

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему

РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ
ІНТЕРНЕТ-МАГАЗИНУ КНИГАРНІ

Виконав: студент 4 курсу, групи КНТ-520
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерна інженерія»

ЗАХАРЧЕНКО К.Р.

(ПРИЗВИЩЕ та ініціали)

Керівник ХОХЛОВ М.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОЗИНА Г.Л.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти бакалаврський
 Спеціальність 123 Комп'ютерна інженерія
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерна інженерія
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
 Завідувач кафедри КУДЕРМЕТОВ Р.К.

“ ” 2024 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ЗАХАРЧЕНКО Костянтина Романовича
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка комп'ютерної системи інтернет-магазину
книгарні

керівник проєкту (роботи) ХОХЛОВ Микола Миколайович
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від “01” квітня 2024 року № 105

2. Строк подання студентом проєкту (роботи) 24 травня 2024 року

3. Вихідні дані до проєкту (роботи) наявні технології для розробки веб-систем, алгоритм авторизації користувача за допомогою сервіса Google.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1) Вибір технологій для реалізації системи;

2) Проєктування бази даних системи;

3) Програмна розробка комп'ютерної системи

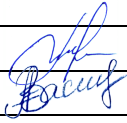
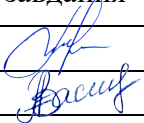
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Алгоритм оформлення замовлення;

Відношення таблиць бази даних системи;

Сторінки сайту системи інтернет-магазину книгарні.

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ХОХЛОВ М.М., ст. викл.		
нормоконтроль	ПОЛЬСЬКА О.В., ст. викл.		

7. Дата видачі завдання 25.02.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних технологій для реалізації системи	20.03.2024 р.	
2	Розробка структури системи	25.03.2024 р.	
3	Розробка алгоритму роботи системи	01.04.2024 р.	
4	Розробка бази даних системи	12.04.2024 р.	
5	Розробка серверної частини	24.04.2024 р.	
6	Розробка клієнтської частини	13.05.2024 р.	
7	Оформлення пояснювальної записки	22.05.2024 р.	
8	Оформлення графічного матеріалу	22.05.2024 р.	

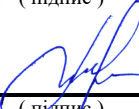
Студент (ка)


(підпис)

Костянтин ЗАХАРЧЕНКО

(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)


(підпис)

Микола ХОХЛОВ

(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 62 с., 22 рис., 11 табл., 1 додаток, 10 джерел

КОМП'ЮТЕРНА СИСТЕМА, EXPRESS, NODE.JS, HTML, CSS, JAVASCRIPT, MYSQL, HEIDISQL

Об'єкт розробки – комп'ютерна система інтернет-магазину книгарні.

Мета проєкту – розробка системи яка надасть клієнту можливість шукати та замовляти книги, читати раніше придбанні електронні книги та прослуховувати аудіо книги, інформація про які зберігається на сервері.

В ході розробки було проведено аналіз наявних рішень. Спроектowana база даних, в результаті чого було визначено сутності, зв'язки між ними та побудована реляційна модель. Розроблена серверна частина додатку за допомогою фреймворку Node.js та бібліотеки Express. Розроблена клієнтська частина за допомогою стандартного набору технологій HTML, CSS та JavaScript.

Розроблена комп'ютерна система має можливості пошуку, перегляду інформації, додавання до кошику книг, оформлення замовлення, авторизація за допомогою сервіса Google, читання електронних та прослуховування аудіо книг, інформація про книги та їх зображення зберігається у базі даних на сервері.

ЗМІСТ

Вступ.....	7
1 Вибір технологій розробки.....	8
1.1 Огляд існуючих рішень	8
1.2 Вибір архітектури web-системи.....	10
1.2.1 Монолітна архітектура.....	10
1.2.2 Serverless архітектура.....	11
1.2.3 Мікросервісна архітектура	13
1.2.4 Клієнт-серверна архітектура	14
1.3 Вибір серверних технологій.....	15
1.3.1 Java	16
1.3.2 Ruby	17
1.3.3 JavaScript	18
1.3.4 Python.....	19
1.3.5 PHP	20
1.4 Вибір клієнтських технологій	21
1.4.1 AngularJS	23
1.4.2 React	24
1.4.3 VueJS.....	24
1.5 Вибір бази даних	25
1.6 Висновки з вибору технологій розробки	27
2 Проєктування системи	28
2.1 Вимоги до системи.....	28
2.2 Проєктування бази даних	30
2.3 Діаграми послідовностей.....	37
3 Розробка комп'ютерної системи.....	39
3.1 Розробка серверної частини	39
3.1.1 Початкова ініціалізація серверу.....	39

3.1.2 Забезпечення захисту бази даних	40
3.1.3 Налаштування авторизації користувача	40
3.1.3 Додавання загальних маршрутів	43
3.2 Розробка клієнтської частини	48
Висновки	56
Перелік джерел посилання	57
Додаток А Основні коди програмних модулів.....	58

ВСТУП

Комерція в Інтернеті відкриває багато можливостей для торгівлі. Останнім часом значно зростає кількість людей, які вважають за краще займатися своєю професійною діяльністю, а також робити особисті покупки саме через Інтернет [1]. Це набагато зручніше для особистих потреб, оскільки не потрібно витратити час на дорогу. Комерційна діяльність в Інтернеті охоплює рекламу, розповсюдження товарів і послуг через мережу. Електронна комерція і інтернет-магазини нерозривно пов'язані між собою.

Інтернет-магазини є однією з форм електронної комерції, яка дозволяє споживачам безпосередньо купувати товари і послуги від продавців через Інтернет, за допомогою браузера або мобільного додатка. Споживачі можуть знайти бажаний товар, відвідавши вебсайт продавця або шукати його серед інших постачальників за допомогою пошукових систем для покупок, які відображають наявність товару та ціни в різних електронних магазинах.

Електронні та аудіо книги стали значною частиною сучасної культури читання, пропонуючи зручність та доступність, які традиційні паперові книги не можуть забезпечити. Ці формати книг відкривають нові можливості для читачів, дозволяючи їм мати бібліотеку в кишені. З точки зору виробництва, електронні та аудіо книги зменшують витрати на друк та розповсюдження, що робить їх більш доступними для споживачів.

Метою кваліфікаційної роботи є розробка інтернет-магазину книгарні, який буде займатись продажем фізичних, електронних та аудіо книг. Актуальність цієї теми підтримується тим що попит на електронні книжки в Україні стрімко зростає.

1 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

Перед початком розробки комп'ютерної системи інтернет-магазину книгарні слід оглянути вже існуючі рішення та визначитися з набором технологій які будуть використовуватися. Для цього потрібно проаналізувати та порівняти між собою наявні технології які доступні для використання та обрати найбільш слушний варіант. Заздалегідь обрані технології розробки допоможуть скоротити час на розробку системи та підвищить загальну її якість.

1.1 Огляд існуючих рішень

При огляді існуючих рішень вирішено звернути увагу на наступні деталі: можливість авторизації через сторонні сервіси, наявність у продажі електронних та аудіо- книг. Для прикладу, було обрано декілька існуючих інтернет-магазинів що спеціалізуються на продажі книжок, а саме «Дім Книги», «Bookling» та «Книгарня Є».

«Дім книги» має застарілий проте функціональний дизайн зображений на рисунку 1.1.

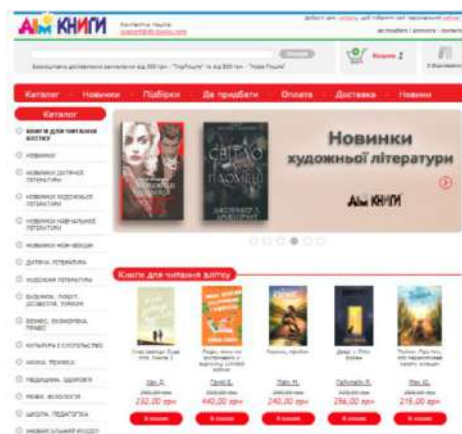


Рисунок 1.1 – Головна сторінка інтернет магазину «Дім Книги»

Для створення облікового запису на сайті можна використовувати свій аккаунт Google що спрощує користувачам процес реєстрації. Каталог надає користувачу зручний спосіб навігації по сайту для пошуку потрібної йому книги. Проте «Дім Книги» не має у наявності електронних та аудіо книг.

«Bookling» на відміну від попереднього прикладу має сучасний, привабливий дизайн зі зручною навігацією що зображено на рисунку 1.2.



Рисунок 1.2 – Головна сторінка інтернет магазину «Bookling»

Попри наявність сучасного дизайну, на цьому сайті відсутня можливість реєстрації за допомогою інших облікових записів, таких як Google або Facebook що змушує користувача створювати новий обліковий запис. «Bookling» також не має у наявності електронних та аудіо книг.

«Книгарня Є» пропонує сучасний і зручний вебсайт, як показано на рисунку 1.3.



Рисунок 1.3 – Головна сторінка інтернет магазину «Книгарня Є»

Сайт дозволяє користувачам створювати облікові записи, однак немає опції авторизації через сторонні сервіси. Навігація по каталогу книг добре організована і зрозуміла, що полегшує пошук необхідних видань. «Книгарня Є» має в асортименті електронні книги, що робить його зручним для користувачів, які віддають перевагу цифровим форматам. Проте, аудіокниги у наявності відсутні.

1.2 Вибір архітектури web-системи

Архітектура системи охоплює потік даних між різними компонентами та способи побудови застосунку що буде виконувати різні функції та оброблювати трафік. Різні архітектури можуть значно відрізнятися один від одного та бути найбільш оптимальними в залежності від ряду чинників, таких як: розмір системи, її складності та інших конкретних вимог. Вибір архітектури для системи є фундаментальним рішенням котре вплине на перелік критичних аспектів: масштабованість, складність обслуговування, можливість оновлення, безпека, продуктивність та вартість.

Серед архітектур які використовуються для побудови web-систем найбільш поширеними є наступні:

- монолітна архітектура;
- serverless архітектура;
- мікросервісна архітектура;
- клієнт-серверна архітектура.

1.2.1 Монолітна архітектура

Монолітна архітектура це найбільш простий спосіб розробки застосунку який відрізняється тим що увесь функціонал розміщується в одному спільному кодовому сховищі. Така архітектура легка у розгортанні завдяки тому, що увесь функціонал знаходиться у одному місці та до нього можна легко отримати доступ. З недоліків можна визначити складність у масштабуванні. Якщо частина

монолітного застосунку потребує масштабування через перенавантаження, то з огляду на тісний зв'язок між усіма частинами масштабувати доведеться увесь застосунок одночасно що є більш складною та вартісною задачею.

Монолітна архітектура добре підходить для прототипування та початкової стадії розробки застосунку що потребує швидкого запуску, маленьких та середніх проєктів де відсутня потреба у великій можливості до масштабування або для проєктів з обмеженими ресурсами які не можуть дозволити розгортання інших видів архітектур. Модель монолітної архітектури зображено на рисунку 1.4.

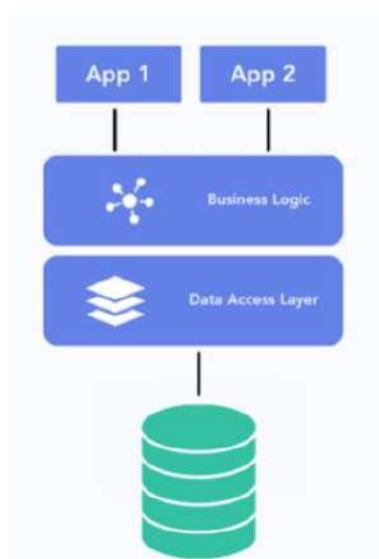


Рисунок 1.4 – Модель монолітної архітектури

Додатки розроблені з використанням монолітної архітектури також зазвичай одну загальну базу даних яка зберігає усі дані додатка від різних його частин. Розгортання монолітного додатка можна виконати з використанням лише одного серверу або віртуальної машини [2].

1.2.2 Serverless архітектура

Serverless архітектура (без серверна архітектура) – це підхід до проєктування програмного забезпечення без потреби налаштування серверної частини застосунку. Розробники, при використанні цієї архітектури, можуть сконцентруватися на написанні та розгортанні коду, а хмарний постачальник надає серверні можливості, бази даних і системи зберігання.

Однією з найпопулярніших без серверних архітектур є «функція як послуга», у ній розробники пишуть код до застосунку як набір окремих функцій. Кожна з функцій виконуватиме певне завдання. Після тестування розробники розгортають ці функції в обліковому записі хмарного постачальника. Коли функція викликається, хмарний постачальник або виконує викликану функцію на вже запущеному сервері, або якщо у момент виклику не запущено жодного сервера під дану функцію, він запускає новий сервер для виконання функції. Цей процес виконання є абстрагованим від погляду розробників, які зосереджуються на написанні та розгортанні коду програми.

Серед переваг даної архітектури є вартість, масштабованість та швидкість оновлення. Хмарні постачальники стягують плату лише під час використання системи, а саме за виклик кожної функції, тому коли система не використовується за не потрібно платити що значно зменшує вартість обслуговування, особливо для невеликих та середніх проєктів. Масштабованість не є проблемою оскільки екземпляри функцій створюються та видаляються автоматично хмарним постачальником, щоб відповідати навантаженню, при такому підході обмеженням у масштабованості є кількість доступних для розгортання серверів у хмарного постачальника. Завдяки тому, що розробникам не потрібно керувати жодними серверами та їх налаштуваннями то це знижує їх навантаження що відкриває можливості до більш швидкої розробки та розгортання нових функцій.

