Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Кафедра інформатики факультету інформатики

Розробка веб-сайту для інтернет аптеки

Текстова частина до курсової роботи

за спеціальністю “Комп’ютерні науки”

Керівник курсової роботи
к.т.н., ас. Калітовський Б.В

_____

“____” _____ 2021 р.

Виконав студент

Задорожний М. В.

“____” _____ 2021 р.

Київ 2021

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

ЗАТВЕРДЖУЮ
Зав.кафедри мультимедійних систем,
доцент, к.ф.-м.н.
_____ Жежерун О.П.
(підпис)
„____”_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту: Задорожному Максиму **факультету:** Інформатики **курсу** - 3

ТЕМА: **Розробка веб-сайту для інтернет аптеки**

Вихідні дані:
-
-
Зміст ТЧ до курсової роботи:
Індивідуальне завдання
Вступ
1 Аналіз предметної області
2 Використані технології
3 Проектування і розробка
Висновки
Список літератури
Додатки

Дата видачі „___” _____ 2021 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Календарний план виконання курсової роботи

Тема: Розробка веб-сайту для інтернет аптеки

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Розгляд та аналіз проблеми	Вересень - грудень 2019р.	
2.	Розробка архітектури та програми аналіз технологій	Січень- лютий 2020р	
3.	Розробка back-end частини	Лютий- березень 2020р.	
4.	Розробка front-end частини	Березень-квітень 2020р.	
5	Написання текстової частини	Квітень 2020р.	
6.	Перегляд праці науковим керівником	Травень 2020р.	
7.	Підготовка до презентації роботи	Травень 2020р	
8.	Презентація курсової роботи	Травень 2020р.	

Студент Задорожний М.В _____

Керівник Калітовський Б.В. _____

“_____” _____ 2020 р.

Зміст

Анотація

Метою даної курсової роботи є розробка веб-додатку на React для інтернет аптеки.

Проведено аналіз аналогічних веб-додатків, які специфікуються на онлайн продажу фармацевтичних засобів. Було проаналізовано основні переваги і недоліки таких веб-додатків.

Провівши аналіз, було визначено основний функціонал, який потребує подібні інтернет аптеки.

Вступ

Електронна комерція - це можливість електронної покупки або продажу в Інтернеті.

За останній рік ринок електронної комерції в Україні виріс на 40% до 107 млрд. гривень, таким чином обсяг ринку зріс на 41% порівняно з 2019 роком, кількість онлайн-платежів також зросла - щонайменше на 50%. Зараз майже 9% усіх покупок в Україні відбувається в Інтернеті - на ринках, в Інтернет-магазинах та соціальних мережах. І за прогнозами такий темп розвитку збережеться ще як мінімум в 2021 році[1].

На такий бурхливий розвиток не в останню чергу вплинула пандемія. Через що на ринку з’явилось багато нових гравців, які до того працювали лише в офлайні, наприклад спеціалізовані магазини, кафе, кав’ярні і навіть невеликі підприємства.

Однією з категорій товарів, яка користується найбільшим попитом на провідних маркетплейсах і до цього часу була традиційно офлайновим сегментом, є категорія медичних товарів, оскільки стало набагато простіше і безпечніше замовити товар ніж стояти в офлайн чергах, мандрувати містом в пошуку потрібних ліків і ризикувати бути зараженим.

Отже, ринок електронної комерції швидко росте і буде рости в найближчому часі, оскільки, навіть якщо пандемія закінчиться, мало вірогідно, що люди відмовляться від зручності онлайн покупок. А тому це створює потребу у великій кількості веб-сервісів для продажу товарів в Інтернеті, які були б зручними у користуванні не тільки клієнтам, а й власникам і адміністраторам цих сайтів.

Мета курсової роботи:

Розробити зручний у користуванні веб-додаток для інтернет аптеки паралельно вивчаючи та засвоюючи навички у роботі з React.js.

Розділ 1. Аналіз предметної області

1.1 Проблематика та актуальність

Розвиток e-commerce неминучий. Все більше і більше українців надають перевагу покупкам онлайн, особливо у час пандемії. Все більше і більше підприємців, як малого так і великого розміру, замислюються над веденням бізнесу онлайн. А тому актуальність питання в створенні персонального онлайн магазину перед підприємцями встає все частіше і частіше і є актуальним як ніколи.

1.2 Аналіз аналогічних продуктів, їх переваги та недоліки

Аптека низьких цін

Є однією з найпопулярніших в Україні. Має великий каталог товарів. Першейшовши на сайт Аптеки низьких цін, неозброєним оком можна побачити те, що сайт справді має сучасний інтерфейс. В око відразу кидаються навігаційні елементи - такі як навігаційна панель та каталог товарів, що дає можливість відразу почати пошук потрібних товарів.

Каталог товарів містить багато пунктів: Медикаменти, Краса та догляд, Дитячі товари, Щоденна гігієна і так далі. Самі ж пункти поділяються на підпункти. Наприклад, Пункт Медикаменти поділяється на Застуда та Грип, Знеболюючі препарати, Протипухлинні препарати і тому подібне. Що є дуже зручно, якщо у клієнт не знає, які саме препарати йому потрібні. Крім того, на сторінках препаратів міститься детальна інформація про сам препарат, що дуже допомагає.

Є можливість зареєструватись на сайті. Це є корисним як для самого клієнта, оскільки він отримує спеціальні бонуси і може відслідковувати історію своїх покупок, так і для адміністрації сайту, оскільки є можливість робити розсилки щодо акцій, спеціальних пропозицій тощо.

Основним недоліком цієї аптеки є складність та нагромадженість інтерфейсу. Так справді, з одного боку це добре, що клієнту під очі

постійно попадають попередження про акції і показується багато додаткової інформації, але в такому випадку дуже складно стає орієнтуватися особливо людям, які не звикли до покупок онлайн.

