МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН-СЕРВІСУ ЗАМОВЛЕННЯ ЛІТЕРАТУРИ»**

Виконала: студентка 4 курсу, групи 6.1210-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Ю.М. Станенко
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(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 СТУДЕНТЦІ

Станенко Юлії Миколаївні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн-сервісу замовлення літератури

керівник роботи Гоменюк Сергій Іванович д.т.н., професор
(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. Опис розробленого програмного забезпечення.

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

6. Консультанти розділів роботи

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

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.	09.01.2024	
2.	Збір вихідних даних.	13.02.2024	
3.	Обробка методичних та теоретичних	12.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	17.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2024	

Студент _____ Ю.М. Станенко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка онлайн-сервісу замовлення літератури»: 63 с., 44 рис., 4 табл., 10 джерел, 1 додаток.

АУТЕНТИФІКАЦІЯ, БАЗА ДАНИХ, ЗАМОВЛЕННЯ ЛІТЕРАТУРИ, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ОНЛАЙН-СЕРВІС, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, API, EXPRESS, MYSQL, NODE.JS, REACT.

Об’єкт дослідження – процес створення програмного продукту.

Предмет дослідження – процес розробки онлайн-сервісу замовлення літератури, його етапи та виявлення можливих труднощів, які можуть виникнути під час цього процесу.

Мета роботи – провести аналіз етапів розробки онлайн-сервісу замовлення літератури та ідентифікувати потенційні проблеми, що можуть виникнути під час цього процесу.

Метод дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, розробки та тестування програмного забезпечення.

Результати та їх новизна: досліджено етапи розробки онлайн-сервісу замовлення літератури, виявлено основні проблеми та застосовано нові методи для збору та аналізу вимог до програмного забезпечення.

Значення та висновки: дослідження визначило основні етапи розробки онлайн-сервісу замовлення літератури та потенційні труднощі, які можуть виникнути на кожному з них. Це дослідження є корисним для практичної діяльності в галузі програмної розробки, дозволяючи розробляти стратегії вирішення проблем на ранніх стадіях. Робота спрямована на покращення ефективності та якості розробки програмного забезпечення, що є важливим для ІТ-індустрії.

SUMMARY

Bachelor's qualifying paper "Development of an Online Book Ordering Service": 63 pages, 44 figures, 4 tables, 10 references, 1 supplement.

AUTHENTICATION, DATABASE, LITERATURE ORDERING, CLIENT-SERVER ARCHITECTURE, ONLINE SERVICE, SOFTWARE DEVELOPMENT, SOFTWARE TESTING, API, EXPRESS, MYSQL, NODE.JS, REACT.

Object of research – the process of creating a software product.

Subject of research – the process of developing an online literature ordering service, its stages, and identifying possible challenges that may arise during this process.

Purpose of the work – to analyze the stages of developing an online literature ordering service and identify potential problems that may arise during this process.

Research method – methods of gathering and analyzing software requirements, modeling, design, development, and testing methods.

Results and their novelty: the stages of developing an online literature ordering service were studied, the main problems were identified, and new methods for gathering and analyzing software requirements were applied.

Significance and conclusions: the study identified the main stages of developing an online literature ordering service and potential challenges that may arise at each of them. This research is useful for practical activities in the field of software development, allowing strategies to be developed for solving problems at early stages. The work is aimed at improving the efficiency and quality of software development, which is important for the IT industry.

ЗМІСТ

ВСТУП

Ця кваліфікаційна робота присвячена розробці онлайн-сервісу замовлення літератури, який забезпечує зручний інтерфейс для користувачів, дозволяючи їм переглядати, замовляти та керувати замовленнями книг через Інтернет. В роботі проведено детальний аналіз предметної області та існуючих аналогів, що дозволило визначити ключові вимоги до системи та сформулювати завдання для її розробки.

Система побудована на основі клієнт-серверної архітектури з використанням сучасних технологій. Клієнтська частина реалізована за допомогою React, що дозволяє створювати динамічні та інтерактивні інтерфейси користувача. Серверна частина, реалізована на платформі Node.js з використанням Express, забезпечує надійну обробку запитів, взаємодію з базою даних MySQL та безпеку даних через JWT.

Проєктування структури бази даних включало створення таблиць для зберігання даних про користувачів, продукти та замовлення, що дозволяє ефективно виконувати складні запити та забезпечувати швидкий доступ до необхідної інформації.

У процесі тестування та відлагодження системи використовувалися як автоматизовані, так і ручні методи, що забезпечило стабільність та надійність роботи системи. Всі поставлені завдання були успішно виконані, а результати роботи підтверджують доцільність обраних підходів та технологій. Онлайн-сервіс готовий до впровадження та подальшої експлуатації, що забезпечить зручний та надійний спосіб замовлення літератури через Інтернет.

Завданнями дослідження є:

– аналіз предметної області та існуючих аналогів;
– огляд тенденцій розвитку ринку онлайн-замовлення літератури;
– постановка завдання на розробку системи;
– проєктування архітектури та структури системи;

– моделювання предметної області засобами UML;
– проєктування структури бази даних;
– вибір технологій та інструментів розробки;
– реалізація інтерфейсу користувача та серверної частини;
– тестування та відлагодження системи.

Об’єктом дослідження є процес замовлення літератури через інтернет, а предметом дослідження – онлайн-сервіси, що забезпечують цей процес.

Історично, розвиток онлайн-сервісів для замовлення літератури розпочався з появою перших інтернет-магазинів, таких як Amazon. З часом ці сервіси еволюціонували, пропонуючи користувачам все більше функціональних можливостей, таких як рекомендаційні системи, рецензії, рейтинги тощо.

В результаті проведеного дослідження було створено прототип онлайн-сервісу замовлення літератури, який відповідає сучасним вимогам та стандартам. Розроблений сервіс забезпечує зручний пошук і замовлення літератури, має інтуїтивно зрозумілий інтерфейс та високу продуктивність.

Теоретичне значення роботи полягає в узагальненні та систематизації знань з проєктування та розробки онлайн-сервісів, а практичне значення – у створенні реального продукту, який може бути впроваджений у комерційне використання.

Новизна даної роботи полягає у використанні сучасних технологій та інструментів розробки, що забезпечують високу продуктивність і зручність користування сервісом.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ АНАЛОГІВ

1.1 Загальна характеристика предметної області

Предметна область онлайн-замовлення літератури охоплює широкий спектр діяльності, пов’язаної з пошуком, замовленням та доставкою книг через інтернет. Вона включає різні аспекти, такі як каталоги книг, системи управління замовленнями, платіжні системи та логістичні процеси доставки.

Основними користувачами онлайн-сервісів для замовлення літератури є індивідуальні читачі, студенти, викладачі, бібліотеки та інші установи, які потребують доступу до книг та інших видів літератури. Каталог літератури є базою даних, що містить інформацію про доступні книги, включаючи назви, авторів, жанри, рецензії, рейтинги, обкладинки та іншу метаінформацію. Це дозволяє користувачам легко знаходити потрібну літературу за допомогою пошукових запитів та фільтрів [1].

Система управління замовленнями відповідає за обробку замовлень, включаючи вибір книг, оформлення замовлення, підтвердження оплати та відстеження статусу замовлення. Вона також може включати функціонал для обробки повернень та обміну книг. Платіжні системи інтегруються з різними платіжними платформами, що дозволяє користувачам здійснювати оплату замовлень за допомогою кредитних карток, електронних гаманців, банківських переказів та інших способів.

Логістика включає процес доставки замовлених книг до кінцевого користувача. Це може охоплювати співпрацю з кур’єрськими службами, поштовими операторами або власну систему доставки. Важливою складовою є також відстеження доставки та забезпечення її своєчасності.

Інтерфейс користувача є вебсайтом або мобільним додатком, через який користувачі взаємодіють із сервісом. Інтерфейс повинен бути інтуїтивно

зрозумілим, зручним та привабливим, щоб забезпечити позитивний досвід користувача. Безпека та конфіденційність включають захист персональних даних користувачів та інформації про їхні замовлення. Це включає використання SSL-сертифікатів, шифрування даних, захист від несанкціонованого доступу та інші заходи безпеки.

Аналітика та звітність є інструментами для аналізу даних про користувачів, замовлення та інші аспекти роботи сервісу. Це дозволяє приймати обґрунтовані рішення щодо покращення сервісу, маркетингових стратегій та оптимізації бізнес-процесів.

Предметна область онлайн-замовлення літератури є динамічною і постійно розвивається завдяки швидким темпам розвитку інформаційних технологій та змінюваним потребам користувачів. Це вимагає від розробників постійного вдосконалення сервісів та впровадження інноваційних рішень для задоволення попиту на сучасному ринку.

1.2 Огляд існуючих онлайн-сервісів для замовлення літератури

На ринку існує багато онлайн-сервісів для замовлення літератури, кожен з яких має свої унікальні особливості, переваги та недоліки. Деякі з найвідоміших сервісів включають Amazon, Barnes & Noble, Google Books, та місцеві онлайн-книгарні. Розглянемо їх детальніше.

Amazon (рис. 1.1) є одним з найбільших і найпопулярніших онлайн-сервісів для замовлення літератури. Він пропонує величезний асортимент книг, від класичної літератури до сучасних бестселерів, як у друкованому, так і в електронному форматах. Користувачі можуть скористатися рецензіями та рейтингами для вибору книг, а також рекомендаціями на основі попередніх покупок.

Google Books (рис. 1.2) є частиною екосистеми Google і пропонує доступ

до мільйонів книг у цифровому форматі. Цей сервіс дозволяє користувачам читати уривки з книг, купувати або орендувати електронні версії, а також шукати конкретну інформацію в текстах книг.

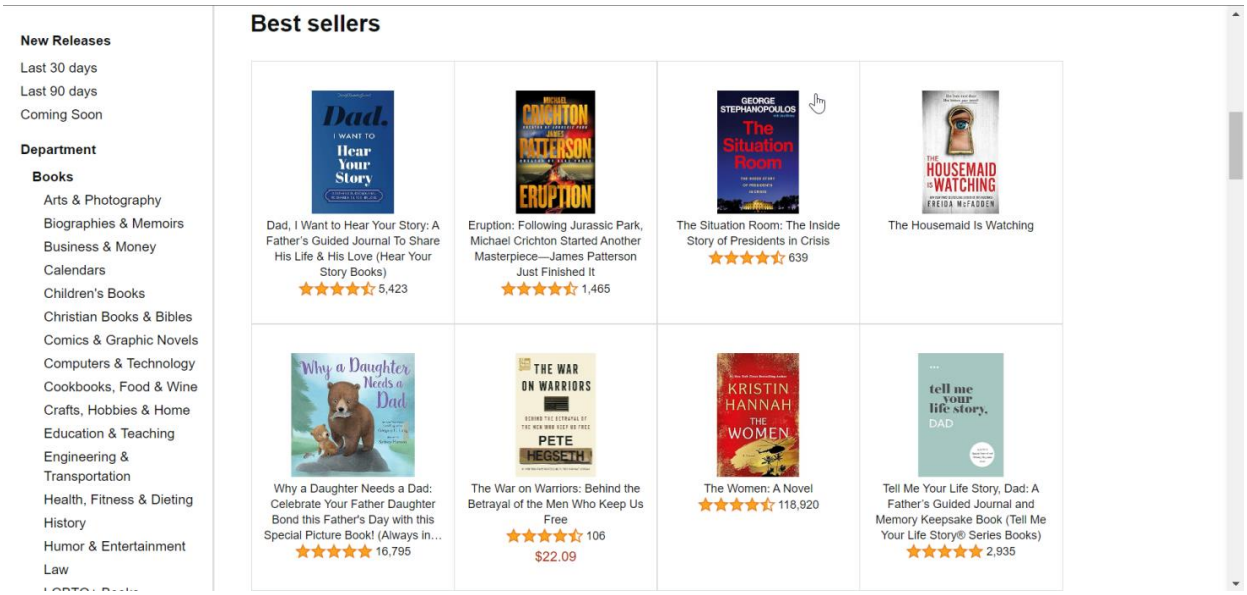

Рисунок 1.1 – Amazon [2]

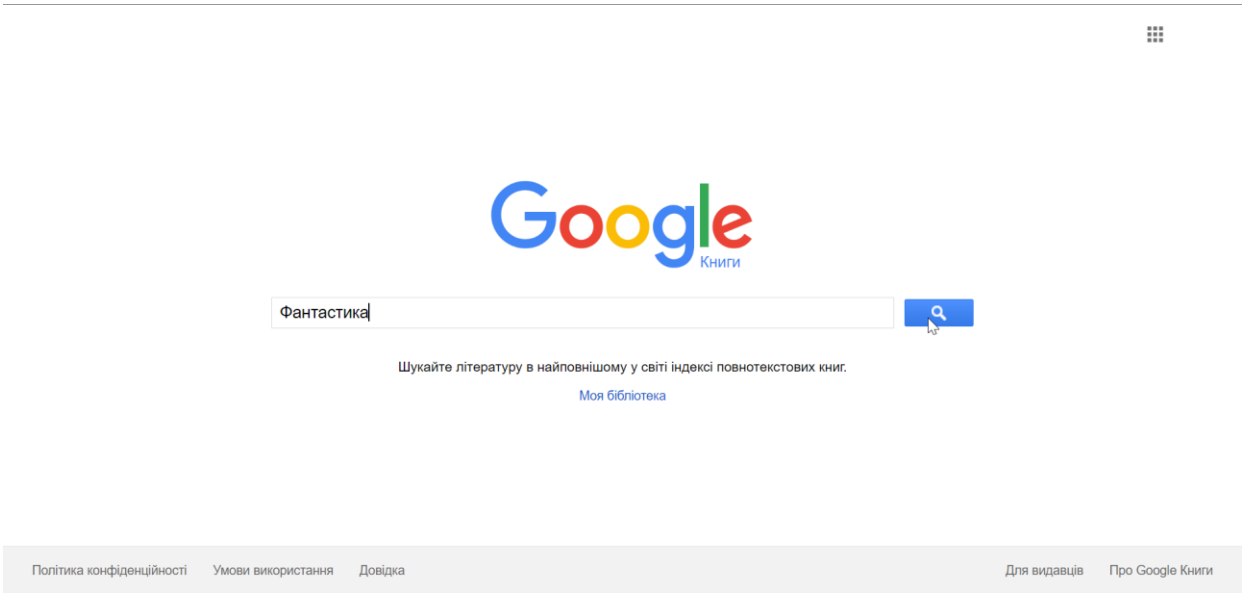

Рисунок 1.2 – Google Books [3]

Barnes & Noble (рис. 1.3) є великим американським ритейлером книг, який також має ефективний онлайн-сервіс. Він пропонує широкий вибір книг, журнали, комікси, а також електронні книги та рідери.