Недоліками у без серверній архітектурі являються втрата контролю, безпека, продуктивність та тестування. Якщо у хмарного постачальника відбудеться збій обладнання, втрата електроенергії або якась інша з поломок які можуть вплинути на один з серверів що використовується, тоді розробники не можуть нічого зробити і повністю залежать від постачальника хмарних послуг, щоб вирішити ці проблеми. Хмарний постачальник може запускати код різних застосунків від різних клієнтів одночасно на одному сервері що може призвести до проблем з безпекою у випадку якщо сервер налаштовано неправильно. У без серверних середовищах поширений такий етап функції як «холодний запуск», коли функція викликається уперше або

після тривалого часу бездіяльності то постачальнику хмарних технологій потрібно розгорнути новий сервер під цю функцію що додає декілька секунд затримки [3].

1.2.3 Мікросервісна архітектура

Мікросервісна архітектура це підхід, коли єдиний додаток складається з сукупності невеликих, самодостатніх, незалежних один від одного сервісів що спілкуються між собою за допомогою таких протоколів як HTTP. Кожен з сервісів розроблений для вирішення конкретної задачі, розгортається індивідуально та має свою базу даних. Оскільки сервіси не тісно зв'язані між собою то вони можуть бути написаними з використанням різних технологій, на різних мовах та використовувати різні вирішення для зберігання даних.

Найбільшою з переваг у мікросервісній архітектурі є те що вона дозволяє швидко змінювати щось у застосунку. Завдяки тому що різні сервіси не мають сильного зв'язку один між одним то можна вносити значні зміни у різні сервіси без страху що ці зміни негативно вплинуть на інші частини додатка. На рисунку 1.5 візуально зображено модель мікросервісної архітектури.

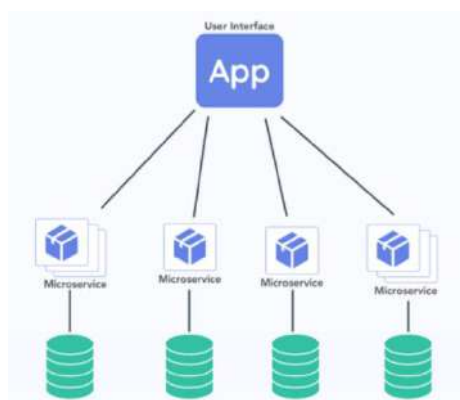


Рисунок 1.5 – Модель мікросервісної архітектури

Відокремлення сервісів також забезпечує надійність роботи додатка, у випадку якщо один з сервісів вийде з ладу, то це скоріш за все не значно вплине на загальну роботу додатка особливо для додатків великого розміру.

Серед недоліків мікросервісної архітектури можна визначити підвищену складність розгортання та обслуговування і мережеві затримки. Для коректної роботи потрібно узгодити формати повідомлень між кожною парою сервісів що

ускладнює процес проєктування та розробки додатка з мікросервісною архітектурою.

1.2.4 Клієнт-серверна архітектура

Клієнт-серверна архітектура являє собою систему у якій багато клієнтів надсилають запити та отримують відповіді від централізованого сервера. Сервери очікують надходження запитів від клієнтів і при їх отриманні надсилають клієнтам відповідь.

Клієнти і сервери є незалежними та можуть працювати паралельно. Немає жорсткої прив'язки клієнтів до серверів. Кожен з серверів має можливість одночасно оброблювати запити від різних клієнтів у той час клієнти можуть звертатися до різних серверів і це не впливає на загальний функціонал додатка. Для використання додатка клієнти повинні знати про наявні сервери, проте для них не обов'язково знати про існування інших клієнтів.

Головною відмінністю клієнт-серверної архітектури від інших є розподіл обов'язків між сервером та клієнтом. У клієнт-серверній архітектурі можна відокремити три рівні операцій:

- рівень представлення даних, який являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень управління даними, який забезпечує зберігання даних та доступ до них.

Найбільш поширеною на даний момент є трирівнева модель клієнт-серверної архітектури. Вона вирізняється тим що база даних відокремлюється від сервера у окремий рівень до якого за потреби сервери надсилаються запити та отримують відповіді. На рисунку 1.6 візуально зображено модель клієнт-серверної взаємодії.

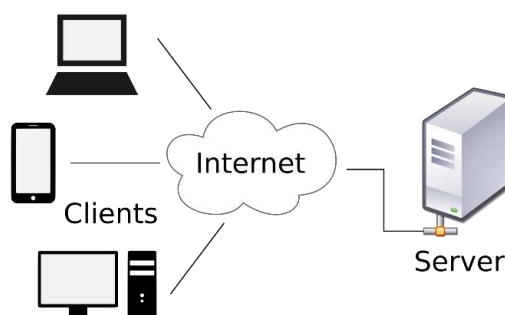


Рисунок 1.6 – Модель клієнт-серверної архітектури

З недоліків при використанні клієнт серверної архітектури можна визначити перш за все централізацію. Централізований сервер може стати вузьким місцем, що обмежуватиме продуктивність системи. Якщо ж цей сервер вийде з ладу, то, у випадку коли це єдиний сервер, увесь додаток перестане працювати. Другим недоліком є масштабованість. З ростом кількості клієнтів, сервер може стати перевантаженням, що призведе до зниження продуктивності, а вирішення проблеми у вигляді додавання нових серверів є складним та дорогим рішенням.

В порівнянні з іншими архітектурами клієнт-серверна може виявитися менш гнучкою і масштабованою. Однак, вона все ще залишається популярною через свою простоту і зручність управління.

1.3 Вибір серверних технологій

Під час вибору серверних технологій слід врахувати наступний перелік факторів: масштабованість, безпека, підтримка, інтеграція, швидкість розробки.

Масштабованість визначає, наскільки легко систему можна розширити для обробки більшої кількості навантаження. Безпека є критично важливою, оскільки сервер часто займається обробкою конфіденційних даних. Підтримка технології зі сторони спільноти та виробників може значно спростити розв'язання проблем та розвиток проєкту. Інтеграція з іншими системами та сервісами дозволяє

створювати гнучкі та масштабовані архітектури. Кожна з мов програмування має свої сильні та слабкі сторони, і її вибір залежить від конкретних вимог проєкту.

1.3.1 Java

Java – це високорівнева, об'єктно-орієнтована мова програмування що широко поширена для розробки вебзастосунків. Вона була популярним вибором серед розробників більше ніж два десятиліття поспіль, з мільйонами Java-застосунків у використанні по сьогоднішній день. Це швидка, безпечна та надійна мова програмування для розробки будь-чого, від мобільних застосунків та корпоративного програмного забезпечення до застосунків обробки даних та серверних технологій [4].

У темі безпеки Java надає надійні функції безпеки, що робить її популярним вибором для розробки безпечних програм. Серед таких функцій такі як:

- інкапсуляція;
- абстракція;
- спадкування;
- вбудований менеджер безпеки, який дозволяє розробникам визначати правила доступу.

Враховуючи вищесказане та враховуючи наявність вбудованих бібліотек безпеки і здатність виконувати код у ізольованому середовищі віртуальної машини, Java можна вважати досить безпечною мовою програмування для роботи з конфіденційними даними. Крім того, Java має перевагу у вигляді незалежності від платформи, що спрощує процес розробки. Будь-яка система, оснащена віртуальною машиною Java, здатна запускати Java-додатки без необхідності модифікації їхнього коду.

Динамічна пам'ять в Java керується автоматичним збирачем сміття. Це означає, що програмісти не повинні явно виділяти та звільняти пам'ять, як це відбувається в багатьох інших мовах програмування. Замість цього, коли об'єкт більше не використовується, збирач сміття автоматично звільняє пам'ять, що була виділена для цього об'єкта. Це значно спрощує розробку програм та зменшує ризик виникнення помилок, пов'язаних з управлінням пам'яттю. Проте це також є й

недоліком, через те що несвоєчасне спрацювання збирача сміття може негативно вплинути на швидкість виконання.

1.3.2 Ruby

Ruby – це об’єктно-орієнтована, динамічна мова програмування з відкритим кодом. Вона була створена з метою зробити програмування більш приємним. Розробка Ruby велась під впливом ряду інших мов програмування, включаючи Perl, Lisp, Smalltalk, Eiffel та Ada. Основною філософією при розробці Ruby була ідея створити мову, яка є простою у вивченні та використанні. Завдяки цьому вона має чистий синтаксис, який легко читати та розуміти. Що робить Ruby чудовим вибором для новачків та спрощує командну взаємодію [5].

Динамічна типізація дозволяє змінювати тип змінних протягом виконання програми. Це зручно для браузерних скриптів, але може призвести до помилок, тому потрібно зберігати обережність під час розробки застосунку.

Будучи динамічною мовою програмування вона використовується для написання скриптів, які виконуються безпосередньо. Вона не потребує компіляції в машинний код перед виконанням, на відміну від мов програмування, таких як C та C++. Також на додачу Ruby є кросплатформеною мовою програмування, що означає що вона може бути використана на багатьох платформах, включаючи Linux, Windows та macOS.

Завдяки активній спільноті, Ruby має велику кількість вбудованих бібліотек, які називаються гемами. Ці геми можуть бути використані для додавання функціональності до програм, що значно пришвидшує розробку. Вони включають все, від систем обробки файлів до вебсерверів.

Однією з найсильніших сторін Ruby є Ruby on Rails, часто просто називається Rails, це фреймворк для розробки вебдодатків. Він значно прискорює процес розробки, а також сприяє створенню додатків, які підлягають масштабуванню та легко підтримувати. Ruby on Rails є популярним рішенням для створення стартапів та вебсервісів завдяки своїй зручності використання.

Однак, одним з недоліків мови Ruby є те, що вона може бути повільнішою, ніж інші мови програмування. Оскільки Ruby є динамічною мовою, вона може

виконуватися повільніше, ніж компільовані мови, такі як C та C++. Ruby відома своїм споживанням пам'яті, особливо у порівнянні з іншими мовами програмування. Це може бути проблемою для додатків, які потребують великої кількості пам'яті для роботи або які повинні розгортатися на пристроях з обмеженою кількістю пам'яті, наприклад телефон. Іншим недоліком Ruby є те що вона хоч і має вбудовану підтримку потоків, проте вона не використовує їх ефективно через Global Interpreter Lock (GIL). GIL робить так, що тільки один потік може виконуватися в будь-який момент часу, що обмежує паралельність в Ruby.

1.3.3 JavaScript

JavaScript (JS) – легка, кросплатформена та одно поточна мова програмування. Вона також відома як мова сценаріїв для вебсторінок. На відміну від більшості мов програмування JS не має концепту вводу або виводу. Вона спроектована запускатися як мова сценаріїв в середовищі виконання і це задача середовища виконання надати механізми зв'язку з зовнішнім світом. Найбільш поширеним середовищем виконання є браузер [6].

JS є мовою програмування, яка використовує динамічну типізацію. Це означає, що розробник може оголошувати змінні без вказівки їх типу, і тип змінної може бути змінений протягом виконання програми. Ця гнучкість робить JS особливо корисним для скриптів, які виконуються в браузері, де типи даних можуть змінюватися в залежності від взаємодії користувача. Однак це також може призвести до помилок, якщо змінні використовуються неправильно, тому розробники повинні бути обережні при використанні динамічної типізації.

Через те що для виконання JS коду потрібне середовище виконання яке може підключитися до бази даних та використовувати файлову систему звичайний браузер не підходить. Для таких задач використовують Node.js.

Node.js – це середовище виконання з відкритим кодом, засноване на мові програмування JavaScript та надає розробникам змогу створювати серверну частину вебдодатків мовою сценаріїв JS. У Node.js використовують парадигму «JavaScript everywhere». Тобто не потрібно вивчати нову мову для написання серверної або клієнтської частини, що полегшує процес розробки, особливо для

маленьких проєктів. Одна з причин створення Node.js та його найбільша перевага це масштабованість. Node.js, на відміну від його попередника Apache, підтримує обробку кількох одночасних підключень.

Node.js має динамічну екосистему, з надмірною кількістю бібліотек, фреймворків та інструментів. Серед ключових компонентів є такі як:

- npm (Node Package Manager) це стандартний менеджер пакетів для Node.js. Він дозволяє встановлювати, керувати та ділитися модулями. В реєстрі npm можна знайти тисячі корисних модулів з відкритим кодом;
- Express.js це популярний фреймворк для Node.js що полегшує створення вебдодатків. Він спрощує маршрутизацію, обробку проміжного програмного забезпечення та керування запитами і відповідями. Багато розробників обирають Express.js для побудови API, вебсерверів та одно сторінкових застосунків;
- Socket.io: для спілкування в реальному часі Socket.io найбільш підходяща бібліотека. Вона забезпечує двонаправлений зв'язок між сервером і клієнтами за допомогою WebSockets або резервних механізмів;
- Mongoose: якщо застосунок взаємодіє з базою даних MongoDB, тоді дана бібліотека надасть зручний спосіб моделювання даних і взаємодії з базою даних. Він надає можливості перевірки схеми, проміжне програмне забезпечення та програмне забезпечення для створення запитів.

