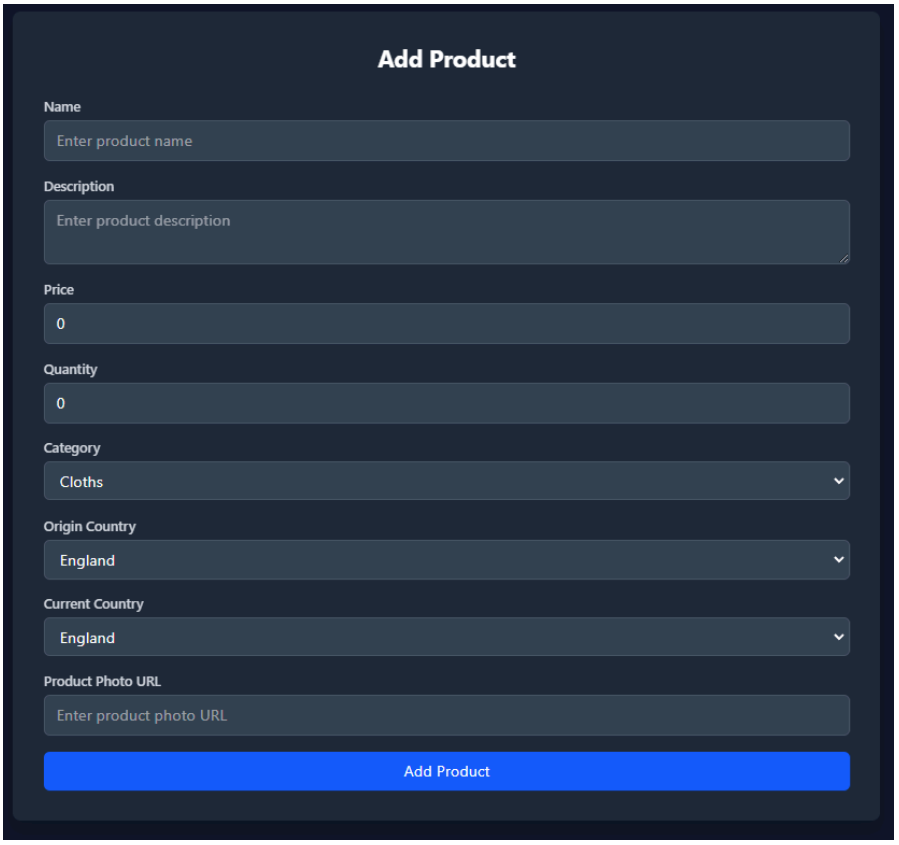


Рис. 16. Макет сторінки додавання продукта.

Також було додано сторінку, на якій продавець може переглядати додані товари.

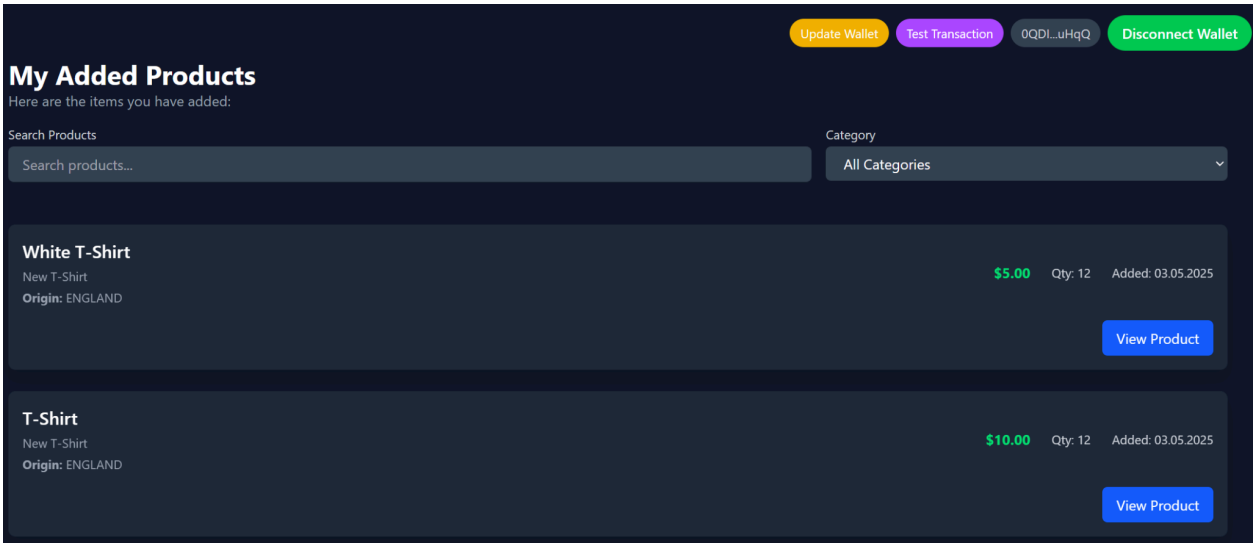


Рис. 17. Макет сторінки перегляду доданих товарів.

Наступною була сторінка для перегляду товару, на якій розміщено фото товару, його опис і ціну. Також на цій сторінці є налаштування для замовлення: можливість вибрати кількість, ввести адресу доставки та натиснути кнопку для придбання товару.

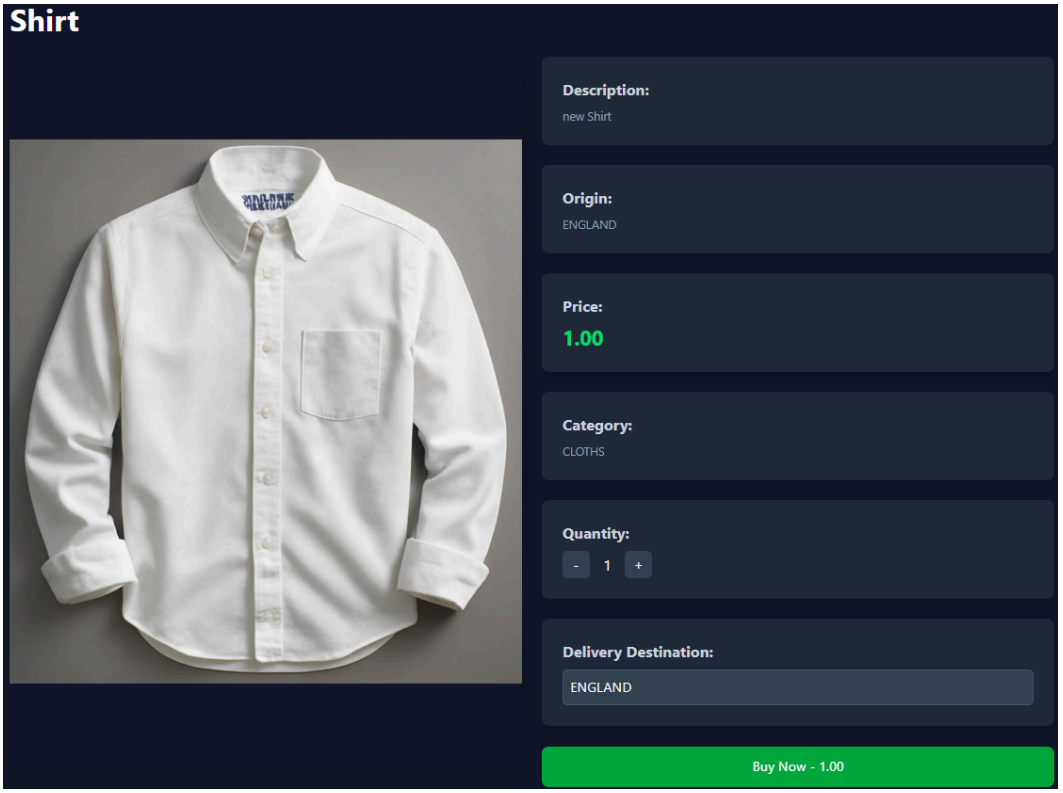


Рис. 18. Макет сторінки товару.

Також на цій сторінці є секція з коментарями та оцінками покупців. У цій секції можна переглянути відгуки інших покупців, а також додати власний коментар.

