МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____
(підпис)

червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: «Інформаційна веб-система торгівельної платформи»

здобувача групи ІН - 93 Мартинов Владислав Ігорович

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Мартинов Владислав

_____
(підпис)

Керівник, доцент, кандидат технічних наук Петров Сергій _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
Ігор
ШЕЛЕХОВ
_____
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-93 Мартинов Владислав Ігорович

1. Тема роботи: «Інформаційна веб-система торгівельної платформи»
затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз та визначення об'єкту дослідження. 2) Огляд технологій, що використовуються для розробки веб-систем. 3) Розробка серверної частини веб-додатку, побудова API. 4) Розробка клієнтської частини.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис)

Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз та визначення об'єкту дослідження	15.05.2023 - 17.05.2023	

2	Огляд технологій що використовуються для створення веб-додатків	17.05.2023 – 18.05.2023	
3	Розробка серверної частини веб-додатку, побудова API	19.05.2023 – 26.05.2023	
4	Розробка клієнтської частини	27.05.2023 – 03.06.2023	
5	Оформлення пояснювальної записки до кваліфікаційної роботи	04.06.2023 – 05.06.2023	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

РЕФЕРАТ

Записка: 51 стр., 18 рис., 2 додаток, 15 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною так як присвячена використанню сучасних веб-технологій при розробці веб-системи для використання в галузі електронної комерції.

Об’єкт дослідження — Інформаційна веб-система торгівельної платформи

Мета роботи — застосування сучасних веб-технологій та використання їх в е-комерційних системах

Методи дослідження — ознайомлення з сучасними інформаційними системами, та вивчення сучасних технологій, що застосовуються в торгівельних веб-системах та існуючих методах просування в глобальній мережі.

Результати — Розроблено інформаційну веб-систему для торгівельної платформи з використанням сучасних веб-технологій, зі змогою використання як на мобільних так і на настільних пристроях з доступом до інтернету та легкою можливістю модифікації старого, та додавання нового функціоналу

ЗМІСТ

ВСТУП

Безперервний та стрімкий розвиток технологій глобальної мережі інтернет відбувається з самого першого дня існування. Збільшення обчислювальних можливостей, швидкості передачі даних та інші фактори, такі як охоплення ринку мобільних пристроїв, тощо зробили глобальну мережу інтернет повсякденністю нашого життя.

З розвитком глобальної мережі, потужної швидкості розвитку набула і торгівля, та просування своїх послуг у цифровому просторі. Завдяки глобалізації, яка набирала оберти паралельно з глобальною мережею, набуло полярності розповсюдження своїх товарів та певних послуг за межами країни дислокації компанії чи то підприємця, який надає певні послуги.

На фоні описаних факів в попередньому абзаці, починаючи з 1990х років, інтернет комерція або електронна комерція почала набирати стрімку швидкість розвитку. На своєму шляху розвитку E-commerce завжди застосовувала нові технології та адаптувалась до реалій сучасного ринку.

Однією з перших технологій які були застосовані в даній галузі – алгоритми рекомендацій, відомий також як алгоритми просування або АП.

АП – математичні методи які застосовуються для пропозиції та рекомендації товарів, послуг, контенту або інших об’єктів користувачам на основі їх інтересів. Головна мета цих алгоритмів саме в тому, щоб надати користувачам персоналізовані рекомендації, які забезпечать покращення взаємодії та задоволення користувачів, але однією з ключових функцій АР є збільшення продажів та залучення нових клієнтів.

Загалом E-commerce має досить обширний перелік переваг, які допомогли їй стати популярним способом для продажів та надання послуг, а саме:

1. Підвищена зручність для клієнтів: електронна комерція дозволяє клієнтам робити покупки в будь-який час і з будь-якого місця, якщо

у них є підключення до Інтернету. Це особливо вигідно для клієнтів, які живуть у віддалених районах або мають напружений графік.

2. Більше охоплення для компаній: електронна комерція дозволяє компаніям продавати свої продукти та послуги клієнтам у всьому світі, а не обмежуватися певним географічним розташуванням.
3. Більше охоплення для компаній: електронна комерція дозволяє компаніям продавати свої продукти та послуги клієнтам у всьому світі, а не обмежуватися певним географічним розташуванням.
4. Підвищена ефективність: електронна комерція може допомогти підприємствам автоматизувати та оптимізувати різноманітні процеси, такі як керування запасами, обробка замовлень і обслуговування клієнтів, що може заощадити час і ресурси.
5. Можливість відстежувати та аналізувати дані про клієнтів: платформи електронної комерції часто надають підприємствам детальну інформацію про своїх клієнтів, включаючи історію покупок, демографічні дані та вподобання. Ці дані можуть допомогти компаніям адаптувати свої маркетингові зусилля та покращити свої продукти та послуги.

Завдяки глобалізації поштового сполучення, та поширенню глобальної мережі на мобільні пристрої та інші типи персональних комп’ютерів, e-commerce набув популярності та став май же не обов’язковим засобом для застосування при створенні нового бізнесу для його стрімкого розвитку.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Основні поняття, терміни та відомості про веб-системи

Для розміщення будь якого веб-додатку в інтернеті потрібно розуміти те, що будь який веб додаток це лише набір файлів з розміткой HTML, таблицями стилів CSS, та скриптів JavaScript. Для постійного доступу будь яких користувачів до веб додатку в будь який час потрібно доменне ім’я, яке буде слугувати унікальним ідентифікатором сторінки, а також хостинг більш зручного розміщення файлів в глобальній мережі.

Складовою будь якого веб додатку є дві невід’ємні частини, це клієнтська Front-end, та серверна Back-end. Для взаємодії між користувачем та клієнтською частиною потрібен web-браузер. Для взаємодії клієнта та сервера та обміну даними використовуються веб запити з клієнтської частини, та відповіді за статус кодами з серверної.

Web-браузер – програма для зчитування, та компонування разом веб-сторінок які складаються з розмітки HTML, каскадних таблиць стилів CSS, скриптів мовою програмування або МП JavaScript, для останнього існують фреймворки для створення як клієнтської так і серверної частини.

Web-сервер – підключений до глобальної мережі інтернет комп’ютер з підтримкою HTTP протоколу для доставки даних ГМ, та дозволяє обробляти зображення, текст, відео, звук та інші дані.

HTML - HyperText Markup Language, стандартна мова розмітки створена для відображення веб-сторінок, що включає, текст, зображення, відео, гіперпосилання.

CSS - Cascading Style Sheets, мова стилів для візуального оформлення веб-сторінок на основі розмітки HTML.

JavaScript – мова програмування для створення динамічних та інтерактивних елементів, а також для взаємодії та валідації даних на сторінках.

Framework (Фреймворк) – набір створених завчасно розробниками скриптів, бібліотек та інструментів, які надають структуру, та прискорюють рішення типових задач, та спрощують використання загальних функціональних можливостей.

Для створення веб-додатку використовувались такі програмні засоби та технології

- Веб сервер або серверна частина на базі Node.js за використанням фреймворку Express.js
- -СУБД Postgresql
- Клієнтська частина на базі МП JavaScript з використанням фреймворку React.js та фреймворком задання стилів React Bootstrap
- Для зв’язку між клієнтською частиною та серверної частинами використовується технологія RestAPI

1.2 Web-сервер на основі Node.js

Сервер, що використовує платформу Node.js для створення веб-додатків та обробки HTTP запитів, що дозволяє використовувати МП JavaScript для написання коду.

1. Асинхронна та подієва модель: Node.js використовує асинхронну та подієву модель, що дозволяє обробляти багато запитів одночасно без блокування потоків. Це дозволяє досягти високої продуктивності і ефективно використовувати ресурси сервера.
2. Вбудовані модулі: Node.js надає ряд вбудованих модулів, які спрощують розробку веб-серверів. Наприклад, модуль HTTP дозволяє створювати HTTP-сервери, обробляти запити та відправляти відповіді клієнту. Також є модулі для обробки маршрутів, роботи з файлами, обробки запитів з різних джерел тощо.
3. Екосистема пакетів: Node.js має широку екосистему пакетів, доступних через менеджер пакетів npm. Це дозволяє використовувати готові

модулі та бібліотеки для розробки різних функцій веб-сервера, таких як маршрутизація, аутентифікація, робота з базами даних, шаблонізація тощо.

4. Розширюваність: Node.js дозволяє створювати розширення на мові C/C++ для оптимізації виконання певних операцій або використання існуючих бібліотек.

Даний веб-сервер може бути написаний за допомогою таких фреймворків як: Exprass.js, Koa.js, Hapi.js тощо. Дані фреймворки розширюють функціонал, та пришвидшують процес розробки за допомогою готової реалізації типових рішень.

1.3 Web-клієнт на основі React.js

Це клієнт створений на бібліотеці React.js що дозволяє створювати клієнтську частину динамічною та з використанням інтерактивного інтерфейсу з використанням підходу компонентів.

Основні переваги клієнтської частини створеної на основі React.js:

1. Компонентна архітектура: React пропонує розбити інтерфейс на компоненти, які відображають певну частину UI та мають свою внутрішню логіку. Компоненти можуть бути вкладені один в одного, утворюючи ієрархію компонентів. Це спрощує організацію та управління складними інтерфейсами.
2. Віртуальний DOM: React використовує віртуальний DOM (Document Object Model) для ефективного оновлення інтерфейсу. Замість безпосереднього змінення реального DOM, React створює віртуальне представлення інтерфейсу, здатне швидко визначати та вносити зміни до реального DOM. Це забезпечує оптимальну продуктивність та швидкість рендерингу.
3. Односторінкові додатки: React дозволяє створювати односторінкові додатки (Single-Page Applications, SPA), де весь інтерфейс

завантажується один раз, а зміни відбуваються динамічно без перезавантаження сторінки. Це забезпечує більш зручний та плавний досвід користувача

Саме через зручну структуру та можливість пере-використання даний фреймворк набув чинної популярності серед технологій для створення веб додатків.