Місцеві онлайн-книгарні часто пропонують унікальний асортимент книг, зосереджений на регіональній літературі та інтересах місцевих читачів. Вони можуть включати незалежні книжкові магазини, які розширили свою діяльність в онлайн-простір. Переваги таких сервісів включають підтримку місцевої економіки, персоналізований підхід до клієнтів, а також можливість швидкого отримання замовлень завдяки місцевій доставці. Недоліки можуть включати обмежений асортимент літератури, відсутність багатьох популярних видань та менше знижок і акцій.

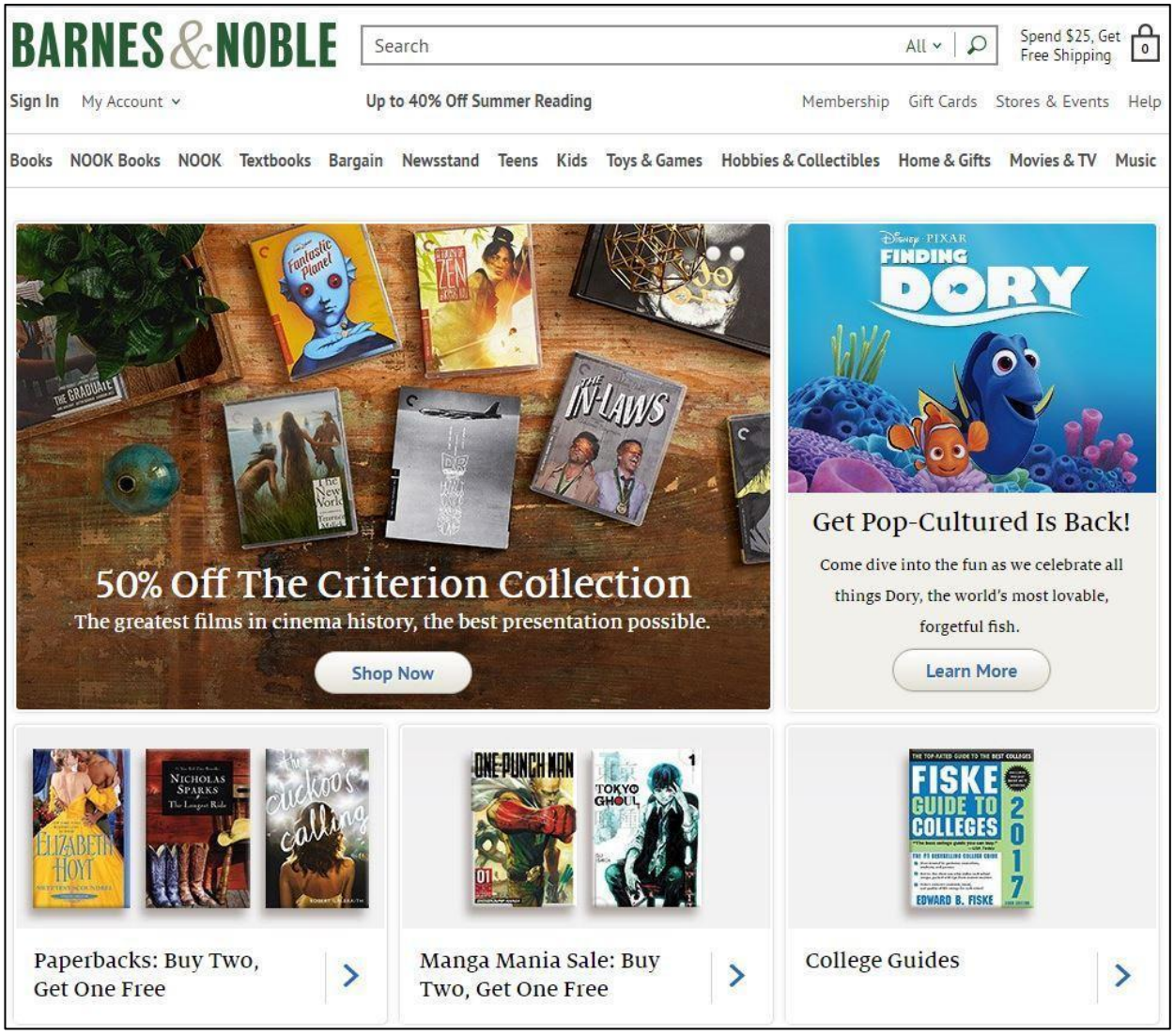

Рисунок 1.3 – Barnes & Noble [4]

Кожен з цих онлайн-сервісів має свої унікальні характеристики, які роблять його привабливим для різних груп користувачів. Вибір сервісу

залежить від індивідуальних потреб, зручності користування, цінової політики та доступності доставки. Розглянемо детальніше Amazon, Barnes & Noble, Google Books та місцеві онлайн-книгарні в таблиці 1.1.

Таблиця 1.1 – Аналіз аналогічних рішень

Рішення	*Переваги*	*Недоліки*
Amazon	Широкий вибір літератури	Високі ціни на деякі книги
	Швидка доставка	
	Наявність знижок та акцій	*Складнощі з доставкою в деякі регіони*
	Зручний інтерфейс	
Barnes & Noble	Широкий асортимент літератури	Обмежена доступність доставки за межі США
	Спеціальні пропозиції для членів клубу	Вищі ціни на деякі видання порівняно з конкурентами
	Можливість замовлення рідкісних та спеціалізованих видань	
Google Books	Зручний доступ до великої кількості літератури	Обмеження доступу до деяких книг через авторські права
	Інтеграція з іншими сервісами Google	Недостатньо розвинута система рекомендацій
	Можливість перегляду та пошуку в тексті	
Місцеві онлайн-книгарні	Підтримка місцевої економіки	Обмежений асортимент літератури
	Персоналізований підхід до клієнтів	Відсутність багатьох популярних видань
	Можливість швидкого отримання замовлень завдяки місцевій доставці	

1.3 Тенденції розвитку ринку онлайн-замовлення літератури

Ринок онлайн-замовлення літератури активно розвивається, відображаючи зміни в споживацьких уподобаннях та технологічні інновації. Розглянемо, що включають основні тенденції.

Зростання популярності електронних та аудіокниг. Все більше користувачів віддають перевагу електронним книгам та аудіокнигам завдяки їх зручності та доступності. Це дозволяє легко зберігати велику кількість книг на одному пристрої та слухати літературу під час виконання інших завдань.

Інтеграція з мобільними додатками. З розвитком смартфонів та планшетів онлайн-сервіси активно розробляють мобільні додатки, що забезпечують зручний доступ до каталогів, замовлення та читання книг у будь-якому місці та в будь-який час. Мобільні додатки дозволяють користувачам отримувати персоналізовані рекомендації, зберігати закладки, нотатки та синхронізувати читання між різними пристроями.

Розвиток рекомендаційних систем. Онлайн-сервіси активно використовують алгоритми машинного навчання та штучного інтелекту для аналізу поведінки користувачів та надання персоналізованих рекомендацій. Це дозволяє підвищити рівень задоволеності користувачів та збільшити обсяги продажів.

Соціальна інтеграція. Все більше платформ інтегрують соціальні функції, дозволяючи користувачам ділитися своїми відгуками, рецензіями та рекомендаціями з друзями та іншими користувачами. Це сприяє створенню спільнот навколо літератури та підвищенню залученості користувачів.

Вдосконалення логістики та доставки. Онлайн-сервіси активно працюють над оптимізацією процесів доставки, скорочуючи час очікування та знижуючи вартість доставки для кінцевих споживачів. Впровадження автоматизованих складів та співпраця з місцевими кур’єрськими службами дозволяють забезпечити швидку та надійну доставку замовлень.

Зростання уваги до безпеки та конфіденційності. У зв’язку зі

збільшенням обсягів онлайн-замовлень, питання захисту персональних даних користувачів стає все більш актуальним. Онлайн-сервіси впроваджують нові заходи безпеки, включаючи шифрування даних, багатофакторну автентифікацію та регулярні перевірки на вразливості.

Підтримка локальних та незалежних видавців. Багато онлайн-сервісів активно співпрацюють з локальними та незалежними видавцями, надаючи їм платформу для поширення своїх книг. Це дозволяє користувачам відкривати нових авторів та підтримувати місцеві видавництва.

Зміни в бізнес-моделях. Деякі сервіси переходять від моделі одноразових покупок до підписних сервісів, що дозволяє користувачам за фіксовану плату отримувати доступ до великої бібліотеки книг. Це робить читання більш доступним та економічно вигідним для споживачів.

Ці тенденції демонструють, як ринок онлайн-замовлення літератури адаптується до змін у технологіях та споживацьких уподобаннях, забезпечуючи користувачам зручний доступ до літератури та підвищуючи рівень їхнього задоволення.

1.4 Постановка завдання

Основною метою дипломної роботи є розробка онлайн-сервісу замовлення літератури, який забезпечить зручний доступ до різноманітних книжкових ресурсів, спростить процес пошуку та замовлення книг, а також забезпечить ефективну роботу системи управління замовленнями і доставки.

Для досягнення цієї мети необхідно виконати ряд завдань.

Аналіз предметної області. Провести детальний аналіз ринку онлайн-замовлення літератури, включаючи огляд існуючих сервісів, їхніх переваг і недоліків, а також тенденцій розвитку даної сфери.

Моделювання предметної області. Використовуючи UML, створити моделі, що відображають основні компоненти та взаємозв’язки системи. Це

включає в себе діаграми випадків використання, діаграми класів, діаграми послідовності тощо.

Проєктування архітектури системи. Розробити архітектуру онлайн-сервісу, яка включатиме як фронтенд (користувацький інтерфейс), так і бекенд (серверна частина, база даних). Вибрати оптимальні технології та інструменти для реалізації даної архітектури.

Розробка структури бази даних. Спроєктувати структуру бази даних, яка забезпечить зберігання та швидкий доступ до інформації про книги, замовлення та користувачів. Забезпечити нормалізацію даних для уникнення дублювання та забезпечення цілісності інформації.

Розробка користувацького інтерфейсу. Створити інтуїтивно зрозумілий та зручний інтерфейс для користувачів сервісу, який дозволить легко знаходити та замовляти книги. Забезпечити адаптивність інтерфейсу для різних пристроїв (комп'ютери, планшети, смартфони).

Реалізація серверної частини. Розробити серверну частину системи, яка забезпечить обробку запитів користувачів, управління замовленнями, автентифікацію та авторизацію користувачів, а також взаємодію з базою даних.

Інтеграція платіжних систем. Здійснити інтеграцію з популярними платіжними системами для забезпечення безпечних та зручних транзакцій користувачів.

Тестування системи. Провести всебічне тестування системи, включаючи модульне тестування, інтеграційне тестування та тестування користувацького інтерфейсу, для виявлення та виправлення помилок.

Оптимізація та відлагодження. Внести необхідні корективи та оптимізації в код для забезпечення високої продуктивності та стабільності роботи сервісу. Врахувати можливі навантаження та забезпечити масштабованість системи.

Документування розробки. Підготувати детальну документацію з описом архітектури системи, структури бази даних, інструкцій для

користувачів та адміністраторів, а також опису процесу розробки та тестування.

Виконання цих завдань дозволить створити функціональний, надійний та зручний онлайн-сервіс замовлення літератури, який відповідатиме сучасним вимогам ринку та задовольнить потреби користувачів у швидкому та зручному доступі до книжкових ресурсів.

2 ПРОЄКТУВАННЯ ОНЛАЙН-СЕРВІСУ ЗАМОВЛЕННЯ ЛІТЕРАТУРИ

2.1 Моделювання предметної області засобами UML

Моделювання предметної області є важливим етапом розробки програмного забезпечення, оскільки воно дозволяє створити формалізоване уявлення про структуру та поведінку системи. Для моделювання предметної області онлайн-сервісу замовлення літератури використано засоби UML.

