МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «РОЗРОБКА ТОРГОВОЇ ОНЛАЙН-ПЛАТФОРМИ ЗАСОБАМИ VUEJS ТА NODEJS»

Виконав: студент 4 курсу, групи 6.1210-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
І.А. Карнаух
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.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 – 2024

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)

“ _____ ” ___________ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Карнауху Івану Андрійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка торгової онлайн-платформи засобами VueJS та NodeJS

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 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. Розробка функціональної торгової онлайн-платформи.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 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.	22.01.2024	
2.	Збір вихідних даних.	16.02.2024	
3.	Обробка методичних та теоретичних	11.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	15.04.2024	
5.	Розробка третього та четвертого розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2024	

Студент ______________ І.А. Карнаух
(підпис) (ініціали та прізвище)

Керівник роботи ______________ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка торгової онлайн-платформи засобами VueJS та NodeJS»: 79 с., 19 рис., 1 табл., 11 джерел, 5 додатків.

ВЕБДОДАТОК В РЕАЛЬНОМУ ЧАСІ, ВЕБРОЗРОБКА, ВЗАЄМОДІЯ КЛІЄНТ-СЕРВЕР, ЕЛЕКТРОННА КОМЕРЦІЯ, ОНЛАЙН-ТОРГОВЕЛЬНА ПЛАТФОРМА, MYSQL, NODE.JS, REST API, VUE.JS.

Об’єкт дослідження – онлайн-торговельна платформа.

Предмет дослідження – процеси проєктування і реалізації функцій онлайн-платформи з використанням Vue.js і Node.js.

Мета роботи: розробка функціональної торгової онлайн-платформи на основі технологій Vue.js та Node.js.

Метод дослідження – аналіз сучасних технологій веброзробки, системний аналіз і проєктування, програмування, тестування.

У даній кваліфікаційній роботі розглядається процес розробки онлайн-торговельної платформи з використанням JavaScript фреймворку Vue.js для клієнтської частини та середовища Node.js для серверної частини. Робота включає аналіз вимог до функціональності, проєктування архітектури та інтерфейсу користувача, реалізацію прототипу платформи, а також її тестування.

Основними завданнями, які були вирішені в рамках даної роботи, є вибір та обґрунтування використання технологій Vue.js та Node.js, розробка клієнтської та серверної частин платформи, інтеграція з базами даних та зовнішніми сервісами.

За результатами дослідження було зроблено висновок, що використання комбінації Vue.js та Node.js є ефективним для створення сучасних вебдодатків торговельної спрямованості, забезпечуючи при цьому гнучкість управління проєктом, швидкість розробки та високу адаптивність до вимог користувачів.

SUMMARY

Bachelor’s qualifying paper «Development of a Market Online Platform Using VueJS and NodeJS»: 79 pages, 19 figures, 1 table, 11 references, 5 supplements.

CLIENT-SERVER INTERACTION, E-COMMERCE, MYSQL, NODEJS, ONLINE TRADING PLATFORM, REAL-TIME WEB APPLICATION, REST API, VUE.JS, WEB DEVELOPMENT.

The object of the study is online trading platform.

The aim of the study is development of a functional online trading platform based on Vue.js and Node.js technologies.

The methods of research are analysis of modern web development technologies, system analysis and design, programming, testing.

This qualification work examines the process of developing an online trading platform using the Vue.js JavaScript framework for the client side and the Node.js environment for the server side. The work includes analyzing functionality requirements, designing the architecture and user interface, implementing the platform prototype, and testing it.

The main tasks that were solved within the framework of this work are the selection and justification of the use of Vue.js and Node.js technologies, development of the client and server parts of the platform, integration with databases and external services.

The study concluded that the use of a combination of Vue.js and Node.js is effective for creating modern web applications for trading, while providing flexibility in project management, development speed and high adaptability to user requirements.

ЗМІСТ

Завдання на кваліфікаційну роботу…………………………………………..2

Реферат 4

Summary 5

Скорочення та умовні познаки 9

Вступ... 10

1 Огляд літератури та наявних рішень.. 11

1.1 Технології розробки вебплатформ 11

1.1.1 Клієнтська частина 11

1.1.2 Серверна частина 11

1.1.3 Система управління базою даних 12

1.1.4 Безпека 13

1.1.5 Керування версіями та спільна розробка 14

1.2 Огляд сучасних тенденцій у розробці торгових онлайн платформ . 14

1.2.1 Контекстуальна персоналізація... 14

1.2.2 Голосовий і візуальний пошук 15

1.2.3 Інтерактивні технології для віртуальних примірок...................... 15

1.2.4 Покращені засоби аналітики та бізнес-інтелекту......................... 15

1.3 Порівняльний аналіз існуючих платформ 16

1.4 Переваги використання VueJS і NodeJS у розробці 17

1.4.1 VueJS: легкість використання та продуктивність 17

1.4.2 NodeJS: єдина мова на клієнті та сервері 18

1.4.3 Ефективна робота з даними в реальному часі 18

1.4.4 Масштабованість і відкрита спільнота 18

1.4.5 Сучасні інструменти та технології... 19

2 Архітектура та проєктування 20

2.1 Визначення вимог до платформи 20

2.1.1 Функціональні вимоги.. 20

2.1.2 Нефункціональні вимоги 21
2.1.3 Юзабіліті (зручність використання) 21
2.2 Проєктування користувацького інтерфейсу VueJS 22
2.2.1 Компонентна архітектура 22
2.2.2 Управління станом з Pinia 23
2.2.3 Роутинг з Vue Router 23
2.2.4 Масштабування архітектури 23
2.2.5 Адаптивність інтерфейсу 23
2.2.6 Оптимізація продуктивності 24
2.3 Проєктування серверної частини з використанням NodeJS 24
2.3.1 Використання Express 24
2.3.2 Безпека 25
2.3.3 Робота з базою даних MySQL 27
2.3.4 Кешування 28
2.3.5 Логування та моніторинг 28
2.4 Вибір бази даних та її структура 29
2.5 Проєктування системи безпеки 30
3 Розробка клієнтської та серверної частини 32
3.1 Розробка клієнтської частини з використанням VueJS 32
3.1.1 Створення основних компонентів інтерфейсу 32
3.1.2 Робота зі станом додатка 34
3.1.3 Взаємодія із сервером через API 36
3.1.4 Реалізація аутентифікації та авторизації 37
3.1.5 Обробка помилок 38
3.2 Розробка серверної частини з використанням NodeJS 38
3.2.1 Створення сервера з використанням Express 39
3.2.2 Розробка API для взаємодії з клієнтською частиною 40
3.2.3 Обробка запитів і керування даними 43
3.3 Інтеграція з базою даних 46
3.3.1 Взаємодія з базою даних MySQL 46

3.3.2 Оптимізація запитів і робота з індексами 47
3.3.3 Забезпечення цілісності даних 48
4 Тестування та налагодження... 50
4.1 План тестування 50
4.2 Автоматизоване тестування 50
4.3 Налагодження та обробка помилок.. 51
Висновки 52
Перелік посилань.. 53
Додаток А Приклад типової структури бази даних.................................. 54
Додаток Б Приклад реалізації API для ресурсу “продукти”.................... 55
Додаток В Код основного файлу клієнтської частини............................. 60
Додаток Г Модуль ініціалізації серверної частини 63
Додаток Д Код контролерів серверної частини 66

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СУБД	Система управління базами даних
ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
AR	Augmented Reality
BASE	Basically Available, Soft State
CRUD	Create Read Update Delete
CSRF	Cross-Site Request Forgery
GDPR	General Data Protection Regulation
HIPAA	Health Insurance Portability and Accountability Act
HTTPS	HyperText Transfer Protocol Secure
IDS	Intrusion Detection System
IPS	Intrusion Prevention System
JSON	JavaScript Object Notation
JWT	JSON Web Tokens
ORM	Object Relation Maping
RBAC	Role Based Access Control
SQL	Structured Query Language
SSL	Secure Socket Layer
TLS	Transport Layer Security
WAF	Web Application Firewall
XSS	Cross Site Scripting

ВСТУП

У сучасному цифровому світі, що стрімко розвивається, електронна комерція займає ключову роль у світовій економіці. Щороку все більше і більше компаній переходять на продаж через онлайн-платформи. Це зумовлено численними перевагами, такими, як розширення ринку збуту, зниження витрат на утримання фізичних магазинів та зменшення операційних витрат. Враховуючи тенденцію, створення потужних, надійних і легко використовуваних платформ для онлайн-торгівлі зараз є актуальною задачею для розробників програмного забезпечення. Vue.js і Node.js є одними з найпопулярніших інструментів у цій галузі, оскільки дозволяють створювати масштабовані, гнучкі та високопродуктивні вебдодатки.

Vue.js та Node.js є одними з найпопулярніших інструментів для цієї галузі, оскільки вони дозволяють створювати масштабовані, гнучкі та високопродуктивні вебдодатки. Vue.js вирізняється своєю простотою та гнучкістю, що дозволяє швидко розробляти користувацькі інтерфейси. Node.js, у свою чергу, ефективно керує серверними завданнями, забезпечуючи високу продуктивність, що є надзвичайно важливим для таких платформ. Використання цих технологій відкриває можливості для створення масштабованої, безпечної та ефективної системи, яка відповідає всім вимогам сучасного ринку електронної комерції.

У даній кваліфікаційній роботі буде розглянуто архітектурні підходи, принципи проєктування клієнтської та серверної частини, а також виклики та переваги, які ці технології приносять у процес розробки.

1 ОГЛЯД ЛІТЕРАТУРИ ТА НАЯВНИХ РІШЕНЬ

1.1 Технології розробки вебплатформ

У цьому розділі проводиться аналіз обраних технологій для розробки онлайн-платформи. Кожен аспект технологічного стека детально розглядається з обґрунтуванням вибору.

1.1.1 Клієнтська частина

Для реалізації динамічного і чуйного користувацького інтерфейсу обрано фреймворк VueJS [1]. Він має легковажну структуру, інтуїтивний синтаксис і дає змогу легко інтегрувати компоненти.

Він також підтримує бібліотеку Pinia, яка забезпечує ефективне управління станом програми, що вкрай важливо для створення складних інтерфейсів онлайн-платформи.

1.1.2 Серверна частина

Мовою програмування для серверної частини було обрано JavaScript та середовище виконання NodeJS. Це забезпечує єдину мову програмування як на клієнті, так і на сервері, що зменшує складність взаємодії між клієнтською та серверною частинами.

Для побудови сервера та обробки маршрутів буде використано фреймворк Express, який відомий своєю гнучкістю та продуктивністю.

1.1.3 Система управління базою даних

У сучасній розробці вебдодатків використання баз даних є ключовим аспектом для забезпечення зберігання та управління даними. Існує декілька основних типів баз даних, кожна з яких має свої переваги та недоліки.

Порівняння реляційних та NoSQL баз даних наведено у таблиці 1.1. Таблиця демонструє основні відмінності між цими двома типами баз даних, включаючи тип даних, схему, масштабованість, підтримку транзакцій, продуктивність, гнучкість, підтримку стандартів, приклади та підходящі сценарії використання. Це дозволяє чітко бачити, в яких випадках краще використовувати реляційні бази даних, а в яких – NoSQL рішення.