1.4 Бібліотека стилів React BOOTSTRAP

Бібліотека стилів що поєднує можливості фреймворку React.js та стилів BOOTSTRAP, одного з найпопулярніших фреймворків для розробки веб-інтерфейсів, що включає в себе великий набір стилів, та компонентів.

Особливості бібліотеки React BOOTSTRAP:

1. Готові компоненти: React Bootstrap надає набір готових компонентів, таких як кнопки, форми, навігація, каруселі, модальні вікна та багато інших. Ці компоненти можуть бути легко використані та налаштовані відповідно до потреб проекту.
2. Адаптивний дизайн: Благодаря використанню Bootstrap, React Bootstrap компоненти автоматично налаштовуються на адаптивний дизайн. Це означає, що компоненти можуть змінювати свій вигляд та розмір в залежності від розміру екрану, що дозволяє добре виглядати на різних пристроях та роздільних здатностях.
3. Легка інтеграція з React: React Bootstrap побудований на базі React.js і працює повністю з React-компонентами. Це означає, що його компоненти можуть бути використані як будь-які інші React-компоненти, і їх можна легко інтегрувати в React-додатки.
4. Конфігуруємість та розширюваність: Компоненти React Bootstrap можуть бути налаштовані та розширені за допомогою властивостей та додаткових CSS-класів.

Виходячи з цього, дана бібліотека може помітно пришвидшити стилізацю та створення інтерфейсу користувача веб-додатку.

1.5 Система управління базами даних PostgreSQL

Система управління базами даних або СУБД, яка надає потужні можливості для збереження, управління та структурування даними, що являється однією з найпопулярніших відкритих SQL-Like СУБД

Основні особливості PostgreSQL

1. Розширена функціональність: PostgreSQL підтримує широкий спектр функціональних можливостей, таких як складні SQL-запити, індексування, транзакції, відношення (таблиці), засоби захисту даних та багато іншого. Вона також має багато вбудованих функцій, таких як математичні операції, рядкові функції, обробка дати та часу, робота з JSON та багато іншого.
2. Розширені можливості масштабування: PostgreSQL може ефективно працювати з великими обсягами даних та високими навантаженнями. Вона підтримує горизонтальне масштабування (розподілені бази даних) та вертикальне масштабування (розширення обладнання).
3. Підтримка стандартів SQL: PostgreSQL дотримується стандартів SQL, зокрема стандартів ANSI SQL та SQL:2008. Це означає, що код, написаний для PostgreSQL, буде переносимим між різними системами баз даних, що підтримують стандарти SQL.
4. Розширюваність та плагіни: PostgreSQL дозволяє розширювати свої можливості за допомогою плагінів і розширень. Вона має велику екосистему сторонніх розширень, які надають додаткові функціональність, таку як підтримка геоданих, повнотекстовий пошук, аналітика тощо.
5. Безпека та захист даних: PostgreSQL надає різні механізми безпеки для захисту даних, включаючи автентифікацію, авторизацію, шифрування,

контроль доступу та інші. Вона також має можливість резервного копіювання та відновлення даних для забезпечення надійності.

Дана СУБД є досить потужною для надійного зберігання, управління та операцій з даними. Вона використовується в різних сферах діяльності та різних розмірах проектів від великих до маленьких.

1.6 Аналіз аналогів інформаційної веб-системи

В першу чергу, інтерфейс інформаційної веб-системи торгівельної платформи повинен бути зрозумілим та простим для користувача, який не користується комп'ютером в великому обсязі. Проте друга не менш важлива вимога – бути зручним у використанні з мобільних пристроїв, планшетів, та на пк з різними розмірами екранів. Компоненти інтерфейсу мають бути так розміщені, щоб не виникало питань як перейти на ту чи іншу потрібну сторінку, та при цьому інтерфейс повинен бути правильного масштабу для того, чітко сприймати команди користувача з мобільних пристроїв

Перед розробкою інформаційної веб-системи потрібно проаналізувати її існуючі та успішні аналоги, було здійснено на прикладі веб-систем Compservice та Bicom (див. рисунок 1.1 та 1.2)

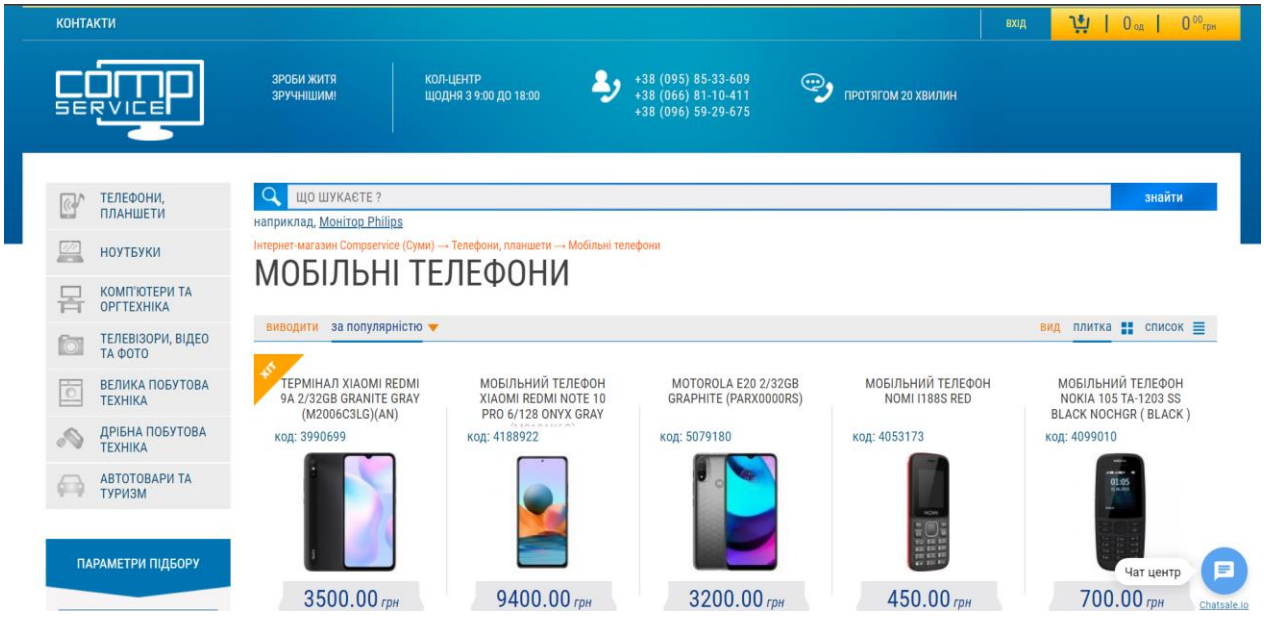

Рисунок 1.1- Головна сторінка веб-системи Compservice

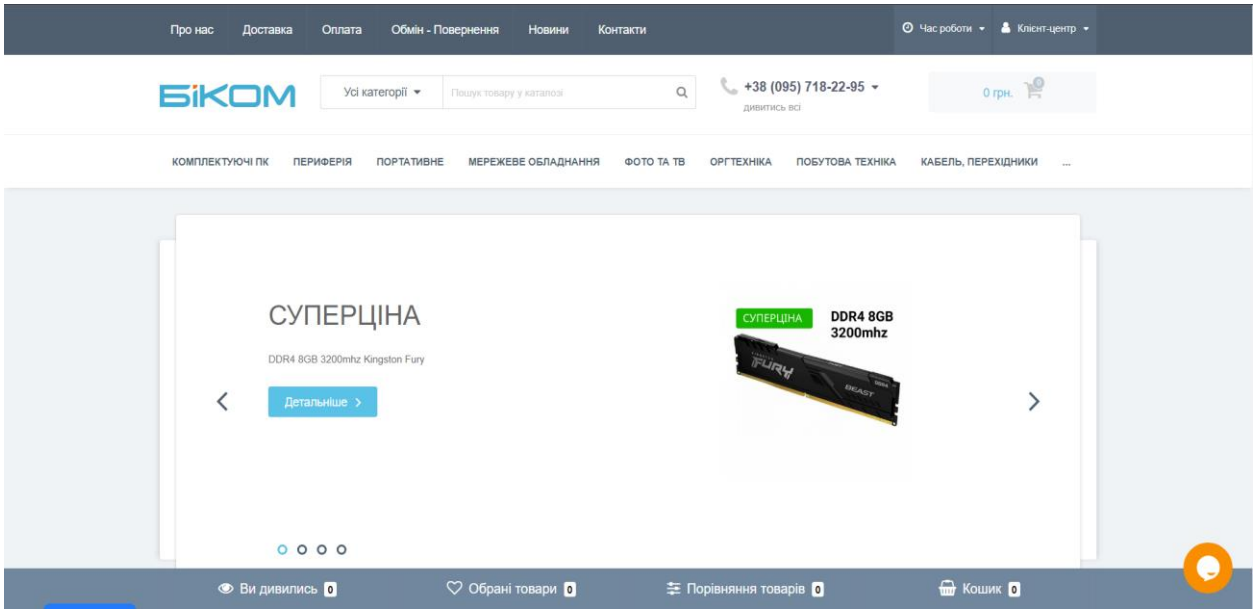

Рисунок 1.2 – Головна сторінка веб-системи Bicom

Основним недоліком є те, що веб-системи в своїх мобільних версіях мають замалий інтерфейс для зручного користування та подекуди хаотичні нагромадженя кнопок та різних панелей інтерфейсу.