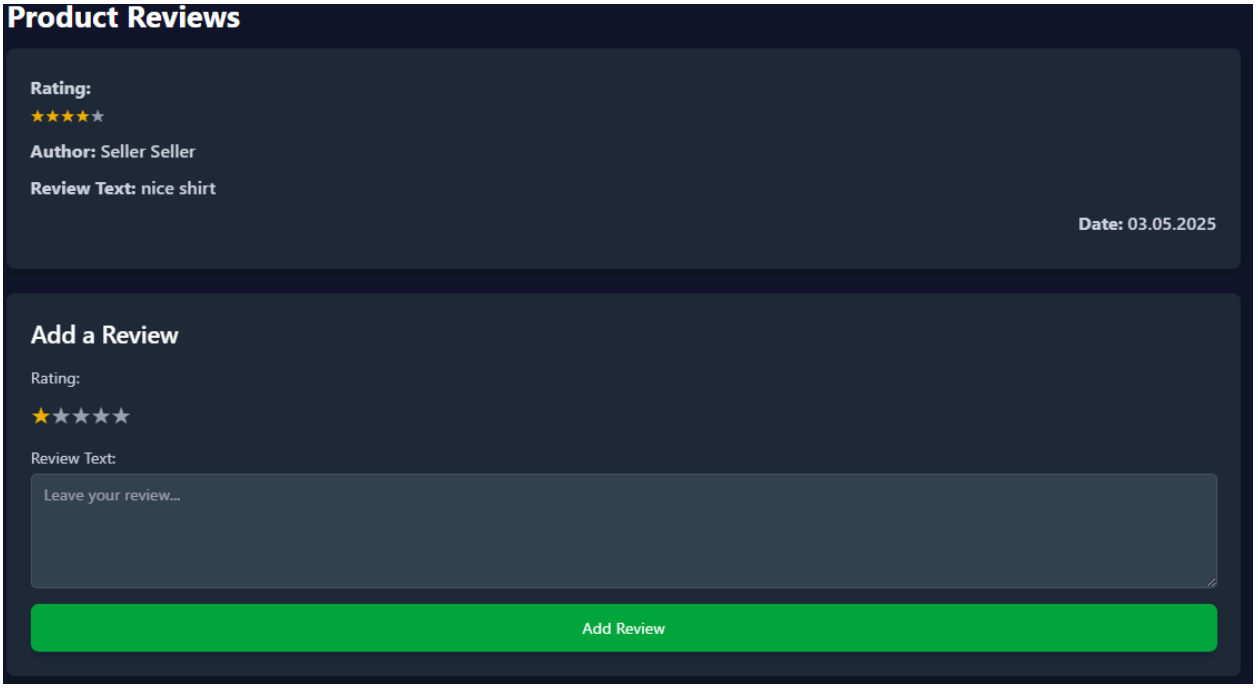


Рис. 19. Макет секції відгуків.

Для замовлених товарів була розроблена окрема сторінка, на якій покупці та продавці можуть переглянути виконані та активні замовлення. На цій сторінці для кожного замовлення відображаються основні дані.

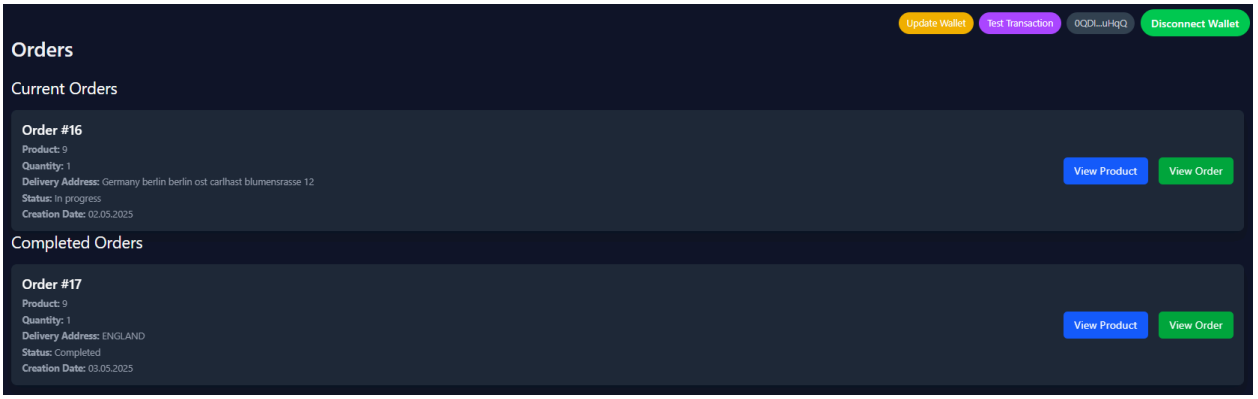


Рис. 20. Макет сторінки замовлень.

Для кожного замовлення створюється сторінка, яка поділяється на дві частини: у першій відображається докладна інформація — номер замовлення,

адреса доставки, кількість товару та дата створення; у другій — інформація про продавця та товар.

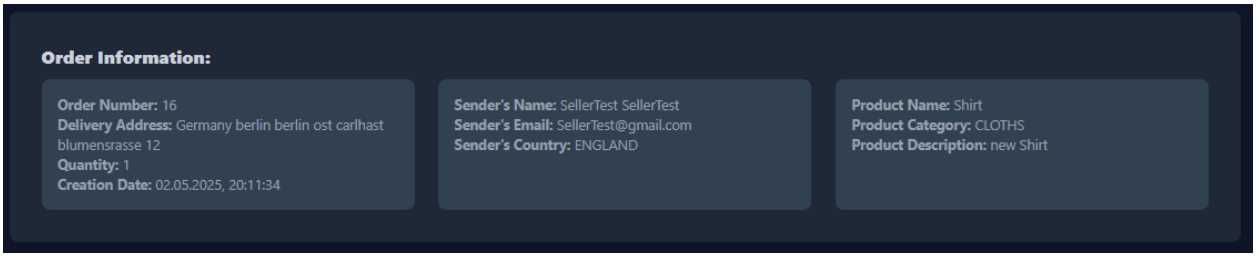


Рис. 21. Макет інформації о замовленні.

У другій частині сторінки знаходиться інформація про транзакцію та взаємодію з нею. У цій секції пишуться хеші, які потрібні для відстеження криптовалютної транзакції та обробки стану товару, а також розміщено кнопки для взаємодії з оплатою. Тут можна підтвердити відправлення товару, його отримання та надсилання коштів.

Рис. 22. Макет взаємодії з транзакціями.

Також була розроблена секція налаштувань, у якій можна переглянути інформацію про акаунт, а також змінити ім’я та країну.

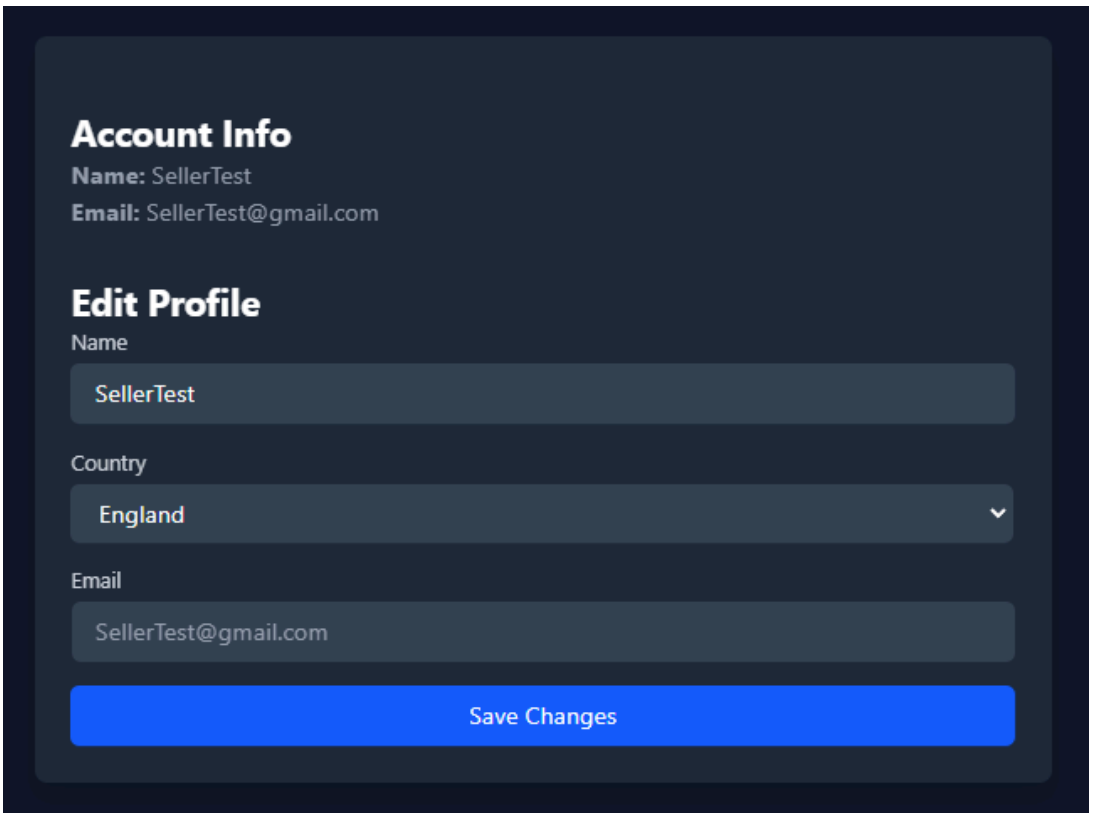


Рис. 23. Макет взаємодії з налаштуваннями акаунту.

