Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Розробка веб-застосунку з використанням технології React

Текстова частина до курсової роботи
за спеціальністю «Комп’ютерні науки»

Керівник курсової роботи
ст. викладач Борозенний С.О.
(прізвище та ініціали)

_____
(підпис)
“__” _____ 2022 р.
Виконав студент Кенийз В.Л.
“__” _____ 2022 р.

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав.кафедри інформатики,

доцент, к.ф.-м.н.

_____ О. П. Жежерун

(підпис)

“___” _____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту 4 року навчання БП «Комп'ютерні науки»
Кенийзу Віталію Людвиковичу
на тему:
Розробка веб-застосунку з використанням технології React

Зміст ТЧ до курсової роботи:

1. Вступ
2. Опис предметної області
3. Теоретичне підґрунтя
4. Реалізація застосунку
5. Висновки по роботі та рекомендації для подальших
6. Список використаних джерел
7. Додаток А
8. Додаток Б

Дата видачі “___” _____ 2022 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітки
1.	Отримання завдання на курсову роботу.	24.10.2021	
2.	Огляд технічної літератури за темою роботи.	15.12.2021	
3.	Аналіз актуальності теми	14.01.2022	
4.	Дослідження теоретичної частини	03.02.2022	
5.	Програмування практичної частини	21.03.2022	
6.	Демонстрація демо-проєкту науковому керівнику	18.04.2022	
7.	Кінцеві правки практичної частини	25.04.2022	
8.	Корегування роботи згідно зауважень наукового керівника	02.05.2022	
9.	Написання теоретичної частини	09.05.2022	
10.	Здача роботи для перевірки на плагіат	20.05.2022	

Студент _____

Керівник _____

“_____”_____

Зміст

Анотація

Дана робота присвячена розробці веб-застосунку та вивченню технології React. Було реалізовано веб-додаток на основі даної бібліотеки та допоміжних інструментів, які зазвичай використовуються у зв’язці з нею. Також робота містить пояснення та практичне використання можливостей бібліотеки до допоміжних пакетів для розробки програми

ВСТУП

Після зростання популярності соціальних мереж впродовж останніх десятиліть попит на рішення, яке потребує: гнучкості, управління величезним обсягом даних, швидкість обробки задач та інші фактори, почав стрімко зростати. І тут на сцену з'явився ReactJS, який є бібліотекою JavaScript, який використовується для створення повторно використовуваних компонентів, як зазначено в офіційної документації React. Нижче наведено визначення: React — це бібліотека для створення відокремлених інтерфейсів. [1] React надзвичайно гнучкий. Вивчивши його, Ви зможете використовувати його на різноманітних платформах для створення якісних інтерфейсів користувача. Його бібліотечний підхід дозволив стати йому таким відомим та зручним інструментом.

Одним з аспектів, які допомогли набрати шалену популярність цій бібліотеці – це концепція одної сторінки (SPA), ідея якої є уникнення перезавантаження застосунку для отримання вмісту веб-сторінки для оновлення інтерфейсу користувача. На відміну від традиційного способу перезавантаження веб-сторінки для отримання даних, використання SPA удосконалює цей етап взаємодії користувача з інтерфейсом та покращує продуктивність програми.

В міру зростання популярності React його екосистема також зросла, щоб охопити різні варіанти використання. Так наразі Ви можете створити статичний сайт за допомогою бібліотеки, використовувати React Native для створення мобільних додатків. Ви навіть можете створювати настільні програми за допомогою такого інструменту, як Electron, який може працювати на Mac і Windows за допомогою технології React.js.

Перелік прийнятих скорочень

API	Інтерфейс прикладної програми
CSS	Каскадні таблиці стилів
DOM	Об'єктна модель документа
ES6	Стандарт мови програмування
HTML	Мова розмітки гіпертексту
HTTP	Протокол передачі даних
I/O	Введення / Виведення
NPM	Менеджер пакетів для JavaScript
SPA	Одно сторінковий застосунок
VDOM	Віртуальний DOM
RAM	Пам’ять з довільним доступом
JSX	Розширення React до синтаксису мови JavaScript
UI	Інтерфейс користувача

РОЗДІЛ 1. ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Наразі швидкими темпами в наше життя приходить тема крипто валют. Вона є на слуху у всіх, і кожен певною мірою стикається і взаємодіє з цим явлення. Тому це стало ідеєю для створення застосунку, який мав стати мостом між користувачем, та крипто біржами і полегшити роботу в цій сфері, за допомогою спрощення певних процесів таких як: авторизація на біржі, виконання операцій, тощо.

Як відомо веб-застосунки використовують різні технологія за для обробки та пошуку інформації як от Express/Nest/Meteor тощо. Та HTML/JS для візуалізації інформації користувачу. Технологія React зарекомендувала себе продуктивністю, надійністю, підтримкою та широким функціоналом, який допоможе в реалізації даної ідеї. Також варто звернути увагу, що за потреби він буде легко масштабуватися і легко взаємодіятиме з іншими частинами програми. Також для серверної частини було обрано Express, який є зручним у використанні та середовища розробки Node, для швидкості виконання коду. Для зберігання даних було використано MySQL.

Отже, після визначення мети проєкту та вибору технологій за допомогою яких буде досягнено кінцевого результату було реалізовано веб-застосунок на основі вищевказаних технологій, а також допоміжних пакетів та бібліотек, серед яких JWT. Також слід зазначити, що в ході роботи з даним стеком технологій було виявлено, що за його допомогою була повністю досягнено поставлених цілей. Також знаходилися всі рішення виниклих проблем в ході роботи. Слід зауважити, що було досягнуто абстрактного розуміння переваг React в частковому випадку та розробки проєктів загалом.

РОЗДІЛ 2. ТЕОРЕТИЧНЕ ПІДРГУНТЯ

2.1 Основні поняття

Цей розділ присвячений більш глибокому зосередженні на технологіях, які використовувалися впродовж розробки застосунку. Ми на нижчому рівні зрозуміємо яким чином тим чи іншим технологіям вдається залишатися на піку ефективності, зручності та простоті, обходячи при цьому конкурентів або ж залишаючись з ними на високому рівні затребуваності та популярності.

2.2 Node

Якщо коротко, NodeJS прийшла на зміну інтернету без стану, який був заснований на парадигмі запитів і відповідей. І ось на сцені з'являється він, NodeJS, який блискуче себе показує в веб-додатках з двостороннім з'єднанням у реальному часі, використовуючи технологію вебсокетів. Працює це під капотом, досить цікаво. У порівнянні з традиційними методами веб-обслуговування, коли кожне з'єднання (запит) породжує новий потік, займаючи системну оперативну пам'ять (RAM) і в кінцевому підсумку вичерпуючи доступну кількість оперативної пам'яті, Node.js працює з одним потоком, використовуючи неблокуючий I/O виклик, що дозволяє йому підтримувати десятки тисяч одночасних з'єднань, що утримуються в циклі подій. [2] Ілюстрація роботи NodeJS в порівнянні з традиційним I/O *наведено в додатку А.1.*

Також слід розповісти про ще одну важливу перевагу NodeJS, а саме менеджер пакетів Node або npm, який дає змогу доступатися до сотень тисяч пакетів, зареєстрованих у системі NodeJS [3]. Реєстр npm є одним із найбільших у світі реєстрів пакетів, наразі в ньому нараховується один мільйон триста тисяч пакетів, величезна кількість з яких є відкритими за допомогою розробників та ентузіастів з усього світу. [4]

2.3 Express

Express.js — це фреймворк JavaScript, заснований на Node.js, який підтримує повноцінну розробку як на front-end, так і на back-end.