Таблиця 1.1 – Порівняння SQL та NOSQL баз даних

Тип даних	Структуровані дані (таблиці)	Неструктуровані дані (документи, графи, ключ-значення)
Схема	Жорстка схема (схема повинна бути визначена заздалегідь)	Динамічна схема (схема може змінюватися)
Масштабованість	В основному вертикальна	Горизонтальна
Транзакції	Підтримка ACID транзакцій	Обмежена підтримка транзакцій (BASE модель)
Продуктивність	Висока для складних запитів	Висока для великих обсягів даних і простих запитів
Гнучкість	Менш гнучка	Більш гнучка
Підтримка стандартів	Стандартизована мова SQL	Відсутність єдиного стандарту
Приклади	MySQL, PostgreSQL, Microsoft SQL Server, Oracle	MongoDB, Cassandra, CouchDB, Redis

Продовження таблиці 1.1

Підходящі сценарії використання	Традиційні додатки, які потребують складних запитів та транзакцій (фінансові системи, ERP-системи)	Додатки, які потребують високої масштабованості та гнучкості (реальні часи аналітики, соціальні мережі, IoT)
Тип даних	Структуровані дані (таблиці)	Неструктуровані дані (документи, графи, ключ-значення)

Для ефективного зберігання та маніпулювання даними обрано реляційну базу даних MySQL. Вона надає можливість стабільного і масштабованого зберігання даних, що є важливим фактором для надійної роботи онлайн-платформи.

1.1.4 Безпека

Щоб забезпечити доступ до функціоналу тільки уповноваженим користувачам, буде реалізовано механізми автентифікації та авторизації за допомогою JSON Web Token (JWT) (див. рис. 1.1).

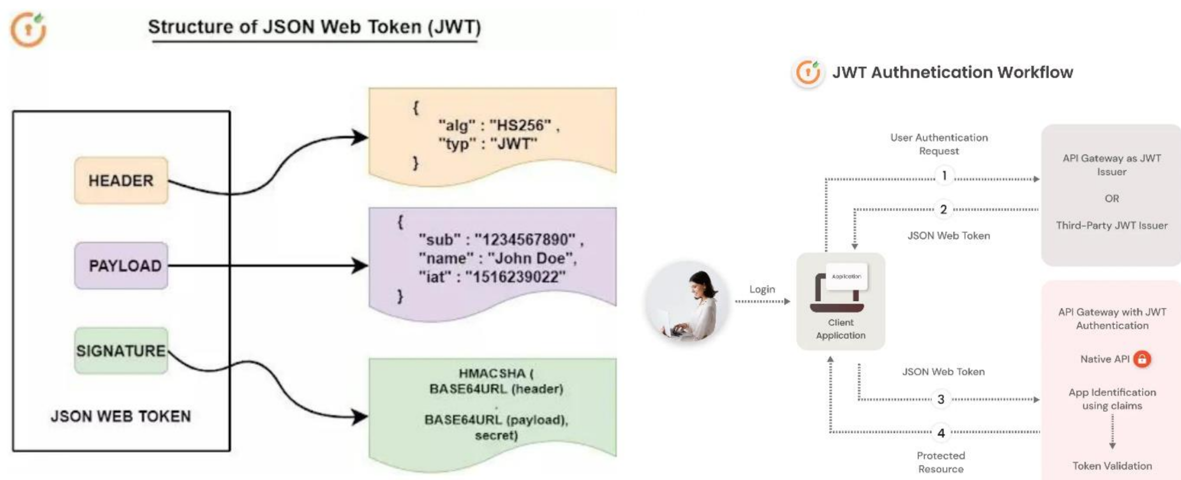

Рисунок 1.1 – Структура та схема JWT [2]

Безпека застосунку буде забезпечена за допомогою використання протоколу HTTPS для захисту передачі даних.

1.1.5 Керування версіями та спільна розробка

Розробка здійснюватиметься з використанням системи контролю версій Git. Це забезпечить ефективне управління змінами, даючи змогу розробникам працювати паралельно через розгалуження і використовувати запити на злиття для інтеграції змін.

Огляд обраних технологій формує технологічну основу для подальшого проєктування та розробки торгової онлайн-платформи, забезпечуючи необхідну продуктивність, безпеку та масштабованість.

1.2 Огляд сучасних тенденцій у розробці торгових онлайн платформ

Сучасний розвиток технологій надає унікальні можливості для розробки, що відображається в ряді актуальних тенденцій. У даному розділі проводиться огляд ключових напрямків, які впливають на сферу електронної комерції та повинні бути враховані під час проєктування і розробки торгової онлайн-платформи.

1.2.1 Контекстуальна персоналізація

Персоналізація – це система, за допомогою якої ви можете динамічно змінювати контент під користувача, ґрунтуючись на його індивідуальних вподобаннях і досвіді минулих покупок [3].

Сучасні торгові платформи дедалі активніше впроваджують

контекстуальну персоналізацію, засновану на даних про поведінку користувачів. Ця тенденція дає змогу надавати більш точні та релевантні рекомендації, що покращує користувацький досвід і підвищує ймовірність здійснення покупок.

1.2.2 Голосовий і візуальний пошук

Розвиток технології розпізнавання голосу та візуального пошуку стає ключовим елементом в електронній комерції. Інтеграція цих можливостей дає змогу користувачам шукати й обирати товари з використанням голосових команд або зображень, що робить процес пошуку зручнішим і ефективнішим.

1.2.3 Інтерактивні технології для віртуальних примірок

З метою підвищення рівня взаємодії з користувачами, торгові платформи впроваджують інтерактивні технології для віртуальних примірок товарів. Це може включати в себе віртуальне примірювання одягу або перегляд товарів у просторі за допомогою доповненої реальності (AR) [4], що сприяє більш впевненим онлайн-покупкам.

1.2.4 Покращені засоби аналітики та бізнес-інтелекту

Сучасні торгові платформи активно використовують передові засоби аналітики та бізнес-інтелекту для збору, аналізу та інтерпретації даних. Це допомагає краще розуміти споживчу поведінку, прогнозувати тенденції та приймати обґрунтовані рішення на основі даних.

Огляд сучасних тенденцій висвітлює ключові напрямки розвитку

торговельних платформ, які слід враховувати при проєктуванні та розробці. Враховуючи ці тенденції, можна створити сучасну, гнучку та конкурентоспроможну електронну торгову платформу.

1.3 Порівняльний аналіз існуючих платформ

Порівняльний аналіз охоплює кілька популярних світових та українських торговельних платформ, надаючи огляд основних особливостей і стратегій, що можуть бути цінними для розробки торговельної онлайн-платформи.

Amazon (https://www.amazon.com/), як одна з провідних торгових платформ у світі, вирізняється своїм широким асортиментом, високим рівнем обслуговування та ефективними механізмами персоналізації. Застосування технологій штучного інтелекту підтримує видачу релевантних рекомендацій, що значно підвищує задоволеність клієнтів.

eBay (https://www.ebay.com/), з унікальною моделлю аукціонів і продажу за фіксованими цінами, акцентує увагу на взаємодії між користувачами та зворотному зв’язку. Це створює атмосферу довіри, стимулюючи активність продавців і покупців.

Alibaba (https://alibaba.com/), найбільша торгова платформа в Китаї, виділяється своєю спеціалізацією на оптовій і роздрібній торгівлі. Великі можливості для взаємодії між постачальниками і покупцями, а також системи забезпечення якості товарів, роблять її ключовим гравцем на світовому ринку.

Etsy (https://www.etsy.com/), орієнтований на ручну роботу та унікальні товари, підтримує індивідуальних продавців і акцентує увагу на унікальності товарів. Це створює особливу спільноту для творчих підприємців.

Rozetka (https://rozetka.com.ua/), будучи одним із найбільших маркетплейсів в Україні, відомий своєю різноманітністю товарів, високим рівнем сервісу та лояльністю клієнтів. Успішне використання технологій та

інноваційні підходи роблять його ключовим гравцем на українському ринку електронної комерції.

Hotline (https://hotline.ua/), популярний маркетплейс в Україні, виділяється своїм зручним інтерфейсом і фокусом на надання порівняльної інформації про товари. Це робить його привабливим для користувачів, які прагнуть зробити усвідомлений вибір.

Порівняльний аналіз дає змогу виділити сильні сторони різних торговельних платформ, включно зі світовими лідерами та українськими гравцями. Аналіз цих платформ надає велику базу для розуміння успішних стратегій, які можна адаптувати при розробці нової торгової онлайн-платформи.

1.4 Переваги використання VueJS і NodeJS у розробці

Розробка клієнтської частини онлайн-платформи базується на використанні VueJS для створення модульних компонентів, що забезпечують гнучкість, масштабованість і легкість підтримки інтерфейсу. Основні компоненти охоплюють глобальні елементи, сторінки з товарами, оформлення замовлення та адміністративні функції.

1.4.1 VueJS: легкість використання та продуктивність

VueJS, як сучасний фреймворк JavaScript, забезпечує легкість у використанні та інтуїтивно зрозумілий синтаксис, що значно спрощує процес розробки. Цей фреймворк дозволяє реалізувати динамічні користувацькі інтерфейси, використовуючи мінімальний обсяг коду. Окрім того, VueJS демонструє високу продуктивність завдяки ефективній системі реактивності, що є критично важливим для створення реактивного інтерфейсу торговельної

платформи. Ці особливості роблять VueJS відмінним вибором для розробки інтерфейсів, що вимагають швидкої роботи та гнучкості у взаємодії з користувачем, що є ключовими аспектами для успішної торговельної платформи.

1.4.2 NodeJS: єдина мова на клієнті та сервері

Вибір NodeJS для серверної частини додатка забезпечує єдину мову програмування на клієнті та сервері, а саме JavaScript. Це знижує складність взаємодії між фронтендом і бекендом, забезпечуючи більш гладку інтеграцію. Такий підхід також прискорює процес розробки і полегшує супровід коду.

1.4.3 Ефективна робота з даними в реальному часі

VueJS та NodeJS забезпечують ефективну обробку даних в реальному часі. Зокрема, підтримка WebSocket у NodeJS дозволяє здійснювати миттєву передачу даних між клієнтом і сервером. Це надзвичайно корисно для моніторингу змін у кошику покупок, статусах товарів та інших критичних параметрах на торговій платформі. Така можливість сприяє підвищенню оперативності реагування на зміни, що відбуваються у системі, та забезпеченню актуальності даних для користувачів.

1.4.4 Масштабованість і відкрита спільнота

Використання VueJS і NodeJS забезпечує високу масштабованість програми. VueJS легко масштабується завдяки компонентній архітектурі, а NodeJS, з підтримкою асинхронних операцій, сприяє ефективному

опрацюванню безлічі запитів. Крім того, обидва фреймворки мають активні та відкриті спільноти розробників, що забезпечує підтримку, оновлення та доступ до багатої екосистеми плагінів і бібліотек.

1.4.5 Сучасні інструменти та технології

Використання VueJS і NodeJS надає доступ до сучасних інструментів і технологій розробки. VueJS легко інтегрується з інструментами збирання, такими як Webpack, що забезпечує ефективне управління залежностями та ресурсами [7]. NodeJS має великий набір модулів у npm, що спрощує впровадження різних функціональних можливостей у бекенд-частину застосунку.

Використання VueJS і NodeJS у розробці торговельної онлайн-платформи забезпечує комбінацію легкості використання, продуктивності, масштабованості та доступу до сучасних технологій, створюючи основу для успішного та інноваційного електронного торговельного майданчика.

