МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ЗАСОБАМИ REACT ТА NODE.JS»**

Виконав: студент 4 курсу, групи 6.1219-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
М.О. Масюкевич
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, Phd, Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Масюкевичу Микиті Олексійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка інформаційної системи засобами React та Node.js

керівник роботи Столярова Анастасія Валеріївна, Phd
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація інформаційної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Кудін О.В., доцент кафедри програмної інженерії		
2	Кудін О.В., доцент кафедри програмної інженерії		
3	Кудін О.В., доцент кафедри програмної інженерії		

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	23.02.2023	
3.	Обробка методичних та теоретичних	16.03.2023	
	джерел.		
4.	Розробка першого та другого розділу.	20.04.2023	
5.	Розробка третього розділу.	17.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	22.06.2023	

Студент _____ М.О. Масюкевич
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка інформаційної системи засобами React та Node.js»: 59 с., 26 рис., 8 джерел, 5 додатків.

ЗАСТОСУНОК, ОНЛАЙН-МАГАЗИН, BACK-END, FRONT-END, NODE.JS, REACT.

Об’єкт дослідження – фреймворк Node.js бібліотеки для роботи з ним.

Мета роботи: розробка інтернет магазину для продажу техніки засобами Node.js та React.

Метод дослідження – проаналізувати предметні області, вивчення та узагальнення.

У кваліфікаційній роботі досліджено предметну область, засоби реалізації веб-додатку інтернет-магазину, обрано найкращий засіб реалізації та розроблено веб-додаток. Розглянуто основні особливості JavaScript, React та Node.js реалізовано базу даних у PostgreSQL.

Клієнтська та серверна частини були розділені з використанням різних технологій, клієнтська частина реалізована з використанням React, серверна частина реалізована з використанням Node.js та працює по принципу запит відповідь, реалізований зручний інтерфейс.

SUMMARY

Bachelor’s qualifying paper «Development of an Information System Using React and Node.js»: 59 pages, 26 figures, 8 references, 5 supplements.

APPLICATION, ONLINE-STORE, BACK-END, FRONT-END, NODE.JS, REACT.

Object of the study is framework Node.js library to work with it.

Aim of the study is development of an online store for the sale of equipment using Node.js and React.

Method of research is analyze subject areas, study and generalization.

In the qualification work, the subject area and means were investigated implementation of the web application of the online store, the best means of implementation was chosen and the web application was developed. The main features of JavaScript, React and Node.js were considered, and the database was implemented in PostgreSQL.

The client and server parts were separated using different technologies, the client part is implemented using React, the server part part is implemented using Node.js and works on the request principle response, a convenient interface is implemented.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології є невід'ємною складовою життя людей. Комп'ютери та програмне забезпечення призначені для зменшення людської участі у деяких повсякденних процесах та їх прискорення. Вони застосовуються майже в усіх сферах, починаючи з банківської сфери і лікарень, і закінчуючи розважальними закладами.

Зростає число випадків, коли працівники бізнесу, науки або державних установ бажають автоматизувати та скоротити людський вплив на процеси, що призводить до появи великої кількості програмного забезпечення.

Якість написання програмного забезпечення є ключовим фактором для іміджу та фінансових результатів замовника, а іноді й для життя людей. Тому, дуже важливо, щоб система, яку випускають на ринок, функціонувала бездоганно, була безпечною та відповідала всім вимогам, які поставив замовник.

Інформаційна система інтернет-магазин потрібна для здійснення онлайн-торгівлі, тобто продажу товарів чи послуг через Інтернет. Вона дозволяє підприємствам створювати власний електронний магазин, де клієнти можуть переглядати каталог товарів, здійснювати замовлення, оплачувати їх та отримувати доставку.

Також ця система допомагає знизити витрати на оренду торгової площі, утримання персоналу та інших витрат, що пов'язані з традиційними методами продажу, дозволяє розширювати аудиторію клієнтів за рахунок залучення покупців з інших міст чи країн.

Інтернет-магазин має ряд переваг, зокрема, швидке та зручне оформлення замовлень, автоматичне формування документів, можливість зберігати історію замовлень та додавати різноманітні зручні опції, такі як рейтинг товарів, відгуки клієнтів, програми лояльності та інше.

Кваліфікаційну роботу присвячено проєктуванню та розробці інформаційної системі інтернет-магазин для магазину електроприладів.

Метою дослідження є розробка інформаційної системи інтернет-магазин засобами React та Node.js.

Об’єктом дослідження є засоби React та Node.js.

Предметом дослідження є інформаційна система інтернет-магазин. У відповідності з метою дослідження ставляться такі завдання:

– дослідити предметну область та ознайомитись з функціоналом та вимогами до обраної інформаційної системи;
– розглянути найбільш популярні засоби;
– зробити проєктування інформаційної системи;
– розробити інформаційну систему інтернет-магазин засобами React та Node.js.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Поняття інформаційної системи

Інформаційні системи класифікують по цілому ряду різних ознак. Для побудови ефективних інформаційних систем важливо знати наступні принципи.

Принцип інтеграції. Даний принцип полягає в тому, що дані, котрі були вже введені в систему, багаторазово використовуються для вирішення сукупності завдання.

Принцип комплексності, який полягає в автоматизації усіх процедур трансформування даних на всіх етапах функціонування інформаційної системи.

Принцип системності, який полягає в обробці різних даних, для отримання інформації, яка необхідна для прийняття рішень.

Інформаційні системи розглядають, як звичайні програмні продукти, але такі системи відрізняються від звичайних прикладних систем. Інформаційні системи можуть суттєво відрізнятися за своїми функціями у залежності від сфери використання. Проте можна виділити декілька властивостей, які є спільними.

Інформаційні системи вирішують питання зберігання, обробки і збору інформації. Базою для таких систем є середовище зберігання даних.

Інформаційні системи мають бути орієнтовані на користувача, який неє експертом в області обчислювальної техніки. Клієнтські програмиінформаційної системи мають бути простим та зрозумілими для кінцевого користувача. Мати зручний, легко освоюваний інтерфейс, який надає всі потрібні функції для роботи кінцевого користувача і в той же час не дає йому можливість виконувати будь-які зайві дії.

1.2 Вимоги, що пред'являються до інформаційних систем

Перелік основних вимог, що пред'являються до інформаційних систем.

Гнучкість. Гнучкість системи є однією із головних критерій її вибору. Якщо компанія планує розвиватися та удосконалювати процеси діяльності або простофункціонує в умовах постійних змін внутрішніх та зовнішніх умов, то це є важливим критерієм. Здатність до адаптації передбачає можливість адаптування інформаційної системи до нових умов та потреб підприємства.

Надійність. Вимога надійності гарантуються створенням резервних копій інформації, що зберігається, а також використанням новітніх програмних і апаратних засобів, виконання операцій протоколювання, забезпеченням якості каналів зв'язку та фізичних носіїв інформації.

Ефективність. Система є ефективною, якщо вона дозволяє вирішувати потрібні завдання у мінімальні терміни. Ефективна інформаційна система є результатом оптимізації даних і методів їх обробки та застосування нетрадиційних методів розробок та проєктування.

Безпека. Під поняттям безпеки ІС, передбачається властивість системи, в силу якої сторонні особи не повині мати доступу до інформаційних системи організації. Вимога безпеки забезпечується сучасними засобами для розробки інформаційних систем, методами захисту інформації, безперервним моніторингом стану безпеки систем і засобів їх захисту.

1.3 Приклади інформаційних систем

Інформаційна система – це комплексний пакет програмних засобів, який призначений для збору, зберігання, обробки, передачі та використання інформації. Існують різноманітні інформаційні системи, оскільки вони використовуються у різних сферах діяльності людини. Нижче наведено декілька прикладів інформаційних систем з різних галузей.

ERP-системи (Enterprise Resource Planning) – це інформаційні системи, що використовуються в бізнесі для автоматизації та оптимізації процесів управління ресурсами підприємства. Наприклад, SAP, Microsoft Dynamics, Oracle E-Business Suite.

CRM-системи (Customer Relationship Management) – це інформаційні системи, які дозволяють підтримувати взаємодію з клієнтами та управляти відносинами з ними. Наприклад, Salesforce, HubSpot, Zoho CRM.

SCM-системи (Supply Chain Management) – це інформаційні системи, які використовуються для управління логістикою та поставками продукції. Наприклад, Oracle Supply Chain Management, SAP Supply Chain Management, JDA Software.

Медичні інформаційні системи – це інформаційні системи, що використовуються в медичній сфері для збору та обробки інформації про пацієнтів та їх лікування. Наприклад, Epic Systems, Cerner Corporation, McKesson Corporation.

Фінансові інформаційні системи – це інформаційні системи, що використовуються в фінансовій сфері для збору та обробки інформації про фінансові операції та інвестиції. Наприклад, Bloomberg Terminal, Thomson Reuters Eikon, FactSet.

Інтернет-магазин – це інформаційна система, що дозволяє здійснювати покупки товарів та послуг через Інтернет. Інтернет-магазини стають все популярнішими, оскільки дозволяють покупцям швидко та зручно знайти потрібний товар, порівняти ціни та замовити його, не виходячи з дому.

1.4 Засоби реалізації back-end частини.

Існує багато засобів для реалізації back-end частини, але найпопулярніші з них – це Node.js, PHP, Python, Ruby on Rails та Java.

Node.js – це відкрите середовище виконання JavaScript на стороні сервера,

яке дозволяє розробникам створювати високопродуктивні та масштабовані додатки. Він має вбудовану підтримку асинхронного програмування та є дуже популярним для створення веб-додатків та API.

PHP – це скриптова мова програмування, яка широко використовується для розробки back-end частини веб-додатків та сайтів. Вона дозволяє розробникам швидко створювати додатки та легко з’єднуватися з різноманітними базами даних.

Python – це інтерпретована скриптова мова програмування, яка часто використовується для створення back-end частини веб-додатків та великих систем. Вона має дуже розвинену екосистему бібліотек та фреймворків, що дозволяє розробникам швидко створювати додатки та спрощує роботу з базами даних.

Ruby on Rails – це фреймворк для створення веб-додатків, який базується на мові програмування Ruby. Він пропонує зручну архітектуру та високу продуктивність, а також широкі можливості для розробки веб-додатків.

Java – це мова програмування та платформа для створення високопродуктивних, масштакерування базами данихбованих та безпечних веб-додатків та сервісів. Вона до зволяє розробникам створювати масштабні додатки та легко підключатися до різноманітних баз даних.

PostgreSQL, MySQL та MongoDB – найпопулярніші системи керування.

1.5 Засоби реалізації front-end частини.

Front-end – це частина веб-додатку, яка відповідає за взаємодію з користувачем, його інтерфейс та візуальну складову. Для розробки front-end частини можна використовувати різні засоби та технології, серед яких основними є HTML, CSS та JavaScript.

HTML (Hypertext Markup Language) – це мова розмітки, яка використовується для створення структури веб-сторінки та відображення її

змісту. За допомогою HTML розмічаються елементи на сторінці, такі як текст, заголовки, таблиці, зображення та інші.

CSS (Cascading Style Sheets) – це мова стилів, яка використовується для задання зовнішнього вигляду веб-сторінки, такого як кольори, шрифти, розміри та положення елементів. CSS дозволяє розділити вміст та його вигляд, що робить код більш структурованим та зручним для редагування.