Згідно документації з офіційного веб-сайту Express, він являє собою структуру Node, яка забезпечує надійний та повноцінний набір інструментів та можливостей як для веб, так і для мобільних додатків [5]. Express більшу частину свого функціоналу покладає на проміжне програмне забезпечення [3]. Серед своїх сильних сторін Express також виділяє надшвидкий I/O, асинхронність та однопотоковість, структура схожа на MVC та надійний API, який робить маршрутизацію дуже легкою.

Експрес-додатки працюють, надсилаючи послідовність викликів на середній рівень. Проміжне програмне забезпечення (middleware) має двосторонній доступ до об’єктів запиту та об’єктів відповіді. Це означає, що використання платформи Express дає вам контроль над усіма об’єктами запиту та відповіді. Це дає вам можливість додавати сеанси, додавати параметри запитів та шаблони.

Нижче на *рисунку 2.1* продемонстрований ланцюжок від запуску запиту до останньої функції, яка надсилає відповідь.

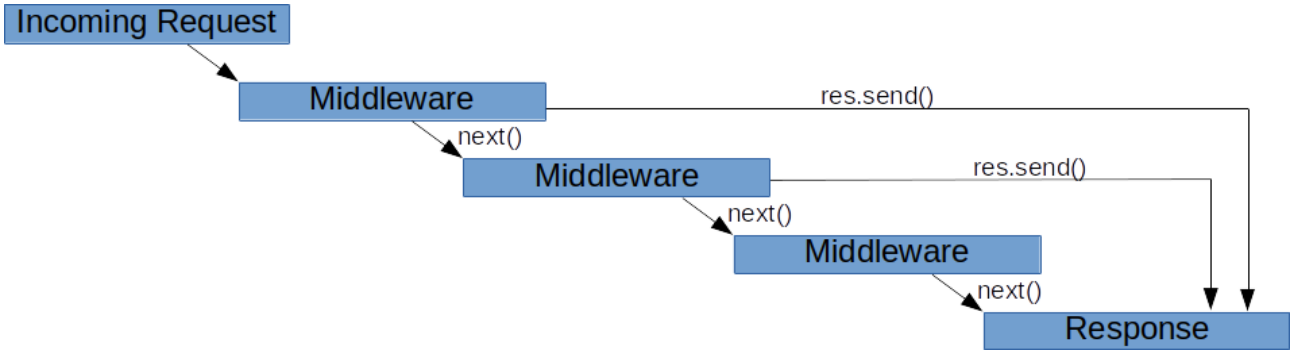

Рисунок 2.1 Запит-відповідь через проміжне програмне забезпечення [6]

Слід зазначити що існують і інші типи middleware, але головним залишається проміжне програмне забезпечення рівня маршрутизатора. Єдине чим воно відрізняється від інших middleware – це обмеження лише екземпляром маршрутизатора [7]

2.4 ReactJS

ReactJS, також відомий як React або React.js, є бібліотекою JavaScript з відкритим вихідним кодом для створення інтерфейсу користувача. Він використовується для обробки шару перегляду в SPA і розробці мобільних додатків. Його підтримують компанія Meta та спільнота розробників і корпорації. React прагне забезпечити швидкість, простоту та масштабованість. Деякі з його найбільш помітних особливостей це JSX, компоненти зі збереженням стану та VDOM.

Командою розробників Meta для вирішення різноманітних проблем веб-розробки або розробки мобільних додатків, які потребують динамічного відображення та стійкості при зростанні даних користувачів, було представлено великий функціонал, який дозволяє комплексно покрити вищезгадані потреби.

2.4.1 JSX

JSX є розширенням синтаксису ECMAScript без будь-якої визначеної семантики. React охоплює той факт, що логіка візуалізації по суті поєднана з іншою логікою інтерфейсу користувача. Замість того, щоб розділяти технології, React використовує слабко пов’язані одиниці, які називаються компонентами і містять обидві логіки. JSX необов'язковий для використання React. Однак JSX є хорошим наочним посібником під час роботи з UI всередині JavaScript. Це також дозволяє React більше слідкувати за кодом та видавати повідомлення про помилки та попередження.

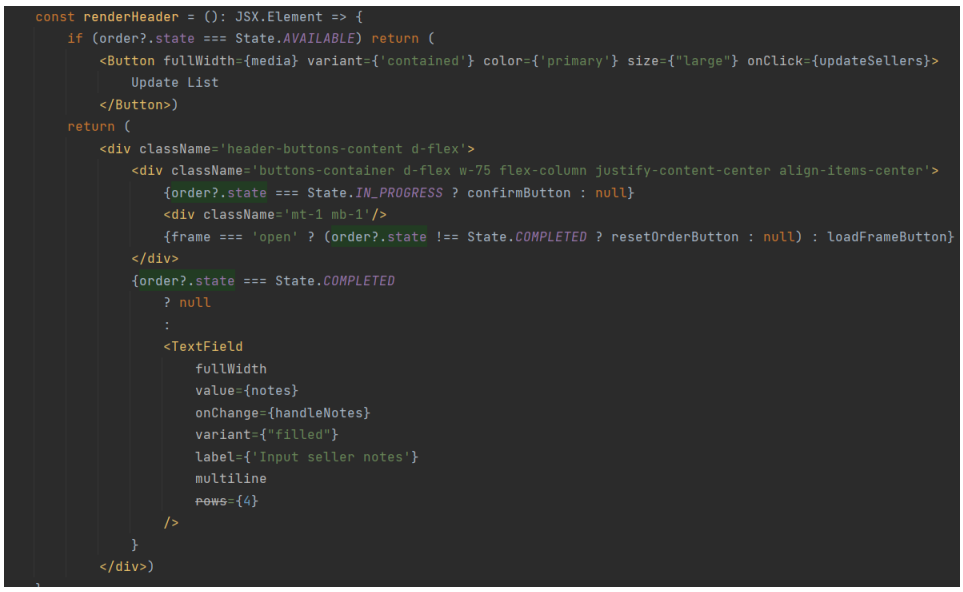

Рисунок 2.2 Приклад JSX

2.4.2 React Компоненти

React дозволяє розробникам розділити інтерфейс користувача на незалежні частини для більш зручного повторно відображення. Такі частини називаються компонентами React. Ці Компоненти реалізують метод візуалізації, який приймає вхідні дані та повертає те, що потрібно відобразити. Кожен компонент має кілька методів життєвого циклу, які можна замінити для виконання коду в певний час протягом процесу. Методи можна викликати за допомогою Reacts API. State — це простий об’єкт JavaScript, який використовується для запису та реакції на події користувача. Кожен визначений компонент на основі класу має власний об’єкт стану. Щоразу, коли стан компонента змінюється, компонент і всі його дочірні компоненти негайно обновляються. Стани зберігають значення в усьому компоненті і можуть передаватися дочірнім компонентам як пропси.

Усі компоненти React - це чисті функції, які не змінюють свої властивості – це строге правило, якого дотримуються всі компоненти. Пропс - це набір

вхідних даних, переданих як параметри компоненту. Отже, компонент React працює як чиста функція, і кожен раз повертає ідентичний результат для тих самих вхідних параметрів (пропсів).[8]

2.4.3 React Хуки

З самого початку в React були наявні класи. На заміну їм прийшли хуки. Хуки — це функції, які «підключаються» до функцій стану та життєвого циклу React із компонентів функцій. Це не означає, що з класами щось не так. Хуки дозволяють значно спростити ваш код і покращать можливість його повторного використання. Також їх введення допомогло викорінити використання слова «this».