2 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ

2.1 Визначення вимог до платформи

Визначення вимог до торговельної онлайн-платформи є ключовим етапом проєктування, який впливає на всі подальші фази розробки. Ефективне формулювання технічних та функціональних вимог забезпечує чітке розуміння задач, які має вирішувати платформа, і сприяє розробці системи, яка відповідатиме потребам користувачів і бізнесу.

2.1.1 Функціональні вимоги

Розглянемо основні функціональні вимоги.

Реєстрація та авторизація користувачів: система повинна надавати можливість реєстрації нових користувачів з подальшою можливістю входу в систему, відновленням забутих паролів, та зміною особистих даних.

Каталог продуктів: платформа повинна надавати можливість створення, редагування, перегляду та видалення продуктів у каталозі. Кожен продукт має мати опис, зображення, ціну та інші відповідні характеристики.

Пошук та фільтрація продуктів: платформа має включати зручні засоби для пошуку продуктів за назвою, категорією, ціною, рейтингом та іншими параметрами.

Управління користувачами: система повинна підтримувати різні рівні доступу, такі як адміністратор, продавець та покупець, із відповідними правами та функціями.

Кошик для покупок: покупці повинні мати можливість додавати товари до кошика, редагувати кількість товарів у кошику, видаляти товари з кошика та переглядати загальну вартість замовлення.

Система знижок та промоцій: можливість автоматичного застосування

знижок або промокодів до покупок, а також проведення акційних кампаній для залучення клієнтів.

Процес покупки: інтеграція засобів оплати для здійснення покупок, включаючи обробку платежів, вибір способу доставки та оформлення замовлення.

2.1.2 Нефункціональні вимоги

Розглянемо нефункціональні вимоги.

Моніторинг та логування: необхідно впровадити систему моніторингу для слідкування за станом і продуктивністю платформи, а також зберігання та ротацію логів дій користувачів та системних подій для полегшення виявлення та виправлення помилок.

Продуктивність: платформа має обробляти запити користувачів швидко та ефективно, незалежно від навантаження.

Масштабованість: система повинна бути здатна масштабуватися для підтримки зростання кількості користувачів та обсягів даних без втрати продуктивності.

Безпека: забезпечення захисту користувацьких даних та транзакцій. Це включає впровадження шифрування даних, безпечного зберігання паролів та виконання вимог PCI DSS для обробки платежів, якщо необхідно інтегрувати сплату карткою напряму на сайті онлайн платформи.

Сумісність: платформа повинна коректно працювати в різних браузерах та на різних пристроях, включаючи мобільні телефони та планшети.

2.1.3 Юзабіліті (зручність використання)

Адаптивність дизайну: інтерфейс має автоматично адаптуватися до різних розмірів екранів і пристроїв, забезпечуючи комфортне використання як

на настільних комп'ютерах, так і на мобільних пристроях.

Інтуїтивно зрозумілий інтерфейс: користувацький інтерфейс повинен бути простим та зрозумілим для всіх категорій користувачів, з мінімальним навчальним порогом.

Доступність: платформа повинна бути доступною для людей з обмеженими можливостями, дотримуючись відповідних стандартів доступності вебконтенту – Web Content Accessibility Guidelines (WCAG) [5].

Під час визначення вимог до платформи важливо спілкуватися з потенційними користувачами та зацікавленими сторонами для збору їх відгуків та вимог, що дозволяє врахувати всі потреби та вдосконалити систему до її реалізації.

2.2 Проєктування користувацького інтерфейсу VueJS

Проєктування користувацького інтерфейсу онлайн-платформи здійснюється за допомогою VueJS, що дозволяє створювати динамічні та інтерактивні вебдодатки. Використання VueJS забезпечує модульний підхід до розробки, де кожен компонент інтерфейсу є незалежним і легко підтримуваним. Це дозволяє швидко створювати, тестувати та інтегрувати нові функції, забезпечуючи при цьому високу продуктивність і якість користувацького досвіду.

2.2.1 Компонентна архітектура

Використання VueJS зосереджується на компонентному підході, що дозволяє структуризувати інтерфейс на перевикористовувані блоки. Компоненти можуть бути організовані у ієрархічній структурі, від простих елементів до складних модулів і шаблонів.

2.2.2 Управління станом з Pinia

Pinia є сучасним і рекомендованим [6] рішенням для управління станом у Vue 3. Вона надає більш простий та гнучкий API порівняно з Vuex і забезпечує легкість інтеграції та використання у великих проєктах. Pinia допомагає організувати глобальний стан, використовуючи концепцію сховищ (store), що спрощує доступ до стану і його мутації через весь додаток.

2.2.3 Роутинг з Vue Router

Vue Router забезпечує управління навігацією всередині продукту, дозволяючи використовувати динамічний роутинг та ленивe завантаження, що підвищує ефективність завантаження ресурсів [11].

2.2.4 Масштабування архітектури

Важливо планувати архітектуру так, щоб вона могла легко масштабуватися. Використання компонентного підходу і Pinia сприяє легкій інтеграції нових функцій та модулів без необхідності переробки існуючих елементів.

2.2.5 Адаптивність інтерфейсу

Адаптивний дизайн є невід’ємною частиною сучасних вебдодатків. Використання CSS-фреймворків, таких як SCSS, SASS, разом з VueJS може значно полегшити реалізацію адаптивності на всіх типах пристроїв.

2.2.6 Оптимізація продуктивності

Pinia та VueJS підтримують численні техніки для оптимізації продуктивності, включаючи асинхронне завантаження компонентів, динамічний імпорт, і ефективне кешування.

Ці аспекти формують основу для створення потужного, масштабованого та ефективного користувацького інтерфейсу, оптимізованого для широкого діапазону користувацьких сценаріїв на торговій онлайн-платформі.

2.3 Проєктування серверної частини з використанням NodeJS

Розробка серверної частини онлайн-платформи здійснюється за допомогою Node.js, що забезпечує масштабованість і високу продуктивність. Використання Node.js дозволяє створювати ефективні серверні додатки, які можуть обробляти велику кількість запитів у режимі реального часу. Основні функції серверної частини включають обробку запитів клієнтів, управління даними в базі даних, аутентифікацію користувачів та інтеграцію з іншими сервісами.

2.3.1 Використання Express

Express – це гнучкий та ефективний вебфреймворк для Node.js, який надає розробникам потужні інструменти для створення вебдодатків та API. У контексті розробки торгової онлайн-платформи, Express використовується для підготовки серверної частини додатку, спрощення маршрутизації, обробки запитів і відповідей, та забезпечення взаємодії з базою даних.

Основні аспекти використання Express розглянемо нижче.

Створення та конфігурація сервера. Express дозволяє легко створювати вебсервер, що слухає вхідні запити на заданих портах. Розробники можуть

конфігурувати середовище сервера, використовуючи мідлвари для додавання додаткової функціональності, такої як обробка JSON-тіл запитів, управління сесіями користувачів, та логування.

Маршрутизація. Express надає механізми маршрутизації, які дозволяють визначати маршрути та їх обробники залежно від HTTP-методу і шляху. Це робить структуру вебдодатку зрозумілою та організованою. Маршрути можуть використовуватися для реалізації API, яке обслуговує запити до бази даних.

Обробка помилок. Express дозволяє реалізувати централізовану обробку помилок через спеціальні мідлвари. Це допомагає управляти помилками, які виникають під час обробки запитів, і відправляти користувачам зрозумілі повідомлення про помилки.

Безпека та автентифікація. Express може бути використаний з різними бібліотеками для забезпечення безпеки, такими як Helmet, які допомагають захистити додаток від відомих вебвразливостей. Також, за допомогою Passport.js або інших мідлварів можна впровадити систему аутентифікації та авторизації.

Оптимізація продуктивності. Express дозволяє оптимізувати продуктивність вебдодатку через компресію відповідей, кешування статичних файлів, та використання асинхронного коду для неблокуючої обробки запитів.

Використання Express у розробці торгової онлайн-платформи значно спрощує створення надійної та масштабованої серверної частини, що є важливим для забезпечення стабільної роботи платформи та ефективної взаємодії з користувачами.

2.3.2 Безпека

Безпека є критично важливою складовою в розробці торгової онлайн-платформи, оскільки вона обробляє чутливі дані користувачів та забезпечує

проведення фінансових транзакцій. Використання VueJS і NodeJS надає розробникам ряд можливостей для забезпечення високого рівня безпеки додатку.

Основні аспекти забезпечення безпеки наведено нижче.

Захист даних користувачів: важливо забезпечити захист персональних даних користувачів. Це включає використання HTTPS для шифрування трафіку між клієнтом і сервером, а також застосування надійних методів шифрування для зберігання паролів і персональної інформації в базі даних (наприклад, bcrypt).

Аутентифікація та авторизація: ефективне управління сесіями користувачів і доступом до ресурсів. JWT (JSON Web Token) забезпечує високу безпеку за рахунок використання підписаних токенів, що гарантує цілісність даних і запобігає їх несанкціонованому зміненню. Завдяки відкритим стандартам (RFC 7519), JWT є сумісним з різними системами, що полегшує його інтеграцію в різні платформи. Крім того, цей підхід дозволяє аутентифікувати користувачів без необхідності звертатися до центрального сервера, що робить його ідеальним для розподілених систем і мікросервісів. Гнучкість JWT дозволяє передавати будь-які дані в токені, що забезпечує додаткову зручність при розробці вебдодатків і сервісів.

Захист від загальних вебвразливостей: важливо захистити додаток від розповсюджених атак, таких як Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), SQL Injection (SQLi) та інших. Для цього можна використовувати бібліотеки, наприклад, helmet для Express, яка допомагає в налаштуванні заголовків HTTP для захисту від низки вразливостей.

Ізоляція та обмеження доступу до системних ресурсів: використання контейнерізації (наприклад, Docker) та обмеження прав доступу на рівні операційної системи допомагає зменшити ризик небажаного доступу до системних ресурсів.

Логування та моніторинг: запис подій і моніторинг системи дозволяють виявляти та реагувати на безпекові інциденти вчасно. Логування доступу до

ресурсів та використання інструментів аналізу логів може допомогти у виявленні аномальної поведінки.

Регулярне оновлення та патчування: підтримка актуальності всіх компонентів системи, включаючи бібліотеки та залежності, забезпечує захист від відомих вразливостей. Використання інструментів, таких як npm audit у Node.js, допомагає виявляти та виправляти вразливості в залежностях.

Обмеження прав доступу до API: контроль доступу на рівні API (наприклад, через OAuth scopes) забезпечує, що користувачі та системні компоненти мають доступ лише до тих ресурсів, які їм необхідні для роботи.

Ці методи та практики формують основу забезпечення безпеки торгової онлайн-платформи, що розробляється з використанням VueJS та NodeJS, і допомагають забезпечити надійність, конфіденційність, доступність, та цілісність даних і системи в цілому.

2.3.3 Робота з базою даних MySQL

У процесі розробки торгової онлайн-платформи особливе місце займає інтеграція з базою даних MySQL. Використання MySQL як системи управління базами даних дозволяє ефективно зберігати, оновлювати та отримувати дані, необхідні для функціонування платформи.

Розглянемо основні аспекти роботи з MySQL.

