Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

ІНТЕРНЕТ-МАГАЗИН З ВИКОРИСТАННЯМ REACT.JS

ONLINE STORE USING REACT.JS

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗ-41
Мороз Роман Вікторович
_____
(підпис)
Керівник:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«8» 06 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: «*Інженерія програмного забезпечення*»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«*02*» *02* 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Мороз Роман Вікторович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: Інтернет-магазин з використанням React.js

Керівник роботи: к.т.н., доцент Ліщина Наталія Миколаївна
затверджені наказом закладу вищої освіти від «*23*» *грудня 2022* р. № *961-01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«14» червня 2023 р.*
3. Вихідні дані до роботи: *технічне завдання, технічне та програмне забезпечення, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу, Мова програмування JavaScript*
4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Актуальність використання React JS у ВЕБ застосунках. Мета роботи та галузь застосування. Аналіз, визначення вимог до розроблюваного програмного забезпечення та його проектування. Реалізація проекту з використанням Node JS. Тестування та валідація застосунку. Висновки.

5. Перелік графічного матеріалу:
Одинадцять рисунків

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Ліщина Н. М.*		
Специфікація вимог до розробленої системи	*Ліщина Н. М.*		
Розробка об'єкта проектування	*Ліщина Н. М.*		
Нормоконтроль	*Ліщина Н. М.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	6,47%		
Академічна доброчесність	*Яцук А.А.*		

7. Дата видачі завдання *«2» лютого 2023 р.*

КАЛЕНДАРНИЙ ПЛАН

№;№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
11	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*05.02.2023 р.*	
22	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2023 р.*	
33	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*10.03.2023 р.*	
44	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2023 р.*	
55	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2023 р.*	
66	*Тестування та налагодження об'єкта проектування*	*01.06.2023 р.*	
77	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*14.06.2023 р.*	

Здобувач вищої освіти _____ *Мороз Р. В.*
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ *Ліщина Н. М.*
(підпис) (прізвище, ініціали)

Рецензія
на кваліфікаційну роботу

Здобувач вищої освіти: Мороз Роман Вікторович
Спеціальність і група: 121 Інженерія програмного забезпечення, ІПЗ-41
Обсяг кваліфікаційної роботи бакалавра: 43 стор., 11 рис., 10 лістингів, 10 літературних джерел
Кількість сторінок записки: 43 сторінок

Тема: ***Інтернет-магазин з використанням React.js мереж***

Коротка характеристика кваліфікаційної роботи та прийнятих рішень:
Кваліфікаційна робота бакалавра складається з трьох розділів, в яких проаналізовані проблематика та поставлені завдання за темою роботи, здійснено теоретичне дослідження та практичну реалізацію інтернет-магазин з використанням React.js мереж.

Самостійні розробки і пропозицій автора: *робота спрямована на розробку сучасного та ефективного онлайн-магазину, що надасть користувачам зручність, швидкість та високу функціональність. Цей проект враховує зміни в інтернет-поведінці споживачів та використовує передові технології, щоб відповідати їхнім потребам та очікуванням.*

Недоліки: *Істотних недоліків в роботі не виявлено.*

Загальний висновок: *У кваліфікаційній роботі бакалавра було спроектовано та створено Інтернет-магазин з використанням React.js мереж. Робота є результатом самостійного дослідження. Істотних недоліків не виявлено. Кваліфікаційна робота здобувача освіти Мороза Романа рекомендується до захисту екзаменаційній комісії та заслуговує відмінної оцінки.*

Рецензент: _____ д. пед. наук, професор
(прізвище, ім'я, по-батькові, посада)

Рецензент кваліфікаційної роботи _____ професор каф. комп. наук Туманова Ю.О.
(підпис)

«8» 06 2023 р.

АНОТАЦІЯ

Мороз Р. В. Створення інтернет-магазину з використанням передових технологій React JS. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення», спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2023.

Ця кваліфікаційна робота присвячена розробці інтернет-магазину з використанням технології React JS. Робота має на меті створити сучасний та ефективний онлайн-магазин, використовуючи технології, які дозволяють покращити швидкість, функціональність та користувацький досвід.

У розділі аналізу та визначення вимог до розроблюваного проекту розглядається актуальність створення онлайн-магазину у сучасному цифровому середовищі, а також встановлюються основні вимоги до його функціональності, дизайну та безпеки.

У наступному розділі проводиться проектування програмного забезпечення, включаючи детальне вивчення технологій React JS та Deno JS, а також розробку архітектури системи. Приділяється увага інтерфейсному дизайну, взаємодії з користувачем та оптимізації швидкості завантаження сторінок. Для забезпечення гнучкості та розгортання проекту використовується Docker.

Після реалізації виконується тестування та валідація, щоб перевірити коректність функціонування системи, виявити та усунути помилки. Здійснюється перевірка безпеки, швидкодії та масштабованості системи з метою забезпечення оптимального досвіду користувачів.

Ключові слова: онлайн-магазин, React JS, Deno JS, архітектура програмного забезпечення, інтерфейсний дизайн, Docker, тестування, безпека.

ANNOTATION

Moroz R.V. Creation of an online store using technology React.js. Manuscript.

Bachelor's qualification work in Software Engineering, specialization – Software Engineering. Lutsk National Technical University. Lutsk, 2023.

This qualification work is dedicated to the development of a next-generation online store using advanced technologies React JS and Deno JS. The objective of this work is to create a modern and efficient online store by leveraging technologies that enhance speed, functionality, and user experience.

The analysis and requirements section examines the relevance of creating an online store in the current digital environment and establishes the main requirements for its functionality, design, and security. It investigates previous solutions in e-commerce and identifies features that can provide a competitive advantage to the proposed online store.

The subsequent section focuses on the software design, including a detailed study of React JS and Deno JS technologies and the development of the system architecture. Attention is given to interface design, user interaction, and optimizing page loading speed.

Docker is utilized to ensure flexibility and deployment of the project, simplifying the deployment process on the chosen hosting platform using render.io.

After implementation, testing and validation are conducted to verify the correctness of the system's functionality, identify and rectify errors. Security, performance, and scalability of the system are also assessed to provide an optimal user experience. The obtained results are used for refinement and improvement of the system, taking into account the data gathered during the testing process. Additionally, the online store is prepared for deployment on the selected hosting platform, ensuring its proper functioning and scalability.

Keywords: online store, React JS, Deno JS, software architecture, interface design, Docker, testing, security.

ЗМІСТ

ВСТУП

У сучасному світі інформаційні технології є невід'ємною частиною нашого повсякденного життя. Вони проникають у різні сфери, забезпечуючи нові можливості та покращення у багатьох галузях. Однією з таких популярних областей розробки програмного забезпечення є створення інтернет-магазинів, які є необхідними для сучасної комерції. Відкриття власного інтернет-магазину стає привабливою можливістю для багатьох підприємців, оскільки це дозволяє залучати клієнтів з усього світу та забезпечувати постійний доступ до продуктів і послуг.

Метою цієї кваліфікаційної роботи бакалавра є створення інтернет-магазину нового покоління з використанням передових технологій React JS та Deno JS. Робота спрямована на розробку сучасного та ефективного онлайн-магазину, що надасть користувачам зручність, швидкість та високу функціональність. Цей проект враховує зміни в інтернет-поведінці споживачів та використовує передові технології, щоб відповідати їхнім потребам та очікуванням.

Об'єктом дослідження є процес створення інтернет-магазину з використанням передових технологій React JS та Deno JS. Це включає вивчення та аналіз різних аспектів розробки, від проектування архітектури та інтерфейсу до інтеграції з базою даних та розгортання на хостингу. Робота також досліджує можливості використання Docker для контейнеризації додатку, що сприяє зручності установки та розгортання.

Предметом дослідження є розробка інтернет-магазину нового покоління, який використовує передові технології React JS та Deno JS. В роботі детально досліджуються принципи та методи створення ефективного та функціонального онлайн-магазину, а також аналізуються особливості роботи з базою даних на мові SQL за допомогою PostgreSQL. Основна увага приділяється розробці інтерфейсу, оптимізації продуктивності та забезпеченню безпеки додатку.

Завданнями на дану роботу є:

– визначення вимог до функціональності та дизайну інтернет-магазину;

– розробка архітектури програмного забезпечення та вибір необхідних технологій;

– реалізація функціональних модулів, включаючи обробку замовлень, оплату, пошук товарів тощо;

– інтеграція з базою даних та забезпечення безпеки персональних даних користувачів;

– тестування та валідація розробленого додатку для перевірки коректності його функціонування;

– підготовка інтернет-магазину до запуску на обраному хостингу за допомогою render.io.

Для досягнення поставлених цілей в роботі оцінюється сучасний стан проблеми створення інтернет-магазинів та аналізуються світові тенденції вирішення подібних задач. Враховуючи особливості сучасного ринку, розробка базується на передових технологіях React JS та Deno JS, що забезпечує більшу ефективність та зручність у роботі з додатком.