Рисунок 2.3 Відмінність компоненту класу та хука [9]

Хук useState можна викликати всередині компонента функції, щоб додати до нього локальний стан. React збереже цей стан між відображеннями. useState повертає масив із двома елементами:

- значення поточного стану
- функція, яка дозволяє оновлювати її (наприклад, викликати її з обробника подій)

Єдиним аргументом useState є початковий стан. Одна важлива відмінність між this.setState в компонентах класу полягає в тому, що за допомогою хуків оновлення змінної стану завжди замінює її замість злиття. [10]

Хук useEffect додає можливість виконувати побічні ефекти від компонента функції. Він служить тій же цілі, що й componentDidMount, componentDidUpdate та componentWillUnmount у компонентах на основі класу, але об'єднаний в єдиний API. За замовчуванням React запускає ефекти після кожного відображення, включаючи початковий. React гарантує, що DOM було оновлено до моменту запуску ефектів. [10]

Також слід зауважити що існують інші хуки призначені для різноманітних задач. І в той же час є можливість створення власних хуків для покриття специфічних та не шаблонних потреб.

```
const [cardNumber, setCardNumber] = useState( initialState: '');
const [bank, setBank] = useState( initialState: '');
const [amount, setAmount] = useState( initialState: 0);
const [loading, setLoading] = useState( initialState: false);
const [dialogOpen, setDialogOpen] = useState( initialState: false);

const {orderId}: { orderId: string } = useParams();

const media = useMediaQuery( query: '(max-width: 768px)');

useEffect( effect: () => {
    (async () => {
        setLoading( value: true);

        const [err, order] = await to(orderService.getOrderById(orderId));
        if (err) throw err;

        setCardNumber( value: order?.card.cardNumber || '');
        setBank( value: order?.card.bank || '');
        setAmount( value: order?.amount || 0);

        const cardList = await binanceService.getUserPayMethods()
        const userCards = cardList.map((card: any) => Customer.parseResponse(card));
        const cardPresent = userCards.some((card: CardProps) =>
            card.cardNumber === order?.card.cardNumber && card.tradeMethodName
                .replace( searchValue: /\s/g, replaceValue: '') === order?.card.bank);
        setCardIsAdded(cardPresent);

        setLoading( value: false);
    })();
}, deps: []);
```

Рисунок 2.4 Приклад використання хуків

2.4.4 VDOM

HTML DOM спочатку був призначений для статичних сторінок і, таким чином, не був оптимізований для створення динамічного інтерфейсу користувача. Коли DOM оновлюється, він повинен оновити кожен вузол і повторно намалювати сторінку відповідним CSS і макетом. Зазвичай SPA містить тисячі динамічно згенерованих вузлів, до яких приєднано слухачі подій. На динамічних сторінках HTML DOM повинен перевіряти зміни в даних кожного вузла через певні інтервали. Це значно знижує продуктивність програми. VDOM був винайдений як рішення цієї неефективності.
VDOM є абстракцією HTML DOM. Він легкий і відокремлений від браузера. Його можна оновити, не впливаючи на фактичний DOM. React має VDOM, вбудований у модуль під назвою ReactDOM. Коли надходять оновлення, React використовує процес, який називається узгодженням, використовуючи алгоритм, який порівнює та зіставляє зміни, щоб знати, які елементи потребують оновлення. React змінить лише ці елементи, не вплинувши на інші. [11]. Відмінність в роботі проілюстровано на *рисунку 2.5*.

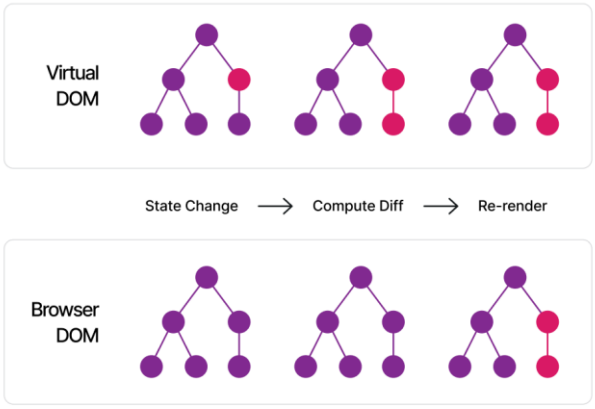

Рисунок 2.5 Порівняння VDOM та Браузерного DOM на різних стадіях зміни стану [12]

2.5 Аутентифікація та авторизація за допомогою JWT

Аутентифікація - це акт підтвердження того, що користувач є тим, за кого себе видає, в той час як авторизація - це процес надання користувачеві дозволу на доступ до певного ресурсу або функцій. Після встановлення того, ким є користувач, та які привілеї він має доступ до ресурсу буде дозволений або відхилений. [13] Серед різноманітних способів аутентифікації та авторизації для веб-додатків опублікована відносно недавно технологія JSON Web Token (JWT) реалізувала нову концепцію збереження статусу користувача [3].

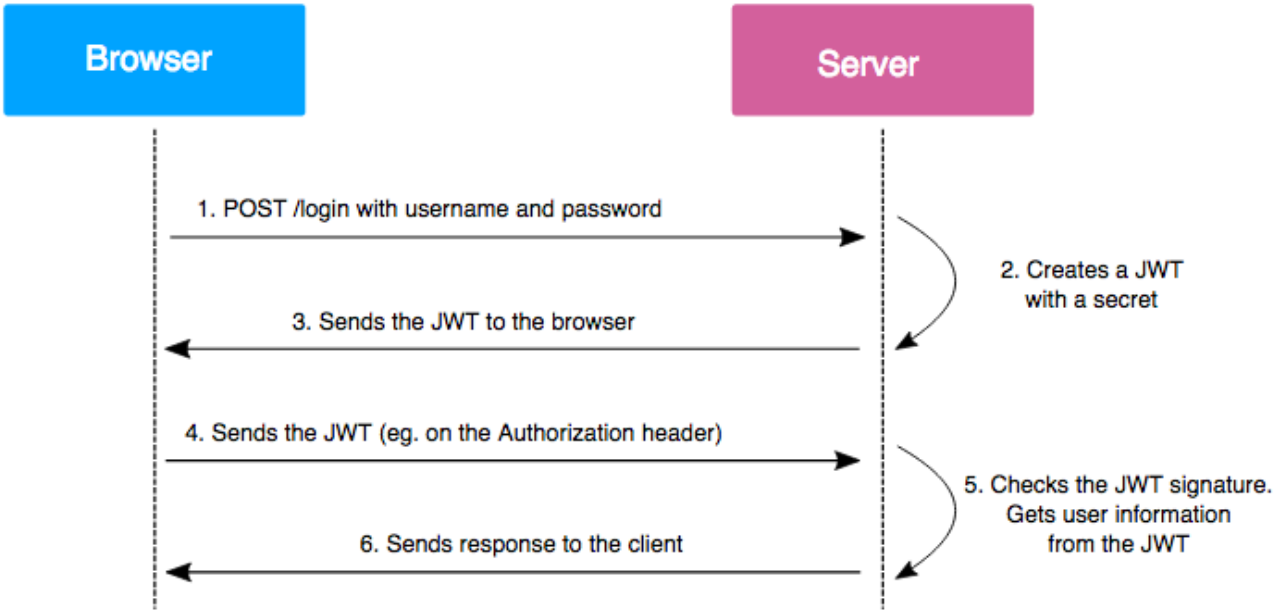

Рисунок 2.6 Процес авторизації JWT [14]

Однією з переваг JWT токенів є те, що їх можна використовувати без резервного сховища. Вся інформація, необхідна для автентифікації користувача, міститься в самому токен. Сервіс потребує лише певного проміжного програмного забезпечення для обробки токена (бібліотеки JWT відкрито доступні для всього, від Express до JVM MVC Frameworks), а також секретний ключ, необхідний для перевірки. Перевірка полягає в підтвердженні підпису та кількох параметрів, таких як термін дії токена. JWT зазвичай має

середній термін дії, а перевірка того, що токен правильно підписаний не вимагає доступу до I/O або доступу до мережі та дуже легко масштабується на сучасному обладнанні веб-сервера.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Розробка додатку

В основі застосунку лежить вбудовування певних сторінок з веб-додатку крипто біржі Binance за допомогою її API. Основним викликом стало витягування потрібної інформації з запитів до API та відображення її в зручному для користування інтерфейсі. Також не менш важливим є те, що загальнодоступної документації не існує, тому довелося в ручну відслідковувати та імітувати ці запити для досягнення мети. Зрештою було досягнено автоматичного підтягування всієї потрібної інформації та зменшення часу на операції, які виконуються на крипто біржі, за допомогою спрощеної перевірки даних.