1.3.4 Python

Python – високорівнева, інтерпретована та об'єктно орієнтована мова програмування з динамічною типізацією. Розроблена у 1989 році і написана на мові програмування C. Завдяки використанню чистого синтаксису Python є легкою у читанні мовою, що робить її простою для розуміння та написання коду. Простота цієї мови програмування зменшує час та зусилля, необхідні для розробки та налагодження застосунків. Python є чудовим вибором для початківців завдяки простоті його синтаксису. На додачу до простоти Python також є кросплатформеною мовою програмування і може бути запущена на різних платформах, включаючи Windows, macOS та Linux.

Широка підтримка бібліотек та фреймворків дозволяє розробникам використовувати існуючі рішення для різних завдань, таких як веброзробка, аналіз даних, наукові обчислення, машинне навчання та багато іншого.

З недоліків Python є те що він є інтерпретованою мовою, що може призвести до меншої продуктивності порівняно з компільованими мовами. Python пропонує способи оптимізації продуктивності для вирішення цієї проблеми, проте їх не так легко реалізувати і навіть коли вони реалізовані, продуктивність все одно, хоч вже й не так сильно, буде меншою ніж у таких мовах як C/C++.

1.3.5 PHP

PHP – це широко використовувана мова програмування з відкритим кодом, що особливо підходить для веброзробки. PHP була оригінально аббревіатурою від Personal Home Page, але тепер вона стоїть за рекурсивною назвою PHP: Hypertext Preprocessor.

PHP є мовою, що виконується на сервері, і результат виконаного PHP-коду, який може бути будь-яким типом даних, таким як згенерований HTML або бінарні дані зображення, може формувати цілу або частину відповіді HTTP. PHP може використовуватися для багатьох завдань програмування поза вебконтекстом, таких як автономні графічні додатки. Код PHP також може виконуватися безпосередньо з командного рядка.

Стандартний інтерпретатор PHP, що працює на двигуні Zend, є вільним програмним забезпеченням, випущеним за ліцензією PHP. PHP була широко портована і може бути розгорнута на більшості вебсерверів на різноманітних операційних системах і платформах.

PHP може генерувати динамічний вміст сторінки, створювати, відкривати, читати, записувати, видаляти та закривати файли на сервері, збирати дані форми, відправляти та отримувати куки, додавати, видаляти, модифікувати дані в базі даних, контролювати доступ користувача, шифрувати дані. З PHP ви не обмежені виводом HTML. Ви можете виводити зображення або PDF-файли. Ви також можете виводити будь-який текст, такий як XHTML та XML.

1.4 Вибір клієнтських технологій

Стандартний набір технологій для створення клієнтської частини вебзастосунка включає HTML (мову текстової розмітки), CSS (каскадні таблиці стилів) та JavaScript як головну мову для розробки клієнтської частини. Щодо HTML, існує декілька старих версій, версія XHTML, яка спробувала інтегрувати XML, та найновіша версія – HTML5.

HTML5 – охоплює ряд нових елементів та атрибутів, що відображають типове використання на сучасних вебсайтах. HTML5 пропонує більш потужні та різноманітні можливості для створення вебсайтів та додатків, що підтримують мультимедіа, взаємодіють з програмними інтерфейсами тощо. HTML5 підтримується всіма сучасними браузерами і більш пристосований до пристроїв. Він може допомогти створити привабливі вебсайти з використанням CSS та JavaScript. HTML5 підтримує SVG та інші формати векторної графіки. Код стає чистішим, головним чином завдяки заміні div-тегів на семантичні елементи, які допомагають краще структурувати вміст вебсторінки та поліпшити читабельність. HTML5 підтримує геолокацію, що дозволяє визначати місцезнаходження користувачів. Нові стандарти були визначені для відтворення мультимедіа (анімація, аудіо, відео) безпосередньо в браузері без необхідності встановлювати додаткові плагіни. Підтримка вебсховища, введена в HTML5, дозволяє зберігати великі обсяги даних додатків локально, не впливаючи на продуктивність вашого вебдодатка.

CSS – це мова, за допомогою котрої описують як буде виглядати документ написаних мовою розмітки. Найчастіше CSS використовують у документах розмічених мовою HTML.

Використання CSS дає наступний перелік можливостей:

- відображення одного й того ж документа у різних стилях;
- використання різних CSS-правил в залежності від розміру монітора або використаного браузера;

- додаткові можливості оформлення;
- простота подальшої зміни дизайну, не потрібно вносити зміни у кожен сторінку.

Правила в CSS працюють по пріоритету, правило що розміщене нижче по каскаду матиме більше ваги. Це дозволяє отримати передбачуваний результат у випадку, коли до одного елемента, одночасно застосовуються декілька стильових правил. Структура CSS правил зображена на рисунку 1.7.



Рисунок 1.7 – Структура CSS правил

Усі CSS правила складаються з селектора і блоку оголошень укладеного у фігурні дужки. Всередині блоку оголошень може знаходитись одне або кілька оголошень, розділених крапкою з комою. Оголошення – це рядок, складений з css властивості та її значення.

Кожне правило починається з селектора, що вказує на ті html-елементи, до яких буде застосоване css-правило [7].

Для CSS існує розширення SASS (Syntactically Awesome Style Sheets) що надає додаткову функціональність та передові можливості, такі як змінні, вкладені правила, міксини та наслідування. SASS – це мова стилів, яка компілюється назад у CSS. Вона дозволяє використовувати усю додаткову функціональність будучи при цьому повністю сумісним з CSS синтаксисом. Також вже до SASS є розширення SCSS що не відрізняється функціонально, але додає підсвітку синтаксису та повертає фігурні дужки для кращої читабельності. Порівняння однакових CSS стилів з використанням різних розширень зображено на рисунку 1.8.

SASS	SCSS	CSS
<pre> nav ul margin: 0 padding: 2px list-style: square li display: inline-block a display: block padding: 5px 10px text-decoration: underline dotted green </pre>	<pre> nav { ul { margin: 0; padding: 2px; list-style: square; } li { display: inline-block; } a { display: block; padding: 5px 10px; text-decoration: underline dotted green; } } </pre>	<pre> nav ul { margin: 0; padding: 2px; list-style: square; } nav li { display: inline-block; } nav a { display: block; padding: 5px 10px; text-decoration: underline dotted green; } </pre>

Рисунок 1.8 – Порівняння стилів SASS, SCSS та CSS

Мову програмування JavaScript вже було детально розглянуто у пункті 1.2.3 проте є те що слід додати у випадку роботи на стороні клієнту. JavaScript у веб додатках використовується для додавання динаміки на сторінку шляхом зміни властивостей окремих елементів, для надсилання запитів до сервера та отримання відповіді, оновлення сторінки, перенаправлення та багато іншого.

JavaScript особливо розкривається як мова програмування при використанні його обширного набору фреймворків серед яких такі як: VueJS, AngularJS, React, Meteor

1.4.1 AngularJS

AngularJS це фреймворк що використовується для створення динамічних вебзастосунків зі структурною архітектурою. Цей фреймворк дозволяє використовувати HTML як мову шаблонів і полегшує розширення синтаксису HTML для чіткого й лаконічного вираження компонентів програми. Такі функції, як зв'язування даних і впровадження залежностей, зменшують необхідний код. Весь процес відбувається виключно в браузері, що робить його слухним доповненням до будь-якої серверної технології [8].

AngularJS розроблено для сумісності з HTML, CSS та JavaScript тому не потрібно вивчати нову мову сценаріїв. Він є простим у масштабуванні. Завдяки декільком вбудованими функціями, функціональність HTML можна розширити шляхом поєднання певної поведінки. Він є налаштовуваним, оскільки до нього можна додавати власні директиви.

Щоб реалізувати архітектуру MVC (Model-View-Controller, Модель-Представлення-Контролер), багато фреймворків потребують розбиття коду на окремі компоненти MVC в той час, як у AngularJS цей процес повністю

автоматизований що допомагає програмістам економити час, упорядковуючи їх код за них.

Для фреймворку Angular мова програмування TypeScript є основою. Оскільки вона служить базовою мовою Angular і є чітко типізованим, він допомагає розробникам уникнути безладного коду що повинно підвищити загальну ефективність написаного коду.

1.4.2 React

React.js є JavaScript бібліотекою з відкритим кодом, що була розроблена командою Facebook, що спрямована на спрощення складного процесу створення інтерактивних інтерфейсів користувача. Інтерфейс користувача створений за допомогою React можна представити як набір компонентів, кожен з яких відповідає за виведення невеликого фрагменту HTML-коду, який можна використовувати багаторазово [9].

Основна роль React у додатка полягає в тому, щоб обробляти рівень перегляду цього додатка забезпечуючи найкраще та найефективніше виконання візуалізації. Замість того, щоб працювати з усім інтерфейсом користувача як єдиним блоком, React.js заохочує розробників розділяти ці складні інтерфейси користувача на окремі багаторазові компоненти, які утворюють будівельні блоки цілого інтерфейсу користувача. При цьому фреймворк ReactJS поєднує швидкість і ефективність JavaScript з більш ефективним методом маніпулювання DOM для швидшого відтворення вебсторінок і створення високодинамічних та адаптивних вебдодатків.

1.4.3 VueJS

VueJS, який зазвичай називають Vue, є прогресивним фреймворком JavaScript із відкритим кодом, створеним як альтернативу більш важким фреймворкам, таким як AngularJS і React. Vue поєднує в собі підходи Angular і спрощені функції для зовнішнього інтерфейсу та розробки додатків. Основна бібліотека Vue зосереджена лише на рівні перегляду та розроблена для поступового впровадження в проєкти [10].

Vue це фреймворк JavaScript, що полегшує розробку графічного інтерфейсу користувача вебсайтів та одно сторінкових застосунків. Він був розроблений таким чином, щоб його можна було поступово адаптувати та уникнути багатьох вкорінених помилок існуючих монолітних фреймворків.

Основна бібліотека Vue легко інтегрується з іншими бібліотеками та існуючими проєктами. Vue також здатний запускати складні одно сторінкові застосунки за допомогою сучасних інструментів і бібліотек підтримки для додаткової функціональності.

Щоб гарантувати, що кілька частин програми, які використовують ті самі дані, завжди матимуть однакову версію цих даних Vue використовує двосторонні методи зв'язування даних.

1.5 Вибір бази даних

Існує багато різних систем керування базами даних (СКБД), кожна з яких призначена для вирішення певних завдань та має свої особливості та унікальні функції. Для вибору розглянемо наступний перелік СКБД: Microsoft Access, MongoDB, PostgreSQL, HeidiSQL та MySQL.

Microsoft Access є чудовим рішенням бази даних для окремих осіб, малого та середнього бізнесу та автономних відділів у великих компаніях. Її дуже легко встановити та зрозуміти. Вона дозволяє користувачам створювати таблиці, форми, запити та звіти. Це дозволяє використовувати Access як інструмент для збору та аналізу даних, а також для створення зручного інтерфейсу для користувачів.

Оскільки вся інформація з бази даних зберігається в одному файлі, це може уповільнити звіти, запити та форми. Технічне обмеження становить 255 одночасних користувачів, але реальне обмеження коливається лише від 10 до 80 що робить її поганим вибором для великих проєктів.

Також варто відзначити, що Access не є найкращим вибором для конфіденційних даних, оскільки він має обмежені можливості для захисту даних.

MongoDB невідмінну від інших баз даних є нереляційною, вона зберігає дані як записи, що складаються з колекцій та документів. Документи складаються з пар полів і значень, вони є основною одиницею даних у MongoDB. Документи схожі на JSON, але використовують варіант під назвою Binary JSON (BSON).

Як і інші бази даних NoSQL, MongoDB не вимагає попередньо визначених схем. Вона може зберігати будь-які типи даних. Це дає користувачам можливість створювати будь-яку кількість полів у документі, що полегшує масштабування баз даних MongoDB порівняно з реляційними базами даних.

Недоліком у плані безпеки є те що автентифікація користувача не ввімкнена за замовчуванням у базах даних MongoDB.

PostgreSQL - це потужна СКБД з відкритим кодом, яка має широкі можливості та призначена для складних проєктів. Вона, як і MySQL, має відкритий код. PostgreSQL гарантує дотримання принципів ACID (атомарність, стійкість до конкуренції, ізолюваність, довіреність), що забезпечує цілісність та надійність даних.

Ця СКБД надає широкий набір функцій для роботи з даними, включаючи підтримку структурованих та неструктурованих даних, географічних даних та багатоверсійну обробку даних.

PostgreSQL підтримує стандартні мови запитів SQL та PL/SQL, що спрощує перехід користувачів, знайомих з цими мовами, на PostgreSQL.

MySQL – це реляційна система керування базами даних (СКБД), яка є однією з найпопулярніших відкритих систем у світі. Розроблена компанією Oracle, MySQL використовує мову структурованих запитів (SQL) для створення, модифікації та керування даними.

Завдяки своєму відкритому коду, MySQL надає можливість безкоштовного використання, модифікації, публікації та розширення свого коду для будь-яких осіб чи організацій. Це робить MySQL ідеальним вибором для розробників, які шукають гнучкість та контроль над своїми проєктами.

MySQL також відома своїми потужними механізмами масштабування, які дозволяють відповідати вимогам навіть найбільших програм. Її власна архітектура реплікації дозволяє великим організаціям масштабувати програми для підтримки мільярдів користувачів.