UML – це уніфікована мова моделювання, яка використовується для візуального представлення, проєктування та документування систем програмного забезпечення. UML надає стандартизовані нотації для створення різноманітних діаграм, які допомагають розробникам, архітекторам та іншим учасникам проєкту зрозуміти структуру та поведінку системи [5].

Основна мета UML полягає у забезпеченні спільної мови для всіх учасників процесу розробки програмного забезпечення, що дозволяє їм ефективно обговорювати і документувати технічні аспекти проєкту. Використання UML допомагає виявляти потенційні проблеми на ранніх етапах розробки, а також полегшує процеси аналізу, проєктування, реалізації та тестування програмних систем. UML дозволяє створювати різні типи діаграм, такі як діаграми класів, послідовності, випадків використання, діяльності та розгортання, що покривають різні аспекти системи і її взаємодії з користувачами та іншими системами [6].

Діаграма класів (рис. 2.1) відображає основні класи системи, їх атрибути, методи та зв’язки між класами.

В нашій системі визначено такі основні класи:

– User: представляє користувача системи з атрибутами id, email, password;
– Product: представляє книгу, яку можна замовити, з атрибутами id,

name, description, price, stock, image_url;

– Order: представляє замовлення з атрибутами id, user_id, total_amount;
– OrderItem: представляє елемент замовлення, що містить інформацію про конкретний продукт у замовленні, з атрибутами order_id, product_id, quantity, price.

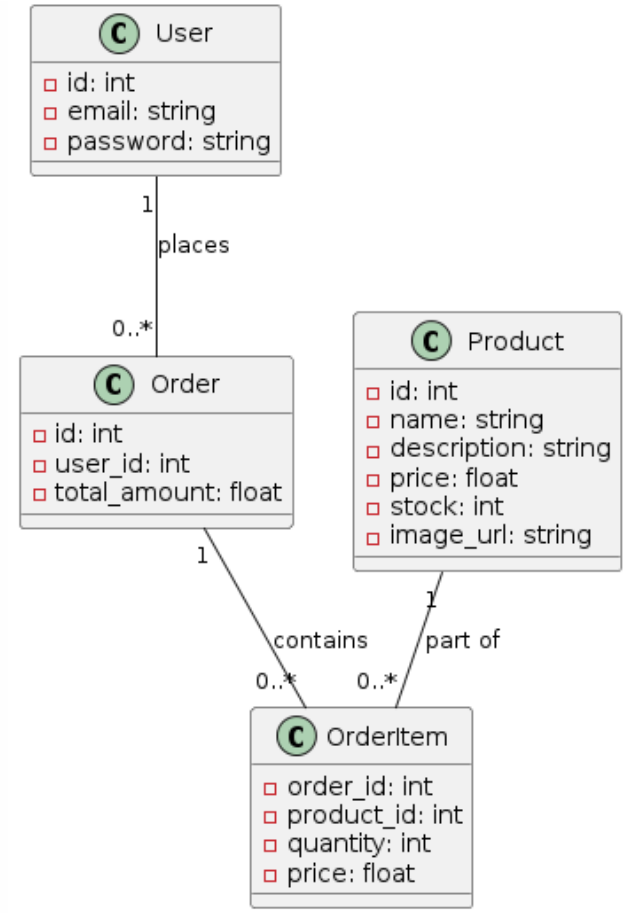

Рисунок 2.1 – Діаграма класів

Зв’язки між класами:

– User має зв’язок один до багатьох з Order (один користувач може мати багато замовлень);
– Order має зв’язок один до багатьох з OrderItem (одне замовлення може містити багато елементів замовлення);
– Product має зв’язок багато до багатьох з OrderItem (один продукт може бути у багатьох елементах замовлення і один елемент замовлення може містити багато продуктів).

Діаграма послідовності (рис. 2.2) показує взаємодію між об’єктами системи у часі для виконання певної функціональності.

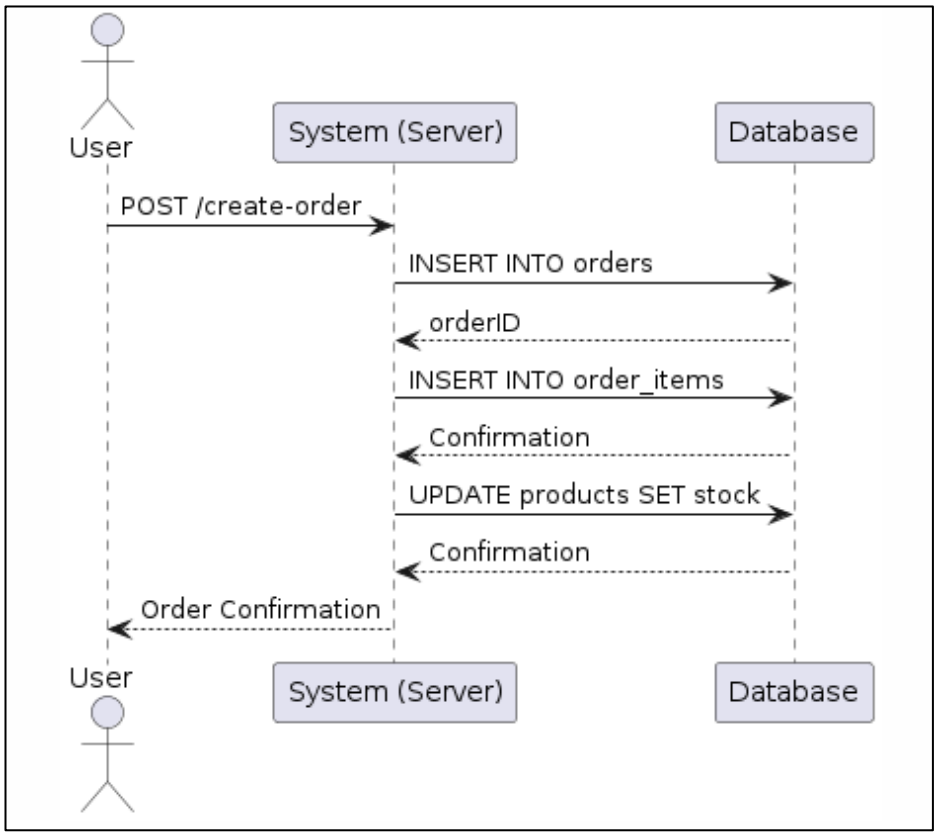

Рисунок 2.2 – Діаграма послідовності

Для нашої системи розглянемо діаграму послідовності для процесу створення замовлення:

– користувач заповнює кошик товарами та ініціює процес створення замовлення;
– система отримує запит на створення замовлення;
– система створює запис у таблиці Order з інформацією про користувача та загальну суму замовлення;
– система створює записи у таблиці OrderItem для кожного продукту у замовленні;
– система оновлює кількість товарів на складі у таблиці Product;
– система повертає підтвердження успішного створення замовлення користувачеві.

Діаграма компонентів (рис. 2.3) відображає високорівневу архітектуру системи, включаючи основні компоненти і їх взаємодію.

Для нашого онлайн-сервісу замовлення літератури визначено такі основні компоненти:

– клієнтський додаток (Front-end): відповідає за взаємодію з користувачем – реалізований за допомогою React;
– серверна частина (Back-end): обробляє запити від клієнтського додатку, взаємодіє з базою даних – реалізований за допомогою Node.js та Express;
– база даних (Database): зберігає інформацію про користувачів, продукти, замовлення та елементи замовлень – використовується MySQL.

Взаємодія між компонентами:

– клієнтський додаток відправляє HTTP-запити до серверної частини для отримання або відправки даних;
– серверна частина обробляє запити, виконує необхідні операції над даними у базі даних та повертає відповіді клієнтському додатку.

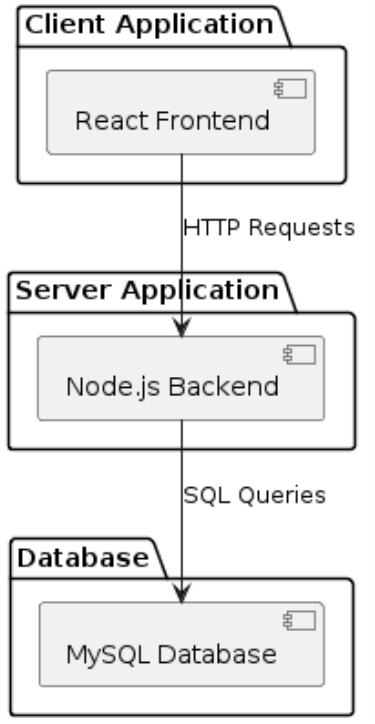

Рисунок 2.3 – Діаграма компонентів

Діаграма розгортання (рис. 2.4) показує фізичну архітектуру системи, включаючи сервери, мережеві з’єднання та розміщення компонентів.

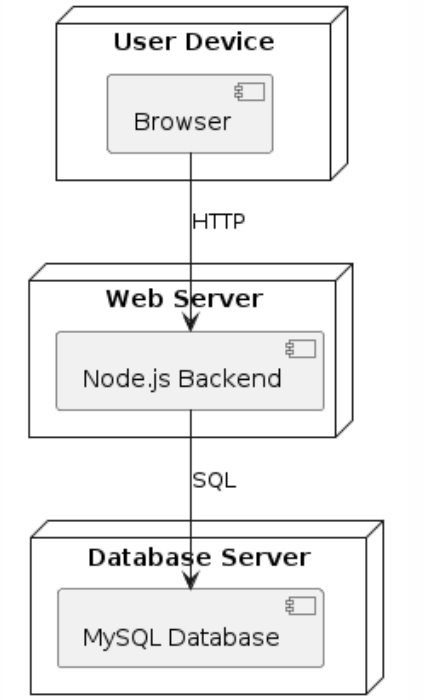

Рисунок 2.4 – Діаграма розгортання

Для нашої системи визначено такі основні вузли:

– користувацький пристрій (User Device): відображає клієнтський додаток, що працює у браузері користувача;
– вебсервер (Web Server): розміщує серверну частину системи;
– сервер бази даних (Database Server): розміщує базу даних MySQL.

Взаємодія між вузлами:

– користувацький пристрій з’єднується з вебсервером через інтернет;
– вебсервер з’єднується з сервером бази даних через локальну мережу або захищений інтернет-з’єднання.

Діаграма випадків використання (рис. 2.5) показує функціональність системи з точки зору користувача.

Основні актори і випадки використання для нашої системи:

– користувач: може реєструватися, входити в систему, переглядати продукти, додавати продукти до кошика, оформляти замовлення,

переглядати свої замовлення;

– адміністратор: може додавати, редагувати та видаляти продукти, переглядати всі замовлення, керувати користувачами.

Рисунок 2.5 – Діаграма випадків використання

Основні випадки використання:

– реєстрація користувача;

– вхід в систему;

– перегляд продуктів;
– додавання продуктів до кошика;
– оформлення замовлення;
– перегляд замовлень користувача;
– додавання продуктів (Адміністратор);
– редагування продуктів (Адміністратор);
– видалення продуктів (Адміністратор);
– перегляд замовлень (Адміністратор);
– керування користувачами (Адміністратор).

Ці UML-діаграми допомагають зрозуміти структуру та взаємодію компонентів системи, що є основою для подальшої розробки і впровадження онлайн-сервісу замовлення літератури.

2.2 Архітектура та структура системи

Архітектура та структура онлайн-сервісу замовлення літератури базується на принципах розподіленої системи з клієнт-серверною архітектурою. Основні компоненти системи включають клієнтське додаток, серверну частину та базу даних.

Архітектура системи складається з трьох основних шарів (табл. 2.1).

Таблиця 2.1 – Архітектура системи

Шар	**Опис**
Презентаційний шар (Front-end)	Реалізований за допомогою React, цей шар відповідає за взаємодію з користувачем.
	Користувачі можуть переглядати доступні книги, додавати їх до кошика, оформляти замовлення, а також переглядати свої замовлення.

Продовження таблиці 2.1

Шар	Опис
Логічний шар (Back-end)	Реалізований за допомогою Node.js та Express, цей шар обробляє бізнес-логіку та взаємодіє з базою даних.
	Обробляє запити від клієнтського додатку, виконує CRUD-операції (створення, читання, оновлення, видалення) з даними про користувачів, продукти та замовлення.
Шар даних (Database)	Використовується MySQL для зберігання даних про користувачів, продукти, замовлення та елементи замовлень.
	Забезпечує надійне зберігання даних та швидкий доступ до них для серверної частини.

Структура системи відображає взаємодію між компонентами та їх розміщення в межах архітектури (табл. 2.2).

Таблиця 2.2 – Структура системи

Компонент	Опис
Клієнтський додаток (React)	Відображає вебінтерфейс для користувачів.
	Використовує HTTP-запити для взаємодії з серверною частиною.
Серверна частина (Node.js + Express)	Обробляє HTTP-запити від клієнтського додатку.
	Виконує аутентифікацію користувачів за допомогою JWT (JSON Web Token).
	Виконує операції з базою даних за допомогою SQL-запитів.
База даних (MySQL)	Зберігає інформацію про користувачів у таблиці users.