3.2 Інтерфейс та функціонал

3.2.1 Сторінка входу

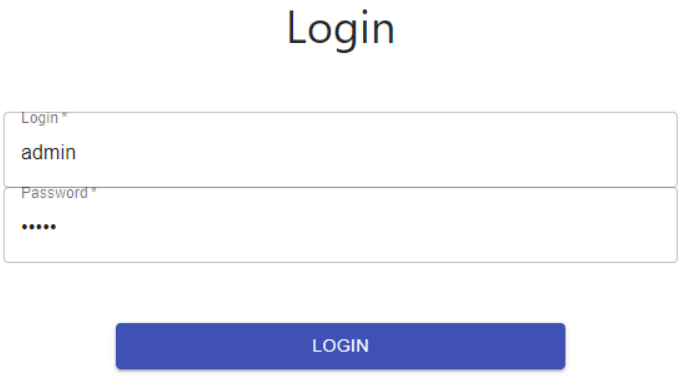

Рисунок 3.1 Інтерфейс входу до застосунку

Інтерфейс входу простий та зрозумілий, на цій сторінці користувач може зробити вхід в застосунок для використання функціоналу, який надається тільки зареєстрованим користувачам.

3.2.2 Основна сторінка

Головна сторінка - це сторінка замовлень, яка є за замовчуванням, на неї користувач потрапляє після успішної авторизації. Також на даній сторінці користувач бачить весь можливий функціонал, який розташований в верхньому лівому куті сторінки та може взаємодіяти з конкретними вікнами.

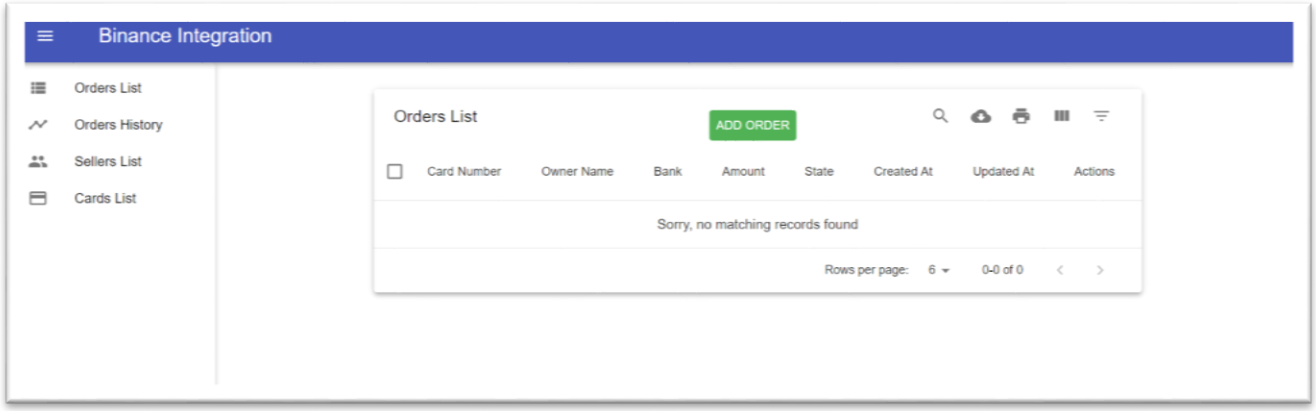

Рисунок 3.2 Головна сторінка застосунку

3.2.3 Сторінка замовлень

Отже на сторінці Orders є функціонал для додавання замовлень, які цікавлять користувача, який представляє собою спливаюче вікно, в якому потрібно ввести карту, на які потрібно зробити переказ, та суму. Також ці поля проходять перевірку на правильність введення інформації, для безпеки користувача. На *рисунку 3.3* продемонстровано вже створене замовлення на задньому фоні та спливаюче вікно, для створення ще одного.

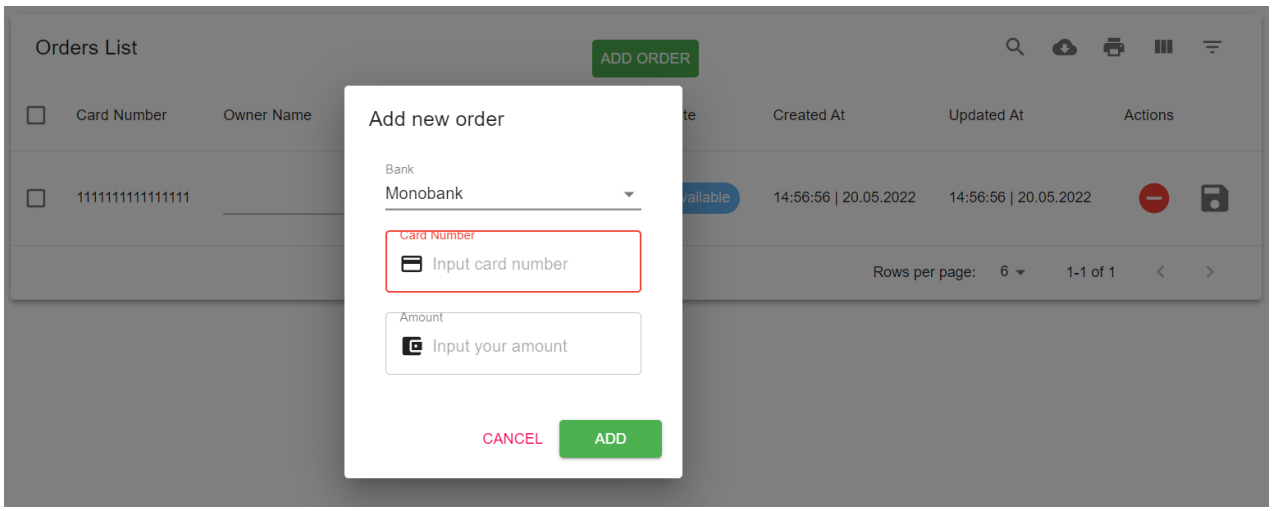

Рисунок 3.3 Створення замовлень та сторінка Orders

Після цього користувач може обробити це замовлення натиснувши на нього і перейшов на наступну інтерфейсу сторінку інтерфейсу, де він має додати цю карту до свого гаманця Binance натиснувши на кнопку біля номеру карти, після чого при успішній обробці цього запиту на крипто біржу, висвітиться сповіщення, та всі можливі продавці, які підходять параметрам замовлення будуть під вантажені в наш інтерфейс з біржі. Користувач може обрати підходящого для нього продавця і продовжити операцію.