Окрім того, MySQL пропонує різноманітні продукти та сервіси, включаючи MySQL HeatWave, Enterprise Edition, OEM/ISV, Cluster CGE, а також безкоштовні вебінари. Ці продукти та сервіси розширюють можливості MySQL, дозволяючи їй служити широкому спектру потреб користувачів.

1.6 Висновки з вибору технологій розробки

Проаналізувавши наведені технології та архітектури, у якості архітектури застосунку було обрано клієнт серверну три-рівневу архітектуру. Для розробки клієнтської частини вебзастосунка буде використовуватися стандартний набір технологій: HTML, CSS та JavaScript без додаткових фреймворків або препроцесорів. Для розробки серверної частини застосунку було обрано мову програмування JavaScript, тому що використання однієї мови для клієнтської та серверної частини полегшить розробку. Для запуску JavaScript застосунку у якості серверного буде використовуватись середовище виконання NodeJS, задля спрощення маршрутизації та налаштування серверу було обрано фреймворк Express.js. Для зберігання даних було обрано СКБД MySQL через широку підтримку та гнучкість.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Вимоги до системи

Процес проєктування комп'ютерної системи інтернет-магазину книгарні починається з визначення вимог до системи. Ці вимоги забезпечують основу для подальшої розробки та реалізації функціональних і нефункціональних аспектів сайту. Система інтернет-магазину книгарні повинна мати наступний функціонал:

- каталог продукції;
- можливість пошуку за назвою, жанром, автором або видавництвом;
- кошик та додавання товарів до нього;
- оформлення замовлення;
- можливість обрання дати відправки замовлення;
- автентифікація користувача за допомогою сторонніх сервісів;
- можливість редагування облікового запису користувачем;
- перегляд доступних користувачу електронних та прослуховування аудіо книг на сайті.

Виходячи з наведеного функціоналу можна поділити користувачів системи на три типи: гість, зареєстрований клієнт та адміністратор.

Гість це не зареєстрований або не авторизований користувач який не має доступу до деяких функцій системи. Перелік функціоналу до якого має гість зображено на рисунку 2.1.



Рисунок 2.1 – Функціонал користувача з роллю «Гість»

Зареєстрований та авторизований користувач має розширений, порівняно з гостем, набір функціоналу до якого він має доступ перелік якого зображено на рисунку 2.2.



Рисунок 2.2 – Функціонал користувача з роллю «Клієнт»

Найвищий рівень доступу до функціоналу системи має адміністратор. Він має повний контроль над системою та доступ до всіх її можливостей. Перелік доступного функціоналу для адміністратора наведено на рисунку 2.3.



Рисунок 2.3 – Функціонал користувача з роллю «Адміністратор»

Для авторизації користувачів на сайті можна використовувати сторонні сервіси замість того щоб зберігати пароль користувача самостійно. Цей підхід дозволяє забезпечити безпеку та зручність у використанні для користувачів.

Замість того, щоб вимагати від користувачів створення та запам'ятовування нового пароля, вони можуть скористатися своїм обліковим записом у вже відомому їм сервісі, такому як Google, Facebook або Twitter, для входу на сайт.

У якості сервісу за допомогою якого буде виконуватись авторизація користувачів було обрано Google. Це було зроблено з кількох причин. По-перше, Google є одним з найпопулярніших сервісів у світі, і багато людей мають аккаунт Google. По-друге, Google має сильні механізми безпеки що робить його безпечним вибором для авторизації користувачів. Крім того, Google надає простий в інтеграції API для авторизації. Враховуючи все вищезазначене, Google було обрано як основний сервіс для авторизації користувачів на сайті.

2.2 Проєктування бази даних

Проєктування бази даних починається з концептуального проєктування яке складається з визначення сутностей, їх атрибутів та відношеннями між ними. Для бази розроблюваної системи було визначено наступний перелік сутностей: «Книги», «Автори», «Жанри», «Видавництво», «Інформація про замовлення», «Формати книг», «Користувачі», «Області» та допоміжні сутності «Автори книг», «Жанри книг» та «Вміст замовлення» що допомагають у встановленні зв'язків багато-до-багатьох.

Сутність «Книги» містить наступний перелік атрибутів: код книги (унікальний ідентифікатор), код видавництва (зовнішній ключ), код формату (зовнішній ключ), назва (текстове, не пусте поле), ціна, кількість сторінок, опис, ISBN (текстове поле, максимум 10 символів), зображення (текстове поле що зберігає назву файлу з зображенням книги), кількість, сховано (Так або Ні, визначає чи буде книга відображатись на сайті), назва файлу (необов'язкове текстове поле яке вказує назву файлу для електронних та аудіо книг).

Сутність «Автори» містить атрибути код автора (унікальний ідентифікатор) та ПІБ (текстове поле).

Сутність «Жанри» містить атрибути код жанру (унікальний ідентифікатор) та назва (текстове поле).

Сутність «Видавництво» окрім атрибутів код видавництва (унікальний ідентифікатор) та назва (текстове поле) має перелік необов'язкових атрибутів інформація котрих може полегшити роботу інтернет-магазину а саме адреса (текстове поле), номер телефону (текстове поле), електронна пошта (текстове поле) та веб-сайт (текстове поле).

Сутність «Інформація про замовлення» включає в себе такі атрибути: код замовлення (унікальний ідентифікатор), ім'я користувача (текстове поле), прізвище користувача (текстове поле), номер телефону (текстове поле), електронна пошта (текстове поле), код області (зовнішній ключ), місто (текстове поле), відділення пошти (текстове поле), час замовлення (час коли запис був доданий у базу даних), код користувача (зовнішній ключ), користувацька дата відправки (дата, не обов'язкове поле), статус оплати замовлення (Так або Ні).

Сутність «Формати книг» містить атрибути код формату (унікальний ідентифікатор), формат (текстове поле, англійська назва формату) та назва (текстове поле, українська назва формату).

Сутність «Користувачі» містить наступний перелік атрибутів: код користувача (унікальний ідентифікатор), код Google (текстове поле), ім'я (текстове поле), прізвище (текстове поле), електронна пошта (текстове поле), номер телефону (текстове поле), код регіону (зовнішній ключ) та місто (текстове поле).

Сутність «Області» містить атрибути код області (унікальний ідентифікатор) та назва (текстове поле).

Для встановлення зв'язків багато-до-багатьох між сутностями було створено допоміжні сутності «Автори книг», «Жанри книг» та «Вміст замовлення» що мають унікальний ідентифікатор складений з двох атрибутів.

Сутність «Автори книг» складається з атрибутів код книги та код автора.

Сутність «Жанри книг» складається з атрибутів код книги та код жанру.

Сутність «Вміст замовлення» складається з атрибутів код книги, код замовлення та кількість.

Наступним етапом складання концептуальної моделі бази даних є визначення зв'язків між сутностями. Виходячи з опису сутностей можна визначити наступні зв'язки:

- «Книги» → «Автори» (багато-до-багатьох), через допоміжну сутність «Автори книг»;
- «Книги» → «Жанри» (багато-до-багатьох), через допоміжну сутність «Жанри книг»;
- «Видавництво» → «Книги» (один-до-багатьох);
- «Книги» → «Інформація про замовлення» (багато-до-багатьох), через допоміжну сутність «Вміст замовлення»;
- «Формати книг» → «Книги» (один-до-багатьох);
- «Користувачі» → «Інформація про замовлення» (один-до-багатьох);
- «Області» → «Користувачі» (один-до-багатьох);
- «Області» → «Інформація про замовлення» (один-до-багатьох).

Після концептуального проєктування, коли сутності та зв'язки між ними визначено переходимо до логічного проєктування, метою якого є перетворення концептуальної моделі в логічну модель з урахуванням особливостей обраної СКБД. В процесі проєктування логічної моделі бази даних визначається кількість і склад таблиць для зберігання вихідної інформації. Кожна таблиця представляється назвою, переліком атрибутів і первинним ключем. Визначаються зв'язки між таблицями, а також обмеження цілісності даних. Деталі про ці таблиці представлені в таблицях 2.1-2.11.

Таблиця 2.1 – Таблиця authors для сутності «Автори»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
author_id	Лічильник	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
name	Текстовий	Текст (255)	-	Так	

Таблиця 2.2 – Таблиця genres для сутності «Жанри»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
genre_id	Лічильник	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
name	Текстовий	Текст (255)	-	Так	

Таблиця 2.3 – Таблиця books_authors для сутності «Автори книг»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
book_id	Числовий	Натуральні числа	-	Так	Складений ключ
author_id	Числовий	Натуральні числа	-	Так	Складений ключ

Таблиця 2.4 – Таблиця books_genres для сутності «Жанри книг»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
book_id	Числовий	Натуральні числа	-	Так	Складений ключ
genre_id	Числовий	Натуральні числа	-	Так	Складений ключ

Таблиця 2.5 – Таблиця publisher для сутності «Видавництво»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
publisher_id	Лічильник	Натуральні числа	-	Так	Первинний ключ
name	Текстовий	Текст (255)	-	Так	
Address	Текстовий	Текст (255)	NULL	Ні	
phone_number	Текстовий	Текст (15)	NULL	Ні	
email	Текстовий	Текст (255)	NULL	Ні	
website	Текстовий	Текст (255)	NULL	Ні	

Таблиця 2.6 – Таблиця books для сутності «Книги»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
book_id	Лічильник	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
publisher_id	Числовий	Натуральні числа	-	Так	Зовнішній ключ
format_id	Числовий	Натуральні числа	1	Так	Зовнішній ключ
name	Текстовий	Текст (255)	-	Так	
price	Числовий	Дійсні числа	0	Так	
page_amout	Числовий	Натуральні числа	-	Так	
description	Текстовий	Текст	NULL	Ні	
ISBN	Текстовий	Текст (10)	NULL	Ні	
image	Текстовий	Текст (255)	-	Так	
quntity	Числовий	Натуральні числа	0	Так	
hidden	ENUM	“Yes”, “No”	“No”	Так	
file_name	Текстовий	Текст (255)	NULL	Ні	

Таблиця 2.7 – Таблиця books_formats для сутності «Формати книг»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
format_id	Лічильник	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
format	Текстовий	Текст (50)	-	Так	
name	Текстовий	Текст (50)	-	Так	

Таблиця 2.8 – Таблиця order_content для сутності «Вміст замовлення»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
order_id	Числовий	Натуральні числа	-	Так	Складений ключ
book_id	Числовий	Натуральні числа	-	Так	Складений ключ
amount	Числовий	Натуральні числа	-	Так	

Таблиця 2.9 – Таблиця order_info для сутності «Інформація про замовлення»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
order_id	Числовий	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
name	Текстовий	Текст (255)	-	Так	
surname	Текстовий	Текст (255)	-	Так	
phone_number	Текстовий	Текст (15)	NULL	Ні	
email	Текстовий	Текст (255)	-	Так	
region_id	Числовий	Натуральні числа	NULL	Ні	Зовнішній ключ
city	Текстовий	Текст (255)	NULL	Ні	
NovaPoshta	Текстовий	Текст (255)	NULL	Ні	
description	Текстовий	Текст	NULL	Ні	
order_time	Позначка часу	Повна дата та час	CURRENT_TIMESTAMP	Ні	
user_id	Числовий	Натуральні числа	NULL	Ні	Зовнішній ключ
paid_status	ENUM	“Yes”, “No”	“No”	Так	
pickUpDate	Дата	Дата	NULL	Ні	

Таблиця 2.10 – Таблиця users для сутності «Користувачі»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
user_id	Числовий	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
google_id	Текстовий	Текст (25)	-	Так	
name	Текстовий	Текст (255)	-	Так	
surname	Текстовий	Текст (255)	NULL	Ні	
email	Текстовий	Текст (255)	-	Так	
phone_number	Текстовий	Текст (15)	NULL	Ні	
region_id	Числовий	Натуральні числа	NULL	Ні	Зовнішній ключ
city	Текстовий	Текст (255)	NULL	Ні	

Таблиця 2.11 – Таблиця regions для сутності «Області»

Атрибут	Тип даних	Допустимі значення	Значення за замовчуванням	Обов'язково	Примітки
region_id	Лічильник	Натуральні числа	AUTO_INCREMENT	Так	Первинний ключ
name	Текстовий	Текст (255)	-	Так	

Спроектowana база даних є нормалізованою і відповідає вимогам третьої нормальної форми. Усі сутності та їх атрибути були ретельно проаналізовані з метою забезпечення, щоб кожен атрибут залежав тільки від первинного ключа сутності та не мав транзитивних залежностей. Це означає, що всі таблиці бази даних містять лише атомарні значення, унікальні рядки, і жодних повторюваних груп. Завдяки нормалізації, структура бази даних оптимізована для уникнення надлишкових даних і мінімізації аномалій при виконанні операцій додавання, оновлення та видалення. Як результат, база даних не тільки забезпечує цілісність і узгодженість даних, але й ефективну організацію зберігання та доступу до інформації.

На основі проведеного концептуального та логічного проектування було складено діаграму взаємозв'язку сутностей бази даних, яка відображає зв'язки між усіма сутностями. Діаграма чітко ілюструє зв'язки багато-до-багатьох, один-до-багатьох та ключові атрибути кожної сутності. Ця діаграма зображена на рисунку 2.4.