Ця кваліфікаційна робота також встановлює взаємозв’язок з іншими дослідженнями та роботами у сфері розробки інтернет-магазинів. Вона буде використана як підґрунтя для подальших досліджень та розробок у галузі електронної комерції та впровадження передових технологій у сфері онлайн-торгівлі.

Загальною метою цієї роботи є створення ефективного та сучасного інтернет-магазину з використанням передових технологій React JS та Deno JS, що відповідає сучасним потребам споживачів і сприяє розвитку електронної комерції.

РОЗДІЛ 1
АНАЛІЗ ТА ПОСТАНОВКА ЗАВДАНЬ ДО ФУНКЦІОНАЛЬНОСТІ ТА ДИЗАЙНУ ІНТЕРНЕТ-МАГАЗИНУ

1.1 Аналіз сучасного стану проблеми

У сучасному світі електронна комерція займає велике значення, а інтернет-магазини стають невід'ємною частиною покупкового процесу для багатьох людей. Завдяки зростанню популярності та доступності Інтернету, все більше споживачів звертаються до онлайн-платформ для придбання товарів та послуг. Однак, зміна вимог споживачів та зростання конкуренції серед інтернет-магазинів ставлять перед розробниками вимоги щодо швидкості, функціональності та дизайну.

Проведений аналіз сучасного стану проблеми включав збір та аналіз доступних даних про ринок електронної комерції та його тенденції. Дослідники вивчали досвід успішних інтернет-магазинів та їхній підхід до функціональності та дизайну. Також були ідентифіковані основні проблеми, з якими зіштовхуються розробники інтернет-магазинів та споживачі. Це допомогло визначити набір ключових функцій та вимог, які необхідно врахувати під час розробки нового інтернет-магазину.

Під час аналізу сучасного стану проблеми, дослідники звертають увагу на такі аспекти:

– розширені можливості електронної комерції – споживачі сьогодні очікують широкого спектру функцій у інтернет-магазинах, таких як швидке й зручне оформлення замовлень, зручні способи оплати, відстеження статусу замовлення, персоналізований підхід тощо. Розробники повинні бути готові задовольнити ці вимоги та впровадити необхідний функціонал у свій інтернет-магазин;

– зростання використання мобільних пристроїв диктує потребу у мобільно-дружніх інтернет-магазинах. Важливо мати адаптивний дизайн, що

дозволяє зручно переглядати та купувати товари на різних пристроях з різними розмірами екранів;

– забезпечення легкості використання та інтуїтивно зрозумілого інтерфейсу є ключовим аспектом успішного інтернет-магазину. Споживачі хочуть знайти потрібний товар швидко і з легкістю, отримати достатню інформацію про товар, а також мати зручні інструменти для навігації та фільтрації;

– ефективний інтернет-магазин повинен мати привабливий зовнішній вигляд, який відповідає бренду компанії. Дизайн магазину повинен створювати позитивне враження і привертати увагу споживачів.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою цієї кваліфікаційної роботи бакалавра є розробка інтернет-магазину нового покоління, який відповідатиме сучасним вимогам споживачів та буде конкурентоздатним на ринку електронної комерції. Головною метою розробки є створення функціонального, зручного в використанні та естетичного інтернет-магазину, який надасть користувачам зручність та ефективність під час покупок онлайн.

Об’єктом дослідження є інтернет-магазин нового покоління, а предметом дослідження – вимоги до його функціональності та дизайну.

Завдання на кваліфікаційну роботу бакалавра включають:

– визначення основних функціональних вимог до інтернет-магазину. Перед розробниками стоїть завдання визначити необхідний функціонал, який забезпечить користувачам зручність та ефективність під час покупок. Це включає можливість перегляду товарів, додавання їх до кошика, оформлення замовлення, обробка платежів, реалізація системи облікових записів користувачів та інші функції;

– аналіз вимог до дизайну інтернет-магазину. Дизайн грає важливу роль у привабливості інтернет-магазину. Розробники повинні провести аналіз вимог

споживачів та розробити дизайн, який буде відповідати їхнім потребам та очікуванням. Це включає розробку привабливого інтерфейсу, зручну навігацію, відповідність бренду компанії тощо;

– вибір та налаштування відповідних технологій та інструментів. Для реалізації інтернет-магазину необхідно вибрати відповідні технології та інструменти розробки. Залежно від вимог інтернет-магазину можуть використовуватися різні технології для фронтенду та бекенду. Наприклад, в даній роботі використовуються технології React JS для фронтенду та Deno JS для бекенду;

– розробка та імплементація інтернет-магазину. На основі визначених вимог до функціональності та дизайну розробляється імплементація інтернет-магазину. Це включає розробку фронтенду, бекенду, налаштування бази даних, реалізацію функцій покупки, оплати, обробки замовлення та інші необхідні функції;

– тестування та валідація інтернет-магазину. Після розробки проводиться тестування системи з метою виявлення та виправлення помилок. Валідація інтернет-магазину включає перевірку відповідності вимогам та перевірку коректності роботи системи;

– документування розробки та написання звіту. Весь процес розробки імплементації інтернет-магазину, включаючи вимоги, дизайн, технології, реалізацію, тестування та валідацію, повинен бути задокументований у вигляді звіту.

Висновки до розділу 1

В результаті виконання цієї кваліфікаційної роботи бакалавра очікується створення функціонального та естетичного інтернет-магазину нового покоління, який відповідатиме сучасним вимогам споживачів. Результати роботи будуть представлені у вигляді звіту, який включатиме опис вимог, аналіз дизайну, вибір технологій, реалізацію системи, тестування та валідацію.

Очікується, що розроблений інтернет-магазин буде готовим до розгортання на сервері та використання споживачами для здійснення покупок онлайн.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

У цьому розділі проведено аналіз та визначення вимог до розроблюваного інтернет-магазину нового покоління. З метою забезпечення зручності та ефективності користувачів під час покупок було визначено основні функціональні вимоги, які повинні бути виконані системою.

Вони включають наступне:

– можливість перегляду товарів;

– додавання товарів до кошика;

– система облікових записів користувачів.

Можливість перегляду товарів передбачає, що інтернет-магазин має мати інтерфейс, що дозволяє користувачам переглядати доступні товари. Це може бути реалізовано через каталог товарів, де товари будуть категоризовані і представлені у зручному форматі для перегляду. Користувачі також повинні мати можливість використовувати пошукову функцію, щоб знайти конкретний товар за назвою або іншими параметрами. Кожен товар повинен мати свою сторінку з детальною інформацією про нього, таку як опис, зображення, ціна, наявність тощо. Це дозволить користувачам ознайомитися з асортиментом товарів і зробити обґрунтований вибір.

Додавання товарів до кошика передбачає можливість користувачам додавати товари, які їм сподобалися, до свого кошика. На кожній сторінці товару має бути присутня кнопка «Додати до кошика». Після натискання на цю кнопку товар повинен бути доданий до кошика користувача. Користувач повинен мати можливість переглядати вміст свого кошика, включаючи список товарів, їх кількість і загальну суму. Крім того, користувач повинен мати можливість змінювати кількість товарів або видаляти їх з кошика. Це дозволить користувачам керувати своїми покупками перед оформленням замовлення.

Система облікових записів користувачів стосується можливості користувачів реєструватися в системі та створювати облікові записи. Після реєстрації користувач повинен мати можливість авторизуватися за допомогою свого облікового запису. Це дозволить зберігати історію замовлень для кожного користувача та надавати персоналізовані налаштування. Користувачі зможуть зберігати свою адресу доставки, налаштовувати сповіщення про акції та нові товари, відстежувати стан своїх замовлень тощо. Крім того, система повинна забезпечувати безпеку облікових записів і даних користувачів, наприклад, за допомогою шифрування паролів та застосування інших заходів безпеки для запобігання несанкціонованому доступу до особистої інформації користувачів.

Ці функціональні вимоги є основою для проектування програмного забезпечення інтернет-магазину нового покоління. Залежно від конкретних потреб бізнесу та користувачів, можуть бути додаткові функції, такі як оформлення замовлення, оплата, відстеження доставки тощо. Подальший аналіз та проектування вимог деталізуватимуть функціональні можливості програмного забезпечення для забезпечення успішного розроблення та впровадження інтернет-магазину.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Для реалізації поставлених функціональних вимог були обрані такі засоби та технології:

– Docker;

– Render.io;

– React JS;

– Node.js;

– HTML, CSS, JavaScript;

– PostgreSQL.

Docker є популярним інструментом контейнеризації, який дозволяє упаковувати програмне забезпечення та його залежності в контейнери. Використання Docker дозволить забезпечити однорідне середовище виконання програми незалежно від операційної системи та забезпечити легку переносимість і масштабованість системи [1]. За допомогою Docker можна створити контейнери для різних компонентів системи, таких як фронтенд, бекенд, база даних, що дозволить ефективно керувати розгортанням і масштабуванням інфраструктури.