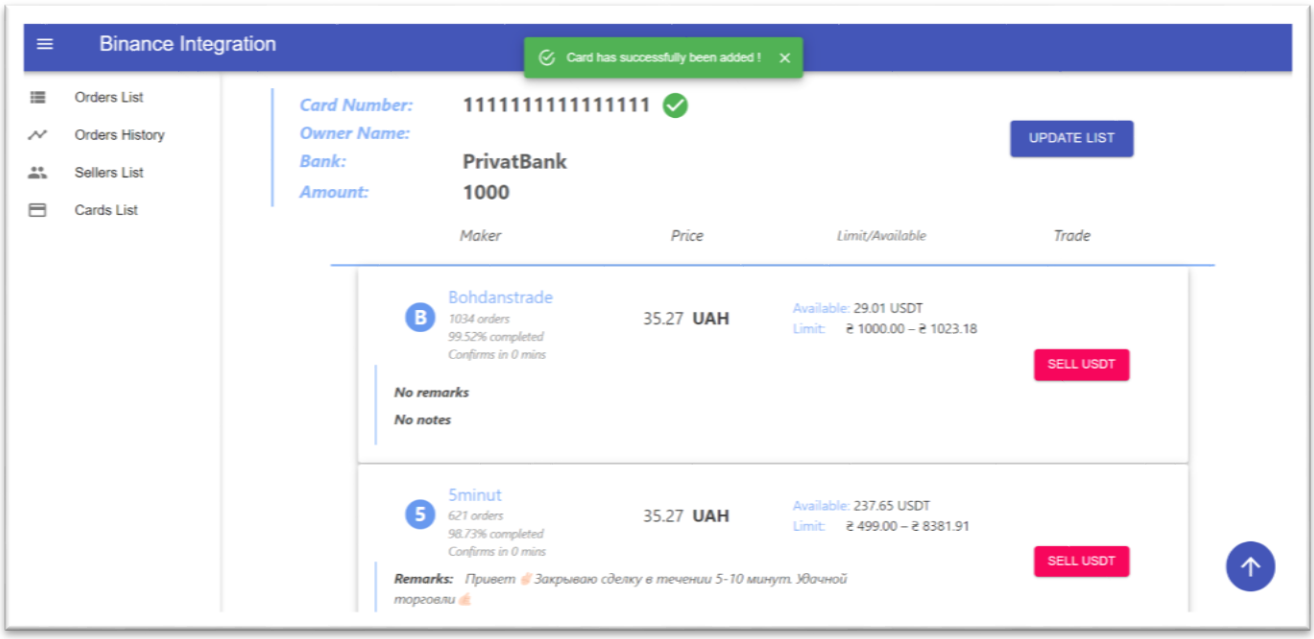

Рисунок 3.4 Інтерфейс з списком ордерів з крито біржі Binance

3.2.4 Сторінка історія замовлень

Як показано *на рисунку 3.5*, на сторінці Orders History користувач бачить історію всіх своїх замовлень та може переглянути інформацію по кожному замовленню та деталі кожного з них.

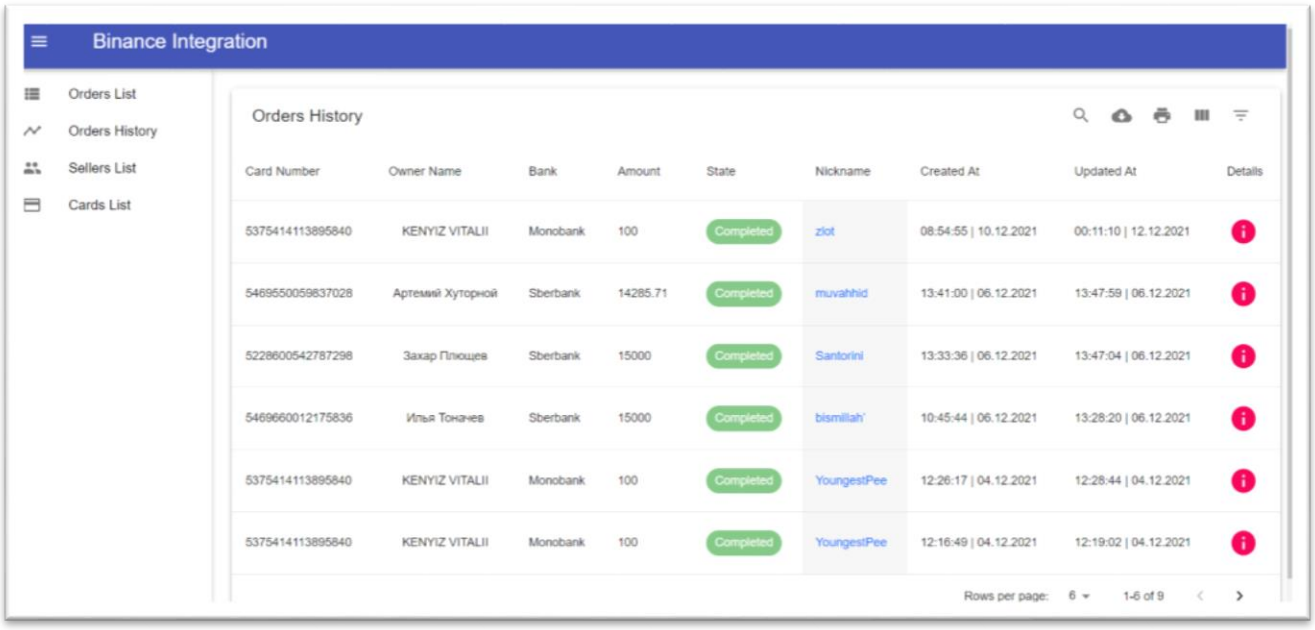

Рисунок 3.5 Сторінка історії замовлень

Деталі замовлення представляють собою фрейм з веб-сайту крипто біржі, в якому було виконано цю операцію, з повною інформацію про замовлення, для легкого розуміння. Це наглядно видно на *рисунку 3.6*.