Моделювання бази даних: структура бази даних розробляється з урахуванням всіх вимог до зберігання інформації про товари, користувачів, замовлення, статистику продажів тощо. Нормалізація даних здійснюється для оптимізації зберігання та забезпечення цілісності даних.

Взаємодія з базою даних: засоби NodeJS, зокрема пакети як mysql2, використовуються для створення з’єднань з базою даних і виконання SQL-запитів. Це забезпечує ефективну роботу з даними, включаючи вставку, оновлення, видалення та вибірку інформації.

Оптимізація запитів: оптимізація SQL-запитів виконується для підвищення продуктивності взаємодії з базою даних. Індексація, використання кешування та планування запитів допомагає зменшити час відповіді сервера при великому обсязі даних.

Безпека доступу до даних: забезпечення безпеки при роботі з базою даних включає реалізацію механізмів аутентифікації, авторизації та шифрування даних. Використання параметризованих запитів та ORM-інструментів допомагає уникнути SQL-ін'єкцій та інших вразливостей.

Міграція та версіонування бази даних: для управління змінами в структурі бази даних використовуються механізми міграцій, які дозволяють контрольовано оновлювати схему бази даних, забезпечуючи сумісність зі старими версіями додатку та мінімізацію ризиків при впровадженні нових функцій.

Ці підходи та методики є важливою складовою успішної розробки торгової онлайн-платформи, дозволяючи забезпечити стабільність, безпеку та високу швидкість обробки даних.

2.3.4 Кешування

Використання кешування, наприклад, через Redis, для зменшення навантаження на базу даних MySQL та прискорення відгуку сервера на запити є ключовим для високопродуктивних систем.

2.3.5 Логування та моніторинг

Систематичне логування операцій, які відбуваються в базі даних, та моніторинг стану серверів допомагає виявляти та оперативно реагувати на потенційні проблеми, забезпечуючи надійність та доступність сервісу.

2.4 Вибір бази даних та її структура

При виборі бази даних для онлайн-торговельної платформи необхідно враховувати кілька ключових факторів, таких як швидкість обробки запитів, масштабованість, безпека, доступність і зручність інтеграції з використовуваними технологіями.

Для вебплатформ, які мають велику кількість транзакцій та користувацьких запитів, популярними виборами є реляційні СУБД (наприклад, MySQL, PostgreSQL) та NoSQL СУБД (наприклад, MongoDB, Cassandra).

Реляційні бази даних ідеально підходять для систем, де необхідна строга цілісність даних і складні запити. Вони підтримують транзакції, що важливо для обробки платежів та замовлень.

NoSQL бази даних пропонують велику масштабованість і гнучкість схем даних, що добре підходить для динамічних додатків, де часто змінюються вимоги до даних. Вони часто використовуються для зберігання великих обсягів структурованих та неструктурованих даних.

Структура бази даних повинна бути розроблена таким чином, щоб оптимально підтримувати всі необхідні бізнес-процеси платформи. Приклад типової структури бази даних для онлайн-торговельної платформи, використовуючи реляційну СУБД винесено у додаток А «Приклад типової структури бази даних».

Вибір між реляційною та NoSQL базою даних залежить від специфіки бізнесу, вимог до обробки та аналізу даних, масштабу додатку, і можливостей команди, яка буде розробляти та підтримувати систему.

Для більшості торгових платформ реляційні бази даних вважаються надійним вибором через їх сильні сторони в транзакційності та консистентності даних.

2.5 Проєктування системи безпеки

Проєктування системи безпеки для онлайн-торговельної платформи є критичним аспектом, що вимагає всебічного підходу для забезпечення захисту даних користувачів, транзакцій та самої платформи. Це включає в себе заходи на рівні програмного забезпечення, інфраструктури та процедур управління.

Ось ключові аспекти системи безпеки, які потрібно враховувати при проєктуванні:

- багатофакторна аутентифікація (MFA): додавання додаткового рівня перевірки, крім пароля, такого як SMS-коди, електронна пошта або аутентифікатори;
- оновлення паролів: політика регулярної зміни паролів та вимоги до їх складності;
- обмеження спроб входу: ліміти на кількість невдалих спроб входу для запобігання атакам за допомогою підбору;
- принцип найменших привілеїв: надання доступу до системних ресурсів тільки в обсязі, необхідному для виконання робочих завдань;
- рольовий контроль доступу (RBAC): розділення доступу до інформації та функціональності на основі ролей користувачів;
- аудит та логування дій: збір і аналіз логів для виявлення незвичайної поведінки та потенційних порушень;
- шифрування: використання сучасних алгоритмів шифрування для захисту даних користувачів при зберіганні (шифрування диска) та передачі (SSL/TLS);
- маскування даних: застосування маскування для чутливих даних у відображеннях і звітах;
- резервне копіювання та відновлення: регулярне резервне копіювання даних та розробка стратегій швидкого відновлення після інцидентів;
- захист від XSS та CSRF атак: реалізація заходів для захисту від міжсайтового скриптування (XSS) і міжсайтової підробки запитів (CSRF);

- безпечне програмування та розробка: використання методологій безпечної розробки для запобігання вразливостям у коді;
- використання Web Application Firewall (WAF): застосування WAF для моніторингу і блокування шкідливого трафіку до додатку;
- системи виявлення та запобігання вторгненням (IDS/IPS): моніторинг мережі та систем на наявність підозрілих дій;
- управління інцидентами: розробка плану реагування на інциденти, включаючи процедури ідентифікації, розслідування, усунення наслідків та відновлення роботи;
- законодавче регулювання: впровадження політик та процедур, які відповідають місцевим та міжнародним законам про захист даних (наприклад, GDPR, HIPAA).

Врахування цих аспектів під час проєктування системи безпеки допоможе забезпечити захист онлайн-торговельної платформи від потенційних загроз та знизити ризики втрати даних та репутації.

3 РОЗРОБКА КЛІЄНТСЬКОЇ ТА СЕРВЕРНОЇ ЧАСТИНИ

3.1 Розробка клієнтської частини з використанням VueJS

Розробка клієнтської частини онлайн-платформи з використанням VueJS включає створення модульних компонентів для різних частин сайту, таких як глобальні компоненти, домашня сторінка, деталі продукту, оформлення замовлення, інформаційна панель користувача та адміністративна панель. Цей підхід забезпечує легкість розробки, тестування та підтримки коду.

3.1.1 Створення основних компонентів інтерфейсу

Компонент заголовка (Header.vue) – містить логотип, навігаційні посилання та рядок пошуку.

Компонент нижнього колонтитула (Footer.vue) – містить інформаційні посилання, іконки соціальних мереж та додаткову навігацію по сайту.

Компонент головного слайдера (MainSlider.vue) – відображає основні акції, які можна переглянути, провівши пальцем або натиснувши на них.

Компонент списку категорій товарів (CategoryList.vue) – відображає всі категорії товарів у вигляді списку з горизонтальною прокруткою.

Компонент списку товарів (ProductList.vue) – відображає список усіх товарів обраної категорії.

Компонент картки товару (ProductCard.vue) – картка для відображення зображення товару, короткого опису та ціни. Може використовуватися на будь-якій сторінці з переліком товарів.

Компонент featured products (FeaturedProducts.vue) – розділ, який використовує ProductCard.vue для відображення популярних товарів.

Компонент галереї продуктів (ProductGallery.vue) – відповідає за відображення зображень продуктів з можливістю масштабування.

Компонент product info (ProductInfo.vue) – відображає детальну інформацію про продукт, включаючи такі параметри, як розмір або колір.

Компонент «Додати до кошика» (AddToCartButton.vue) – кнопка для додавання товарів до кошика.

Компонент відгуків клієнтів (CustomerReviews.vue) – відображає відгуки та оцінки від покупців.

Компонент підсумку кошика (CartSummary.vue) – відображає підсумок всіх товарів у кошику, з можливістю редагування кількості або видалення товарів.

Компонент форми доставки (ShippingForm.vue) – збирає інформацію про доставку від користувача.

Компонент варіантів оплати (PaymentOptions.vue) – дозволяє користувачеві вибрати спосіб оплати і ввести платіжні реквізити.

Компонент форми профілю (ProfileForm.vue) – дозволяє користувачам оновлювати інформацію про свій профіль.

Компонент історії замовлень (OrderHistory.vue) – показує минулі замовлення та їх поточний статус.

Компонент списку бажань (Wishlist.vue) – відображає всі елементи, які користувач додав до свого списку бажань.

Компонент управління продуктами (ProductManagement.vue) –надає інтерфейс для додавання, редагування або видалення продуктів.

Компонент управління замовленнями (OrderManagement.vue) – відображає список усіх замовлень з інструментами для оновлення їхнього статусу або перегляду більш детальної інформації.

Компонент управління користувачами (UserManagement.vue) –дозволяє адміністрували облікові записи та дозволи користувачів.

Ця структура на основі компонентів допоможе ефективно керувати розвитком онлайн-маркетплейсу, гарантуючи, що інтерфейс залишається модульним і підтримуваним.

Усі реалізовані компоненти інтерфейсу наведено на рисунку 3.1.

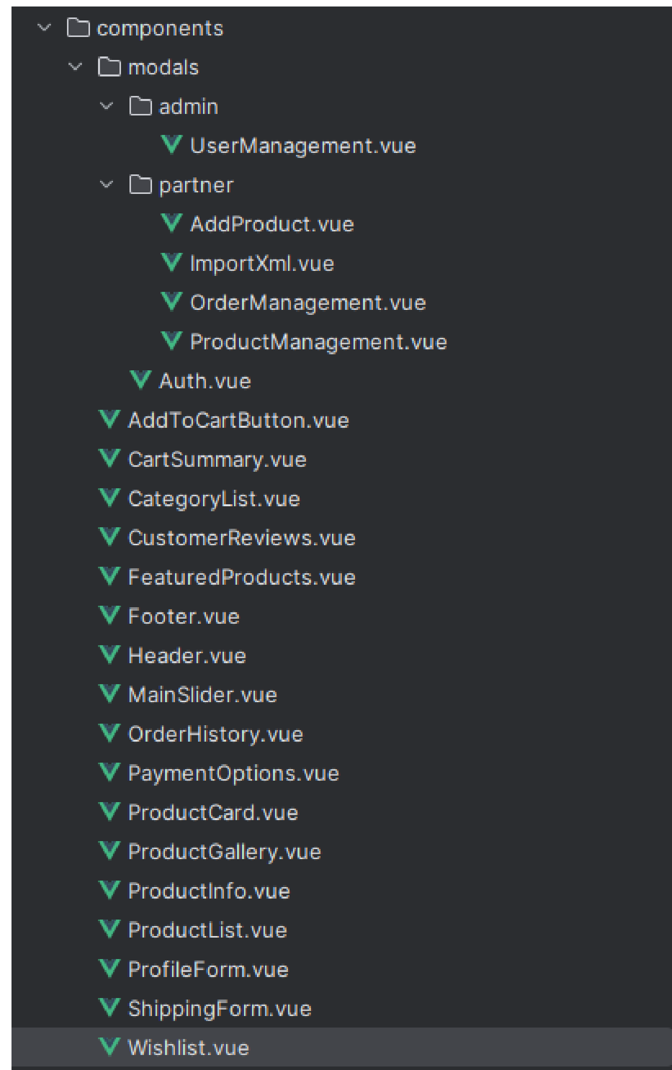

Рисунок 3.1 – Компоненти інтерфейсу

3.1.2 Робота зі станом додатка

Для управління станом додатку використовується Pinia, сучасна альтернатива Vuex для Vue.js 3. Pinia забезпечує ефективне управління станом через централізоване сховище, що дозволяє легко отримувати доступ до даних та реактивно відображати їх у компонентах Vue. Використання Pinia сприяє підвищенню продуктивності та спрощенню архітектури додатка, оскільки цей інструмент надає зручні API для створення та маніпуляції станом.