Render.io є хмарною платформою для розгортання та хостингу веб-додатків. Використання Render.io дозволить швидко розгорнути та масштабувати інтернет-магазин, забезпечуючи високу доступність та швидкодію роботи. Render.io забезпечує автоматичне масштабування, моніторинг та керування серверами, що дозволяє зосередитися на розробці додатку, не витрачаючи час на налаштування серверного середовища [2].

React JS є популярною JavaScript бібліотекою для розробки користувацьких інтерфейсів. Використання React JS дозволить створити ефективний та динамічний фронтенд інтернет-магазину, де користувачі зможуть переглядати товари, шукати конкретні товари та переглядати детальну інформацію про них. React JS використовує компонентний підхід до розробки і забезпечує швидке оновлення інтерфейсу за допомогою віртуального DOM, що дозволяє створити плавний та зручний користувацький досвід [3].

Node.js є платформою для виконання JavaScript-коду на сервері. Використання Node.js дозволить реалізувати бекенд інтернет-магазину, який буде обробляти запити користувачів, зберігати та отримувати дані з бази даних та забезпечувати логіку роботи системи. Node.js забезпечує неблокуючий та подієвий ввід/вивід, що дозволяє створити швидку та масштабовану серверну частину додатку.

HTML, CSS та JavaScript є основними технологіями для створення веб-інтерфейсу. Використання цих технологій дозволить створити зручний та привабливий користувацький інтерфейс для інтернет-магазину. HTML

використовується для створення структури сторінок, CSS – для оформлення та стилізації, а JavaScript – для динамічних ефектів та взаємодії з користувачем.

PostgreSQL є потужною та надійною реляційною базою даних. Використання PostgreSQL дозволить зберігати дані про товари, користувачів та замовлення, забезпечуючи ефективний доступ до них та забезпечуючи цілісність та безпеку даних. PostgreSQL надає багато функціональностей для оптимізації та управління базою даних, таких як транзакції, індекси, внешні ключі та інші, що робить його ідеальним вибором для зберігання важливих даних інтернет-магазину.

Вибір цих засобів та технологій забезпечить створення функціонального, ефективного та зручного інтернет-магазину нового покоління з можливістю перегляду товарів, додавання товарів до кошика та оформлення замовлень. Комбінація Docker, Render.io, React JS, Node.js, HTML, CSS, JavaScript та PostgreSQL дозволить реалізувати всі необхідні функції і забезпечить зручний та злагоджений досвід користувача.

В результаті проведеного аналізу потреб і вимог користувачів були визначені основні функціональні вимоги до інтернет-магазину нового покоління. Виявлено, що користувачі очікують зручного інтерфейсу, можливості перегляду товарів, додавання їх до кошика та оформлення замовлень. Також вимоги стосуються безпеки та швидкодії системи.

Висновки до розділу 2

Враховуючи ці вимоги, були обрані відповідні засоби та технології для реалізації поставлених завдань.

Використання Docker дозволить створити уніфіковане середовище виконання програми, що спрощує переносимість і масштабованість системи.

Render.io надасть можливість швидко розгорнути та масштабувати інтернет-магазин з високою доступністю та швидкодією.

React JS дозволить створити динамічний та ефективний фронтенд інтерфейсу для зручного взаємодії з користувачем.

Використання Node.js забезпечить реалізацію бекенду системи, обробку запитів користувачів та взаємодію з базою даних.

HTML, CSS та JavaScript використовуватимуться для створення привабливого та зручного користувацького інтерфейсу.

PostgreSQL буде використовуватись для зберігання та ефективного доступу до даних про товари, користувачів та замовлення.

Загальний вибір цих засобів та технологій дозволить створити функціональний, ефективний та зручний інтернет-магазин, який відповідає потребам та вимогам користувачів.

РОЗДІЛ 3
СТВОРЕННЯ ІНТЕРНЕТ-МАГАЗИНУ З ВИКОРИСТАННЯМ REACT.JS

3.1 Практична реалізація об’єкта проектування

3.1.1 Клієнт-серверна модель

Клієнт-серверна модель (рис. 3.1) є однією з найпоширеніших архітектурних моделей, що використовуються для розробки програмних додатків та систем. Ця модель передбачає розділення функціональності та обов’язків між двома основними компонентами - клієнтом і сервером. Клієнт відповідає за взаємодію з користувачем та відображення інформації, тоді як сервер забезпечує обробку запитів, доступ до ресурсів та надання відповідей клієнту [4].

Основні принципи та характеристики клієнт-серверної моделі:

– розділення обов’язків: у цій моделі функції клієнта та сервера розділені, що дозволяє їх незалежно розвивати, масштабувати та змінювати без впливу на один одного. Клієнт відповідає за інтерфейс користувача, взаємодію з користувачем та відображення даних, тоді як сервер забезпечує обробку бізнес-логіки, зберігання даних та доступ до ресурсів;

– клієнтська частина: клієнт може бути реалізований у вигляді десктопної програми, мобільного додатку або веб-браузера. Він надсилає запити до сервера для отримання даних, виконання операцій або зміни стану системи. Клієнтська частина зазвичай включає інтерфейс користувача, логіку взаємодії та валідацію даних;

– серверна частина: сервер обробляє запити, які надсилаються з клієнта, та надає відповіді з необхідною інформацією. Вона містить бізнес-логіку, оброблює запити, зберігає та отримує дані з бази даних, виконує необхідні обчислення та взаємодіє з іншими системами чи послугами;

– комунікація: клієнт та сервер спілкуються між собою за допомогою мережевих протоколів, таких як HTTP. Клієнт надсилає запити до сервера, який обробляє їх і надсилає відповіді назад. Цей процес відбувається за принципом

«запит-відповідь», де клієнт очікує на відповідь від сервера після надсилання запиту;

– масштабованість: клієнт-серверна модель дозволяє масштабувати систему незалежно від сторін клієнта та сервера. Це означає, що можна збільшувати обсяги обробки даних та кількість користувачів шляхом масштабування серверної інфраструктури, без впливу на клієнтську частину;

– безпека: клієнт-серверна модель дозволяє реалізувати заходи безпеки, такі як аутентифікація, авторизація та шифрування даних. Сервер може забезпечити контроль доступу до ресурсів та захист від несанкціонованого доступу.

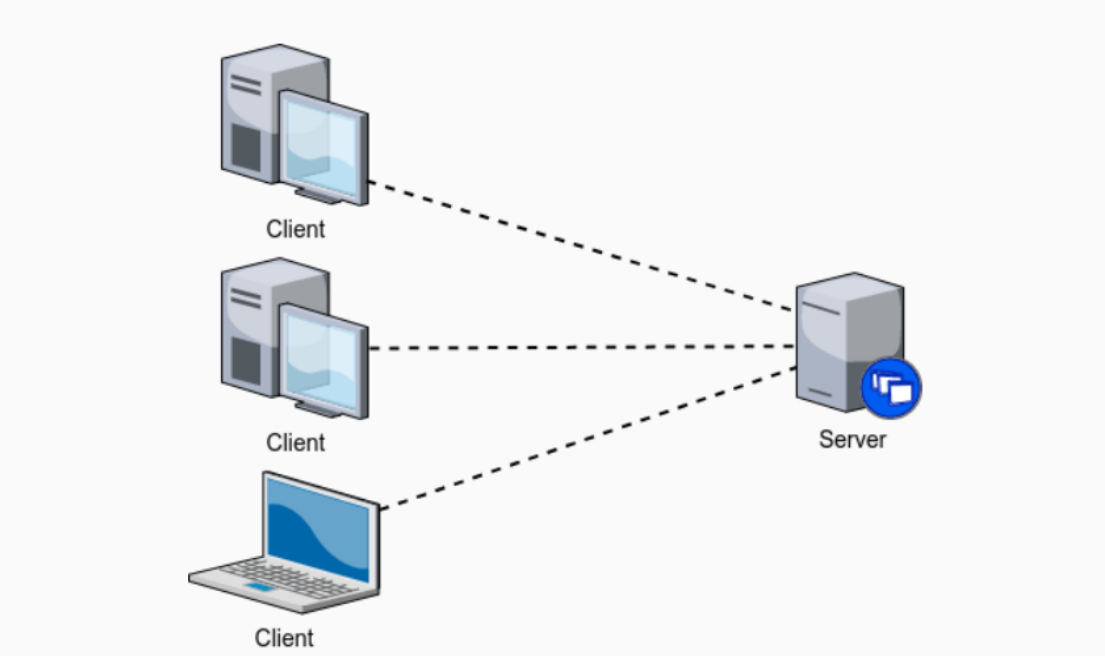

Рисунок 3.1 – Клієнт-серверна модель веб-додатку

Застосування клієнт-серверної моделі у проекті з використанням React JS та Deno JS дозволяє ефективно розподілити функціональність між клієнтом та сервером, забезпечуючи високу продуктивність, масштабованість та безпеку системи. Клієнтська частина, побудована на React JS, забезпечує швидкий та зручний інтерфейс для користувачів, тоді як серверна частина, розроблена з використанням Deno JS, забезпечує обробку запитів, доступ до бази даних та

виконання бізнес-логіки. Ця модель є надійним та гнучким підходом для розробки сучасних інтернет-магазинів, які задовольняють потреби користувачів та підприємств.

