Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інтернет магазину продажу туристичного спорядження «Еверест»

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Стрильбіцький О.В.
(підпис) (прізвище та ініціали)

Керівник _____ Литвиненко Я.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Яцишин В.В.
(підпис) (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«____» _червня_ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Стрильбіцькому Олександру Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інтернет магазину продажу туристичного спорядження «Еверест»

Керівник роботи Литвиненко Ярослав Володимирович, д.т.н., про. професор кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*29*» *квітня* 2024 року № *4/7-470.*

2. Термін подання студентом завершеної роботи 24 червня 2024р.

3. Вихідні дані до роботи Літературні та інтернет джерела, які застосовані для розробки Інтернет магазину продажу туристичного спорядження "Еверест"

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання. 1.1 Поняття, види та застосування веб сайтів. 1.2 Структура створення веб сайту. 1.3 Аналіз та постановка завдання. 1.4 Висновок до першого розділу. 2. Проектування та опис технологій. 2.1 Опис технологій клієнтської частини коду. 2.2 Опис серверної частини коду. 2.3 Опис технології бази даних. 2.4 Опис структури бази даних та проекту. 2.5 Висновок до другого розділу. 3. Практична частина. 3.1 Серверна частина коду. 3.2 Клієнтська частина коду. 3.3 Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. 4.1 Ергономічні проблеми безпеки Життєдіяльності. 4.2 Проведення інструментів з охорони праці. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С. к.т.н., доцент кафедри МТ		

7. Дата видачі завдання 29 січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	30.01.2024	*Виконано*
2.	Підбір джерел про поняття веб сайтів	31.01.2024-03.02.2024	*Виконано*
3.	Опрацювання джерел по темі кваліфікаційної роботи	04.02.2024-06.02.2024	*Виконано*
4.	Виконання дослідження щодо необхідних технологій Розроблення веб сайту	07.02.2024-11.02.2024	*Виконано*
5.	Оформлення розділу «Аналіз предметної області та постановка завдання»		
6.	Оформлення розділу «Проектування та опис технологій»		
7.	Оформлення розділу «Практична частина»		
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»		
9.	Виконання завдання до підрозділу «Основи охорони праці»		
10.	Оформлення кваліфікаційної роботи		
11.	Нормоконтроль		
12.	Перевірка на плагіат		
13.	Попередній захист кваліфікаційної роботи		
14.	Захист кваліфікаційної роботи		

Студент _____ Стрильбіцький О.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Литвиненко Я.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інтернет магазину продажу туристичного спорядження «Еверест» // Кваліфікаційна робота освітнього рівня «Бакалавр» // Стрильбіцький Олександр Вікторович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2024 // С. 58, рис. – 34, табл. – 0, кресл. – 0, додат. – 3, бібліогр. – 40.

Ключові слова: веб-сайт, туризм, товари, Еверест, магазин, JavaScript, ReactJS, ExpressJS.

Кваліфікаційна робота присвячена розробці інтернет магазину з продажу туристичних товарів. В першому розділі кваліфікаційної роботи описано основні поняття про веб сайти, їх структуру, призначення та спосіб створення.

В другому розділі кваліфікаційної роботи подано опис технологій, за допомогою яких відбувалася розробка.

В третьому розділі кваліфікаційної роботи проведено процес написання коду для клієнтської та серверної частини коду, а також підключення до баз даних.

В четвертому розділі кваліфікаційної роботи досліджено питання ергономічних проблем безпеки життєдіяльності та проведення інструктажів з охорони праці.

ANNOTATION

Development of an Online Store for Selling of Tourist Equipment «Everest» // Qualification work of the educational level «Bachelor» // Strylbitskyi Olexandr // Ternopil Ivan Puluj National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-42 // Ternopil, 2024 // P. 58, fig. – 34, tabl. – 0, chair. – 0, annexes. – 3, references – 40.

Keywords: website, tourism, products, Everest, shop, JavaScript, ReactJS, ExpressJS.

The qualification work is devoted to the development of an online store selling tourist products. The first section of the qualification paper describes the basic concepts of web sites, their structure, purpose and method of creation.

The second section of the qualification work describes the technologies used in the development.

In the third section of the qualification work, the process of writing the code for the client and server part of the code, as well as connecting to databases, was carried out.

In the fourth section of the qualification work, the issue of ergonomic problems of life safety and conducting occupational safety briefings is investigated.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) – це набір правил і протоколів, що дозволяють різним програмам взаємодіяти між собою. MIT (англ. Massachusetts Institute of Technology) – Массачусетський технологічний інститут.

IDE (англ. Integrated Development Environment) – це інтегроване середовище розробки, яке забезпечує розробників інструментами для написання, редагування, тестування та налагодження коду в одному додатку. IDE зазвичай включає текстовий редактор, компілятор або інтерпретатор, засоби для відладки та інші корисні інструменти.

Інтерфейс – це сукупність засобів і правил, що дозволяють взаємодіяти між собою різним системам, програмам або користувачам. Інтерфейс визначає способи обміну інформацією та керування функціями системи.

Роут (англ. Route) – це механізм, який дозволяє визначати, яка компонента повинна відображатися в інтерфейсі користувача в залежності від URL-адреси. Роутинг у ReactJS зазвичай реалізується за допомогою бібліотеки React Router.

Скрипт – це невелика програма або набір інструкцій, які автоматизують виконання задач у програмному середовищі.

Фреймворк – це структура або платформа для розробки програмного забезпечення, яка надає готові компоненти та інструменти для спрощення створення додатків.

ЗМІСТ

ВСТУП

Актуальність теми. В сучасному світі все більш стає популярним інтернет покупки. Люди прагнуть розумно розподіляти свій час, тому онлайн сервіси створені для економії часу та зусиль пошуку. Це особливо важливо для галузі туристичного спорядження, де покупці шукають зручність, широкий вибір товарів і доступність інформації про продукт.

Мета і задачі дослідження. Метою цієї кваліфікаційної роботи є створення веб сайту, який дозволятиме розміщувати товари туристичного напрямку, матиме простий дизайн та функціонал, буде зручним як для простого користувача, так і для адміністраторів даного магазину. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– проаналізувати поняття та застосування веб-сайтів;

– проаналізувати, який набір технологій буде потрібний для створення веб-сайту;

– розробити основні компоненти-сторінки сайту;

– розробити авторизацію на веб-сайті;

– розробити панель для адміністрації для додавання товарів.

Практичне значення одержаних результатів. Розроблений під час виконання поставлених задач веб-сайт для онлайн продажу товарів для туризму створить ефективний бізнес, який може стати як хороший додатком до існуючого оффлайн магазину, так і повноцінним самостійним магазином. Також він буде легкий у використанні, до нього можливо буде застосувати різноманітні маркетингові стратегії, такі як персоналізована реклама, електронні бюлетені, соціальні медіа та інше, щоб збільшити зацікавлену аудиторію.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Поняття, види та застосування веб-сайтів

Усі компанії, незалежно від їх галузі, повинні мати сильну присутність в Інтернеті, щоб розширюватися та виходити на ширший світовий ринок. Найефективніший спосіб досягти цього – створити веб-сайт і спрямувати на нього потенційних клієнтів.

Веб-сайт – це сукупність загальнодоступних взаємопов'язаних веб-сторінок, які мають одне доменне ім'я. Веб-сайти можуть створюватися та підтримуватися окремою особою, групою, підприємством або організацією для виконання різноманітних цілей. По суті, будь-хто, хто прагне створити онлайн-присутність, поділитися інформацією чи залучитися до спільноти, може отримати вигоду від наявності веб-сайту. У сучасну цифрову епоху простий статичний веб-сайт може виконувати роль візитної картки, а динамічний веб-сайт може замінити вітрину. Вони обидва дозволяють ділитися інформацією з глобальною аудиторією [1].

Перший веб-сайт був створений у 1991 році Тімом Бернерсом-Лі, британським фізиком із CERN. Сайт розміщувався на серверах CERN і містив інформацію про проект World Wide Web. Домен досі підтримується і користувачі можуть дослідити вигляд початку Інтернету. Вигляд сайту наведений на рисунку 1.1.

Рисунок 1.1 – Вигляд першого сайту

У 1993 році ЦЕРН оголосив, що кожен може отримати доступ до всесвітньої павутини та користуватися нею безкоштовно. В результаті кількість веб-сайтів швидко зросла. До кінця 1994 року налічувалося близько 3000 веб-сайтів. Сьогодні існує понад 1 мільярд веб-сайтів. З них, за оцінками, близько 2 мільйонів сайтів є активними. Вони розміщуються на серверах і для їх відвідування потрібен веб-браузер. Доступ до веб-сайту можна отримати безпосередньо, ввівши його URL-адресу або виконавши пошук у пошуковій системі [2].

При вводі веб-адреси або доменного ім’я у браузер, комп’ютер надсилає запит на сервер хостингу. Цей запит проходить через систему доменних імен (DNS)адреси Інтернет-протоколу (IP) щоб знайти сервер. Веб-служба використовує такі протоколи, як Hypertext Transfer Protocol (HTTP), Hypertext Transfer Protocol Secure (HTTPS) і File Transfer Protocol (FTP) для передачі інформації та файлів через Інтернет-протоколу (IP).

Отримавши запит, сервер надсилає запитану веб-сторінку разом із зображеннями та іншими файлами на комп’ютер. Час завантаження веб-сайту залежить від ряду факторів, таких як швидкість сервера, якість інтернет-з’єднання, розмірі складність веб-сайту [3]. Принцип роботи зображений на рисунку 1.2.

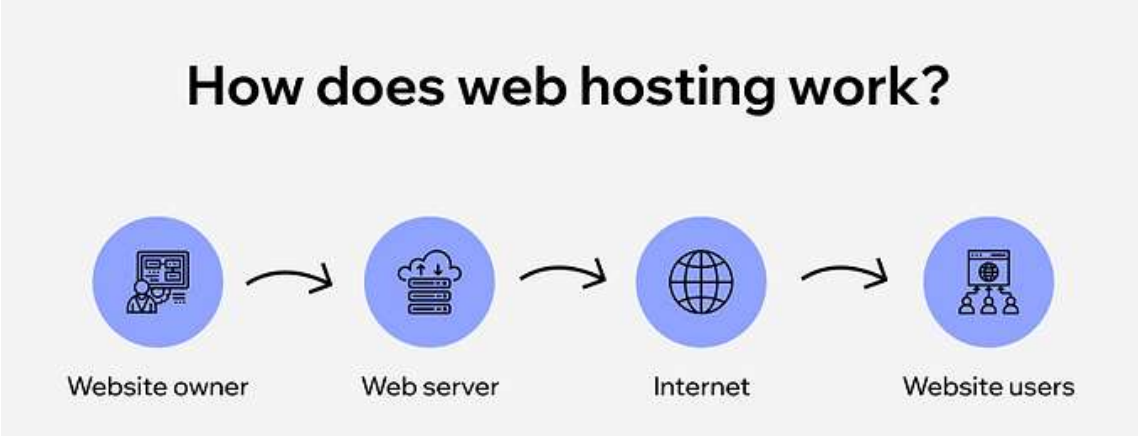

Рисунок 1.2 – Схема запиту до веб-хостингу

Головне призначення сайту – це надання інформації. У зв’язку з стрімкою швидкістю розвитку мережі Інтернет, відбувається втягування все більшої кількості людей в сферу використання Інтернет-технологій та ресурсів.

Різні види веб-сайтів мають різні цілі залежно від цільової аудиторії. Деякі веб-сайти орієнтовані на продаж продуктів, інші – на надання практичної інформації, а інші – лише для розваги. Найбільш поширені серед них є інформаційні, розважальні, комерційні, блоги та соціальні мережі.

Мета веб-сайту, орієнтованого на інформацію, полягає в тому, щоб передати конкретну, корисну інформацію конкретному користувачу/аудиторії, щоб читач дізнався щось нове або краще зрозумів тему. Ці веб-сайти орієнтовані на більш практичну інформацію та можуть містити інструкції, поради та підказки, виправлення та ремонт, керівництво, інформацію про підтримку, вказівки, інструкції тощо. Приклад інформаційного сайту наведений на рисунку 1.3.

Рисунок 1.3 – Приклад інформаційного сайту

Розважальні веб-сайти демонструють цікаву інформацію для відвідувачів. Інтернет-журнали, веб-сайти, орієнтовані на плітки, новини про знаменитостей, спортивні події, фільми, ігри, мистецтво, гумористичні веб-сайти тощо. Вони створені для зручності навігації та часто оновлюються, щоб користувачі

поверталися за додатковою інформацією. Їх можна зробити більш привабливими, використовуючи динамічний вміст, такий як відео, подкасти, слайд-шоу тощо. На рисунку 1.4 зображено приклад розважального сайту.

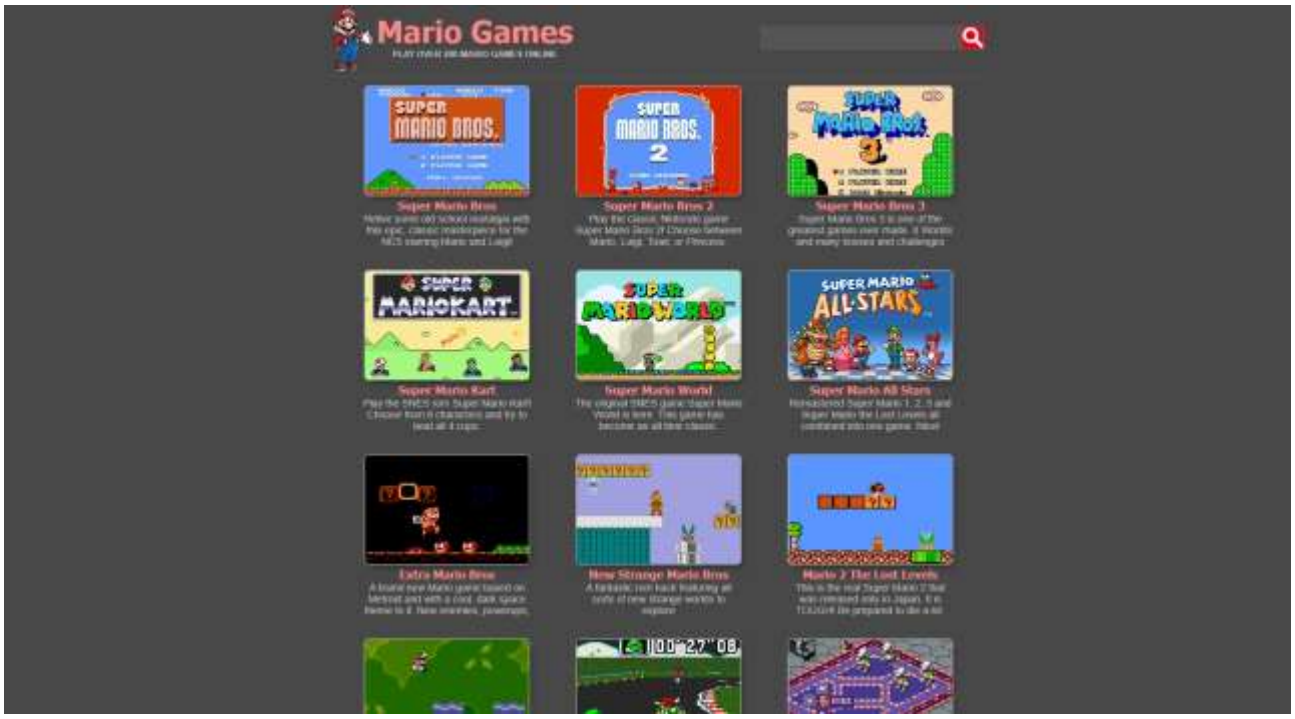

Рисунок 1.4 – Приклад розважального сайту

Веб-сайти електронної комерції орієнтовані на продаж товарів користувачам. Найбільш успішні сайти ретельно налаштовані для максимізації відсотка покупок. Для досягнення високої ефективності, ці сайти повинні впроваджувати новітні методики онлайн-продажів, які, як доведено, підвищують ймовірність покупки відвідувачами. Є багато важливих елементів, які входять до створення успішного веб-сайту електронної комерції, наприклад усунення затримок під час процесу купівлі, плавність і легкість оформлення замовлення, створення веб-сайту швидким і привабливим, підвищення продажів користувачам пов’язаних продуктів, спонукання колишніх покупців до повторних покупок, ремаркетинг для відвідувачів, які ще не зробили покупки, використання належних варіантів оплати, наявність мобільного дизайну тощо. На рисунку 1.5 зображено приклад комерційного сайту.