Також реалізовано сторінки реєстрації та входу користувачів. На сторінці реєстрації зазначаються ім’я, прізвище, електронна пошта, пароль і країна, а також вибір ролі — продавець або покупець.

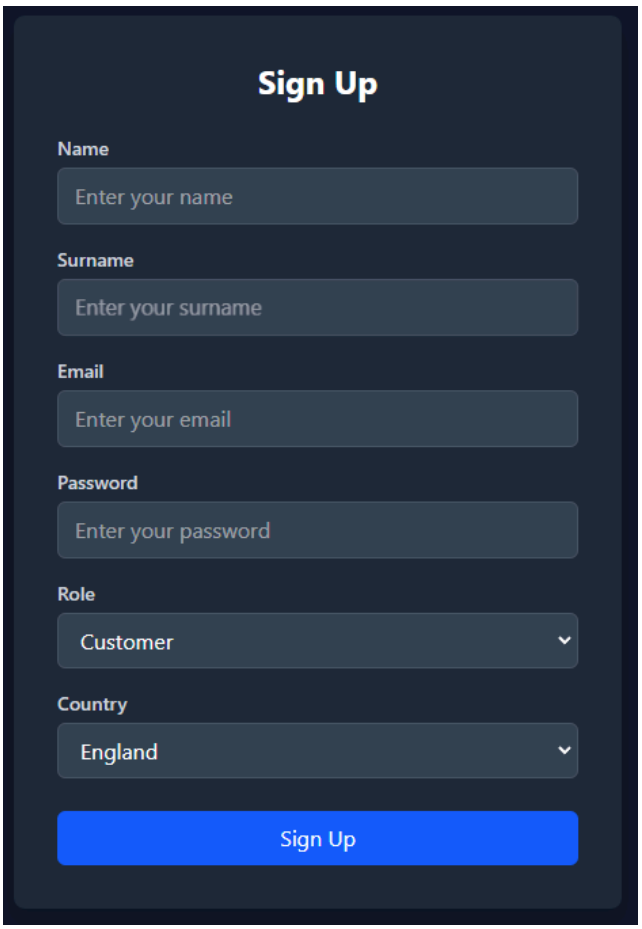


Рис. 24. Макет реєстрації юзера.

Для входу в акаунт потрібно лише електронну пошту та пароль. Також є кнопка для переходу на сторінку реєстрації.

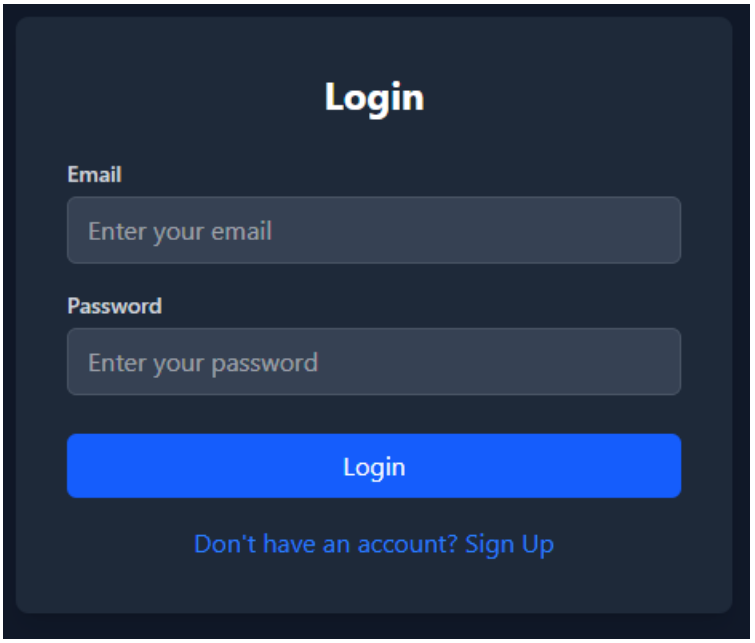


Рис. 25. Макет логіну юзера.

2.7 Пристосування програми для мобільних пристроїв

На сьогодні мобільні пристрої є найпоширенішим способом взаємодії з інтернет-ресурсами, тому проект адаптували під їхні екрани. Для цього використали React та Vite, які полегшують налаштування застосунку під різні розміри дисплеїв. Основні механізми адаптивного дизайну:

- CSS Grid і Flexbox: застосовано для побудови гнучкої верстки, що підлаштовується під розміри екрана;
- CSS-медіа-запити: налаштовують вигляд сторінок на різних пристроях;
- Ліниве завантаження (lazy loading): оптимізовано завантаження зображень для швидкої роботи;
- Адаптивні форми: забезпечують зручність заповнення на малих екранах;
- Інтеграція гаманця: використано @tonconnect/ui-react для мобільного доступу до криптогаманця.[6]

Ці технології гарантують ефективну роботу застосунку на мобільних пристроях та покращують досвід користувачів.

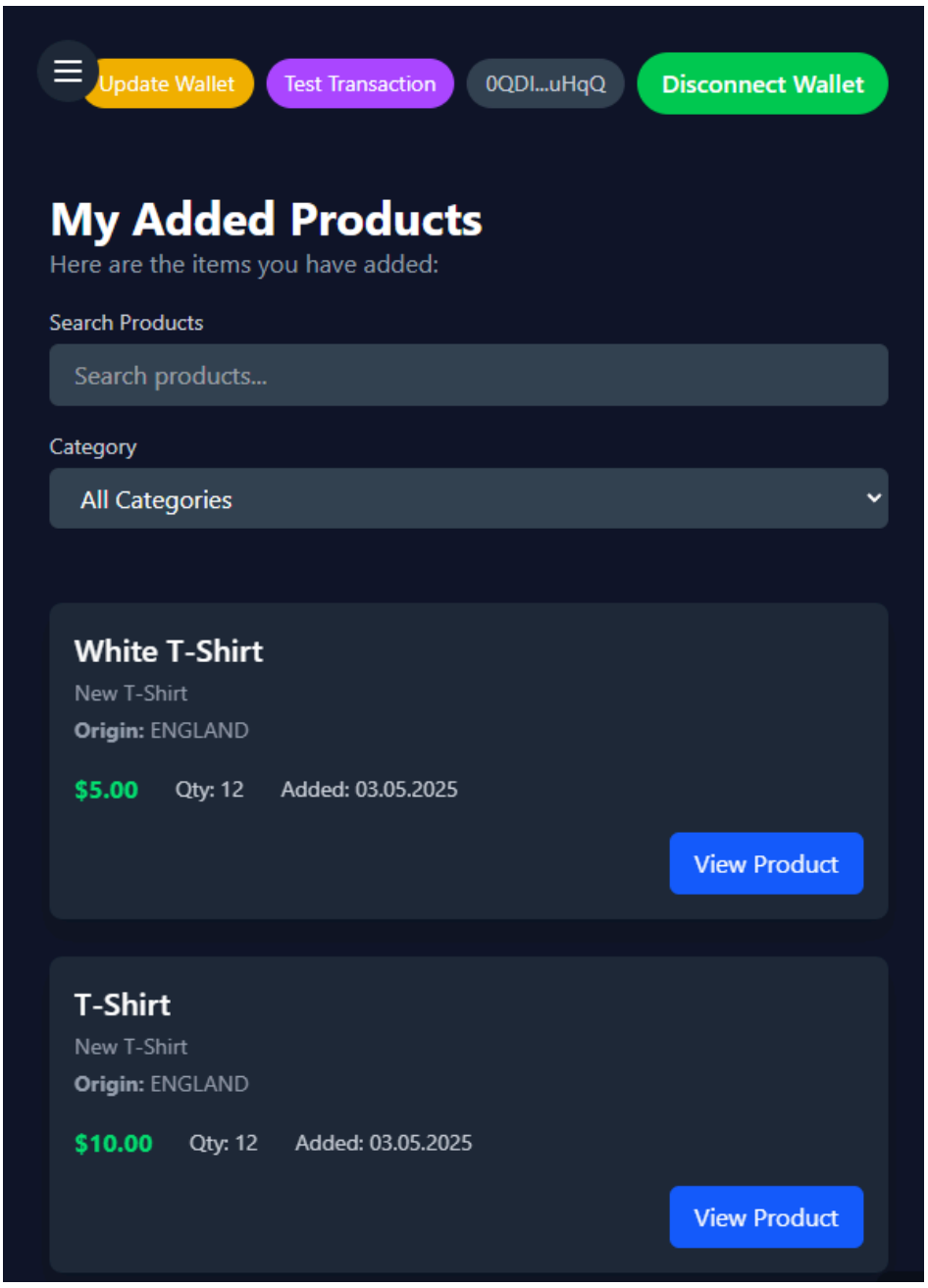


Рис. 26. Приклад вигляду сторінки з мобільного пристрою.