3.1.2 Рівнева модель побудови проекту

Архітектура з чотирьох рівнів (рис. 3.2), є типовою для багатьох веб-додатків і може бути ефективною для створення інтернет-магазину. Давайте розглянемо кожен рівень детальніше:

– Views (вигляди або представлення). Рівень views відповідає за візуальне відображення даних і взаємодію з користувачем. У контексті інтернет-магазину, views відповідають за створення користувацького інтерфейсу, який включає в себе веб-сторінки, форми, кнопки та інші елементи, які відображаються на екрані користувача. Views отримують дані з контролерів і передають їх на відображення користувачу;

– Controllers (контролери). Рівень controllers відповідає за обробку запитів користувача і взаємодію з views та services. Контролери отримують запити від користувача через views, обробляють їх, виконують необхідні операції та взаємодіють з рівнем services для отримання необхідних даних або виконання бізнес-логіки. Після обробки контролери повертають відповідь до views для відображення результатів користувачу;

– Services (сервіси). Рівень services відповідає за виконання бізнес-логіки та взаємодію з базою даних. Сервіси забезпечують логіку, необхідну для обробки запитів, виконання операцій з базою даних та роботи з іншими залежностями. У випадку інтернет-магазину, сервіси можуть включати функції для отримання інформації про товари, обробки замовлень, керування користувачами тощо;

– Database (база даних). Рівень бази даних відповідає за зберігання та управління даними, які використовуються в інтернет-магазині. Він включає в себе таблиці, де зберігаються дані про товари, користувачів, замовлення тощо. Сервіси можуть взаємодіяти з базою даних для отримання, оновлення та збереження необхідних даних.

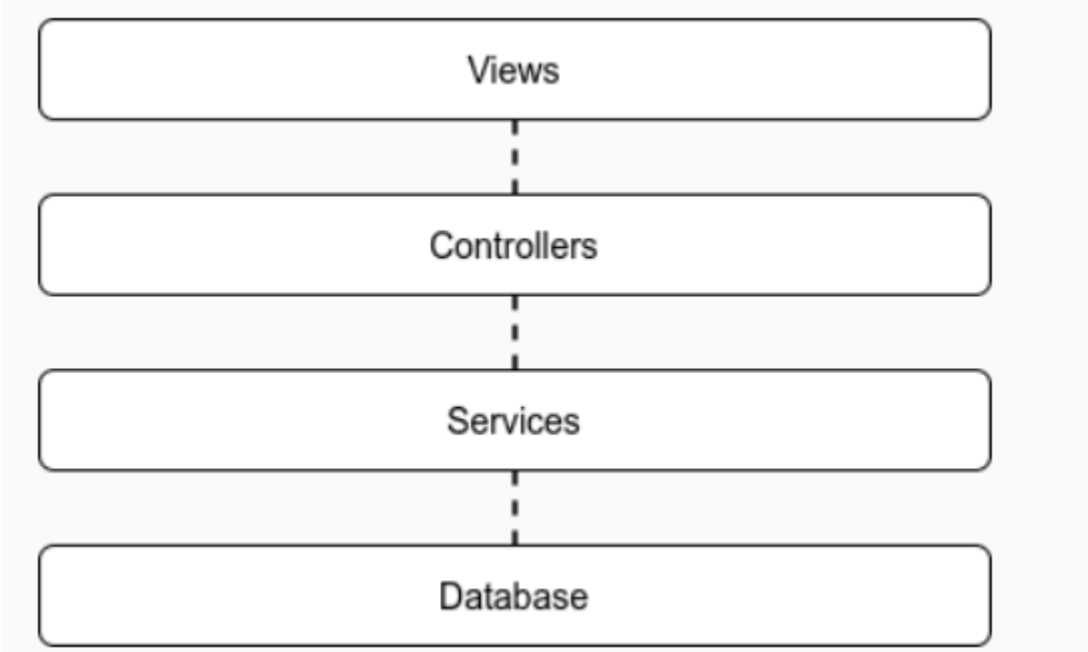

Рисунок 3.2 – Шарова модель архутектури веб-додатку

У цій архітектурі views представляють інтерфейс, через який користувачі взаємодіють з додатком, контролери обробляють запити користувачів, сервіси виконують бізнес-логіку та взаємодіють з базою даних, а база даних забезпечує зберігання та доступ до даних.

Ця архітектура дозволяє забезпечити розділення відповідальностей, модульність, легкість тестування та підтримку коду. Кожен рівень може бути розроблений і підтримуваний незалежно, що спрощує розробку, підтримку та розширення проекту.

Використання клієнт-серверної моделі з чотирьох рівнів дозволяє ефективно реалізувати функціональні вимоги інтернет-магазину та забезпечити гнучкість та розширюваність проекту.

3.1.3 Database

Рівень Database (база даних) є критичним компонентом у структурі інтернет-магазину, оскільки він відповідає за зберігання і керування даними, що використовуються в магазині. Його основні завдання включають

забезпечення взаємодії з базою даних, виконання запитів та отримання результатів.

У коді з файлу database.js (рис. 3.3) ми бачимо наступні кроки.

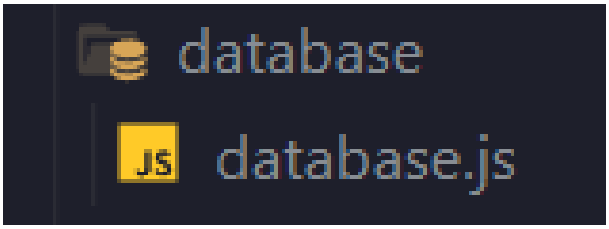

Рисунок 3.3 – Будова рівню Database

Імпорт необхідних залежностей в лістингу 3.1.

Лістинг 3.1 – Код файлу database.js

```
import { Pool } from "https://deno.land/x/postgres@v0.17.0/mod.ts";
```

кінець лістингу 3.1

У цьому кроці ми імпортуємо модулі, необхідні для роботи з базою даних PostgreSQL [5].

Визначення кількості одночасних з’єднань та створення пулу з’єднань з базою даних в лістингу 3.2.

Лістинг 3.2 – Код файлу database.js

```
const CONCURRENT_CONNECTIONS = 3;
const connectionPool = new Pool(DATABASE_URL, CONCURRENT_CONNECTIONS);
```

кінець лістингу 3.2

Ми визначаємо кількість одночасних з’єднань (CONCURRENT_CONNECTIONS) і створюємо пул з’єднань (connectionPool), який дозволить нам ефективно керувати з’єднаннями до бази даних.

Оголошення функції executeQuery, яка виконує запит до бази даних за допомогою пулу з’єднань.

Загальний код файлу database.js в лістингу 3.3.

Лістинг 3.3 – Код файлу database.js

```
  const executeQuery = async (query, params) => {
  const response = {};
  let client;

  try {
    client = await connectionPool.connect();
    const result = await client.queryObject(query, params);
    if (result.rows) {
      response.rows = result.rows;
    }
  } catch (e) {
    response.error = e;
  } finally {
    try {
      await client.release();
    } catch (e) {
      console.log(e);
    }
  }

  return response;
};
```

кінець лістингу 3.3

Ця функція executeQuery отримує запит query та параметри params і виконує його шляхом отримання з’єднання з пулу (client = await connectionPool.connect()) та виконання запиту (client.queryObject(query, params)). Результат запиту зберігається в об’єкті response. Якщо запит повертає рядки, вони зберігаються у response.rows. У разі виникнення помилки, вона обробляється і зберігається у response.error. Після виконання запиту з’єднання з базою даних повертається до пулу для подальшого використання.

Експорт функції executeQuery для використання в інших частинах проекту в лістингу 3.4.

Лістинг 3.4 – Код файлу database.js

```
   export { executeQuery };
```

кінець лістингу 3.4

Цей крок дозволяє експортувати функцію executeQuery, щоб вона була доступна для використання в інших частинах проекту, наприклад, у контролерах чи сервісах.

Така архітектура бази даних дозволяє забезпечити ефективну взаємодію з базою даних і виконання необхідних запитів у різних частинах проекту.

3.1.4 Services

Рівень Services (сервіси) є ключовим рівнем архітектури додатка і відповідає за бізнес-логіку. У цьому рівні розміщені функції, які виконують специфічні операції з даними, оброблюють логіку застосунку та взаємодіють з іншими рівнями, такими як база даних або представлення (рис. 3.4).

Загальний код файлу pageServices.js в лістингу 3.5.