Pinia ініціалізується та підключається до Vue додатка як стандартний плагін (див. рис 3.2). Це дозволяє легко інтегрувати стан у кореневий компонент Vue.

```
import { createPinia } from 'pinia';
import { createApp } from 'vue';
import App from './App.vue';

const app = createApp(App);
const pinia = createPinia();
app.use(pinia);
app.mount('#app');
```

Рисунок 3.2 – Ініціалізація Pinia

Сховище (store) у Pinia використовується для організації та зберігання різних частин стану додатка. Кожен стор має свій власний стан, гетери (getters) та дії (actions). Компоненти Vue можуть використовувати стори, отримуючи доступ до стану та викликаючи дії безпосередньо, що забезпечує реактивне оновлення UI при зміні стану. Визначення сховища наведено на рисунку 3.3.

```
import { defineStore } from 'pinia';

export const useUserStore = defineStore('user', {
  state: () => ({
    userData: null,
    isLoggedIn: false,
  }),
  getters: {
    isAuthenticating: (state) => state.userData !== null,
  },
  actions: {
    logIn(user) {
      this.userData = user;
      this.isLoggedIn = true;
    },
    logOut() {
      this.userData = null;
      this.isLoggedIn = false;
    },
  },
});
```

Рисунок 3.3 – Визначення сховища у Pinia

Pinia дозволяє легко розділити стан на модулі, що спрощує управління станом великих додатків. Кожен аспект стану може бути інкапсульований в окремі стори, що підтримують чітку структуру та забезпечують легку підтримку коду.

Pinia може інтегруватися з іншими бібліотеками, такими як Vue Router для синхронізації стану з URL або Axios для здійснення HTTP-запитів, зберігаючи відповіді в стані.

3.1.3 Взаємодія із сервером через API

Для взаємодії з сервером використовується Axios, популярна бібліотека JavaScript для виконання HTTP-запитів. Axios забезпечує простий та зрозумілий інтерфейс для взаємодії з REST API, підтримку промісів для асинхронної обробки даних та гнучкі методи для конфігурації запитів. Це допомагає розробникам легко інтегрувати мережеві запити у додаток, а також забезпечує надійний механізм для обробки помилок та повторних спроб.

Axios – це популярна бібліотека в JavaScript, яка використовується для здійснення HTTP-запитів з браузера. Вона надає зручний API для виконання запитів GET, POST, PUT, DELETE та інших методів.

Під час роботи з API важливо коректно обробляти відповіді сервера та помилки. Axios дозволяє легко керувати цим процесом, використовуючи Promise (приклад виконання POST запиту наведено на рисунку 3.4).

```
axios.post('/orders', { productId: 123, quantity: 1 })
  .then(response => {
    console.log('Order created:', response.data);
  })
  .catch(error => {
    console.error('Error creating order:', error);
});
```

Рисунок 3.4 – Приклад обробки відповіді від API серверу

Для захисту даних та управління доступом до API, важливо використовувати аутентифікацію, часто через токени (наприклад, JWT). Axios дозволяє легко налаштувати заголовки для кожного запиту.

3.1.4 Реалізація аутентифікації та авторизації

Аутентифікація та авторизація користувачів в системі реалізовані за допомогою JWT (JSON Web Tokens).JWT дозволяє безпечно передавати інформацію між клієнтом та сервером як JSON-об’єкт, що може бути підписаний за допомогою секретного ключа. Це забезпечує, що дані не можуть бути підроблені чи змінені без відома обох сторін.

Аутентифікація з JWT використовує токени доступу для підтвердження ідентичності користувача та токени оновлення для продовження сесії без потреби повторного введення облікових даних.

Після реалізації аутентифікації користувачів треба додати токен JWT до параметрів axios, щоб авторизувати подальші запити до сервера. Додавання параметру «jwt» до запиту наведено на рисунку 3.5.

```
axios.interceptors.request.use((request) => {
  if (!request.params) {
    request.params = {};
  }
  request.params.jwt = useUserStore().userData?.jwt;
  return request;
});
```

Рисунок 3.5 – Axios інтерцептор для додавання JWT

Для контролю доступу до маршрутів на основі стану аутентифікації можна використовувати перевірки у Vue Router (див. рис. 3.6).

```
import { createRouter, createWebHistory } from 'vue-router';
import { useUserStore } from '@/stores/user';
const routes = [
  { path: '/', name: 'Home', component: () => import('@/components/Home.vue') },
  { path: '/protected', name: 'Protected', component: () => import('@/components/Protected.vue'), meta: { requiresAuth: true } }
];
const router = createRouter({
  history: createWebHistory(),
  routes
});
router.beforeEach((to, from, next) => {
  const userStore = useUserStore();
  if (to.meta.requiresAuth && !userStore.isLoggedIn) {
    next({ url: '/' });
  } else {
    next();
  }
});
export default router;
```

Рисунок 3.6 – Vue Router з захищеними маршрутами

3.1.5 Обробка помилок

Якщо серверна частина додатку поверне критичну помилку, то на сторінці додатку з’явиться плашка з інформацією про помилку (приклад наведено на рисунку 3.7).

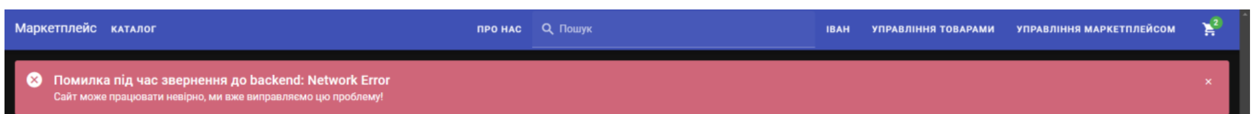

Рисунок 3.7 – Приклад інформаційної плашки про помилку сервера

3.2 Розробка серверної частини з використанням NodeJS

Розробка серверної частини онлайн-платформи здійснюється за допомогою Node.js, що забезпечує масштабованість і високу продуктивність. Використання Node.js дозволяє створювати ефективні серверні додатки, які можуть обробляти велику кількість запитів у режимі реального часу.

3.2.1 Створення сервера з використанням Express

Серверна частина додатку побудована на базі Node.js з використанням фреймворку Express. Express спрощує розробку серверних додатків за допомогою мідлварів та роутингу, забезпечуючи потужний набір функцій для веб та мобільних додатків [8]. Завдяки Express, сервер легко масштабується, підтримує різноманітні розширення та плагіни, що робить його ідеальним для обробки запитів і відправки відповідей клієнту.

Основні етапи створення сервера за допомогою Express [10]:

– створення нового каталогу проєкту та ініціація проєкту Node.js;
– здійснення встановлення Express та необхідних пакетів за допомогою npm;
– створення базової структури сервера (див. рис. 3.8);
– організація маршрутизації та контролерів;
– підключення до бази даних;
– запуск сервера.

```
const express = require('express');
const cors = require('cors');
const morgan = require('morgan');

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

app.use(cors());
app.use(morgan('tiny'));

app.get('/', (req, res) => {
  res.send('Welcome to the Trading Platform API!');
});

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

Рисунок 3.8 – Створення базової структури сервера

3.2.2 Розробка API для взаємодії з клієнтською частиною

API сервера розроблено для забезпечення взаємодії з клієнтською частиною додатку. Використання REST архітектури дозволяє легко інтегрувати різні частини системи, забезпечуючи стандартизований інтерфейс для CRUD операцій (створення, читання, оновлення, видалення). API підтримує аутентифікацію та авторизацію, валідацію даних, а також обробку помилок для надійної роботи додатку.

Планування API.

Спочатку необхідно визначити основні ресурси, якими буде оперувати API. Наприклад, для торгової платформи основними ресурсами можуть бути продукти, замовлення, користувачі тощо.

Важливо спланувати методи доступу до цих ресурсів, зазвичай використовуючи стандартні HTTP-методи: GET для отримання даних, POST для створення нових записів, PUT або PATCH для оновлення існуючих, та DELETE для їх видалення.

Реалізація маршрутів API.

Маршрути слід організувати таким чином, щоб вони логічно відповідали структурі ресурсів та дій, які можна з ними виконувати.

Кожен маршрут повинен мати відповідний обробник, який імплементує логіку доступу до даних.

Використання розширень та мідлварів.

Для зручності розробки та забезпечення додаткового функціоналу використовуються різноманітні мідлвари. Наприклад, cors для налаштування політики CORS.

Для логування та діагностики можна використовувати morgan або аналогічні бібліотеки.

Реалізація контролерів.

Контролери обробляють бізнес-логіку, що стосується обробки запитів. Це дозволяє відокремити логіку взаємодії з базою даних від маршрутизації,

спрощуючи розробку та тестування.

Тестування API.

Для забезпечення стабільності та надійності API важливо проводити його тестування. Використовуються як автоматичні тести (наприклад, з використанням Postman, Jest, Mocha), так і ручне тестування.

Приклад реалізації API для ресурсу «продукти» наведено у додатку Б «Приклад реалізації API для ресурсу “продукти”».

Список реалізованих контролерів, маршрутів та моделей наведено на рисунку 3.9.

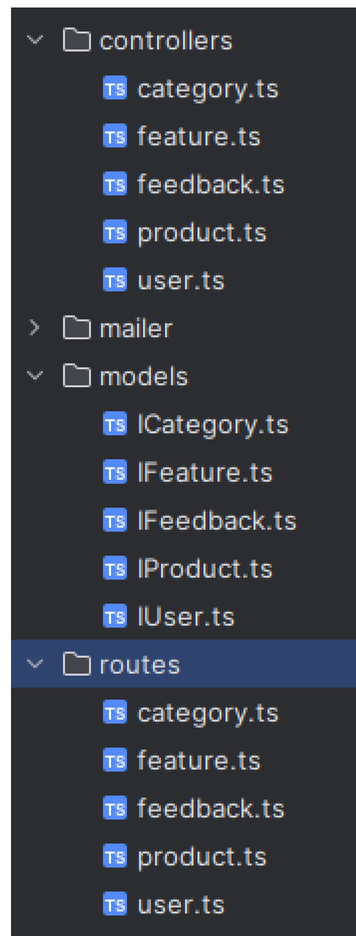

Рисунок 3.9 – Список реалізованих контролерів, маршрутів та моделей

Для доступу до функцій, які потребують ідентифікації користувача (додавання та видалення товарів і тд) було реалізовано систему аутентифікації

за допомогою JWT (див. рисунок 3.11). Для більш комфортної розробки було розроблено middleware для перевірки аутентифікації користувача (див. рисунок 3.12). Якщо верифікація токену не буде пройдено – буде повернуто JSON об'єкт (див. рис. 3.10).

```
{ success: false, error: `Аутентифікація невірна`, reauth: true }
```

Рисунок 3.10 – Відповідь сервера при недійсному токену

Цей підхід до розробки API забезпечує чітку структуру та легкість управління кодом, дозволяє ефективно інтегрувати клієнтську частину торгової платформи з сервером та сприяє гнучкості розширення функціональності системи.