Продовження таблиці 2.2

Компонент	Опис
База даних (MySQL)	Зберігає інформацію про продукти у таблиці products.
	Зберігає інформацію про замовлення у таблиці orders.
	Зберігає інформацію про елементи замовлень у таблиці order_items.

Взаємодія між компонентами.

Клієнтський додаток:

– відправляє HTTP-запити до серверної частини для отримання списку продуктів, додавання продуктів до кошика, оформлення замовлень та інше;
– отримує відповіді від серверної частини та відображає їх користувачам.

Серверна частина:

– отримує запити від клієнтського додатку;
– обробляє запити, взаємодіє з базою даних та повертає відповіді клієнтському додатку.

База даних:

– приймає SQL-запити від серверної частини для виконання операцій з даними;
– повертає результати виконання запитів до серверної частини.

Ця архітектура забезпечує модульність, масштабованість та зручність підтримки системи, дозволяючи легко оновлювати окремі компоненти без впливу на інші частини системи.

2.3 Проєктування структури бази даних

Проєктування структури бази даних є важливим етапом розробки системи, оскільки від цього залежить ефективність зберігання та обробки даних. Для нашого онлайн-сервісу замовлення літератури структура бази даних розроблена таким чином, щоб забезпечити оптимальну організацію даних і швидкий доступ до них.

Таблиця users зберігає інформацію про користувачів системи:

– id (INT) – унікальний ідентифікатор користувача, первинний ключ (PRIMARY KEY);
– email (VARCHAR(100)) – електронна адреса користувача;
– password (VARCHAR(255)) – хешований пароль користувача;
– phone (VARCHAR(20)) – номер телефону користувача.

Таблиця products зерігає інформацію про книги, які можна замовити.

– id (INT) – унікальний ідентифікатор продукту, первинний ключ (PRIMARY KEY);
– name (VARCHAR(100)) – назва книги;
– description (TEXT) – опис книги;
– price (DECIMAL(10,2)) – ціна книги;
– stock (INT) – кількість книг на складі;
– image_url (VARCHAR(255)) – URL зображення книги.

Таблиця orders зберігає інформацію про замовлення користувачів:

– id (INT) – унікальний ідентифікатор замовлення, первинний ключ (PRIMARY KEY);
– user_id (INT) – ідентифікатор користувача, який зробив замовлення, зовнішній ключ (FOREIGN KEY), що посилається на id в таблиці users;
– order_date (DATETIME) – дата та час замовлення;
– total_amount (DECIMAL(10,2)) – загальна сума замовлення.

Таблиця order_items зберігає інформацію про конкретні книги у замовленні:

- id (INT) – унікальний ідентифікатор елемента замовлення, первинний ключ (PRIMARY KEY);
- order_id (INT) – ідентифікатор замовлення, зовнішній ключ (FOREIGN KEY), що посилається на id в таблиці orders;
- product_id (INT) – ідентифікатор продукту, зовнішній ключ (FOREIGN KEY), що посилається на id в таблиці products;
- quantity (INT) – кількість продукту у замовленні;
- price (DECIMAL(10,2)) – ціна продукту на момент замовлення.

Наведемо приклади SQL-запитів для створення цих таблиць у базі даних MySQL (рис. 2.6).

```
CREATE TABLE users (
   id INT AUTO_INCREMENT PRIMARY KEY,
   email VARCHAR(100) NOT NULL,
   password VARCHAR(255) NOT NULL,
   phone VARCHAR(20)
);
CREATE TABLE products (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   description TEXT,
   price DECIMAL(10,2) NOT NULL,
   stock INT NOT NULL,
   image_url VARCHAR(255)
);
CREATE TABLE orders (
   id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT,
   order_date DATETIME NOT NULL,
   total_amount DECIMAL(10,2) NOT NULL,
   FOREIGN KEY (user_id) REFERENCES users(id)
);
CREATE TABLE order_items (
   id INT AUTO_INCREMENT PRIMARY KEY,
   order_id INT,
   product_id INT,
   quantity INT NOT NULL,
   price DECIMAL(10,2) NOT NULL,
   FOREIGN KEY (order_id) REFERENCES orders(id),
   FOREIGN KEY (product_id) REFERENCES products(id)
);
```

Рисунок 2.6 – Створення таблиць БД

Таблиця users:

– поле id є первинним ключем (PRIMARY KEY), яке автоматично збільшується (AUTO_INCREMENT);
– поле email є обов’язковим (NOT NULL) та має унікальні значення;
– поле password є обов’язковим (NOT NULL);
– поле phone є необов’язковим.

Таблиця products:

– поле id є первинним ключем (PRIMARY KEY), яке автоматично збільшується (AUTO_INCREMENT);
– поле name є обов’язковим (NOT NULL);
– поле description зберігає текстову інформацію про книгу;
– поле price є обов’язковим (NOT NULL);
– поле stock є обов’язковим (NOT NULL) та вказує на кількість книг на складі;
– поле image_url зберігає URL зображення книги.

Таблиця orders:

– поле id є первинним ключем (PRIMARY KEY), яке автоматично збільшується (AUTO_INCREMENT);
– поле user_id є зовнішнім ключем (FOREIGN KEY), що посилається на поле id в таблиці users;
– поле order_date є обов’язковим (NOT NULL) та зберігає дату та час створення замовлення;
– поле total_amount є обов’язковим (NOT NULL).

Таблиця order_items:

– поле id є первинним ключем (PRIMARY KEY), яке автоматично збільшується (AUTO_INCREMENT);
– поле order_id є зовнішнім ключем (FOREIGN KEY), що посилається на поле id в таблиці orders;
– поле product_id є зовнішнім ключем (FOREIGN KEY), що посилається на поле id в таблиці products;

– поле quantity є обов’язковим (NOT NULL) та вказує на кількість продукту в замовленні;
– поле price є обов’язковим (NOT NULL) та вказує на ціну продукту на момент замовлення.

```
INSERT INTO users (email, password, phone) VALUES
('user1@example.com', 'password1', '1234567890'),
('user2@example.com', 'password2', '0987654321');

INSERT INTO products (name, description, price, stock, image_url) VALUES
('Убити пересмішника', 'Роман Гарпер Лі, опублікований у 1960 році.', 10.99, 20, 'http://example.com/tokillamockingbird.jpg'),
('1984', 'Дистопічний роман Джорджа Орвелла.', 8.99, 15, 'http://example.com/1984.jpg'),
('Гордість і упередження', 'Роман Джейн Остін.', 12.99, 10, 'http://example.com/prideandprejudice.jpg'),
('Великий Гетсбі', 'Роман Ф. Скотта Фіцджеральда.', 14.50, 7, 'http://example.com/thegreatgatsby.jpg'),
('Мобі Дік', 'Роман Германа Мелвілла.', 9.99, 12, 'http://example.com/mobydick.jpg');

INSERT INTO orders (user_id, order_date, total_amount) VALUES
(1, '2023-06-11 12:00:00', 39.47),
(2, '2023-06-12 14:30:00', 23.48);

INSERT INTO order_items (order_id, product_id, quantity, price) VALUES
(1, 1, 2, 10.99),
(1, 2, 1, 8.99),
(1, 3, 1, 12.99),
(2, 4, 1, 14.50),
(2, 5, 1, 9.99);
```

Рисунок 2.7 – Приклад заповнення таблиць тестовими даними

Також створимо дігараму структури бази даних, яка відображає основні таблиці та зв’язки між ними (рис. 2.8).

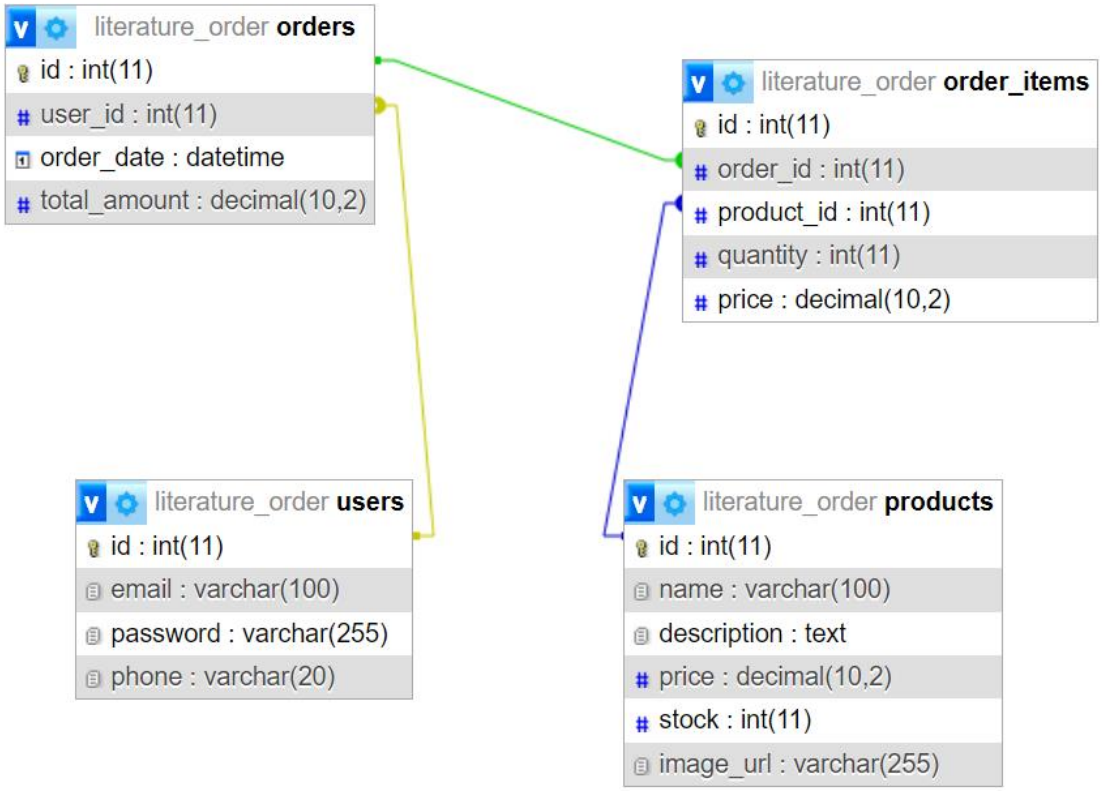

Рисунок 2.8 – Діаграма бази даних

Ця структура бази даних забезпечує оптимальне зберігання даних про користувачів, продукти та замовлення. Завдяки чітким зв’язкам між таблицями, система може легко виконувати складні запити та забезпечувати швидкий доступ до даних.

2.4 Вибір технологій та інструментів розробки

Розробка онлайн-сервісу замовлення літератури вимагає використання сучасних технологій та інструментів, які забезпечать надійність, масштабованість та зручність в обслуговуванні системи. Для реалізації презентаційного шару обрано фреймворк React. Він дозволяє створювати динамічні та інтерактивні інтерфейси користувача з використанням компонентного підходу, що полегшує розробку та підтримку коду. React також має велику спільноту розробників та безліч готових бібліотек, що пришвидшує процес розробки [7].

Для серверної частини вибір пав на Node.js з використанням фреймворку Express. Node.js дозволяє використовувати одну мову програмування – javasctipt – для розробки як клієнтської, так і серверної частини, що спрощує навчання та зменшує кількість необхідних технологій. Express, у свою чергу, є легким та гнучким фреймворком для розробки вебдодатків та API, що робить його ідеальним вибором для нашої системи [8].

Для зберігання даних обрано реляційну базу даних MySQL. Вона забезпечує надійне зберігання даних, підтримує складні запити та транзакції, а також має добре задокументовані бібліотеки для інтеграції з Node.js. MySQL також є відкритим програмним забезпеченням, що знижує витрати на розробку та експлуатацію системи [9].

Для забезпечення безпеки використано JWT для аутентифікації користувачів. JWT дозволяє створювати захищені токени, які можуть бути використані для перевірки автентичності запитів без необхідності зберігати сесії на сервері. Це підвищує масштабованість системи та спрощує управління сесіями.

Для полегшення роботи з кросс-доменною політикою використано бібліотеку CORS. CORS дозволяє налаштовувати політику доступу до ресурсів сервера з різних доменів, що забезпечує безпечну взаємодію клієнтського додатку з сервером.

Для обробки HTTP-запитів та роботи з JSON-даними використано бібліотеку Axios. Вона забезпечує простий та ефективний спосіб виконання запитів з клієнтського додатку до серверної частини, а також підтримує обробку відповідей і помилок.

Для управління станом клієнтського додатку обрано бібліотеку Redux. Вона дозволяє централізовано зберігати стан додатку, що полегшує управління даними та забезпечує їх узгодженість між різними компонентами.

Для розробки та тестування інтерфейсу користувача використано бібліотеку Material-UI. Вона забезпечує набір готових компонентів, що відповідають принципам матеріального дизайну, що дозволяє швидко

створювати сучасні та привабливі інтерфейси.

Для тестування системи використано Jest та Mocha. Jest забезпечує зручне тестування компонентів React, а Mocha дозволяє проводити модульні тести серверної частини, що гарантує стабільність та надійність коду.

Для управління версіями коду використано систему контролю версій Git, а для співпраці в команді та зберігання репозиторіїв обрано платформу GitHub. Це забезпечує ефективне відстеження змін у коді, спільну роботу над проєктом та зберігання резервних копій коду.