Перевагами даних сайтів є описані нижче властивості:

- Відсутність зайвої реклами
- Швидкий доступ та завантаження веб-системи
- Досить сучасний дизайн
- Зрозумілість інтерфейсу для користувача
- Гарно підібрані кольори елементів інтерфейсу веб-системи

1.7 Постановка завдання

Створити інформаційну веб-систему для торгівельної платформи, яка здатна надавати можливості перегляду товарів, перегляд товарів по бренду чи типу товару, залишати заявку на замовлення. Функціонал з боку адміністратора веб-системи повинен включати в себе додавання нових товарів, додавання характеристики до товару, додавання нових брендів, додавання нових типів товарів, обробляти замовлення.

2. МОВИ ПРОГРАМУВАННЯ ТА ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Огляд використання технології Node.js та Express.js

Node.js — це міжплатформне середовище виконання JavaScript із відкритим кодом, яке виконує код JavaScript без веб-браузера. Це дозволяє розробникам запускати JavaScript на стороні сервера, створюючи програми на стороні сервера за допомогою JavaScript. Node.js побудовано на движку JavaScript V8 Chrome і дозволяє розробникам писати JavaScript на сервері. Він призначений для створення масштабованих мережевих програм і особливо корисний для програм реального часу, які вимагають високопродуктивної системи. Деякі з ключових особливостей Node.js включають його керовану подіями архітектуру, неблокуючу модель введення-виведення та здатність обробляти кілька одночасних з’єднань із високою пропускною здатністю. Він зазвичай використовується для створення веб-серверів, програм реального часу та мережевих інструментів.

Node.js базується на архітектурі, керованій подіями, і неблокуючій моделі введення-виведення, що робить його легким і ефективним. Він добре підходить для створення масштабованих, високопродуктивних веб-серверів та інших типів мережевих програм.

Однією з ключових переваг використання Node.js є його здатність обробляти велику кількість одночасних з’єднань із високою пропускною здатністю. Це досягається за допомогою моделі, керованої подіями, яка дозволяє серверу обробляти запити асинхронно, а не блокувати виконання програми під час очікування відповіді. Це робить його привабливим вибором для створення програм реального часу, таких як сервери чату та онлайн-ігри.

Node.js також має велике й активне співтовариство розробників, яке додало широкий спектр модулів до репозиторію npm (Node Package Manager), що полегшує додавання функцій до програм Node.js.

Крім того, що Node.js використовується для створення веб-серверів і програм реального часу, Node.js також широко використовується для створення інструментів командного рядка та для створення серверних служб для веб- і мобільних програм.

Є кілька причин, чому ви можете вибрати Node.js для свого проекту:

1. Швидкість: Node.js побудовано на механізмі Google V8 JavaScript, що означає, що він швидко виконує код JavaScript. Це робить його гарним вибором для створення швидких додатків у реальному часі.
2. Велика та активна спільнота розробників: Node.js має велику та активну спільноту розробників, яка додала широкий спектр високоякісних модулів до репозиторію npm. Це спрощує додавання нових функцій до програми Node.js.
3. Підходить для створення різноманітних додатків: Node.js — це універсальне середовище виконання, яке можна використовувати для створення широкого спектру додатків, включаючи веб-сервери, інструменти командного рядка та серверні служби для веб- і мобільних додатків.
4. Легко освоїти: якщо ви вже знаєте JavaScript, вам буде легко почати роботу з Node.js. Його синтаксис і основні концепції будуть знайомі, що робить його хорошим вибором для розробників, які вже знайомі з JavaScript.

2.2 Огляд використання технології React.js

Клієнтський веб-додаток на основі React.js це клієнтська частина веб додатку створена на даному фреймворку. Використання сучасного фреймворку для розробки користувацького інтерфейсу веб додатків дозволяє розробникам створювати інтерактивні та швидкодіючі додатки, які здатні забезпечити багатофункціональність та зручний досвід використання платформи.

Історія походження React.js починається в компанії Facebook, яка випустила його у 2013 році. У той час веб-розробка зустрічала проблему швидкодії та

ефективності, особливо при зміні стану додатку. Тому команда розробників Facebook зрозуміла потребу у новому підході до розробки інтерфейсу. Результатом їхньої роботи став React.js - бібліотека JavaScript, яка мала великий вплив на веб-розробку.

Переваги використання даного фреймворку:

1. Швидкість та ефективність роботи завдяки використанню віртуального DOM.
2. Масштабованість та повторне використання компонентів, що спрощує розробку та підтримку коду.
3. Велика активна спільнота розробників та наявність багатьох сторонніх бібліотек та компонентів.
4. Можливість легко інтегрувати з іншими інструментами та фреймворками, наприклад, Redux, React Router тощо.
5. Стабільність та постійне оновлення завдяки активному розвитку та підтримці від Facebook.

Проте не обійшлося без нижче описаних недоліків:

1. Потреба у додаткових інструментах та бібліотеках, таких як Webpack або Babel, що може збільшити складність налаштування проекту.
2. Необхідність вивчення JSX - спеціального синтаксису, що поєднує JavaScript та HTML, для створення компонентів.
3. Потреба у вмінні ефективно управляти станом додатку та розумінні принципів реактивного програмування.
4. Відсутність вбудованої підтримки для інших платформ, таких як мобільні пристрої чи десктоп-додатки, хоча є сторонні рішення для цього.

Проте ці недоліки та переваги можуть лише заважати, або навпаки поліпшувати розробку додатку лише в конкретних випадках з певними вимогами.

2.3 Використання фреймворку React BOOTSTRAP

Даний фреймворк комбінує моливості React та BOOTSTRAP, та надає можливості для швидкої та ефективної розробки стильних інтерфейсів веб-додатків надаючи набір готових компонентів та стилів, які легко використовувати.

Історія походження React Bootstrap пов'язана з появою самого Bootstrap - популярного фреймворку для розробки веб-додатків. Bootstrap був розроблений в компанії Twitter з метою спростити та уніфікувати процес розробки інтерфейсу. Згодом, розробники виявили потребу у поєднанні Bootstrap з можливостями React.js для створення більш інтерактивних та динамічних додатків, що призвело до створення React Bootstrap.

Переваги використаня даної технології:

1. Швидка та проста розробка: React Bootstrap надає готові компоненти та стилі з Bootstrap, що спрощує розробку інтерфейсу та зменшує час, потрібний для розробки.
2. Гнучкість: Завдяки використанню React.js, React Bootstrap дозволяє змінювати та налаштовувати компоненти за допомогою JavaScript, що дозволяє створювати динамічні та інтерактивні елементи інтерфейсу.
3. Компонентна модель: React Bootstrap використовує компонентну модель React.js, що дозволяє розробникам легко комбінувати та повторно використовувати компоненти для швидкої та зручної розробки інтерфейсу.

Недоліком використання двної технології є те що функціональність стилізації привязана до стилів та класів BOOTSTRAP, створюючи тим самим обмеження та складності при власному налаштуванні стилів або при використанні. А також залежність від версії даного фреймворку. Збільшуючи розмір завантажуваного коду може призвести до сповільнення швидкості завантаження веб-додатку.

2.4 Застосування RestAPI та ввідна інформація про Application programming interface

RestAPI або Representational State Transfer Application Programming Interface – архітектурний стиль для створення та взаємодії між різними програмними компонентами або системами за допомогою мережевого протоколу HTTP. RestAPI використовує стандартні HTTP методи, такі як GET, POST, PUT і DELETE, для передачі даних та виконання операцій над ресурсами.

Історія походження даного поняття починається ще з часів стандартизації веб протоколів HTTP і публікації Роя Філдінга архітектурного стилю REST в своїй докторській дисертації, де були описані правила для розробки масштабних та ефективних мереживих архітектур.

Переваги використання RestAPI:

1. Стандартизація: RestAPI використовує стандартні HTTP методи та статуси, що спрощує розуміння та взаємодію з API для розробників.
2. Незалежність від платформи: RestAPI може бути використаний з будь-якою мовою програмування та платформою, оскільки використовує загальні протоколи та стандарти.
3. Скалківність: RestAPI дозволяє гнучко масштабувати систему, оскільки ресурси можуть бути доступні та використовувані незалежно один від одного.
4. Легкість кешування: RestAPI підтримує кешування на рівні протоколу HTTP, що поліпшує швидкодію та знижує навантаження на сервер.

Недоліки використання архітектурної технології:

1. Залежність від протоколу HTTP: RestAPI використовує HTTP як основний протокол, що може бути обмеженням для деяких випадків використання, де інші протоколи були б більш підход.
2. Обмеженість виразності: RestAPI має обмежений набір HTTP методів (GET, POST, PUT, DELETE), що може бути недостатньо для виразності деяких складних вимог або операцій.

3. Передача зайвих даних: RestAPI передає дані у вигляді тексту (наприклад, JSON або XML), що може включати зайву інформацію та призводити до збільшення обсягу передачі даних.
4. Складність управління станом: RestAPI не має вбудованих механізмів для управління станом на сервері та клієнті. Це може вимагати додаткових зусиль для збереження та керування станом додатків.
5. Відсутність стандартизованого опису API: RestAPI не має стандартизованого способу опису API, що може призводити до неоднозначності та складнощів у розумінні та використанні API для розробників.