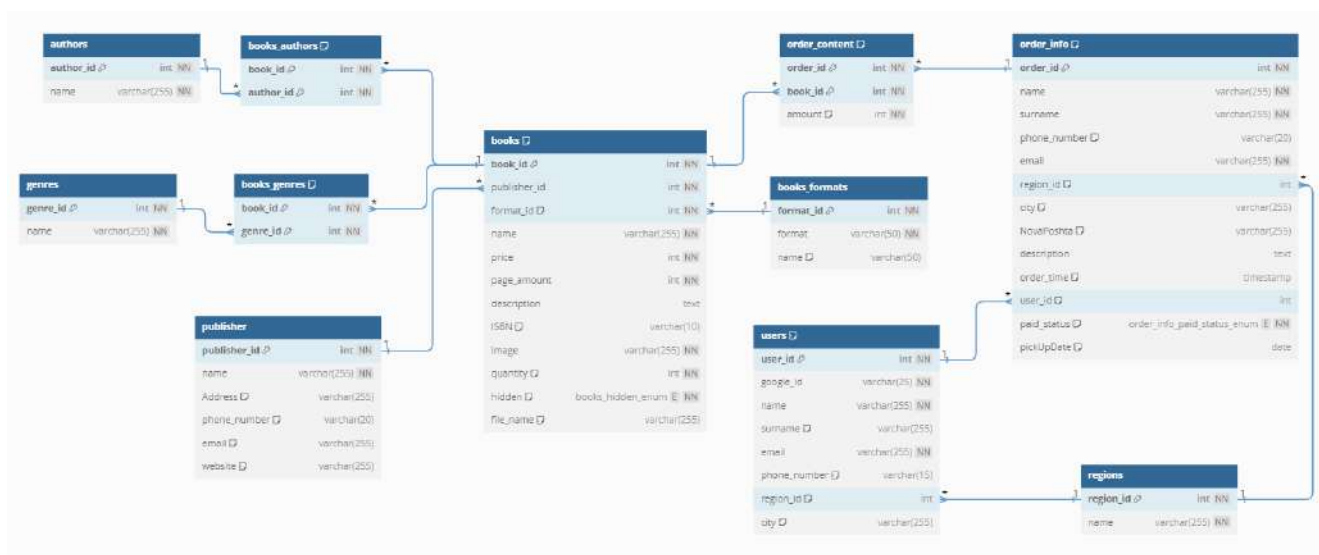


Рисунок 2.4 – Діаграма взаємозв'язку сутностей бази даних

2.3 Діаграми послідовностей

Діаграми послідовностей демонструють порядок виконання кожної операції в системі, від початку до завершення. Ці графічні зображення відображають взаємодію між об'єктами системи та послідовність обміну повідомленнями між ними. Такий підхід дозволяє чітко уявити, як система реагує на зовнішні події та як вона взаємодіє з користувачами або іншими системами. В нашому випадку об'єктами системи виступають клієнт, веб-сайт, Node.js сервер, Google API та MySQL сервер. Надісланні повідомлення відображаються у вигляді горизонтальних стрілок від одного об'єкту системи до іншого.

Розглянемо процес оформлення замовлення клієнтом. Клієнт уводить інформацію про замовлення, натискаючи кнопку «Оформити замовлення» веб-сервер виконує валідацію полів уведених користувачем потім відправляє на Node.js запит який в свою чергу відправляє запити у базу даних та повертає отриманий результат. У даному варіанті передбачається що клієнт не авторизований детальна діаграма послідовностей цього процесу зображена на рисунку 2.5.

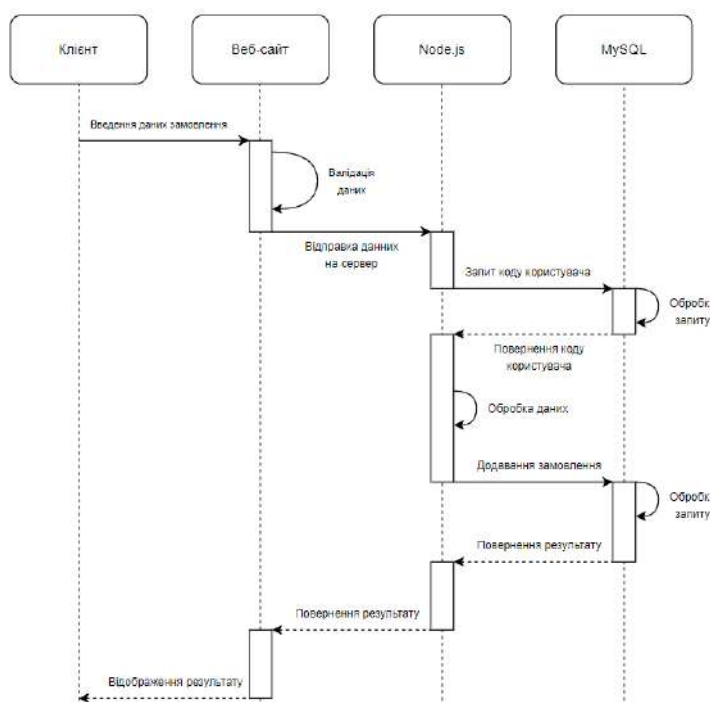


Рисунок 2.5 – Діаграма послідовностей прецеденту «Оформлення замовлення»

Авторизація клієнта відбувається з використанням service Google. У даному варіанті передбачається що клієнт успішно проходить авторизацію та вже має зареєстрований аккаунт. При натисканні кнопки «Авторизуватися з Google» клієнта перенаправляється на сторінку авторизації Google, після проходження авторизації та надання дозволу на використання сайтом даних Google повертає на сервер інформацію про успішну авторизацію. Клієнт перенаправляється на сторінку користувача, веб-сайтом виконується запит до серверу на отримання даних клієнта. Сервер перевіряє чи клієнт авторизований, запитує у бази даних його дані та повертає їх веб-сайту, дані відображаються клієнту. Діаграма послідовностей цього процесу зображена на рисунку 2.6.

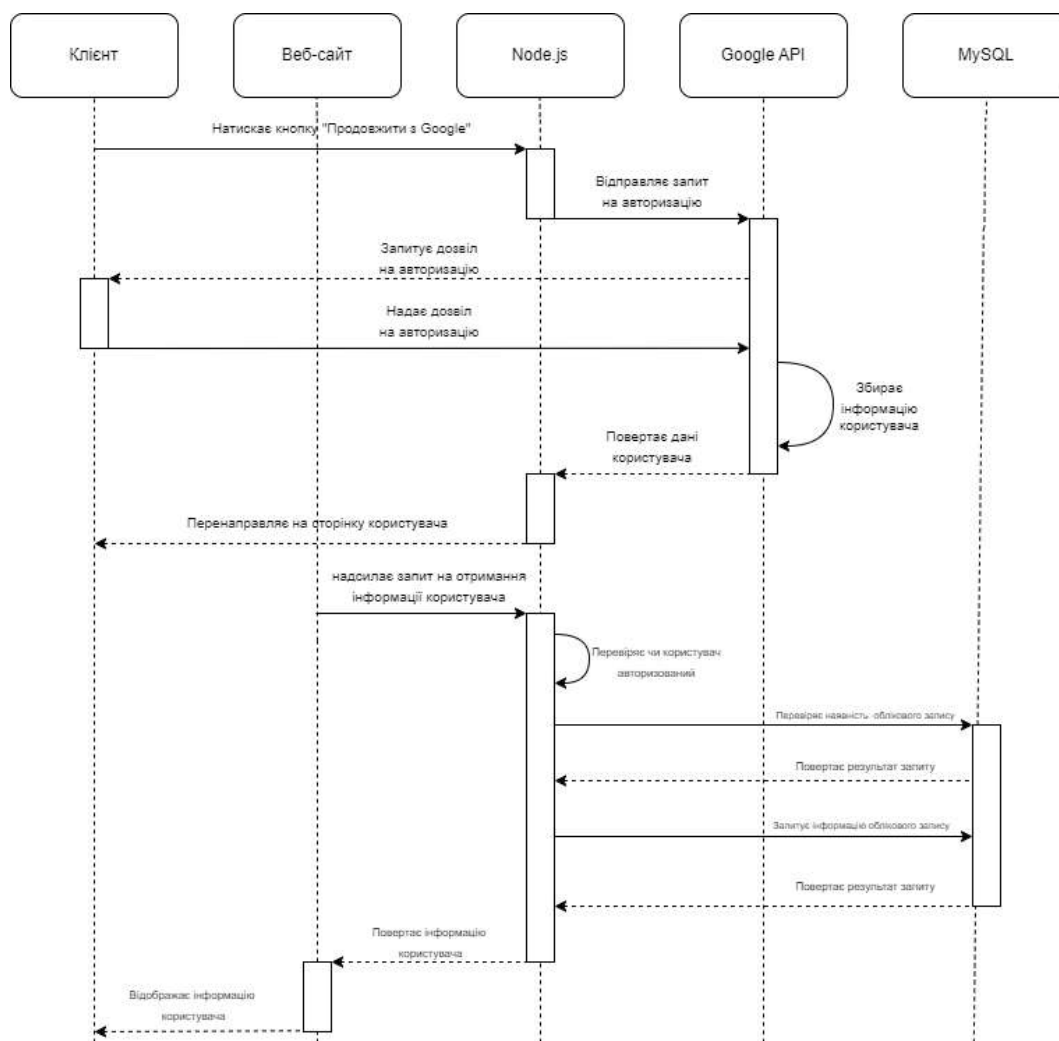


Рисунок 2.6 – Діаграма послідовностей прецеденту «Авторизація користувача»

3 РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ

3.1 Розробка серверної частини

Визначивши вимоги до системи та спроектувавши базу даних потрібно почати розробку системи. Розробка системи починається з розробки серверної частини, яка відповідає за логіку роботи додатка, обробку запитів від клієнтів та взаємодію з базою даних.

3.1.1 Початкова ініціалізація серверу

Розробка серверної частини починається з налаштування базових елементів сервера, таких як створення з'єднання з базою даних, налаштування підпрограмного забезпечення для обробки запитів та запуск сервера. У нашому прикладі використовується Node.js з фреймворком Express та базою даних MySQL. У лістингу 3.1 наведено процес ініціалізації сервера, а саме ініціалізація Express застосунку, визначення порту на якому буде працювати сервер. Після цього створюється підключення до бази даних, `app.listen()` встановлює прослуховування на визначеному порту.

Лістинг 3.1 – Ініціалізація сервера

```
const app = express();
const port = 2210;

var connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: '',
  database: 'owl-books'
});
app.use(cors({origin: ['http://localhost:5500',
'http://127.0.0.1:5500']})));
app.use(bodyParser.json());
app.listen(port, function() {
  console.log('Server is up and running!');
});
```

3.1.2 Забезпечення захисту бази даних

Під час роботи з базою даних потрібно забезпечити захист від потенційних атак типу SQL-ін'єкцій. Для цього запити до бази даних виконуються у параметризованому вигляді, коли замість того щоб в запит вставляти напряду змінну користувача використовується символ '?' як заповнювач для змінних, які потім передаються до запиту у вигляді списку параметрів. Це дозволяє базі даних правильно екранувати змінні та уникнути ін'єкцій. Приклад такого запиту наведено у лістингу 3.2

Лістинг 3.2 – Запит для отримання назв параметрів

```
var paramsQuery = `SELECT (SELECT \`name\` FROM books_formats
WHERE format_id = ?) AS book_format, (SELECT \`name\` FROM genres
WHERE genre_id = ?) AS genre_name, (SELECT \`name\` FROM authors
WHERE author_id = ?) AS author_name, (SELECT \`name\` FROM publisher
WHERE publisher_id = ?) AS publisher_name`;
(await connection).query(paramsQuery, [book_type, genre,
author, publisher])
```

3.1.3 Налаштування авторизації користувача

Для авторизації користувача за допомогою сторонніх сервісів використовується бібліотека passport з стратегією passport-google-oauth2. Для ініціалізації passport стратегією Google потрібно вказати «код клієнта», «секрет клієнта» та вказати адресу зворотного виклику, яка використовується для повернення результатів запитів процес цієї ініціалізації наведено у лістингу 3.3.

Лістинг 3.3 – Ініціалізація passport стратегією Google

```
passport.use(new GoogleStrategy(
  {
    clientID: GOOGLE_CLIENT_ID,
    clientSecret: GOOGLE_CLIENT_SECRET,
    callbackURL: "http://localhost:2210/auth/google/callback",
    passReqToCallback: true,
  }, function(request, accessToken, refreshToken, profile, done){
    return done(null, profile);
  })
);
```

Так як авторизація користувача виконується за допомогою сервіса Google то для системи потрібно створити OAuth 2.0 запис, у якому потрібно вказати назву клієнту, прописати адреси з яких користувач може виконувати запити на авторизацію та вказати адресу на яку дозволено перенаправляти користувача після вдалої авторизації. Після створення та налаштування сторінки застосунку Google надає «код клієнта» та «секрет клієнта» що використовуються при налаштуванні підключень. На рисунку 3.1 зображена сторінка застосунку у сервісі Google.