Як середовище розробки обрано Visual Studio Code [10] (рис. 2.9).

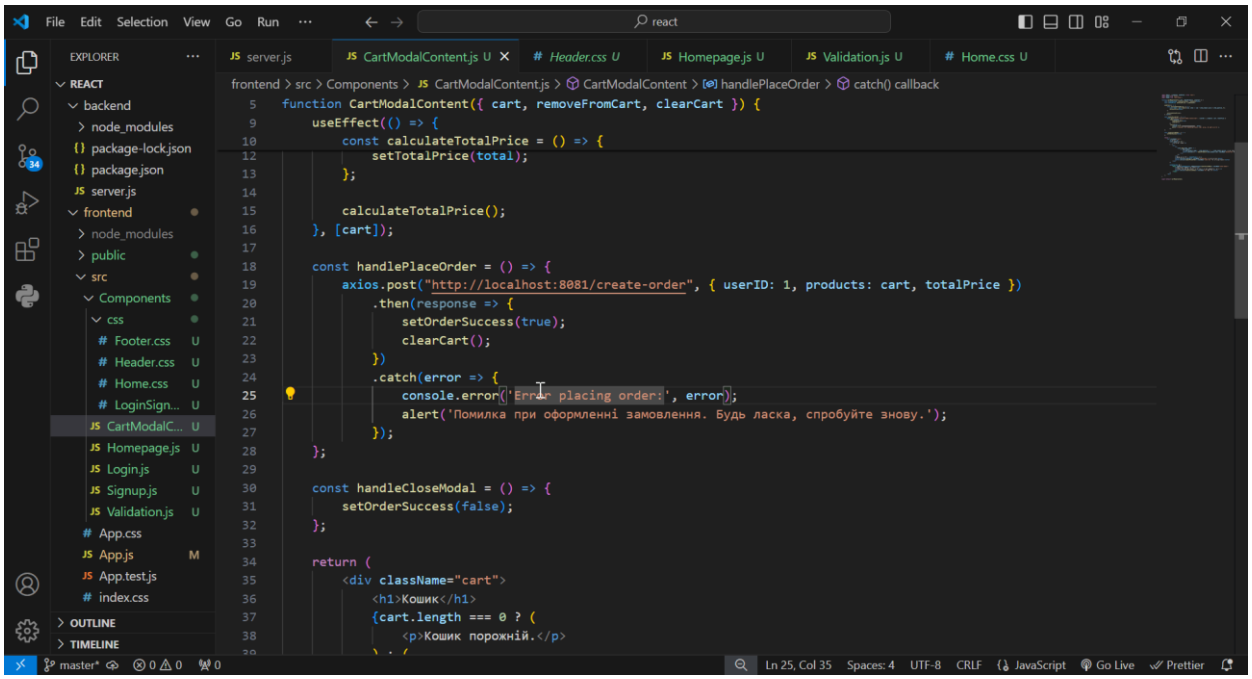

Рисунок 2.9 – Розробки у Visual Studio Code

Загалом, вибір технологій та інструментів розробки базувався на їх сумісності, популярності, активній спільноті та можливостях забезпечити ефективну розробку, тестування та підтримку онлайн-сервісу замовлення літератури.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ОНЛАЙН-СЕРВІСУ

3.1 Розробка інтерфейсу користувача

Розробка інтерфейсу користувача є основним етапом створення онлайн-сервісу замовлення літератури. Вона включає проєктування, розробку та тестування всіх елементів, з якими взаємодіє користувач. Основною метою є забезпечення зручності, інтуїтивності та ефективності користування сервісом.

Вимоги до інтерфейсу користувача:

– інтерфейс повинен бути інтуїтивно зрозумілим для користувача. Всі основні функції, такі як реєстрація, вхід, перегляд продуктів, додавання до кошика та оформлення замовлення, мають бути легко доступними;
– навігація повинна бути простою та логічною, з чітко позначеними кнопками та посиланнями;
– інтерфейс повинен бути адаптованим для різних пристроїв, включаючи десктопи, планшети та смартфони – це забезпечить зручність використання сервісу на будь-якому пристрої;
– вебсторінки повинні завантажуватися швидко, щоб користувачі не відчували затримок під час користування сервісом;
– інтерфейс повинен мати сучасний та привабливий дизайн, використання кольорів, шрифтів та графічних елементів повинно бути гармонійним та відповідати загальному стилю сервісу;
– інтерфейс повинен забезпечувати безпечну аутентифікацію та передачу даних, всі форми, які збирають конфіденційну інформацію, повинні бути захищені.

Процес розробки інтерфейсу користувача, включаючи всі етапи від початкового дизайну до тестування та вдосконалення, опишемо детально в таблиці 3.1, де будуть наведені основні кроки, методи та інструменти, що використовувалися на кожному з етапів цього процесу.

Таблиця 3.1 – Процес розробки інтерфейсу

Етап	**Опис**
Проєктування макету	Проєктування інтерфейсу розпочинається з створення макетів основних сторінок та екранів. Це включає головну сторінку, сторінку каталогу продуктів, сторінку кошика, сторінку оформлення замовлення, сторінки реєстрації та входу.
	Використовуються інструменти для прототипування, такі як Figma або Sketch, щоб створити візуальні макети та визначити розташування основних елементів інтерфейсу.
Розробка компонентів	Інтерфейс реалізується за допомогою фреймворку React, що дозволяє створювати динамічні та інтерактивні компоненти. Кожен компонент відповідає за окремий елемент інтерфейсу, такий як кнопка, форма, таблиця продуктів тощо.
	Використовуються компоненти бібліотеки Material-UI для забезпечення сучасного вигляду та узгодженості стилів.
Створення навігації	Реалізується зручна навігація між сторінками за допомогою React Router. Це дозволяє користувачам легко переходити між різними розділами сервісу.
Інтеграція з бекендом	Інтерфейс взаємодіє з серверною частиною через API. Використовується бібліотека Axios для виконання HTTP-запитів до бекенду та отримання необхідних даних.
	Реалізується функціональність для відправки форм реєстрації та входу, отримання списку продуктів, додавання продуктів до кошика та оформлення замовлень.
Тестування інтерфейсу	Проводиться ретельне тестування інтерфейсу на різних пристроях та браузерах для забезпечення його коректної роботи.

Продовження таблиці 3.1

Етап	Опис
Тестування інтерфейсу	Використовуються інструменти для автоматизованого тестування компонентів, такі як Jest та React Testing Library.
Оптимізація та покращення	На основі зворотного зв’язку від користувачів та результатів тестування проводиться оптимізація інтерфейсу. Це може включати покращення швидкості завантаження, виправлення помилок, покращення зручності використання.

Код файлів Login.js і Signup.js реалізує компоненти для реєстрації та входу користувачів в онлайн-сервіс замовлення літератури, використовуючи бібліотеки React, Axios, та React Router. Компоненти Login та Signup забезпечують інтерфейс для введення облікових даних користувачами, валідацію цих даних, взаємодію з серверною частиною для автентифікації та реєстрації користувачів, а також обробку помилок.

Компонент Login (рис. 3.1) відповідає за вхід користувачів. Використовуючи React, стан компонента керується за допомогою хука useState. Стан містить об’єкт values, який зберігає введені користувачем дані електронної пошти та пароля. Крім того, стан включає об’єкти errors, isModalOpen, loginError, та auth, які використовуються для управління помилками, відображення модального вікна з повідомленням про помилки, зберігання повідомлень про помилки входу та зберігання стану автентифікації користувача відповідно.

Для надсилання HTTP-запитів до сервера використовується бібліотека Axios. Axios налаштовується для автоматичного включення кукі у запити за допомогою axios.defaults.withCredentials = true. Навігація після успішного входу користувача реалізується за допомогою хука useNavigate з React Router.

Функція handleInput оновлює стан values при введенні користувачем

даних у відповідні поля форми. Функція handleSubmit виконується при відправці форми входу. Вона спочатку виконує валідацію введених даних, використовуючи функцію Validation. Якщо дані відповідають вимогам, надсилається POST-запит до сервера для автентифікації користувача. У разі успіху стан автентифікації оновлюється, і користувач перенаправляється на головну сторінку. У разі помилки відображається повідомлення про помилку у модальному вікні.

```
return (
   <div className="form-container">
     <div className="form-header">
       <div className="text">Логін</div>
       <div className="underline"></div>
     </div>
     {auth.isAuthenticated && (
       <div>Вітаємо, користувач з ID: {auth.userID}</div>
     )}

     {loginError && (
       <Modal isOpen={isModalOpen} onRequestClose={closeModal} className="login-
modal">
         <div>{loginError}</div>
         <button onClick={closeModal}>Close</button>
       </Modal>
     )}

     <form action="" onSubmit={handleSubmit}>
       <div className="inputs">
         <input type="email" name="email" placeholder="Емайл" onChange={handleInput}
required />
         <input type="password" name="password" placeholder="Пароль"
onChange={handleInput} required />
         {errors.password && <span className="error">{errors.password}</span>}
       </div>

       <div className="submit-container">
         <Link to="/signup" className="submit gray">Реєстрація</Link>
         <button type="submit" className="submit">Логін</button>
       </div>
     </form>
   </div>
 );
```

Рисунок 3.1 – Фрагмент Login

Компонент Signup (рис. 3.2) відповідає за реєстрацію нових користувачів. Стан цього компонента також керується за допомогою хука useState та містить об’єкти values та errors. Функція handleInput оновлює стан values при введенні користувачем даних у відповідні поля форми. Функція handleSubmit виконується при відправці форми реєстрації. Вона спочатку виконує валідацію введених даних, використовуючи функцію Validation. Якщо дані відповідають вимогам, надсилається POST-запит до сервера для створення нового користувача. У разі успіху користувач перенаправляється на сторінку входу.

```
function Signup() {

  const [values,setValues] = useState({
    email: '',
    password: '',
  })

  const nav = useNavigate();
  const [errors, setErrors] = useState({});

  const handleInput = (event) => {
    setValues(prev => ({...prev, [event.target.name]: [event.target.value]}))
  }

  const handleSubmit = (event) => {
    event.preventDefault();
    setErrors(Validation(values));
    if(errors.password === "") {
      axios.post("http://localhost:8081/signup", values)
      .then(res => {
        nav('/');
      })
      .catch(err => console.log(err));
    }
  }
```

Рисунок 3.2 – Фрагмент Signup

Функція Validation (рис. 3.3) використовується для перевірки правильності введених даних. Вона приймає об’єкт values і повертає об’єкт

error, який містить повідомлення про помилки валідації. У даному випадку, функція перевіряє, чи відповідає пароль шаблону, який вимагає наявності як мінімум 8 символів, включаючи хоча б одну цифру та одну літеру.

```
function Validation(values) {
   let error = {}
   const password_pattern = /^(?=.*[a-zA-Z])(?=.*\d).{8,}$/

   !password_pattern.test(values.password) ? error.password = "Пароль має містити 8 символів,
цифру та літеру" : error.password = ""
   return error;
}

export default Validation;
```

Рисунок 3.3 – Фрагмент Validation

Компонент Modal з бібліотеки React Modal використовується для відображення повідомлень про помилки входу у вигляді модального вікна. Це забезпечує зручний спосіб повідомлення користувачів про проблеми, що виникають під час автентифікації.

CSS-файли LoginSignup.css використовуються для стилізації компонентів Login та Signup. Вони визначають зовнішній вигляд форм, кнопок, полів вводу та інших елементів інтерфейсу, забезпечуючи привабливий та зручний дизайн.

Таким чином, цей код реалізує повноцінні компоненти для реєстрації та входу користувачів, забезпечуючи зручний та безпечний інтерфейс для взаємодії з онлайн-сервісом замовлення літератури. Він охоплює всі необхідні аспекти, включаючи введення та валідацію даних, взаємодію з сервером, обробку помилок та навігацію, що робить його важливою складовою частиною системи.

Код файлів Homepage.js і CartModalContent.js реалізує компоненти для головної сторінки та кошика онлайн-сервісу замовлення літератури, використовуючи бібліотеки React, Axios, та React Modal. Компоненти Homepage та CartModalContent забезпечують інтерфейс для перегляду

продуктів, додавання їх до кошика, оформлення замовлень, а також обробку помилок.