```
export async function getUserByJWT(token: string | string[] | undefined) {
  if (!token) {
    return null;
  }

  if (Array.isArray(token)) {
    token = token[0];
  }

  const user = jwt.verify(token, process.env.JWT_SECRET_KEY);
  if (user) {
    return {
      ...user,
      jwt: token,
    };
  }

  return null;
}
```

Рисунок 3.11 – Генерація JWT для аутентифікованого користувача

```
app.use(async (req: Request, res: Response, next: NextFunction) => {
  const headers = req.headersDistinct;
  const jwt = headers['authorization'];

  if (jwt) {
    const user = await getUserByJWT(jwt);
    if (!user) {
      return res.json({
        success: false,
        error: `Аутентифікація невірна`,
        reauth: true
      });
    }
  }

  next();
});
```

Рисунок 3.12 – Middleware для перевірки аутентифікації користувача

3.2.3 Обробка запитів і керування даними

Обробка запитів на сервері виконується через маршрутизацію Express, яка дозволяє визначити обробники для різних HTTP-запитів по специфічних URL-адресах. Робота з даними ведеться через будувач запитів Knex, що дозволяє виконувати операції з базою даних MySQL без прямого написання SQL-запитів. Knex забезпечує об’єктно-реляційне відображення, що спрощує роботу з даними та знижує ймовірність помилок.

Розглянемо основні аспекти обробки запитів і керування даними.

Структуризація обробки запитів. При розробці API важливо забезпечити чітку структуру обробки запитів. Кожен запит повинен проходити через ряд мідлварів, які можуть виконувати логування, валідацію даних,

аутентифікацію користувачів та інші важливі функції. Обробники запитів (контролери) мають бути відповідальними за виконання бізнес-логіки, а також за формування відповіді клієнту.

Використання мідлварів для оптимізації обробки. CORS (Cross-Origin Resource Sharing) важливий для контролю доступу до ресурсів API з інших доменів, morgan використовується для логування запитів, що допомагає у діагностиці та моніторингу роботи сервера.

Взаємодія з базою даних. Для роботи з реляційними базами даних, такими як MySQL, можна використовувати бібліотеку Knex.js, яка є потужним інструментом SQL-запитів та побудови схеми для Node.js. Knex дозволяє взаємодіяти з базою даних за допомогою високорівневих абстракцій, підтримуючи будівничий запитів, міграції, індексацію та інші операції.

Налаштування Knex.js. Першим кроком є інсталяція Knex та драйвера для MySQL. Для цього потрібно встановити пакети knex та mysql2: npm install knex mysql2. Створення файлу конфігурації Knex для налаштування з'єднання з базою даних наведено на рисунку 3.13.

Використання Knex для обробки даних. З Knex можна легко створювати таблиці, вставляти записи, виконувати запити та оновлювати дані, використовуючи зручний JavaScript API.

Транзакції. Knex підтримує транзакції, що забезпечує консистентність даних при виконанні комплексних операцій.

Реалізація CRUD операцій. Центральним елементом API є реалізація CRUD операцій для ресурсів додатку. Кожна операція має відповідати конкретному маршруту та методу HTTP.

Приклад створення таблиці за допомогою Knex наведено на рисунку 3.14;

Приклад вставки даних у таблицю наведено на рисунку 3.15.

Приклад виконання запиту на вибірку наведено на рисунку 3.16.

Приклад запиту на оновлення даних наведено на рисунку 3.17.

Приклад на видалення даних наведено на рисунку 3.18.

```
Knex({
  client: KnexMysql,
  connection: {
    host: process.env.MYSQL_HOST,
    port: Number(process.env.MYSQL_PORT),
    user: process.env.MYSQL_USER,
    password: process.env.MYSQL_PASS,
    database: process.env.MYSQL_DB,
    timezone: '+03:00',
    enableKeepAlive: true,
    keepAliveInitialDelay: 180_000,
  },
  pool: {
    min: 0,
    max: 1000,
    propagateCreateError: false,
  },
}
```

Рисунок 3.13 – Ініціалізація Knex

```
const knex = require('knex')(require('./knexfile').development);
knex.schema.createTable('products', table => {
  table.increments('id').primary();
  table.string('name').notNullable();
  table.decimal('price', 14, 2).notNullable();
  table.text('description');
  table.timestamps(true, true);
}).then(() => console.log("Table 'products' created"))
  .catch(err => console.error("Error creating table 'products':", err))
  .finally(() => knex.destroy());
```

Рисунок 3.14 – Приклад створення таблиці за допомогою Knex

```
knex('products').insert({
  name: 'New Product',
  price: 99.99,
  description: 'A new amazing product'
}).then(() => console.log("Product inserted"))
  .catch(err => console.error("Error inserting product:", err));
```

Рисунок 3.15 – Приклад вставки даних у таблицю

```
knex.select('*').from('products')
  .then(products => console.log('Products:', products))
  .catch(err => console.error("Error fetching products:", err));
```

Рисунок 3.16 – Приклад виконання запиту на вибірку

```
knex('products').where({ id: 1 }).update({
  price: 79.99,
}).then(() => console.log("Product updated"))
  .catch(err => console.error("Error updating product:", err));
```

Рисунок 3.17 – Приклад запиту на оновлення даних

```
knex('products').where({ id: 1 }).del()
  .then(() => console.log("Product deleted"))
  .catch(err => console.error("Error deleting product:", err));
```

Рисунок 3.18 – Приклад на видалення даних

Ці підходи і методики в обробці запитів і керуванні даними дозволяють забезпечити високу продуктивність, надійність та масштабованість серверної частини торгової онлайн-платформи. Використання чіткої архітектури та засобів розробки в Express сприяє ефективній інтеграції з клієнтською частиною та іншими компонентами системи.

3.3 Інтеграція з базою даних

Інтеграція з базою даних є ключовим аспектом розробки онлайн-платформи, забезпечуючи зберігання та доступ до необхідних даних.

3.3.1 Взаємодія з базою даних MySQL

Інтеграція з базою даних MySQL виконана за допомогою Knex, який служить як інструмент будування запитів. Knex дозволяє розробникам

використовувати JavaScript для виконання операцій з базою даних, забезпечуючи безпечний та ефективний спосіб керування даними. Це включає створення таблиць, запити на вибірку, вставку, оновлення та видалення даних, а також міграції та управління схемами бази даних.

3.3.2 Оптимізація запитів і робота з індексами

Оптимізація запитів та ефективна робота з індексами в базі даних є критично важливими для забезпечення швидкої реакції додатку на дії користувачів та ефективного керування великим об’ємом даних. В цьому контексті важливо зосередитися на належному проєктуванні запитів та стратегічному використанні індексів для підвищення продуктивності бази даних.

Оптимізація запитів до бази даних починається з аналізу найбільш часто використовуваних або найбільш вимогливих запитів. Для цього може бути використано аналітичні інструменти, які визначають «вузькі місця» в запитах, дозволяючи розробникам зосередитися на їх оптимізації. Також, важливо звернути увагу на структуру SQL-запитів, зокрема, використання підзапитів, з’єднань (joins), та умовних операторів, які можуть впливати на продуктивність [9].

Індекси грають ключову роль у підвищенні швидкості пошуку даних. Важливо розуміти, які поля в таблицях найчастіше використовуються для пошуку чи сортування, і створювати індекси на цих полях. Правильне використання індексів може значно знизити кількість часу, необхідного для виконання запитів, оскільки індекс дозволяє базі даних ефективно локалізувати дані без необхідності сканування всієї таблиці.

Крім того, слід враховувати типи індексів, такі як унікальні індекси, які забезпечують унікальність даних у певному полі, та композитні індекси, які включають кілька полів. Вибір правильного типу індексу залежить від

конкретних вимог до запитів і може суттєво вплинути на продуктивність системи.

Оптимізація бази даних – це неодноразовий процес. Регулярний моніторинг продуктивності бази даних і аналіз запитів дозволяють виявити нові можливості для оптимізації. Важливо також періодично переглядати і оновлювати індекси, особливо в умовах зміни обсягів даних або зміни шаблонів доступу до даних.

За допомогою цих методик розробники можуть забезпечити високу продуктивність додатків, ефективно керувати великими обсягами даних і забезпечувати швидку відповідь на запити користувачів.

3.3.3 Забезпечення цілісності даних

Забезпечення цілісності даних є фундаментальною складовою при проєктуванні та розробці будь-якої інформаційної системи. Цілісність даних забезпечує точність, послідовність та надійність інформації, що зберігається в базі даних, і є критичною для прийняття обґрунтованих рішень та підтримки безперервної бізнес-діяльності.

Методи забезпечення цілісності даних наступні.

Встановлення обмежень у базі даних допомагає гарантувати, що дані вводяться у визначеному форматі та діапазоні. Обмеження включають унікальні ключі, первинні ключі, зовнішні ключі для визначення взаємозв'язків між таблицями, а також перевірки (check constraints), що забезпечують введення допустимих значень.

Управління транзакціями з дотриманням принципів ACID (Atomicity, Consistency, Isolation, Durability) допомагає забезпечити, що всі операції з даними виконуються надійно. Атомарність гарантує, що транзакція або повністю виконується, або не виконується зовсім; послідовність забезпечує дотримання всіх обмежень бази даних; ізоляція запобігає взаємному впливу

паралельних транзакцій; стійкість гарантує збереження результатів виконаної транзакції.

Ведення логів змін, що відбуваються в базі даних, є важливим для відстеження та аналізу історії транзакцій. Аудит дозволяє ідентифікувати та виправляти помилки, а також проводити розслідування у випадку несанкціонованого доступу або інших порушень.

Забезпечення регулярного бекапування даних та розробка ефективних стратегій відновлення після збоїв є критичними для захисту даних від втрати або пошкодження. Відновлення даних має бути протестоване, щоб гарантувати, що воно може бути швидко здійснене у критичних ситуаціях.

Безпека даних, включаючи шифрування даних, є ще одним важливим аспектом для забезпечення їхньої цілісності. Застосування сильних методів шифрування для зберігання та передачі даних допомагає захистити їх від несанкціонованого доступу та витоку інформації.

Забезпечення цілісності даних не тільки підвищує довіру до інформаційної системи з боку користувачів, але й відіграє критичну роль у підтримці безперервності бізнес-процесів, зниженні ризиків, пов’язаних з обробкою даних, та забезпеченні їхньої актуальності та доступності.

4 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

4.1 План тестування

Тестування є критично важливим етапом розробки, що забезпечує якість та надійність програмного забезпечення. План тестування включає в себе наступні етапи:

- визначення цілей тестування: перевірка функціональності, продуктивності, безпеки та зручності використання онлайн-платформи;
- розробка тестових сценаріїв: включення різних випадків використання, які охоплюють всі основні функції платформи;
- вибір методів тестування: використання як ручного, так і автоматизованого тестування для досягнення максимального покриття;
- визначення середовища тестування: налаштування тестового середовища, яке максимально наближене до реального оточення користувачів.

4.2 Автоматизоване тестування

Для автоматизованого тестування використовуються наступні інструменти:

- JUnit/Jest: для тестування JavaScript коду на рівні компонентів;
- Selenium/WebDriver: для інтеграційного та регресійного тестування вебінтерфейсу;
- Postman: для тестування REST API.

Основні кроки автоматизованого тестування включають:

- розробка тестових скриптів: написання тестів для ключових функцій платформи;

– виконання тестів: запуск тестів та аналіз результатів;
– виправлення помилок: корекція виявлених дефектів і повторне тестування.

4.3 Налагодження та обробка помилок

Налагодження та обробка помилок є важливими аспектами забезпечення стабільної роботи платформи. Вони включають наступні кроки:

– збір логів: використання логування для моніторингу роботи системи та виявлення помилок;
– аналіз логів: аналіз зібраних логів для ідентифікації причин помилок;
– виправлення помилок: розробка та впровадження виправлень для виявлених дефектів;
– перевірка виправлень: повторне тестування для підтвердження усунення помилок.

Ретельне тестування та налагодження допомагають забезпечити високу якість кінцевого продукту, задовольнити потреби користувачів та забезпечити надійність і безпеку торгової онлайн-платформи.

ВИСНОВКИ

Кваліфікаційна робота бакалавра була спрямована на розробку функціональної торгової онлайн-платформи з використанням сучасних технологій Vue.js для клієнтської частини та Node.js для серверної частини. Робота охоплює все: від початкового аналізу і визначення вимог до детального проєктування архітектури, розробки і тестування платформи.

Основні результати дослідження демонструють, що використання комбінації Vue.js та Node.js є вкрай ефективним для створення масштабованих, високопродуктивних вебдодатків, здатних відповідати сучасним вимогам до електронної комерції. Таке технологічне поєднання забезпечує гнучкість управління проєктом, швидкість розробки та високу адаптивність до змінних вимог користувачів.

Реалізація проєкту включала розробку інтерактивних інтерфейсів користувача, ефективну обробку даних на сервері, інтеграцію з базами даних та зовнішніми сервісами. Особлива увага була приділена безпеці, оптимізації продуктивності та забезпеченню належної взаємодії між клієнтською та серверною частинами.

Проєкт підтвердив, що грамотне використання фреймворків Vue.js та Node.js може значно підвищити ефективність розробки та управління сучасними вебдодатками. Отриманий досвід та розроблені рішення можуть бути застосовані для подальших розробок у цій галузі, що робить внесок у загальний розвиток сфери електронної комерції.

Таким чином, виконана робота не тільки забезпечує важливий академічний внесок у вивчення і застосування передових вебтехнологій, але й слугує практичною основою для майбутніх інновацій у створенні ефективних онлайн-торгових платформ.

ПЕРЕЛІК ПОСИЛАНЬ

1. The Progressive JavaScript Framework. URL: https://vuejs.org/ (дата звернення: 10.04.2024).
2. What is JWT (JSON Web Token)? How does JWT Authentication work? URL: https://www.miniorange.com/blog/what-is-jwt-json-web-token-how-does-jwt-authentication-work/ (дата звернення: 10.04.2024).
3. Персоналізація в маркетингу: закордонні кейси та практичні поради для eCommerce. URL: https://turumburum.ua/blog/personalizaciya-v-marketingu-zakordonni-keysi-ta-praktichni-poradi-dlya-ecommerce (дата звернення: 11.04.2024).
4. Virtual fitting in augmented reality. URL: https://aestar.com.ua/en/ar-try-on-clothes/ (дата звернення: 11.04.2024).
5. Web Content Accessibility Guidelines (WCAG) 2.1. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 15.04.2024).
6. New Framework-level Recommendations. URL: https://v3-migration.vuejs.org/recommendations (дата звернення: 15.04.2024).
7. Macrae C. Vue.js: Up and Running. O'Reilly Media, 2018. 174 p.
8. Krol J. Web Development with Mongodb and Node.Js : 3rd ed. Packt Publishing, Limited, 2017. 330 p.
9. DuBois P. MySQL : 4th ed. Pearson Education, 2008. 1304 p.
10. API Design with Node.js. URL: https://hendrixer.github.io/API-design-v4/ (дата звернення: 17.04.2024).
11. Ribeiro H. R. Vue.js 3 Cookbook. Packt Publishing, 2020. 562 p.

ДОДАТОК А

Приклад типової структури бази даних

Users (Користувачі)

user_id: Первинний ключ

username: Ім’я користувача

password: Хеш пароля

email: Електронна адреса

role: Роль на платформі (наприклад, покупець, продавець, адміністратор)

Products (Продукти)

product_id: Первинний ключ

name: Назва продукту

description: Опис продукту

price: Ціна

stock: Кількість на складі

Orders (Замовлення)

order_id: Первинний ключ

user_id: Зовнішній ключ, що посилається на Users

date: Дата замовлення

total: Загальна вартість замовлення

OrderDetails (Деталі замовлення)

order_details_id: Первинний ключ

order_id: Зовнішній ключ, що посилається на Orders

product_id: Зовнішній ключ, що посилається на Products

quantity: Кількість продукту в замовленні

price: Ціна за одиницю товару

Categories (Категорії)

category_id: Первинний ключ

name: Назва категорії

description: Опис категорії

ДОДАТОК Б

Приклад реалізації API для ресурсу “продукти”