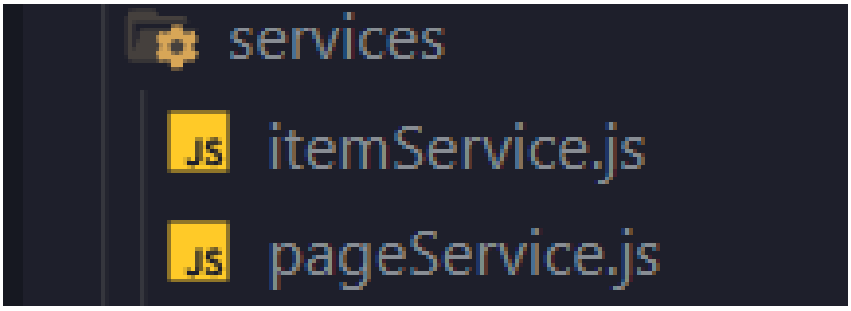

Рисунок 3.4 – Будова рівню Services

Лістинг 3.5 – Код файлу pageServices.js

```
 import { executeQuery } from "../database/database.js";

const create = async (name) => {
  await executeQuery("INSERT INTO store_pages (name) VALUES ($name);",
  {name: name},
  );
};

const deactivatePage = async (id) => {
  await executeQuery("UPDATE store_pages SET active = false WHERE id = $id;",
  {id: id},
  );
};

const getPageName = async (id) => {
  return await executeQuery("SELECT name FROM store_pages WHERE id = $id;",
  {id: id},
  );
};
```

Продовження лістингу 3.5

```
const findAllActivePages = async () => {
  return await executeQuery("SELECT * FROM store_pages WHERE active = true;");
};

const countAllPages = async () => {
  const number = await executeQuery("SELECT COUNT(*) FROM store_pages;");
  const result = parseInt(number.rows[0].count);
  if (result == undefined || result < 1) {
    return "No shopping lists yet.";
  } else {
    return `Shopping lists: ${result}`;
  }
}

export { create, deactivatePage, findAllActivePages, getPageName, countAllPages
};
import { Pool } from "https://deno.land/x/postgres@v0.17.0/mod.ts";
```

кінець лістингу 3.5

У файлі pageService.js містяться сервісні функції, пов’язані з операціями над сторінками в інтернет-магазині. Розглянемо їх детальніше:

– create. Ця функція створює нову сторінку в інтернет-магазині. Вона використовує функцію executeQuery з рівня бази даних для виконання запиту INSERT, який додає новий запис з назвою сторінки;

– deactivatePage. Ця функція вимикає сторінку в інтернет-магазині. Вона використовує функцію executeQuery з рівня бази даних для виконання запиту UPDATE, який оновлює поле active на false для вказаної сторінки;

– getPageName. Ця функція отримує назву сторінки за її ідентифікатором. Вона використовує функцію executeQuery з рівня бази даних для виконання запиту SELECT і повертає результат;

– findAllActivePages. Ця функція отримує всі активні сторінки з бази даних. Вона використовує функцію executeQuery з рівня бази даних для виконання запиту SELECT і повертає результат;

– countAllPages. Ця функція підраховує кількість усіх сторінок в інтернет-магазині. Вона використовує функцію executeQuery з рівня бази даних для виконання запиту SELECT COUNT(*) і повертає результат.

У файлі itemService.js знаходяться сервісні функції, пов’язані з операціями над елементами на сторінках в інтернет-магазині, код файлу pageService.js в лістингу 3.6.

Лістинг 3.6 – Код файлу pageService.js

```
  import { executeQuery } from "../database/database.js";

const create = async (name, storePageId) => {
  await executeQuery(
    "INSERT INTO store_page_items (name, store_page_id) VALUES ($name ,
$storePageId);",
    {name: name, storePageId: storePageId},
  );
};

const findAllItems = async (storePageId) => {
    const notCollected = await executeQuery("SELECT * FROM store_page_items
WHERE collected = false AND store_page_id = $storePageId ORDER BY name ASC;",
    {storePageId: storePageId},
    );
    const collected = await executeQuery("SELECT * FROM store_page_items WHERE
collected = true AND store_page_id = $storePageId ORDER BY name ASC;",
    {storePageId: storePageId},
    );
    return { pageId: storePageId, nonCollected: notCollected, collected:
collected };

};
const countAllItems = async () => {
    const number = await executeQuery("SELECT COUNT(*) FROM store_page_items;");
    const result = parseInt(number.rows[0].count);
    if (result == undefined || result < 1) {
      return 0;
    } else {
      return `Store pages: ${result}`;
    }
}

const collectItem = async (id) => {
    await executeQuery("UPDATE store_page_items SET collected = true WHERE id =
$id;",
    {id: id},
    );
};

export { create, collectItem, findAllItems, countAllItems };
```

кінець лістингу 3.6

– create. Ця функція створює новий елемент на сторінці в інтернет-магазині. Вона використовує функцію executeQuery з рівня бази даних для виконання запиту INSERT, який додає новий запис з назвою елемента та ідентифікатором сторінки, на якій він розміщений;

– findAllItems. Ця функція отримує всі елементи, які не зібрані та зібрані на певній сторінці в інтернет-магазині. Вона використовує функцію executeQuery з рівня бази даних для виконання запитів SELECT та повертає результат у вигляді об’єкта, що містить не зібрані та зібрані елементи, а також ідентифікатор сторінки;

– countAllItems. Ця функція підраховує кількість усіх елементів в інтернет-магазині. Вона використовує функцію executeQuery з рівня бази даних для виконання запиту SELECT COUNT(*) і повертає результат;

– collectItem. Ця функція позначає елемент як зібраний на основі його ідентифікатора. Вона використовує функцію executeQuery з рівня бази даних для виконання запиту UPDATE, змінюючи значення поля collected на true для вказаного елемента.

Ці сервісні функції надають необхідний інтерфейс для взаємодії з базою даних та забезпечують виконання різних операцій з даними для сторінок та елементів в інтернет-магазині. Вони імпортують функцію executeQuery з рівня Database, щоб виконувати запити до бази даних.

3.1.5 Controllers

Рівень контролерів (Controllers) є ключовим рівнем архітектури додатка і відповідає за обробку вхідних запитів і взаємодію з рівнем сервісів. У контролерах містяться функції, які обробляють HTTP-запити і викликають відповідні функції сервісів для виконання бізнес-логіки (рис. 3.5).

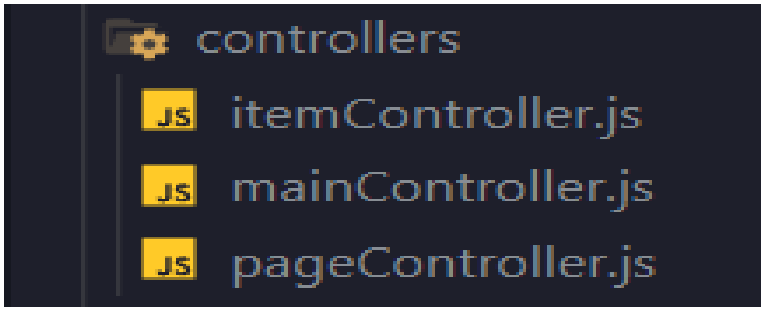

Рисунок 3.5 – Будова рівню Controllers

Файл itemController.js містить контролери, пов’язані з елементами на сторінках інтернет-магазину. Давайте розглянемо кожен з них детальніше.

Код файлу itemController.js в лістингу 3.7.

Лістинг 3.7 – Код файлу itemController.js

```
   import { renderFile } from "https://deno.land/x/eta@v2.0.0/mod.ts";
import * as itemService from "../services/itemService.js";
import * as pageService from "../services/pageService.js";
import * as requestUtils from "../utils/requestUtils.js";
const responseDetails = {

  headers: { "Content-Type": "text/html;charset=UTF-8" },
};

const addItem = async (request) => {

    const url = new URL(request.url);
    const urlParts = url.pathname.split("/");
    const pageId = urlParts[2];

    const formData = await request.formData();
    const name = formData.get("name");

    await itemService.create(name, pageId);

    return requestUtils.redirectTo(`/pages/${ pageId }`);
};
const collectItem = async (request) => {
    const url = new URL(request.url);
    const urlParts = url.pathname.split("/");
    const pageId = urlParts[2];
    const itemId = urlParts[3];

    await itemService.collectItem(itemId);

    return requestUtils.redirectTo(`/pages/${ pageId }`);
};

const viewPageItems = async (request) => {
    const url = new URL(request.url);
    const urlParts = url.pathname.split("/");
    const pageId = urlParts[2];

    const data = {
        page_items: await itemService.findAllItems(pageId),
        page_name: await pageService.getPageName(pageId),
    };
    console.log("items");
    console.log(data.page_items.nonCollected);
    console.log(data.page_items.collected);
    console.log(data.page_name.rows[0]);
    return new Response(await renderFile("page.eta", data), responseDetails);
};
export { addItem, collectItem, viewPageItems };
```

кінець лістингу 3.7

– addItem. Ця функція обробляє POST-запит для додавання нового елемента на сторінку. Вона отримує дані з форми запиту, такі як назва елемента та ідентифікатор сторінки. Далі вона викликає функцію create з itemService.js, передаючи отримані дані для створення нового елемента. Після цього функція повертає редирект до сторінки зі списком елементів сторінки;

– collectItem. Ця функція обробляє POST-запит для позначення елемента як зібраного. Вона отримує ідентифікатори сторінки та елемента з URL-адреси запиту і викликає функцію collectItem з itemService.js, передаючи ідентифікатор елемента для оновлення статусу зібраності. Після цього функція повертає редирект до сторінки зі списком елементів сторінки;

– viewPageItems. Ця функція обробляє GET-запит для відображення списку елементів на сторінці. Вона отримує ідентифікатор сторінки з URL-адреси запиту і викликає функції findAllItems та getPageName з itemService.js та pageService.js відповідно. Ці функції виконують запити до бази даних для отримання всіх елементів сторінки та назви сторінки відповідно. Дані передаються до шаблону page.eta, який відображає список елементів та назву сторінки. Результат відправляється у відповідь у вигляді HTML-сторінки.