JavaScript – це скриптова мова програмування, яка використовується для створення динамічних елементів на веб-сторінці та взаємодії з користувачем. JavaScript дозволяє додавати веб-сторінці різноманітні функції, такі як анімація, валідація форм, розумні фільтри та інші.

Окрім основних засобів, існує безліч фреймворків та бібліотек, які допомагають розробникам зробити процес створення front-end частини більш швидким та ефективним. React, Angular та Vue.js – це три найбільш популярних фреймворки для розробки фронтенд-додатків. Кожен з них має свої переваги та недоліки, але загалом вони забезпечують ефективну та продуктивну розробку фронтенду.

React використовується для створення високопродуктивних, масштабованих та легко підтримуваних веб-додатків. Це робиться за допомогою віртуального DOM, який дає змогу підтримувати додаток в уніфікованому стані та забезпечувати ефективне оновлення відображення стану додатку на основі змін у даних. React забезпечує дуже гнучкий та розширюваний інтерфейс програмування (API), що дозволяє використовувати його з різноманітними бібліотеками та інструментами.

Angular від Google – це повний фреймворк, який забезпечує всі потрібні компоненти для створення розширюваного та масштабованого додатку, включаючи систему роутингу, підтримку тестування, залежності та впровадження. Angular має вбудовану систему обробки помилок, валідації та безпеки. Angular підтримує двостороннє зв’язування даних, що дозволяє автоматично оновлювати відображення даних після їх зміни.

Vue.js – це легковаговий фреймворк з підхідом «вид-модель», що дозволяє

розробникам зосередитися на створенні компонентів та швидкій розробці. Vue.js має просту та зрозумілу документацію, яка дозволяє швидко навчитися використовувати фреймворк. Vue.js дозволяє використовувати інтерполяцію, що спрощує виведення даних на сторінку, та підтримує використання шаблонів для компонентів. Однією з головних переваг Vue.js є те, що він пропонує зручну систему роботи з директивами, що дозволяє зменшити кількість коду та зробити розробку більш зрозумілою.

1.6 Аналіз вимог

Аналіз вимог до інформаційної системи інтернет-магазин є важливим етапом в розробці системи. На цьому етапі вимоги, зібрані на попередньому етапі, детально вивчаються і аналізуються з метою зрозуміти, як краще їх реалізувати та виконати потреби користувачів. Основна мета аналізу вимог – зрозуміти, що потрібно зробити для створення системи, яка повністю відповідає потребам користувачів та вимогам бізнесу.

У процесі аналізу вимог розробник визначає, які вимоги є важливими, які можуть бути змінені або відкинуті, а також які можливості системи є найбільш корисними та цінними для користувачів та бізнесу.

На основі аналізу вимог до інформаційної системи інтернет-магазин можна зробити такі висновки:

- система повинна бути зручною та простою для користувачів, що дозволить залучити більше клієнтів до магазину;
- система повинна бути масштабованою та готовою до розширення, щоб вона могла вміщати більше товарів та обробляти більше замовлень;
- система повинна бути надійною та безпечною, щоб запобігти можливості крадіжки даних користувачів та інших негативних наслідків;
- система повинна бути гнучкою та здатною пристосовуватися до різних потреб та вимог користувачів.

2 ПРОЄКТУВАННЯ

2.1 Цілі та потреби інформаційної системи інтернет-магазин

Основні цілі та потреби інформаційної системи інтернет-магазин можуть включати низку моментів. Розглянемо їх детальніше.

Продаж товарів: головною метою інформаційної системи інтернет-магазин є продаж товарів онлайн. Система повинна забезпечувати швидкий та безпечний процес замовлення товарів, оплати та доставки.

Управління товаром: система повинна забезпечувати зручне та ефективне управління асортиментом товарів, їхньою наявністю та цінами.

Керування замовленнями: система повинна забезпечувати зручне та ефективне керування замовленнями, від введення замовлення до його доставки.

Аналітика та звітність: система повинна забезпечувати аналіз даних про продажі, замовлення та поведінку користувачів для вдосконалення бізнес-процесів та планування стратегії розвитку магазину.

Комунікація з клієнтами: система повинна забезпечувати можливість комунікації з клієнтами через електронну пошту, телефон, чат або інші канали.

Захист даних: система повинна забезпечувати захист персональних даних користувачів, операцій з платежами та іншої конфіденційної інформації.

2.2. Проєктування інформаційної системи інтернет-магазин

Проєктування інформаційної системи для інтернет-магазину включає створення та організацію всіх необхідних компонентів, щоб забезпечити ефективну та безперебійну роботу магазину в онлайн-середовищі. Основна мета такої системи полягає у забезпеченні зручного і безпечного способу покупки товарів або послуг через Інтернет.

Нижче наведені ключові етапи проєктування інформаційної системи для інтернет-магазину.

Аналіз вимог: спочатку важливо визначити вимоги до системи. Це включає встановлення функціональних та нефункціональних вимог, визначення потреб клієнтів, вивчення конкуренції та встановлення основних цілей проєкту.

Проєктування бази даних: інтернет-магазин потребує бази даних для зберігання інформації про товари, клієнтів, замовлення та інше. Важливо розробити ефективну структуру бази даних, враховуючи потреби магазину та його масштабування в майбутньому.

Вибір технологій: обираються потрібні технології та інструменти для розробки системи. Це можуть бути веб-фреймворки, мови програмування, бази даних, платіжні шлюзи та інші компоненти, необхідні для реалізації функціональності магазину.

Розробка інтерфейсу: дизайн та розробка користувацького інтерфейсу має бути зручним та привабливим для клієнтів. Інтерфейс повинен дозволяти користувачам швидко знаходити товари, здійснювати пошук за категоріями або ключовими словами, переглядати детальну інформацію про товари.

Розробка функціональності: після визначення вимог та інтерфейсу розпочинається розробка функціональних модулів системи, таких як реєстрація та авторизація користувачів, каталог товарів, керування кошиком, оплата та обробка замовлень, система відгуків та оцінок, звіти і аналітика, інтеграція з платіжними шлюзами та доставками, адміністративний модуль тощо.

Тестування та впровадження: після розробки системи проводиться тестування для перевірки її функціональності, продуктивності та безпеки. Після успішного тестування система готова до впровадження. Здійснюється розгортання системи на веб-сервері, конфігурування необхідних сервісів, запуск магазину в робочому середовищі та підключення до доменного імені.

Підтримка та післяпродажне обслуговування: після впровадження важливо забезпечити підтримку та післяпродажне обслуговування системи. Це включає виправлення помилок, підтримку користувачів, оновлення

функціональності та безпеки, а також регулярні резервні копії бази даних та системи загалом.

Проєктування інформаційної системи для інтернет-магазину є складним процесом, який вимагає уваги до деталей, знань в області розробки програмного забезпечення та вміння визначати потреби та очікування користувачів. Правильне проєктування та реалізація інформаційної системи допоможуть покращити досвід покупців, забезпечити ефективну роботу магазину та сприяти розвитку бізнесу в онлайн-середовищі (див. рис. 2.1) [1].

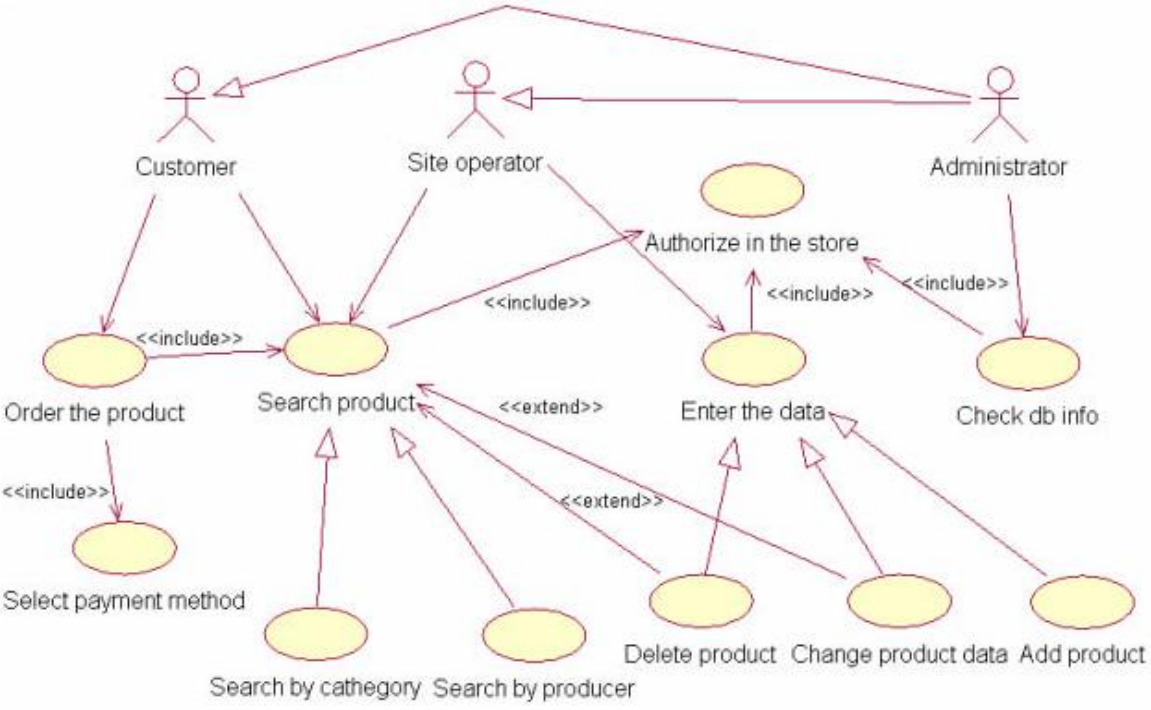

Рисунок 2.1 – Діаграма варіантів використання для інтернет-магазину

Клієнт запитує сторінку магазину.

Опис: клієнт ініціює запит на сторінку магазину, щоб переглянути доступні товари та послуги.

Дії: клієнт відправляє запит на сторінку магазину.

Веб-сайт магазину обробляє запит і повертає відповідь зі списком товарів і послуг.

Клієнт вибирає товар.

Опис: клієнт переглядає список товарів та обирає конкретний товар для

покупки.

Дії: клієнт обирає товар зі списку товарів, які були отримані від веб-сайту магазину.

Перевірка наявності товару.

Опис: система перевіряє наявність обраного клієнтом товару на складі.

Дії: веб-сайт магазину перевіряє наявність товару на складі, шляхом запиту до бази даних.

Веб-сайт магазину повертає відповідь клієнту з інформацією про наявність товару.

Додавання товару в кошик.

Опис: клієнт додає обраний товар в свій кошик для подальшої покупки.

Дії: клієнт вибирає опцію «Додати в кошик» для обраного товару.

Веб-сайт магазину додає обраний товар до кошика клієнта.

Перевірка доступних способів доставки.

Опис: система перевіряє доступні способи доставки для клієнта.

Дії: веб-сайт магазину запитує у системи доставки про доступні способи доставки для обраного товару та адреси клієнта.

Вибір способу доставки.

Опис: клієнт обирає певний спосіб доставки зі списку доступних варіантів.