```
import { Request, Response } from 'express';
import IProduct from "../models/IProduct";
import {QueryResult, ResultSetHeader} from "mysql2";
import {getUserByJWT} from "../utils";
import IUser from "../models/IUser";
const { XMLParser, XMLBuilder, XMLValidator} = require("fast-xml-parser");


export async function getProducts(req: Request, res: Response): Promise<Response>
{
  const partner = req.query.partner as string;

  let query = `SELECT * FROM products`;
  const params: (string | number)[] = [];

  if (partner) {
    query += ` WHERE owner = ?`;
    params.push(partner);
  }

  const products = await global.mysql.raw(query, params);
  return res.json(products[0]);
}

export async function getProduct(req: Request, res: Response): Promise<Response>
{
  const id = req.params.id;
  const products = await global.mysql.raw(`SELECT * FROM products WHERE id =
```

```
?`, [id])
  return res.json(products[0])
}

export async function createProduct(req: Request, res: Response) {
  const jwt = req.headers['authorization'];
  const user = await getUserByJWT(jwt);

  if (!user) {
    return res.json({
      success: false,
    });
  }

  if (!req.body.xml) {
    const newProduct: IProduct = req.body;

    return res.json(await addProduct(newProduct, user));
  }

  const parser = new XMLParser();
  const products = parser.parse(req.body.xml)?.root?.element;
  const ids: number[] = [];

  console.log(products)

  for (let idx = 0; idx < products.length; idx ++) {
    const res = await addProduct(products[idx], user);

    if (res.success) {
      ids.push(res.id!);
    }
```

```
  }

  return res.json({
    success: true,
    id: ids,
  });
}

async function addProduct(product: IProduct, user: IUser) {
  const images = product.images;
  delete product.images;

  product.owner = Number(user.id);
  product.params = JSON.stringify(product.params || {});

  try {
    const id = (await global.mysql.raw(`INSERT INTO products SET ?`, [product]))[0]
as ResultSetHeader;

    if (images) {
      images.forEach((image) => {
        global.mysql.raw(`INSERT INTO product_images (product, image) VALUES
(?, ?)`, [product.id, image]);
      });
    }

    return {
      success: true,
      id: id.insertId
    };
  } catch (e) {
    console.error(e);
```

```
    return {
      success: false,
      // @ts-ignore
      error: e.message
    };
  }
}

export async function updateProduct(req: Request, res: Response) {
  const jwt = req.headers['authorization'];
  const user = await getUserByJWT(jwt);

  if (!user) {
    return res.json({
      success: false,
    });
  }

  const id = Number(req.body.id);

  if (!await isProductOwner(id, user.id)) {
    return;
  }

  const updateProduct: IProduct = req.body;
  await global.mysql.raw(`UPDATE products SET ? WHERE id = ?`, [updateProduct,
id])
  return res.json({
    success: true,
  });
}
```

```
export async function deleteProduct(req: Request, res: Response) {
  const jwt = req.headers['authorization'];
  const user = await getUserByJWT(jwt);

  if (!user) {
    return res.json({
      success: false,
    });
  }

  const id = Number(req.params.id);

  if (!await isProductOwner(id, user.id)) {
    return;
  }

  await global.mysql.raw(`DELETE FROM products WHERE id = ?`, [id])

  return res.json({
    success: true,
  });
}

async function isProductOwner(productId: number, userId: number) {
  const product: IProduct[] = (await global.mysql.raw('SELECT id FROM products
WHERE id = ? AND owner = ?', [productId, userId]))[0] as IProduct[];

  return product.length >= 1;
}
```

ДОДАТОК В

Код основного файлу клієнтської частини

```
<script setup>
import {onMounted, ref} from "vue";
import router from "@/router/router.js";

import Header from "@/components/Header.vue";
import Footer from "@/components/Footer.vue";
import CategoryList from "@/components/CategoryList.vue";

import axios from "axios";
import {useUserStore} from "@/store/user.js";
import {useProductStore} from "@/store/product.js";
import {useCategoryStore} from "@/store/category.js";
import {useFeatureStore} from "@/store/feature.js";

const categoryShow = ref(false);

const productStore = useProductStore();
const categoryStore = useCategoryStore();
const featureStore = useFeatureStore();

const selectCategory = (category) => {
  router.push(`/products/category=${category}`);

  categoryShow.value = false;
}
```

```
const error = ref('');

axios.interceptors.response.use((response) => response, (err) => {
  error.value = err;
});

axios.interceptors.request.use((request) => {
  request.headers['Authorization'] = useUserStore().userData?.jwt;

  return request;
});

onMounted(async () => {
    let response = await axios.get('http://127.0.0.1:3030/product');
    productStore.set(response.data);

    response = await axios.get('http://127.0.0.1:3030/category');
    categoryStore.set(response.data);

    response = await axios.get('http://127.0.0.1:3030/feature');
    featureStore.set(response.data);
})
</script>

<template>
    <v-app>
        <Header @openCategoryList="categoryShow = !categoryShow"/>
        <v-main>
            <v-alert
                    v-if="error"
```

```
closable
                    class="mb-2"
                    type="error"
                    :title="`Помилка під час звернення до backend:
${error.message}`"
                    :text="`Сайт може працювати невірно, ми вже виправляємо
цю проблему!`"
                />

                <CategoryList v-show="categoryShow"
@selectCategory="selectCategory"/>
                <router-view v-if="!categoryShow"/>
            </v-main>
            <Footer/>
        </v-app>

        <notifications position="bottom right"/>
    </template>

    <style scoped>
    .v-main {
        margin: 1em;
    }
    </style>
```

ДОДАТОК Г

Модуль ініціалізації серверної частини

```
require('dotenv').config();

import { connect } from "./database";
global.mysql = connect;

import Mailer from "./mailer";
global.mailer = Mailer;

import {getUserByJWT} from "./utils";

import express, {NextFunction} from "express";
import { Request, Response } from 'express';
const morgan = require('morgan');
const cors = require('cors');

import CategoryRoutes from "./routes/category";
import ProductRoutes from "./routes/product";
import UserRouters from "./routes/user";
import Feedback from "./routes/feedback";
import Feature from "./routes/feature";

const app = express();
const port = process.env.PORT;
const http = require('http');
const { Server } = require('socket.io');
const server = http.createServer(app);
```