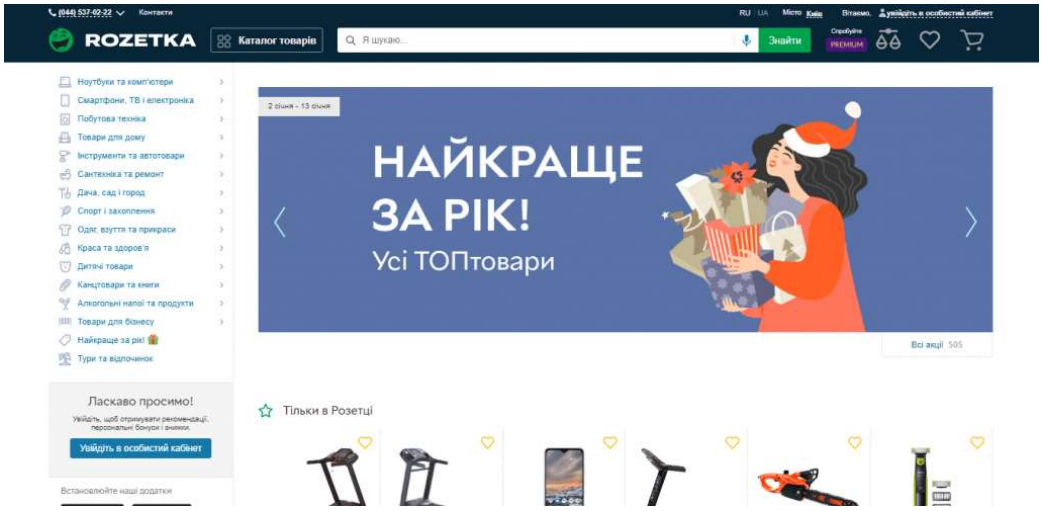

Рисунок 1.5 – Приклад комерційного сайту

Мета блогу – створити регулярно оновлюваний веб-сайт або веб-сторінку, яка зазвичай ведеться окремою особою чи невеликою групою, написана в неформальному або розмовному стилі. Блоги можна дуже легко створити в Інтернеті за допомогою низки безкоштовних сервісів, таких як Wordpress. Є багато особистих і професійних блогів, які цікаво читати і які дають дуже особисте розуміння життя людини. На рисунку 1.6 зображено приклад блогу.

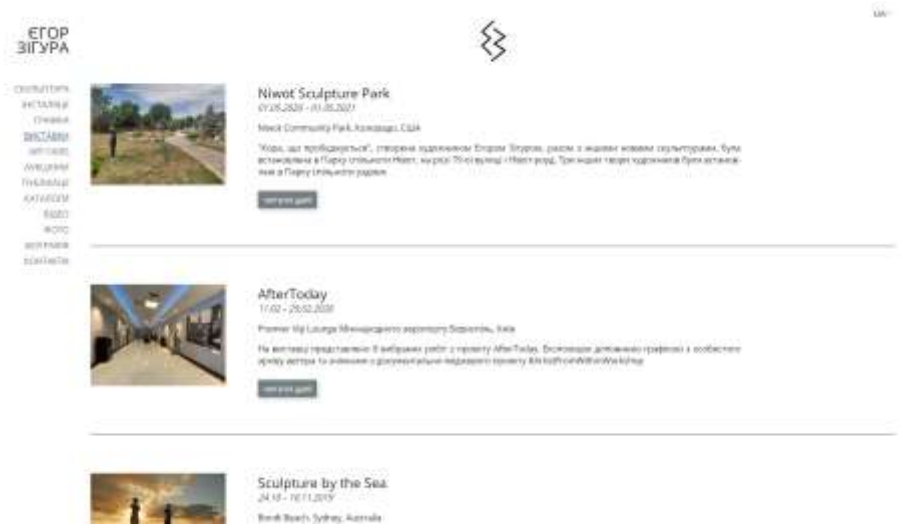

Рисунок 1.6 – Приклад блогу

Блоги можуть бути розважальними, використовуватися як онлайн-журнал або використовуватися компаніями, щоб тримати своїх клієнтів у курсі подій.

Відмінною рисою блогу є те, що його дуже легко вести непідготовленій людині, яка має незначні технічні знання або взагалі не володіє ними.

Мета веб-сайтів соціальних мереж полягає в тому, щоб полегшити обмін інформацією та спілкування з друзями, родиною, колегами, знайомими та навіть незнайомими людьми. Веб-сайти соціальних мереж дозволяють швидко й легко створити мережу зв’язків, щоб підтримувати зв’язок, ділитися щоденним досвідом, фотографіями, інтересами, уподобаннями тощо. На рисунку 1.7 зображено приклад соціальної мережі.

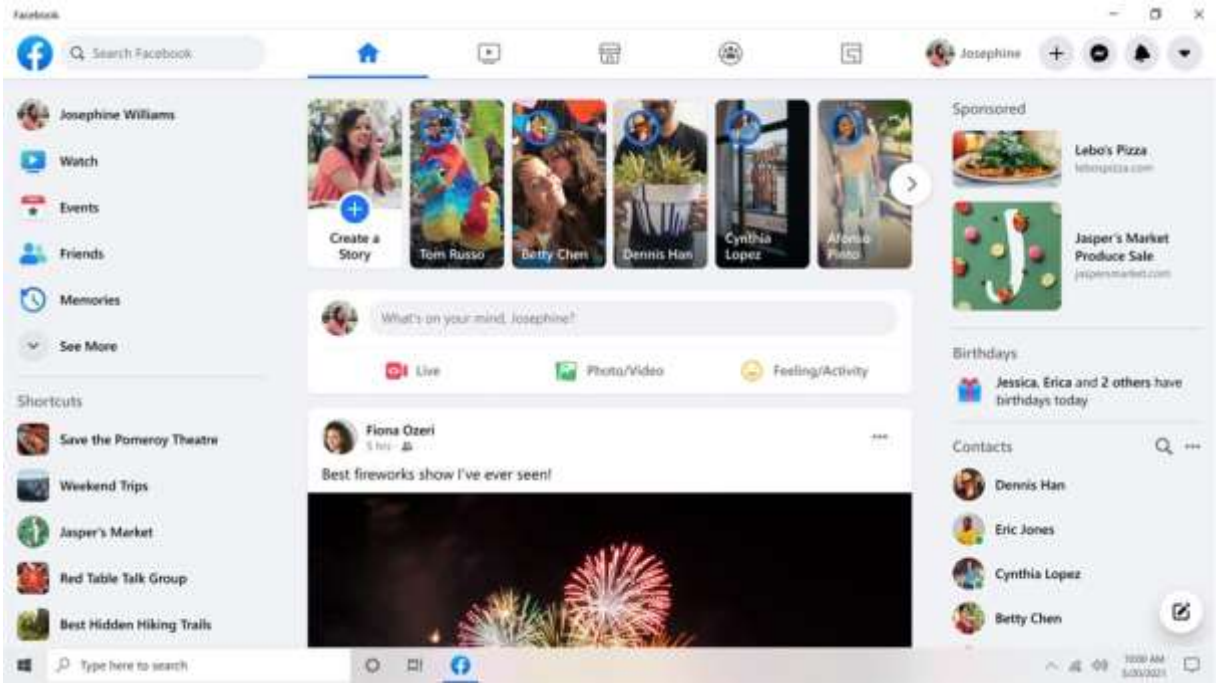

Рисунок 1.7 – Приклад блогу

Соціальні мережі можна використовувати як для особистих, так і для комерційних цілей. Підприємства використовують соціальні мережі, щоб налагоджувати прямі зв’язки зі своїми клієнтами, що дозволяє їм отримувати відгуки про їхні продукти та послуги та дозволяє їм більше дізнатися про те, чого їхні клієнти дійсно потребують і чого хочуть [4].

1.2 Структура створення веб-сайту

Структура сайту складається з 2 основних частин: frontend та back-end. Інтерфейсний розробник відповідає за програмне забезпечення, яке спеціалізується на створенні та розробці інтерфейсу користувача (UI) та взаємодії з користувачем (UX) веб-сайтів і веб-додатків. Основний обов'язок полягає в тому, щоб візуальні та інтерактивні аспекти веб-сайту або програми були зручними, естетично привабливими та функціонально ефективними.

Інтерфейсні розробники працюють із різними технологіями, інструментами та мовами, зокрема:

– HTML (HyperText Markup Language): стандартна мова розмітки, яка використовується для створення структури та макета веб-сторінок.

– CSS (каскадні таблиці стилів): мова таблиці стилів, яка використовується для керування представленням, форматуванням і зовнішнім виглядом веб-сторінок, наприклад кольорами, шрифтами та макетом.

– JavaScript - це мова програмування, яка дає змогу розробникам створювати інтерактивні елементи, анімації та інші динамічні компоненти для веб-сайтів і веб-додатків.

Для оптимізації своєї роботи та розробки складних і інтерактивних користувацьких інтерфейсів фронтенд-розробники часто використовують бібліотеки та фреймворки, такі як React, Angular або Vue.js. Також вони активно співпрацюють із бекенд-розробниками, які займаються серверною логікою та управлінням даними, щоб забезпечити безперебійну інтеграцію між клієнтськими та серверними частинами веб-додатків чи сайтів. Фронтенд розробка потребує технічних знань і креативності, щоб гарантувати, що користувацькі інтерфейси веб-сайтів виглядатимуть правильно та функціонуватимуть належним чином. Фронтальні розробники співпрацюють із бекенд-розробниками, дизайнерами та аналітиками взаємодії з користувачами [5].

Бекенд-веб-розробка – це створення основи веб-сайтів і програм шляхом написання коду, який об'єднує все – сервери, бази даних та інші технології.

Розробка бекенда передбачає створення API, які дозволяють різним програмним компонентам спілкуватися один з одним. Приклад принципу взаємодії зображений на рисунку 1.8.

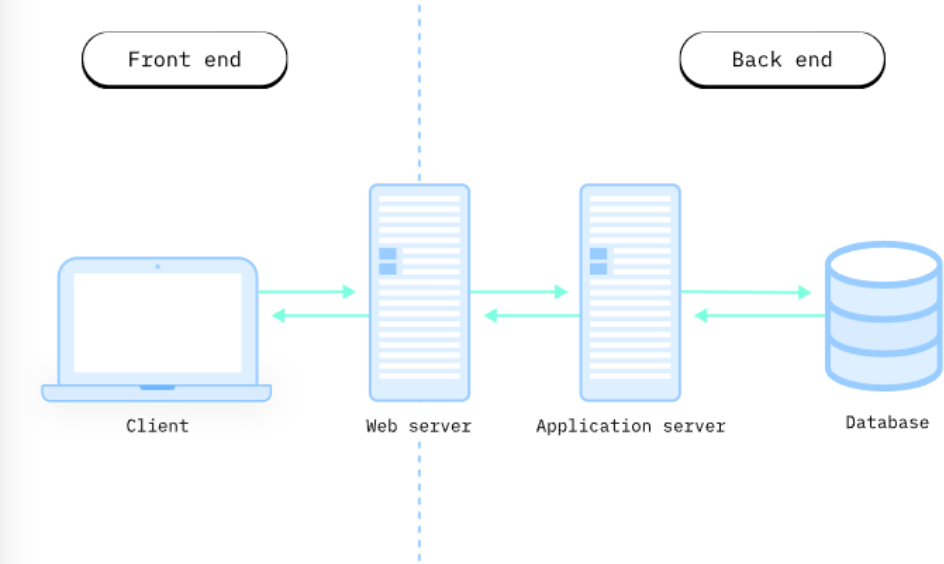

Рисунок 1.8 – Схема взаємодії баз даних з сервером

Розробники бекенду відповідають за те, щоб усе працювало за лаштунками веб-сайту чи програми. Вони використовують свої технічні знання та навички вирішення проблем для створення систем, які обробляють дані та виконують різні функції. Їм також потрібно добре співпрацювати з іншими розробниками, щоб створити загальну структуру веб-сайту чи програми.

Основна відповідальність розробника серверної частини полягає в тому, щоб зрозуміти, що необхідно для роботи веб-програми, і написати код, який робить ці програми функціональними та логічними. Веб-програми стосуються всього, що використовує веб-браузери та веб-технології; таким чином, він включає не лише веб-сайти, а й соціальні мережі та програми. Технологія, яка використовується для роботи з ними, складається з комбінації серверів, баз даних і програм; таким чином, робота бекенд-розробника ніколи не є простою та прямою для розуміння користувачів, тому ця професія широко відома як робота «за лаштунками» [6].

Роль бекенд-розробника в команді розробників є вирішальною. Бекендер повинен тісно співпрацювати з інтерфейсом, щоб гарантувати, що

функціональність узгоджується з користувацьким досвідом. Функціональність є життєво важливою характеристикою будь-якого веб-сайту. Крім того, важливо робити постійні оновлення, а також контролювати та організовувати інформацію бази даних безпечним і правильним способом [7].

Такі бази даних, як PostgreSQL, MySQL**,** MongoDB і Oracle**,** чудово підходять для пошуку, зберігання та зміни даних. Крім того, такі мови сценаріїв, як Ruby і Python, а також скомпільовані мови, такі як C# і Java, є перевагами при кодуванні на стороні сервера. Хороший розробник добре володіє принаймні однією з цих мов і добре розуміє ще одну або дві мови. Крім того, знання архітектур веб-розробки (таких як модель-представлення-контролер, шаблон шини подій та шаблон інтерпретатора) може допомогти розробити найбільш підходящу систему архітектури відповідно до атрибутів якості до початку значної розробки програмного забезпечення [8]. Щодо технічних знань, це також може допомогти зрозуміти інструменти зовнішньої технології, такі як JavaScript, HTML і CSS. Робота з цими інструментами полегшує спілкування між розробниками та забезпечує узгодженість обох членів команди [9].