Файл pageController.js містить контролери, пов’язані зі сторінками інтернет-магазину.

Давайте розглянемо їх детальніше. Код файлу pageController.js в лістингу 3.8.

Лістинг 3.8 – Код файлу pageController.js

```
   import { renderFile } from "https://deno.land/x/eta@v2.0.0/mod.ts";
import * as pageService from "../services/pageService.js";
import * as requestUtils from "../utils/requestUtils.js";

const responseDetails = {
  headers: { "Content-Type": "text/html;charset=UTF-8" },
};

const addPage = async (request) => {
  const formData = await request.formData();
  const name = formData.get("name");
```

Продовження лістингу 3.8

```
  await pageService.create(name);

  return requestUtils.redirectTo("/pages");

};
const deactivatePage = async (request) => {
  const url = new URL(request.url);
  const urlParts = url.pathname.split("/");
  const pageId = urlParts[2];

  await pageService.deactivatePage(pageId);

  return requestUtils.redirectTo(`/pages`);
};

const viewPages = async (request) => {
  const data = {
    store_pages: await pageService.findAllActivePages(),
  };
  console.log("Pages");
  console.log(data.store_pages.rows);
  return new Response(await renderFile("pages.eta", data), responseDetails);
};
export { addPage, deactivatePage, viewPages };
```

кінець лістингу 3.8

– addPage. Ця функція обробляє POST-запит для додавання нової сторінки в магазин. Вона отримує дані з форми запиту, такі як назва сторінки, і викликає функцію create з pageService.js, передаючи отримані дані для створення нової сторінки. Після цього функція повертає редирект до сторінки зі списком сторінок магазину;

– deactivatePage. Ця функція обробляє POST-запит для деактивації сторінки. Вона отримує ідентифікатор сторінки з URL-адреси запиту і викликає функцію deactivatePage з pageService.js, передаючи ідентифікатор сторінки для зміни її статусу на неактивний. Після цього функція повертає редирект до сторінки зі списком сторінок магазину;

– viewPages. Ця функція обробляє GET-запит для відображення списку активних сторінок магазину. Вона викликає функцію findAllActivePages з pageService.js, щоб отримати всі активні сторінки магазину. Дані передаються до шаблону pages.eta, який відображає список сторінок. Результат відправляється у відповідь у вигляді HTML-сторінки. Файл mainController.js

містить контролер, пов’язаний з головною сторінкою магазину. Код файлу mainController.js в лістингу 3.9;

Лістинг 3.9 – Код файлу mainController.js

```
import { renderFile } from "https://deno.land/x/eta@v2.0.0/mod.ts";
import * as pageService from "../services/pageService.js";
import * as itemService from "../services/itemService.js";
import * as requestUtils from "../utils/requestUtils.js";

const responseDetails = {
    headers: { "Content-Type": "text/html;charset=UTF-8" },
};

const viewMain = async (request) => {
    const data = {
      store_pages: await pageService.countAllPages(),
      page_items: await itemService.countAllItems(),
    };

    return new Response(await renderFile("main.eta", data), responseDetails);
};
export { viewMain };
```

кінець лістингу 3.9

– viewMain. Ця функція обробляє GET-запит для відображення головної сторінки магазину. Вона викликає функції countAllPages та countAllItems з pageService.js та itemService.js відповідно, щоб отримати кількість всіх сторінок та елементів в магазині. Дані передаються до шаблону main.eta, який відображає статистику магазину. Результат відправляється у відповідь у вигляді HTML-сторінки.

Кожен з цих контролерів використовує відповідні функції сервісів (наприклад, itemService.js, pageService.js) для отримання даних та виконання бізнес-логіки. Вони також використовують допоміжні функції, такі як requestUtils.js для обробки запитів та renderFile з пакету Eta для відображення HTML-шаблонів.

3.1.6 Views

Розділ, присвячений шару Views.

У рівні views інтернет-магазину відбувається візуальне відображення даних та взаємодія з користувачем. В цьому шарі створюються веб-сторінки,

форми, кнопки та інші елементи, які сприймаються та використовуються користувачем для взаємодії з системою. Views відповідають за представлення даних та їх структуру відповідно до потреб інтерфейсу користувача.

У веб-магазині, наприклад, views можуть містити сторінки для перегляду списку товарів, детального перегляду окремого товару, оформлення замовлення, реєстрації користувача та інших функцій, необхідних для взаємодії з магазином. Вони відображають інформацію з бази даних, отриману від контролерів, і дозволяють користувачам взаємодіяти з системою шляхом заповнення форм, вибору опцій та натискання кнопок (рис. 3.6).

Код файлу Layout.eta в лістингу 3.10.

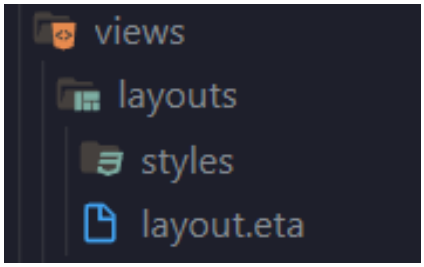

Рисунок 3.6 – Будова шару Views

Лістинг 3.10 – Код файлу Layout.eta

```
<!doctype html>
<html lang="en">
  <head>
    <title>web store</title>
    <link rel="stylesheet" type="text/css" href="layouts/styles/layout-
styles.css">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <meta charset="utf-8">
  </head>
  <body>
        <%~ it.body %>
  </body>
</html>
```

кінець лістингу 3.10

Файл Layout.eta представляє загальну структуру HTML-сторінки, яка використовується в шаблонах views. Цей файл містить базовий шаблон для всіх сторінок інтернет-магазину.

Основні елементи коду з файлу Layout.eta:

<!doctype html>: Ця лінія вказує на тип документа і HTML-версію.

<html lang="en">: Визначає мову сторінки, у цьому випадку англійську.

<title>web store</title>: Встановлює заголовок сторінки, який відображатиметься в заголовку вкладки браузера.

<link rel="stylesheet" type="text/css" href="layouts/styles/layout-styles.css">: Підключає зовнішній файл CSS для оформлення сторінки.

<meta name="viewport" content="width=device-width, initial-scale=1.0">: Визначає налаштування відображення на мобільних пристроях.

<meta http-equiv="X-UA-Compatible" content="ie=edge">: Вказує на режим сумісності з браузерами.

<meta charset="utf-8">: Встановлює кодування символів для сторінки.

<body>: Тіло сторінки, куди будуть вставлятись вміст та компоненти з інших шаблонів.

<%~ it.body %>: Цей тег вставляє вміст сторінки, який буде згенерований з відповідного шаблону views.

Файл Layout.eta використовується як основний шаблон для всіх сторінок інтернет-магазину, і його вміст буде змінюватись в залежності від конкретного вигляду (view), який використовується для відображення сторінки.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Тестування та налагодження є невід’ємною частиною розробки та впровадження інформаційно-комп’ютерних систем. Цей процес дозволяє виявити помилки, перевірити відповідність системи вимогам та забезпечити її стабільну та надійну роботу перед випуском в експлуатацію. У цьому розділі ми розглянемо тестування та налагодження інформаційно-комп’ютерних систем, а також перелік видів тестування, які зазвичай застосовуються для таких випадків.