The screenshot shows the 'Client ID for Web application' page in the Google Cloud Console. The page is titled 'Client ID for Web application' with a 'DELETE' button. It contains several sections:

- Name:** A text field containing 'OwlBooks'. Below it, a note states: 'The name of your OAuth 2.0 client. This name is only used to identify the client in the console and will not be shown to end users.'
- Additional information:** A section containing 'Client ID' (a long alphanumeric string), 'Creation date' (May 4, 2024 at 8:09:02 AM GMT+3), and 'Client secrets' (a long alphanumeric string). Below the secrets, there is a note about rotating secrets and a link to 'Learn more'. A status indicator shows 'Enabled'.
- Authorized JavaScript origins:** A section for 'For use with requests from a browser'. It contains two text fields for 'URIs 1' and 'URIs 2', both containing 'http://localhost:5500'. There is an '+ ADD URI' button.
- Authorized redirect URIs:** A section for 'For use with requests from a web server'. It contains one text field for 'URIs 1' containing 'http://localhost:2210/auth/google/callback'.

Рисунок 3.1 – Сторінка веб-застосунку у сервісі Google

Для налаштування авторизації на сервері потрібно у проміжне програмне забезпечення express передати сесію з налаштуваннями, ініціалізацію passport та налаштовану сесію passport. Також було створено функцію `isLoggedIn()`, що перевіряє чи користувач авторизувався і відповідно від результату пропускає його далі або повертає на сторінку авторизації. Процес налаштування сервера для авторизації користувача наведено у лістингу 3.4.

Лістинг 3.4 – Налаштування авторизації на сервері

```
app.use(session({
  secret: 'cats',
  resave: false,
  saveUninitialized: true
}));
```

```

app.use(passport.initialize());
app.use(passport.session());

function isLoggedIn(req, res, next) {
  res.setHeader('Access-Control-Allow-Credentials', true);
  req.user ? next() : res.status(401).send( '<script>
window.location.href = "http://localhost:5500/user.html"
</script>');
}

```

Для авторизації було налаштовано маршрут через який користувач буде її виконувати та маршрут зворотного виклику який в залежності від успіху авторизації перенаправляє користувача на інший маршрут, що наведено у лістингу 3.5.

Лістинг 3.5 – Маршрути авторизації та зворотного виклику

```

app.get('/auth/google',
  passport.authenticate('google',{ scope: [ 'email', 'profile']})
);
app.get('/auth/google/callback',
  passport.authenticate('google', {
    successRedirect: '/close-window',
    failureRedirect: '/auth/google/failure'
  })
);

```

Після налаштування процесу авторизації потрібно впевнитись що користувач має можливість вийти зі свого аккаунту. Для цього було створено маршрут, що знищує поточну сесію користувача, деавторизуючи його. Цей маршрут наведено у лістингу 3.6.

Лістинг 3.6 – Шлях деавторизації користувача

```

app.get('/logout', (req, res) => {
  res.setHeader('Access-Control-Allow-Credentials', true);
  req.logout(err => { if (err) { return next(err); }
  req.session.destroy(() => { res.send('Goodbye!'); });
});
});

```

3.1.3 Додавання загальних маршрутів

Далі було створено маршрут для отримання інформації про користувача з бази даних. Для цього використовується раніше визначена функція `isLoggedIn`, щоб впевнитися, що користувач авторизований. Після цього виконується функція `createNewUser`, імплементація котрої наведена у лістингу А.1, яка виконує кілька запитів до бази даних. Спочатку перевіряється, чи даний користувач вже є в базі даних; якщо ні, виконується запит на створення нового запису. Наприкінці виконується запит на отримання усіх даних користувача, які йому повертаються. Код створення цього маршруту наведено у лістингу 3.7.

Лістинг 3.7 – Маршрут отримання інформації про користувача

```
app.get('/protected', isLoggedIn, async (req, res) => {
  try {
    let info = await createNewUser(req);
    res.status(200).json(info);
  } catch (error) {
    res.status(500).send('Error retrieving data from
database');
  });
});
```

Задля отримання користувачем інформації про товари у його кошику створено шлях `cart`. У нього користувач передає список товарів які у нього додані до кошику та отримує інформацію про них. Код цього шляху наведено у лістингу 3.8.

Лістинг 3.8 – Маршрут отримання інформації про товари у кошику

```
app.get('/cart', async (req, res) => { try {
  const { ids } = req.query;
  let search = (typeof ids=='undefined' || ids=='') ? "-1" : ids;
  var cartQuery = `SELECT books.format_id, books.book_id,
books.image, books.\`name\`, books.price, ( SELECT authors.\`name\`
FROM authors JOIN books_authors ON authors.author_id =
books_authors.author_id WHERE books_authors.book_id = books.book_id
LIMIT 1) AS author_name FROM books WHERE books.book_id IN (?);`;
  (await connection).query(cartQuery, [search])
  .then((result) => { res.send(result[0]); });
} catch (error) {
  res.status(500).send('Error retrieving data from database');
});
});
```

Також були розроблені маршрути для пошуку товарів, додавання нового замовлення, маршрут для оновлення інформації про користувача у базі даних, маршрут для отримання списку усіх куплених коли небудь користувачем товарів та маршрути для скачування користувачем електронних та аудіо книг. Далі наведено їх детальний опис.

Маршрут для пошуку товарів приймає до шести параметрів: пошуковий запит, тип книги, жанр, автор, видавництво та режим сортування. Потім в залежності від наданих користувачем параметрів створюється потрібний запит та масив відповідних параметрів що надсилаються у базу даних. Отримавши відповідь, сервер надсилає інформацію користувачу. Ця інформація включає в себе код товару, його кількість на складі, назву товару, ціну, зображення та стан товару, прихований він чи ні. Імплементация цього маршруту наведена у лістингу А.2.

Маршрут додавання нового замовлення при надходженні запиту перевіряє чи користувач зареєстрований та додає до запису замовлення його ідентифікатор якщо так. Імплементацию цього процесу можна побачити у лістингу 3.9.

Лістинг 3.9 – Частина коду маршруту додавання нового замовлення

```
const [user] = await (await connection).query(`SELECT user_id
FROM users WHERE google_id = ?;`, [google_id]);
let user_id = user.length > 0 ? user[0].user_id : undefined;
let addOrderQuery = `INSERT INTO order_info (`name`, surname,
phone_number, email, region_id, city, NovaPoshta, pickUpDate,
`description`${user_id !== undefined ? ', user_id' : ''}) VALUES
(?, ?, ?, ?, ?, ?, ?, ?, ?`${user_id !== undefined ? ', ?' : ''});`;
let queryParams = [name, surname, phone_number, email,
region_id, city, NovaPoshta, pickUpDate, description];
if (user_id !== undefined) {queryParams.push(user_id);}
await (await connection).beginTransaction();
const [result] = await (await connection).query(addOrderQuery,
queryParams);
orderId = result.insertId;
```

З метою завершення процесу додавання нового замовлення, після створення запису про замовлення, необхідно також додати записи до таблиці order_content, яка зберігає інформацію про товари та їх кількість в цьому замовленні. Після отримання ідентифікатора нового замовлення orderId, виконується вставка даних

про кожну книгу в замовленні до таблиці `order_content`. Для цього створюється масив `orderContentQueries`, де кожен елемент масиву відповідає за виконання запиту до бази даних для вставки інформації про окремий товар в замовленні. Метод `Promise.all` використовується для одночасного виконання всіх запитів, що забезпечує ефективність та паралельність операцій. Увесь процес додавання нового замовлення до бази даних виконано після методу `beginTransaction()`, який розпочинає транзакцію, дозволяючи у випадку виникнення проблем скористатися методом `rollback()` для відміни цих змін та методом `commit()` для підтвердження змін у випадку коли ніяких помилок не сталося під час виконання запитів. Код шляху для додавання нового замовлення наведено у лістингу А.3.

Користувачам надана можливість оновлювати дані свого облікового запису за допомогою відправки запитів на маршрут `save-user`. До цього маршруту додано метод-обробник `isLoggedIn` щоб переконатися що тільки авторизовані користувачі можуть змінити свої дані. Після перевірки авторизації користувача та збору наданих ним параметрів виконується формування запиту `UPDATE` до бази даних який виконує оновлення даних.

Далі було додано новий маршрут `books` для отримання користувачем списку всіх куплених ним книг у певному форматі. Цей маршрут реалізовано у лістингу 3.10.

Лістинг 3.10 – Маршрут отримання придбаних користувачем книг

```
app.get('/books', isLoggedIn, async (req, res) => {
  const google_id = req.user.sub;
  const { format_id } = req.query;
  var booksQuery;

  if (!format_id) { return res.status(400).send('Missing
required query parameters'); }

  try {
    booksQuery = `SELECT book_id FROM books WHERE format_id = ?
AND book_id IN ( SELECT book_id FROM order_content WHERE order_id IN
( SELECT order_id FROM order_info WHERE user_id = ( SELECT user_id
FROM users WHERE google_id = ? ) AND paid_status = 'Yes' ) );`

    const bookIds = await (await connection).query(booksQuery,
[format_id, google_id]);
```

```

    res.send(bookIds[0]);
  } catch (error) {
    console.error(`books: error: ${error}`);
    res.status(500).send('Error retrieving data from database');
  });
};

```

Цей маршрут реалізовано з використанням функції `isLoggedIn` для перевірки авторизації користувача. Спочатку з запиту отримується параметр `format_id`, який вказує на формат книг. Якщо параметр відсутній, повертається помилка 400.

Основна частина маршруту складається з SQL-запиту, який вибирає книги певного формату, куплені авторизованим користувачем. Запит використовує вкладені підзапити для пошуку книг, що належать до замовлень цього користувача, де статус оплати є "Yes". Цей маршрут дозволяє користувачам зручно отримувати список всіх куплених ними книг у заданому форматі.

Щоб користувачі могли завантажувати файли придбаних книг, створено маршрут для електронних та аудіо книг. Обидва маршрути використовують метод `isLoggedIn` для перевірки авторизації користувача. Реалізацію цих маршрутів наведено у лістингу 3.11.

Лістинг 3.11 – Маршрути отримання користувачем файлів придбаних книг

```

app.get('/mp3', isLoggedIn, async (req, res) => {
  getFile(req, res, './audio_books/')
});

app.get('/pdf', isLoggedIn, async (req, res) => {
  getFile(req, res, './electronic_books/')
});

```

Наведені маршрути викликають функцію `getFile` передаючи до неї об'єкти `req` та `res` для зв'язку з клієнтом та шлях до папки з файлами книг відповідного типу. Функція `getFile` обробляє запит на отримання файлу, перевіряючи, чи користувач дійсно володіє запитаною книгою. Код цієї функції наведено у лістингу 3.12.

Лістинг 3.12 - Функція для отримання файлу книги

```

async function getFile(req, res, folder_path) {
  try {

```

```

    var fileQuery;
    const google_id = req.user.sub;
    const { book_id } = req.query;
    if (!book_id) { return res.status(400).send('Missing required
query parameters'); }
    fileQuery = ` SELECT EXISTS ( SELECT 1 FROM order_info WHERE
order_id IN ( SELECT order_id FROM order_content WHERE book_id = ? )
AND user_id = ( SELECT user_id FROM users WHERE google_id = ? ) AND
paid_status = 'Yes' ) AS isNotNULL`;
    const isOwned = await (await connection).query(fileQuery,
[book_id, google_id]);
    if(isOwned[0][0].isNotNULL != 1) {
        return res.status(401).send("You don't own this book");
    }
    fileQuery = `SELECT file_name FROM books WHERE book_id = ?`;
    var file_name = await (await connection).query(fileQuery,
[book_id])
    file_name = file_name[0][0].file_name;
    const filePath = path.join( __dirname,
`${folder_path}${file_name}` );
    if(fs.existsSync(filePath)) {
        res.sendFile(filePath);
    } else {
        return res.status(404).send('Book not found');
    }
} catch (error) {
    console.error("getFile error: " + error);
    return res.status(500).send('Server error');
}}

```

Функція `getFile` спочатку перевіряє наявність необхідного параметра `book_id` у запиті. Якщо параметр відсутній, повертається помилка 400. Потім виконується SQL-запит для перевірки, чи придбав користувач книгу. Запит використовує параметри `book_id` та `google_id` для пошуку відповідного запису в базі даних. Якщо книга не належить користувачу, повертається помилка 401. Далі виконується ще один SQL-запит для отримання назви файлу книги з бази даних. Отримана назва файлу використовується для формування шляху до файлу на сервері. Якщо файл існує, він надсилається користувачу. Якщо файл не знайдено, повертається помилка 404. Ця функція дозволяє надійно перевіряти права доступу користувачів до файлів книг і забезпечує можливість завантаження файлів лише для тих користувачів, які дійсно володіють відповідними книгами.

3.2 Розробка клієнтської частини

Після завершення розробки серверної частини переходимо до опису клієнтської частини системи. Клієнтська частина є інтерфейсом, з яким взаємодіють користувачі, і включає в себе реалізацію всіх необхідних функцій для зручного її використання. Розробка клієнтської частини охоплює створення інтерфейсу користувача, обробку подій, взаємодію з сервером та інші аспекти, що забезпечують цілісне функціонування системи.

Клієнтська частина складається з п'яти сторінок: головно сторінка з товаром, сторінка пошуку, сторінка з детальним описом товару, обліковий запис користувача та сторінка оформлення замовлення.

Кожна з сторінок формується динамічно коли користувач її відриває. Так як сторінки мають багато спільних елементів то вони замінюються на пусті блоки що заповнюються при відкритті. Це дозволяє вносити зміни в повторювані елементи одразу на усіх сторінках де вони використовуються.