2.8 Реалізація роутингу між сторінками

Для налаштування роутингу між компонентами застосунку використано бібліотеку react-router-dom, яка є стандартом для реалізації маршрутизації в проектах на React. За допомогою цієї бібліотеки була реалізована навігація між сторінками без необхідності перезавантаження всієї сторінки, що

дозволило створити зручний інтерфейс з динамічним контентом. Першим кроком було встановлення бібліотеки за допомогою наступної команди в терміналі: npm install react-router-dom.

Після цього імпортовано хук useNavigate з react-router-dom, що дозволило здійснювати програмну навігацію між компонентами.

```
import { useNavigate } from 'react-router-dom';

const Shop: React.FC = () => {

  const navigate = useNavigate();
```

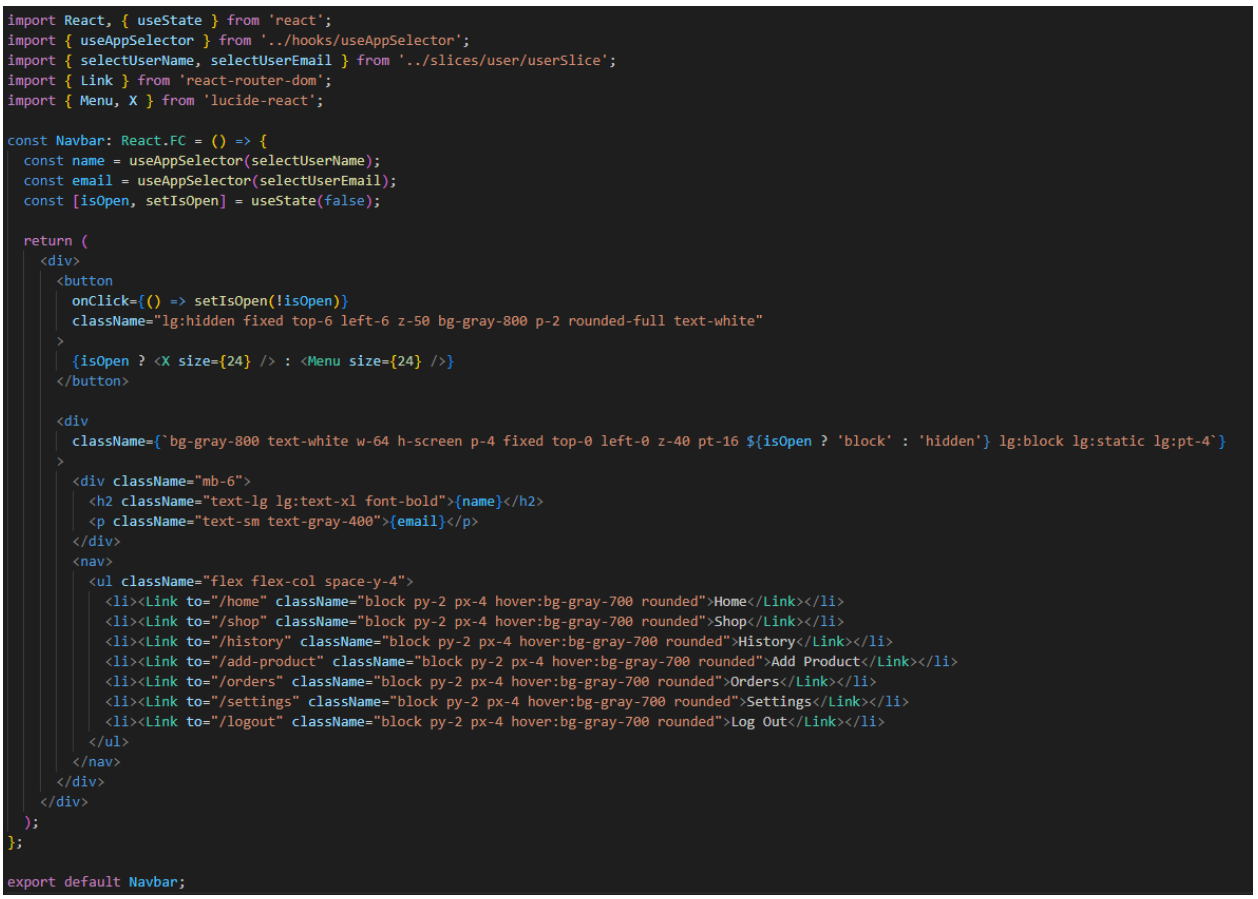
```
<button
  onClick={() => navigate(`/product/${prod.id}`)}
  className="mt-auto bg-blue-600 py-2 rounded text-white hover:bg-blue-700 transition-all duration-300 transform hover:scale-105"
>
  View Details
</button>
```

Рис. 27. Приклад використання роумінгу на сторінці Shop.

У цьому прикладі при натисканні на кнопку, викликається функція handleViewProduct, яка здійснює перехід на сторінку товару за допомогою navigate(/product/${productId}), де productId — це унікальний ідентифікатор товару.

Також для забезпечення зручного роутингу та навігації між сторінками застосунку створено навбар (панель навігації), який дозволяє користувачам швидко переходити між різними розділами застосунку, такими як Home, Shop, History, Orders, Add Product, Settings та Log Out.

Навбар реалізовано за допомогою компонента Navbar. Для мобільних пристроїв передбачено адаптивне меню з кнопкою бургер-меню, яке відкривається при натисканні на іконку меню. Це дозволяє економити місце на екрані мобільних пристроїв, одночасно зберігаючи функціональність навігації.



```tsx
import React, { useState } from 'react';
import { useAppSelector } from '../hooks/useAppSelector';
import { selectUserName, selectUserEmail } from '../slices/user/userSlice';
import { Link } from 'react-router-dom';
import { Menu, X } from 'lucide-react';

const Navbar: React.FC = () => {
  const name = useAppSelector(selectUserName);
  const email = useAppSelector(selectUserEmail);
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <button
        onClick={() => setIsOpen(!isOpen)}
        className="lg:hidden fixed top-6 left-6 z-50 bg-gray-800 p-2 rounded-full text-white"
      >
        {isOpen ? <X size={24} /> : <Menu size={24} />}
      </button>

      <div
        className={`bg-gray-800 text-white w-64 h-screen p-4 fixed top-0 left-0 z-40 pt-16 ${isOpen ? 'block' : 'hidden'} lg:block lg:static lg:pt-4`}
      >
        <div className="mb-6">
          <h2 className="text-lg lg:text-xl font-bold">{name}</h2>
          <p className="text-sm text-gray-400">{email}</p>
        </div>
        <nav>
          <ul className="flex flex-col space-y-4">
            <li><Link to="/home" className="block py-2 px-4 hover:bg-gray-700 rounded">Home</Link></li>
            <li><Link to="/shop" className="block py-2 px-4 hover:bg-gray-700 rounded">Shop</Link></li>
            <li><Link to="/history" className="block py-2 px-4 hover:bg-gray-700 rounded">History</Link></li>
            <li><Link to="/add-product" className="block py-2 px-4 hover:bg-gray-700 rounded">Add Product</Link></li>
            <li><Link to="/orders" className="block py-2 px-4 hover:bg-gray-700 rounded">Orders</Link></li>
            <li><Link to="/settings" className="block py-2 px-4 hover:bg-gray-700 rounded">Settings</Link></li>
            <li><Link to="/logout" className="block py-2 px-4 hover:bg-gray-700 rounded">Log Out</Link></li>
          </ul>
        </nav>
      </div>
    </div>
  );
};

export default Navbar;
```

Рис. 28. Реалізація Navbar.

У навбарі використовується компонент Link з react-router-dom для створення посилань на різні маршрути в застосунку. Це дозволяє користувачам швидко переміщатися між сторінками без перезавантаження сторінки, що значно підвищує ефективність і зручність роботи з застосунком.

Таким чином, завдяки навбару користувач може зручно і швидко здійснювати навігацію по застосунку, а адаптивний дизайн гарантує, що панель навігації буде виглядати коректно як на великих екранах, так і на мобільних пристроях.