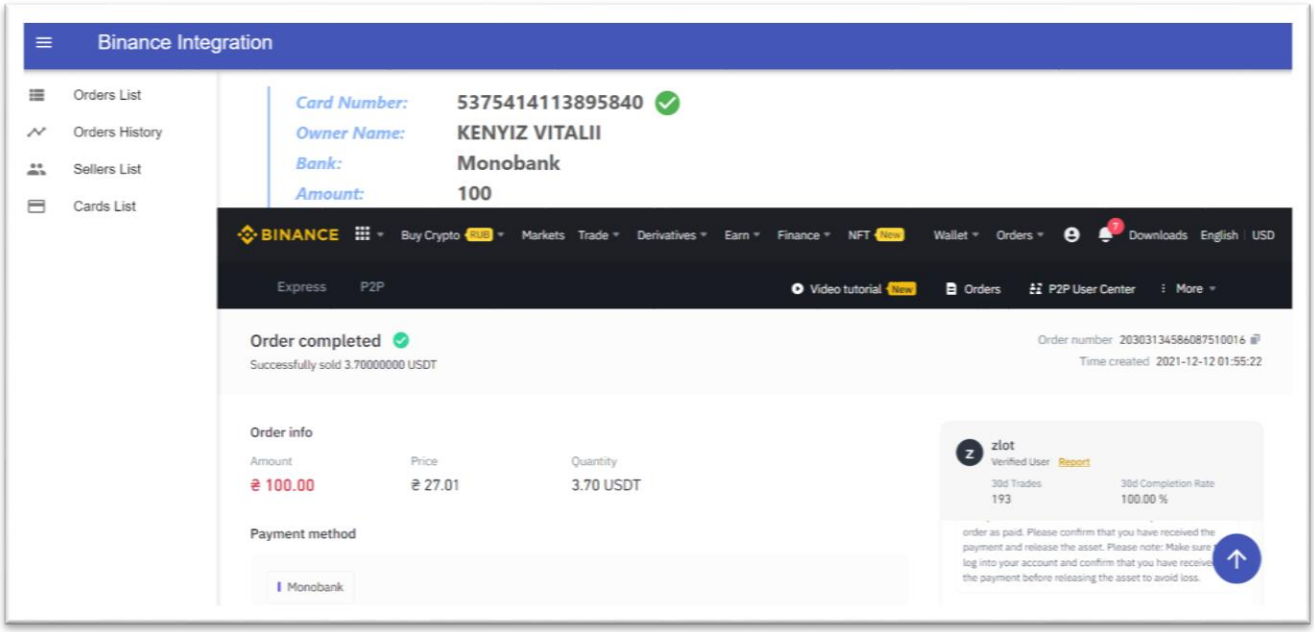

Рисунок 3.6 Фрейм з крипто біржі та деталі замовлення

3.2.2 Сторінка продавців

Сторінка Sellers демонструє користувачу всіх продавців з крипто біржі, з якими він взаємодіяв та інформацію про них, яка була взята з крипто біржі, та нотаток залишених самим користувачем. Це видно на *рисунку 3.7*.

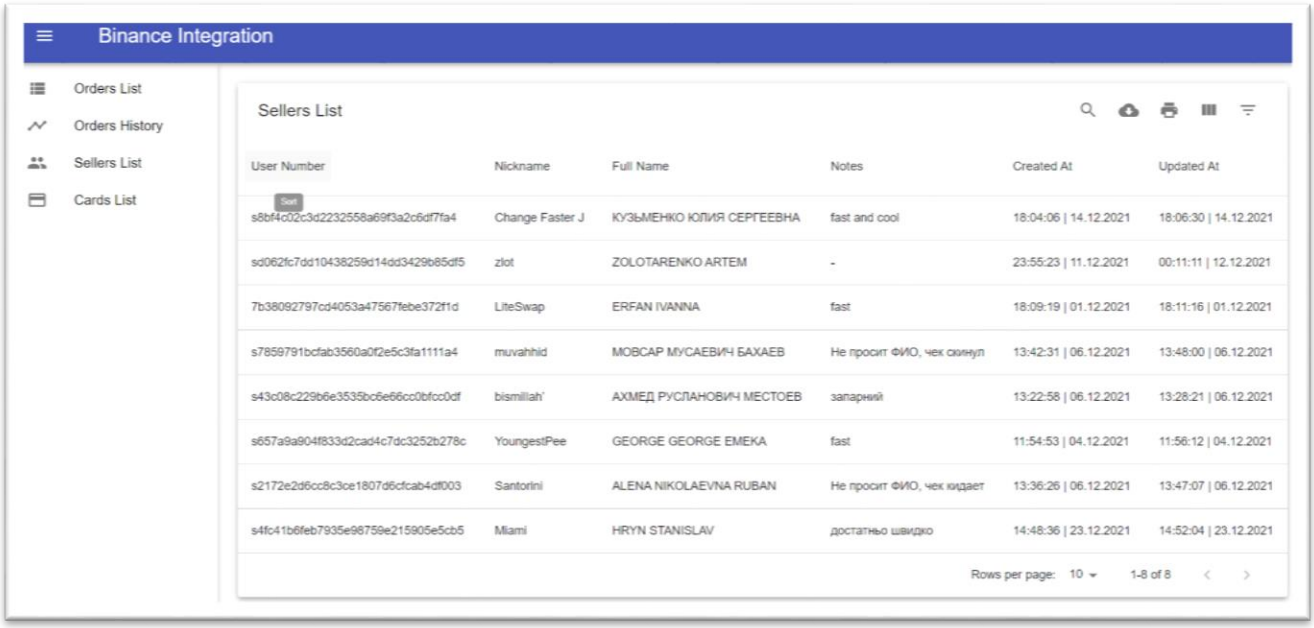

Рисунок 3.7 Сторінка продавців

Також натиснувши на конкретного продавця, відкриється сторінка з історію всіх замовлень, які були виконані ним для користувача. На даній сторінці повторно використовується сторінка Orders History та додатково певні відмінності Це продемонстровано на *рисунку 3.8*

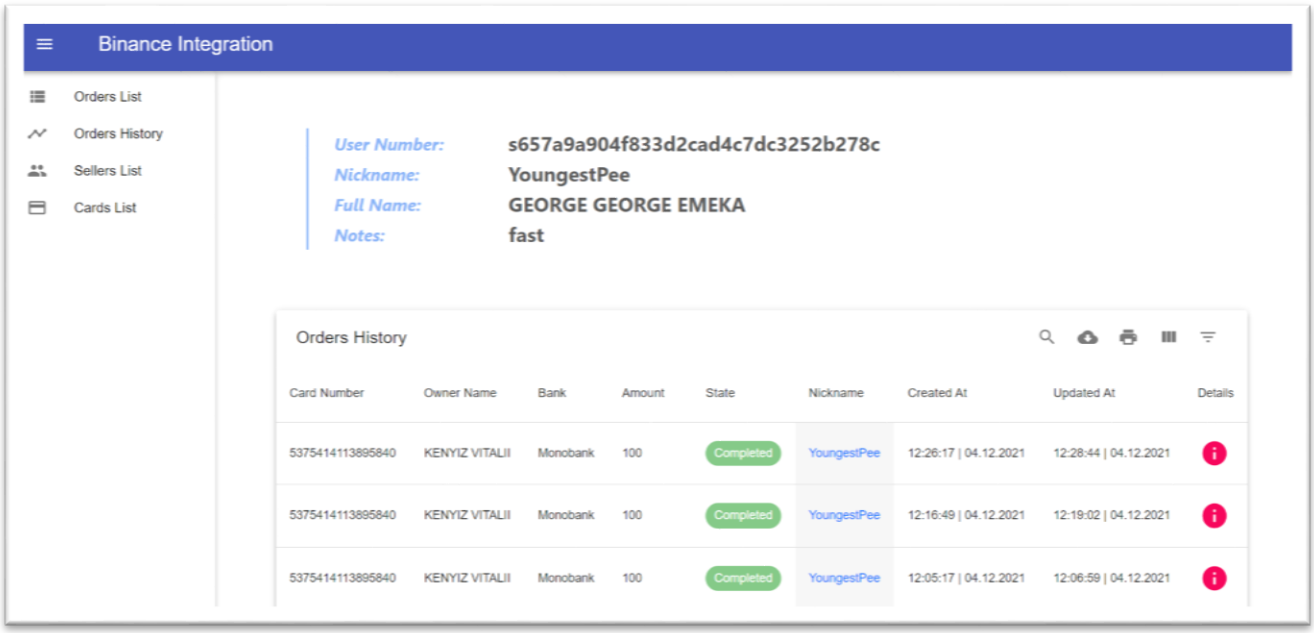

Рисунок 3.8 Замовлення виконані певним продавцем

3.2.2 Сторінка карт

Сторінка Card List слугує для того, щоб користувач в зручний спосіб міг переглянути карти, які знаходять в його гаманці на крипто біржі Binance та за потреби зміг маніпулювати ними, для подальших створень замовлень, тощо. Це видно на *рисунку 3.9.*