1.3 Аналіз та постановка завдання

Перед безпосереднім створенням сайту, потрібно створити план дій. Необхідно дізнатись тематику, ціль сайту. Кожен сайт складається з багатьох компонентів, які взаємодіють між собою. Для початку потрібно розглянути які є можливості та пропозиції у виборі середовища програмування.

Веб-розробка розвивалася дуже швидкими темпами, і пам’ятаючи про це, звернення уваги на IDE (інтегроване середовище розробки) є важливим аспектом, оскільки воно допомагає вам підвищити свої навички на наступний рівень. IDE допомагають програмістам реалізувати свої думки та побачити роботу веб-сайтів. Його основні функції включають код, редагування, тестування та налагодження. Крім того, розширені IDE пропонують багато

функцій, які можна використовувати для автоматизації, візуалізації та налаштування. Розглянемо найпопулярніші IDE [10].

VS Code є найбільш використовуваним і популярним IDE для веб-розробників, кожен веб-розробник зіштовхнувся б з ним. Його розроблено компанією Windows і містить інструменти та розширення для багатьох мов (C#, C++, PHP і Python) і підтримує 36 різних мов програмування. Він підтримує Windows, Mac і Linux, усі операційні системи. У нього зразу після установки присутні JavaScript, TypeScript і NodeJS. Найкраще те, що він підтримує IntelliSense, який надає інтелектуальні рекомендації щодо коду за допомогою інструментів інтелектуального завершення коду, щоб допомогти розробникам швидко завершити код. Створення середовищ розробки в хмарі під час роботи в Windows, Mac, Android, iOS, Інтернеті та в хмарі. Додавання плагінів дозволяє розширити його функціональність [11]. Основні функції Visual Studio:

– інтегрований GitHub;
– засоби автоматизованого тестування;
– підсвічування синтаксису;
– імпортовані модулі та готові шаблони;
– створення програмних компонентів і керування ними.

Будучи однією з найкращих IDE для Java, IntelliJ IDEA також можна використовувати для HTML, JavaScript, SQL та інших мов. Він має можливість автоматично додавати зручні інструменти, що відповідають контексту. IntelliJ IDEA має інтелектуальну підтримку кодування та може виконувати аналіз потоку даних. Індексація коду для надання відповідних пропозицій і доповнення коду. Щоб уникнути повторюваних заяв, він автоматизує кілька завдань. Головний мотив – підвищення продуктивності розробників. Він поставляється з вбудованими інструментами для розробників і величезною підтримкою спільноти, яка допоможе вам, якщо виникли проблеми [12].

Основні особливості IntelliJ IDEA:

– виявлення повтору фрагментів коду;
– підтримка Google App Engine, Grails, GWT;
– інструменти розгортання та налагодження.

Webstorm – це популярна IDE для веб-розробників, яка сумісна з Windows, Mac і Linux. Він підтримує такі мови, як HTML, JavaScript, Node.js, Angular, TypeScript, CSS, React тощо. Це найрозумніша IDE JavaScript, що робить її найкращою для веб-розробки. Він також має чудові функції завершення коду та рефакторинг для популярних фреймворків. Він виявляє помилки в коді за допомогою функції аналізу якості коду. Можлива інтеграція WebStorm з такими лінтерами, як Stylelint і ESLint. Крім того, він має вбудований HTTP-клієнт у редакторі для редагування, створення та запуску HTTP-запитів [13]. Основні особливості WebStorm:

– велика підтримка плагінів;
– правильна навігація;
– вбудований відладчик;
– потужний і настроюваний;
– автозаповнення коду та найкраща компіляція коду.

Наступним кроком є вибір серед баз даних. База даних є обов'язковою для веб-додатків для належного зберігання, обробки та керування даними. Це допомагає компонентам програми ефективно отримувати правильні дані, обробляти їх і надавати точні результати кінцевим користувачам. Крім того, це також допомагає підвищити продуктивність, швидкість, безпеку та масштабованість програмного забезпечення. Але все це можливо тільки в тому випадку, якщо обрано надійну та оновлену базу даних.

MySQL є однією з найвживаніших і перевірених часом баз даних для веб-додатків. Кожен розробник веб-додатків знає конфігурацію MySQL або має повні знання щодо її налаштування.

Для бази даних MySQL використовується мова структурованих запитів, яка дозволяє підключати її до будь-якої веб- та мобільної програми. Крім того, ви можете використовувати його для надання послуг PaaS (платформа як послуга), SaaS (програмне забезпечення як послуга) і DBaaS (база даних як послуга) своїм клієнтам і зацікавленим сторонам [14].

MySQL пропонує перевагу сумісності між платформами, що дозволяє працювати на машинах Windows, Linux і macOS. Основні функцій включають:

– MySQL надає функціональність ACID (атомність, узгодженість, ізоляція, довговічність).

– Оскільки це реляційна база даних, продуктивність і швидкість завантаження високі в будь-якій ситуації.

– Його можна використовувати з будь-якою технологією розробки веб-додатків, як-от.NET, PHP, JavaScript тощо.

– Він доступний у різних версіях відповідно до ваших вимог, наприклад MySQL Heatwave, який надає повністю керовані послуги з аналізом у реальному часі.

– Це забезпечує безпеку даних, а також збереження їх цілісності та конфіденційності.

База даних Oracle досить популярна серед підприємств і державних установ завдяки своїй надзвичайній масштабованості, легкому управлінню та надійній безпеці.

Oracle пропонує функціональні можливості СУБД із багатомодельними зв’язками, використовуючи вдосконалення кількох робочих навантажень. Крім того, він покращує обробку даних завдяки інтеграції AutoML, що також допомагає краще приймати рішення [15]. Крім того, його реальні кластери додатків є додатковою перевагою для користувачів, оскільки вони забезпечують:

– просте масштабування бази даних між примірниками;

– ефективне балансування навантаження;

– максимальна доступність даних, резервування та гнучкість для покращення обробки;

– надання вбудованого менеджера відновлення (RMAN), який допомагає відновити дані про збої;

– дозвіл користувачеві виконувати процедурне програмування, встановлюючи розширення PL/SQL;

– налаштування кілька екземплярів Oracle на одному сервері;

– використання мінімальних ресурсів та допомога заощадження коштів.

PostgreSQL або Postgres – це відома база даних у сфері розробки веб-додатків. Вона в основному розширює мову структурованих запитів (SQL), але

більш ефективно. Крім того, він вважається абсолютно безпечним і забезпечує високу ймовірність збереження цілісності даних.

Крім того, це також підтримує створення унікальних функцій та типів даних відповідно до вимог розробника. Таким чином, логіка програми працюватиме безперебійно, а адміністраторам буде легко розуміти роботу програмного забезпечення та керувати ним [16].

База даних Postgres також пропонує такі переваги:

– падання вбудованих функцій аварійного відновлення, наприклад PITR (відновлення на певний момент часу);

– підтримка доступних механізмів автентифікації, включаючи LDAP, SSPI, SCRAM-SHA-256, GSSAPI тощо;

– підтримка міжнародних символів та забезпечення повнотекстового пошуку.

Отже, проаналізувавши пропозиції ринку, було обрано для використання фреймворки ReactJS і Express, мова програмування JavaScript, з використанням HTML, CSS та Bootstrap. Зберіганням даних буде займатися БД PostgreSQL. Серверна частина також буде написана на Javascript, а саме NodeJS.

1.4 Висновок до першого розділу

У першому розділі кваліфікаційної роботи було розглянуто загальне поняття веб-сайтів, типи та їх застосування, демонстрацію структури, методи створення. Було проаналізовано основні популярні інтегровані середовища програмування та бази даних. Для розробки було обрано мову написання JavaScript (з використанням HTML/CSS), фреймворки ReactJS та Express, базу даних PostgreSQL. Розробка відбуватиметься у середовищі WebStorm.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА ОПИС ТЕХНОЛОГІЙ

2.1 Опис технологій клієнтської частини коду

Без JavaScript складно уявити веб-розробку. Це динамічна, об’єктно-орієнтована мова, яка використовується для реалізації створення інтерактивних веб-сторінок, керування мультимедією, анімування графіків. JavaScript є невід'ємною частиною сучасної веб-розробки. Він використовується для додавання інтерактивності на веб-сторінки, маніпуляції DOM (Document Object Model), обробки подій та багато іншого.

Основним фреймворком, який використовуватиметься буде ReactJS. ReactJS – це популярна бібліотека JavaScript, яка використовується для створення користувацьких інтерфейсів, особливо односторінкових додатків. Вони містять як і частини інтерфейсу, так і функціональність. Приклад компонент зображений на рисунку 2.1.

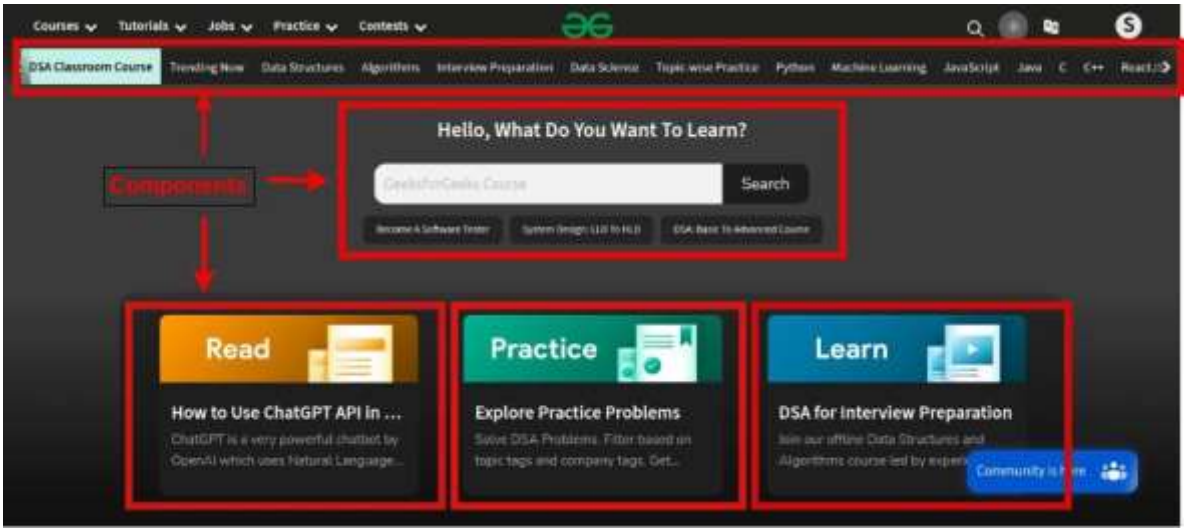

Рисунок 2.1 – Приклад компонент

Основна ідея React полягає в компонентному підході та використанні віртуального DOM, що дозволяє створювати динамічні та високопродуктивні додатки. Компоненти є сталим шаблоном, який може застосовуватися безліч раз у коді.

Вся суть в тому, щоб поділити великий код на маленькі універсальні частинки, що дозволяє зробити код легким для читання, а середовище розробки незабрудненим. Цей метод дуже схожий з функціями в JavaScript, у результаті виклику компоненти вона повертає HTML-елементи. React підтримує застосовує JSX – розширення, яке дає змогу створювати компоненти з стилізацією та логікою. Основними видами є функціональні (приймають параметр, а повертають елемент), і класові (використовують свій синтаксис і застосовувалися до появи хуків) [17].

Іншим основним елементом React є хуки. Хуки (hooks) в React – це спеціальні функції, які надають змогу використовувати можливості React, такі як стан і методи життєвого циклу, у функціональних компонентах [18].

Також для розробки потрібно буде інсталювати декілька розширень у проект. У Webstorm це робиться за допомогою команди install у терміналі. При встановленні важливо обрати правильну директиву [19]. Вигляд терміналу зображений на рисунку 2.2.

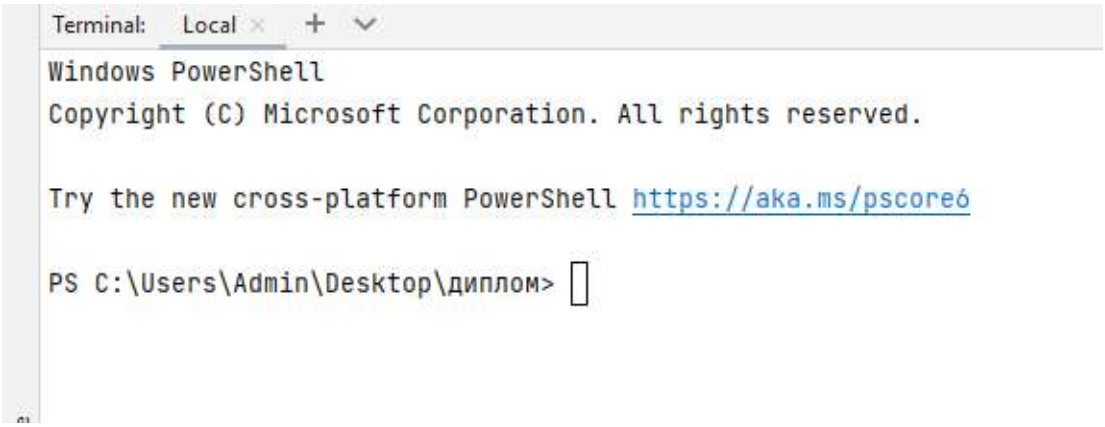

Рисунок 2.2 – Вигляд терміналу

Для стилізації було обрано React Bootstrap. Bootstrap – це бібліотека готових шаблонів-елементів. У ній можна знайти різноманітні кнопки, слайдери, фігури та списки, приклад шаблону зображений на рисунку 2.3.