Дії: клієнт переглядає список доступних способів доставки, який отримав від веб-сайту магазину.

Клієнт обирає певний спосіб доставки, який найбільше відповідає його потребам.

Підтвердження замовлення.

Опис: клієнт підтверджує своє замовлення та надсилає його для обробки.

Дії: клієнт підтверджує замовлення, переглядаючи підсумкову інформацію про товари, спосіб доставки та інші деталі.

Клієнт надсилає запит на підтвердження замовлення до веб-сайту магазину.

Запит на оплату.

Опис: клієнт ініціює запит на оплату замовлення.

Дії: клієнт відправляє запит на оплату замовлення до веб-сайту магазину.

Перенаправлення на платіжну систему.

Опис: веб-сайт магазину перенаправляє клієнта на платіжну систему для здійснення оплати замовлення.

Дії: веб-сайт магазину перенаправляє клієнта на платіжну систему, передаючи необхідну інформацію про замовлення та суму платежу.

Клієнт переходить на платіжну систему для завершення оплати.

Оплата замовлення.

Опис: клієнт здійснює оплату замовлення через платіжну систему.

Дії: клієнт вводить необхідні дані для оплати, такі як реквізити картки або інші деталі, в залежності від вибраної платіжної системи.

Платіжна система обробляє запит на оплату та здійснює транзакцію.

Платіжна система повертає підтвердження оплати до веб-сайту магазину.

Отримання підтвердження замовлення.

Опис: веб-сайт магазину отримує підтвердження про успішну оплату замовлення.

Дії: веб-сайт магазину отримує підтвердження оплати від платіжної системи.

Веб-сайт магазину оновлює статус замовлення на «оплачено».

Відправка підтвердження замовлення.

Опис: веб-сайт магазину відправляє клієнту підтвердження про успішне замовлення.

Дії: веб-сайт магазину генерує підтвердження замовлення, яке містить деталі замовлення, інформацію про доставку та інші відповідні дані.

Веб-сайт магазину відправляє підтвердження замовлення клієнту по електронній пошті або через повідомлення на веб-сайті.

Обробка замовлення.

Опис: веб-сайт магазину ініціює процес обробки замовлення.

Дії: веб-сайт магазину передає інформацію про замовлення до системи

обробки замовлень.

Система обробки замовлень перевіряє наявність товарів на складі, готує замовлення для доставки та оновлює статус замовлення.

Виконання доставки.

Опис: система обробки замовлень передає замовлення до служби доставки для виконання доставки клієнту.

Дії: система обробки замовлень передає деталі замовлення до служби доставки.

Служба доставки бере на себе відповідальність за доставку замовлення вказаному клієнтом місці.

Завершення замовлення.

Опис: замовлення вважається завершеним після успішної доставки та отримання клієнтом.

Дії: клієнт отримує замовлення від служби доставки.

Клієнт підтверджує отримання замовлення на веб-сайті магазину або через електронну пошту.

Веб-сайт магазину оновлює статус замовлення на «завершено».

Система обробки замовлень оновлює інформацію про стан замовлення.

Це опис кожного прецедента, які відображені на діаграмі послідовностей для Інтернет-магазину. Кожен прецедент представляє окрему дію або послідовність дій, які відбуваються між клієнтом, веб-сайтом магазину, системою обробки замовлень та службою доставки. Цей процес дозволяє клієнтам переглядати, вибирати, замовляти товари та послуги через Інтернет-магазин зручно та ефективно.

2.3 Проєктування дизайну у Figma

Figma – це онлайн-інструмент для дизайну інтерфейсів, який дозволяє створювати макети, прототипи та документацію. Він є одним з найпопулярніших

інструментів для дизайну, особливо для роботи в команді, тому що дозволяє взаємодіяти та спільно працювати над проєктами [2].

Розглянемо основні функції Figma.

Створення макетів та макетів прототипів: Figma дозволяє створювати макети сторінок інтерфейсів, використовуючи інструменти для рисування та макетування. Крім того, ви можете створювати прототипи, щоб продемонструвати роботу функцій та переходів між сторінками.

Робота в команді: Figma дозволяє створювати проєкти та додавати до них колег, щоб спільно працювати над дизайном. Користувачі можуть коментувати макети, залишати відгуки та вносити зміни у режимі реального часу.

Робота з бібліотеками та шаблонами: Figma містить бібліотеки елементів дизайну та шаблони, які можна використовувати для прискорення процесу розробки. Крім того, ви можете створювати власні бібліотеки та додавати до них елементи, щоб вони були доступні для повторного використання.

Експорт: Figma дозволяє експортувати макети та прототипи у різних форматах, таких як PNG, SVG, JPG, PDF та ін.

Збереження та синхронізація: всі дані зберігаються в хмарі, що дозволяє користувачам отримувати доступ до проєктів з будь-якого пристрою та в будь-який час. Крім того, Figma автоматично зберігає зміни та синхронізує їх між всіма користувачами, які працюють над проєктом. Це дозволяє уникнути втрати даних та забезпечує безпеку роботи з проєктами.

Інтеграції: Figma має багато інтеграцій з іншими інструментами, такими як Jira, Trello, Slack та багатьма іншими. Це дозволяє взаємодіяти з іншими інструментами, щоб спрощувати роботу з проєктами та зменшувати час, потрібний для завершення проєктів.

Figma є дуже потужним інструментом для дизайну інтерфейсів (див. рис. 2.2) та є популярним вибором для роботи в команді над проєктами. Він дозволяє створювати макети та прототипи, спільно працювати над проєктами, використовувати бібліотеки та шаблони, експортувати дані, зберігати та синхронізувати їх в хмарі, а також інтегрувати з іншими інструментами.

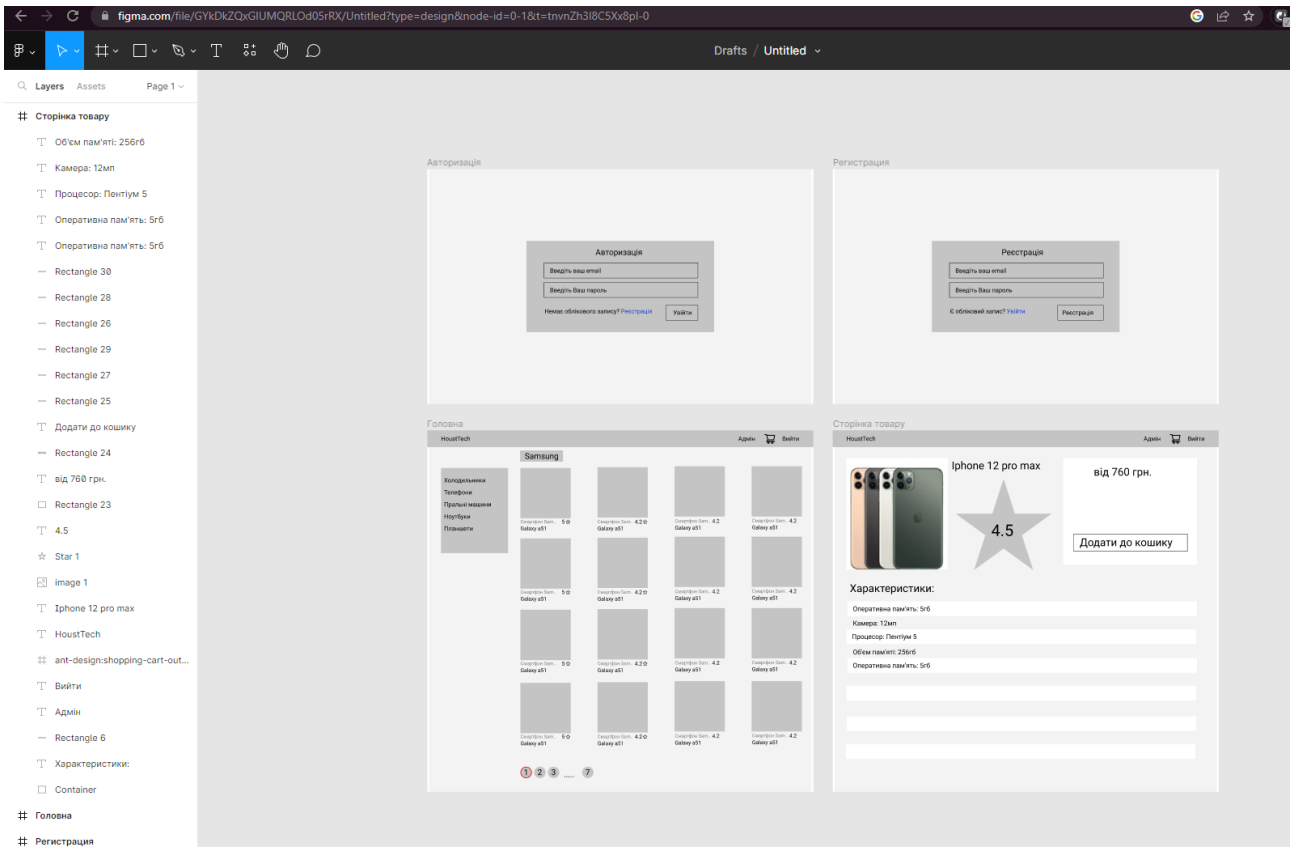

Рисунок 2.2 – Дизайн інтернет-магазину у Figma

2.4 Створення діаграми бази даних

Процес створення діаграми бази даних може бути розбитий на кілька етапів. Нижче наведено загальний опис кожного з етапів.

Визначення вимог. Першим етапом процесу створення діаграми бази даних є визначення вимог до бази даних. В цьому етапі необхідно збирати вимоги до даних, які будуть зберігатися в базі даних, та визначати зв’язки між ними.

Створення концептуальної моделі. Після визначення вимог до бази даних, необхідно створити концептуальну модель, яка відображає структуру даних та їх зв’язки між собою. Концептуальна модель має бути чіткою та логічною.

Створення логічної моделі. Після створення концептуальної моделі, наступним етапом є створення логічної моделі. Логічна модель відображає структуру даних та їх зв’язки між собою, але вже з урахуванням деталей бази даних, таких як типи даних та обмеження.

Створення фізичної моделі. Після створення логічної моделі, наступним етапом є створення фізичної моделі. Фізична модель відображає структуру даних та їх зв’язки між собою, але вже з урахуванням реалізації бази даних, таких як типи даних, індекси та ключі.

Розгортання бази даних. Після створення діаграми бази даних (див. рис. 2.3), наступним етапом є розгортання бази даних [3]. Це означає створення бази даних та таблиць, які відповідають структурі даних, визначеній на попередніх етапах.

Налагодження бази даних. Після створення та розгортання бази даних наступним етапом є її налагодження. Цей етап включає в себе створення індексів, визначення зв’язків між таблицями, а також налаштування доступу до даних. Крім того, налагодження бази даних може включати в себе оптимізацію запитів для поліпшення продуктивності бази даних.