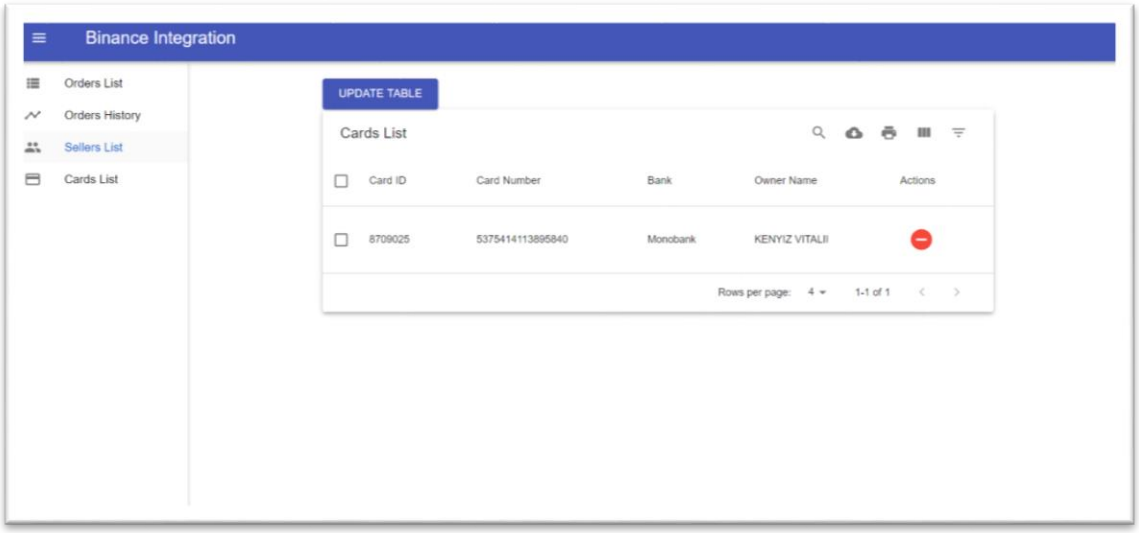

Рисунок 3.9 Сторінка зі списком карт користувача

ВИСНОВКИ ПО РОБОТІ ТА РЕКОМЕНДАЦІЇ ДЛЯ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

У роботі описано підходи та переваги розробки застосунків з використанням бібліотеки React та допоміжних технологій. Було проаналізовано та застосовано на прикладі переваги цієї бібліотеки та здобуто більш глибоке розуміння всіх нюансів та особливостей даної бібліотеки.

Загалом курсова робота містить опис певних підходів до вирішення поставлених питань, розглядає також допоміжні технології, які були використанні у зв’язці з бібліотекою React. На основі досліджень було розроблено систему, яка містить зручний та ефективний функціонал, використовує можливості та переваги бібліотеки, за для покращення розробки та користування веб-застосунками.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Getting Started [Електронний ресурс] // React. – 2022. – Режим доступу до ресурсу: https://reactjs.org/docs/getting-started.html

2 Tomislav C. Why The Hell Would I Use Node.js? A Case-by-Case Tutorial [Електронний ресурс] / Capan Tomislav // Toptal. – 2021. – Режим доступу до ресурсу: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

3 Hoque S. Full-stack React Projects Second Edition. Birmingham, UK [Електронний ресурс] / Shama Hoque // Packt Publishing. – 2020. – Режим доступу до ресурсу: https://books.google.com.ua/books?id=097dDwAAQBAJ&pg=PP3&lpg=PP3&dq=4+Shama+Hoque.+Full-stack+React+Projects+Second+Edition.+Birmingham,+UK:+Packt+Publishing;+2020.&source=bl&ots=CMF47nj6s2&sig=ACfU3U0_A4Cs-goszkE0o6joVbzwBKw3YQ&hl=uk&sa=X&ved=2ahUKEwilk6_x8cvwAhVps4sKHQpyB40Q6AEwB3oECAgQAw#v=onepage&q=4%20Shama%20Hoque.%20Full-stack%20React%20Projects%20Second%20Edition.%20Birmingham%2C%20UK%3A%20Packt%20Publishing%3B%202020.&f=false

4 npm (software) [Електронний ресурс] // Wikipedia. – 2022. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Npm_(software)

5 Express [Електронний ресурс] // Express. – 2022. – Режим доступу до ресурсу: https://expressjs.com.

6 What is middleware? How it works in nodeJS? A Simple implementation [Електронний ресурс] // medium. – 2019. – Режим доступу до ресурсу: https://medium.com/dataseries/what-is-middleware-how-it-works-in-nodejs-a-simple-implementation-485bcb9c3d53

7 Using middleware [Електронний ресурс] // Express. – 2022. – Режим доступу до ресурсу: https://expressjs.com/en/guide/using-middleware.html

8 Components and Props [Електронний ресурс] // React. – 2022. – Режим доступу до ресурсу: https://reactjs.org/docs/components-and-props.html

9 Bishard E. The Guide to Learning React Hooks (Examples & Tutorials) [Електронний ресурс] / Eric Bishard // KendoReact. – 2021. – Режим доступу до ресурсу: https://www.telerik.com/kendo-react-ui/react-hooks-guide/

10 Heruc K. A high level overview of React Hooks [Електронний ресурс] / Kristian Heruc // React Tricks. – 2019. – Режим доступу до ресурсу:

https://reacttricks.com/a-high-level-overview-of-react-hooks/

11 ReactJS | Virtual DOM [Електронний ресурс] // GeekforGeeks. – 2022. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/reactjs-virtual-dom/

12 Goncharenko O. Vue vs. React — Which One To Choose in 2022? [Update May-June] [Електронний ресурс] / Oleg Goncharenko // BROCODERS. – 2022. – Режим доступу до ресурсу: https://brocoders.com/blog/react-vs-vue-comparison-2021/

13 Zuraida A. Authorization and Authentication [Електронний ресурс] / Audira Zuraida // Meduim. – 2018. – Режим доступу до ресурсу: https://medium.com/@audira98/authorization-and-authentication-cd0a7c994278

14 Garcia R. JWT tokens and security – working principles and use cases [Електронний ресурс] / Romain Garcia // VAADATA. – 2016. – Режим доступу до ресурсу: https://www.vaadata.com/blog/jwt-tokens-and-security-working-principles-and-use-cases/