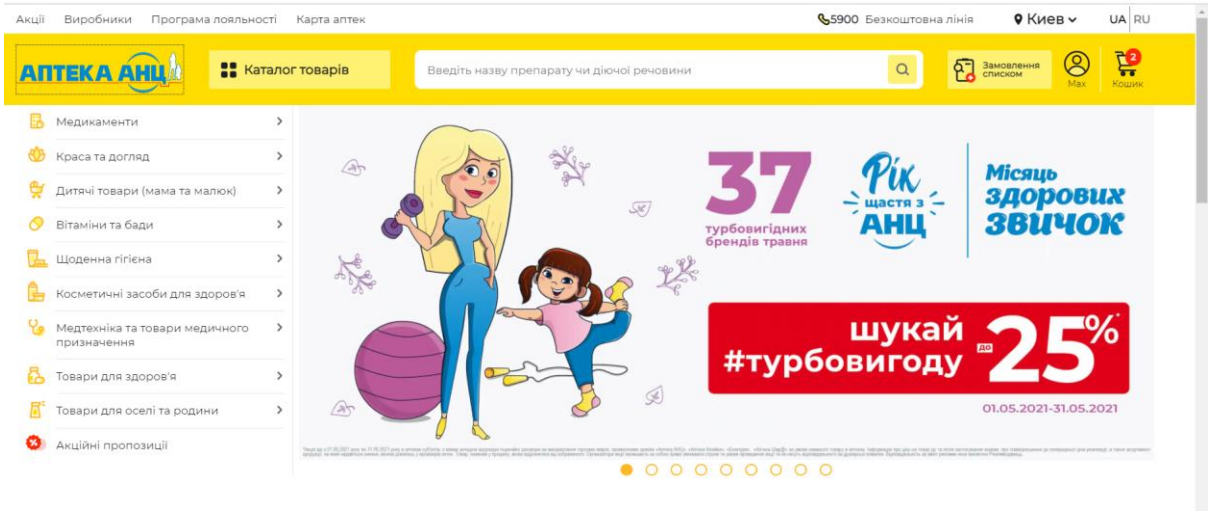

Рисунок 1.1 – сайт аптеки "АНЦ"

До схожих висновків я дійшов проаналізувавши інші сайти, які належать великим мережам аптек, такі як apteka24, viridis, Аптека низьких цін. На рисунку 1.1 зображено будова основної сторінки аптеки АНЦ.

Що цікаво, в моєму аналізі я дійшов висновку, що в той час як великі мережі в основному мають гарно зроблені сайти - майже неможливо знайти сайт якоїсь локальної аптеки, яка не належить певній мережі. Дуже часто такі аптеки, якщо і мають веб-сторінку, наприклад Фармація в Києві, то на ній не має можливості купувати і замовляти товари. На рисунку 1.2 ми можемо побачити будову основної сторінки аптеки "Фармація" та побачити, що вони не мають опції замовити товар.

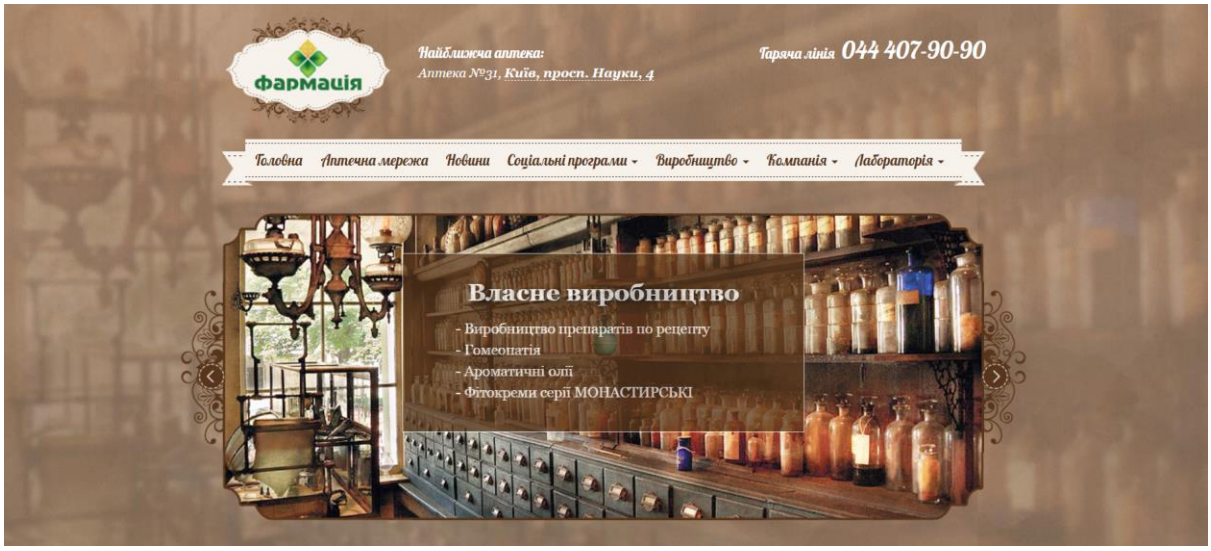

Рисунок 1.2 – основна сторінка аптеки "Фармація"

1.3 Постановка завдання, опис вимог

Отже, проаналізувавши наявні інтернет аптеки можна прийти до висновку, що в моєму проекті мені слід сфокусуватися на створенні веб-додатку не для великих мереж аптек, а для невеликої локальної аптеки, з простим та зручним інтерфейсом.

Ролі користувачів: гість, авторизований користувач, адміністратор.

Функціонал гостя:

- Перегляд Головної сторінки сайту
- Перегляд Сторінки товару
- Додавання товарів у кошик

Функціонал авторизованого користувача:

- Перегляд Головної сторінки сайту
- Перегляд Сторінки товару
- Додавання товарів у кошик
- Можливість оформлення замовлення
- Перегляд історії замовлень

Функціонал адміністратора:

- Перегляд всіх товарів

- Можливість додавання нових товарів
- Можливість редагування товарів
- Перегляд всіх замовлень
- Можливість позначити замовлення як доставлене

Функціонал сторінок

Головна сторінка:

- Список всіх товарів з інформацією про них
- МОжливість переходу на сторінку представлених товарів

Сторінка товару:

- Більш детальна інформація про товар
- Можливість вибору кількості товару для покупки
- Додавання товару в кошик

Сторінка Кошик:

- Список товарів у кошику
- Відображення ціни, кількості, та суми товарів
- Можливість оформити замовлення, яку доступне лише авторизованим користувачам
- Перехід до сторінки вибору Адреси доставки

Сторінка Адреси доставки:

- Можливість заповнення форми
- Перехід до сторінки вибору оплати

Сторінка Вибору оплати:

- Можливість вибору оплати
- Перехід до сторінки Розміщення замовлення

Сторінка Розміщення замовлення:

- Можливість розмістити замовлення
- Перегляд змісту попередніх сторінок

Сторінка Замовлення:

- Можливість оплати
- Інформація про доставку
- Інформація про оплату

Головна панель навігації:

- Доступ до Кошика доступна усім
- Доступ до Сторінки з Входом в аккаунт Гостю
- Доступ до Профілю з Історією замовлень Авторизованим користувачам
- Доступ до Профілю для Виходу з Авторизації Авторизованим користувачам
- Адміністратор має спеціальний доступ до панелі Адміністратора,у якому є доступ до Сторінки всіх Товарів
- Адміністратор має спеціальний доступ до панелі Адміністратора,у якому є доступ до Сторінки всіх Замовлень

Сторінка Авторизації:

- Авторизація через емейл та пароль
- Можливість Зареєструвати Аккаунт

Сторінка з Історією замовлень:

- Доступна Авторизованому користувачу
- Відображає інформацію по замовленню

Сторінка Всіх товарів:

- Доступна Адміністратору
- Відображає всі товари
- Відображає інформацію про них
- Можливість створити новий товар
- Можливість Редагувати
- Можливість Видалення товару

Сторінка створення товару/його редагування:

- Заповнення інформації про товар

- Апдейт інформації про товар

Сторінка Всіх замовлень:

- Доступна Адміністратору
- Перегляд Інформації про Замовлення
- Можливість визначити Замовлення як доставленого

1.4 Висновки до розділу 1

В даному розділі були розглянуті актуальність створення інтернет аптеки та проведено аналіз існуючих інтернет аптек. На основі проведеного аналізу було зроблено висновки щодо доцільності створення, виявлено позитивні та негативні сторони в існуючих аптеках. Було визначено майбутній функціонал аптеки.

Розділ 2. Використані технології

Для розробки свого веб-додатку я планую використати стек технологій з абревіатурою

MERN(**React.js + Node.js + Express + MongoDB**).

Стек MERN є об’єднанням цих чотирьох технологій. Ця комбінація дозволяє розробникам створювати як фронтенд, так і бекенд для веб-застосунків. У стеці MERN ми використовуємо JavaScript на стороні клієнта та Node.js на стороні сервера. На рисунку 2.1 можна побачити структуру стеку MERN.

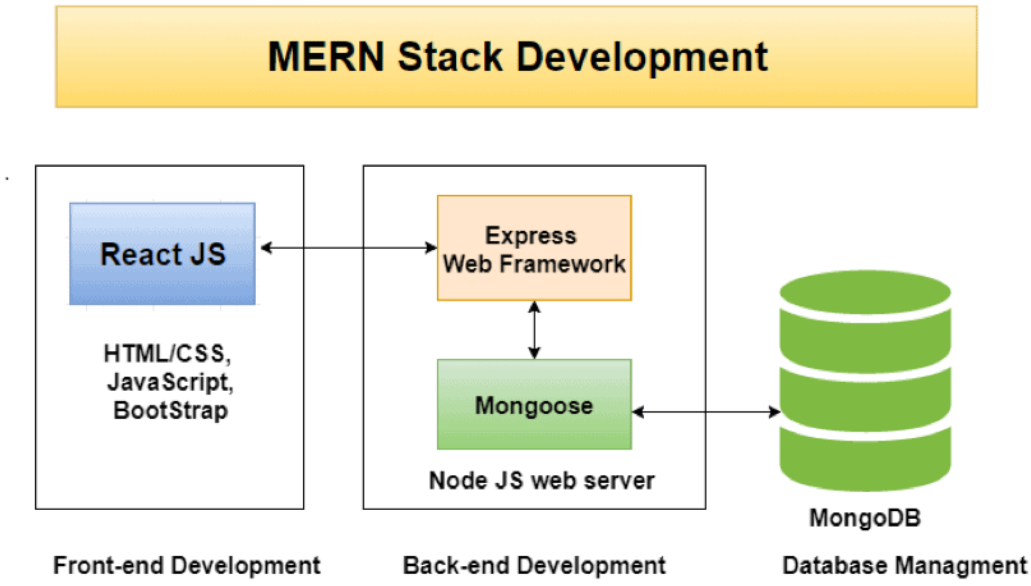

Рисунок 2.1 – структура стеку MERN[2]

Технології, які використовуються в MERN стеці:

2.1 MongoDB

- документо-орієнтована СКБД.[3]

Дозволяє керувати базами даних. Використовує нереляційний підхід. На відміну від реляційного підходу, дані в MongoDB зберігаються в документах, які в свою чергу об'єднуються в колекції.

Такі документи на відміну від рядків в реляційних БД можуть зберігати складну по структурі інформацію.
Кожен документ представляє собою JSON-подібну структуру під назвою BSON. Що дозволяє не описувати схеми баз даних як в реляційному підході, оскільки є можливість змінювати базу даних з часом.
MongoDB написана на C++, що дозволяє її портувати на різні платформи такі як Windows, MacOS, Linux.
На відміну від реляційних баз даних, які зберігають однотипні структуровані об'єкти, MongoDB має колекції, які дають можливість зберігати різні за структурою та властивостями об'кти.
Є багато різних модулів для роботи з mongodb на javascript, але в моєму випадку використовується найбільш популярний - Mongoose.

2.2Node.js

Дуже швидка мова, оскільки є асинхронною. Тісно пов'язана з Javascript. Дозволяє виконувати Javascript код на сервері.

2.3Express - гнучкий веб-фреймворк для додатків на Node.js.

Полегшує маршрутизацію, використання middleware, обробку запитів. Використовується для створення REST API. REST API відповідає веб-сайту, який буде отримувати дані за допомогою HTTP-запитів.

2.4React.js

React.js - JavaScript бібліотека для побудови користувацьких інтерфейсів. Відповідає за фронтендову частину веб-застосунку. Відомий своєю швидкістю завантаження сторінок веб-сайтів, анімацій та інших різних переходів. Через це є, мабуть, найкращим вибором для розробки веб-додатків.[4]
React був створений Facebook у 2013 році. Зараз його використовують багато провідних компаній світу: Netflix, Instagram Yahoo і так далі.

Імплементує підхід Single Page Application, який дозволяє не перезавантажувати сторінку при використанні, що результується у швидкості й економії часу. Замість багатьох сторінок, які потребують перезавантаження - є одна сторінка, яка під завантажує інші компоненти. Більшість ресурсів завантажується лише один раз за час існування веб-додатку, натомість циркулюють саме дані.[5]

Таке стало можливим завдяки підходу AJAX (Asynchronous Javascript And Xml), який заключається в фоновому відправленні запитів на сервер, і отриманні відповіді, яка змінює стан додатку та його зовнішній вигляд. При зміні сторінки додатку змінюється її DOM(Document Object Model) дерево, що є дуже ресурсовитратною операцією, оскільки спочатку DOM дерево було статичним і не передбачало динамічних операцій.