Компонент Homepage (рис. 3.4) відповідає за відображення списку доступних продуктів та забезпечує функціональність пошуку та фільтрації продуктів.

```
function Homepage({ addToCart, cart, removeFromCart, clearCart }) {
  const [products, setProducts] = useState([]);
  const [searchTerm, setSearchTerm] = useState('');
  const [orderModalOpen, setOrderModalOpen] = useState(false);
  const [cartModalOpen, setCartModalOpen] = useState(false);
  const [selectedProduct, setSelectedProduct] = useState(null);
  const [quantity, setQuantity] = useState(1);
  const navigate = useNavigate();

  useEffect(() => {
    axios.get("http://localhost:8081/products")
      .then(res => setProducts(res.data))
      .catch(err => console.log(err));
  }, []);

  const handleSearchChange = (event) => {
    setSearchTerm(event.target.value);
  };

  const filteredProducts = products.filter(product =>
    product.name && product.name.toLowerCase().includes(searchTerm.toLowerCase())
  );

  const handleOrder = (product) => {
    setSelectedProduct(product);
    setOrderModalOpen(true);
  };

  const confirmOrder = () => {
    addToCart(selectedProduct, quantity);
    setOrderModalOpen(false);
    setSelectedProduct(null);
    setQuantity(1);
  };

  const handleLogout = () => {
    axios.post("http://localhost:8081/logout")
      .then(res => navigate('/'))
      .catch(err => console.log(err));
  };
```

Рисунок 3.4 – Фрагмент Homepage

Стан компонента керується за допомогою хуків useState та useEffect. Хук useState використовується для збереження списку продуктів (products), пошукового терміну (searchTerm), стану модального вікна замовлення (orderModalOpen), стану модального вікна кошика (cartModalOpen), вибраного продукту (selectedProduct) та кількості замовлених продуктів (quantity). Хук useEffect виконує завантаження списку продуктів з сервера при першому завантаженні компонента.

Функція handleSearchChange оновлює стан searchTerm при введенні користувачем пошукового запиту. Функція handleOrder відкриває модальне вікно замовлення для обраного продукту. Функція confirmOrder додає обраний продукт до кошика та закриває модальне вікно замовлення. Функція handleLogout виконує вихід користувача, надсилаючи POST-запит до сервера, та перенаправляє користувача на сторінку входу.

Компонент CartModalContent (рис. 3.5) відповідає за відображення вмісту кошика та оформлення замовлення. Стан цього компонента також керується за допомогою хуків useState та useEffect. Хук useState використовується для збереження стану успішного оформлення замовлення (orderSuccess) та загальної вартості продуктів у кошику (totalPrice). Хук useEffect виконує обчислення загальної вартості продуктів у кошику при кожній зміні вмісту кошика.

Функція handlePlaceOrder виконує оформлення замовлення, надсилаючи POST-запит до сервера з даними про користувача, продукти та загальну вартість. У разі успіху стан orderSuccess оновлюється, і кошик очищується. У разі помилки відображається повідомлення про помилку. Функція handleCloseModal закриває модальне вікно успішного оформлення замовлення. Компонент Modal з бібліотеки React Modal використовується для відображення модальних вікон замовлення та успішного оформлення замовлення. Це забезпечує зручний спосіб взаємодії користувачів з системою під час додавання продуктів до кошика та оформлення замовлень.

```
return (
      <div className="cart">
        <h1>Кошик</h1>
        {cart.length === 0 ? (
          <p>Кошик порожній.</p>
        ) : (
          <div>
            <ul>
              {cart.map((item, index) => (
                <li key={index}>
                  <p>{item.product.name} - {item.quantity} шт. - {item.product.price} грн
за одиницю</p>
                  <button onClick={() => removeFromCart(item.product.id)}
className='delete-btn'>Видалити</button>
                </li>
              ))}
            </ul>
            <p>Загальна вартість: {totalPrice} грн</p>
            <button onClick={clearCart} className='delete-btn'>Очистити кошик</button>
            <button onClick={handlePlaceOrder} className='order-btn'>Оформити
замовлення</button>
          </div>
        )}

        {orderSuccess && (
          <Modal isOpen={orderSuccess} onRequestClose={handleCloseModal}
className="order-modal">
            <h2>Замовлення успішно оформлено!</h2>
            <p>Дякуємо за ваше замовлення. Ми зв'яжемося з вами найближчим
часом.</p>
            <button onClick={handleCloseModal} className='btn'>Закрити</button>
          </Modal>
        )}
      </div>
  );
```

Рисунок 3.5 – Фрагмент CartModalContent

CSS-файл Home.css (ліст. 3.6) використовується для стилізації компонентів Homepage та CartModalContent. Він визначає зовнішній вигляд заголовків, кнопок, полів вводу, карток продуктів та інших елементів інтерфейсу, забезпечуючи привабливий та зручний дизайн.

```
.container {
   width: 100%;
   max-width: 1200px;
   padding-left: 1rem;
   padding-right: 1rem;
   margin: 0 auto;
}

.products {
   margin: 2.8rem 0;
}

.product {
   overflow: hidden;
   box-shadow: 0 0 4px fade(#000, 25%);
   transition: box-shadow 0.2s linear;
   height: auto;
}

.product:hover {
   box-shadow: 0 0 25px fade(#000, 25%);
}

.product img {
   position: absolute;
   bottom: 10px;
   right: 10px;
   padding-top: 30px;
}

.count {
   margin-bottom: 15px;
}
```

Рисунок 3.6 – Фрагмент Home.css

Таким чином, цей код реалізує функціональність для перегляду продуктів, пошуку, додавання до кошика та оформлення замовлень, забезпечуючи зручний та інтерактивний інтерфейс для користувачів онлайн-сервісу замовлення літератури. Загалом, розробка інтерфейсу користувача є важливим етапом, що визначає зручність та ефективність використання онлайн-сервісу замовлення літератури.

3.2 Серверна частина

Серверна частина нашого онлайн-сервісу замовлення літератури реалізована за допомогою Node.js та фреймворку Express. Вона відповідає за обробку запитів від клієнтського додатку, взаємодію з базою даних та забезпечення аутентифікації користувачів. Нижче наведено детальний опис коду серверної частини.

Цей код імпортує необхідні модулі (рис. 3.7):

– express: фреймворк для створення вебсерверів;
– mysql: бібліотека для взаємодії з базою даних MySQL;
– cors: бібліотека для налаштування крос-доменної політики доступу;
– jsonwebtoken: бібліотека для роботи з JSON Web Tokens;
– cookieParser: бібліотека для обробки кукі.

```
const express = require("express");
const mysql = require("mysql");
const cors = require("cors");
const jwt = require("jsonwebtoken");
const cookieParser = require("cookie-parser");
```

Рисунок 3.7 – Імпорт необхідних модулів

Цей код створює екземпляр Express-додатку та налаштовує його для використання JSON-парсера та обробника кукі (рис. 3.8).

```
const app = express();
app.use(express.json());
app.use(cookieParser());
```

Рисунок 3.8 – Налаштування Express-додатку

Цей блок коду налаштовує політику CORS, дозволяючи запити з клієнтського додатку, який працює на http://localhost:3000 (рис. 3.9). Встановлюються дозволені методи запитів та передача кукі.

```
const corsOptions = {
   origin: 'http://localhost:3000',
   methods: ["POST", "GET", "PUT", "DELETE"],
   credentials: true,
   optionSuccessStatus: 200
};
app.use(cors(corsOptions));
```

Рисунок 3.9 – Налаштування CORS

Цей код створює з’єднання з базою даних MySQL, використовуючи вказані параметри підключення (рис. 3.10).

```
const db = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "",
   database: "literature_order"
});
```

Рисунок 3.10 – Налаштування підключення до бази даних

Ця функція перевіряє наявність та валідність JWT у кукі користувача (рис. 3.11). Якщо токен відсутній або недійсний, повертається помилка авторизації. Інакше додаються дані користувача до запиту та виконується наступна функція у ланцюжку обробки.

```
const verifyUser = (req, res, next) => {
   const token = req.cookies.token;
   if (!token) {
      return res.json({ error: "You are not authorized" });
   } else {
      jwt.verify(token, "jwt-secret-key", (err, decoded) => {
         if (err) {
            return res.json({ error: "Token is not valid" });
         } else {
            req.id = decoded.id;
            req.table_number = decoded.table_number;
            next();
         }
      });
   }
};
```

Рисунок 3.11 – Функція перевірки автентичності користувача

Цей маршрут обробляє запит на отримання даних для головної сторінки (рис. 3.12). Функція verifyUser перевіряє автентичність користувача перед поверненням даних.

```
app.get('/homepage', verifyUser, (req, res) => {
    return res.json({
        status: "Success",
        id: req.id,
        username: req.email,
    });
});
```

Рисунок 3.12 – Обробка запитів

Цей маршрут обробляє реєстрацію користувача, додаючи новий запис до таблиці users у базі даних (рис. 3.13).

```
app.post('/signup', (req, res) => {
    const sql = "INSERT INTO users (email, password) VALUES (?)";
    const values = [
        req.body.email,
        req.body.password
    ];
    db.query(sql, [values], (err, data) => {
        if (err) {
            return res.json("Error");
        }
        return res.json(data);
    });
});
```

Рисунок 3.13 – Реєстрація користувача

Цей маршрут обробляє вхід користувача (рис. 3.14). Якщо надані правильні облікові дані, створюється JWT та зберігається у кукі. Інакше повертається помилка авторизації.

```
app.post('/login', (req, res) => {
   const { email, password } = req.body;
   const sql = "SELECT  FROM users WHERE email = ? AND password = ?";
   db.query(sql, [email, password], (err, data) => {
      if (err) {
         return res.status(500).json({ error: "Database query error" });
      }
      if (data.length > 0) {
         const userID = data[0].id;
         const username = data[0].email;
         const token = jwt.sign({ userID, username }, "jwt-secret-key", { expiresIn: "1d" });
         res.cookie('token', token, { httpOnly: true });
         return res.json({ status: "Success", userID: userID });
      } else {
         return res.status(404).json({ status: "Login failed", message: "No account found!" });
      }
   });
});
```

Рисунок 3.14 – Вхід користувача

Цей маршрут обробляє вихід користувача, видаляючи JWT з кукі (рис. 3.15).

```
app.post('/logout', (req, res) => {
   res.clearCookie("token");
   return res.json({ status: "Success" });
});
```

Рисунок 3.15 – Вихід з системи

Цей маршрут повертає список всіх продуктів з таблиці products (рис. 3.16).

```
app.get('/products', (req, res) => {
   const sql = "SELECT  FROM products";
   db.query(sql, (err, data) => {
      if (err) {
         return res.status(500).json({ error: "Error retrieving products" });
      }
      return res.json(data);
   });
});
```

Рисунок 3.16 – Витягнення списку продуктів з БД

Цей маршрут обробляє створення нового замовлення (рис. 3.17). Перевіряється правильність даних замовлення, додається новий запис у таблиці orders та order_items, а також оновлюється кількість продуктів на складі.

```
app.post('/create-order', (req, res) => {
   const { userID, products, totalPrice } = req.body;
   if (!userID || !products || !Array.isArray(products) || products.length === 0) {
      return res.status(400).json({ error: "Invalid order data" });
   }

   const orderSql = "INSERT INTO orders (user_id, total_amount) VALUES (?, ?)";
   db.query(orderSql, [userID, totalPrice], (err, result) => {
      if (err) {
         console.error("Database error:", err);
         return res.status(500).json({ error: "Error creating order" });
      }

      const orderID = result.insertId;
      const orderItems = products.map(product => [orderID, product.product.id, product.quantity, product.product.price]);

      const orderItemsSql = "INSERT INTO order_items (order_id, product_id, quantity, price) VALUES ?";
      db.query(orderItemsSql, [orderItems], (err, result) => {
         if (err) {
            console.error("Database error:", err);
            return res.status(500).json({ error: "Error adding order items" });
         }

         const updateStockSql = "UPDATE products SET stock = stock - ? WHERE id = ?";
         products.forEach(product => {
            db.query(updateStockSql, [product.quantity, product.product.id], (err, result) => {
               if (err) {
                  console.error("Database error:", err);
                  return res.status(500).json({ error: "Error updating stock" });
               }
            });
         });

         res.json({ status: "Success", message: "Order created successfully" });
      });
   });
});
```

Рисунок 3.17 – Процес створення замовлення

Цей код запускає сервер на порту 8081 та виводить повідомлення про успішний запуск (рис. 3.18).

```
app.listen(8081, () => {
   console.log("Server is running on port 8081");
});
```

Рисунок 3.18 – Запуск сервера

Серверна частина забезпечує всі необхідні операції для обробки запитів від клієнтського додатку, взаємодію з базою даних та забезпечення безпеки за допомогою JWT.

3.3 Тестування та відлагодження системи

Тестування та відлагодження системи є важливими етапами розробки, які забезпечують її надійність, стабільність та відповідність вимогам. Процес тестування включає в себе як автоматизовані тести, так і ручне тестування різних аспектів системи.

Автоматизоване тестування було проведено з використанням бібліотек Jest та Mocha. Для клієнтської частини, реалізованої за допомогою React, були створені модульні тести, які перевіряли роботу окремих компонентів інтерфейсу, їх взаємодію та коректність відображення даних. Тестування серверної частини здійснювалося з використанням Mocha, що дозволяло перевіряти правильність обробки запитів, взаємодію з базою даних та відповідність бізнес-логіки.

Ручне тестування включало перевірку всіх основних функціональних можливостей системи. Це включало реєстрацію та вхід користувачів, перегляд списку продуктів, додавання продуктів до кошика, оформлення замовлень та перевірку історії замовлень. Кожна з цих функцій була перевірена на

коректність роботи, зручність використання та відповідність очікуванням.

Відлагодження системи здійснювалося за допомогою інструментів розробника браузера для клієнтської частини та Node.js для серверної частини. Використання цих інструментів дозволяло виявляти та виправляти помилки в коді, аналізувати роботу запитів та відповіді сервера, а також моніторити стан бази даних.