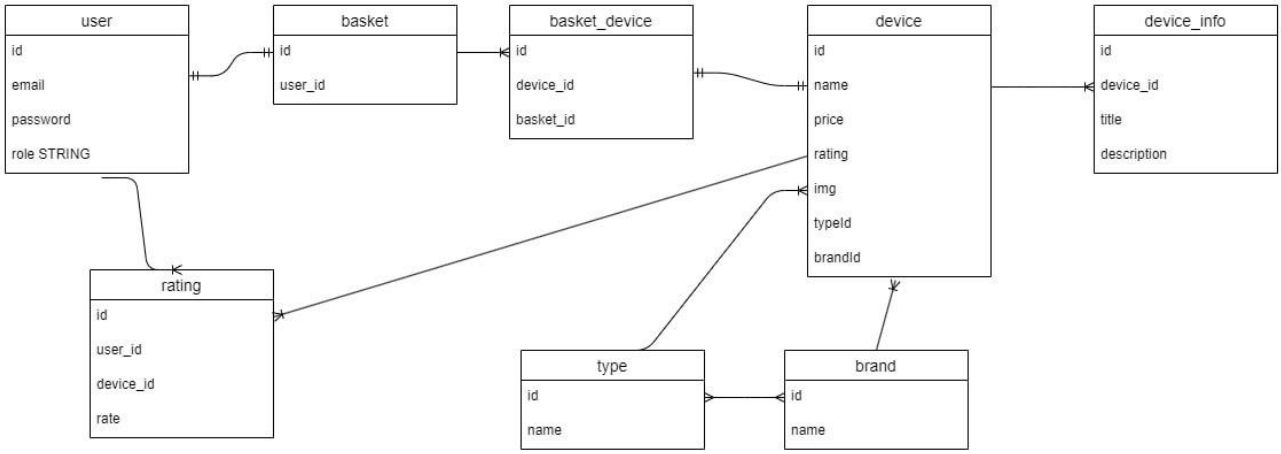

Рисунок 2.3 – Діаграма бази даних

У діаграмі бази даних можуть бути різні типи зв’язків між таблицями.

Один до одного (*One-to-One*). Кожен запис в одній таблиці має відповідний запис в іншій таблиці і навпаки. Такий тип зв’язку зазвичай використовується, коли інформація, яка відноситься до одного запису, міститься у двох окремих таблицях, для полегшення роботи з даними.

Один до багатьох (*One-to-Many*). Кожен запис в одній таблиці може мати багато відповідних записів в іншій таблиці, але кожен запис в іншій таблиці має

лише один відповідний запис у першій таблиці. Такий тип зв’язку використовується, коли один запис може мати багато пов’язаних даних, наприклад, замовлення в інтернет-магазині має багато товарів.

Багато до багатьох (*Many-to-Many*). Кожен запис в одній таблиці може мати багато відповідних записів в іншій таблиці, і навпаки. Такий тип зв’язку використовується, коли кожен запис може мати багато пов’язаних даних, і кожен запис у другій таблиці також може мати багато пов’язаних даних. Для забезпечення такого типу зв’язку необхідно використовувати додаткову таблицю-зв’язок, яка містить ключі обох таблиць.

3 РОЗРОБКА

3.1 Створення Back-end частини

Створення інтернет-магазину на Node.js може бути виконане в декілька етапів [4]. Розглянемо їх нижче.

Встановлення Node.js. Цей етап полягає в встановленні Node.js на вашому комп'ютері або сервері. Ви можете завантажити інсталятор Node.js з офіційного веб-сайту.

Вибір фреймворка. Є багато фреймворків, доступних для розробки веб-додатків на Node.js, таких як Express.js, Sails.js, Meteor.js, Koa.js та інші. Вам потрібно вибрати фреймворк, який найкраще підходить для вашого проєкту.

Налаштування бази даних. Ви можете використовувати різні бази даних, такі як MongoDB, MySQL, PostgreSQL, SQLite та інші. Вам потрібно встановити та налаштувати базу даних, щоб забезпечити збереження даних про товари, замовлення та іншу інформацію, необхідну для роботи магазину.

Розробка API. Для створення інтернет-магазину на Node.js ви можете розробити API для обміну даними між фронтендом та бекендом. Вам потрібно розробити API для додавання товарів до кошика, оформлення замовлення, оплати, доставки та інші дії, необхідні для роботи магазину.

Ініціалізація проєкту Node.js є необхідним етапом перед розробкою будь-якого веб-додатку на Node.js. Цей процес включає в себе створення файлів та каталогів, необхідних для проєкту, а також налаштування його середовища для розробки та випуску.

3.1.1 Ініціалізація проєкту

Основна мета ініціалізації проєкту Node.js полягає в налаштуванні його залежностей та структури проєкту. Залежності – це пакети, які потрібні для

функціонування вашого проєкту, такі як бібліотеки, фреймворки та інші засоби, які ви плануєте використовувати в проєкті. Встановлення та налаштування цих залежностей допоможе забезпечити правильне функціонування вашого проєкту.

Крім того, ініціалізація проєкту Node.js також дозволяє налаштувати вихідну структуру проєкту (див. рис. 3.1). Це може включати створення каталогів та файлів, необхідних для проєкту, таких як файли з конфігурацією, файли з даними, файли з логами та інші. Керування структурою проєкту допоможе зробити ваш код більш організованим та легко зрозумілим для інших розробників, які працюватимуть з проєктом в майбутньому.

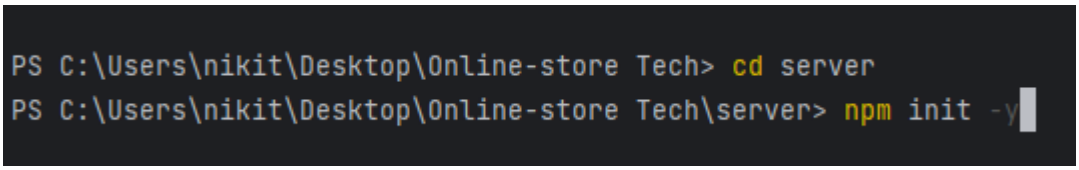

Рисунок 3.1 – Ініціалізація проєкту

Крім того, ініціалізація проєкту Node.js дозволяє налаштувати різні параметри проєкту, такі як середовище виконання, версії пакетів, сторонні бібліотеки та інші. Це допоможе забезпечити максимальну продуктивність та ефективність вашого проєкту на Node.js.

3.2 База даних

PGAdmin – це безкоштовний графічний інтерфейс для управління Системою Керування Базами Даних PostgresSQL (СКБД) [5]. Він дозволяє адміністраторам баз даних та розробникам зручно керувати і взаємодіяти з базами даних PostgresSQL.

Наведемо основні функції PGAdmin.

Створення та управління базами даних. PGAdmin дозволяє створювати бази даних, таблиці та інші об’єкти баз даних. Він також дозволяє редагувати та видаляти об’єкти баз даних.

Перегляд та редагування даних. PGAdmin дозволяє переглядати та редагувати дані в таблицях баз даних. Крім того, він підтримує редагування даних в режимі реального часу.

Виконання SQL-запитів. PGAdmin дозволяє виконувати SQL-запити на базі даних. Він підтримує виконання запитів на вибірку, вставку, оновлення та видалення даних.

Аналіз баз даних. PGAdmin дозволяє аналізувати бази даних та генерувати звіти про їх стан. Він надає різноманітні засоби для візуалізації даних та статистики про базу даних.

Управління користувачами та ролями. PGAdmin дозволяє керувати доступом користувачів до баз даних. Він підтримує створення ролей користувачів та надання їм прав доступу до об’єктів бази даних.

PGAdmin є потужним інструментом для роботи з базами даних PostgresSQL, який надає розширені функції управління базами даних та зручний інтерфейс користувача (див. рис. 3.2).

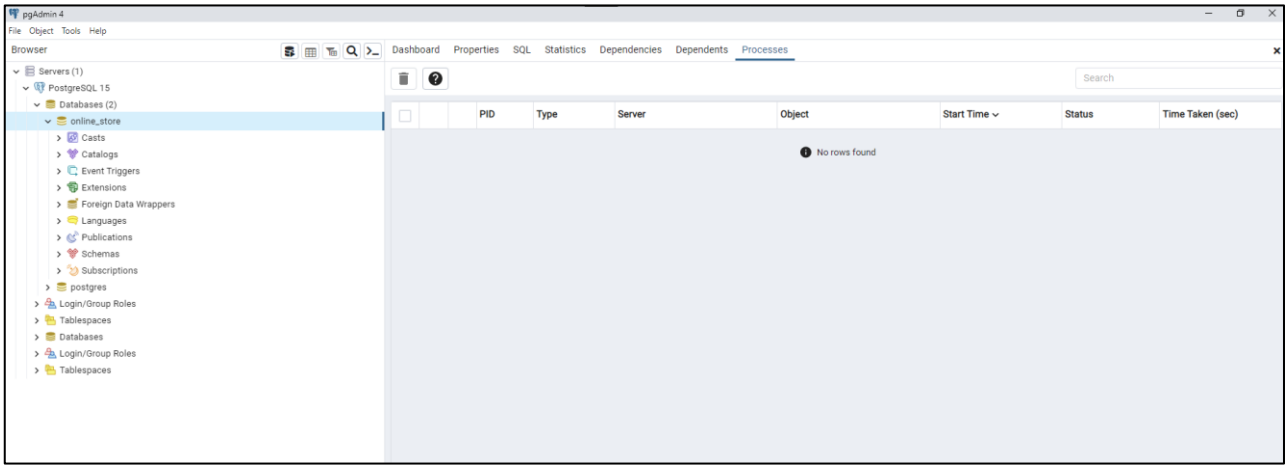

Рисунок 3.2 – Інтерфейс програми PGAdmin 4

3.2.1 Підключення бази даних

Підключення бази даних до серверу є ключовим етапом для забезпечення правильної роботи веб-додатку або програмного забезпечення. Щоб підключити

базу даних до серверу, необхідно виконати наступні кроки.

Крок перший. Встановіть та налаштуйте СКБД на сервері. Наприклад, для PostgresSQL необхідно встановити PostgreSQL та налаштувати файл конфігурації для дозволу зовнішніх з'єднань.

Крок другий. Створіть базу даних та користувача для підключення до бази даних. Користувач повинен мати достатній рівень дозволів для виконання необхідних операцій з базою даних.

Крок третій. Налаштуйте параметри підключення до бази даних у програмному коді. Наприклад, для підключення до бази даних PostgresSQL на Node.js, можна використовувати пакет node-postgres та налаштувати параметри підключення, такі як хост, порт, ім'я бази даних, ім'я користувача та пароль.

Крок четвертий. Перевірте підключення до бази даних. Для цього можна виконати простий запит до бази даних, наприклад, вибірку даних з таблиці. Якщо підключення працює правильно, то запит буде успішно виконано та будуть отримані необхідні дані.

Важливо забезпечити безпеку при підключенні бази даних до серверу. Необхідно захистити параметри підключення, забезпечити правильний рівень дозволів для користувачів та захистити систему від SQL-ін'єкцій та інших атак.

Для роботи з базою даних створимо файл db.js, в якому реалізуємо підключення до бази даних (рис. 3.3).

```
const {Sequelize} = require('sequelize')

module.exports = new Sequelize(
    process.env.DB_NAME, // Назва ДБ
    process.env.DB_USER, // Користувач
    process.env.DB_PASSWORD, // Пароль
    {
        dialect: 'postgres',
        host: process.env.DB_HOST,
        port: process.env.DB_PORT
    }
)
```

Рисунок 3.3 – Код файлу db.js

Для створення моделей бази даних (БД) у JavaScript можна використовувати різні бібліотеки та фреймворки, такі як Sequelize, Mongoose, Knex тощо. Розглянемо процес створення моделей у Sequelize.