Тому React використовує віртуальний DOM, який є більш легкою копією справжнього DOM. React DOM порівнює стани віртуального DOM дерева і знаходить мінімальну кількість маніпуляцій, які потрібно провести з справжнім DOM деревом. React DOM бере на себе оновлення DOM для його відповідності React-елементам. Такий підхід взаємодії з веб-додатком є більш швидким, ніж якщо працювати з Dom безпосередньо.

Веб-додатки на React побудовані з компонентів. Такі компоненти є незалежними і їх можна переносити із проекту в проект.

2.5 Висновок до розділу 2

В даному розділі були розглянуті використані технології, було проаналізовано їх плюси та мінуси та основні властивості, які роблять саме їх зручним у використанні.

Розділ 3. Проектування і розробка

3.1 Створити React-app

Створити React-app за допомогою команди:

npx create-react-app frontend

У нас створюється папка frontend з App.js

Таким чином у нас створюється простий додаток, з яким ми вже можемо працювати.

Щоб запустити наш React додаток ми використовуємо команду npm start, яка на локальному хості запускає наш додаток, який відображає App.js.

Наступне на черзі це побудувати спеціальні Screens(екрани), функціональні компоненти, які відповідають за відображення сторінок нашого веб-додатку, але перед цим маємо завантажити react-router-dom.

У моєму випадку є лише один кореневий елемент DOM. Для рендерингу елементів використовується функція ReactDOM.render() з React-елементом та кореневим DOM вузлом у якості аргументів.

```
ReactDOM.render(
  <Provider store ={store}>
    <App />
    </Provider>,document.getElementById('root')
);
```

Рисунок 3.1

Він дозволяє створювати Routes(шляхи), які поєднують між собою наші Екрани. Кожному Екрану відповідає певний Шлях. Але щоб ці шляхи працювали, ми маємо огорнути головну частину в App.js в об'єкт <BrowserRouter>, оскільки саме він визначає набір допустимих маршрутів і, коли до додатку приходить запит, саме він співставляє URL запиту і шляхів.

```
<Route path="/product/:id" component={ProductScreen} exact></Route>
```

Рисунок 3.2 – Атрибути шляху

У самого шляху є 2 атрибути. Перший - це path, шаблон адреси, з яким співставляється URL запиту, та component - компонент, який відповідає за опрацювання запиту(в даному випадку я такі компоненти називаю Екранами). На рисунку 3.2 якраз зображено дані 2 атрибути.

Також для навігації по сторінкам замість посилання простим тегом a, використовується компонент Link. Він дозволяє оновити контент сторінки без її перезавантаження.

Тепер, коли наші Routes готові, можна почати створювати Screens, які будуть відображати певний вміст. Наприклад, на рисунку 3.3 знизу можна побачити список Товарів для відображення на Головній Сторінці.

```
return loading ? <div>Loading...</div> :
  error ? <div>{error}</div> : (
    <ul className="products">
      {
        products.map((product) =>(
          <li key={product._id}>
            <div className="product">
              <Link to={'/product/' + product._id}>{product.name}
                <img className="product-image" src={product.image} alt="product"></img>
              </Link>

              <div className="product-brand">Виробник {product.brand}</div>
              <div className="product-price">Ціна ₴{product.price}</div>
              <div className="product-rating">Відгуки: {product.numReviews}</div>
            </div>

          </li>
        ))
      }
    </ul>
  )
```

Рисунок 3.3 – список товарів для відображення на головній сторінці

3.2 Створити сервер на NodeJS

Щоб відобразити вміст, ми перш за все маємо створити Node.js додаток на бекенді, який дасть нам можливість створити API, які повернуть потрібні нам дані.

Для цього перш за все ми маємо ввести команду: npm init, яка створює в нашій основній папці package.json

Наступний крок, створити server.js на бекенді, який є вхідною точкою до нашого додатку. Для цього ми використовуємо Express, який є фреймворком для Node.js. Він дозволяє нам створити простий сервер. Express, по суті, являє собою серію викликів функцій тимчасової обробки.Функції тимчасової обробки або middleware - це функції, які мають доступ до об'єкта запиту (req), об'єкту відповіді (res) і до наступної функції тимчасової роботи в циклі "запит-відповідь" додатка.

```
const app = express();
app.get('/', (req, res) => {
  res.send('Server is ready');
});
app.listen(5000, () => {console.log("Server started at http://localhost:5000")});
```

Рисунок 3.4 – використання фрейморку Express

app.get(‘/’, (req, res).. - наш перший route на бекенді. У нього є handler, який приймає запит та повертає відповідь. В даному випадку відповідь - це повідомлення, що сервер готовий до запуску.

Щоб запустити сервер маємо викликати listen метод, який приймає порт, і в даному випадку виводить повідомлення в консоль.

Для отримання даних ми повинні створити routes, у яких перший аргумент - адреса, функція, яка б повертала дані.

```
app.use("/api/products", productRouter);
```

Рисунок 3.5 – використання Routes

3.3 Redux

Redux - потрібен нам для того, щоб управляти станами нашого веб-додатку.

Встановлення Redux: npm install redux react-redux.

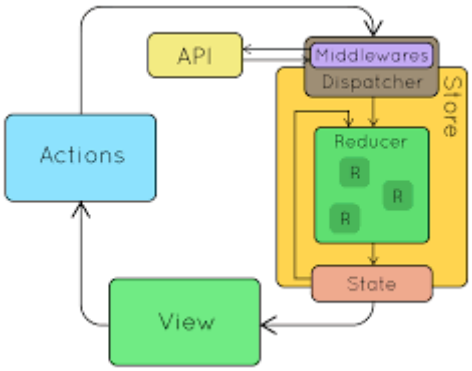

Рисунок 3.6 – функціонування Redux

Але як само він працює в нашому додатку. На рисунку 3.6 View - це наші Screens. У них ми відправляємо (dispatch) спеціальні Дії (Actions). Ці самі Actions в свою чергу відсилають свої запити до redux store, щоб змінити стан нашого додатку. Отже, щоб змінити статус нам потрібно створити action, а потім відправити його до store. У store є 2 головних елементи: 1)Reducers та 2)Стани. Reducer - це така собі функція, яка отримує на вхід поточний стан нашого додатку, робить певні зміни та повертає новий стан. Маючи історію станів, ми можемо краще їх відслідковувати та не опинитися в ситуації з непередбачуваним станом.