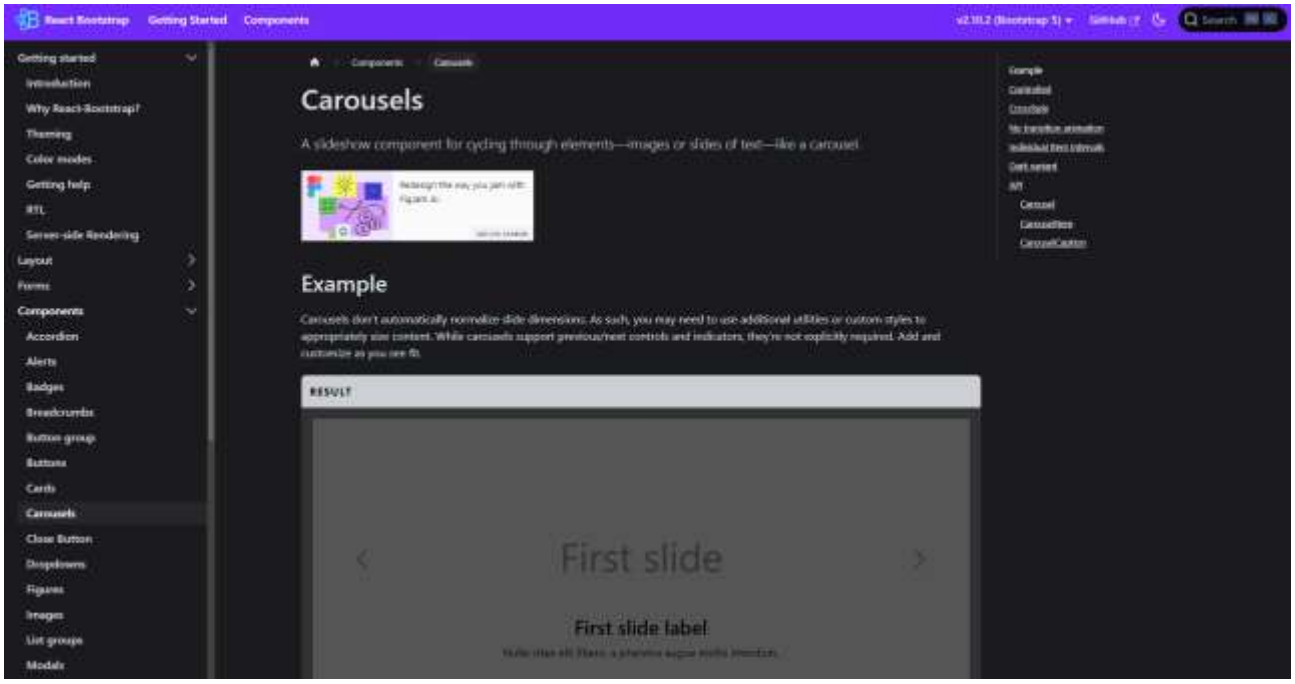

Рисунок 2.3 – Бібліотека Bootstrap

Bootstrap дозволяє значно зменшити час на розробку та має зручну документацію, в якій можливо змінити все на свій вибір [20]. Він включає в себе всі основні компоненти такі як:

– компоненти макета: <Container>, <Row>, <Col> – забезпечують створення адаптивного макета з використанням системи сітки Bootstrap;

– компоненти форми: <Form>, <FormControl>, <FormGroup>, <Button> – допомагають створювати стильні та функціональні форми;

– компоненти навігації: <Navbar>, <Nav>, <NavItem> – дозволяють створювати навігаційні панелі та меню;

– компоненти сповіщень та модальних вікон: <Alert>, <Modal> – забезпечують прості способи показу сповіщень і діалогових вікон;

– інші компоненти: <Card>, <Carousel>, <Tooltip>, <Popover> – забезпечують додаткові елементи інтерфейсу.

React Router DOM – це бібліотека для керування маршрутизацією у додатках на основі React. Вона дозволяє створювати навігацію між сторінками, яка не вимагає перезавантаження браузера. Динамічна маршрутизація дозволяє змінювати контент без оновлень, що є швидше порівняно з традиційною

навігацією по сторінках. Це свідчить про те, що користувальницький досвід є кращим, а додаток має загальну кращу продуктивність.

Це одне з основних розширень, воно відповідає за маршрути (роути), які будуть відображатися, коли користувач заходитиме на певну адресу. Приклад такої структури зображений на рисунку 2.4. Також воно добавляє тег Switch та хук useNavigate, які значно спрощують навігацію. За допомогою тега Redirect користувач перенаправляється з одного шляху на інший [21]. React Router DOM добавляє свої компоненти, а саме:

– BrowserRouter: використовується для обгортання вашого додатку, надаючи контекст маршрутизації;

– Routes і Route: визначають маршрути та компоненти, які потрібно відображати;

– Link: створює посилання для навігації без перезавантаження сторінки;

– NavLink: створює навігаційні посилання з можливістю додавання активного класу.

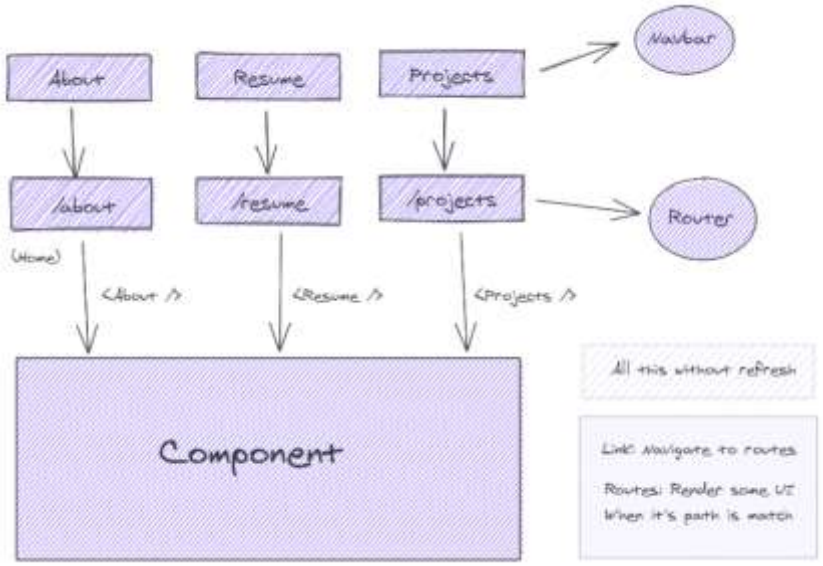

Рисунок 2.4 – Структура маршрутів

MobX – це бібліотека для керування станом, яка спрощує цей процес і робить його масштабованим завдяки прозорому використанню функціонального реактивного програмування. Вона автоматично відстежує всі зміни та

використання даних під час виконання програми, створюючи дерево залежностей, яке фіксує взаємозв'язки між станом і результатами. Це дозволяє обчисленням, залежним від стану, наприклад, компонентам React, виконуватися тільки тоді, коли це дійсно необхідно. В результаті, немає потреби в ручній оптимізації компонентів за допомогою складних і часто помилкових методів, таких як меморізація та селектори [22]. На рисунку 2.5 показано приклад роботи MobX.

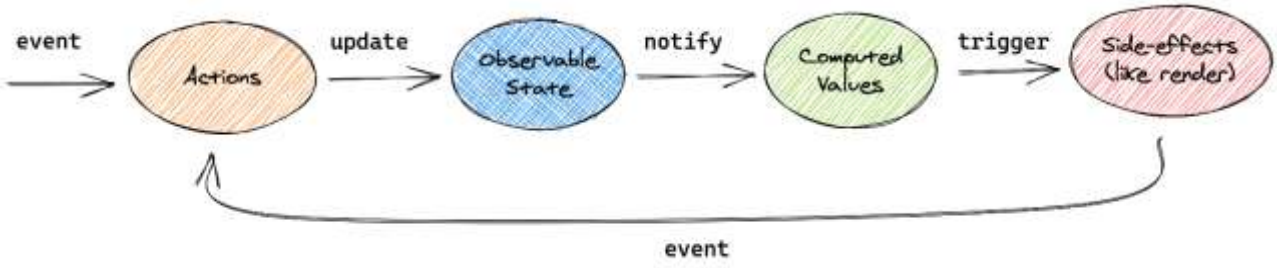

Рисунок 2.5 – Схема роботи MobX

CORS або Cross-Origin Resource Sharing – це механізм, який дозволяє запитувати багато ресурсів (наприклад, шрифти, JavaScript тощо) на веб-сторінці з іншого домену за межами домену, з якого походить ресурс. Це технологія, яка дозволяє веб-програмі звільнитися від обмежень політики однакового джерела, яка є заходом безпеки, який запобігає взаємодії ресурсів із різних джерел один з одним.

У контексті програми React CORS вступає в дію, коли програмі потрібно запитувати ресурси з іншого джерела. Наприклад програма React працює на localhost:3000 і їй потрібно отримати дані з API, що працює на localhost:5000. Ці два мають різне походження, оскільки вони працюють на різних портах. Без CORS додаток React не зможе отримати ці дані через політику того самого походження. Але якщо CORS увімкнено на стороні сервера, додаток може вільно запитувати ресурси з цього API. Сервер включає певні HTTP-заголовки (наприклад, Access-Control-Allow-Origin) у свої відповіді, щоб повідомити браузеру, що це нормально дозволяти запити з різних джерел [23].

Dotenv – популярний модуль для керування змінними середовища, який завантажує конфігурацію з.env файлу в process.env. Хоча Dotenv є поширеним вибором, існує також багато альтернатив, і конкретне рішення може залежати від мови та вимог проекту. Мета однакова: відокремити конфіденційну конфігураційну інформацію від коду та безпечно зберігати її в середовищі [24].

Nodemon скорочено від «Node Monitor». Його головна роль – спостерігати за файлами в проекті Node.js і автономно перезапускати сервер після виявлення змін. Ця функція значно допомагає у швидкій розробці програм Node.js. Коли виявляється зміна, він витончено обробляє перезапуск сервера. Цей спрощений процес прискорює цикл розробки, дозволяючи розробникам зосередитися на написанні коду та тестуванні без повторних переривань ручного перезапуску сервера. Це замінна оболонка для вузла. Для використання nodemon не потрібні додаткові зміни у коді [25].

2.2 Опис технологій серверної частини коду

Node.js – це середовище виконання, яке дозволяє виконувати JavaScript на стороні сервера. Він створений на основі двигуна V8 JavaScript від Chrome, який компілює JavaScript у ефективний машинний код. Node.js працює на основі однопотокової архітектури, керованої подіями, використовуючи цикл подій для обробки кількох одночасних операцій без блокування. Принцип роботи зображений на рисунку 2.6.

Коли клієнт надсилає запит на сервер Node.js, запит додається до черги подій. Цикл подій постійно перевіряє цю чергу та обробляє кожен запит. Якщо запит включає операцію введення-виведення, Node.js передає його в ядро системи, яке обробляє його асинхронно. Після завершення операції введення-виведення ядро сповіщає Node.js, виконуючи відповідну функцію зворотного виклику [26].

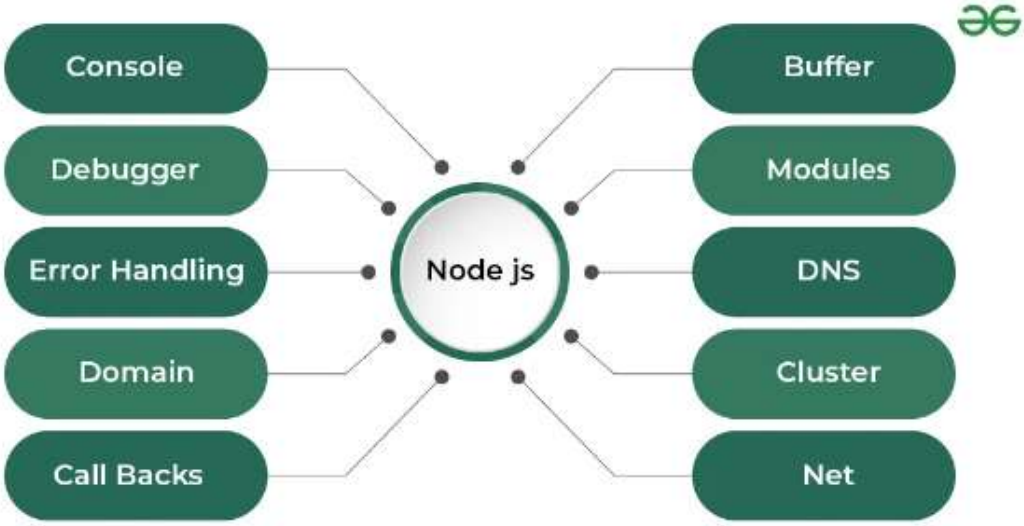

Рисунок 2.6 – Принцип роботи Node.JS

Express.js є провідним фреймворком для Node.js, призначеним для розробки веб-додатків та API. Його вважають стандартним серверним фреймворком для Node.js. Створення бекенду з нуля для додатка на Node.js може бути вкрай виснажливим і вимагати значних часових витрат. Від конфігурації портів до маршрутизації, написання всіх шаблонних кодів відволікає від основної мети – розробки бізнес-логіки додатка. У традиційних веб-додатках веб-сервер чекатиме на надсилання запитів HTTP від клієнта. Отримавши HTTP-запит, сервер вибере відповідний обробник маршруту та делегує йому подальші дії для цього запиту. Зазвичай написання обробника маршруту з нуля може бути трохи обтяжливим у Node. Express надає методи для визначення того, яка функція викликається для конкретного HTTP-дієслова (GET, POST, PUT тощо) і шаблону URL-адреси (маршрут).

Express – це вільна структура, яка дає розробникам свободу структурувати свій код на свій розсуд, а не нав’язує певну структуру коду. Одне з місць, де можна помітити цю позицію, – це застосування проміжного ПЗ. Проміжне програмне забезпечення дозволяє виконувати операції над запитами та відповідями, що переміщуються через маршрути, і широко використовується в програмах Express. Проміжне програмне забезпечення можна застосовувати як на рівні програми, так і на рівні маршруту, а також об’єднувати разом.

Express, будучи частиною стеку MEAN (MongoDB, Express.js, Angular.js, Node.js), є ключовим інструментом у сучасній розробці повного стеку JavaScript. Node.js – це середовище виконання, яке дозволяє запускати JavaScript на стороні сервера. Він не надає жодних спеціальних інструментів для створення веб-додатків. Ось тут і з'являється Express.js. Він забезпечує рівень абстракції над Node.js, спрощуючи складні завдання програмування на стороні сервера, такі як маршрутизація, керування сеансами та обробка запитів HTTP запити/відповіді. Він також добре інтегрується з іншими популярними бібліотеками Node.js і проміжним програмним забезпеченням, що робить його популярним вибором для створення веб-додатків.

Express.js спрощує процес створення веб-додатків, надаючи простий і гнучкий API для розробки цих додатків і API. Він призначений для керування основними функціями веб-сервера, дозволяючи розробникам зосередитися на унікальних функціях своїх програм. Express.js забезпечує високий ступінь модульності, дозволяючи розробникам вибирати з широкого спектру модулів Node.js для різних функціональних можливостей, таких як інтеграція бази даних, механізми шаблонів і керування сеансами [27].

Postman – це платформа розробки та тестування API, яка пропонує різноманітні функції. Це дозволяє розробникам легко створювати та ділитися запитами та колекціями API, автоматизувати тестування, імітувати API та контролювати продуктивність. Postman також надає інструменти для співпраці та документування, що дозволяє командам ефективніше працювати разом і ефективно передавати інформацію про поведінку API. Postman надає потужний візуальний редактор, який допоможе вам легко та ефективно розробляти API. Завдяки інтуїтивно зрозумілому інтерфейсу користувача можна створювати кінцеві точки API та встановлювати їхні параметри, навантаження запитів і відповідей, а також налаштування автентифікації. Візуальний редактор також дозволяє документувати ваш API за допомогою чітких і лаконічних описів, що полегшує іншим розробникам розуміння та використання вашого API [28].

Автентифікація JSON Web Token (JWT) – це метод без збереження стану безпечної передачі інформації між сторонами як об'єкт JavaScript Object Notation

(JSON). Він часто використовується для автентифікації та авторизації користувачів у веб-додатках та API.

У світі автентифікації «без стану» означає механізм, у якому сервер не підтримує жодного стану сеансу між запитами. У системі автентифікації без стану кожен запит є самостійним і містить усю необхідну інформацію для автентифікації та авторизації користувача або організації. У випадку автентифікації JWT це відбувається у формі маркера [29].