Sequelize – це ORM-бібліотека, що дозволяє взаємодіяти з реляційними СКБД у зручний спосіб. Для створення моделей потрібно спочатку встановити Sequelize та драйвер для конкретної СКБД. Потім потрібно створити об’єкт Sequelize (див. рис. 3.4), вказавши параметри підключення до БД.

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: 'postgres'
});
```

Рисунок 3.4 – Приклад створення об’єкту Sequelize

Після цього можна створювати моделі, визначаючи їх поля та типи даних. Наприклад, створимо модель «User» з полями «id», «name» та «age» (див. рис. 3.5).

```
const User = sequelize.define('user', {
  id: {
    type: Sequelize.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  name: {
    type: Sequelize.STRING,
    allowNull: false
  },
  age: {
    type: Sequelize.INTEGER,
    allowNull: false
  }
});
```

Рисунок 3.5 – Приклад створення моделі «User»

У цьому прикладі ми використовуємо метод define для створення моделі «user». Параметр «user» вказує назву таблиці у БД, до якої буде прив’язана модель. Далі вказуємо поля моделі та їх типи даних. У полі «id» використовуємо параметри primaryKey та autoIncrement для автоматичної генерації унікального ідентифікатора. Параметр allowNull: false вказує, що поля «name» та «age» обов’язкові для заповнення.

Для взаємодії з БД у Sequelize використовуються методи моделі (див. рис. 3.6), наприклад, для створення нового запису в БД.

```
User.create({ name: 'John', age: 30 })
  .then(user => {
    console.log(user.toJSON());
  });
```

Рисунок 3.6 – Приклад метод «create» моделі «User»

У цьому прикладі ми використовуємо метод create моделі User для створення нового запису в БД з ім’ям «John» та віком 30.

3.3 Реєстрація користувачів

Система реєстрації користувачів у інтернет-магазині – це важлива складова частина інформаційної системи, що дозволяє користувачам створювати облікові записи, щоб здійснювати покупки на сайті. Розглянемо основні компоненти системи реєстрації користувачів.

Форма реєстрації: користувач заповнює форму з основними даними, які необхідні для створення облікового запису, такі як ім’я, прізвище, адреса електронної пошти, пароль тощо.

Валідація даних: валідація полягає у перевірці правильності введення користувачем даних. Наприклад, перевірка того, чи правильно введено

електронну адресу, чи відповідає введений пароль вимогам до довжини та складності.

Збереження даних: після того, як користувач заповнив форму реєстрації та пройшов валідацію, дані про нього зберігаються у базі даних. У більш складних системах можуть бути використані системи шифрування, щоб збереження даних користувачів було безпечним.

Авторизація: після створення облікового запису користувач може ввійти на сайт, використовуючи свої електронну адресу та пароль.

Забули пароль: якщо користувач забув свій пароль, він може натиснути на відповідну кнопку та ввести свою електронну адресу. Система надішле на вказану адресу листа з посиланням на форму зміни пароля.

Підтвердження електронної адреси: щоб забезпечити більшу безпеку та захист від спаму, деякі системи реєстрації вимагають підтвердження електронної адреси.

3.3.1 JSON Web Token

JWT (JSON Web Token) – це стандарт веб-токена, який використовується для автентифікації та авторизації користувачів [6]. JWT складається з трьох частин (див. рис. 3.7): заголовка, заяви та підпису. Заголовок та заява містять у собі закодовану інформацію про користувача та деякі дані про токен, які можуть бути використані для перевірки та підтвердження його автентичності. Підпис забезпечує захист від підробки.

Реєстрація через JWT токен передбачає наступні кроки:

– користувач вводить свої особисті дані для реєстрації;
– інформація про користувача зберігається у базі даних, а пароль – в захешованому вигляді;
– при вході в систему користувач вводить свій логін та пароль;
– сервер перевіряє правильність введеного логіна та пароля;

– якщо дані коректні, сервер генерує JWT токен та надсилає його на клієнтську сторону;
– клієнт зберігає отриманий JWT токен у локальному сховищі;
– при кожному наступному запиті до сервера, клієнт додає отриманий токен до заголовку запиту;
– сервер перевіряє підпис токену та достовірність інформації, що міститься в заяві;
– якщо перевірка успішна, сервер повертає запитані дані.

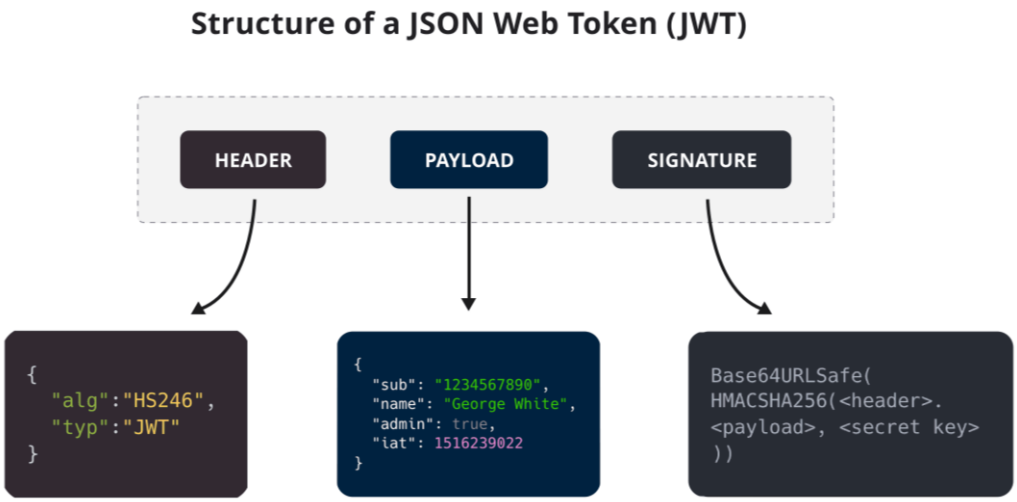

Рисунок 3.7 – Структура JWT

JWT токен дозволяє позбутися необхідності в збереженні стану сесії на сервері, що полегшує роботу з багатокористувацькими додатками, дозволяє створити розподілену систему, зменшує навантаження на сервер і забезпечує високу безпеку автентифікації.

Для реалізації системи автентифікації інсталюємо дві залежності «jsonwebtoken» та «bcrypt», для цього використаємо команду «npm install jsonwebtoken bcrypt».

«Jsonwebtoken» – це модуль, який дозволяє створювати та перевіряти JSON Web Tokens.

«bcrypt» – це модуль для хешування паролів у Node.js. Він використовує

бібліотеку bcrypt для створення хешів паролів, які можна зберігати в базі даних.

Хешування паролів – це процес перетворення пароля у незворотний формат, що забезпечує безпеку паролів користувачів, які зберігаються у базі даних (див. рис. 3.8).

```
1 usage
async registration(req, res, next) : Promise<any>  {
    const {email, password, role} = req.body
    if (!email || !password) {
        return next(ApiError.badRequest('Некорректный email или password'))
    }
    const candidate = await User.findOne({where: {email}})
    if (candidate) {
        return next(ApiError.badRequest('Пользователь с таким email уже существует'))
    }
    const hashPassword = await bcrypt.hash(password,  salt: 5)
    const user : Model<any, TModelAttributes>  = await User.create( values: {email, role, password: hashPassword})
    const basket : Model<any, TModelAttributes>  = await Basket.create( values: {userId: user.id})
    const token = generateJwt(user.id, user.email, user.role)
    return res.json({token})
}
```

Рисунок 3.8 – Код системи реєстрації

3.4 Створення Front-end частини

Створення Front-end частини веб-додатку за допомогою JavaScript та React включає в себе наступні кроки.

Створення проєкту React за допомогою create-react-app або аналогічного інструменту (див. рис. 3.9) [7].