```
const io = new Server(server);

app.use(morgan('tiny'));
app.use(cors({
  origin: process.env.ALLOW_ORIGIN,
  optionsSuccessStatus: 200,
  methods: '*',
}));
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

app.use(async (req: Request, res: Response, next: NextFunction) => {
  const headers = req.headersDistinct;
  const jwt = headers['authorization'];

  if (jwt) {
    const user = await getUserByJWT(jwt);
    if (!user) {
      return res.json({
        success: false,
        error: `Аутентифікація невірна`,
        reauth: true
      });
    }
  }

  next();
});

app.use('/category', CategoryRoutes);
```

```
app.use('/product', ProductRoutes);
app.use('/user', UserRouters);
app.use('/feedback', Feedback);
app.use('/feature', Feature);

app.listen(port, () => {
  console.log(`listening on port ${port}`);
});

io.on('connection', (socket) => {
  console.log(`connected new user to socket: ${socket.id}`);
});
```

ДОДАТОК Д

Код контролерів серверної частини

Д.1 Category.ts

```
import {Request, Response} from 'express';
import ICategory from "../models/ICategory";

export async function getCategories(req: Request, res: Response): Promise<Response> {
  const categories = await global.mysql.select("*").from('categories');
  return res.json(categories);
}

export async function getCategory(req: Request, res: Response): Promise<Response>  {
  const id = req.params.id;
  const categories = await global.mysql.raw(`SELECT * FROM categories WHERE id = ?`, [id])
  return res.json(categories[0])
}

export async function createCategory(req: Request, res: Response) {
  const newPost: ICategory = req.body;
  await global.mysql.raw(`INSERT INTO categories SET ?`, [newPost])
  return res.json({
    message: "Post created"
  });
}
export async function updateCategory(req: Request, res: Response) {
```

```
  const id = req.params.id;
  const updatePost: ICategory = req.body;
  await global.mysql.raw(`UPDATE categories SET ? WHERE id = ?`, [updatePost, id])
  return res.json({
    message: "Post updated"
  })
}

export async function deleteCategory(req: Request, res: Response) {
  const id = req.params.id;
  await global.mysql.raw(`DELETE FROM categories WHERE id = ?`, [id])
  return res.json({
    message: "Post deleted"
  })
}
```

Д.2 Feature.ts

```
import { Request, Response } from 'express';

export async function getFeatures(req: Request, res: Response) {
  return res.json(await global.mysql.select('*').from('features'));
}
```

Д.3 Feedback.ts

```
import { Request, Response } from 'express';
import IFeedback from "../models/IFeedback";
```

```
import {ResultSetHeader} from "mysql2";
import {getUserByJWT} from "../utils";

export async function getFeedbacks(req: Request, res: Response): Promise<Response> {
  const partner = req.query.partner as string;

  let query = `SELECT * FROM feedbacks`;
  const params: (string | number)[] = [];

  if (partner) {
    query += ` WHERE owner = ?`;
    params.push(partner);
  }

  const feedbacks = await global.mysql.raw(query, params);
  return res.json(feedbacks[0]);
}

export async function getFeedback(req: Request, res: Response): Promise<Response>  {
  const id = req.params.id;
  const feedbacks = await global.mysql.raw(`SELECT * FROM feedbacks WHERE id = ?`, [id])
  return res.json(feedbacks[0])
}
export async function createFeedback(req: Request, res: Response) {
  const jwt = req.headers['authorization'];
  const user = await getUserByJWT(jwt);

  if (!user) {
```

```
    return res.json({
      success: false,
    });
  }

  const newFeedback: IFeedback = req.body;
  newFeedback.owner = user.id;

  try {
    const id = (await global.mysql.raw(`INSERT INTO feedbacks SET ?`,
[newFeedback]))[0] as ResultSetHeader;

    return res.json({
      success: true,
      id: id.insertId
    });
  } catch (e) {
    return res.json({
      success: false,
      // @ts-ignore
      error: e.message
    });
  } finally {
  }
}
export async function updateFeedback(req: Request, res: Response) {
  const id = req.params.id;
  const updatePost: IFeedback = req.body;
  await global.mysql.raw(`UPDATE feedbacks SET ? WHERE id = ?`,
[updatePost, id])
  return res.json({
```

```
    message: "Post updated"
  })
}

export async function deleteFeedback(req: Request, res: Response) {
  const id = req.params.id;
  await global.mysql.raw(`DELETE FROM feedbacks WHERE id = ?`, [id])
  return res.json({
    message: "Post deleted"
  })
}
```

Д.4 Product.ts

```
import { Request, Response } from 'express';
import IProduct from "../models/IProduct";
import {QueryResult, ResultSetHeader} from "mysql2";
import {getUserByJWT} from "../utils";
import IUser from "../models/IUser";
const { XMLParser, XMLBuilder, XMLValidator} = require("fast-xml-parser");

export async function getProducts(req: Request, res: Response): Promise<Response> {
  const partner = req.query.partner as string;

  let query = `SELECT * FROM products`;
  const params: (string | number)[] = [];

  if (partner) {
```

```
    query += ` WHERE owner = ?`;
    params.push(partner);
  }

  const products = await global.mysql.raw(query, params);
  return res.json(products[0]);
}

export async function getProduct(req: Request, res: Response): Promise<Response> {
  const id = req.params.id;
  const products = await global.mysql.raw(`SELECT * FROM products WHERE id = ?`, [id])
  return res.json(products[0])
}

export async function createProduct(req: Request, res: Response) {
  const jwt = req.headers['authorization'];
  const user = await getUserByJWT(jwt);

  if (!user) {
    return res.json({
      success: false,
    });
  }

  if (!req.body.xml) {
    const newProduct: IProduct = req.body;

    return res.json(await addProduct(newProduct, user));
  }
```

```
  const parser = new XMLParser();
  const products = parser.parse(req.body.xml)?.root?.element;
  const ids: number[] = [];

  console.log(products)

  for (let idx = 0; idx < products.length; idx ++) {
    const res = await addProduct(products[idx], user);

    if (res.success) {
      ids.push(res.id!);
    }
  }

  return res.json({
    success: true,
    id: ids,
  });
}

async function addProduct(product: IProduct, user: IUser) {
  const images = product.images;
  delete product.images;

  product.owner = Number(user.id);
  product.params = JSON.stringify(product.params || {});

  try {
    const id = (await global.mysql.raw(`INSERT INTO products SET ?`,
[product]))[0] as ResultSetHeader;
```

```
    if (images) {
      images.forEach((image) => {
        global.mysql.raw(`INSERT INTO product_images (product, image)
VALUES (?, ?)`, [product.id, image]);
      });
    }

    return {
      success: true,
      id: id.insertId
    };
  } catch (e) {
    console.error(e);

    return {
      success: false,
      // @ts-ignore
      error: e.message
    };
  }
}

export async function updateProduct(req: Request, res: Response) {
  const jwt = req.headers['authorization'];
  const user = await getUserByJWT(jwt);

  if (!user) {
    return res.json({
      success: false,
    });
  }
```

```
  const id = Number(req.body.id);

  if (!await isProductOwner(id, user.id)) {
    return;
  }

  const updateProduct: IProduct = req.body;
  await global.mysql.raw(`UPDATE products SET ? WHERE id = ?`,
[updateProduct, id])
  return res.json({
    success: true,
  });
}

export async function deleteProduct(req: Request, res: Response) {
  const jwt = req.headers['authorization'];
  const user = await getUserByJWT(jwt);
  if (!user) {
    return res.json({
      success: false,
    });
  }

  const id = Number(req.params.id);

  if (!await isProductOwner(id, user.id)) {
    return;
  }

  await global.mysql.raw(`DELETE FROM products WHERE id = ?`, [id])
```

```
  return res.json({
    success: true,
  });
}

async function isProductOwner(productId: number, userId: number) {
  const product: IProduct[] = (await global.mysql.raw('SELECT id FROM products WHERE id = ? AND owner = ?', [productId, userId]))[0] as IProduct[];

  return product.length >= 1;
}
```

Д.5 User.ts

```
import { Request, Response } from 'express';
import IUser from "../models/IUser";
import * as crypto from "crypto";
import {QueryResult} from "mysql2";
import {getUserByJWT} from "../utils";
const jwt = require('jsonwebtoken')


export async function getUser(req: Request, res: Response): Promise<Response> {
  const email = req.query.email;
  const password = req.query.password as string;

  const hash = crypto.createHmac('sha256', process.env.HASH_KEY as string).update(password).digest('hex');
```

```
const user: IUser[] = (await global.mysql.raw(`SELECT * FROM users WHERE email = ? AND password = ? LIMIT 1`, [email, hash]))[0] as IUser[];

  if (user.length > 0) {
    return res.json({
      success: true,
      data: await getUserFromDb(user[0].id),
    });
  } else {
    return res.json({
      success: false,
      error: 'Невірний логін або пароль'
    });
  }
}

export async function createUser(req: Request, res: Response) {
  const name = req.body.name;
  const email = req.body.email;
  const password = req.body.password;

  if (!global.mailer.isEmailValid(email)) {
    return res.json({
      success: false,
      error: `Невалідна єлектронна пошта`,
    });
  }

  const hash = crypto.createHmac('sha256', process.env.HASH_KEY as string).update(password).digest('hex');
```

```
try {
  const id = await global.mysql.raw(`INSERT INTO users (name, email, password, role) VALUES (?, ?, ?, ?)`, [name, email, hash, 'User']) as QueryResult;

  return res.json({
    success: true,
    data: await getUserFromDb(id[0].insertId),
  });
} catch (e) {
  return res.json({
    success: false,
    // @ts-ignore
    error: e.message
  })
} finally {
}
}

async function createToken(user: IUser) {
 delete user.password;

 const token = jwt.sign(user, process.env.JWT_SECRET_KEY, { expiresIn: '365d' });

 await global.mysql.raw(`INSERT INTO tokens (user, token) VALUES (?, ?)`, [user.id, token]);

 return token;
}

async function getUserFromDb(id) {
```

```
  const user = (await global.mysql.raw(`SELECT * FROM users WHERE id = ?`, [id]))[0][0] as IUser;

  user.jwt = await createToken(user);
  return user;
}

export async function updateUser(req: Request, res: Response) {
  const id = req.params.id;
  const updatePost: IUser = req.body;
  await global.mysql.raw(`UPDATE users SET ? WHERE id = ?`, [updatePost, id])

  return res.json({
    message: "Post updated"
  })
}

export async function deleteUser(req: Request, res: Response) {
  /* const id = req.params.id;
  await global.mysql.raw(`DELETE FROM users WHERE id = ?`, [id])
  */
  return res.json({
    message: "Post deleted"
  })
}

export async function checkUser(req: Request, res: Response) {
  const jwt = req.headers['authorization'];
  const user = await getUserByJWT(jwt);
```

```
    if (!user) {
      return res.json({
        success: false,
      });
    }

    return res.json({
      success: true,
      data: user
    });
  }

  export async function restoreUser(req: Request, res: Response) {
    const email = req.body.email;

    const password = crypto.randomUUID();
    const hash = crypto.createHmac('sha256', process.env.HASH_KEY as string).update(password).digest('hex');

    await global.mysql('users').update({
      password: hash,
    }).where('email', email);

    global.mailer.sendMail(email, 'restore', {
      pass: password
    });

    return res.json({
      success: true
    });
  }
```