Принципи використання архітектурної технології описану на схематично (див. рисунок 2.1), де використовуються клієнт, сервер, прошарок у вигляді RestAPI, запит поступає з клієнту до прошарку у вигляді архітектурної технології, після чого вона переправляє структурований запит на сервер у вигляді правильно створеного шляху та файлів, тощо. Після обробки сервер дає відповідь через той же прошарок.

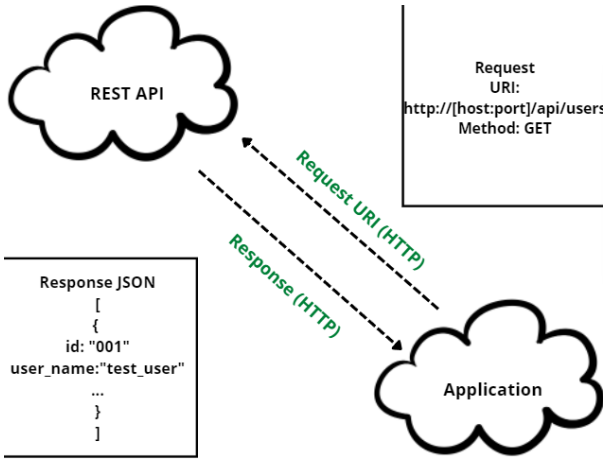

Рисунок 2.1 - Модель REST API

Клієнт - це програма, яка ініціює запит до API. Це може бути веб-браузер, мобільний додаток або будь-який інший додаток, який надсилає запити через Інтернет.

Запит - запит надсилається клієнтом до API та містить інформацію про операцію, яку клієнт хоче виконати. Запит містить метод (наприклад, GET, POST, PUT, DELETE) і URI (уніфікований ідентифікатор ресурсу), який визначає ресурс, з яким потрібно діяти. Він також може включати корисне навантаження даних, наприклад дані форми або об’єкт JSON.

Сервер - це програма, яка отримує запит від клієнта та обробляє його. Він може отримувати дані з бази даних або виконувати інші операції на основі запиту.

Відповідь відповідь надсилається сервером клієнту у відповідь на запит. Він містить код стану, що вказує на результат запиту (наприклад, 200 для успіху, 404 для не знайдено), а також може містити корисні дані.

- 200 OK: успішне виконання запиту
- 404 Not Found: ресурс не знайдено
- 400 Bad Request: невірний запит
- 201 Created: створено
- 401 Unauthorized: несанкціонований вхід/ помилка авторизації
- 403 Forbidden: користувач не має доступу до ресурсу
- 500 Internal Server error: помилка серверу
- 502 Bad Gateway: невалідний респонс від серверу
- 503 Service Unavailable: помилка на стороні серверу

Ресурс - це об’єкт даних, який API надає клієнтам. Він ідентифікується URI, і ним можна керувати за допомогою методів HTTP, таких як GET, POST, PUT і DELETE.

- GET: запит до ресурсу
- POST: створення нового ресурсу
- PUT: оновлення існуючого ресурсу
- DELETE: видалення існуючого ресурсу
- PATCH: часткова зміна ресурсу

- OPTIONS: опис параметрів з’єднання з ресурсом

3. ОСОБЛИВОСТІ РОЗРОБКИ ТОРГІВЕЛЬНОЇ ПЛАТФОРМИ

3.1 Проектування бази даних

Дані будемо зберігати в базі даних. В якості СУБД буде обрано PostgreSQL (див. рисунок 3.1)

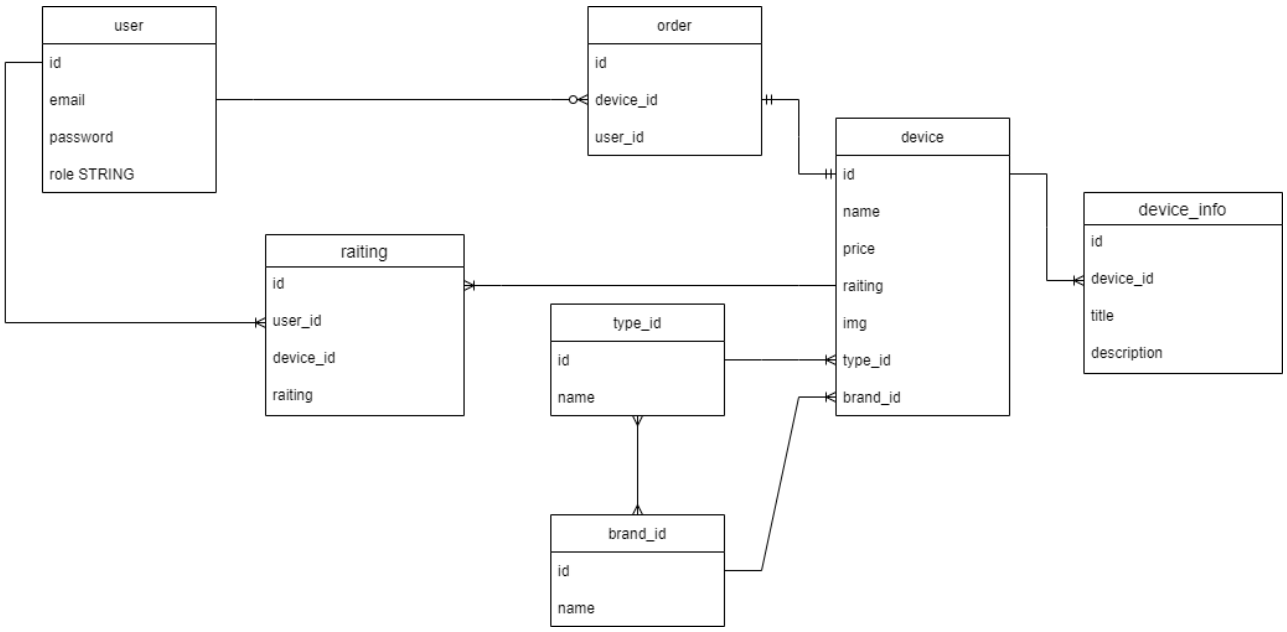

Рисунок 3.1 – Діаграма сутностей та зв’язків між ними веб-системи

USER:

Id – ідентифікатор користувача

Email – електронна пошта користувача

Password – пароль користувача

Role – роль облікового запису користувача

CART:

Id – ідентифікатор корзини яка належить користувачу

User_id – ідентифікатор користувача корзини

CART_DEVICE:

Id – ідентифікатор пристрою в корзині

Device_id – ідентифікатор пристрою

Cart_id – ідентифікатор корзини користувача

DEVICE:

Id – ідентифікатор девайсу

Name – назва девайсу

Price – ціна девайсу

Raiting – рейтинг девайсу

Img – зображення девайсу

Type_id – ідентифікатор типу девайсу

Brand_id – ідентифікатор бренду девайсу

RAITING:

Id – ідентифікатор рейтингу

User_id – ідентифікатор користувача (того який поставив оцінку)

Device_id – ідентифікатор девайсу (якому було поставлено рейтинг)

Raiting – кількість балів рейтингу

DEVICE_INFO:

Id – ідентифікатор інформації девайсу

Device_id – ідентифікатор девайсу

Title – назва девайсу

Description – опис девайсу

TYPE_ID:

Id – ідентифікатор типу девайсу

Name – назва типу девайсу

BRAND_ID:

Id – ідентифікатор бренду виробника девайсу

Name – назва бренду

3.2 Побудова Application programming interface

Для побудови структури API було використано таку схему (див. рисунок 3.2.)

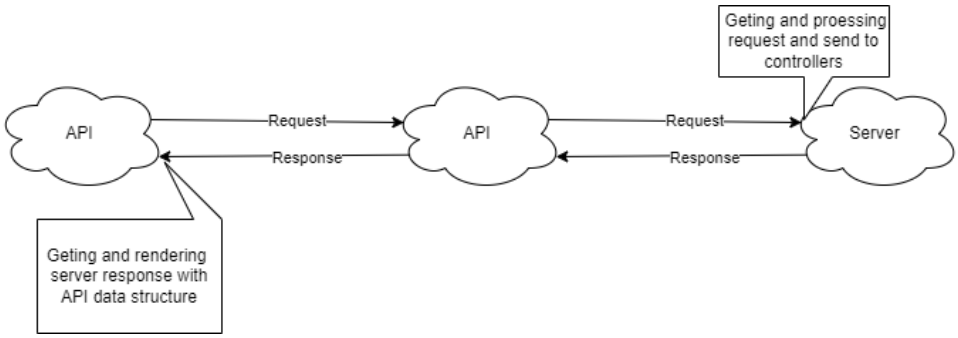

Рисунок 3.2 – Схематичне зображення структури проекту та принципу роботи API

При відправленні запиту з клієнта, він формує заголовок, післячого у вигляді запиту йде до прошарку RstAPI, в якому він структурується та доходить до сервера по заданному в API шляху, та доставляє запит до потрібного контролеру, після чого, контролер на сервері вмикає викликаний функціонал, оправцьовує запит, за потреби взаємодіє з СУБД, та у разі успішного виконання повертає результат у вигляді відповіді, яка надсилається до клієнта через той же прошарок API, який потім обробляється на клієнті та відображаються у вигляді змін інтерфейсу користувача та актуалізації даних в ньому.