Онлайн-сервіс замовлення літератури складається з кількох основних компонентів, кожен з яких виконує свою роль у забезпеченні функціональності системи. Презентаційний шар, реалізований за допомогою React, забезпечує взаємодію з користувачем через зручний та інтуїтивно зрозумілий інтерфейс. Серверна частина, побудована на Node.js з використанням Express, обробляє запити від клієнтського додатку, взаємодіє з базою даних MySQL та забезпечує безпеку даних через JWT.

На головній сторінці (рис. 3.19) користувач бачить популярні книги та новинки, має можливість пошуку та фільтрації продуктів.

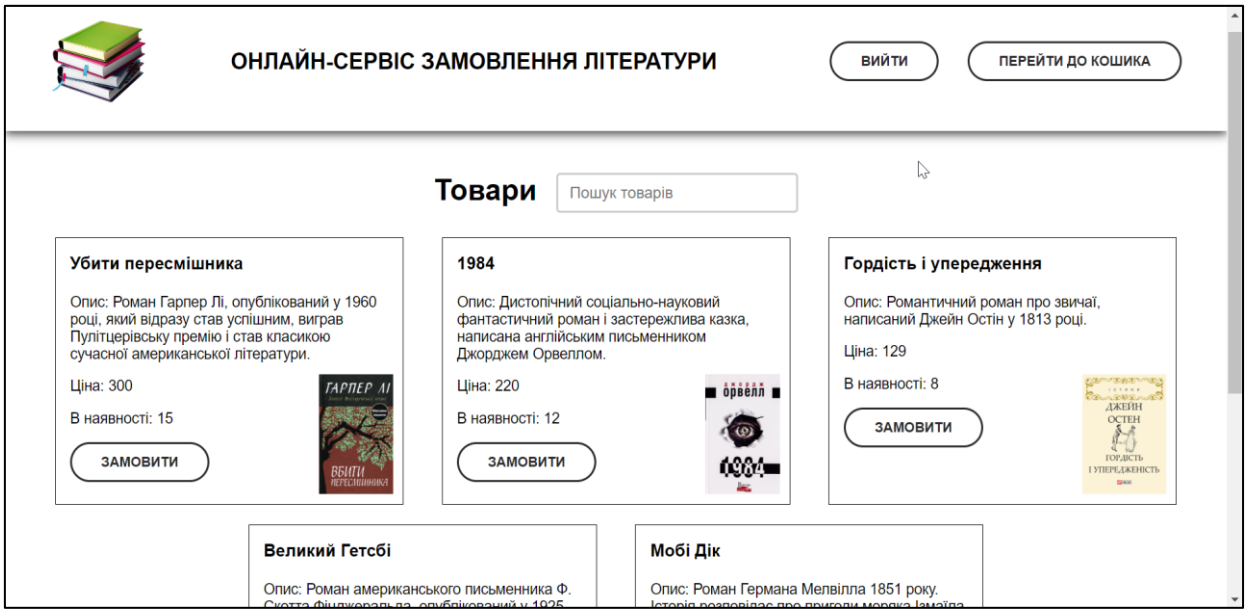

Рисунок 3.19 – Головна сторінка

Користувачі можуть переглядати детальну інформацію про книги, додавати їх до кошика, а також оформляти замовлення (рис. 3.20, 3.21).

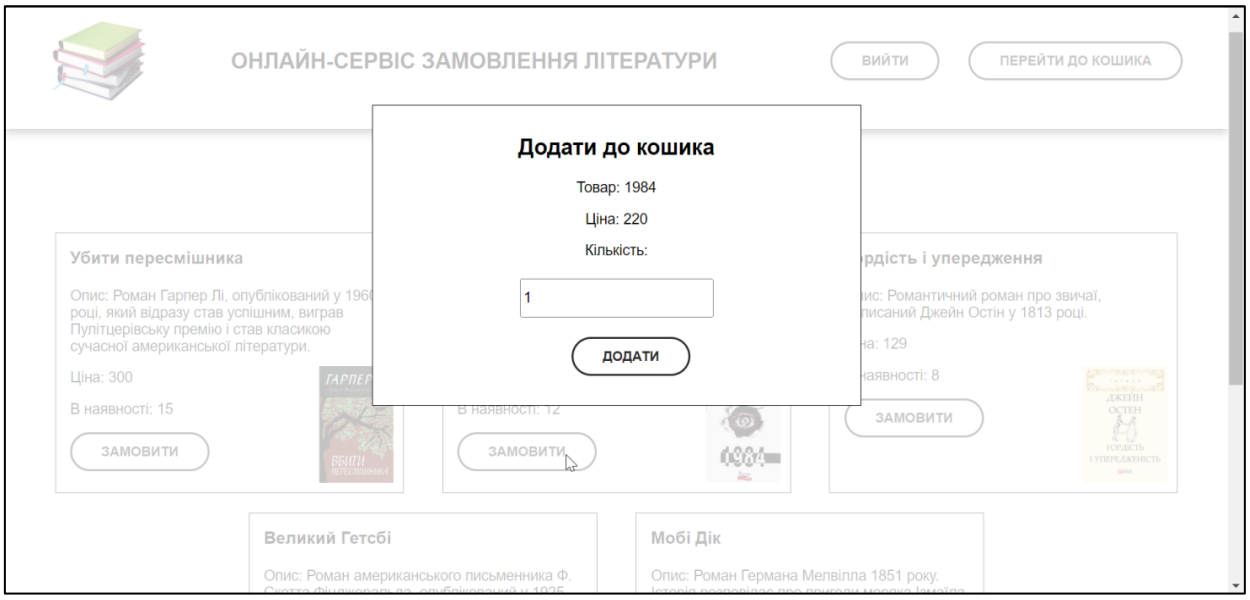

Рисунок 3.20 – Додавання товарів до кошика

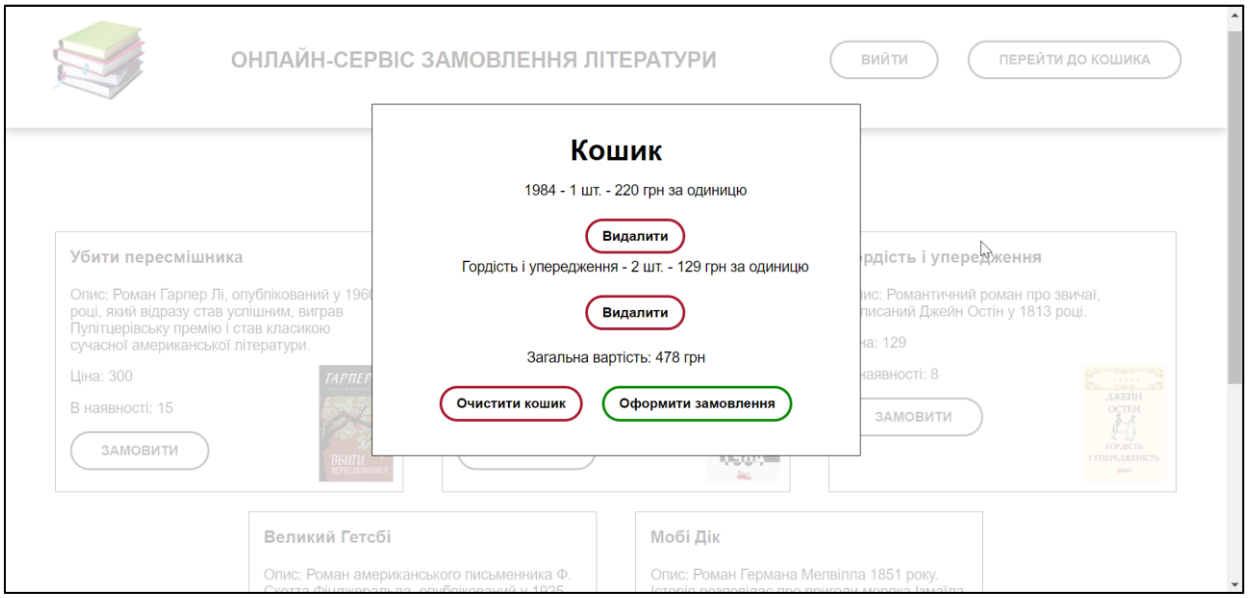

Рисунок 3.21 – Кошик та оформлення замовлення

Реєстрація та вхід користувачів (рис. 3.22, 3.23) забезпечуються формами з валідацією даних та безпечною передачею інформації.

Серверна частина забезпечує обробку всіх запитів від клієнтів. Після реєстрації або входу користувача система генерує JWT, який зберігається у кукі та використовується для аутентифікації подальших запитів. Сервер також обробляє запити на отримання списку продуктів, створення замовлень та управління даними користувачів.

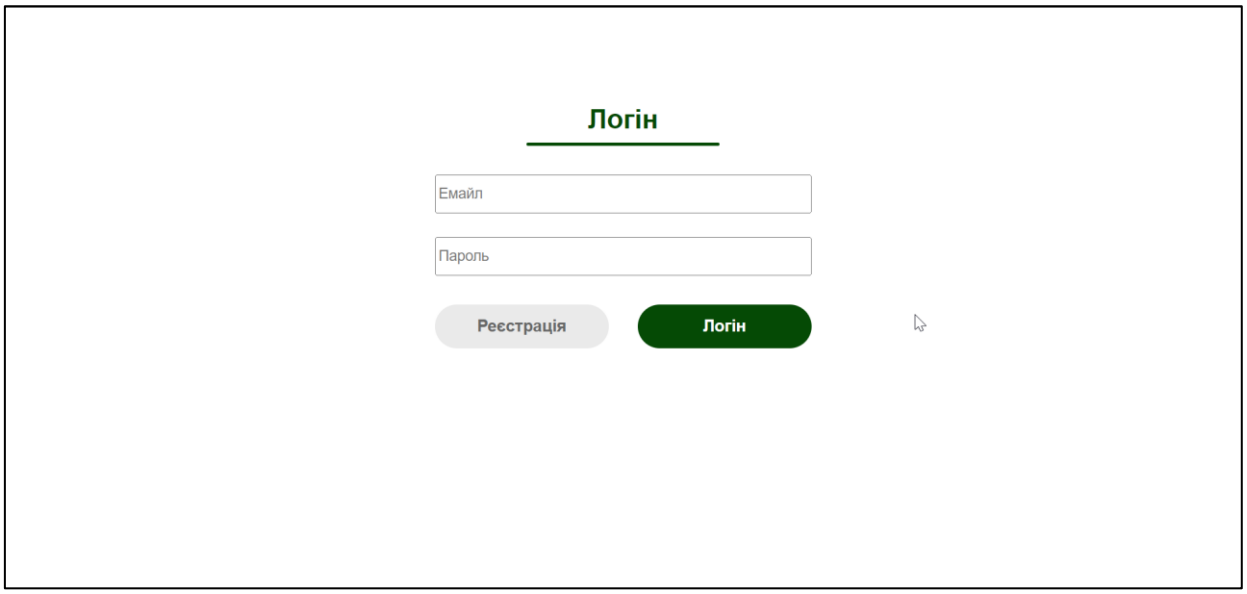

Рисунок 3.22 – Форма входу

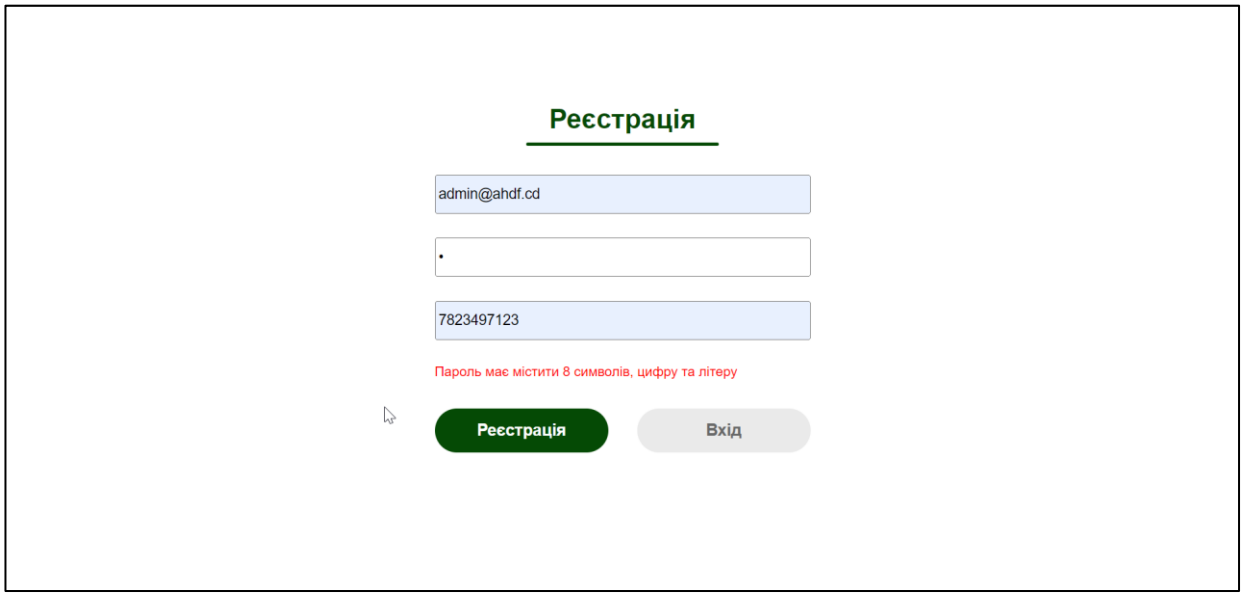

Рисунок 3.23 – Валідація реєстрації

База даних MySQL зберігає всі необхідні дані, включаючи інформацію про користувачів, продукти та замовлення (рис. 3.24). Таблиці users, products, orders та order_items забезпечують структуру даних та зв'язки між ними, що дозволяє швидко та ефективно виконувати запити до бази даних.