Маркер JSON складається з трьох частин:

– заголовок, що містить інформацію про тип маркера та алгоритми, які використовуються для створення підпису;

– корисне навантаження, що містить «заяви» (ідентифікатор та перевірки автентифікації), зроблені користувачем, які можуть включати ідентифікатор користувача, ім’я користувача, адресу електронної пошти та метаінформацію про роботу маркера;

– для перевірки цілісності токена використовується підпис, або криптографічний механізм.

2.3 Опис технології бази даних

PostgreSQL – це об’єктно-реляційна система керування базами даних із відкритим кодом, відома своєю надійністю, цілісністю даних і широким набором функцій. Він може обробляти розширені типи даних, складні запити, зовнішні ключі, тригери та представлення, а також процедурні мови для збережених процедур. PostgreSQL має широкі можливості розширення, що дозволяє користувачам додавати нові функції, типи даних та інші функції. Його сувора відповідність стандартам SQL у поєднанні з підтримкою властивостей ACID (атомарність, узгодженість, ізоляція, довговічність) робить його ідеальним вибором для розробників і підприємств, яким потрібна масштабована, ефективна та безпечна база даних.

Вибір PostgreSQL як рішення для бази даних пропонує унікальне поєднання переваг, які задовольняють широкий спектр потреб управління

даними. Однією з головних причин розгляду PostgreSQL є його виняткова підтримка розширених типів даних і складної функціональності. Це включає власну підтримку JSON, геометричних даних і користувацьких типів, що забезпечує гнучке й ефективне зберігання та пошук даних, які можна адаптувати до найскладніших і різноманітних моделей даних. Здатність PostgreSQL легко обробляти складні запити, транзакції та обширні операції зі сховищами даних робить його ідеальним для програм, які вимагають детальної аналітики, обробки в реальному часі та високої цілісності даних.

Ще однією причиною використання PostgreSQL є його сильний акцент на розширюваності та відповідності стандартам. Він розроблений таким чином, щоб бути дуже гнучким, що дозволяє користувачам розширювати базу даних власними функціями, операторами та типами даних. У поєднанні з відкритим вихідним кодом PostgreSQL може розвиватися відповідно до потреб проекту, сприяючи інноваціям і гарантуючи, що система баз даних залишатиметься перспективною.

PostgreSQL надає численні переваги, які роблять його чудовим рішенням для широкого спектру вимог до керування базами даних. Відомий своєю надійністю, повним набором функцій і гнучкістю, він справляється зі складними завданнями з керування та обробки даних із надзвичайною ефективністю. Ця складна система баз даних розроблена для підтримки великомасштабних програм і сховищ даних, що дозволяє користувачам створювати масштабовані та безпечні системи.

PostgreSQL вирізняється продуктивністю та масштабованістю, легко обробляючи великі обсяги даних і одночасні транзакції. Він використовує складні методи оптимізації для забезпечення ефективного зберігання та пошуку даних, що робить його придатним для налаштувань із високим попитом. Його архітектура пропонує горизонтальну масштабованість, розділення та реплікацію, що дозволяє програмам легко масштабуватися у відповідь на зростання обсягів даних і трафіку користувачів.

Надійна підтримка транзакцій є фундаментальною для PostgreSQL, забезпечуючи цілісність і узгодженість даних через повну відповідність ACID і

кілька рівнів ізоляції транзакцій. Ця структура захищає від аномалій даних і гарантує надійну обробку транзакцій, навіть у складних багатокористувацьких налаштуваннях.

Широкі можливості оптимізації запитів PostgreSQL забезпечують ефективне виконання складних запитів. Він містить такі можливості, як сканування лише індексу, сканування купи растрових зображень і оптимізація генетичних запитів для скорочення часу виконання запитів. Це робить PostgreSQL ідеальним для аналізу даних і програм бізнес-аналітики, які потребують швидкого й точного пошуку даних [30].

Pg-hstore – це бібліотека для Node.js, яка забезпечує підтримку типу даних hstore у PostgreSQL. hstore – це розширення PostgreSQL, яке дозволяє зберігати ключ-значення пари у вигляді одного поля таблиці. Це дуже зручно для зберігання структурованих даних, де кількість і типи полів можуть бути змінними або не визначеними заздалегідь [31].

Основні функції pg-hstore:

– серіалізація: перетворення об'єктів JavaScript у формат hstore для зберігання в базі даних PostgreSQL;

– десеріалізація: перетворення даних hstore з PostgreSQL назад у об'єкти JavaScript.

Sequelize є інструментом ORM для Node.js, який працює з такими базами даних як Postgres, MySQL, MariaDB, SQLite та Microsoft SQL Server. Він пропонує повноцінну підтримку транзакцій, управління відносинами між таблицями, а також можливості для реплікації даних. Sequelize використовує проміси для асинхронної роботи з базами даних, що забезпечує більш гнучке управління кодом. Однією з переваг Sequelize є його незалежність від конкретної системи управління базами даних. Це означає, що ви можете без зусиль перейти з однієї бази даних на іншу, просто змінивши налаштування у файлі конфігурації, при цьому більшість вашого коду не потребуватиме змін [32]. Принцип дії зображено на рисунку 2.7

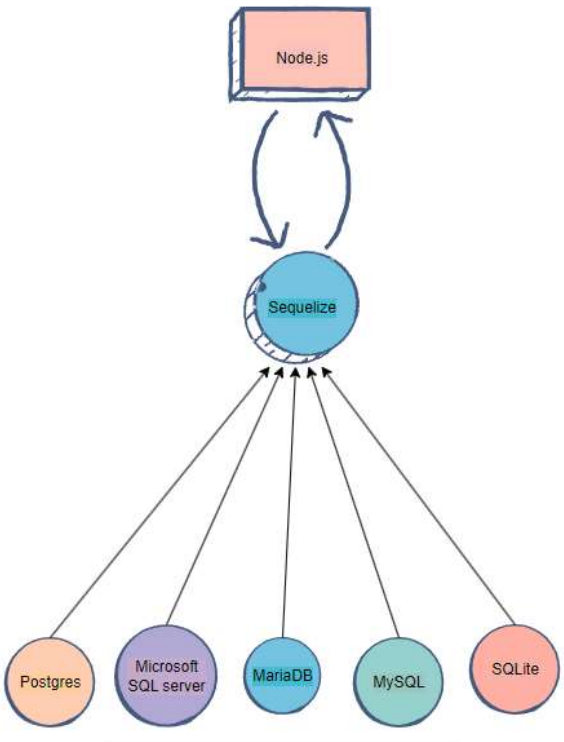

Рисунок 2.7 – Принцип дії Sequelize

PgAdmin 4 є відмінним відкритокодовим інструментом для адміністрування Postgres. Він розроблений для ефективного моніторингу та управління серверами PostgreSQL та EDB Advanced Server, незалежно від того, чи є вони локальними чи віддаленими. Завдяки зручному графічному інтерфейсу, pgAdmin полегшує процес створення, управління та маніпулювання об’єктами баз даних, а також надає різноманітні інструменти для оптимізації роботи з базами даних.pgAdmin можна інсталювати в двох режимах: робочий стіл і сервер. Режим настільного комп’ютера встановлюється як окрема програма, яка використовується одним і тим же користувачем операційної системи, тоді як до режиму сервера можна отримати доступ через мережу, що дозволяє використовувати його кільком користувачам [33].

2.4 Опис структури бази даних та проекту

Структура бази даних полягає у формуванні сутностей та правильній їх розстановці. Для створення діаграми було використано сервіс draw.io.

Найпершою сутністю буде користувач. У неї, як і у інших, повинен бути присутнім параметр id для ідентифікації. Також у користувача буде пароль, емейл та роль, щоб у подальшому налаштовувати доступ. Наступною сутністю буде корзина, яка матиме id та user_id, параметр який буде зв'язувати корзину з конкретним користувачем. Тип зв'язку буде один до одного, адже кожен користувач буде мати одну власну корзину. Наступною таблицею буде товар з параметрами id, назвою, ціною, рейтингом та зображенням. Також у товару буде 2 внутрішніх ключі, а саме typeId і brandId. Відповідно створюєм 2 сутності з назвою і id. Типи і бренди по відношенню до товару будуть мати з'язок один до багатьох: одному типу та одному бренду може належати декілька товарів. А між цими двома сутностями буде зв'язок багато до багатьох, тому що один бренд може належати багатьом типам і навпаки. Наступною таблицею буде інформація про товар з параметрами device_id, назва характеристики та її опис. Товар матиме декілька характеристик, тому зв'язок один до багатьох. Добавляємо додаткову сутність з інформацією про товари в корзині. Одна корзина може мати багато товарів. А також ця таблиця з'єднана з товаром один до одного. Останньою сутністю буде рейтинг. Він з'єднаний з користувачем зв'язком багато до одного: один користувач зможе поставити безліч відгуків. А також рейтинг це один з параметрів товару, тому відповідно поєднуєм ці дві таблиці. В результаті вийшла таблиця сутностей, яка зображена на рисунку 2.8.

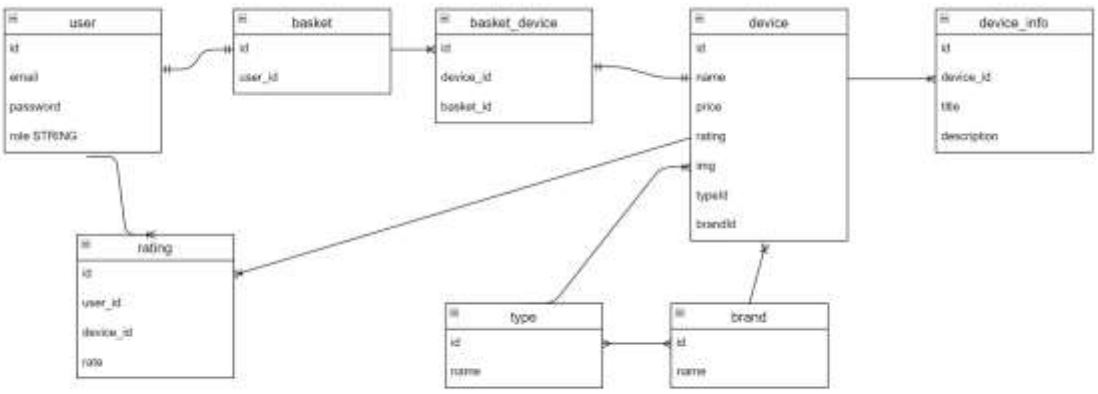

Рисунок 2.8 – Таблиця сутностей

Структура проекту поділена на 2 частини, а саме серверна та клієнтська. У server буде декілька основних папок:

– Controllers;
– Error;
– MiddleWare;
– Models;
– Routes;
– Static.

У client будуть присутні такі папки:

– Public;
– Src;
– Assets;
– Components;
– http;
– pages;
– store;
– utils.

Розглянемо деякі з них детальніше. У папці Controllers буде 4 файли, які зображені на рисунку 2.9.

Рисунок 2.9 – Папка Controllers

У UserController відбуватиметься функції авторизації та реєстрації і провірки на доступ. У файлах BrandController та TypeController будуть однакові функції, а

саме створення або сортування. DeviceController міститиме скрипт для створення, сортування та виведення одного конкретного товару

У папці Models будуть знаходитися моделі даних. У файлі models.js за допомогою Sequelize буде проводитись опис даних. Через функцію define будуть створюватися об’єкти з параметрами сутностей. Також у даному файлі будуть описані зв’язки між таблицями БД.

У директорії Routes будуть присутні маршрути, по яким будуть відкриватися компоненти на сайті. Існуватиме основний файл index, у нього будуть експортуватися всі роути. Відповідно 4 маршрути: бренд, товар, тип та користувач, які зображені на рисунку.

У папці MiddleWare буде скрипт, який буде відправляти інформацію про помилки, імпортуючи їх з файлу ApiError. Папка src є основною для фронтенд частини, в ній міститимуться App.js та index.js. Основною директорією для React є Components. У цій папці будуть компоненти, які відмальовуватимуться за роутами.

2.5 Висновок до другого розділу

У цьому розділі було висвітлено особливості фреймворків, які будуть використовуватися для розробки. Також описано базу даних, наведено структуру сутностей і показано структуру проекту з детальним поясненням основних елементів проекту. Наведено інформацію про всі додаткові розширення до фреймворків, які будуть необхідні при роботі.

РОЗДІЛ 3. ПРАКТИЧНА ЧАСТИНА

3.1 Серверна частина коду

Створюєм директорії server та client, відповідно серверна та клієнтська частини сайту. У папці server створюємо файл index.js, цей файл є основним і саме з нього і буде починатися запуск. Командою у терміналі npm init ініціалізуєм проект, після чого з'являється файл package.json. Далі інсталюємо всі необхідні фреймворки, а саме Express, Pg, Pg-hstore, Sequelize. Після цього процесу результат інсталювання можна побачити у полі dependencies у файлі package.json, фрагмент якого зображений на рисунку 3.1.

```
"dependencies": {
  "@testing-library/jest-dom": "^5.11.9",
  "@testing-library/react": "^11.2.3",
  "@testing-library/user-event": "^12.6.2",
  "axios": "^0.21.4",
  "bootstrap": "^4.6.2",
  "jwt-decode": "^3.1.2",
  "mobx": "^6.12.3",
  "mobx-react-lite": "^3.4.3",
  "react": "^17.0.2",
  "react-bootstrap": "^1.4.3",
  "react-dom": "^17.0.1",
  "react-router-dom": "^5.3.4",
  "react-scripts": "4.0.1",
  "web-vitals": "^0.2.4"
```

Рисунок 3.1 – Фрагмент файлу з встановленими фреймворками

Також інсталюєм nodemon для того, щоб при кожній зміні коду сервер не потрібно було вручну перезавантажувати, а робити це автоматично. Наступним етапом пишемо скрипт, який буде запускати застосунок в режимі розробки: у файлі package.json створюємо строку dev, в яку записуємо nodemon index.js. Далі переходимо до створеного раніше файлу index.js. За допомогою функції require імпортуємо фреймворк Express. Далі створюємо константи PORT і App. PORT

відповідає за порт на якому буде працювати сайт, а App буде відповідно запускати Express.

Наступним важливим етапом буде підключення до бази даних. Створюємо файл db.js, у якому ми експортуємо об’єкт з класу Sequelize. Так як було обрано PostgreSQL, то заходимо в застосунок PgAdmin, і створюємо нову базу даних. Вертаємося до файлу db.js і передаємо дані БД, а саме ім’я користувача, назва БД, а також об’єкт з полями dialect з значенням postgres, пароль та хост.

Імпортуємо Sequelize в файл index.js. Далі створюємо функцію, яка і буде викликати базу даних, називаємо її start, робимо її асинхронною. У функцію вставляєм конструкцію try-catch, щоб виявляти потенційні помилки. В блок try вставляємо функцію Sequelize.authenticate, яка і відповідає за підключення бази даних, а також Sequelize.sync, яка додатково буде запускати синхронізацію з структурою БД.

Переходимо до опису моделей даних. Створюємо окрему директорію models з однойменним файлом у форматі js. У ньому імпортуєм клас DataTypes з Sequelize, який відповідає за опис полів у даних. Першою моделлю буде User(користувач). За допомогою функції define, передаємо об’єкт User з такими даними:

– id (тип даних INTEGER, первинний ключ і автоінкрементування);
– email (тип даних STRING, унікальний);
– password (тип даних STRING);
– role (тип даних STRING, по замовчуванню користувач).