3.2 Структура серверної частини

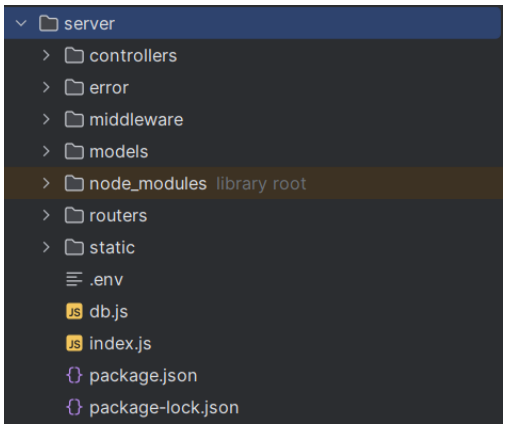

Рисунок 3.3 – вставка зі знімком файлової структури проекту в середі розробки

Контролери відповідають за обробку запитів (додавання та отримання даних про об’єкти) о бази данних

Основні роутери при побудові API

```
router.use('/user', userRouter);
router.use('/type', typeRouter);
router.use('/brand', brandsRouter);
router.use('/device', deviceRouter);
```

За допомогою даних роутерів виконується взаємодія саме з контроллерами які вже допомогають обробляти запити

3.2.1 Реалізація deviceController.js

Одним з найскладніших моментів в написанні контролера було збереження товару в базі даних, оскільки потрібно було ще й зберігати зображення до кожного з товарів, таким чином було прийнято рішення використовувати принципом статиків, для збереження об’єктів на сервері, а для завдання унікальної назви, яка вноситься до бази даних на сервері було використано пакет UUID, генерація та запис зображення виглядає наступним чином:

Спочатку зображення виділяється з тілу запиту до сервера, потім за допомогою деструктуризації ми отримуємо об’єкт з яким вже продовжується подальша робота, паралельно генеруємо «рандомне» ім’я для зображення з приставкою формату файлу. Після того як було отримано саме зображення до змінної та сформовано ім’я файлу, сам файл записується до директорії на сервері в папку static.

```
await img.mv(path.resolve(path.resolve(), 'static', fileName));
```

назва використовується двічі, вона надається файлу який було записано на сервері, а також як один з параметрів об’єкта який зберігається на сервері

```
const device = await Device
```

```
.create({name, price, brandId, typeId, img: fileName});
```

Для отримання списку девайсів використовується метод наявний в пакеті до Node.js Sequelize

```
Device.findAndCountAll({limit, offset});
```

В даному випадку до метода передається сторінка та обмеження кількості девайсів на сторінці.

Для отримання конкретного девайсі використовується його айді, що демонструє код описаний нижче

```
await Device.findOne(
   {
      where: {id},
      include: [{model:DeviceInfo, as: 'info'}]

   }
```

3.2.2 Реалізація brandController.js

Концепція цього контролера досить проста та подібна до контролера типів девайсів

```
async create(req, res) {
   const {name} = req.body;
   const brand = await Brand.create({name});
   return res.json(brand);
}

async getAll(req, res) {
   const brands = await Brand.findAll();
   return res.json(brands);
}
```

з коду який описаний вище видно, що для створення та отримання брендів використовуються стандартні методи пакету Sequelize, при створенні передається параметр ім’я, ідентифікатору генерується самою базою даних.

3.2.3 Реалізація typeController.js

Програмний код даного контролера ідентичний до контролера бренду.

```
async create(req, res) {
    const {name} = req.body;
    const type = await Type.create({name});
    return res.json(type);
}

async getAll(req, res) {
    const types = await Type.findAll();
    return res.json(types)
}
```

3.2.4 Реалізація userController.js

Написання програмного коду для даного контролера дещо відрізняється від попередньої, адже для створення облікових записів потрібно шифрувати паролі, для цього використовуються пакет bcrypt для хэшування даних та пакет JWT для створення JWT токенів

Для реєстрації з тіла запиту береться імейл, пароль та роль користувача, після чого виконується валідація отриманих даних щоб унеможливити реєстрацію користувача без імейлу чи пароля, а також без ролі.

```
async registration(req, res, next) {
    const {email, password, role} = req.body;
    if(!email || !password){
        return next(ApiError.badRequest("Incorrect email or password"));
```

```
    }
    const candidate = await User.findOne({where: {email}});
    if(candidate){
        return next(ApiError.badRequest("User email is used, try new one"));
    }
    const hashPassword = await bcrypt.hash(password,5);
    const user = await User.create({email, role, password: hashPassword});
    const cart = await CartDevice.create({userId: user.id});
    const token = generateJWT(user.id, user.email, user.role);
    return res.json({token})
}
```

Якщо всі дані коректні, то пароль хешується за допомогою пакету bcrypt, сіль для створення хешу було обрано 5 разів, щоб не перенавантажувати сервер після цього відправляється запит на сервер для створення нового користувача, а у якості відповіді формується JWT токен який потім за допомогою JSON відправляється на клієнт.

3.2.5 Обробка помилок при запитах з клієнта на сервер

Для обробки помилок використовується модуль ApiError.js

В ньому обробляється 3 типові помилки, які можуть відбутись при запиті, а на відповіді під час помилки буде відправлено статус помилки та повідомлення, код даного модуля виглядає наступним чином

```
class ApiError extends Error{
    constructor(status, message) {
        super();
        this.status = status;
        this.message = message;
    }
    static badRequest(message){
```

```
        return new ApiError(404, message);
    }
    static internal(message){
        return new ApiError(500, message);
    }
    static forbidden(message){
        return new ApiError(403, message);
    }
}

export default ApiError;
```

при цьому ключовим з компонентів обробки помилок запитів на сервер є компонент ErrorHandlingMiddleware.js, в якому саме відбувається виявлення конкретної помилки, а саме який у неї тип, та вже надсилання повідомлення зі статусом помилки та повідомлення про те чому саме відбулась помилка, скрипт виглядає наступним чином:

```
import ApiError from "../error/ApiError.js";

export default  function(err, req, res, next){
    if(err instanceof ApiError){
        return res.status(err.status).json({
            message: err.message,
        })
    }
    return res.status(500).json({
        message: "Unknown error",
    })
}
```

у випадку якщо помилка не знайдена в ApiError то буде відправлятись помилка зі статус кодом 500 та повідомлення «невідома помилка»

3.3 Структура клієнтської частини

Клієнтська частина проекту веб системи складається з файлів зображеним нижче у знімках (див. рисунок 3.4 – рисунок 3.6)

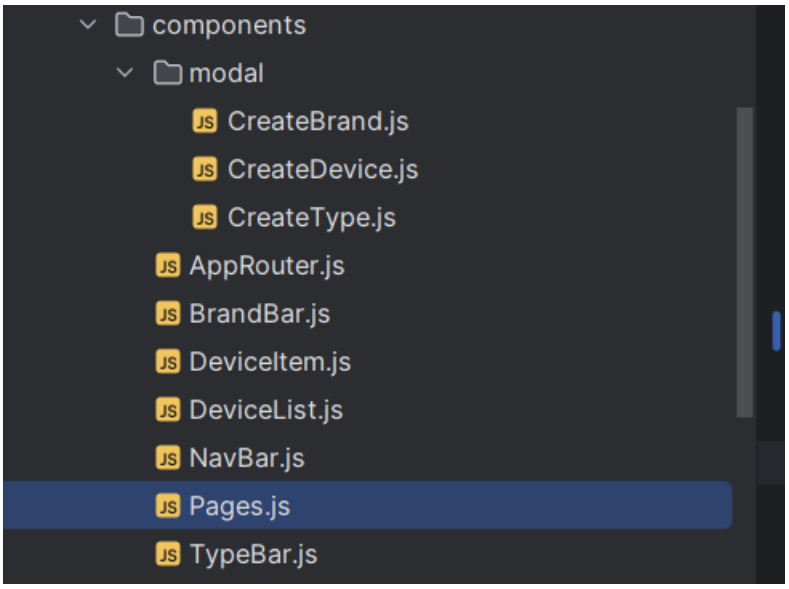

Рисунок 3.4 – зображення всіх створених компонентів у проекті на клієнтській частині.

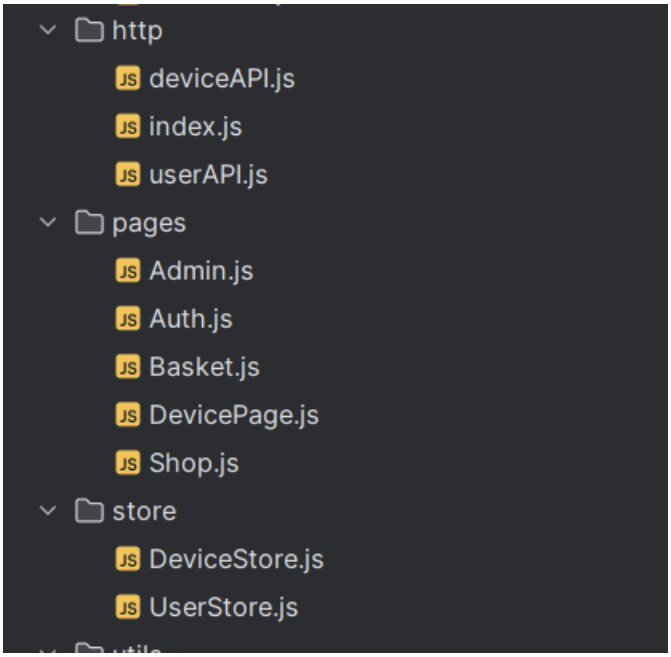

Рисунок 3.5 – Зображення структури обробки HTTP протоколів, сторінок які відображаються та сховищ для збереження інформації яка обробляється зі сторони клієнту

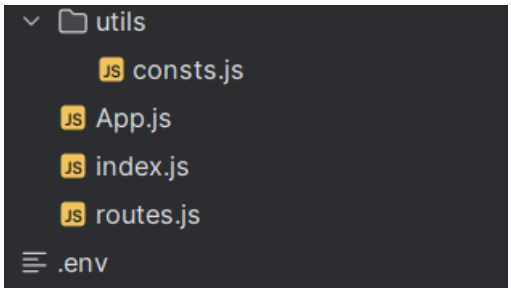

Рисунок 3.6 – Зображення структури утіліті, в якому зберігаються константи, та базових елементів для взаємодії функціоналу між собою, та роутери.