```
PS C:\Users\nikit\Desktop\front-end> npx create-react-app .
```

Рисунок 3.9 – Створення проєкту React

Цей інструмент автоматично налаштує необхідні залежності та структуру проєкту (див. рис. 3.10).

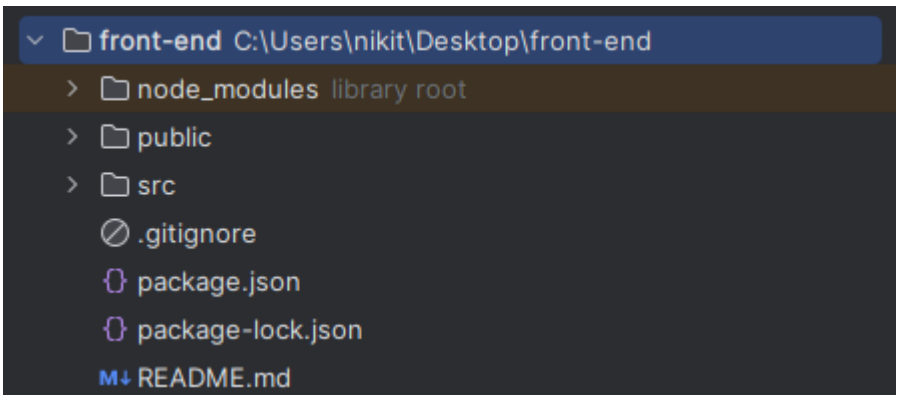

Рисунок 3.10 – Проєкт після виконання команди на рис. 3.9

Розробка компонентів, які відображатимуться на сторінці. Компоненти в React – це незалежні модулі, які містять логіку та представлення частини інтерфейсу користувача. Компоненти можна вкладати один в одного, утворюючи ієрархію компонентів.

Визначення стилів для компонентів. Для стилізації компонентів можна використовувати CSS, SASS або інші препроцесори CSS. Також можна використовувати бібліотеки стилів, такі як Bootstrap або Material UI.

Реалізація взаємодії між компонентами та зв’язок з серверною частиною додатку. Для зв’язку з сервером можна використовувати AJAX запити, Fetch API або бібліотеки, такі як Axios або jQuery.

Тестування та налагодження коду. Для тестування можна використовувати різні інструменти, такі як Jest, Enzyme або React Testing Library. Для налагодження коду можна використовувати DevTools у браузері.

Підготовка для публікації. Після завершення розробки Front-end частини додатку необхідно зібрати та оптимізувати код для публікації. Для цього можна використовувати інструменти, такі як Webpack або Parcel.

3.4.1 Інсталяція залежностей

Команда «npm install axios react-router-dom mobx mobx-react-lite» встановлює три пакети:

а) axios: це бібліотека, яка дозволяє виконувати запити до сервера за допомогою HTTP-протоколу (вона забезпечує простий та зрозумілий інтерфейс для взаємодії з API сервера);

б) react-router-dom: це бібліотека для навігації між сторінками в React додатках (вона забезпечує різноманітні способи навігації, включаючи роутинг на основі URL-адрес, використання історії браузера та багато іншого);

в) mobx та mobx-react-lite: ці дві бібліотеки дозволяють легко реалізувати патерн State Management в React додатках (mobx забезпечує зручний спосіб для створення та управління станом додатка, а mobx-react-lite робить його легко доступним для використання в React компонентах).

3.4.2 Створення сторінки авторизації та реєстрації

Створення Front-end сторінки реєстрації на сайті можна зробити за допомогою React та деяких додаткових бібліотек. Перед початком роботи необхідно встановити бібліотеки за допомогою команди «npm install».

Основним елементом сторінки реєстрації є форма, яка складається з декількох полів (рис. 3.11), наприклад, поле введення імені, електронної пошти, пароля та кнопки «Зареєструватися». Така сама форма для авторизації (рис. 3.12).

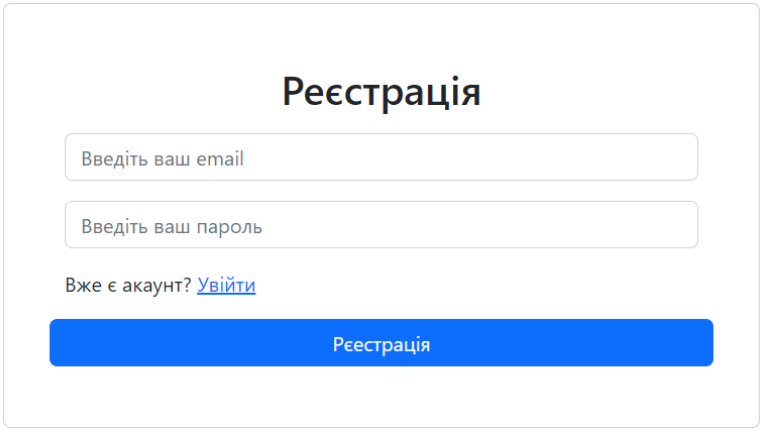

Рисунок 3.11 – Сторінка реєстрації

Щоб зробити цю форму, необхідно створити компонент React зі станом, який зберігає дані, введені користувачем в форму.

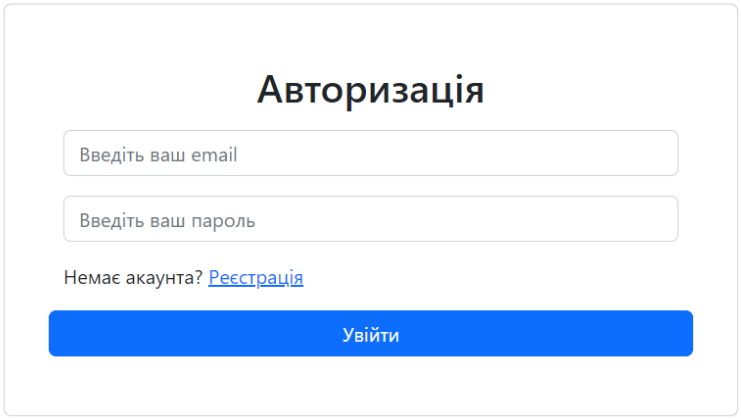

Рисунок 3.12 – Сторінка авторизації

Для зручності обробки форми можна використати бібліотеку axios, яка дозволяє відправляти запити на сервер. При натисканні на кнопку «Зареєструватися», потрібно створити обробник подій, який відправить дані з форми на сервер за допомогою axios.post().

При успішній реєстрації користувача на сервері буде створений запис у базі даних з введеними даними. Для збереження даних, які ввів користувач, можна використати сторонній стейт-менеджер, наприклад, MobX. Цей стейт-менеджер дозволяє зручно управляти даними, які зберігаються в програмі, та забезпечує їх автоматичне оновлення при зміні стану.

Також можна додати валідацію введених даних на стороні клієнта за допомогою бібліотеки react-validation. Ця бібліотека дозволяє перевіряти введені дані, наприклад, чи вони відповідають формату електронної пошти або чи містяться всі необхідні символи в паролі.

3.4.3 Створення переходу між сторінками

Для створення переходу між сторінками веб-сайту на Front-end можна використовувати бібліотеку React Router.

Перш за все, потрібно встановити React Router за допомогою команди npm install react-router-dom. Після встановлення можна створювати компоненти сторінок та визначати, як вони будуть відображатись при переході на них (див. рис. 3.13).

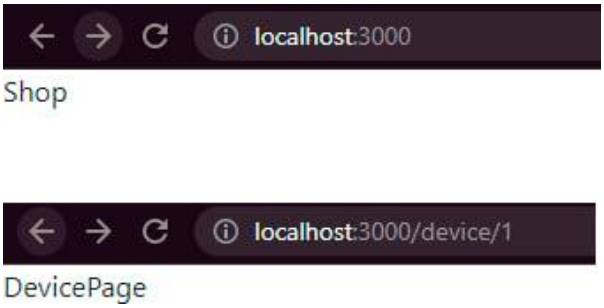

Рисунок 3.13 – Перехід між сторінками

Для цього необхідно імпортувати необхідні компоненти з React Router, такі як BrowserRouter для обгортання всього додатку та Route для визначення шляху до компоненту сторінки (рис. 3.14).

```
const AppRouter : FunctionComponent<object>  = observer( baseComponent: () => {
    const {user} = useContext(Context)

    return (
        <Switch>
            {user.isAuth && authRoutes.map(({path : string , Component}) =>
                <Route key={path} path={path} component={Component} exact/>
            )}
            {publicRoutes.map(({path : string , Component}) =>
                <Route key={path} path={path} component={Component} exact/>
            )}
            <Redirect to={SHOP_ROUTE}/>
        </Switch>
    );
});
```

Рисунок 3.14 – Реалізація переходу між сторінками

Наприклад, для створення переходу на сторінку «About» при кліку на кнопку «Про нас» можна написати наступний код (див. рис. 3.15).

```
import React from 'react';
import { BrowserRouter, Route, Link } from 'react-router-dom';
import About from './About';

function App() {
  return (
    <BrowserRouter>
      <nav>
        <ul>
          <li>
            <Link to="/">Головна</Link>
          </li>
          <li>
            <Link to="/about">Про нас</Link>
          </li>
        </ul>
      </nav>

      <Route path="/about">
        <About />
      </Route>
    </BrowserRouter>
  );
}