Аналогічним методом створюємо моделі корзини та товаром в корзині з полем id. Наступною важливою моделлю буде Device (товар), який буде мати наступні поля:

– id (тип даних INTEGER, первинний ключ і автоінкрементування);
– name (тип даних STRING, унікальний, обов’язкове для заповнення);
– price (тип даних INTEGER, обов’язкове для заповнення);
– rating (тип даних INTEGER, за замовчуванням 0);
– img (тип даних STRING, обов’язкове для заповнення).

Моделі Brand і Type будуть мати однакові поля, а саме id (тип даних INTEGER, первинний ключі і автоінкрементування) та name (тип даних STRING, унікальний, обов’язкове для заповнення).

Останніми моделлями даних будуть Rating і DeviceInfo. Створюємо поля для Rating:

– id (тип даних INTEGER, первинний ключ і автоінкрементування);

– rate (тип даних INTEGER, обов’язкове для заповнення);

Додаємо поля для DeviceInfo:

– id (тип даних INTEGER, первинний ключ і автоінкрементування);

– title (тип даних STRING, обов’язкове для заповнення);

– description (тип даних STRING, обов’язкове для заповнення).

Повертаємося до файлу index.js і імпортуємо моделі. Після пророблених дій у pgAdmin з’являються створені таблиці, подані на рисунку 3.2.

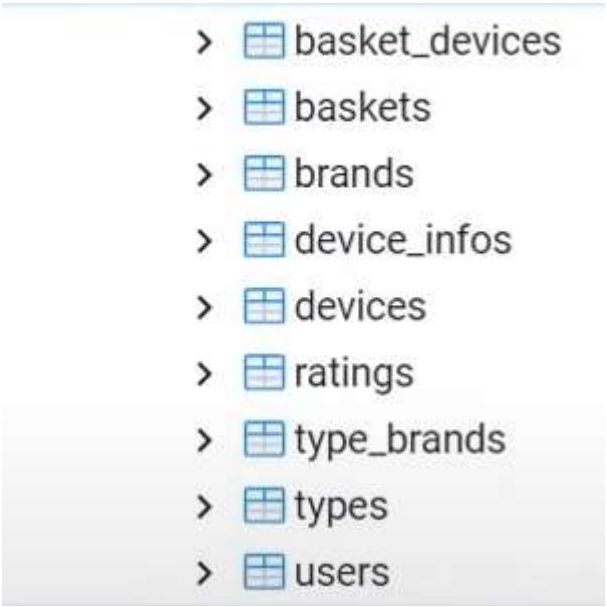

Рисунок 3.2 – Таблиці сутностей

Тепер імпортуємо функцію cors з однойменного пакету. Щоб перевірити чи працюють запити до БД запускаємо програму Postman і відправляємо запит GET. Бачимо тестове повідомлення, що усе працює, яке показано на рисунку 3.3.

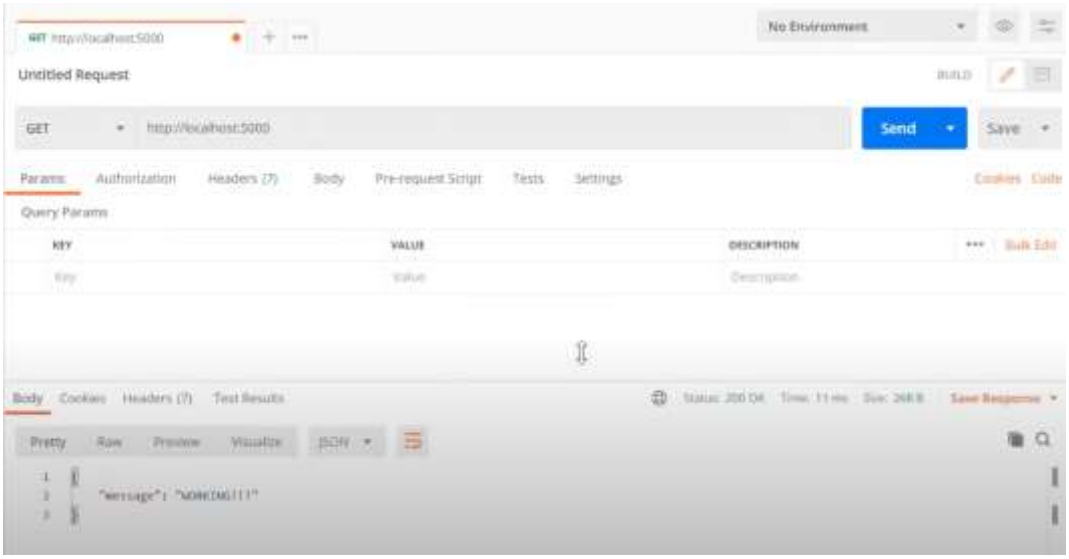

Рисунок 3.3 – Повідомлення в Postman

Наступним етапом буде створення routes (шляхів). Створюємо окрему директорію з назвою routes та відповідними файлами. Структура routes зображена на рисунку 3.4.

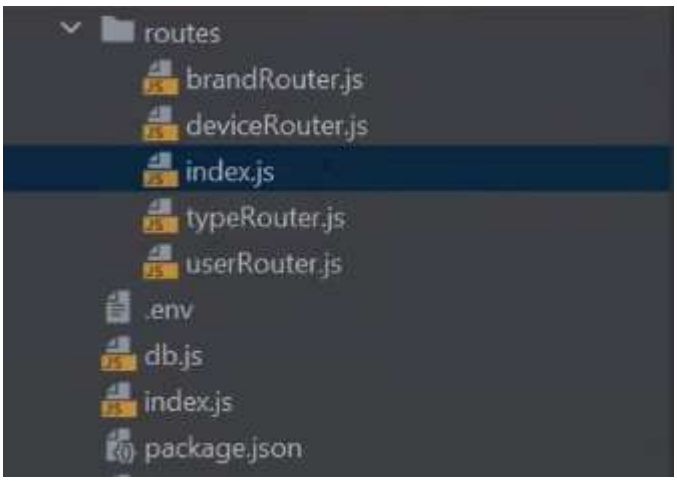

Рисунок 3.4 – Папка з маршрутами

Імпортуємо у файл index.js фреймворк Express. Створюємо об’єкт основного роутера, який будемо експортувати в інші підрядні роутери. Далі у кожен роутер добавляємо запити, які вони будуть використовувати. BrandRouter матиме запити post і get. User матиме 2 post-запити реєстрація та логін та один get для перевірки авторизації користувача. Для DeviceRouter буде 3 методи: 2 get запити і 1 post.

Роутери створені, далі потрібно підключити їх до основного серверного файлу, який буде використовувати їх. Імпортуємо в index.js основний роутер. Викликаємо функцію use, передаючи першим параметром URL-адресу, а другим сам імпортований роутер. Перевіряємо чи раніше створені роути працюють коректно. На рисунку 3.5 зображено тестове повідомлення, що сповіщає про те, що все працює.

Рисунок 3.5 – Повідомлення про коректну роботу

Оскільки відобразилося тестове повідомлення, значить все зроблено правильно. Наступним етапом буде налаштування реєстрації за допомогою JWT токену.

Інсталюємо в нашу директорію jsonwebtoken, та bcrypt, щоб хешувати паролі, та не тримати їх в відкритому вигляді в БД. Створюємо файл UserController.js, в який імпортуєм bcrypt [34]. Добавляєм асинхронну функцію реєстрації, яка містить об’єкт з значеннями email, password, role. За допомогою циклу if, робимо умову, якщо поля з поштою та паролем пусті, то повертаєм помилку. Також робимо перевірку, чи є вже зареєстований користувач з поштою, яка введена в поле, якщо умова задовільна, то виводим відповідне сповіщення, що така пошта на сервері уже існує. Далі, якщо пошта ще не зареєстрована, приймаєм введений пароль у функцію хешування, яка відбувається за допомогою bcrypt. У параметрах потрібно вказати сам пароль і кількість необхідних хешувань. Після цього етапу, починається функція, яка створює

безпосередньо самого користувача. Також створюємо особисту корзину з товарами, в яку приймаєм унікальний ідентифікатор користувача.

Тепер імпортуєм JWT у наш файл. Викликаєм функцію sign, в яку ми передаєм реєстраційні дані користувача першим параметром, другим секретний код, який обрано самостійно, а третім параметром тривалість життя самого токену. Тривалість рекомендовано обирати невелику, щоб при можливій крадіжці токен змінювався швидко. При тестовому вводі даних, формується токен, що зображений на рисунку 3.6.

Рисунок 3.6 – Формування токену JWT

Далі налаштовуємо авторизацію. Створюємо відповідну функцію findOne, яка буде перевіряти чи є надана пошта в БД. Якщо умова не задовільняється, то повертаємо помилку, яка сповіщає, що дані не знайдені. При виконанні умови робимо функцію, яка розшифровує пароль, який лежить в БД і прив'язаний до пошти, і перевіряє ідентичність з наданим користувачем паролем. Далі, якщо всі дані співпадають, перетворюємо їх в токен і передаємо на клієнт. Перевіряємо за допомогою PostMan чи формується JWT токен. Результат перевірки зображений на рисунку 3.7.

Рисунок 3.7 – Перевірка формування токену при авторизації

Наступним кроком буде створення окремої функції для перевірки валідності токену. У створеному файлі authMiddleware.js створюємо цикл try, який буде приймати токен. Якщо токену не існує, то повертаємо помилку з статусом 401. Якщо токен є валідним, то він передається в цикл, який його розшифровує. Добавляємо тестове повідомлення і надсилаємо запит через Postman, щоб провірити чи працює належним чином авторизація. Процес перевірки зображений на рисунку 3.8.

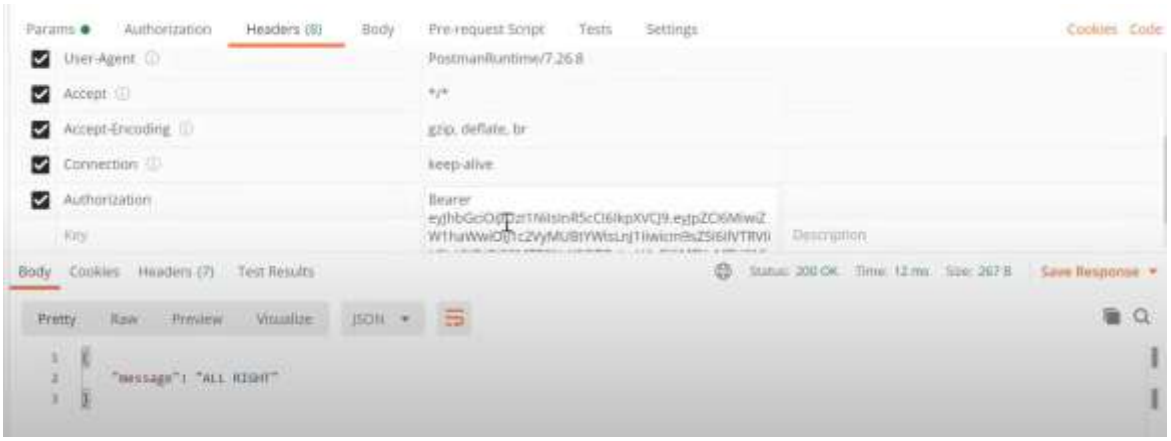

Рисунок 3.8 – Перевірка роботи авторизації

Далі замість тестового повідомлення добавляємо функцію, яка буде перезаписувати токен для авторизованого користувача. Для цього використовуємо generateJwt, в який передаємо дані (email, id, role).

Для того, щоб звичайний користувач не міг добавляти товари в магазин потрібно налаштувати доступ для ролей. Створюєм файл CheckRole.js, в ньому робимо ту ж структуру, що й для авторизації, лише добавляємо умову декодування ролі і зрівняння з наданими даними. Якщо роль не адміністратор, то виводиться сповіщення про відсутність доступу.

3.2 Клієнтська частина коду

Для початку роботи інсталюєм необхідні модулі, а саме axios, react-router-dom, mobx, mobx-react-lite, react-bootstrap [35]. Створюємо файл App.js, який і є основним для даної частини коду. Запускаємо за допомогою команди npm start

файл і бачимо тестове повідомлення, що свідчить про правильне інсталювання React.

У папці Pages створюємо усі необхідні файли. Тут будуть знаходитись сторінки веб-сайту. Першим компонентом буде AppRouter.js, в якому буде описано логіку навігації по сторінках. На деякі сторінки зможуть зайти всі користувачі, а на інші лише авторизовані. Імпортуємо модулі Switch, Route, Redirect з пакету react-router-dom. Структура папки pages зображена на рисунку 3.9.

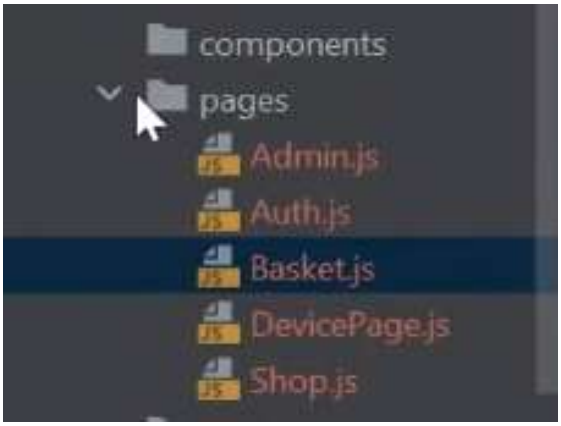

Рисунок 3.9 – Структура папки pages

Редактуємо файл App.js. Огортаємо весь код в тег BrowseRouter, імпортуєм React та BrowseRouter. Сюди добавляєм створений раніше AppRouter.

Далі створимо файл маршрутів. У файлі routes.js буде 2 масиви: маршрути сторінок для всіх користувачів та маршрути тільки для авторизованих. У кожному об’єкті буде параметр path, тобто шлях або посилання, та компонент. Для зручності всі параметри path кладемо у окремий файл consts.js у директиві utils. З цього файлу будуть експортуватися всі шляхи, так як кількість сторінок може збільшуватися і так легше редагувати код.

Роути, які будуть доступні тільки авторизованим користувачам це адмін-панель та корзина. Поміщаємо їх у масив authRoutes. Всі решта роутів, відповідно, у масив publicRoutes. Посилання створені, тепер потрібно їх активізувати. Переходим до файлу AppRouter, і повертаємо компонент Switch. У ньому за допомогою методу map витягуємо з об’єкту шлях і компоненту. Також

ставимо ключ exact, щоб пересвідчитись, що шлях точно співпадає. Перед методом map ставимо перевірку isAuth, щоб посилання були доступні тільки авторизованим користувачам. Та ж сама структура і для публічних роутів, проте без перевірки на авторизацію. Останнім маршрутом в Switch буде Redirect, який, якщо користувач ввів неіснуюче посилання, відправлятиме на головну сторінку. Далі створюємо в директиві store файл DeviceStore.js, у який поміщаємо constructor, в якому будуть зберігатися дані про товари, а саме бренди, типи і назви. Також для кожної змінної робимо запити GET.