Отже, перше, що ми маємо створити - це redux store. Для цього потрібно 2 речі. В InitialState, де ми задаємо початковий стан та reducer. Сам store ми створюємо за допомогою функції createStore, яка імпортується з redux. Цій функції достатньо 2 параметрів, але це не ліміт. Особливо тут, ми

використовуємо Redux thunk, оскільки він дає можливість посилати AJAX запити в наших Actions. AJAX запити - це запити, які виконуються в фоновому режимі і є асинхронними.
Для поєднання всіх reducers використовуєтья combineReducers, яка приймає на вхід об’єкт, який містить всі визначені reducers в додатку.

```
const store = createStore(
   reducer, initialState, composeEnhancer(applyMiddleware(thunk)));
```

Рисунок 3.7 – Redux Store

Для того, щоб Redux store запрацював для всіх компонентів додатку, ми маємо огорнути його компонентом Provider при чому на найвищому рівні, оскільки всі компоненти можуть взаємодіяти з store. Це можна побачити на рисунку 3.8.

```
ReactDOM.render(
  <Provider store ={store}>
   <App />
   </Provider>,document.getElementById('root')
);
```

Рисунок 3.8 – компонент Provider

Наступний крок - створення Actions, які ми використовуємо для отримання data з бекенду.та для зміни стану

```
const listProducts = () => async (dispatch) => {
    try {
        dispatch({ type: PRODUCT_LIST_REQUEST });
        const { data } = await axios.get("/api/products");
        dispatch({ type: PRODUCT_LIST_SUCCESS, payload: data });
    }
    catch (error) {
        dispatch({ type: PRODUCT_LIST_FAIL, payload: error.message });
    }
}
```

Рисунок 3.9 – функція listProducts

listProducts - це функція, яка приймає dispatch як параметр. При dispatch ми змінюємо стан Redux, відсилаємо Action з даними(payload: data) та певний тип в залежності від успішності цього Action. Дане виконання можна побачити на рисунку 3.9.

стан,

Тепер у відповідь на Action ми маємо створити reducer; який приймає стан та action. і на основі типу Action повертає новий стан. При FAIL стан не змінюється, при REQUEST loading: true, оскільки ми очікуємо на відповідь.При SUCCESS ми отримуємо дані.

```
function productListReducer(state = { products :[]}, action){
    switch (action.type) {
        case PRODUCT_LIST_REQUEST:
            return {loading: true};
        case PRODUCT_LIST_SUCCESS:
            return {loading: false, products: action.payload};
        case PRODUCT_LIST_FAIL:
            return {loading: false, error: action.payload}
        default:
            return state;
    }
}
```

Рисунок 3.10 - Reducer

3.4 Зв'язок з mongoDB

Для зв'язку з mongoDB використовується mongoose. Завдяки цьому нам не доводиться працювати з mongoDB напряму, що пришвидшує і полегшує процес розробки.

Mongoose - один із найпопулярніших модулів для роботи з mongodb на javascript. Mongoose дозволяє дуже зручно працювати з базою даних. В ньому все будується на схемах(Schema) даних, з яких в свою чергу створюються моделі. Тобто створюється модель даних, що зберігаються в базі, а mongoose вже допомагає їх типізувати, валідувати, будувати бізнес логіку поверх них, створювати запити і т.д.

Отже, для того, щоб створити модель, нам в першу чергу потрібно створити Схему. На рисунок 3.11 відображено створення Схеми.

```
const productSchema = new mongoose.Schema({
    name: {type:String, required: true, unique:true},
    catgory: {type:String, required: true},
    image:{type:String, required: true},
    price:{type:Number, required: true},
    brand: {type:String, required: true},
    numReviews: {type:Number, required: true},
    countInStock: {type:Number, required: true},
},{
    timestamps: true,
});

const Product = mongoose.model('Product', productSchema);

export default Product;
```

Рисунок 3.11 – використання Mongoose

Для створення моделі використовується mongoose.Schema, яка приймає як аргумент об'єкт, який містить у собі всі поля в даному випадку Product. Треба зазначити, що нам не потрібно задавати поле id, оскільки mongoDB створить його автоматично. Встановлення required: true - означає, що це поле є обов'язковим для заповнення, встановлення unique: true означає,

що повтори заборонені. timestamps дозволяє нам у базі даних бачити дату та час, коли певний об’єкт був створений і оновлений.
Для створення самої моделі використовується mongoose.model(Name, Schema), яка приймає 2 аргументи Назва та Схема.

Для підключення до бази даних використовується mongoose.connect(‘mongodb://localhost/drug-store’, {Options})

3.5 Routers

Екземпляр Router являє собою комплексну систему проміжних оброблювачів і маршрутизації;
Функція express.Router () використовується для створення нового об'єкта маршрутизатора. Ця функція використовується, коли нам потрібно створити новий об'єкт маршрутизатора замість того, щоб створювати нові маршрути в server.js.

```
const productRouter = express.Router();

productRouter.get('/:id', expressAsyncHandler(async (req,res) => {
  const product = await Product.findById(req.params.id);
  if(product){
      res.send(product);
  }
  else{
      res.status(404).send({message: "Product not found"});
  }
}));
```

```
app.use("/api/products", productRouter);
```

Рисунок 3.12 – функція Express.Router()

В даному випадку productRouter - маршрутизатор. Спрацьовує, коли додаток отримує запит типу GET за адресою "api/products/:id". В результаті обробки повертається товар з переданим id. expressAsyncHandler запобігає безперервному завантаженню сторінки в разі помилки. Коли стається помилка, то ця помилка передається до іншої функції (middleware), яка передає помилку на фронтенд.

```
app.use((err, req, res, next) => {
    res.status(500).send({ message: err.message });
  });
```

Рисунок 3.13 -

3.6 Авторизація

Для авторизації/реєстрації аналогічно до Товарів створюється Модель User та userRouter, у якому створюються api.

```
userRouter.post('/signin',expressAsyncHandler(async (req, res) =>{
    const user = await User.findOne({email: req.body.email});
    if(user){
        if (bcrypt.compareSync(req.body.password, user.password)) {
            res.send({
              _id: user._id,
              name: user.name,
              email: user.email,
              isAdmin: user.isAdmin,
              token: getToken(user),
            });
            return;
      }
    }
    res.status(401).send({ message: 'Invalid email or password' });

}));
```

Рисунок 3.14 – модель userRouter

Для api /signin використовується запит типу POST, оскільки повертається token, який ідентифікує User для подальших запитів. Робиться AJAX запит для перевірки існування заданого емейлу, адже саме за ним та паролем відбувається авторизація. Далі йде перевірка пароля, який зіставляється з хешованим паролем з бази даних за допомогою bcrypt.compareSync(). bcrypt - це криптографічна функція створення ключа, що використовується для безпечного зберігання паролів і використовує алгоритм Eksblowfish.