Додаток А

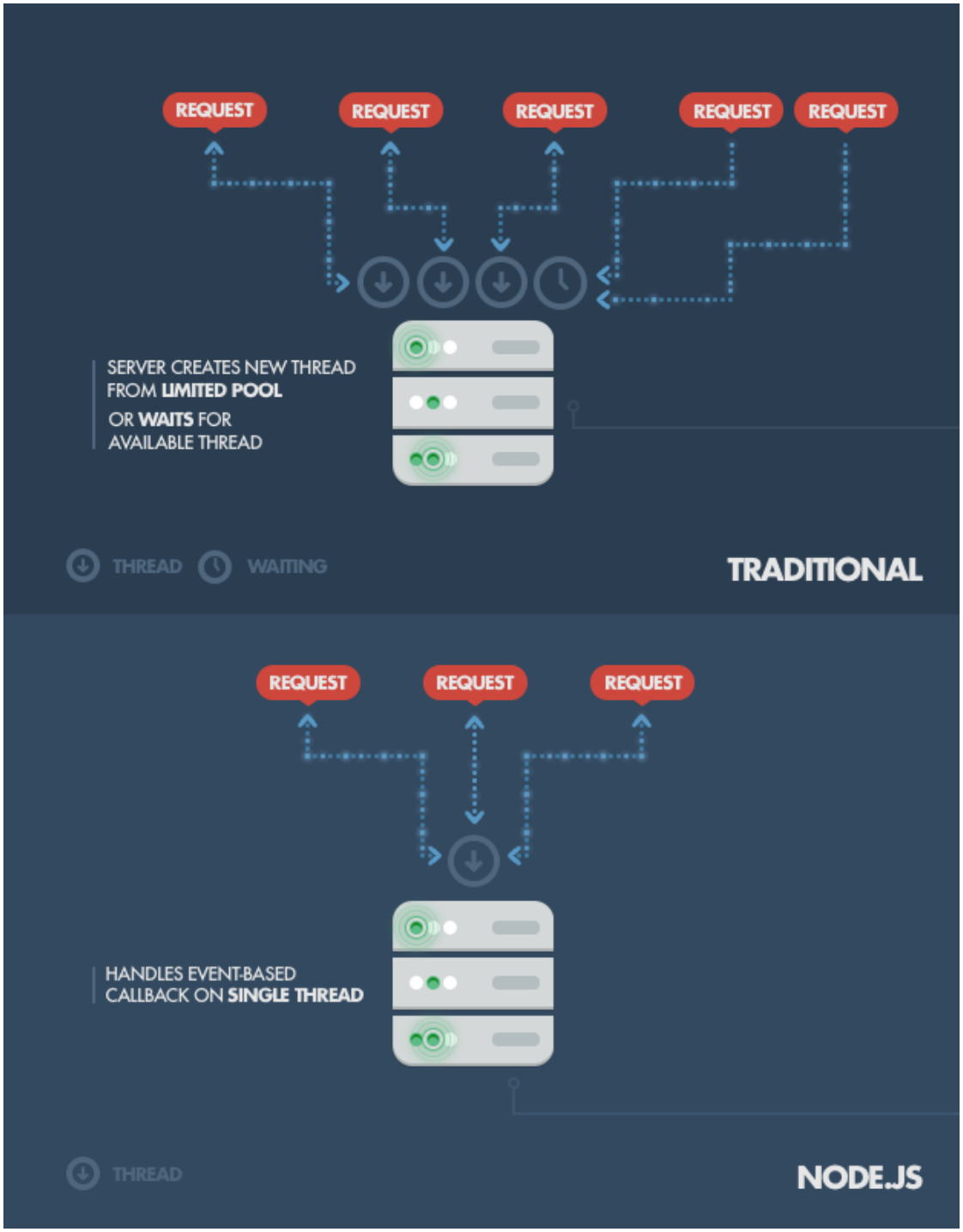

Рисунок 1. Робота звичайного I/O в порівнянні з NodeJS [2]

Додаток Б
Рисунки вихідних кодів

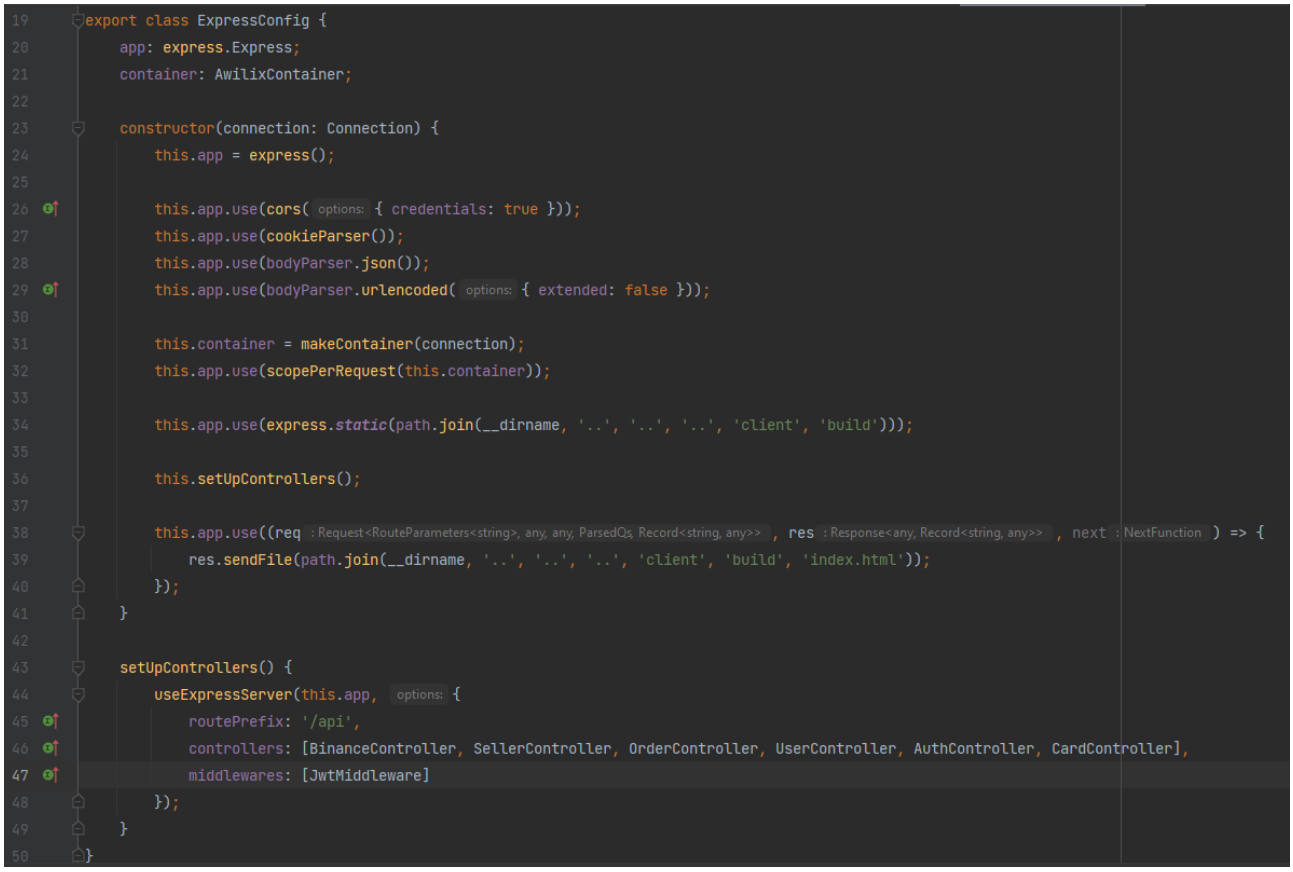

Рисунок 1 Налаштування серверу

```
@JsonController( baseRoute: '/auth')
export class AuthController {

    @Get( route: '/authenticate')
    async authenticate(@Req() req: ContainerReq, @Res() res: Response): Promise<any> {
        const { authService }: { authService: AuthService } = req.container.cradle;

        const token = req.cookies.token;
        if (!token) return res.status( code: 401);

        const [err, user] = await to(authService.authenticate(token));
        if (err) throw err;

        const newToken = generateJwtToken( user: {
            id: user.id,
            login: user.login,
        });

        res.cookie( name: 'token', newToken);

        return {
            message: 'Authentication has been successful',
            user,
            token: newToken
        }
    }

```

Рисунок 2 Контролер аутентифікації за допомогою JWT

```
@Middleware( options: { type: 'before' })
export class JwtMiddleware {
    use(request: ContainerReq, response: Response, next: NextFunction): any {
        if (request.originalUrl.includes('/auth')
            || request.originalUrl.includes('/login')
            || request.originalUrl.includes('/outer')
        ) {
            return next();
        }

        const token = request.cookies.token;

        let jwtPayload: any;

        try {
            jwtPayload = jwt.verify(token, process.env.JWT_SECRET);
        } catch (e) {
            response.status( code: 401).json( body: { msg: ' User not logged in'} );
            return;
        }

        const data = jwtPayload.data;
        const newToken = generateJwtToken( user: { id: data.id, login: data.login });

        response.cookie( name: 'token', newToken);

        next();
    }
```

Рисунок 3 Використання проміжного програмного забезпечення (middleware)