export default App;
```

Рисунок 3.15 – Приклад реалізації переходу між сторінками

У цьому прикладі ми використали компонент Link для створення посилання на сторінку «About», а також компонент Route для визначення шляху до компоненту About.

3.4.4 Створення головної сторінки

Створення головної сторінки інтернет-магазину може бути досить складним процесом, оскільки вона повинна бути привабливою та зручною для користувачів. Нижче наведено кілька кроків, які можна виконати для створення ефективної головної сторінки на сайті інтернет-магазину:

Розробка дизайну: перш за все, потрібно розробити дизайн головної

сторінки, що відповідає вашому бренду та продуктам. Розробка дизайну може включати в себе вибір кольорів, шрифтів, відступів та розміщення елементів.

Структура сторінки: наступним кроком є визначення структури сторінки. Головна сторінка (див. рис. 3.16) повинна містити важливу інформацію про ваші продукти та послуги, такі як категорії товарів, акції та новинки.

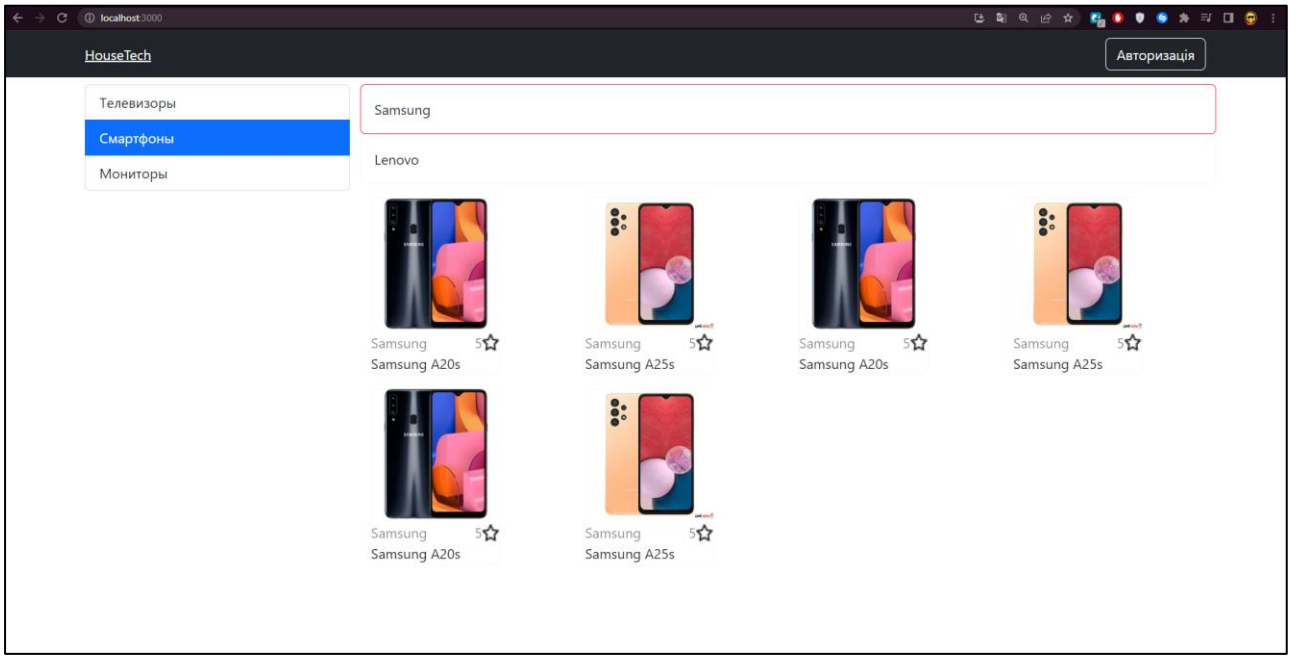

Рисунок 3.16 – Головна сторінка з товаром

Контент: наповнення головної сторінки контентом є ключовим етапом в створенні ефективної головної сторінки. Контент повинен бути корисним для користувачів, містити переваги вашого товару, пропозиції та інші цікаві матеріали, які зможуть зацікавити відвідувачів.

Функціональність: головна сторінка повинна мати деяку функціональність для зручності відвідувачів. Наприклад, можливість швидкого пошуку товарів, реєстрації або авторизації користувачів, додавання товарів до кошика та інше.

Адаптивність: важливо враховувати адаптивність головної сторінки на різних пристроях, таких як мобільні телефони, планшети та настільні комп’ютери [8]. Головна сторінка повинна бути оптимізована для різних екранів.

Адаптивність (Responsive design) – це процес розробки веб-сайту таким

чином, щоб він правильно відображався на різних пристроях, таких як мобільні телефони, планшети та настільні комп’ютери. Основною метою адаптивної веб-розробки є забезпечення зручної та ефективної навігації користувачів на всіх пристроях, що збільшує їх задоволення від відвідування сайту та знижує відсоток відмов.

Адаптивність досягається за допомогою використання технологій, які дозволяють змінювати розміри та розташування елементів веб-сторінки в залежності від розміру екрана пристрою, на якому вона відображається. Наприклад, ширина блоків та шрифтів може змінюватися в залежності від ширини екрана, а розташування меню може змінюватися в залежності від орієнтації екрану.

При розробці адаптивної веб-сторінки слід бути уважним до відображення контенту та функціональності на різних пристроях. Також важливо враховувати швидкість завантаження сторінки, особливо на мобільних пристроях з обмеженою швидкістю Інтернету. Всі ці фактори допомагають забезпечити позитивний досвід користувача та збільшити конверсії на сайті.

Створення сторінки окремого товару в інтернет-магазині на Node.js та React передбачає наступні кроки. Нижче пройдемось по кожному послідовно.

Створення макету сторінки товару – відображення основної інформації про товар, зображень, відгуків, ціни, розмірів, характеристик тощо. Для створення макету можна використовувати HTML та CSS.

Розробка API на Node.js – це дозволяє отримувати інформацію про товар з бази даних та передавати її на фронтенд за допомогою HTTP-запитів. API можна написати з використанням Express.js.

Створення компонента React – це дозволяє відображати інформацію про товар на фронтенді та забезпечити можливість користувача здійснювати замовлення товару. Для створення компонента можна використовувати JSX та бібліотеку React.

Налаштування маршрутів на Node.js – це дозволяє показувати правильну сторінку товару в залежності від URL-адреси, яку вводить користувач. Для цього

можна використовувати модуль React Router.

Підключення стилів до компонента React – це дозволяє забезпечити зовнішній вигляд компонента та взаємодію з користувачем. Для підключення стилів можна використовувати CSS або бібліотеки стилів, наприклад, Bootstrap або Material UI.

Підключення даних про товар через API – це дозволяє відображати актуальну інформацію про товар на сторінці. Для підключення даних можна використовувати бібліотеку Axios для виконання HTTP-запитів на сервер.

Додавання функціоналу для замовлення товару – це дозволяє користувачеві додати товар у кошик та здійснити замовлення. Для цього можна використовувати форми в React та API на Node.js для збереження замовлення

3.5 Тестування

Процес тестування інформаційної системи – це процес перевірки та оцінки роботи програмного забезпечення, яке входить до складу цієї системи. Його мета полягає в тому, щоб переконатися, що інформаційна система працює правильно, відповідно до вимог та очікувань користувачів.

Тестування інформаційної системи включає в себе створення тестових сценаріїв та даних для перевірки різних аспектів системи, таких як функціональність, надійність, продуктивність, безпеку, сумісність та інші. Ці сценарії виконуються з метою виявлення помилок та дефектів в програмному забезпеченні, що дозволяє їх виправити до того, як інформаційна система буде введена в експлуатацію.

Тестування інформаційної системи може проводитися на різних етапах її розробки та впровадження, включаючи модульне тестування, інтеграційне тестування та системне тестування. При цьому використовуються різні методи та підходи до тестування, такі як ручне тестування, автоматизоване тестування,

тестування методом чорного ящика та тестування методом білого ящика.

Після запуску у браузері відкривається localhost:3000, на якому ми можемо побачити головну сторінку нашого сайту (рис. 3.17).

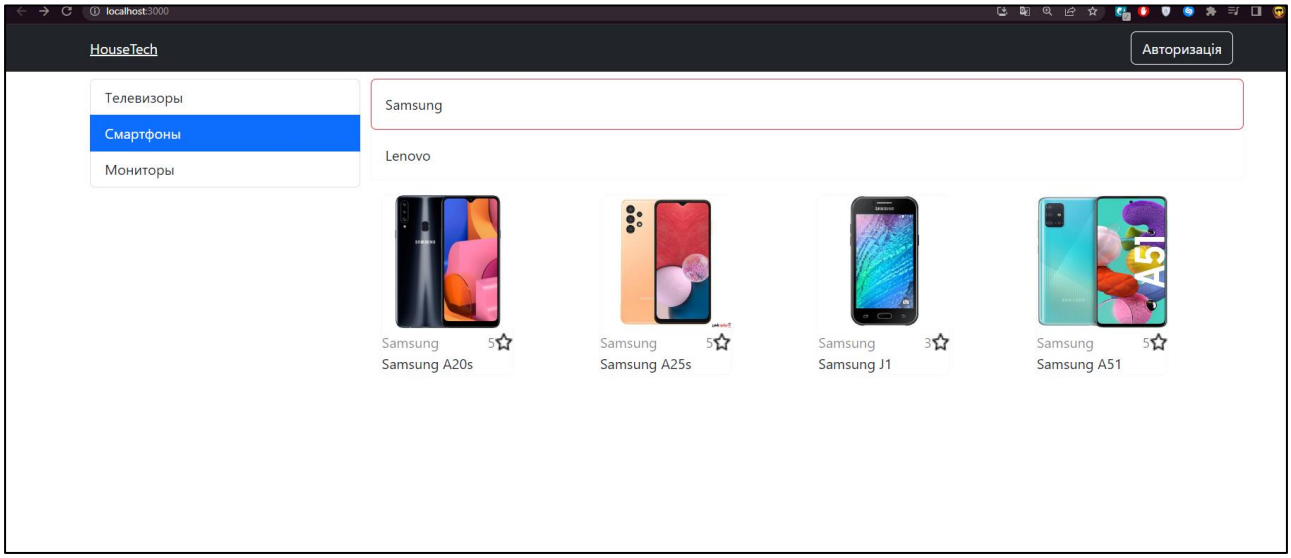

Рисунок 3.17 – Головна сторінка інтернет-магазину

На головній сторінці ми бачимо список товарів які є в нашому магазині. Як приклад видалимо декілька товарів та змінимо деякі (рис. 3.18).

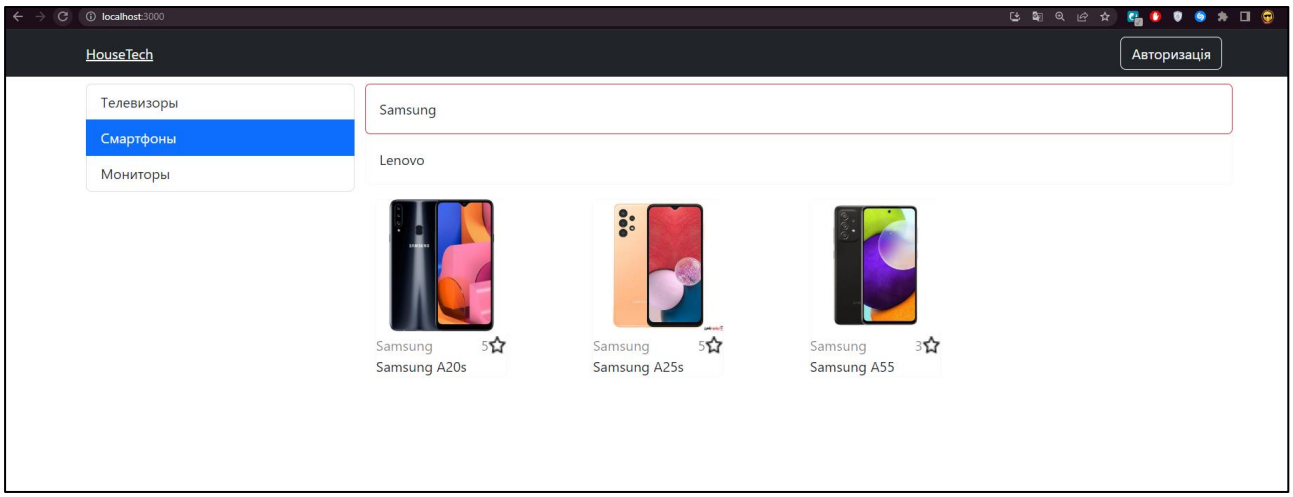

Рисунок 3.18 – Головна сторінка після зміни товарів

Після додавання нових товарів, натиснемо на товар який нас цікавить, та перейдемо на сторінки з ним (див. рис. 3.19).

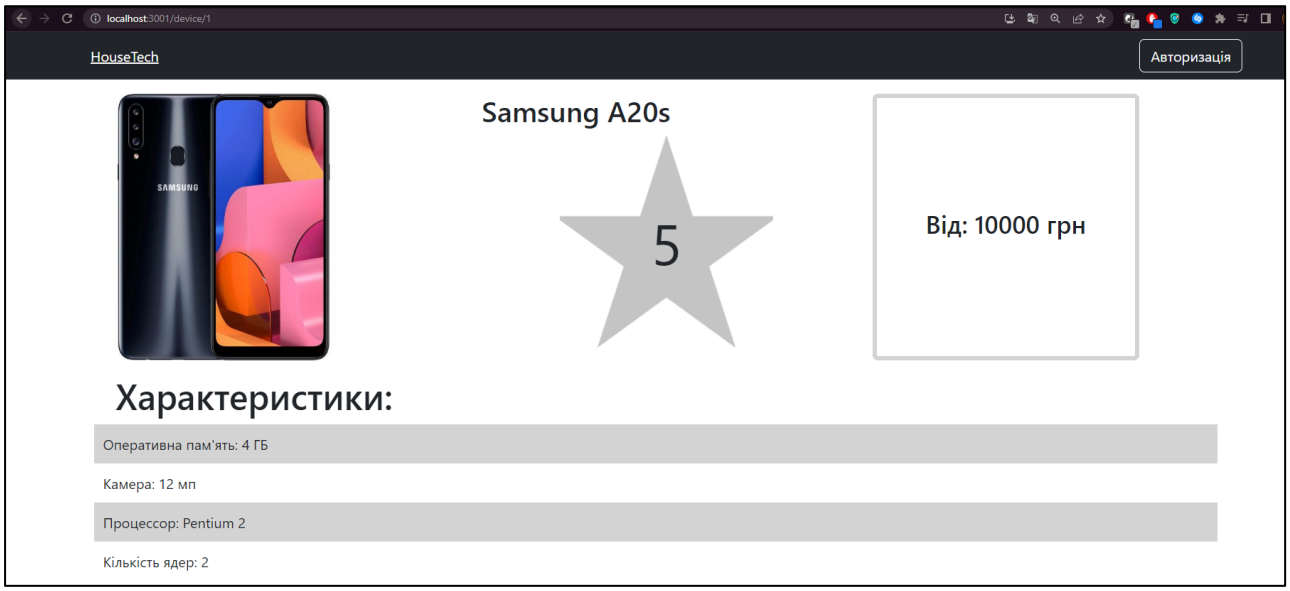

Рисунок 3.19 – Сторінка окремого товару

У кожного товару є своя назва, рейтинг, ціна та деякі характеристики стосовно цього товару, це все ми можемо подивитися на сторінці товару (рис. 3.20).

Також кожний товар має свій унікальний ідентифікатор (id), завдяки якому виконується перехід на сторінку певного товару.