Якщо true, повертається дані користувача та token, який буде згенерований з jsonwebtoken. Для створення токену використовується функція jwt.sign({a, b}) з бібліотеки jsonwebtoken, яка приймає 2 параметри - дані користувача, та secret, який є ключем.

Наступний крок, створити signin action та reducer. signin action буде відповідати за отримання та відправлення(dispatch) даних з /api/users/signin, зберігання їх у локальному сховищі(localStorage), що дозволить User бути зареєстрованим в системі навіть після закриття сторінки.

В userSigninReducer ми як параметр отримуємо action. І в залежності від типу параметра action ми відправляємо дані. case USER_SIGNOUT повертає пустий обє’кт, що значить, що дані клієнта були стерті.

```
function userSigninReducer(state = {}, action) {
  switch (action.type) {
    case USER_SIGNIN_REQUEST:
      return { loading: true };
    case USER_SIGNIN_SUCCESS:
      return { loading: false, userInfo: action.payload };
    case USER_SIGNIN_FAIL:
      return { loading: false, error: action.payload };
    case USER_SIGNOUT:
      return {};
    default:
      return state;
  }
}
```

Рисунок 3.15 – userSigninReducer

І нарешті в самому SigninScreen ми маємо можливість відправити(dispatch) signin action при натисканні на кнопку.

3.7 Демонстрація результатів

Рисунок 3.16 – Header, в залежності від стану User

Header присутній на всіх сторінках. У header використовується conditional rendering і його елементи залежать від авторизації користувача та чи є користувач Адміністратором. Щоб отримати цю інформацію використовується хук useSelector, який отримує її від state.

```
const userSignin = useSelector((state) => state.userSignin);
  const { userInfo } = userSignin;
```

Щоб доступ до панелі Адміністратора був постійний і недоступний іншим користувачам використовується спеціальний AdminRoute.

```
function AdminRoute({ component: Component, ...rest }) {
  const userSignin = useSelector((state) => state.userSignin);
  const { userInfo } = userSignin;
  return (
    <Route
      {...rest}
      render={(props) =>
        userInfo && userInfo.isAdmin ? (
          <Component {...props}></Component>
        ) : (
          <Redirect to="/signin" />
        )
      }
    ></Route>
  );
}
```

Рисунок 3.17 – AdminRoute

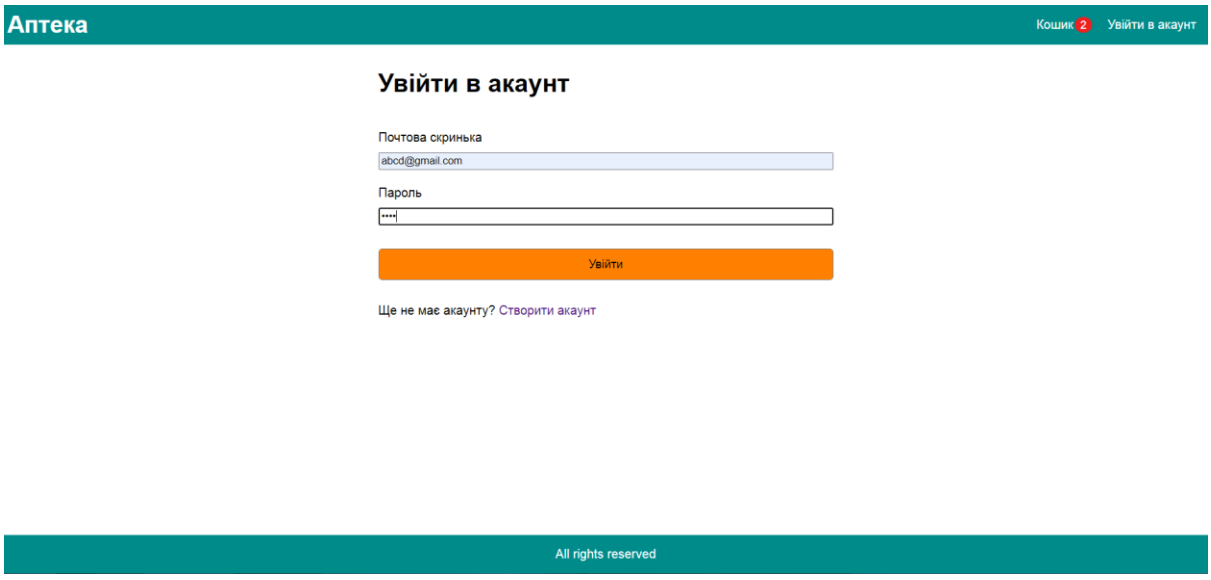

Рисунок 3.18 – Сторінка авторизації

Використовується form при onSubmit якої виконується submitHandler, який завдяки preventDefault не дозволяє формі перезапуститися, відправляє signin action. В input onChange використовуються setEmail, setPassword з

React хуком useState, який дозвляє нам зберігати дані між викликами функцій, оскільки зазвичай ми їх втрачаємо, якщо не користуємося класами. Єдиним аргументом useState є його початковий стан, а повертає він стан та функцію, яка цей стан оновлює. Далі за допомогою хуку useEffect відбувається redirect користувача на попередній Screen.