У файлі app.tsx налаштовано всі маршрути для навігації між різними сторінками застосунку, використовуючи компоненти з бібліотеки react-router-dom: BrowserRouter as Router, Routes, Route та Navigate. Це дозволяє користувачам зручно та швидко переходити між сторінками, такими як логін, реєстрація, головна сторінка, магазин, історія замовлень тощо, без

необхідності перезавантажувати сторінку, забезпечуючи плавну і безперервну навігацію.

BrowserRouter обгортає всю навігацію та дозволяє працювати з роутингом на клієнтському боці, забезпечуючи підтримку історії браузера для переходів між сторінками. Routes є контейнером, що містить усі маршрути, кожен з яких визначає шлях, за яким відображається певний компонент сторінки. Route відповідає за конкретний маршрут, шлях до якого веде до певної сторінки, і пов'язує цей шлях з компонентом, який буде відображено. Navigate використовується для перенаправлення користувачів на інші сторінки, наприклад, для автоматичного перенаправлення на сторінку логіну, якщо користувач не автентифікований.

Приклад реалізації маршрутів включає сторінки для логіну /login, реєстрації /signup, головної сторінки /home, магазину /shop, детальної інформації про товар /product/:id, а також перегляду замовлень користувача /orders.

Завдяки цьому налаштуванню роутінгу користувач може безперешкодно переміщатися між різними частинами застосунку, що значно підвищує зручність та ефективність використання платформи.

2.9 Управління станом та взаємодія компонентів

Для опису компонентів у застосунку використовували інтерфейси, які забезпечують управління станом компонентів програми та взаємодію з сервером.

У нашому застосунку використано кілька слайсів для управління різними аспектами стану, зокрема для користувача, продуктів, замовлень та відгуків. Ось перелік слайсів:

- userApiSlice – відповідає за взаємодію з сервером для аутентифікації користувачів та обробки запитів.

- userSlice – управляє локальним станом користувача на клієнтській стороні.
- productApiSlice – відповідає за взаємодію з сервером для отримання та оновлення даних про продукти.
- productSlice – управління локальним станом продуктів.
- orderApiSlice – обробляє запити для отримання та обробки замовлень.
- orderSlice – зберігає локальний стан замовлень.
- reviewApiSlice – забезпечує взаємодію з сервером для додавання та отримання відгуків.
- reviewSlice – управляє локальним станом відгуків.

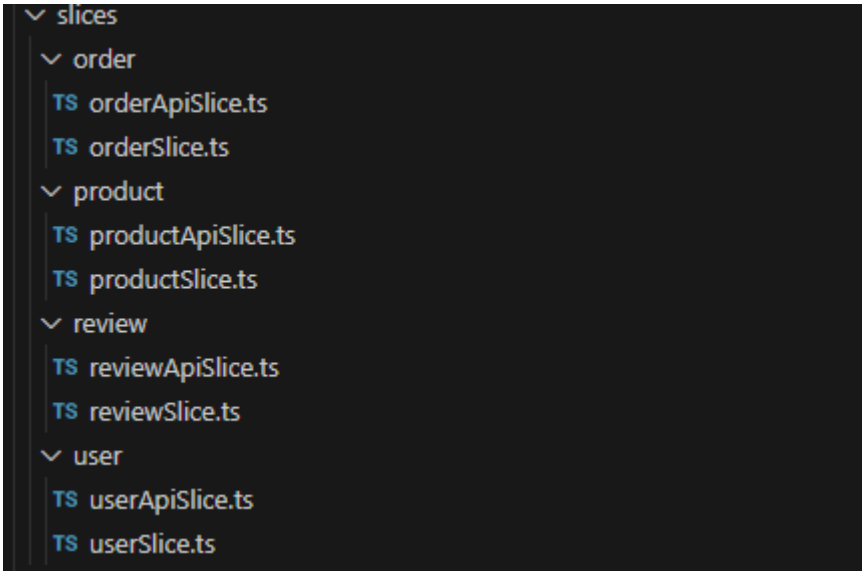


Рис. 29. Перелік слайсів.

Почнемо з опису userApiSlice та userSlice:

1. userApiSlice

userApiSlice використовує Redux Toolkit для створення API, яке відповідає за виконання запитів до серверу, зокрема для реєстрації, авторизації користувача, а також для обробки транзакцій та платіжних операцій. Основні функції, які виконує userApiSlice:

- signin і signup – мутатори для реєстрації та авторизації користувача. Вони надсилають POST запити на сервер для отримання або реєстрації користувача.
- fetchUserById – отримує дані користувача по ID з серверу.
- connectWallet – дозволяє користувачу прив'язати нову адресу гаманця до його акаунту.
- sendPayment, receivePayment, shipPayment – ці запити дозволяють обробляти платежі, підтверджувати транзакції та виконувати інші фінансові операції.

2. userSlice

- userSlice відповідає за збереження та оновлення локального стану користувача у Redux store. Він дозволяє ефективно зберігати дані про користувача, такі як ім’я, прізвище, email, країна, роль та адреса гаманця. Основні функції:
- setUser – оновлює стан користувача після отримання даних з серверу.
- clearUser – очищає стан користувача при виході з акаунту.
- updateName, updateCountry, updateWalletAddress – ці функції дозволяють оновлювати окремі дані користувача, наприклад, ім’я або країну.

```
updateName: (state, action: PayloadAction<string>) => {
  state.name = action.payload;
},
updateCountry: (state, action: PayloadAction<string>) => {
  state.country = action.payload;
},
updateWalletAddress: (state, action: PayloadAction<string | null>) => {
  state.walletAddress = action.payload;
},
```

Рис. 30. Реалізація updateName, updateCountry, updateWalletAddress .

Ці слайси взаємодіють з сервером і зберігають дані про користувача на клієнтській стороні. Вони дозволяють зберігати та оновлювати ці дані у глобальному стані додатку, що дає змогу оперативно реагувати на зміни без необхідності повторно запитувати сервер.

Опис productApiSlice та productSlice:

1. productApiSlice

productApiSlice відповідає за взаємодію з сервером для отримання та додавання продуктів, а також для обробки запитів, пов'язаних із конкретними товарами, зокрема за їхнім ідентифікаційним номером або назвою. Він створений за допомогою Redux Toolkit Query, що дозволяє легко виконувати HTTP запити до сервера та обробляти відповіді.

Основні ендпоінти productApiSlice:

- fetchProducts: Використовується для отримання списку всіх продуктів. Запит відправляється на сервер для отримання даних про продукти. Результатом є масив продуктів, який потім передається для подальшої обробки.
- addProduct: Запит для додавання нового продукту. Відправляється на сервер POST запит з даними нового продукту.
- deleteProduct: Використовується для видалення продукту за допомогою ID.
- fetchProductByName: Використовується для отримання продукту за його назвою. Використовує параметр назви, щоб знайти товар на сервері.
- fetchProductById: Цей ендпоінт відповідає за отримання даних конкретного продукту за його ID.

```
endpoints: builder => ({
  fetchProducts: builder.query<Product[], void>({
    query: () => ({
      url: '/products',
      headers: {
        'ngrok-skip-browser-warning': 'skip-browser-warning'
      }
    }),
    transformResponse: (response: { success: boolean; count: number; products: Product[] }) =>
      response.products,
  }),
  addProduct: builder.mutation<Product, ProductPayload>({
    query: (newProduct) => ({
      url: '/add-product',
      method: 'POST',
      body: newProduct.product,
    }),
  }),
  deleteProduct: builder.mutation<{ id: number }, { id: number }>({
    query: (productId) => ({
      url: `/products/${productId.id}`,
      method: 'DELETE',
    }),
  }),
  fetchProductByName: builder.query<Product, string>({
    query: (name) => ({
      url: `/product/${name}`,
      method: 'GET',
    }),
    transformResponse: (response: ProductResponse) => response.product,
  }),
  fetchProductById: builder.query<Product, number>({
    query: id => `/products/${id}`,
    transformResponse: (response: ProductResponse) => response.product,
  }),
```

Рис. 31. Реалізація ендпоінтів productApiSlice.

2. productsSlice

productsSlice управляє локальним станом додатку, зберігаючи інформацію про продукти, отримані з сервера, а також про статуси запитів (наприклад, завантаження, помилки тощо). Він забезпечує збереження даних у Redux store та дозволяє зручно керувати ними на клієнтській стороні. Основні частини productsSlice:

- setProducts: Цей редьюсер оновлює список продуктів у локальному стані. Може бути викликаний вручну для оновлення стану.
- setProductById: Відповідає за оновлення стану з інформацією про продукт за ID.
- clearProducts: Очищає дані про продукти, скидаючи їх до початкового стану.

- extraReducers: У цьому розділі визначені логіка для оновлення стану в залежності від результату запитів API. Він охоплює різні етапи запитів: pending, fulfilled, rejected.