Тестування та відлагодження системи дозволили забезпечити високу якість роботи сервісу, зручність використання та відповідність функціональним вимогам. Система готова до експлуатації та може бути

розширена у майбутньому для додавання нових функцій та покращення користувацького досвіду.

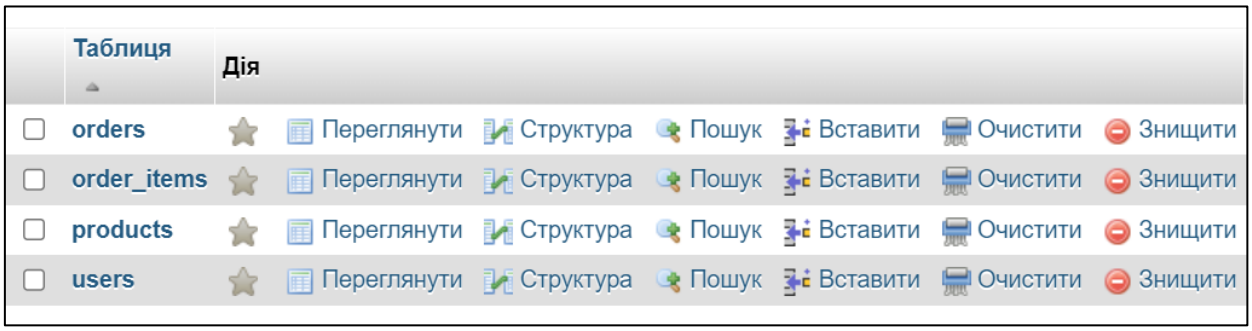

Рисунок 3.24 – База даних

ВИСНОВКИ

У ході розробки онлайн-сервісу замовлення літератури було досягнуто кілька важливих результатів, які підтверджують ефективність обраних рішень та технологій.

Перш за все, було виконано детальний аналіз предметної області та існуючих аналогів. Це дозволило визначити основні вимоги до системи та сформулювати завдання для її розробки. На основі цього аналізу було створено проєкт архітектури системи, який забезпечує її модульність, масштабованість та зручність в обслуговуванні.

Для розробки клієнтської частини системи було обрано фреймворк React, що дозволило створити інтуїтивно зрозумілий та зручний інтерфейс користувача. Завдяки використанню компонентного підходу вдалося досягти високої гнучкості та повторного використання коду. Використання бібліотеки Material-UI забезпечило сучасний дизайн та узгодженість стилів інтерфейсу.

Серверна частина системи, реалізована за допомогою Node.js та Express, забезпечує надійну обробку запитів від клієнтського додатку та взаємодію з базою даних MySQL. Використання JWT для аутентифікації користувачів дозволило забезпечити високий рівень безпеки даних.

Проєктування структури бази даних було виконано з урахуванням вимог до зберігання даних про користувачів, продукти та замовлення. Завдяки чітким зв'язкам між таблицями система може ефективно виконувати складні запити та забезпечувати швидкий доступ до необхідної інформації.

Процес тестування та відлагодження системи дозволив виявити та виправити помилки, забезпечити стабільність та надійність роботи системи. Використання автоматизованих тестів для клієнтської та серверної частин значно прискорило цей процес та підвищило його ефективність.

У результаті виконаної роботи було створено функціональний онлайн-сервіс замовлення літератури, який відповідає вимогам до сучасних

вебдодатків. Система забезпечує зручний інтерфейс для користувачів, надійний захист даних та ефективну обробку замовлень. Вона може бути легко розширена та модифікована у майбутньому для додавання нових функцій та покращення користувацького досвіду.

Таким чином, поставлені завдання було успішно виконано, а результати роботи підтверджують доцільність обраних підходів та технологій. Онлайн-сервіс готовий до впровадження та подальшої експлуатації, що забезпечить його користувачам зручний та надійний спосіб замовлення літератури через інтернет.

ПЕРЕЛІК ПОСИЛАНЬ

1. Бондарчук Т. А. Віртуальний світ бібліотек, 2015. С. 39-45.

2. Amazon.com. URL: https://www.amazon.com/books-used-books-textbooks/b?ie=UTF8&node=283155 (дата звернення: 15.03.2024).

3. Google Books. Google Books. URL: https://books.google.com/ (дата звернення: 25.03.2024).

4. Barnes and Noble Consumer Verified Ratings & Reviews. ResellerRatings. URL: https://www.resellerratings.com/store/Barnes_and_Noble (дата звернення: 11.04.2024).

5. Rumpe B. Modeling with UML. Cham : Springer, 2016. 281 p.

6. Eriksson H. E., Penker M. Business modeling with UML. New York, 2000. 12 p.

7. Banks A., Porcello E. Learning React: functional web development with React and Redux. O'Reilly Media, 2017. 208 p.

8. Teixeira P. Professional Node. js: Building Javascript based scalable software. John Wiley Sons, 2012. 408 p.

9. Widenius M., Axmark D. MySQL reference manual: documentation from the source. O'Reilly Media, 2002. 802 p.

10. Microsoft. Visual Studio Code – Code Editing. Redefined. Visual Studio Code – Code Editing. Redefined. URL: https://code.visualstudio.com/ (дата звернення: 20.04.2024).

ДОДАТОК А

Онлайн-сервіс літератури

Це код складається з кількох частин, що реалізують бекенд на Node.js з використанням Express та MySQL, а також фронтенд на React. Основне завдання цієї системи – забезпечення реєстрації, авторизації та управління кошиком користувача для онлайн-сервісу замовлення літератури.

А.1 Налаштування бекенду

На рисунку А.1 зображено рядки імпортують необхідні модулі та налаштовують Express додаток для роботи з JSON, куками та підтримкою CORS.

```
const express = require("express");
const mysql = require("mysql");
const cors = require("cors");
const jwt = require("jsonwebtoken");
const cookieParser = require("cookie-parser");

const app = express();
app.use(express.json());
app.use(cookieParser());
```

Рисунок А.1 – Налаштування бекенду

А.2 Налаштування CORS

На рисунку А.2 зображено блок який налаштовує CORS для взаємодії з фронтендом, що працює на http://localhost:3000.

```
const corsOptions = {
   origin: 'http://localhost:3000',
   methods: ["POST", "GET", "PUT", "DELETE"],
   credentials: true,
   optionSuccessStatus: 200
};
app.use(cors(corsOptions));
```

Рисунок А.2 – Налаштування CORS

А.3 Налаштування підключення до бази даних

На рисунку А.3 зображено налаштування підключення до MySQL бази даних з необхідними параметрами.

```
const db = mysql.createConnection({
   host: "localhost",
   user: "root",
   password: "",
   database: "literature_order"
});
```

Рисунок А.3 – Налаштування бази даних

А.4 Перевірка користувача

На рисунку А.4 зображена функція перевіряє наявність та валідність JWT токена у куках. Якщо токен не дійсний або відсутній, користувач не авторизований.

```
const verifyUser = (req, res, next) => {
   const token = req.cookies.token;
   if (!token) {
      return res.json({ error: "You are not authorized" });
   } else {
      jwt.verify(token, "jwt-secret-key", (err, decoded) => {
```

```
            if (err) {
                return res.json({ error: "Token is not valid" });
            } else {
                req.id = decoded.id;
                req.table_number = decoded.table_number;
                next();
            }
        });
    }
};
```

Рисунок А.4 – Перевірка користувача

А.5 Маршрут для отримання домашньої сторінки

На рисунку А.5 зображено маршрут використовує verifyUser для аутентифікації та повертає інформацію про користувача.

```
app.get('/homepage', verifyUser, (req, res) => {
    return res.json({
        status: "Success",
        id: req.id,
        username: req.email,
    });
});
```

Рисунок А.5 – Маршрут для отримання домашньої сторінки

А.6 Маршрут для логіну користувача

На рисунку А.6 зображено маршрут перевіряє вхідні дані користувача і, якщо вони вірні, створює JWT токен та повертає його клієнту.

```
app.post('/login', (req, res) => {
    const { email, password } = req.body;
    const sql = "SELECT * FROM users WHERE email = ? AND password = ?";
    db.query(sql, [email, password], (err, data) => {
```

```
        if (err) {
            return res.status(500).json({ error: "Database query error" });
        }
        if (data.length > 0) {
            const userID = data[0].id;
            const username = data[0].email;
            const token = jwt.sign({ userID, username }, "jwt-secret-key", { expiresIn: "1d" });
            res.cookie('token', token, { httpOnly: true });
            return res.json({ status: "Success", userID: userID });
        } else {
            return res.status(404).json({ status: "Login failed", message: "No account found!" });
        }
    });
});
```

Рисунок А.6 – Маршрут для логіну користувача

А.7 Маршрут для створення замовлення

На рисунку А.7 зображено маршрут створює нове замовлення, додає продукти до замовлення та оновлює кількість товарів на складі.

```
app.post('/create-order', (req, res) => {
    const { userID, products, totalPrice } = req.body;
    if (!userID || !products || !Array.isArray(products) || products.length === 0) {
        return res.status(400).json({ error: "Invalid order data" });
    }

    const orderSql = "INSERT INTO orders (user_id, total_amount) VALUES (?, ?)";
    db.query(orderSql, [userID, totalPrice], (err, result) => {
        if (err) {
            console.error("Database error:", err);
            return res.status(500).json({ error: "Error creating order" });
        }

        const orderID = result.insertId;
        const orderItems = products.map(product => [orderID, product.product.id,
product.quantity, product.product.price]);

        const orderItemsSql = "INSERT INTO order_items (order_id, product_id, quantity, price)
VALUES ?";
```

```
        db.query(orderItemsSql, [orderItems], (err, result) => {
            if (err) {
                console.error("Database error:", err);
                return res.status(500).json({ error: "Error adding order items" });
            }

            const updateStockSql = "UPDATE products SET stock = stock - ? WHERE id = ?";
            products.forEach(product => {
                db.query(updateStockSql, [product.quantity, product.product.id], (err, result) => {
                    if (err) {
                        console.error("Database error:", err);
                        return res.status(500).json({ error: "Error updating stock" });
                    }
                });
            });

            res.json({ status: "Success", message: "Order created successfully" });
        });
    });
});
```

Рисунок А.7 – Маршрут для створення замовлення

А.8 Компонент CartModalContent

На рисунку А.8 зображено компонент відповідає за відображення вмісту кошика та оформлення замовлення.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import Modal from 'react-modal';

function CartModalContent({ cart, removeFromCart, clearCart }) {
    const [orderSuccess, setOrderSuccess] = useState(false);
    const [totalPrice, setTotalPrice] = useState(0);

    useEffect(() => {
        const calculateTotalPrice = () => {
            const total = cart.reduce((sum, item) => sum + item.product.price * item.quantity, 0);
            setTotalPrice(total);
        };
```

```
    calculateTotalPrice();
  }, [cart]);

  const handlePlaceOrder = () => {
    axios.post("http://localhost:8081/create-order", { userID: 1, products: cart, totalPrice })
      .then(response => {
        setOrderSuccess(true);
        clearCart();
      })
      .catch(error => {
        console.error('Error placing order:', error);
        alert('Помилка при оформленні замовлення. Будь ласка, спробуйте знову.');
      });
  };

  const handleCloseModal = () => {
    setOrderSuccess(false);
  };

  return (
    <div>
      <h2>Ваш кошик</h2>
      {cart.length === 0 ? (
        <p>Ваш кошик порожній.</p>
      ) : (
        <div>
          <ul>
            {cart.map((item, index) => (
              <li key={index}>
                {item.product.name} - {item.product.price} грн. x {item.quantity}
                <button onClick={() => removeFromCart(item.product.id)}>Видалити</button>
              </li>
            ))}
          </ul>
          <h3>Загальна вартість: {totalPrice} грн.</h3>
          <button onClick={handlePlaceOrder}>Оформити замовлення</button>
        </div>
      )}
      <Modal
        isOpen={orderSuccess}
        onRequestClose={handleCloseModal}
        contentLabel="Order Success Modal"
        ariaHideApp={false}
```

```
        >
          <h2>Замовлення оформлено успішно!</h2>
          <button onClick={handleCloseModal}>Закрити</button>
        </Modal>
      </div>
  );
}

export default CartModalContent;
```

Рисунок А.8 – CartModalContent

Цей компонент відображає вміст кошика, розраховує загальну вартість та дозволяє користувачеві оформити замовлення. При успішному оформленні замовлення показується модальне вікно з повідомленням про успіх.

Наша система забезпечує повний цикл обробки замовлень для онлайн-сервісу літератури: від реєстрації та авторизації користувачів до оформлення та обробки замовлень. Бекенд реалізований на Node.js з використанням Express та MySQL, тоді як фронтенд використовує React для динамічного взаємодії з користувачем.