Тестування інформаційно-комп’ютерних систем включає в себе виконання різних видів тестів для перевірки різних аспектів системи. Основна мета тестування полягає в тому, щоб виявити дефекти, помилки та недоліки у функціональності та продуктивності системи перед її впровадженням. Ось декілька типових видів тестування, які застосовуються під час розробки та налагодження інформаційно-комп’ютерних систем:

– функціональне тестування. Цей вид тестування спрямований на перевірку правильності функціонування системи відповідно до вимог, специфікацій та очікувань. Він перевіряє, чи працюють функції системи належним чином, чи відбувається правильна обробка даних та взаємодія з користувачем;

– тестування продуктивності. Цей вид тестування оцінює продуктивність та швидкодію системи під навантаженням. Він вимірює час відгуку системи, швидкість обробки операцій, масштабованість та стабільність під високим навантаженням;

– тестування безпеки. Цей вид тестування спрямований на виявлення потенційних вразливостей та захисту системи від зловмисників. Він перевіряє систему на наявність уразливостей, перехоплення даних, несанкціонований доступ та інші атаки;

– тестування зручності використання. Цей вид тестування оцінює, наскільки система є зручною та доступною для користувачів. Він оцінює інтерфейс користувача, навігацію, зрозумілість та зручність використання системи;

– тестування сумісності. Цей вид тестування перевіряє, як система взаємодіє з іншими програмними та апаратними засобами. Він перевіряє, чи сумісна система з різними операційними системами, браузерами, базами даних та іншими зовнішніми компонентами;

– автоматизоване тестування. Цей підхід використовує автоматизовані засоби для виконання тестів швидко та ефективно. Він дозволяє автоматизувати

тестування рутинних операцій, прискорювати процес виконання тестів та покращувати точність результатів.

Це лише деякі з загальних видів тестування, які можуть бути застосовані при розробці та налагодженні інформаційно-комп'ютерних систем. Вибір конкретних видів тестування залежить від потреб проекту, його вимог та характеристик системи (рис. 3.7).

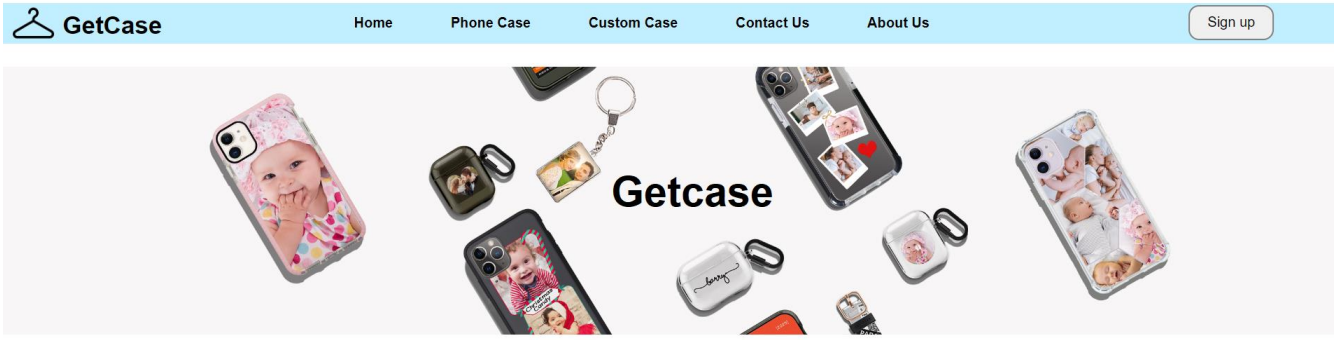

Рисунок 3.7 – Головна сторінка веб-магазину

Це допомагає забезпечити високу якість та надійність інформаційно-комп'ютерних систем перед їх впровадженням у реальне середовище.

Після проведення тестів, веб-додаток, пройшов усе без зауважень. На рисунках (рис. 3.8 – 3.11) зображено вигляд сторінок сайту.

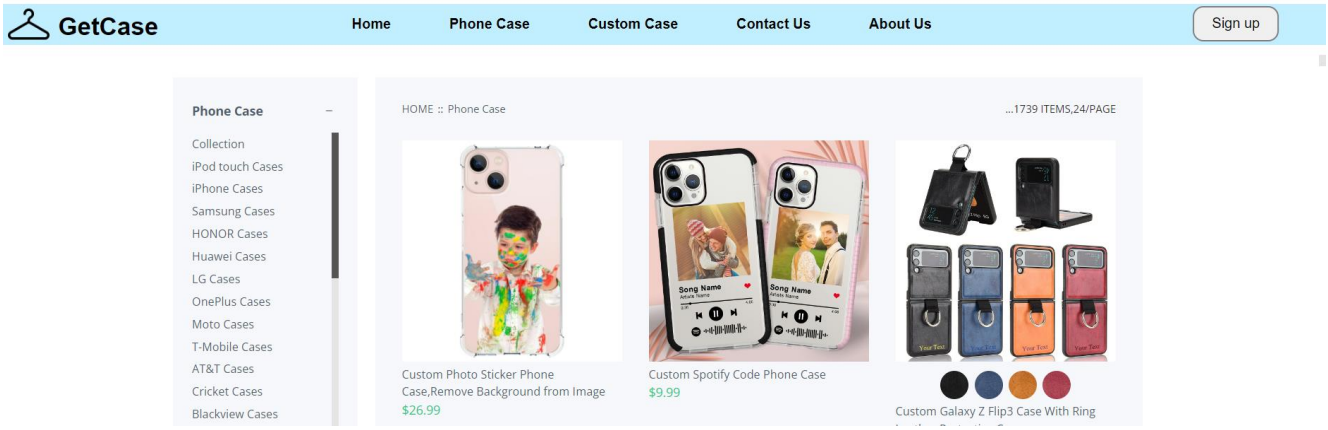

Рисунок 3.8 – Сторінка «Phone Case» веб-додатку

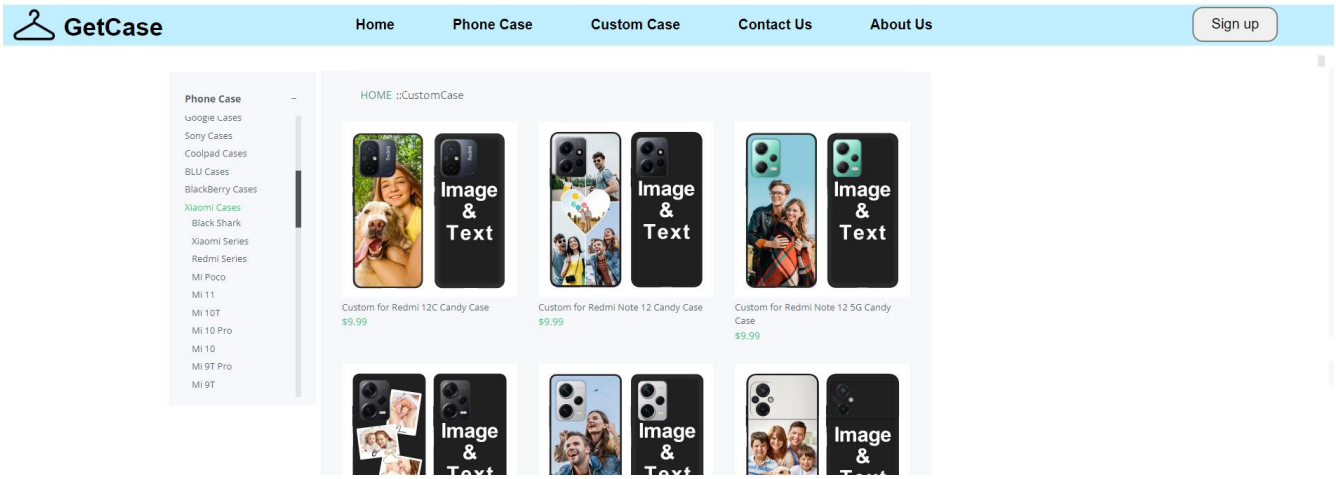

Рисунок 3.9 – Сторінка «Custom Case»

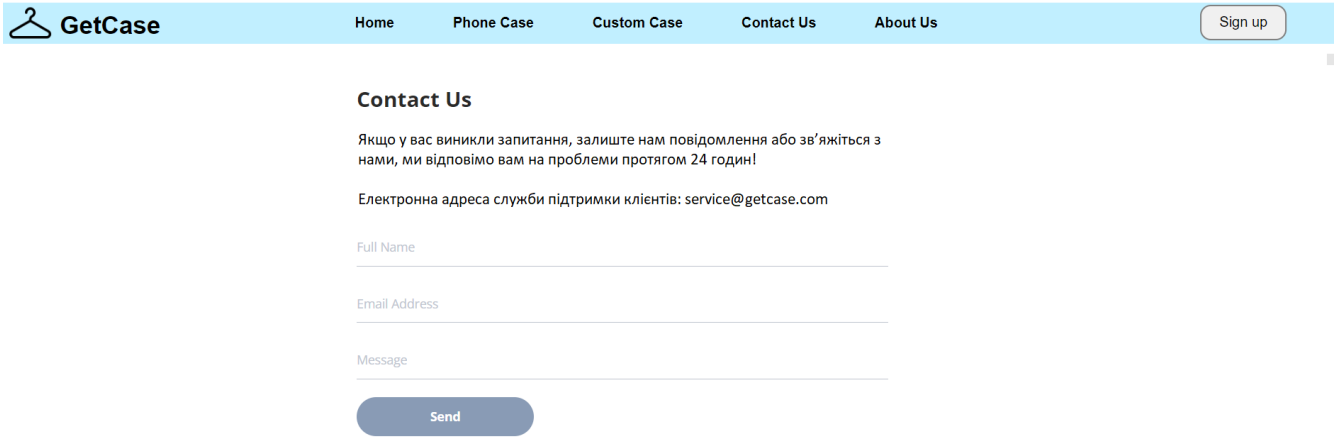

Рисунок 3.10 – Сторінка «Contact Us»