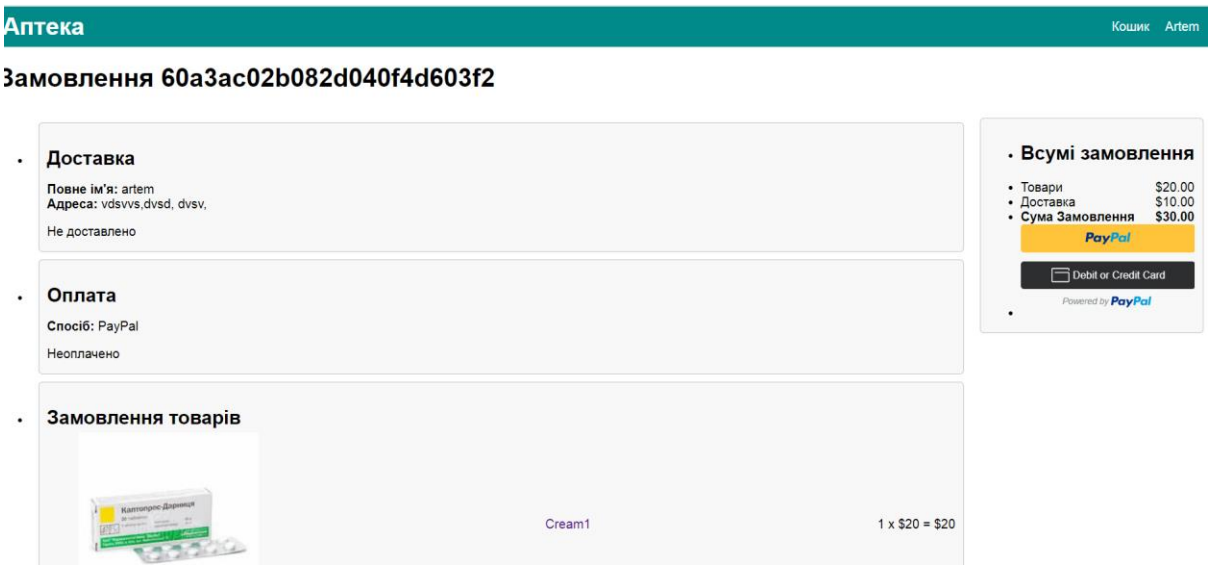

Рисунок 3.19 – Сторінка оплати

На цій сторінці відбувається оплата за допомогою PayPal. Для реалізації оплати перш за все створюється спеціальний api/config/paypal, який надає id користувача.