Опис orderApiSlice та orderSlice

1. orderApiSlice

orderApiSlice використовується для взаємодії з сервером щодо запитів, що стосуються замовлень. Це API, яке дозволяє виконувати різні операції, такі як отримання списку замовлень, створення нових замовлень, відправка платежів і підтвердження відправлення товару. Всі ці запити обробляються через Redux Toolkit Query, що забезпечує просте й ефективне управління станом запитів.

Основні ендпоінти orderApiSlice:

- fetchOrders: Цей запит виконується для отримання списку замовлень. Він повертає всі замовлення для користувача в вигляді масиву об'єктів типу Order.
- addOrder: Запит для створення нового замовлення. Відправляється POST-запит із даними замовлення.
- sendPayment: Запит для підтвердження платежу. Він відправляє хеш транзакції і ID замовлення на сервер.
- shipPayment: Запит для підтвердження відправки товару. Після виконання цього запиту, статус замовлення змінюється.

```
endpoints: (builder) => ({
  fetchOrders: builder.query<Order[], void>({
    query: () => ({
      url: `/orders`,
      method: 'GET',
      headers: {
        'ngrok-skip-browser-warning': 'skip-browser-warning',
      },
    }),
    transformResponse: (response: { success: boolean; orders: Order[] }) =>
      response.orders,
  }),
  addOrder: builder.mutation<Order, OrderPayload>({
    query: (newOrder) => ({
      url: '/buy-payment',
      method: 'POST',
      body: newOrder.order,
    }),
  }),
  sendPayment: builder.mutation<Order, { tx_hash: string; orderId: number }>({
    query: ({ tx_hash, orderId }) => ({
      url: '/send-payment',
      method: 'POST',
      body: { tx_hash, orderId },
    }),
  }),
  shipPayment: builder.mutation<Order, { tx_hash: string; orderId: number }>({
    query: ({ tx_hash, orderId }) => ({
      url: '/ship-payment',
      method: 'POST',
      body: { tx_hash, orderId },
    }),
  }),
```

Рис. 32. Реалізація ендпоінтів orderApiSlice.

2. orderSlice

orderSlice відповідає за локальне зберігання стану замовлень у Redux store. Він дозволяє керувати такими аспектами, як статуси замовлень (завантаження, успішне виконання, помилки) та містить редьюсери для оновлення стану замовлень.

Основні частини orderSlice:

- setOrders: Цей редьюсер дозволяє вручну оновлювати список замовлень.
- setOrderById: Редьюсер, що дозволяє оновити конкретне замовлення за його ID.

- clearOrders: Очищає список замовлень і скидає всі дані про замовлення.
- extraReducers: Логіка для обробки різних стадій запитів API (наприклад, fetchOrders, addOrder, sendPayment). Це дозволяє оновлювати статуси замовлень залежно від результату запиту (наприклад, якщо запит на отримання замовлень пройшов успішно, статус змінюється на "succeeded").

```
const orderSlice = createSlice({
  name: "order",
  initialState,
  reducers: {
    setOrders: (state, action: PayloadAction<OrderState['orders']>) => {
      state.orders = action.payload;
    },
    setOrderById: (state, action: PayloadAction<Order>) => {
      state.orderById = action.payload;
    },
    clearOrders: (state) => {
      state.orders = [];
      state.orderById = null;
    },
  },
  extraReducers: (builder) => {
    builder.addMatcher(
      orderApiSlice.endpoints.fetchOrders.matchFulfilled,
      (state, { payload }) => {
        state.status = 'succeeded';
        state.orders = payload;
      }
    );
```

Рис. 33. Реалізація orderSlice.

Опис reviewApiSlice та reviewSlice

1. reviewApiSlice

reviewApiSlice реалізує API для роботи з відгуками користувачів на продукти. Цей API дозволяє створювати нові відгуки, а також отримувати існуючі відгуки за конкретними продуктами. Для цього використовуються ендпоінти, що підтримують взаємодію з бекендом через запити GET і POST. Основні ендпоінти reviewApiSlice:

- addReview: Цей ендпоінт дозволяє додавати новий відгук для конкретного продукту. Запит приймає рейтинг і текст відгуку, а також ідентифікатор продукту, до якого додається відгук.
- fetchReviewsByProductId: Запит для отримання відгуків по конкретному продукту за його ID. Він повертає масив відгуків, а також інформацію про пагінацію (загальна кількість відгуків, кількість сторінок тощо).

```
endpoints: (builder) => ({
  addReview: builder.mutation<Review, ReviewPayload>({
    query: (newReview) => ({
      url: `/products/${newReview.productId}/reviews`,
      method: 'POST',
      body: {
        rating: newReview.rating,
        text: newReview.text || null,
      },
      headers: {
        'Content-Type': 'application/json',
        'ngrok-skip-browser-warning': 'skip-browser-warning',
      },
    }),
  }),
  fetchReviewsByProductId: builder.query<ReviewsResponse, number>({
    query: (productId) => ({
      url: `/products/${productId}/reviews`,
      method: 'GET',
      headers: {
        'Content-Type': 'application/json',
        'ngrok-skip-browser-warning': 'skip-browser-warning',
      },
    }),
    transformResponse: (response: { success: boolean; reviews: Review[]; pagination: any }) => ({
      reviews: response.reviews,
      pagination: response.pagination,
```

Рис. 34. Реалізація ендпоінтів reviewApiSlice.

2. reviewSlice

reviewSlice керує станом відгуків в Redux store. Він зберігає відгуки для продуктів і має механізми для їх оновлення та очищення. Це дозволяє зберігати локальний стан відгуків користувачів, оновлюючи його при отриманні нових відгуків або додаванні нових відгуків через API.
Основні частини reviewSlice:

- setReviews: Редьюсер для оновлення списку відгуків вручну. Він дозволяє змінювати масив відгуків в залежності від нових даних.
- clearReviews: Редьюсер для очищення відгуків із стану. Це необхідно для скидання стану після завершення операцій.

- extraReducers: Логіка для обробки різних станів запитів API.

Зокрема, додається підтримка різних стадій для addReview та fetchReviewsByProductId, таких як "loading", "succeeded", та "failed". Це дозволяє коректно оновлювати стан при успішних або невдалих запитах.

```
reducers: {
  setReviews: (state, action: PayloadAction<ReviewState['reviews']>) => {
    state.reviews = action.payload;
  },
  clearReviews: (state) => {
    state.reviews = [];
    state.reviewByProductId = null;
  },
},
extraReducers: (builder) => {
  builder.addMatcher(
    reviewApiSlice.endpoints.addReview.matchFulfilled,
    (state, action) => {
      state.status = 'succeeded';
      state.reviews.push(action.payload);
    }
```

Рис. 35. Реалізація Редьюсерів та екстраредьюсерів reviewSlice.

Також в программі були реалізовані редюсери та екстраредюсери. Редюсери та екстраредюсери є основними інструментами для управління станом у Redux. Вони забезпечують зміни стану додатку на основі різних дій, що дозволяє ефективно керувати станом програми та взаємодіяти з компонентами.

- Редюсери відповідають за оновлення стану програми у відповідь на певні дії. Вони визначають, як саме має змінюватися стан при виконанні конкретної дії.
- Екстраредюсери допомагають обробляти асинхронні дії, наприклад, коли виконуються API-запити. Вони дозволяють реагувати на різні стани запиту (наприклад, успішне або неуспішне завершення) і автоматично оновлювати стан.

Приклад редюсерів та екстраредюсерів:

Редюсери:

- setUser: Оновлює стан користувача (наприклад, ім'я, email, роль).

- clearUser: Очищає всі дані користувача.

Екстраредюсери:

- userApiSlice.endpoints.signin.matchFulfilled: Змінює стан користувача при успішному вході в систему.
- productApiSlice.endpoints.fetchProducts.matchRejected: Оновлює стан при невдачі завантаження продуктів, зберігаючи повідомлення про помилку.

Ці механізми допомагають оптимізувати взаємодію компонентів, забезпечуючи актуальність стану та обробку асинхронних запитів.

Також для роботи программи використовувались хуки React такі як:

- useState – для збереження локального стану (пошукові запити, фільтрація товарів, кількість товару).
- useEffect – для побічних ефектів (запити до API, оновлення даних).
- useNavigate – для навігації між сторінками.
- useSelector – для отримання значень зі сховища Redux (дані користувача).
- useDispatch – для відправлення дій у Redux.
- useMutation та useQuery – для асинхронних операцій з API (отримання товарів, обробка покупок).
- useTonConnectUI – для інтеграції з гаманцем через TonConnect.

У файлі store.ts налаштовано Redux store для управління станом програми з використанням бібліотеки redux-persist для збереження даних між сесіями. Це дозволяє зберігати важливу інформацію, таку як дані користувачів, продукти чи замовлення, у localStorage, що забезпечує збереження стану навіть після перезавантаження браузера.