Переходимо до стилізування самих сторінок. Створюємо файл NavBar, вказуємо перевірку, так як вид цієї сторінки буде залежати від авторизованості користувача. Підключаємо бібліотеку Bootstrap, і створюємо саму структуру. Добавляємо тег NavLink з назвою магазину, який буде при натисканні відправляти користувача на головну сторінку. Також на цій сторінці створюєм кнопки адмін-панелі та авторизації та виходу з аккаунту.

Наступним кроком буде стилізування вікно авторизації та реєстрації. Заходимо в створений файл Auth.js, добавляєм Container. Встановлюєм необхідні стилі та обов’язкову форму з двома параметрами: електронна пошта і пароль. Також будуть кнопки Увійти та посилання на реєстрацію. Вигляд модального вікна авторизації зображений на рисунку 3.10.

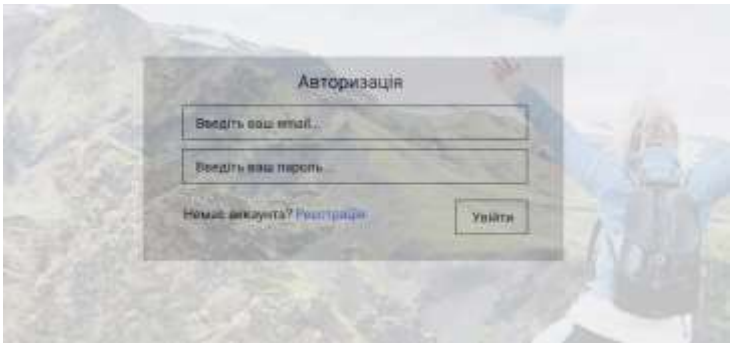

Рисунок 3.10 – Вікно авторизації

Структура реєстрації ідентична, проте посилання буде на авторизацію. Вигляд модального вікна зображений на рисунку 3.11.

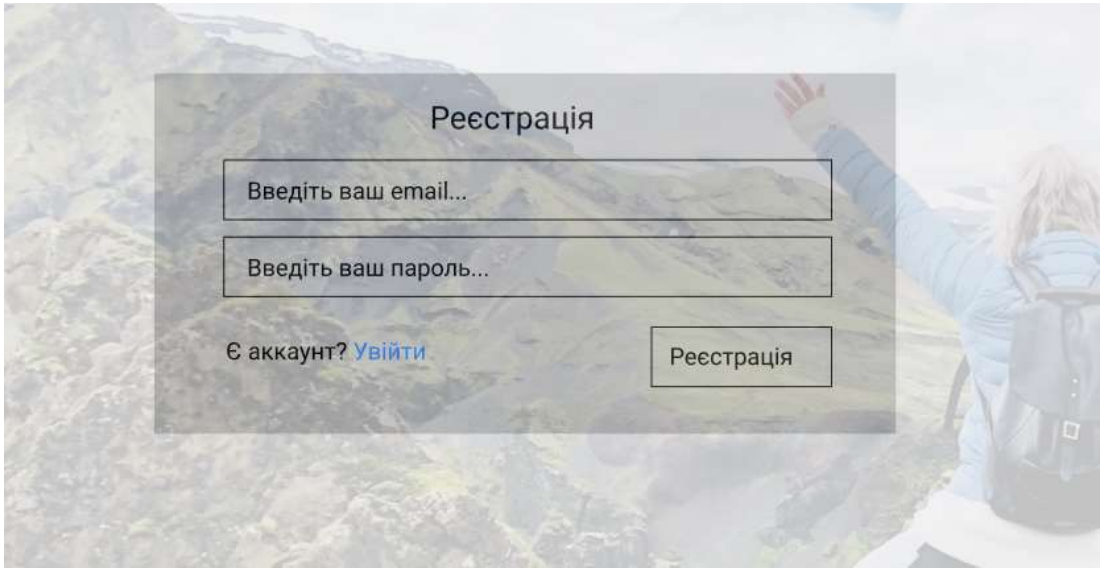

Рисунок 3.11 – Вікно реєстрації

Далі створюємо список типів товару. Для цього в файлі TypeBar.js за допомогою методу map витягуємо типи. Вигляд вікна з типами показано на рисунку 3.12.

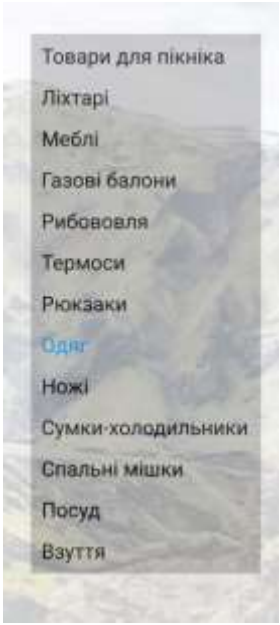

Рисунок 3.12 – Список типів товарів

Наступним важливим елементом є список брендів BrandBar. Знову використовуємо метод map і переносимо бренди з файлу на сайт. На рисунку 3.13 зображено список брендів.

Рисунок 3.13 – Список брендів товарів

Також створюємо сам список товарів DeviceList. Імпортуємо дані з файлу, добавляємо піктограму зірки для рейтингу та регулюємо розмір фото товару. Товари бренду P1G зображені на рисунку 3.14.

Рисунок 3.14 – Товари бренду P1G

Далі стилізуємо окрему сторінку товару, при виборі його користувачем. За допомогою унікального ід витягуємо інформацію про товар. Сторінка з окремим товаром зображена на рисунку 3.15.

Рисунок 3.15 – Вигляд сторінки з конкретним товаром

Останньою сторінкою буде адмін-панель. Через неї адміністратор зможе добавляти типи, бренди і товари. Імпортуємо Module для модальних вікон. Всередину модального вікна добавляємо форму з відповідними полями. Вигляд цієї функції показано на рисунку 3.16.

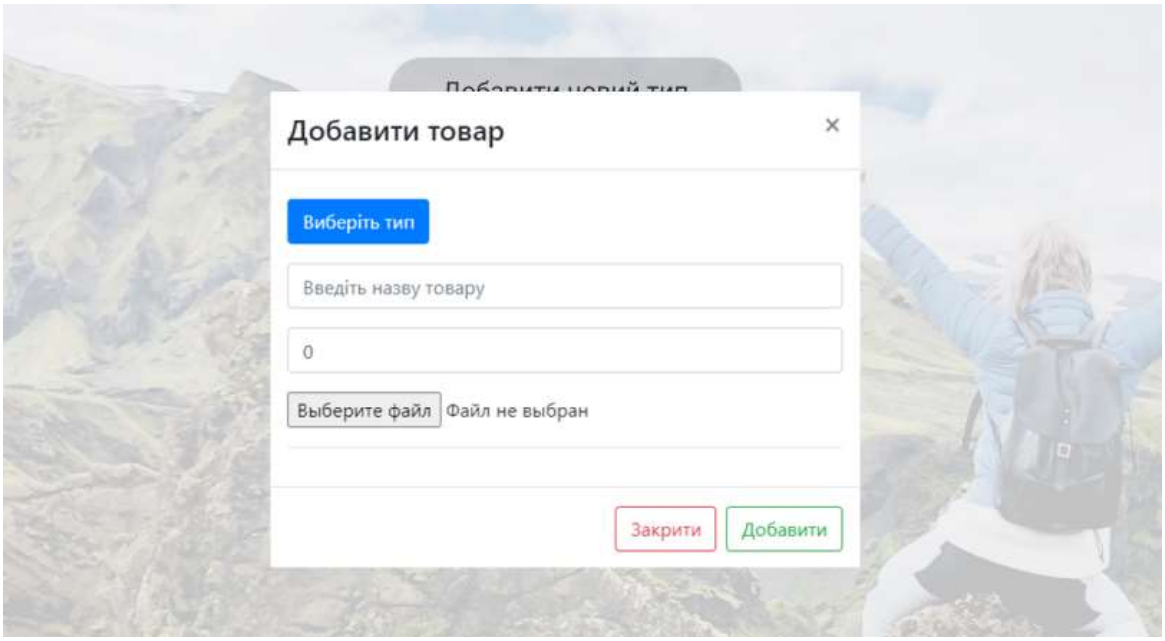

Рисунок 3.16 – Модальне вікно створення типів

Фінальний вигляд магазину Еверест можна побачити на рисунку 3.17. Присутні товари, які відсортовані по типу і бренду, авторизовано адмін-панель.

Рисунок 3.17 – Фінальний вигляд сайту

Розробка сайту завершена, всі функції допрацьовані до кінця. При проведенні тестування помилок не виявлено, все коректно відображається.

3.3 Висновок до третього розділу

У цьому розділі описана практична частина проекту. Реалізовано основну структуру веб сайту з підключенням до баз даних. Створено моделі даних та зв’язок з ними, скрипт для опрацювання помилок, вивід товарів, їх фільтрація.

У даному проекті успішно функціонує система реєстрації та авторизації через JWT токен. У адміністраторів наявна своя панель з модальним вікном, яка дозволяє редагувати список товарів. Також реалізовані функції для перевірки ролі і обмеження доступу незареєстрованих користувачів.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Ергономічні проблеми безпеки життєдіяльності

Ергономіка – це наука, що вивчає взаємодію людини з елементами системи, в якій вона працює, з метою оптимізації ефективності та комфорту. Правильне впровадження ергономічних принципів на підприємстві сприяє підвищенню продуктивності праці, зниженню ризику професійних захворювань та травм. Однак, нехтування ергономікою може призвести до численних проблем, що негативно впливають на здоров'я працівників та ефективність роботи.

Мускуло-скелетні розлади (МСД) є однією з найпоширеніших ергономічних проблем на підприємствах. Вони включають біль у спині та шиї, що частіше виникає у працівників, які довгий час проводять у сидячому положенні або виконують роботи, пов’язані з підняттям важких предметів. Синдром карпального тунелю спричиняється повторюваними рухами кистей і зап'ястків, що характерно для працівників, які часто використовують комп'ютерну мишу або інструменти. Тендиніт і бурсит виникають внаслідок повторюваних рухів або перенапруження суглобів [36].

Неправильне положення тіла під час роботи може призвести до хронічного дискомфорту та травм. Наприклад, невірна висота робочого столу може призвести до незручності та напруження м’язів спини і шиї. Неправильна організація робочого місця, така як недостатній простір для ніг, невірно розташовані монітори і клавіатури, відсутність підлокітників на стільцях, також може спричинити дискомфорт і напруження.

Робота, що вимагає повторюваних рухів, може спричинити мікротравми м’язів і суглобів. Це особливо актуально для працівників на конвеєрах, у сфері обробки даних, на виробництві тощо.

Довготривалий вплив вібрації та шуму може негативно вплинути на здоров'я працівників. Вібраційна хвороба виникає внаслідок тривалого впливу вібрації на руки і тіло, що характерно для операторів важкої техніки та

будівельників. Порушення слуху виникає через тривалий вплив високих рівнів шуму, що може спричинити втрату слуху та інші слухові проблеми.

Неправильна організація робочого процесу може спричинити високий рівень стресу і психологічного навантаження. Стрес і вигоряння виникають через високий темп роботи, невизначеність завдань, недостатню підтримку з боку керівництва. Психологічний дискомфорт може бути наслідком невідповідних умов праці, наприклад, поганого освітлення або невідповідної температури, що може спричинити незадоволення та зниження мотивації.

Поліпшення робочих місць включає використання регульованих меблів, таких як стільці, столи та монітори з можливістю регулювання висоти і нахилу. Оптимізація розташування інструментів передбачає розміщення часто використовуваних предметів у легкодоступних місцях згідно з рекомендаціями ДСТУ ISO 6385:2016 «Ергономічні принципи проектування робочих систем».

Впровадження перерв та змін у роботі сприяє зменшенню напруження м’язів і суглобів. Регулярні короткі перерви є важливими для зменшення напруження. Ротація завдань, яка включає зміну типу робіт, зменшує повторювані рухи та рівномірно розподіляє навантаження.

Навчання працівників є ключовим аспектом в запобіганні ергономічних проблем. Тренінги з ергономіки навчають працівників правильним робочим позам, методам підняття важких предметів і використанню ергономічного обладнання. Програми фізичної підготовки заохочують працівників до фізичної активності, що зміцнює м’язи та зменшує ризик травм.

Контроль середовища включає зниження рівня шуму за допомогою звукоізоляції та захисних засобів для слуху, а також впровадження обладнання та інструментів для зменшення впливу вібрацій [37].

4.2 Проведення інструктажів з охорони праці

Інструктажі з питань охорони праці проводяться на всіх підприємствах, установах і організаціях незалежно від характеру їх трудової діяльності, підлеглості і форми власності. Мета інструктажу – навчити працівника

правильно і безпечно для себе і навколишнього середовища виконувати свої трудові обов’язки [38].

Інструктажі за часом і характером проведення поділяють на: вступний, первинний, повторний, позаплановий та цільовий.

Вступний інструктаж проводиться з усіма працівниками, які щойно прийняті на роботу (постійну або тимчасову), незалежно від їх освіти, стажу роботи за цією професією або посади; працівниками, які знаходяться у відрядженні на підприємстві й беруть безпосередню участь у виробничому процесі; з водіями транспортних засобів, які вперше в’їжджають на територію підприємства; учнями, вихованцями та студентами навчально-виховних закладів перед початком трудового й професійного навчання в лабораторіях, майстернях на полігонах тощо.

Вступний інструктаж проводить спеціаліст відділу охорони праці або особа, що призначена наказом для проведення цієї роботи. Місце проведення вступного інструктажу – кабінет охорони праці або інше приміщення, обладнане наочними матеріалами. Програма вступного інструктажу розробляється відділом охорони праці. Програму та тривалість інструктажу затверджує роботодавець.

Запис про проведення вступного інструктажу робиться в спеціальному журналі, а також у документі про прийняття працівника на роботу, де розписуються інструктуючий та проінструктований працівники.

Первинний інструктаж проводиться на робочому місці до початку роботи з новоприйнятим працівником або працівником, який буде виконувати нову для нього роботу, студентом, учнем та вихованцем перед роботою в майстернях, лабораторіях, дільницях тощо. Первинний інструктаж проводиться індивідуально або для групи осіб спільного фаху за програмою, складеною з урахуванням вимог відповідних інструкцій з охорони праці та інших нормативних актів про охорону праці і технічної документації.

Повторний інструктаж проводиться на робочому місці з усіма працівниками: на роботах із підвищеною небезпекою – один раз на квартал; на інших роботах – один раз у півріччя. Мета інструктажу – поновити знання та уміння виконувати працівником роботу правильно і безпечно. Проводиться

інструктаж індивідуально або для групи працівників, що виконують однотипні роботи, за програмою первинного інструктажу в повному обсязі.

Позаплановий інструктаж проводиться з працівниками на робочому місці або в кабінеті охорони праці у таких випадках:

– при введенні в дію нових або змінених нормативних актів про охорону праці;

– при зміні технологічного процесу, заміні або модернізації устаткування, приладів та інструментів, вихідної сировини, матеріалів та інших факторів, що впливають на охорону праці;

– при порушенні працівником нормативних актів, що може призвести до травми, отруєння або аварії;

– на вимогу працівника органу державного нагляду або вищої за ієрархією державної чи господарської організації при виявленні недостатнього знання працівником безпечних прийомів праці і нормативних актів про охорону праці;

– при перерві в роботі виконавця робіт більше, ніж 30 календарних днів (для робіт із підвищеною небезпекою), а для решти робіт – більше 60 днів.