В підрозділах нижче будуть описані шляхи кодування важливих елементів відображених на зображеннях (див рисунок 3.4 – рисунок 3.6)

3.3.1 Використані пакетні залежності

Для підвищення зручності та швидкості розробки, були використані додаткові NPM пакети, окрім тих, що використовуються автоматично при створенні Front-end додатку на основі React.js, а саме:

- AXIOS - JavaScript-бібліотека, яка використовується для здійснення HTTP-запитів з браузера або з середовища Node.js. Вона надає простий та зрозумілий інтерфейс для виконання запитів до сервера і обробки отриманих відповідей.
- MOBX - бібліотека управління станом (state management) для JavaScript-програм, зокрема для React та React Native додатків. Вона дозволяє спростити управління та синхронізацію стану додатка та реагувати на зміни цього стану.
- REACT BOOTSTRAP

3.3.2 Реалізація routers.js

```
export const authRoutes = [
    {
        path: ADMIN_ROUTE,
        Component: Admin
    }
]
export const publicRoutes = [
    {
        path: SHOP_ROUTE,
        Component: Shop
    },
    {
        path: LOGIN_ROUTE,
        Component: Auth
    },
    {
        path: REGISTRATION_ROUTE,
        Component: Auth
    },
    {
        path: DEVICE_ROUTE + '/:id',
        Component: DevicePage
    }
]
```

З даної ділянки коду можна побачити те, що інформація структурується та розділяється на авторизованого користувача та не авторизованого, після чого відбувається розділення за правами та можливостями користувачів. Всі шляхи маршрутизації описані в ділянці коду знаходяться в файлі констант для зручності використання, та пришвидшення написання коду, константи описані нижче:

```
export const ADMIN_ROUTE = '/admin'
export const LOGIN_ROUTE = '/login'
export const REGISTRATION_ROUTE = '/registration'
export const SHOP_ROUTE = '/'
export const BASKET_ROUTE = '/basket'
export const DEVICE_ROUTE = '/device'
```

3.4 Опис інтерфейсу інформаційної веб-системи

З функціоналу розробленого інформаційної веб-системи можна помітити, що при вході на головну сторінку є кнопка авторизації (див. рисунок 3.7) , вона створена для адміністратора системи, за для доступу до функціоналу адміністрування (чит. розділ 3.3), завдяки обробці, система дізнається, чи користувач є авторизованим.

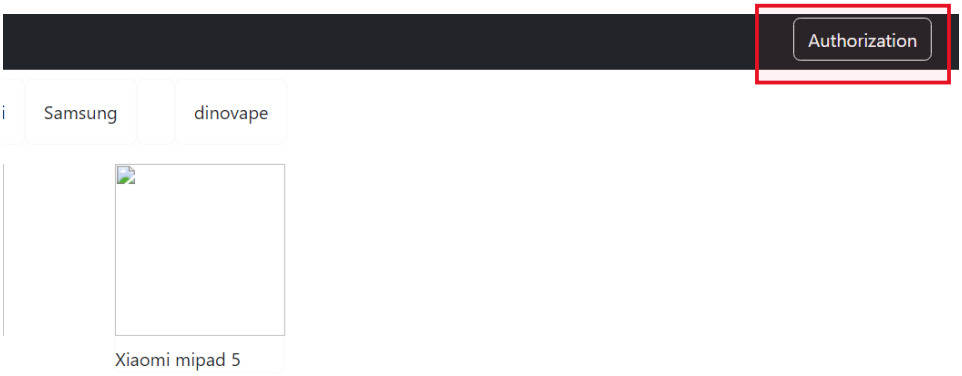

Рисунок 3.7 – елемент головної сторінки з кнопкою авторизації

Після виклику компонента авторизації, з’являється вікно для введення та обробки даних (див. рисунок 3.8)

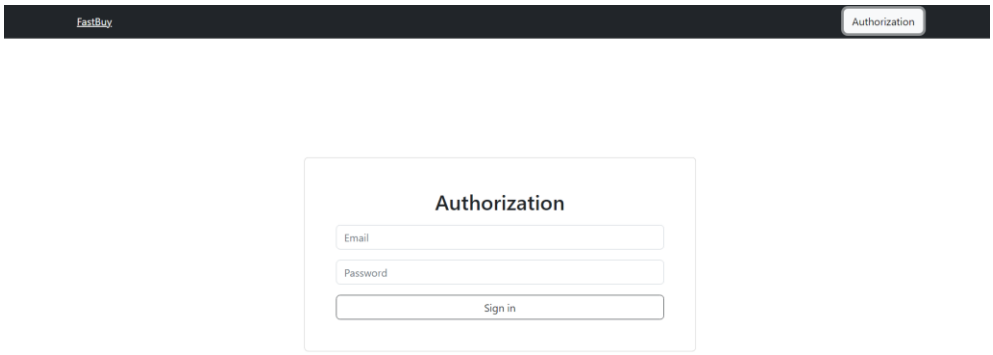

Рисунок 3.8 – вікно авторизації веб системи

Було проаналізовано головну сторінку веб-системи (див. рисунок 3.9), на якій можна побачити, шапку сторінки на якій є надпис з назвою веб-системи, а також кнопка авторизації до неї адміністрації, з ліва від списку товарів розташовано меню вибору типів товарів, з верхньої частини тіла головної сторінки над списком товарів розміщено меню вибору бренду, що цікавий користувачу. По середині розташовується список товарів з посторінковим виводом кожного товару. При натисканні на сторінку товару відбувається перехід до компоненти в якій відображається структура товару (див. рисунок 3.10) , що цікавить користувача, ціною, його зображенням, та характеристиками виведеними у вигляді таблиці.

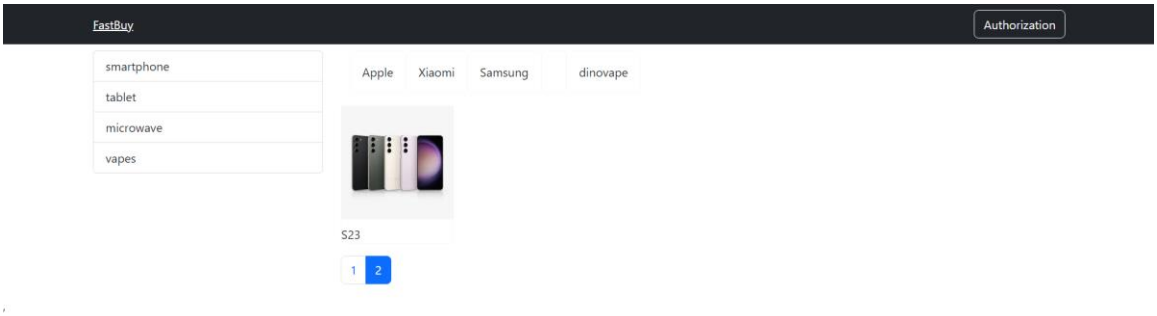

Рисунок 3.9 – головна сторінка інформаційної веб-системи

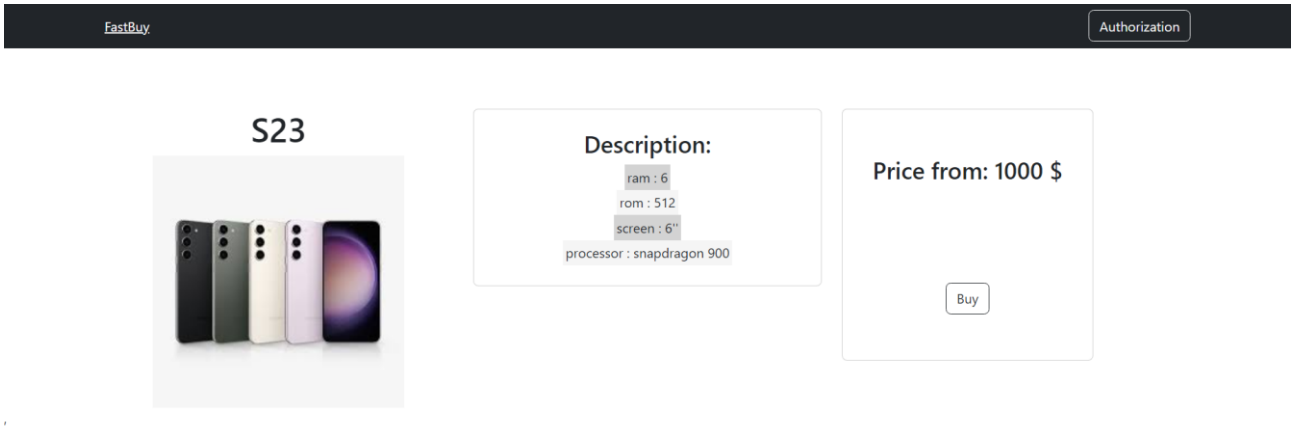

Рисунок 3.10 – сторінка товару з його зображенням, описом та ціною

При авторизації, шапка головної сторінки видозмінюється (див. рисунок 3.11) та додається кнопка виходу користувача із системи, та кнопка переходу до панелі адміністрування веб-системи.