Основними частинами цього файлу є слайси (reducers), кожен з яких відповідає за управління різними аспектами стану програми. Наприклад, userReducer відповідає за збереження даних про користувача, productReducer — за продукти, orderReducer — за замовлення, а reviewReducer — за відгуки.

Завдяки використанню redux-persist, дані цих слайсів зберігаються в localStorage, що дає змогу користувачеві зберігати свій стан навіть після закриття або перезавантаження браузера.

Для забезпечення збереження стану програми використаввалась конфігурація persistConfig, в якій вказано, які саме дані мають зберігатися. Через метод persistReducer ці дані постійно оновлюються, що дозволяє користувачеві працювати з актуальними даними без необхідності повторно їх завантажувати після перезавантаження сторінки.

Крім того, для виконання асинхронних запитів до сервера була використована API slices для взаємодії з такими даними, як користувачі, продукти, замовлення та відгуки. Ці запити виконуються через middleware, що дозволяє асинхронно обробляти запити та оновлювати стан програми залежно від результатів запитів.

Отже, store.ts забезпечує централізоване управління станом застосунку та ефективну взаємодію між клієнтською частиною програми та сервером. Завдяки цьому додаток може зберігати важливі дані, обробляти запити до API та оновлювати інтерфейс без перезавантаження сторінки, що значно підвищує ефективність і зручність використання.

src > TS store.ts > ...

```ts
import { configureStore } from "@reduxjs/toolkit";
import { persistStore, persistReducer } from "redux-persist";
import storage from "redux-persist/lib/storage";
import userReducer from "./slices/user/userSlice";
import { userApiSlice } from "./slices/user/userApiSlice";
import productReducer from "./slices/product/productSlice";
import { productApiSlice } from "./slices/product/productApiSlice";
import reviewsReducer from "./slices/review/reviewSlice";
import { reviewApiSlice } from "./slices/review/reviewApiSlice";
import orderReducer from "./slices/order/orderSlice";
import { orderApiSlice } from "./slices/order/orderApiSlice";

const persistConfig = {
  key: "root",
  storage,
};

const persistedUserReducer = persistReducer(persistConfig, userReducer);
const persistedProductReducer = persistReducer(persistConfig, productReducer);
const persistedOrderReducer = persistReducer(persistConfig, orderReducer);

const store = configureStore({
  reducer: {
    user: persistedUserReducer,
    product: persistedProductReducer,
    order: persistedOrderReducer,
    review: reviewsReducer,
    [userApiSlice.reducerPath]: userApiSlice.reducer,
    [productApiSlice.reducerPath]: productApiSlice.reducer,
    [reviewApiSlice.reducerPath]: reviewApiSlice.reducer,
    [orderApiSlice.reducerPath]: orderApiSlice.reducer,
  },
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware().concat(
      userApiSlice.middleware,
      productApiSlice.middleware,
      reviewApiSlice.middleware,
      orderApiSlice.middleware
    ),
});

export const persistor = persistStore(store);

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

export default store;
```

Рис. 36. Реалізація store.ts.

Також реалізовано кастомні хуки. Кастомні хуки — функції, створені розробниками для повторного використання специфічної логіки у різних компонентах застосунку. Вони дозволяють абстрагувати повторювану логіку, таку як обробка запитів до сервера, взаємодія з формами або збереження стану в окремі функції.

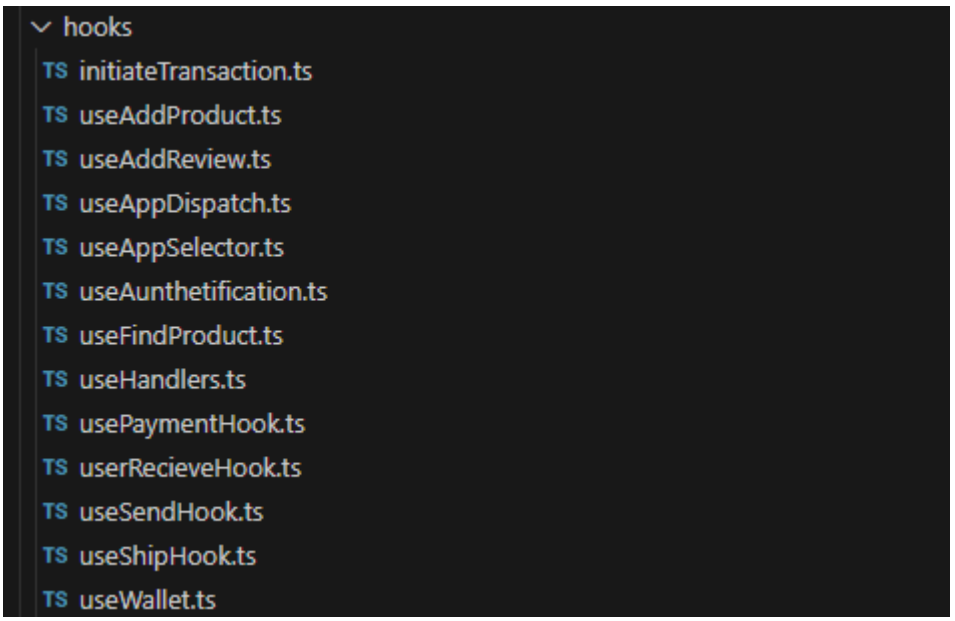


Рис. 37. Усі кастомні хуки.

Для прикладу короткий опис функцій, що виконують кастомні хуки в програмі:

- useAddProduct: Хук для додавання нового продукту. Він відправляє дані на сервер за допомогою useAddProductMutation та оновлює список продуктів у Redux.
- useWallet: Хук для роботи з гаманцем через TonConnect. Відслідковує підключення та відключення гаманця, оновлюючи адресу гаманця у стані.
- usePayment: Хук для обробки платежів. Використовує useBuyPaymentMutation для виконання транзакцій та ініціалізації транзакцій через useInitiateTransaction.
- useAddReview: Хук для додавання відгуку. Він використовує useAddReviewMutation для відправки відгуку на сервер і оновлює список відгуків у Redux після успішного додавання.
- useShipPayment: Хук для обробки платежів при відправці товару. Він ініціює транзакцію через useInitiateTransaction, а потім реєструє платіж на сервері через useShipPaymentMutation, оновлюючи статус і обробляючи можливі помилки.

```
export const useAddReview = () => {
  const [addReview, { isLoading, error }] = useAddReviewMutation();
  const dispatch = useDispatch();
  const [isReviewAdded, setIsReviewAdded] = useState(false);
  const reviews = useSelector(selectReviewByProductId);
  const handleAddReview = async (productId: number, reviewData: { rating: number; text?: string }) => {
    try {
      const response = await addReview({
        productId,
        rating: reviewData.rating,
        text: reviewData.text,
      }).unwrap();

      if (reviews) {
        dispatch(setReviews([...reviews, response]));
      } else {
        dispatch(setReviews([response]));
      }
      setIsReviewAdded(true);
      return { success: true };
    } catch (err: any) {
      console.error("Ошибка добавления отзыва:", err);

      const errorMessage = err?.data?.message || err?.message || "Неизвестная ошибка";

      return { success: false, error: errorMessage };
    }
  };
  return { handleAddReview, isLoading, error, isReviewAdded };
};
```

Рис. 38. Реалізація хука useAddReview.

Також у програмі були реалізовані різні компоненти такі як Button, NavBar, WalletConnectButton. У застосунку компоненти використовуються для організації та повторного використання функціоналу, що дозволяє створювати зручний і гнучкий інтерфейс. Всі ці компоненти виконують окремі ролі, забезпечуючи відповідні частини застосунку згідно із завданнями.

1. Компонент Button:

Цей компонент реалізує кнопку з можливістю передачі різних стилів і кольорів, а також обробки кліків. Завдяки цьому компоненту можна централізовано керувати стилями кнопок по всьому застосунку.

```tsx
interface ButtonProps {
    children: React.ReactNode;
    className?: string;
    active?: boolean;
    onClick?: () => void;
    textColor: string;
    backgroundColor?: string;
    style?: React.CSSProperties;
}

const Button = ({ children, className, active = false, onClick, textColor, backgroundColor, style }: ButtonProps) => {

    return (
        <button
            onClick={onClick}
            disabled={!active}
            style={{ color: textColor, backgroundColor: backgroundColor, ...style}}
            className={`${className} w-full text-center py-3 rounded-lg font-semibold`}
        >
            {children}
        </button>
    )
};

export default Button;
```

Рис. 39. Реалізація хука Button.