Позаплановий інструктаж проводиться індивідуально або для групи працівників спільного фаху. Обсяг і зміст інструктажу визначається для кожного окремого випадку залежно від причин і обставин, що викликали необхідність його проведення[39].

Цільовий інструктаж проводиться у таких випадках:

– при виконанні разових робіт, що не пов’язані безпосередньо з основними роботами працівника;

– при ліквідації наслідків аварії і стихійного лиха;

– при виконанні робіт, що оформляються нарядом-допуском, письмовим дозволом та іншими документами;

– в разі проведення екскурсій або організації масових заходів з учнями та вихованцями (екскурсії, походи, спортивні заходи тощо).

Цільовий інструктаж фіксується нарядом-допуском або іншим документом, що дозволяє проведення робіт.

Первинний, повторний, позаплановий та цільовий інструктажі проводить безпосередньо керівник робіт (начальник або інструктор підприємства). Перевірка знань здійснюється усним опитуванням або за допомогою технічних засобів навчання, а також перевіркою навичок виконання робіт відповідно до вимог безпеки.

Первинний, повторний та позаплановий інструктажі, стажування та допуск до роботи реєструються в спеціальних журналах. При цьому обов'язкові підписи як інструктованого, так і інструктуючого. Журнали інструктажів повинні бути пронумеровані, прошнуровані і скріплені печаткою.

Працівники, що не пов'язані з обслуговуванням обладнання, використанням інструменту, збереженням сировини, матеріалів тощо, можуть бути звільнені від первинного, повторного та позапланового інструктажу за наказом (розпорядженням) керівника підприємства по узгодженню з державним інспектором Держпромгірнагляду.

Роботодавець або керівник структурного підрозділу зобов'язаний видати працівнику примірник інструкції з охорони праці за його професією або вивісити її на робочому місці [40].

4.3 Висновок до четвертого розділу

Проведення інструктажів з охорони праці є критично важливим для забезпечення безпеки на робочому місці. Це дозволяє працівникам бути обізнаними з потенційними ризиками та правильними процедурами дій у разі надзвичайних ситуацій. Ергономічні проблеми на підприємстві можуть значно впливати на здоров'я працівників і ефективність роботи. Впровадження ергономічних принципів і заходів може не лише покращити умови праці, а й знизити кількість професійних захворювань і травм, підвищити продуктивність і мотивацію працівників.

Загалом, інструктажі з охорони праці та усунення ергономічних проблем є не лише законодавчою вимогою, але й етичною необхідністю, що сприяє створенню безпечного та здорового робочого середовища.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено сайт для продажу туристичного спорядження Еверест. Успішно створено головну сторінку з товарами, їх типами та брендами, адмін-панель з можливостями редагування, інтегровано систему авторизації та реєстрації з відповідним підключенням до баз даних.

У першому розділі було проаналізовано наступне:

– поняття веб сайту;
– структура створення веб сайтів;
– види та застосування веб сайтів;
– вибір технологій для розробки.

У другому розділі було сформовано наступне:

– опис технологій для серверного коду;
– опис технологій для клієнтського коду;
– опис технологій баз даних;
– опис структури проекту та формування сутностей.

У третьому розділі було спроєктовано наступне:

– підключення до баз даних, токену JWT;
– методи авторизації та реєстрації;
– компоненти товару, брендів та типів;
– функціональну панель для адміністраторів.

У розділі «Безпека життєдіяльності, основи охорони праці» були опрацьовані питання про ергономіку та проведення інструктажів з заходів безпеки.

ПЕРЕЛІК ДЖЕРЕЛ

1. What is a Website? [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-a-website/

2. A Look Back At The Very First Website Ever Launched, 30 Years Later [Електронний ресурс] – Режим доступу до ресурсу: https://www.npr.org/2021/08/06/1025554426/a-look-back-at-the-very-first-website-ever-launched-30-years-later

3. Understand How Websites Work in 5 Easy Steps [Електронний ресурс] – Режим доступу до ресурсу: https://webupon.com/blog/how-websites-work/

4. Different Types of Websites [Електронний ресурс] – 15.06. 2023– Режим доступу до ресурсу: https://www.geeksforgeeks.org/different-types-of-websites/

5. Front-End Development: The Complete Guide [Електронний ресурс] – Режим доступу до ресурсу: https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide

6. Segmentation and Statistical Processing of Geometric and Spatial Data on Self-Organized Surface Relief of Statically Deformed Aluminum Alloy. // Iaroslav Lytvynenko, Pavlo Maruschak, Sergiy Lupenko, Sergey Panin // Applied Mechanics and Materials, 2015, Vol. 770, pp. 288-293.

7. Software for segmentation, statistical analysis and modeling of surface ordered structures // I.V. Lytvynenko, P.O. Maruschak, S.A. Lupenko, Yu. I. Hats, A. Menou, S.V. Panin // MECHANICS, RESOURCE AND DIAGNOSTICS OF MATERIALS AND STRUCTURES (MRDMS-2016): Proceedings of the 10th International Conference on Mechanics, Resource and Diagnostics of Materials and Structures. AIP Publishing, 2016, Vol. 1785, No.1, pp. 030012-1-030012-7.

8. Methodology of the Formation of Sports Matches Statistical Information Using Neural Networks. Sorokivska O., Lytvynenko I., Sorokivskyi O., Kozbur H., Strutynska I.CEUR Workshop Proceedings, 2023, 3628, pp. 389–403.

9. What is Backend Development? Skills, Salary, Roles & More [Електронний ресурс] – Режим доступу до ресурсу:

https://www.simplilearn.com/tutorials/programming-tutorial/what-is-backend-development

10. What is an IDE (Integrated Development Environment)? [Електронний ресурс] Режим доступу до ресурсу: https://aws.amazon.com/what-is/ide/?nc1=h_ls

11. What is Visual Studio Code? Microsoft’s extensible code editor [Електронний ресурс] – Режим доступу до ресурсу: https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html

12. IntelliJ IDEA overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html

13. About WebStorm [Електронний ресурс] – Режим доступу до ресурсу: https://www.componentsource.com/product/webstorm/about

14. What is MySQL? Everything You Need to Know [Електронний ресурс] – Режим доступу до ресурсу: https://ua.talend.com/resources/what-is-mysql/

15. What Is an Oracle Database? [Електронний ресурс] – Режим доступу до ресурсу: https://www.solarwinds.com/resources/it-glossary/oracle-database

16. PostgreSQL: About [Електронний ресурс]. – Режим доступу до ресурсу: https://www.postgresql.org/about/.

17. What is React.js? Uses, Examples, & More [Електронний ресурс]. – Режим доступу до ресурсу: https://blog.hubspot.com/website/react-js

18. Introducing Hooks [Електронний ресурс]. – Режим доступу до ресурсу: https://legacy.reactjs.org/docs/hooks-intro.html

19. Terminal [Електронний ресурс]. – Режим доступу до ресурсу: https://www.jetbrains.com/help/webstorm/terminal-emulator.html

20. What is React Bootstrap ? Learn the Basics [Електронний ресурс]. – Режим доступу до ресурсу: https://www.theknowledgeacademy.com/blog/react-bootstrap/

21. What is react-router-dom ? [Електронний ресурс]. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/what-is-react-router-dom/

22. Mobx With React [Електронний ресурс]. – Режим доступу до ресурсу: https://www.scaler.com/topics/react/mobx-react/

23. React CORS Guide: What It Is and How to Enable It [Електронний ресурс]. – Режим доступу до ресурсу: https://www.stackhawk.com/blog/react-cors-guide-what-it-is-and-how-to-enable-it/

24. React Native Dotenv - Using Environment Variables in React Native [Електронний ресурс]. – Режим доступу до ресурсу: https://www.scaler.com/topics/react-native-dotenv/

25. Для чого використовується Nodemon [Електронний ресурс]. – Режим доступу до ресурсу: https://foxminded.ua/nodemon-shcho-tse/

26. What Is Node.js? Complex Guide for 2023 [Електронний ресурс]. – Режим доступу до ресурсу: https://www.netguru.com/glossary/node-js

27. Express/Node introduction [Електронний ресурс]. – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

28. What is Postman Used for? [Електронний ресурс]. – Режим доступу до ресурсу: https://apidog.com/blog/what-is-postman/

29. Introduction to JSON Web Tokens [Електронний ресурс]. – Режим доступу до ресурсу: https://jwt.io/introduction

30. What is PostgreSQL? [Електронний ресурс]. – Режим доступу до ресурсу: https://aws.amazon.com/rds/postgresql/what-is-postgresql/

31. PostgreSQL Extensions: hstore [Електронний ресурс]. – Режим доступу до ресурсу: https://www.timescale.com/learn/postgresql-extensions-hstore

32. Як Sequelize допомагає в роботі з базами даних у Node.js [Електронний ресурс]. – Режим доступу до ресурсу: https://foxminded.ua/sequelize-shcho-tse/

33. pgAdmin 4 Architecture [Електронний ресурс]. – Режим доступу до ресурсу: https://www.enterprisedb.com/blog/understanding-pgadmin-4-architecture

34. Hashing in Action: Understanding bcrypt [Електронний ресурс]. – Режим доступу до ресурсу: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/

35. Axios Docs [Електронний ресурс]. – Режим доступу до ресурсу: https://axios-http.com/uk/docs/intro

36. Що таке ергономічність? [Електронний ресурс]. – Режим доступу до ресурсу: https://ergo.place/shcho-take-erhonomichnist-tsili-zavdannia-ta-vymohy-do-robochoho-prostoru-ta-erhonomichnykh-mebliv/

37. ДСТУ 7950:2015 Дизайн і ергономіка. Робоче місце під час виконання робіт стоячи. Загальні ергономічні вимоги [Текст]. - Чинний від 2016-09-01. - Київ : УкрНДНЦ, 2016. - III, 7 с. : рис. - (Національний стандарт України)

38. Інструктажі з питань охорони праці: види, особливості проведення [Електронний ресурс]. – Режим доступу до ресурсу: https://vinetc.org.ua/novini-ta-podii/instruktazhi-pitan/

39. Організація проведення інструктажів з питань охорони праці та безпеки життєдіяльності [Електронний ресурс]. – Режим доступу до ресурсу: https://rozvytok-osvity.te.ua/orhanizatsiya-provedennya-instruktazhiv/

40. ДСТУ ISO 45001:2019 «Системи управління охороною здоров'я і безпекою праці. Вимоги та настанови щодо використання». – [Чинний від 2019-03-01]. – Київ: ДП «УкрНДНЦ», 2019. – 68 с. – (Національні стандарти України).

ДОДАТКИ

Додаток А

Код в програми в файлі App.js

```
import React, {useContext, useEffect, useState} from 'react';
import {BrowserRouter} from "react-router-dom";
import AppRouter from "./components/AppRouter";
import NavBar from "./components/NavBar";
import {observer} from "mobx-react-lite";
import {Context} from "./index";
import {check} from "./http/userAPI";
import {Spinner} from "react-bootstrap";

const App = observer(() => {
    const {user} = useContext(Context)
    const [loading, setLoading] = useState(true)

    useEffect(() => {
        check().then(data => {
            user.setUser(true)
            user.setIsAuth(true)
        }).finally(() => setLoading(false))
    }, [])

    if (loading) {
        return <Spinner animation={"grow"}/>
    }

    return (
        <BrowserRouter>
            <NavBar />
            <AppRouter />
        </BrowserRouter>
    );
});

export default App;
```

Додаток Б

Код в програми в файлі UserController.js

```
const ApiError = require('../error/ApiError');
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const {User, Basket} = require('../models/models')

const generateJwt = (id, email, role) => {
    return jwt.sign(
        {id, email, role},
        process.env.SECRET_KEY,
        {expiresIn: '24h'}
    )
}

class UserController {
    async registration(req, res, next) {
        const {email, password, role} = req.body
        if (!email || !password) {
            return next(ApiError.badRequest('Некоректний email або
пароль'))
        }
        const candidate = await User.findOne({where: {email}})
        if (candidate) {
            return next(ApiError.badRequest('Користувач з таким
email уже існує'))
        }
        const hashPassword = await bcrypt.hash(password, 5)
        const user = await User.create({email, role, password:
hashPassword})
        const basket = await Basket.create({userId: user.id})
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }

    async login(req, res, next) {
        const {email, password} = req.body
        const user = await User.findOne({where: {email}})
        if (!user) {
            return next(ApiError.internal('Користувач не
знайдений'))
        }
        let comparePassword = bcrypt.compareSync(password,
user.password)
        if (!comparePassword) {
            return next(ApiError.internal('Вказаний неправильний
пароль'))
        }
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }
```

```
    async check(req, res, next) {
        const token = generateJwt(req.user.id, req.user.email,
req.user.role)
        return res.json({token})
    }
}

module.exports = new UserController()
```

Додаток В

Код в програми в файлі models.js

```
const sequelize = require('../db')
const {DataTypes} = require('sequelize')

const User = sequelize.define('user', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    email: {type: DataTypes.STRING, unique: true,},
    password: {type: DataTypes.STRING},
    role: {type: DataTypes.STRING, defaultValue: "USER"},
})

const Basket = sequelize.define('basket', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
})

const BasketDevice = sequelize.define('basket_device', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
})

const Device = sequelize.define('device', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    name: {type: DataTypes.STRING, unique: true, allowNull: false},
    price: {type: DataTypes.INTEGER, allowNull: false},
    rating: {type: DataTypes.INTEGER, defaultValue: 0},
    img: {type: DataTypes.STRING, allowNull: false},
})

const Type = sequelize.define('type', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    name: {type: DataTypes.STRING, unique: true, allowNull: false},
})

const Brand = sequelize.define('brand', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    name: {type: DataTypes.STRING, unique: true, allowNull: false},
})

const Rating = sequelize.define('rating', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    rate: {type: DataTypes.INTEGER, allowNull: false},
})
```

```
const DeviceInfo = sequelize.define('device_info', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    title: {type: DataTypes.STRING, allowNull: false},
    description: {type: DataTypes.STRING, allowNull: false},
})

const TypeBrand = sequelize.define('type_brand', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
})


User.hasOne(Basket)
Basket.belongsTo(User)

User.hasMany(Rating)
Rating.belongsTo(User)

Basket.hasMany(BasketDevice)
BasketDevice.belongsTo(Basket)

Type.hasMany(Device)
Device.belongsTo(Type)

Brand.hasMany(Device)
Device.belongsTo(Brand)

Device.hasMany(Rating)
Rating.belongsTo(Device)

Device.hasMany(BasketDevice)
BasketDevice.belongsTo(Device)

Device.hasMany(DeviceInfo, {as: 'info'});
DeviceInfo.belongsTo(Device)

Type.belongsToMany(Brand, {through: TypeBrand })
Brand.belongsToMany(Type, {through: TypeBrand })

module.exports = {
    User,
    Basket,
    BasketDevice,
    Device,
    Type,
    Brand,
    Rating,
    TypeBrand,
    DeviceInfo
}
```