Усі сторінки мають такі повторювані елементи, що замінені на пусті блоки, як головне тіло (body), шапка (header), підвал (footer) та кошик. При завантаженні будь якої зі сторінок викликається метод `buildPage` який знаходить пусті елементи які є заглушками та наповнює їх зміст.

Шапка на кожній сторінці містить поле для пошуку, логотип магазину та кнопку облікового запису. Логотип магазину представляє з себе посилання на головну сторінку.

Відкрита користувачем головна сторінка викликає функцію `getAllData()` для наповнення сторінки. Ця функція надсилає на сервер `fetch` запит по маршруту пошуку, проте не вказуючи пошуковий запит через що сервер повертає клієнту список усіх неприхованих товарів, після чого вони додаються до сторінки після тексту «Наші Книжки», попередньо завантаживши з сервера інформацію про назви, ціни, інформація про наявність та зображення кожного з товарів.

Поряд з ціною товару розміщена кнопка «Купити» що додає обраний товар у кошик та відображає користувачу невелике тимчасове повідомлення про те що товар було додано до кошику. Якщо кількість товару що повертає сервер менше однієї одиниці то замість кнопки «Купити» буде відображено червоний напис «Немає в наявності». Приклад обох випадків зображено на рисунку 3.2.



Рисунок 3.2 – Товари з різним статусом наявності

Додані до кошику товари зберігаються в локальному сховищі браузера у вигляді JSON об'єкта що складається з пар код книги та її кількість. При завантаженні сторінки та при будь яких змінах у товарах кошику він оновлюється. Кошик відображає усі додані клієнтом до нього товари, їх назву, ціну, автора та кількість. Також біля кожного товару у кошику є кнопка видалити щоб користувач міг прибрати його. Угорі кошику розміщена кнопка його закриття, написана кількість унікальних товарів та розміщена кнопка «Очистити кошик». Поряд з кількістю товару розміщено дві кнопки для збільшення та зменшення його кількості у кошику. При змінні кількості товару у кошику шляхом додавання нового, натискання кнопок регулювання кількості або вписавши в поле кількості нове значення, перед тим як приймати його зміну клієнт надсилає запит до сервера для отримання інформації про доступну кількість товару. Якщо число яке клієнт намагається увести є більшим за наявну то йому виводиться про це повідомлення

та значення кількості не змінюється. Електронні та аудіо книги у кошику не мають поля для зміни кількості. Знизу кошика знаходиться загальна сума замовлення, яка оновлюється при будь-яких змінах кошику, та кнопка «Перейти до оформлення замовлення», ці елементи не відображаються користувачу якщо кошик порожній. Приклад вигляду наповненого кошику зображено на рисунку 3.3.

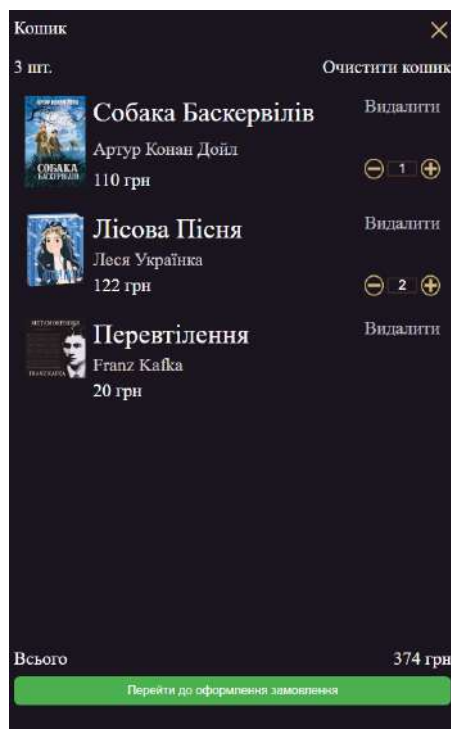


Рисунок 3.3 – Наповнений товарами кошик

Одразу під шапкою на більшості сторінок розміщуються навігаційні елементи для переходу у розділи з друкованими, електронними або аудіо книгами, кнопка кошику та вибір режиму сортування.

Кожен товар будь де на сайті є посиланням на сторінку з його детальним описом. Для того аби не створювати окрему сторінку під кожен товар це досягається шляхом додавання в адресу сторінки ідентифікатора. При завантаженні такої сторінки JS скрипт надсилає на сервер запит надаючи цей ідентифікатор і отримавши детальний опис наповнює сторінку. На сторінці наявне зображення товару, його назва, перелік авторів, жанрів, кількість сторінок, видавництво, ISBN код, доступна кількість товару та його детальний опис, назва сторінки у браузері

змінюється на назву відкритого товару. Якщо книга продається у електронному або аудіо варіанті то поле з кількістю доступного товару відсутнє. Для аудіо книг поле кількість сторінок замінюється на її довжину. У правій частині сторінки розміщена інформація про ціну товару, поле для введення бажаної кількості та кнопка «Купити». Вигляд сторінки товару зображено на рисунку 3.4.



Рисунок 3.4 – Сторінка товару

На сторінці товару користувач може натиснути на ім'я автора, на один з жанрів або на назву видавництва. Що перенаправить користувача на сторінку пошуку саме по цьому автору, жанру або видавництву на яке натиснув користувач.

При завантаженні користувачем сторінки облікового запису виконується перевірка на те чи авторизований він. Якщо користувач не авторизований замість сторінки облікового запису йому буде відображене повідомлення про не авторизацію та буде надана кнопка для авторизації. Вигляд неавторизованої сторінки користувача зображено на рисунку 3.5.

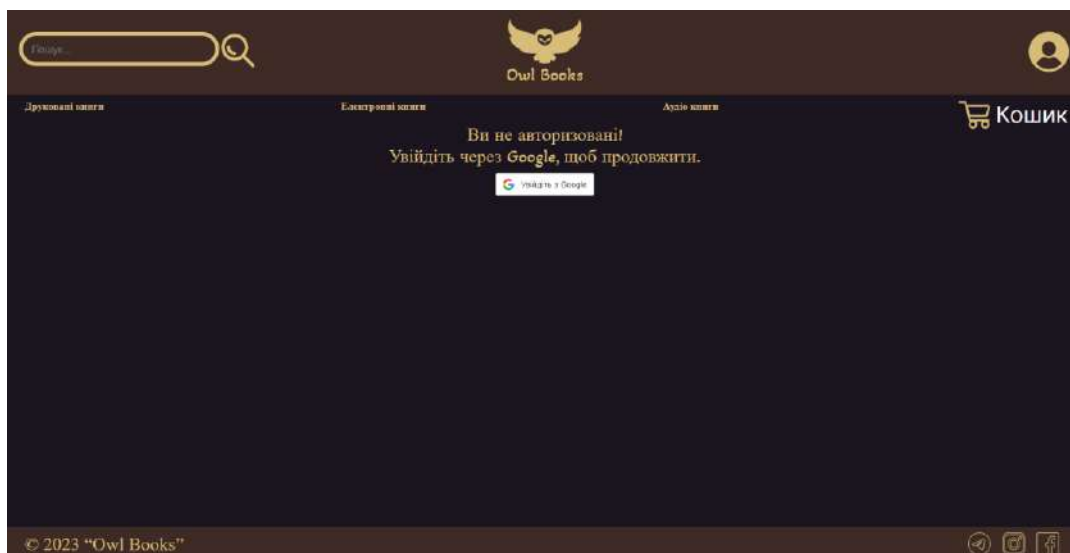


Рисунок 3.5 – Сторінка неавторизованого користувача

При натисканні на цю кнопку користувача перенаправляє на сторінку підтвердження авторизації. Після проходження якої або у випадку коли користувач вже був авторизований завантажується сторінка облікового запису. Вона складається з зображення користувача, його інформації що зберігається на сервері, кнопок «Зберегти зміни» та «Вийти з аккаунту» та списку придбаних користувачем електронних та аудіо книг. При першій авторизації єдина інформація яка буде відображатись на сторінці користувача буде його ім'я та прізвище що вказанні в його обліковому записі Google та електронна адреса. Користувач може змінити інформацію що вказана на його сторінці та натиснувши кнопку «Зберегти зміни» на сервер відправиться запит на оновлення інформації користувача. Також якщо авторизований користувач купував електронній або аудіо книги вони будуть відображатись на сторінці його облікового запису поряд з двома кнопками «Читати» («Слухати» для аудіо книг) та «Скачати» що дозволяє користувачу не виходячи з сайту прочитати придбану книги або скачати її. Вигляд сторінки користувача зображено на рисунку 3.6.

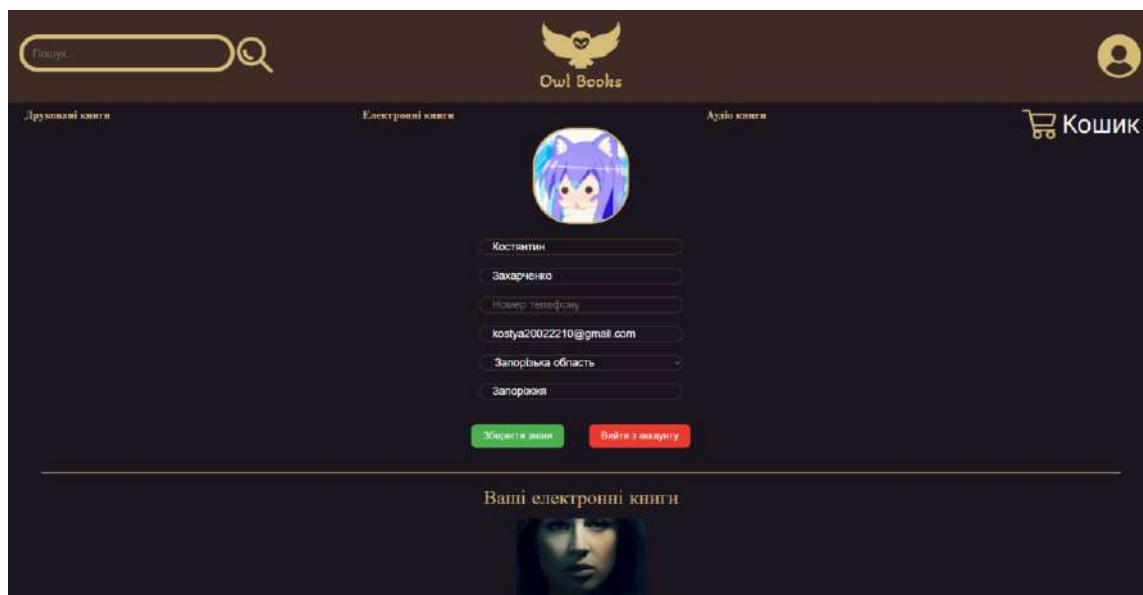


Рисунок 3.6 – Сторінка авторизованого користувача

Сторінка пошуку при відкритті завантажує з сервера список книг та їх деталі так само як і головна сторінка. Проте, на відміну від головної сторінки, сторінка пошуку при завантаженні інформації враховує в запиті параметри вказані у адресі сторінки. Параметр *s* відповідає за назву, *t* за тип книги, *g* за жанр, *a* – автор, *p* – видавець та *o* що відповідає за сортування. В залежності від того які параметри вказані в адресі змінюється підпис сторінки угорі. Код функції що відповідає за наповнення сторінки пошуку наведено у лістингу А.4.

Сторінка оформлення замовлення відкривається за допомогою кнопки у кошику при наявності у ньому товарів. Сторінка складається з двох частин: наповнення кошику та інформація про замовлення. Наповнення кошику за функціоналом не відрізняється від кошику. Інформація про замовлення містить поля ім'я, прізвище, телефон, електронна пошта, область місто, відділення пошти та коментар. Також там наявний прапорець для вибору користувачем зручної йому дати замовлення. Вибір дати замовлення обмежено таким чином щоб користувач зміг обрати мінімум три дні наперед але не більше місяця наперед. Це виконується за допомогою функції код якої наведено у лістингу 3.13.

Лістинг 3.13 – Функція налаштування меж вибору дати

```
function setDateInputBounds() {
    const now = new Date();
    const minDate = new Date(now);
    minDate.setDate(now.getDate() + 3);
    const maxDate = new Date(now);
    maxDate.setDate(now.getDate() + 31);
    const formatDate = date => date.toISOString().split('T')[0];
    dateInput.min = formatDate(minDate);
    dateInput.max = formatDate(maxDate);
}
```

У випадку коли в кошику користувача з товарів наявні лише електронні або аудіо книги то блок інформації про замовлення змінюється таким чином що залишаються лише поля ім'я, прізвище та електронна пошта. Якщо цю сторінку відкриває авторизований користувач то його дані запитуються з сервера та автоматично заповнюються поля. Сторінка оформлення замовлення зображена на рисунку 3.7.