```
this._devices = [
    {id: 1, name: 'Samsung A20s ', price: 10000,rating: 5, img:'https://cdn0.sellbe.com/p4/s-4102/product4/4436/622589.jpg'},
    {id: 2, name: 'Samsung A25s',price: 15000, rating: 5, img:'https://specifications-pro.com/wp-content/uploads/2023/04/Samsung-Galaxy-A25-600x600.jpg'},
    {id: 3, name: 'Samsung A55',price: 1000, rating: 3, img:'https://jasystore.com/wp-content/uploads/2021/08/samsung-galaxy-a54-1.jpg'},
]
```

Рисунок 3.20 – Уся інформація про товар у файлах

Після цього натиснемо на кнопку «Реєстрація», та пройдемо етап реєстрації на сайті (рис. 3.21).

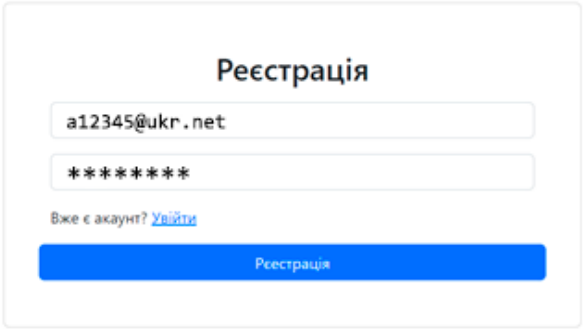

Рисунок 3.21 – Реєстрація на сайті

Після введення своїх даних у поле реєстрації, у консолі можемо побачити JWT-token (рис. 3.22) який буде використовуватися подалі для авторизації на нашому сайті.

```
> re.jwtDecode("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6InJlZGV2dG9vbHMiLCJpYXQiOjE1MTYyMzkwMjJ9.dad2JN0Jz9Yc
aLusKtt4xr6Fb5gpA9XMQof5r_7q8ug")
```

Рисунок 3.22 – Отримання JWT токену після реєстрації

Після авторизації перейдемо на головну сторінку (рис. 3.23).

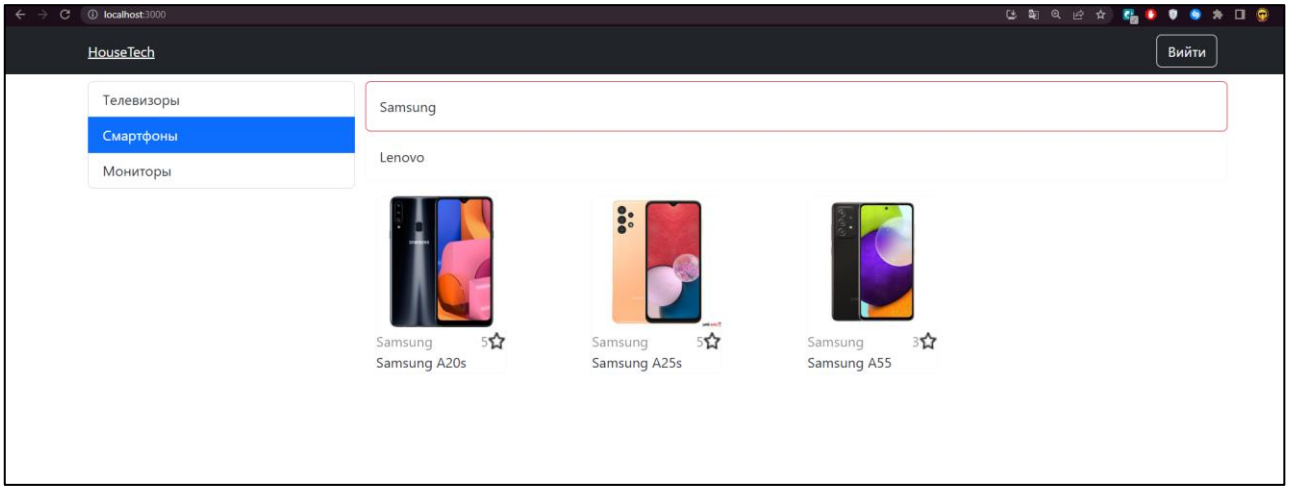

Рисунок 3.23 – Головна сторінка після авторизації

ВИСНОВКИ

Після створення інтернет-магазину на React та Node.js можна зробити наступні висновки.

React та Node.js – потужні та популярні технології, які використовуються для розробки сучасних веб-додатків, в тому числі й інтернет-магазинів.

Розробка на React та Node.js дозволяє створити високоякісний та швидкий інтернет-магазин з мінімальними затримками відповіді та високою продуктивністю.

React дозволяє розробникам розбити веб-додаток на компоненти, що дозволяє зручно організовувати роботу зі структурою сайту та зробити розробку більш ефективною.

Інтернет-магазин на React та Node.js може бути легко розширений та змінений, що дозволяє вам змінювати та доповнювати функціональність магазину під свої потреби.

Однак, розробка на React та Node.js вимагає певного рівня технічної компетенції, тому перед тим, як розпочати проєкт, важливо визначитися з рівнем своїх знань та можливостей.

Загалом, створення інтернет-магазину на React та Node.js дозволяє створити високоякісний та ефективний веб-додаток, який може бути успішним у комерційному плані. Однак, це потребує певної кількості знань та досвіду в розробці веб-додатків на цих технологіях.

ПЕРЕЛІК ПОСИЛАНЬ

1. Основи уніфікованої мови моделювання UML. Моделі і методи проектування інформаційних систем. URL: https://elearning.sumdu.edu.ua/free_content/lectured:de1c9452f2a161439391120eef364dd8ce4d8e5e/20151203140326/204841/index.html (дата звернення: 02.04.2023).
2. Онлайн ресурс Figma. Figma. URL: https://www.figma.com (дата звернення: 02.04.2023).
3. UML для бізнес-моделювання. Evergreens. URL: https://evergreens.com.ua/ua/articles/uml-diagrams.html (дата звернення: 04.04.2023).
4. Налаштування Node.js проекта. 8host. URL: https://www.8host.com/blog/nastrojka-proekta-node-js-pri-pomoshhi-typescript/ (дата звернення: 05.04.2023).
5. PGAdmin 4 guide. Serverspace. URL: https://serverspace.io/support/help/how-to-install-and-configure-pgadmin-4-in-server-mode-on-ubuntu-22-04/ (дата звернення: 07.04.2023).
6. Використання JWT-web token у веб-застосунку. highload.today. URL: https://highload.today/blogs/jwt-avtorizatsiya/ (дата звернення: 08.04.2023).
7. Getting started with React. legacy.reactjs. URL: https://legacy.reactjs.org/docs/getting-started.html (дата звернення: 10.04.2023).
8. Адаптивність веб сайту. Webtune. URL: https://webtune.com.ua/statti/internet-marketing/yak-pereviryty-adaptyvnist-za-dopomogoyu-brauzera/ (дата звернення: 10.04.2023).

ДОДАТОК А

Файл package.json

```
{
  "name": "front-end",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "axios": "^1.4.0",
    "bootstrap": "^5.2.3",
    "mobx": "^6.9.0",
    "mobx-react-lite": "^3.4.3",
    "react": "^18.2.0",
    "react-bootstrap": "^2.7.4",
    "react-dom": "^18.2.0",
    "react-router-dom": "^5.2.0",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
```

```
  "extends": [
    "react-app",
    "react-app/jest"
  ]
},
"browserslist": {
  "production": [
    ">0.2%",
    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
}
```

ДОДАТОК Б

Код Front-end частини авторизації на сайті

```
import React from 'react';
import {Container, Form} from "react-bootstrap";
import Card from "react-bootstrap/Card";
import Button from "react-bootstrap/Button";
import Row from "react-bootstrap/Row";
import {NavLink,useLocation} from "react-router-dom";
import {LOGIN_ROUTE, REGISTRATION_ROUTE} from "../utils/consts";

const Auth = () => {
    const location = useLocation()
    const isLogin = location.pathname === LOGIN_ROUTE
    return (
        <Container className="d-flex justify-content-center align-items-center"
        style={{height: window.innerHeight - 54}}
        >
            <Card style={{width:600}} className="p-5">
                <h2 className="m-auto">{isLogin ? 'Авторизація' : 'Рєестрація'}</h2>
            <Form className="d-flex flex-column">
                <Form.Control
                    className="mt-3"
                    placeholder="Введіть ваш email"
                />
                <Form.Control
                    className="mt-3"
                    placeholder="Введіть ваш пароль"
                />
```

```
                <Row className="d-flex justify-content-between mt-3">
                    {isLogin ?
                        <div>
                        Немає аккаунта? <NavLink to={REGISTRATION_ROUTE}>Реєстрація</NavLink>
                        </div>
                        :
                        <div>
                            Вже є аккаунт? <NavLink to={LOGIN_ROUTE}>Увійти</NavLink>
                        </div>
                    }
                    <Button
                        className = "mt-3 align-self-end"
                        varian={"outline-success"} >
                        {isLogin ? 'Увійти' : 'Рєестрація'}
                    </Button>
                </Row>

            </Form>
            </Card>
        </Container>
    );
};

export default Auth;
```

ДОДАТОК В

Код файл окремої сторінки товару в інтернет-магазині

```
import React from 'react';
import {Container, Image} from "react-bootstrap";
import Col from "react-bootstrap/Col";
import Row from "react-bootstrap/Row";
import bigStar from '../assets/bigStar.png'
import Card from "react-bootstrap/Card";
import Button from "react-bootstrap/Button";
const DevicePage = () => {
   const device = {id: 1, name: 'Samsung A20s', price: 10000,rating: 5, img:'https://cdn0.sellbe.com/p4/s-4102/product4/4436/622589.jpg'}
   const description = [
      {id:1, title: "Оперативна пам'ять", description: '4 ГБ'},
      {id:2, title: "Камера", description: '12 мп'},
      {id:3, title: "Процессор", description: 'Pentium 2'},
      {id:4, title: "Кількість ядер", description: '2'}
   ]
   return (
      <Container className='mt-3'>
         <Row>
         <Col md={4}>
            <Image width={300} height={300} src={device.img}/>
         </Col>
         <Col md={4}>
            <Row className='d-flex flex-column align-items-center'>
               <h2>{device.name}</h2>
               <div
```

```
                    className='d-flex align-items-center justify-content-center'
                    style={{background: url(${bigStar}) no-repeat center center, width:240, height:240, backgroundSize: 'cover', fontSize:64}}>
                    {device.rating}
                </div>
            </Row>
        </Col>
        <Col md={4}>
            <Card
                className='d-flex flex-column align-items-center justify-content-around'
                style={{width: 300, height: 300, fontSize: 32, border: '5px solid lightgray'}}
                    >
                <h3>Від: {device.price} грн </h3>
                <Button variant={'outline-dark'}>Додати до кошику</Button>
            </Card>
        </Col>
        </Row>
        <Row className="d-flex flex-column m-3">
            <h1>Характеристики:</h1>
            {description.map((info,index) =>
                <Row key={info.id} style={{background: index % 2 === 0 ? 'lightgray' : 'transparent', padding: 10}}>
                    {info.title}: {info.description}
                </Row>)}
        </Row>
      </Container>
   );
};
export default DevicePage;
```

ДОДАТОК Г

Код файлу у якому реалізовані моделі бази даних

```
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
}
```

ДОДАТОК Д

Код файлу у я кому реалізовано Back-End частину реєстрації та авторизації

```
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('Некорректный email или password'))
        }
        const candidate = await User.findOne({where: {email}})
        if (candidate) {
            return next(ApiError.badRequest('Пользователь с таким email уже существует'))
        }
        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('Пользователь не найден'))
        }
        let comparePassword = bcrypt.compareSync(password, user.password)
        if (!comparePassword) {
            return next(ApiError.internal('Указан неверный 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()
```