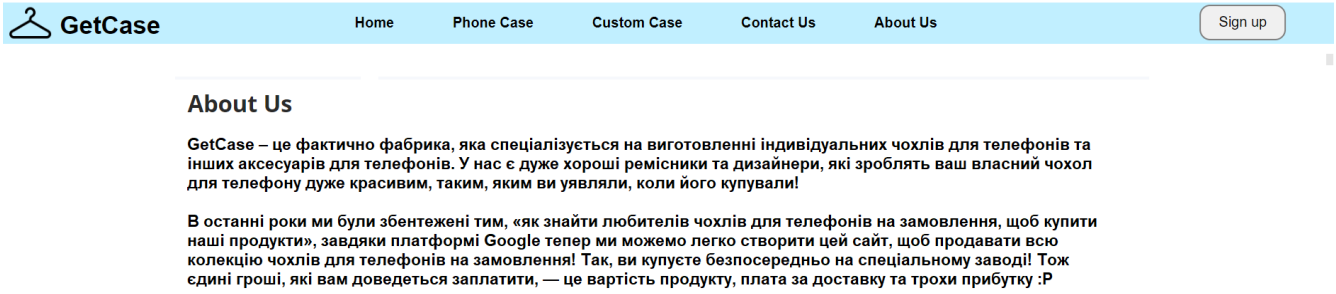

Рисунок 3.11 – Сторінка «About Us»

Висновки до розділу 3

Підсумок виконаної роботи в шарах і їх взаємодії.

Клієнтсько-серверна модель дозволяє ефективно розділити функціональність між клієнтами і сервером. Клієнти відправляють запити серверу, а сервер обробляє їх і повертає відповіді. Ця модель забезпечує швидку обробку запитів і відповідей, полегшує роботу з розподіленими системами і покращує масштабованість системи.

Шарова модель впроваджує структурну організацію системи, розбиваючи її на окремі компоненти: views, database, controllers та services. Кожен з цих шарів має свою відповідальність і виконує конкретні завдання.

Шар views відповідає за відображення даних користувачам. Він забезпечує інтерфейс, через який користувачі можуть спілкуватися з системою. Views використовують дані з бази даних і сервісів для відображення релевантної інформації користувачам.

Шар database відповідає за зберігання та управління даними системи. Він забезпечує доступ до бази даних і забезпечує правильність збереження, оновлення та вилучення даних.

Шар controllers відповідає за обробку запитів користувачів і управління взаємодією між views, database та services. Він приймає запити від користувачів, виконує необхідні операції з даними і направляє їх відповідно до логіки системи.

Шар services відповідає за бізнес-логіку системи. Він містить комплексні операції та функції, які забезпечують основний функціонал системи. Services взаємодіють з базою даних, виконують операції над даними і надають необхідні результати контролерам.

В цілому, робота в шарах і їх взаємодія дозволяють побудувати структуровану, модульну і ефективну інформаційно-комп’ютерну систему. Кожен шар виконує свою функцію і взаємодіє з іншими шарами, створюючи повноцінний функціонал системи. Правильна організація шарів сприяє

полегшенню розробки, підтримки та масштабування системи, а також забезпечує гнучкість і зручність для подальших змін і модифікацій.

ВИСНОВКИ

У цій кваліфікаційній роботі була проведена розробка та реалізація інформаційно-комп’ютерної системи, в основі якої лежить клієнтсько-серверна модель та шарова модель з розподіленими компонентами. Цей підхід дозволяє ефективно організувати та керувати комплексними процесами обробки даних та забезпечує зручний доступ до інформації користувачів.

Розробка системи включала в себе розбиття функціональності на різні шари (views, database, controllers та services), кожен з яких має свої відповідності та функціональні обов’язки. Використання шарової моделі дозволяє забезпечити модульність системи, зручну масштабованість та покращити її підтримку.

Шар views відповідає за візуальне представлення даних та взаємодію з користувачем. Він забезпечує створення зрозумілого та ергономічного інтерфейсу, що дозволяє користувачам легко взаємодіяти з системою.

Шар database відповідає за зберігання та управління даними. Використовуючи базу даних, цей шар забезпечує постійне зберігання та доступ до інформації, а також забезпечує безпеку та цілісність даних.

Шар controllers відповідає за обробку вхідних запитів та керування бізнес-логікою системи. Він забезпечує прийом та перетворення запитів, взаємодію зі шарами views та services, а також виконання потрібних операцій над даними.

Шар services містить функції, які реалізують бізнес-логіку системи. Він забезпечує виконання операцій над даними, взаємодію з базою даних та іншими зовнішніми сервісами, а також реалізацію різноманітних функцій, необхідних для роботи системи.

Взаємодія між цими шарами здійснюється за допомогою викликів функцій та передачі даних. Кожен шар відповідає за свої обов’язки і може використовувати функціональні можливості інших шарів для досягнення своїх цілей. Ця взаємодія дозволяє забезпечити добре організовану та розширювану архітектуру системи.

В процесі виконання бакалаврської роботи на тему «Інтернет-магазин з використанням React.js» було:

– визначено основні функціональні вимоги до інтернет-магазину;

– проведено аналіз вимог до дизайну веб-додатку та розроблено привабливий інтерфейс, а також зручну навігацію;

– вибрано та налаштовано відповідні технології та інструменти, для реалізації інтернет-магазину, React JS для фронтенду та база даних з застосуванням PostgreSQL;

– розроблено та імплементовано інтернет-магазин, на основі визначених вимог до функціональності та дизайну;

– застосовувалися різні види тестування, такі як модульне тестування, інтеграційне тестування та системне тестування, для перевірки правильності роботи окремих компонентів та їх взаємодії. Це дозволило виявити та виправити помилки та проблеми ще на ранніх етапах розробки, покращити якість та надійність системи;

– проведено документування розробки інтернет-магазину та написання звіту включаючи вимоги, дизайн, технології, реалізацію, тестування та валідацію.

Загальний результат роботи полягає у створенні функціональної та добре організованої інформаційно-комп’ютерної системи з використанням клієнтсько-серверної та шарової моделей. Вона демонструє ефективну роботу та забезпечує зручний та надійний доступ до даних та функціональності для користувачів. Результати роботи будуть використані в різних сферах, де потрібна розробка складних інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Try Docker Compose (n.d.). URL: ttps://docs.docker.com/compose/gettingstarted/ (дата звернення: 10.05.2023).

2. Web services | Cloud Hosting for Developers (n.d.). URL: https://render.com/docs/web-services (дата звернення: 10.05.2023).

3. The Modern JavaScript Tutorial. (n.d.). URL: https://javascript.info/ (дата звернення: 10.05.2023).

4. Aalto FITech101 Courses. (n.d.). URL: https://fitech101.aalto.fi/web-software-development/1-introduction-and-tooling/ (дата звернення: 10.05.2023).

5. PostgreSQL Tutorial (n.d.). URL: https://www.postgresqltutorial.com/ (дата звернення: 10.05.2023).

6. GeteFan. (n.d.). GitHub - GeteFan/web-store. GitHub. URL: https://github.com/GeteFan/web-store (дата звернення: 10.05.2023).

7. «Learn web development» посібник з розробки сайтів | MDN. (2023, February 20). URL: https://developer.mozilla.org/en-US/docs/Learn (дата звернення: 10.05.2023).

8. Full stack open. (n.d.). URL: https://fullstackopen.com/en/ (дата звернення: 10.05.2023) (дата звернення: 10.05.2023).

9. Visual Studio Code (n.d.). URL: https://ru.wikipedia.org/wiki/Visual_Studio_Code (дата звернення: 10.05.2023).

10. Get started with GitHub documentation (n.d.). URL: https://docs.github.com/en/get-started (дата звернення: 10.05.2023).

ДОДАТКИ

Додаток А

Структура бази даних

```
CREATE TABLE store_pages (
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL,
  active BOOLEAN DEFAULT TRUE
);

CREATE TABLE store_page_items (
  id SERIAL PRIMARY KEY,
  store_page_id INTEGER REFERENCES store_pages(id),
  name TEXT NOT NULL,
  collected BOOLEAN DEFAULT FALSE
);
```