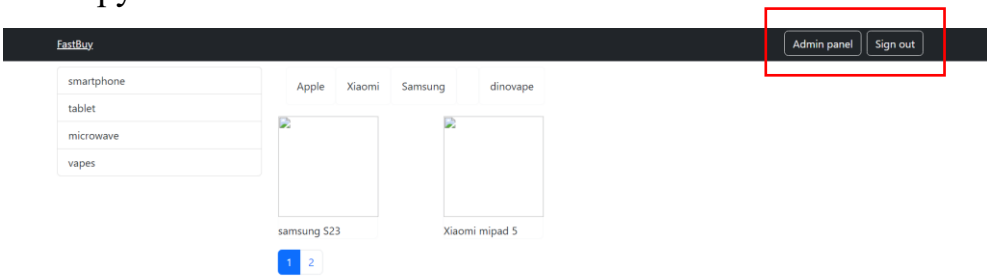

Рисунок 3.11 – вигляд головної сторінки авторизованого користувача.

Коли кнопку переходу до панелі адміністрування активовано, розгортається інтерфейс (див рисунок 3.11) що надає змогу викликати модальні вікна (див. рисунок 3.12 – рисунок 3.15), що обробляють інформацію та заносять її до веб системи.

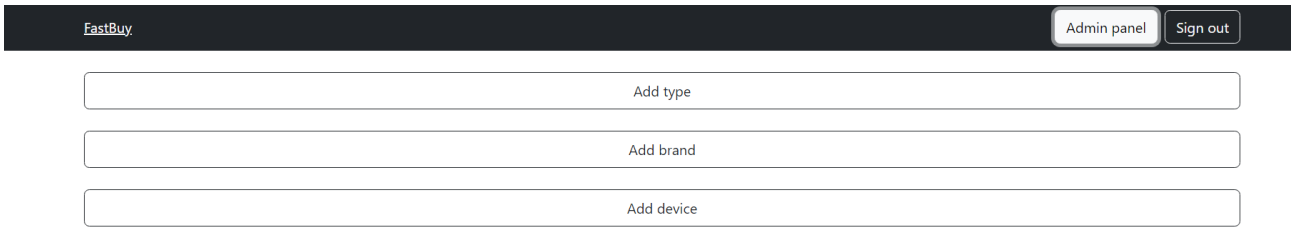

Рисунок 3.12 – панель адміністрування Веб-системи

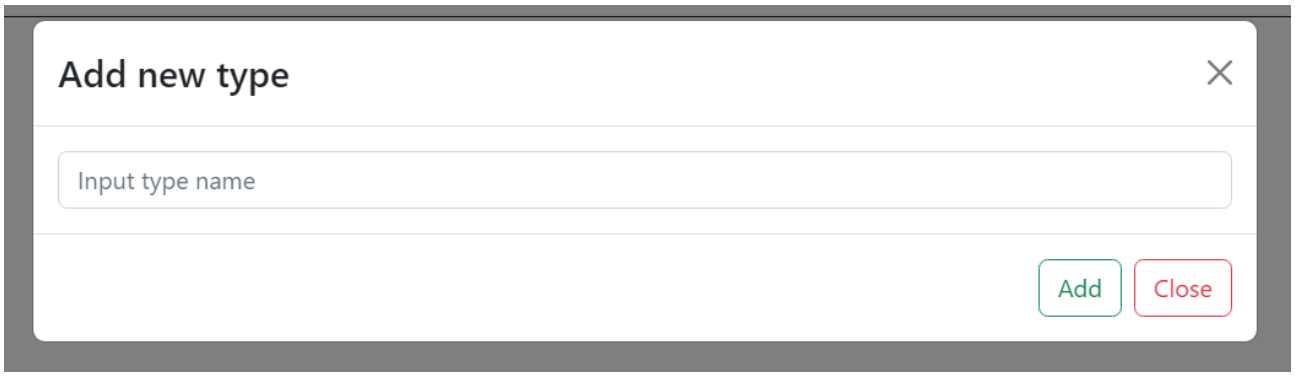

Рисунок 3.13 – Модальне вікно додавання нового типу

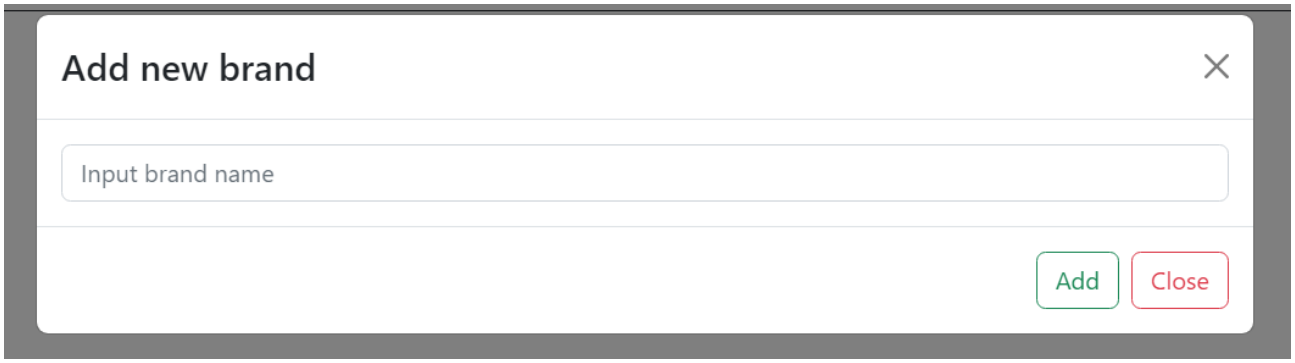

Рисунок 3.14 – Модальне вікно додавання нового бренду

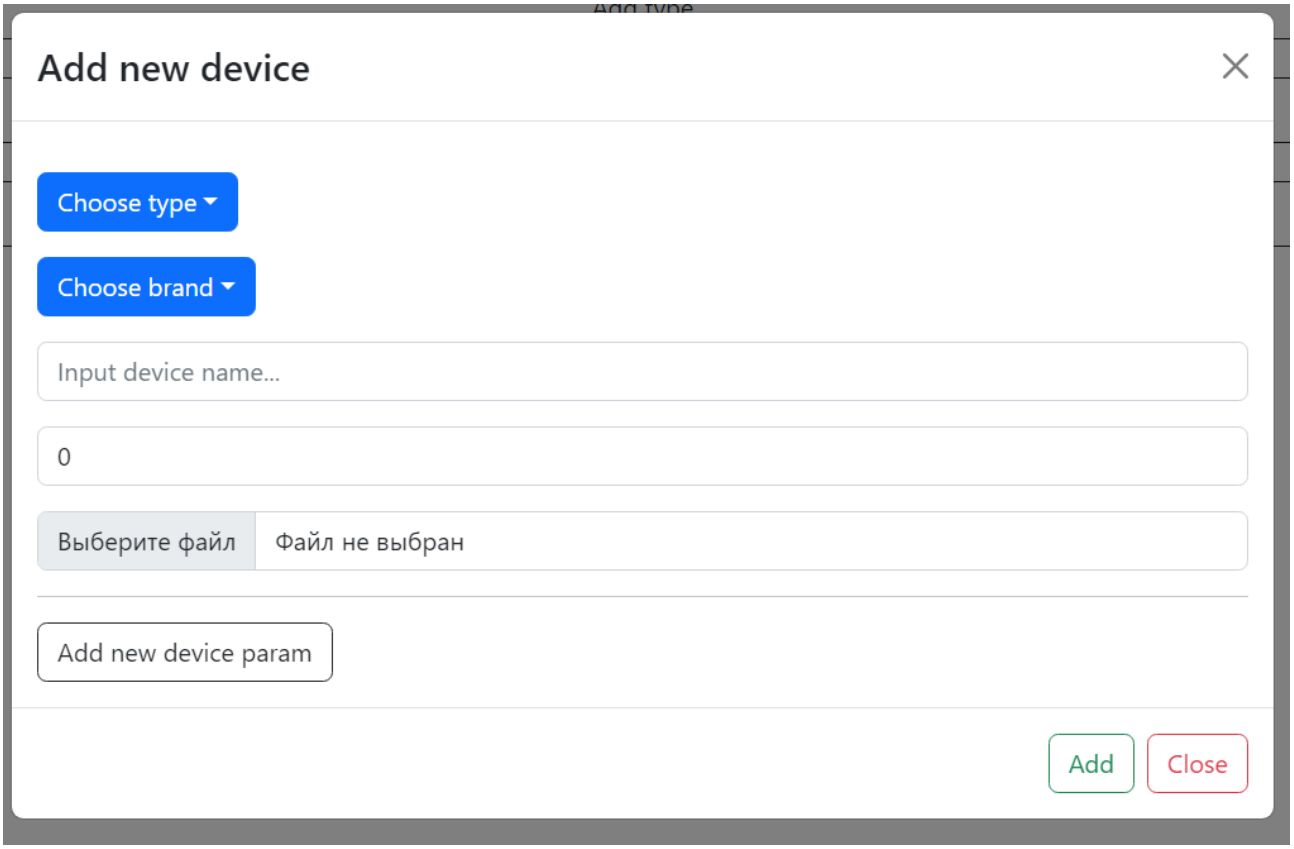

Рисунок 3.15 – Модальне вікно додавання нового пристрою до Веб-системи

ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було розроблено інформаційну веб-систему торгівельної платформи. Проаналізовано аналогічні веб-системи, та виявлено в них переваги та недоліки, які були враховані у створенні подібної за своєю концепцією торгівельної платформи.