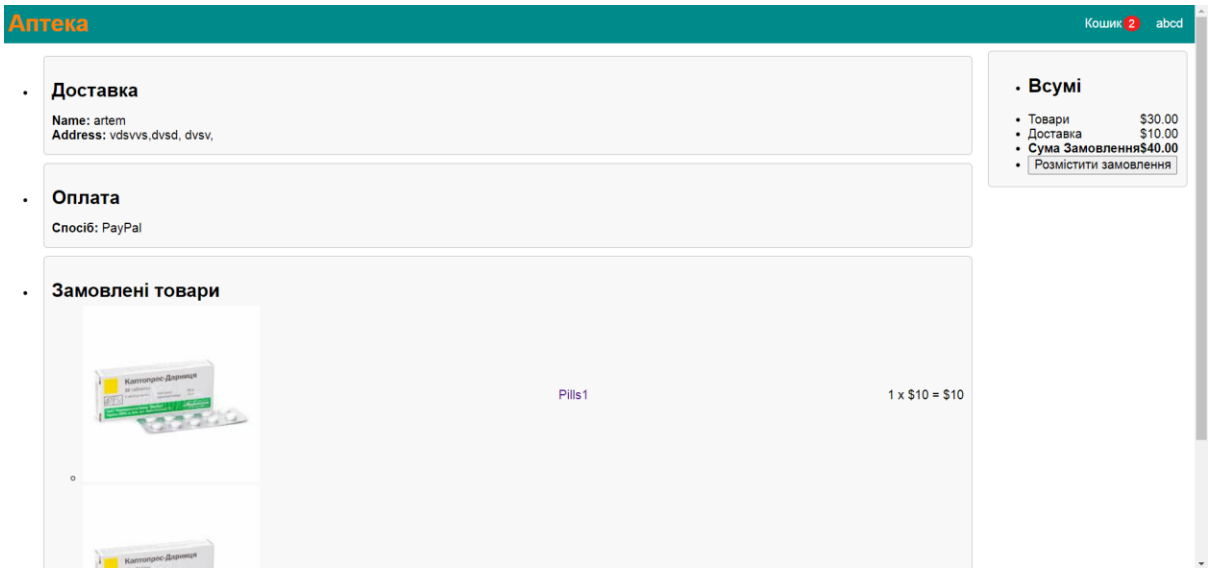

Рисунок 3.20 – Сторінка оформлення товару

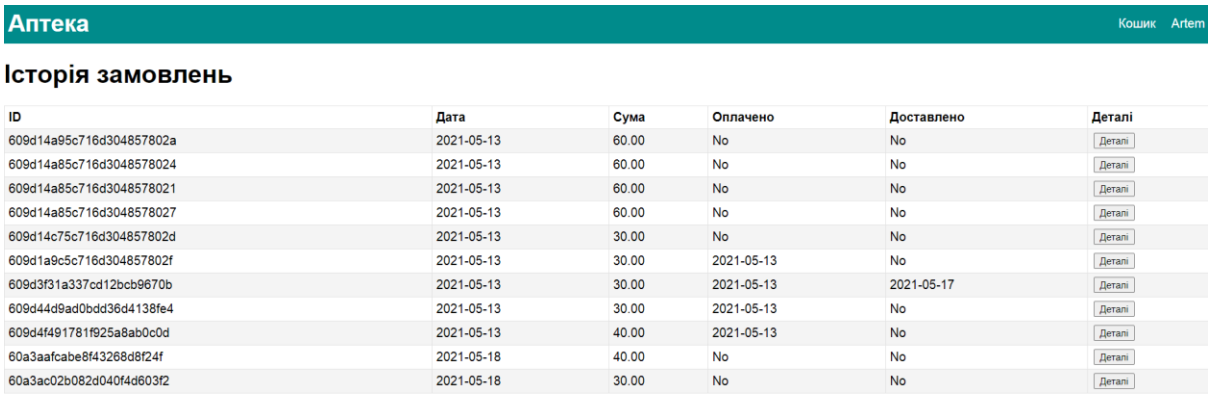

Рисунок 3.21 – Сторінка з історіями замовлень

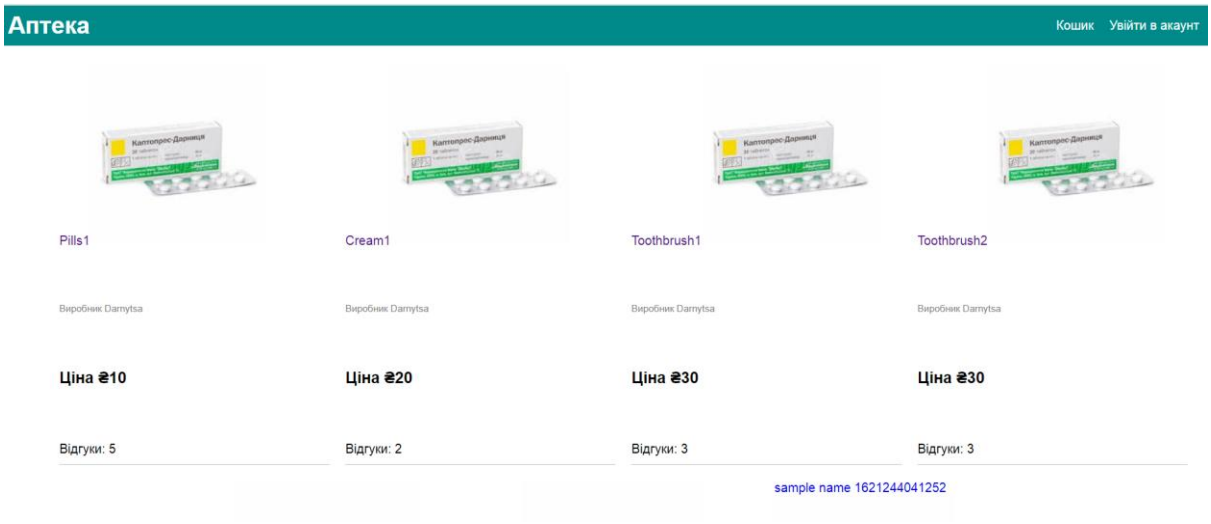

Рисунок 3.22 – Головна сторінка

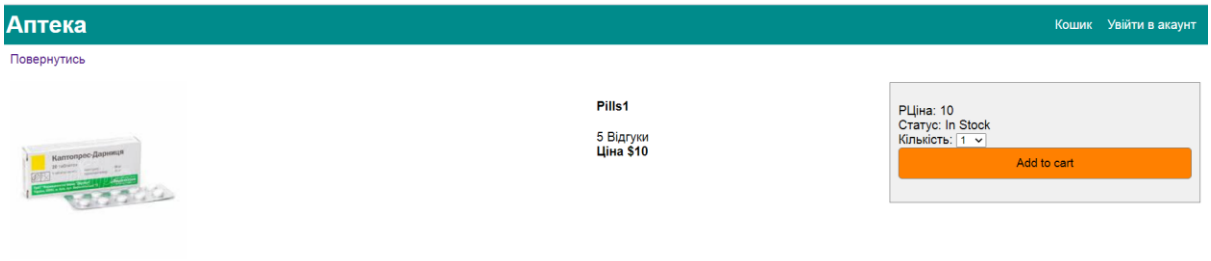

Рисунок 3.23 – Сторінка товару

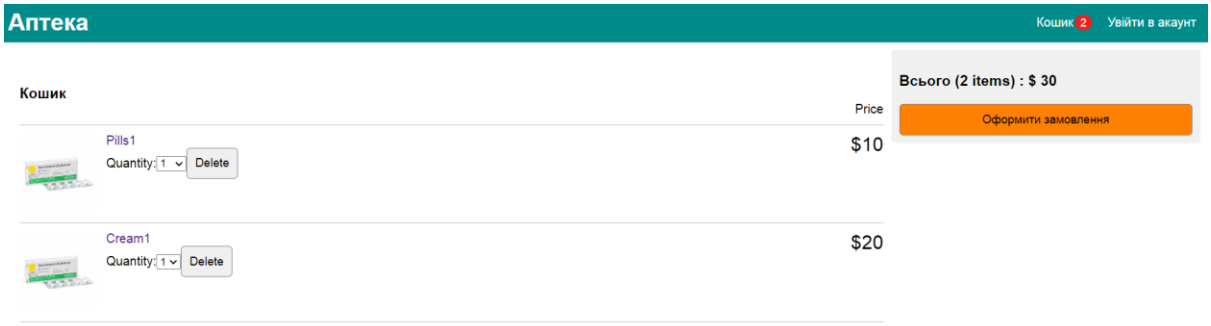

Рисунок 3.24 – Сторінка кошику

3.8 Висновки до розділу 3

В даному розділі особливу увагу було приділено розробці веб-додатку, а саме способу розробки основних елементів, з яких будується веб-додаток. Було детально розглянуто Redux, підключення до MongoDB, створення серверу на Node.js та багато інших компонентів, з яких складається веб-додаток.

Висновки

Отже, по завершенню роботи мною було створено простий веб-додаток для інтернет аптеки, який з малою вірогідністю може конкурувати з великими мережами аптек, але тим не менш по функціоналу може підійти невеликим локальним аптекам. Поставлені вимоги до функціоналу застосунку хоч і були виконані, але залишають за собою великий простір для розвитку і покращення.

За час роботи над проектом, над подібним якому я до цього не працював, я набув багато корисних навичок, став більше розуміти як саме працюють веб-додатки. Дізнався чому додатки на React такі швидкі, і яку роль в цьому грає ReactDOM, що таке Routes, як створювати сервер на Node.js. Що таке Redux і як він керує станами нашого додатку. Ознайомився з mongoDB і способом зберігання у ньому даних, з модулем mongoose і як він працює з базами даних в mongoDB. Як саме влаштована маршрутизація, і яку роль в ній грає Express. Зміг в повний мірі попрацювати як над бекендом, так і над фронтендом, що дало розуміння, як вони поєднані між собою.

Можливі покращення:

- Розробити систему пошуку товарів
- Розробити систему сортування товарів
- Створити Каталог товарів
- Інтегрувати інші сервіси оплати(наприклад, LiqPay)
- Інтегрувати доставку Новою Поштою/Укр. Почтою

Список використаної літератури

1. Нагорский В. Рынок e-commerce в Украине в 2020 году Джерело: https://rau.ua/ru/news/e-commerce-v-ukrayini-2020/ [Електронний ресурс] / Віктор Нагорский. – 2020. – Режим доступу до ресурсу: https://rau.ua/ru/news/e-commerce-v-ukrayini-2020/.
2. How does MERN stack works [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.bocasay.com/how-does-the-mern-stack-work/.
3. Введение в MongoDB [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://metanit.com/nosql/mongodb/1.1.php.
4. Разработка SPA на React, NodeJS, Express и MongoDB [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: http://krambertech.github.io/spa-webinar/.
5. Маршрутизация Определение Маршрутов [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://metanit.com/web/react/4.1.php.
6. Single-page application vs. multiple-page application [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58.