2. Компонент Navbar:

Компонент навігації відповідає за створення бокового меню та обробку переходів між сторінками застосунку. Для мобільних пристроїв передбачено адаптивне «бургер-меню», яке відкривається натисканням на іконку, економлячи місце на екрані. Дані користувача, такі як ім’я та електронна пошта, зберігаються в Redux-сховища.

3. Компонент WalletConnectButton:

Цей компонент управляє підключенням користувача до блокчейн-гаманця через TonConnect, дозволяючи підключати та відключати гаманець, а також виконувати транзакції через блокчейн. Компонент також надає зручний інтерфейс для підтвердження або скасування змін в адресі гаманця.

Усі ці компоненти вносять свій внесок у структуру застосунку, забезпечуючи його функціональність, зручність та масштабованість.

2.10 Застосування блокчейн-технологій у розробці клієнтської частини

У цьому проєкті використовувалися блокчейн-технології. Вони допомагають у забезпеченні безпеки транзакцій, роблять програму децентралізованою та зберігають інформацію про взаємодію під час купівлі товарів. Інтеграція блокчейн-технологій у клієнтській частині була реалізована за допомогою TON (The Open Network). Це надає можливість виконувати блокчейн-транзакції децентралізовано, тобто без посередників, що суттєво знижує витрати на адміністрування та підвищує ефективність обробки даних.

Основними компонентами, які використовує програма для роботи з блокчейном, є:

- TonConnect — сервіс, за допомогою якого можна прив’язати TON-гаманці до сторонніх ботів, програм або сайтів. Завдяки йому можна швидко здійснювати транзакції та обробляти платежі прямо на вказаний гаманець TON у TON Space. Підключення через TonConnect дозволяє користувачам мати надійний доступ до своїх криптовалютних активів та виконувати операції в безпечному середовищі.
- TON SDK — набір інструментів для розробників, який дозволяє працювати з TON-блокчейном, створювати контракти, підключати гаманці та здійснювати транзакції[7].
- Smart Contracts — протокол транзакцій, призначений для автоматичного виконання, контролю або документування подій і дій відповідно до умов контракту чи угоди. Смарт-контракти зменшують потребу в довірених посередниках, арбітражних витратах та імовірності шахрайства, а також скорочують кількість зловмисних або випадкових винятків.

Застосування блокчейн-технологій у моєму проєкті дозволяє зберігати конфіденційність даних, забезпечує високий рівень безпеки під час транзакцій і створює умови для безпосередніх, миттєвих та безпечних платежів.

2.12 Процес розгортання програми

У процесі деплою та розгортання моєї програми використовувались декілька технологій для забезпечення доступності та належної роботи клієнтської та серверної частини програми. Основними інструментами для розгортання стали Netlify[8], ngrok та Docker.

Для розгортання фронтенду було обрано Netlify. Це популярна платформа для хостингу статичних сайтів, що дозволяє швидко налаштувати процес деплою за допомогою інтеграції з репозиторіями на GitHub. Netlify автоматично створює процес деплою при кожному пуші в репозиторій, що значно спрощує підтримку актуальної версії програми.

Для тестування та налагодження API використовувався ngrok. Ngrok дозволяє створити тимчасову публічну URL-адресу для локального серверу, що забезпечує доступ до нього з Інтернету. Це дуже корисно під час розробки, коли сервер ще не розгорнутий на постійній хостинг-платформі.

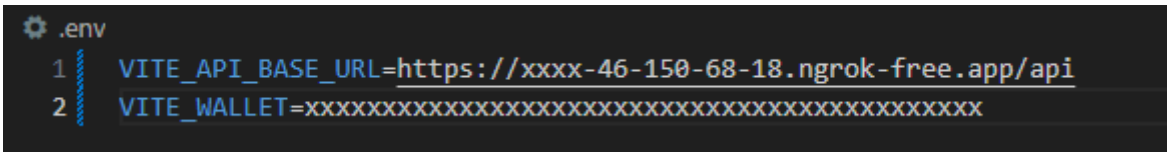


Рис. 40. Приклад налаштування .env під ngrok для фронтенду.

У цьому випадку, VITE_API_BASE_URL вказує на URL, створений ngrok для доступу до API. Це дозволяє фронтенду звертатися до серверної частини, що працює локально.

Для забезпечення сумісності середовищ та спрощення налаштувань серверної частини програми використовувався Docker. Docker дозволяє створювати контейнеризовані середовища, що можуть бути легко перенесені між різними машинами та операційними системами. Це забезпечує відсутність проблем з налаштуванням оточення, оскільки контейнер завжди працює однаково, незалежно від середовища.

Використання Docker для контейнеризації цієї частини програми дозволяє створити універсальне середовище для розгортання серверної частини як на локальних, так і на хмарних серверах. Docker забезпечує зручний спосіб упаковки всіх залежностей у контейнер, що робить розгортання програми на різних платформах швидким і простим. Завдяки використанню Netlify, ngrok та Docker, була можливість налаштувати безперешкодний процес деплою, тестування та контейнеризації програми, що значно покращує ефективність розробки та забезпечує стабільну роботу програми в різних середовищах.

2.13 Перспективи розвитку та удосконалення клієнтської частини системи

До планів розвитку та удосконалення клієнтської частини проєкту входять покращення функціональності та глобалізація програми. По-перше, у планах – додавання інших мов: наразі проєкт підтримує лише англійську, у подальшому планується інтегрувати іспанську, французьку, німецьку та португальську, що потенційно збільшить аудиторію користувачів сайту та дасть змогу залучити продавців і покупців з інших частин світу. Також до плану глобалізації входять розширення географії проєкту на країни Південної та Північної Америки, інших країн Європи та Азії, що приверне багато нових користувачів[9].

До перспектив проєкту також належить удосконалена взаємодія покупців і продавців. У цей пункт входять додавання підтримки листування між продавцем та покупцем, що зробить користувацький досвід більш позитивним.

Заплановано створення мобільного застосунку для платформ iOS та Android. Це дуже важливо, тому що ринок мобільних додатків швидко розвивається.

Для підвищення зручності використання можна також додати персоналізовані рекомендації на основі штучного інтелекту. Це зробить користування програмою більш зручним, а також покращить взаємодію із сайтом для покупців[10].

ВИСНОВКИ

Метою цієї дипломної роботи було розробити клієнтську частину сервісу для ефективного управління ланцюгами постачання за допомогою блокчейн-технологій. У ході виконання дипломної роботи було обґрунтовано її мету та актуальність. Було поставлено завдання щодо реалізації програми, після чого оглянуто основні використані технології. Далі описано функціональність програми за допомогою use case-діаграм, у яких показано види взаємодії покупців і продавців із сервісом.

У процесі написання роботи налаштовано проект, у якому налаштовано з усіма ключовими параметрами для його функціонування. Також розроблено візуальний макет сайту для всіх сторінок та секцій програми. Інтерфейс адаптовано під мобільні пристрої, оскільки це найпопулярніший спосіб взаємодії з інтернет-ресурсами.

Детально описано використання інтерфейсів, компонентів та хуків програми, адже саме на них базується логіка клієнтської частини. Для демонстрації результатів наведено процес розгортання через Netlify, ngrok та Docker, що дозволило показати повну функціональність сервісу.

У кінці дипломної роботи описано, як можна вдосконалити клієнтську частину в майбутньому, а саме: додати нові мови, розробити програмний застосунок, впровадити персоналізовані рекомендації тощо.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. PostgreSQL 17.4 documentation: веб-сайт. URL: https://www.postgresql.org/docs/current/index.html (дата звернення 13.03.25)
2. Node.js v23.11.0 documentation: веб-сайт. URL: https://nodejs.org/docs/latest/api/ (дата звернення 10.03.25)
3. React v19.1 documentation: веб-сайт. URL: https://react.dev/ (дата звернення 11.03.25)
4. Docker documentation: веб-сайт. URL: https://docs.docker.com/ (дата звернення 25.03.25)
5. TypeScript documentation: веб-сайт. URL: https://www.typescriptlang.org/docs/ (дата звернення 12.03.25)
6. @tonconnect/ui-react documentation: веб-сайт. URL: https://docs.ton.org/v3/guidelines/ton-connect/guidelines/developers (дата звернення 25.03.25)
7. TON SDK Official Documentation (2021). *TON SDK: Software Development Kit for the TON blockchain*. [Online]. Available: https://github.com/ton-blockchain/ton.
8. Netlify documentation: веб-сайт. URL: https://docs.netlify.com/ (дата звернення 30.03.25)
9. Townsend, P. L. *Internationalization and Localization Using Microsoft .NET* / P. L. Townsend, 2003. 384 с. (Addison-Wesley)
10. Kleppmann, M. *Designing Data-Intensive Applications* / M. Kleppmann, 2017. 600 с. (O'Reilly Media)