Підчас розробки було виконано всі поставлені вимоги, та виконано тестування клієнтської частини на різних браузерах та з мобільних пристроїв. Розробка велась за допомогою мереживної технології RestAPI, мови програмування JavaScript та фреймворків у вигляді Node.js, Express.js та React.js. За для прискорення стилізації була використана бібліотека React BOOTSTRAP, та по мінімуму використовувались CSS стилі, у якості способу збереження даних була використана СУБД PostgreSQL.

В результаті виконаної роботи було отримано інформаційну веб-систему для торгівельної платформи, яку можна використовувати у мереживній торгівлі чи то як сайт з інформацією від місцевого виробника, продавця певних товарів чи послуг. Над функціоналом сайту було виконано тестування на виявлення помилок та недоліків.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Фелдінг, Рой Томас, Taylor, Річард. "Архітектура веб-програмного забезпечення". O'Reilly Media, 2007. - 576 с.
2. Мардан, Азат. "Express.js практичне керівництво". O'Reilly Media, 2014. - 384 с.
3. Шевром, Ларрі М. "Веб-розробка з Express та Node.js для початківців". Apress, 2014. - 352 с.
4. Вейн, Лоуренс. "Express.js. Повне керівництво з розробки веб-додатків". O'Reilly Media, 2016. - 256 с.
5. Цуі, Дэйв. Повний курс з React: Розробка сучасних веб-додатків. O'Reilly Media, 2020. - 360 с.
6. Робін Вірдерс, Том Байндер. "React і React Native: Повний курс". O'Reilly, 2018. - 790 с.
7. Адам Бодуч. "React для веб-розробників". Packt Publishing, 2019. - 398 с.
8. Масон, Майкл. "RESTful Web API: принципи, практика та патерни". O'Reilly Media, 2013. - 408 с.
9. Макфарланд, Стівен. "API Design Patterns: Покроковий підхід до проектування розподілених систем". O'Reilly Media, 2021. - 404 с.
10. Кауфман, Скотт. "Designing Web APIs: Building APIs That Developers Love". O'Reilly Media, 2018. - 270 с.
11. Офіційна документація React.js [Електронний ресурс]
12. Офіційна документація Express.js [Електронний ресурс]
13. Офіційна документація React BOOTSTRAP [Електронний ресурс]
14. Офіційна документація Git [Електронний ресурс]
15. Офіційна документація PostgreSQL [Електронний ресурс]

ДОДАТКИ

Додаток А – Код компонентів головної сторінки, шапки, сторінки девайсу, списку брендів, списку пристроїв

А.1 Код компонента Shop.js

```
import React, {useContext, useEffect} from 'react';
import {Container} from "react-bootstrap";
import Row from "react-bootstrap/Row";
import Col from "react-bootstrap/Col";
import TypeBar from "../components/TypeBar";
import BrandBar from "../components/BrandBar";
import DeviceList from "../components/DeviceList";
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {fetchBrands, fetchDevices, fetchTypes} from "../http/deviceAPI";
import Pages from "../components/Pages"
import data from "bootstrap/js/src/dom/data";

const Shop = observer(() => {
    const {user} = useContext(Context)
    const {device} = useContext(Context)



    useEffect(() => {
        fetchTypes().then(data => device.setTypes(data))
        fetchBrands().then(data => device.setBrand(data))
        fetchDevices(null, null, 1, 2).then(data => {
            device.setDevice(data.rows)
            device.setTotalCount(data.count)
        })
```

```
    }, [])

    useEffect(() => {
        fetchDevices(device.selectedType.id, device.selectedBrand.id, device.page,
2).then(data => {
            device.setDevice(data.rows)
            device.setTotalCount(data.count)
        })
    }, [device.page, device.selectedType, device.selectedBrand,])

    return (
        <Container>
            <Row className="mt-2">
                <Col md={3}>
                    <TypeBar/>
                </Col>
                <Col md={9}>
                    <BrandBar/>
                    <DeviceList/>
                    <Pages/>
                </Col>
            </Row>
        </Container>
    );
});

export default Shop;
```

А.2 Код компонента BrandBar.js

```
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {Card, Container} from "react-bootstrap";

const BrandBar = observer (() => {
   const {device} = useContext(Context);
   return (
      <Container className="d-flex flex-wrap">
         {device.brands.map((brand)=>
            <Card
               key={brand.id}
               className="p-3"
               onClick={()=> device.setSeectedBrand(brand)}
               border={ brand.id === device.selectedBrand.id ? "dark" : "light"}
               style={{cursor:"pointer"}}
            >
               {brand.name}
            </Card>
         )}
      </Container>
   );
});

export default BrandBar;
```

Код компонента TypeBar.js

```
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
```

```
import {ListGroup} from "react-bootstrap";

const TypeBar = observer(() => {
   const {device} = useContext(Context);

   return (
      <ListGroup >
         {device.types.map(type =>
            <ListGroup.Item
               style={{cursor:"pointer"}}
               active={type.id === device.selectedType.id}
               onClick={()=> device.setSelectedType(type)}
               key={type.id}
            >
               {type.name}
            </ListGroup.Item>
         )}
      </ListGroup>
   );
});

export default TypeBar;
```

А.3 Код компонента DeviceList

```
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {Row} from "react-bootstrap";
import DeviceItem from "./DeviceItem";
```

```
const DeviceList = observer( () => {
    const {device} = useContext(Context);
    return (
        <Row className="d-flex">
            {device.devices.map(device =>

                <DeviceItem key={device.id} device={device}/>
            )}
        </Row>
    );
});

export default DeviceList;
```

А.4 Код компонента Pages.js

```
import React, {useContext} from 'react';
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {Pagination} from "react-bootstrap";

const Pages = observer(() => {
    const {device} = useContext(Context)
    const pageCount = Math.ceil((device.totalCount / device.limit) + 1)
    const pages = []

    for (let i = 0; i < pageCount; i++) {
        pages.push(i+1)
    }
```

```
    return (
        <Pagination className="mt-3">
            {pages.map(page =>
                <Pagination.Item
                    key={page}
                    active={device.page === page}
                    onClick={() => device.setPage(page)}
                >
                    {page}
                </Pagination.Item>
            )}
        </Pagination>
    );
});

export default Pages;
```

Додаток Б – Контролери, роутери, обробка запитів до них та відповідей сервера

Б.1 Реалізація Index.js

```
import express from "express";
import dotenv from "dotenv";
import sequelize from "./db.js"
import cors from "cors";
import fileUpload from "express-fileupload";
import routers from "./routers/index.js";
import errorHandlingMiddleware from "./middleware/ErrorHandlingMiddleware.js";
import * as path from "path";


dotenv.config()

const PORT = process.env.PORT || 3000;
const app = express();

app.use(cors());
app.use(express.json());
app.use(express.static(path.resolve(
   path.resolve(), 'static'
   ))

);
app.use(fileUpload({ }));

app.use('/api', routers);
//errors handler
```

```
app.use(errorHandlingMiddleware);

const startApp = async () => {
    try {
        await sequelize.authenticate();
        await sequelize.sync();

        app.listen(PORT, () => {
            console.log('server started on port ' + PORT);
            console.log('http://localhost:' + PORT);
        })
    } catch (err) {
        console.log(err);
    }
}

startApp();
```

Б.2 Реалізація routers/index.js

```
import Router from "express";
import brandsRouter from "./brandsRouter.js";
import deviceRouter from "./deviceRouter.js";
import typeRouter from "./typeRouter.js";
import userRouter from "./userRouter.js";

const router = new Router();

router.use('/user', userRouter);
router.use('/type', typeRouter);
router.use('/brand', brandsRouter);
```

```
router.use('/device', deviceRouter);


export default router;
```

Б.3 Реалізація BrandRouter.js

```
import Router from "express";
import BrandController from "../controllers/brandController.js";
import checkRoleMiddleware from "../middleware/checkRoleMiddleware.js";

const router = new Router();

router.post('/', checkRoleMiddleware("ADMIN"), BrandController.create);
router.get('/', BrandController.getAll);


export default router;
```

Б.4 Реалізація DeviceRouter.js

```
import Router from "express";
import DeviceController from "../controllers/deviceController.js";
import checkRoleMiddleware from "../middleware/checkRoleMiddleware.js";

const router = new Router();

router.post('/', checkRoleMiddleware("ADMIN"), DeviceController.create);
router.get('/', DeviceController.getAll);
router.get('/:id', DeviceController.getOne);
```

```
export default router;
```

Б.5 Реалізація TypeRouter.js

```
import Router from "express";
import TypeController from "../controllers/typeController.js";
import checkRoleMiddleware from "../middleware/checkRoleMiddleware.js";

const router = new Router();

router.post('/', checkRoleMiddleware('ADMIN'), TypeController.create);
router.get('/', TypeController.getAll);



export default router;
```

Б.6 Реалізація UserRouter.js

```
import Router from "express";
import UserController from "../controllers/userController.js";
import userController from "../controllers/userController.js";
import autMiddleware from "../middleware/autMiddleware.js";

const router = new Router();

router.post('/registration', UserController.registration);
router.post('/login', userController.login);
router.get('/auth', autMiddleware, userController.check);



export default router;
```