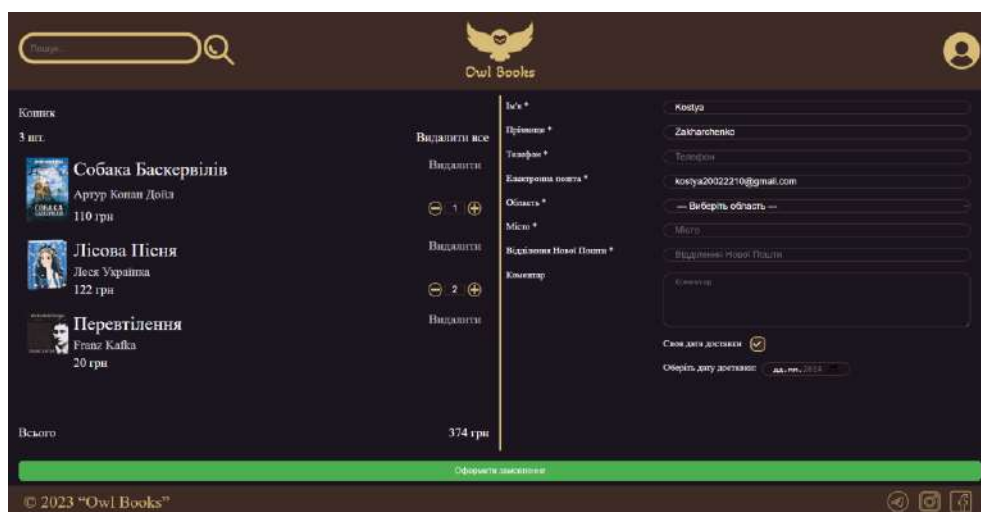


Рисунок 3.7 – Сторінка оформлення замовлення

При натисканні кнопки «Оформити замовлення» виконується перевірка обов'язкових полів, якщо поле що перевіряється не проходить перевірку до нього додається клас `error` що підсвічує це поле червоним щоб користувачу було зрозуміло куди треба звернути увагу. Коли помилок при перевірці обов'язкових полів не було виявлено відправляється запит на сервер з вмістом кошику. Якщо

сервер повертає код 200 (успіх) то кошик користувача очищується та виводиться повідомлення про успішне замовлення. У випадку коли сервер повертає повідомлення про помилку то користувачу виводиться повідомлення про помилку та кошик не очищується. Результат успішного замовлення зображено на рисунку 3.8.

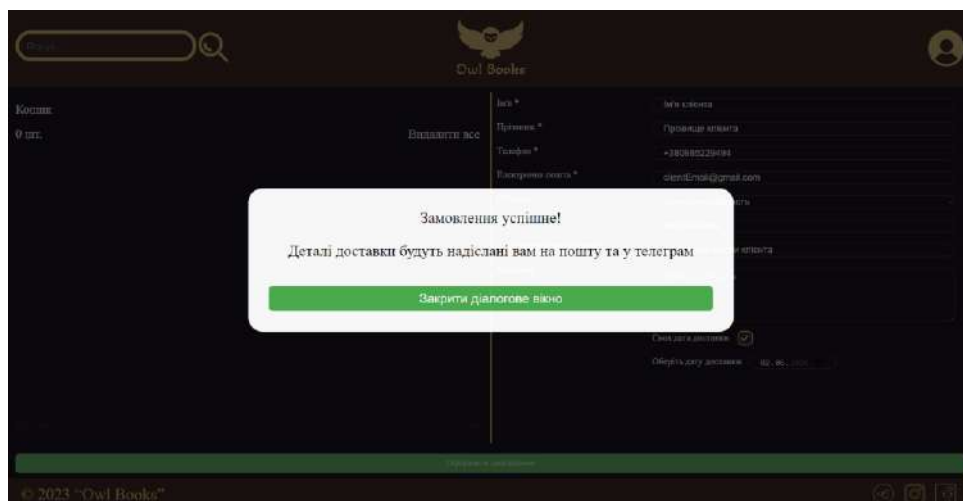


Рисунок 3.8 – Повідомлення про успішне замовлення

ВИСНОВКИ

В результатів виконання дипломної роботи була розроблена комп'ютерна система для інтернет книгарні що надає користувачам можливість шукати книжки, отримувати їх опис, входити в обліковий запис за допомогою сервіса Google, переглядати куплені електронні та аудіо книги та оформлювати замовлення. Головною метою проєкту було створити систему, яка дозволяє користувачам шукати та замовляти книжки у інтернет-магазині книгарні.

Для досягнення цієї мети були використані різні технології та інструменти розробки. Мовою програмування було обрано JavaScript, що забезпечило широкі можливості для реалізації функціональності системи. Графічний інтерфейс клієнта було реалізовано за допомогою набору технологій HTML, CSS та JavaScript, а зв'язок клієнта з сервером було реалізовано через fetch запити зі сторони клієнта та фреймворк Node.js з бібліотекою Express на серверній стороні. В процесі розробки було спроектовано та реалізовано базу даних MySQL. Що дозволило забезпечити ефективне зберігання та доступ до даних, необхідних для функціонування системи.

Результатом роботи стала функціональна система, яка забезпечує користувачам зручний та інтуїтивно зрозумілий інтерфейс для пошуку та купівлі книжок в інтернет-книгарні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Як попри повномасштабну війну працювали українські інтернет-магазини у 2023 році. Асоціація ритейлерів України – The profile association of retail market players. URL: <https://rau.ua/novyni/internet-magazini-u-2023-roci/> (дата звернення: 30.04.2024).
2. Архітектура застосунку: компоненти та популярні патерни. FoxmindEd. URL: <https://foxminded.ua/arkhitektura-zastosunku/> (дата звернення: 01.05.2024).
3. Serverless architecture: what it is & how it works | datadog. Datadog. URL: <https://shorturl.at/mFQZ8> (дата звернення: 30.04.2024).
4. What is java? - java programming language explained - AWS. Amazon Web Services, Inc. URL: <https://aws.amazon.com/what-is/java> (дата звернення: 30.04.2024).
5. Що таке Ruby? Де використовується ця мова програмування. lemon.school. URL: <https://lemon.school/blog/navishho-potribna-mova-programuvannya-ruby> (дата звернення: 30.04.2024).
6. Introduction to JavaScript - GeeksforGeeks. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/introduction-to-javascript/> (дата звернення: 01.05.2024).
7. Що таке CSS. Український веб-довідник. URL: https://css.in.ua/article/shcho-take-html_10 (дата звернення: 01.05.2024).
8. AngularJS. AngularJS. URL: <https://docs.angularjs.org/guide/introduction> (дата звернення: 01.05.2024).
9. Herbert D. What is react.js? Uses, examples, & more. HubSpot Blog | Marketing, Sales, Agency, and Customer Success Content. URL: <https://blog.hubspot.com/website/react-js> (дата звернення: 01.05.2024).
10. Vue.js. Vue.js - The Progressive JavaScript Framework | Vue.js. URL: <https://vuejs.org/guide/introduction> (дата звернення: 01.05.2024).

ДОДАТОК А

ОСНОВНІ КОДИ ПРОГРАМНИХ МОДУЛІВ

Лістинг А.1 – Функція отримання даних про користувача

```

async function createNewUser(req, res) {
  try {
    const google_id = req.user.id;
    const name = req.user.given_name;
    const surname = req.user.family_name;
    const email = req.user.email;
    var createNewUserQuery = `SELECT CASE WHEN EXISTS (SELECT 1
FROM users WHERE google_id = ?) THEN 'true' ELSE 'false' END AS
result;`;
    const userExistsResult = await (await
connection).query(createNewUserQuery, [google_id]);
    // if user is not in DB, create one
    if (userExistsResult && userExistsResult[0] &&
userExistsResult[0][0].result == 'false') {
      try {
        await (await connection).beginTransaction();
        createNewUserQuery = `INSERT INTO \owl-
books\.\`users\` (\`google_id\`, \`name\`, \`surname\`, \`email\`)
VALUES (?, ?, ?, ?);`;
        await (await connection).query(createNewUserQuery,
[google_id, name, surname, email]);
        (await connection).commit();
      } catch (error) {
        (await connection).rollback();
        console.error(`create-new-user: error: ${error}`);
        throw new Error('Error creating new user');
      }
    }
    createNewUserQuery = `SELECT * FROM users WHERE google_id =
?;`

    const userResult = await (await
connection).query(createNewUserQuery, [google_id]);
    const dbUser = userResult[0][0];
    const info = {
      google_id: google_id,
      picture: req.user.picture,
      name: dbUser.name,
      surname: dbUser.surname,
      email: dbUser.email,
      phone_number: dbUser.phone_number,
      region_id: dbUser.region_id,
      city: dbUser.city
    }
  }
}

```

```

    };
    console.log(info);
    return info;
  } catch (error) {
    console.error(`create-new-user: error: ${error}`);
    throw new Error('Error retrieving data from database');
  }
}

```

Лістинг A.2 – Маршрут пошуку книг за параметрами

```

app.get('/search', async (req, res) => {
  try {
    const {
      name,
      book_type,
      genre,
      author,
      publisher,
      orderby
    } = req.query;
    var params = [];
    let query = `SELECT DISTINCT b.book_id, b.quantity,
b.\`name\`, b.price, b.image, b.hidden FROM books b LEFT JOIN
books_authors ba ON b.book_id = ba.book_id LEFT JOIN books_genres bg
ON b.book_id = bg.book_id WHERE b.hidden = 'No' `;
    if(book_type) {
      query += ` AND b.format_id = ?`;
      params.push(book_type);
    }
    if (name) {
      query += ` AND b.\`name\` LIKE ?`;
      params.push(`%${name}%`);
    }
    if (publisher) {
      query += ` AND b.publisher_id = ?`;
      params.push(publisher);
    }
    if (author) {
      query += ` AND ba.author_id = ?`;
      params.push(author);
    }
    if (genre) {
      query += ` AND bg.genre_id = ?`;
      params.push(genre);
    }
    if (orderby === 'price_asc') {
      query += ` ORDER BY b.price ASC`;
    } else if (orderby === 'price_des') {
      query += ` ORDER BY b.price DESC`;
    } else if (orderby === 'alphab_asc') {
      query += ` ORDER BY b.\`name\` ASC`;
    } else if (orderby === 'alphab_des') {

```

```

        query += ` ORDER BY b.\`name\` DESC`;
    }
    (await connection).query(query, params)
    .then((result) => {
        res.send(result[0]);
    })
    .catch((error) => {
        console.error(error);
        res.status(500).send('Error retrieving data from
database');
    });
} catch (error) {
    console.error(error);
    res.status(500).send('Error retrieving data from
database');
}});

```

Лістинг А.3 – Маршрут додавання нового замовлення

```

app.get('/add-order', async (req, res) => {
    try {
        const {
            google_id, name, surname, phone_number, email, region_id,
            city, NovaPoshta, pickUpDate, description, content } = req.query;
        const parsedContent = JSON.parse(content);
        const bookIds = parsedContent.map(item => item.id);
        const amounts = parsedContent.map(item => item.quantity);
        let orderId;
        const [user] = await (await connection).query(`SELECT
user_id FROM users WHERE google_id = ?;`, [google_id]);
        let user_id = user.length > 0 ? user[0].user_id :
undefined;
        // Construct the base query
        let addOrderQuery = ` INSERT INTO order_info (\`name\`,
surname, phone_number, email, region_id, city, NovaPoshta,
pickUpDate, \`description\`${user_id !== undefined ? ', user_id' :
''}) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?${user_id !== undefined ? ',
?' : ''});`;
        // Query parameters
        let queryParams = [name, surname, phone_number, email,
region_id, city, NovaPoshta, pickUpDate, description];
        if (user_id !== undefined) {
            queryParams.push(user_id);
        }
        // Begin transaction
        await (await connection).beginTransaction();
        // Execute the query
        const [result] = await (await connection).query(
addOrderQuery, queryParams);
        orderId = result.insertId;
        // Insert into order_content table
        const orderContentQueries = bookIds.map(async (book_id, i)
=> ((await connection).query(`INSERT INTO order_content (order_id,
book_id, amount) VALUES (?, ?, ?);`, [orderId, book_id, amounts[i]]))

```

```

    ));
    await Promise.all(orderContentQueries);
    // Commit transaction
    await (await connection).commit();
    res.status(201).json({ orderId });
  } catch (error) {
    await (await connection).rollback();
    console.error(`add-order: error: ${error}`);
    res.status(500).send('Error processing your request');
  });
});

```

Лістинг А.4 – Функція побудови сторінки пошуку

```

import { buildRequestString } from "../input-check.js";

export async function fetchSearchData() {
  return new Promise(async (resolve) => {
    const searchParams = new URLSearchParams(
      window.location.search);
    const name = searchParams.get('s');
    const book_type = searchParams.get('t');
    const genre = searchParams.get('g');
    const author = searchParams.get('a');
    const publisher = searchParams.get('p');
    const orderby = searchParams.get('o');
    const tittle = document.querySelector('.tittle');
    var params = {book_type: book_type, genre: genre, author:
      author, publisher: publisher, }
    const paramsQuery =
      buildRequestString('http://localhost:2210/params', params)
    let paramsResponse = await fetch(paramsQuery);
    let paramsNames = await paramsResponse.json();
    if(name != null) {
      tittle.innerHTML += "\tНазва: " + name;
    }
    if(paramsNames.author_name != null) {
      tittle.innerHTML += "\tАвтор: " +
        paramsNames.author_name;
    }
    if(paramsNames.book_format != null) {
      tittle.innerHTML += "\tФормат: " +
        paramsNames.book_format;
    }
    if(paramsNames.publisher_name != null) {
      tittle.innerHTML += "\tВидавництво: " +
        paramsNames.publisher_name;
    }
    if(paramsNames.genre_name != null) {
      tittle.innerHTML += "\tЖанр: " + paramsNames.genre_name;
    }
    params.name = name;
    params.orderby = orderby;
    const query =
      buildRequestString('http://localhost:2210/search', params);

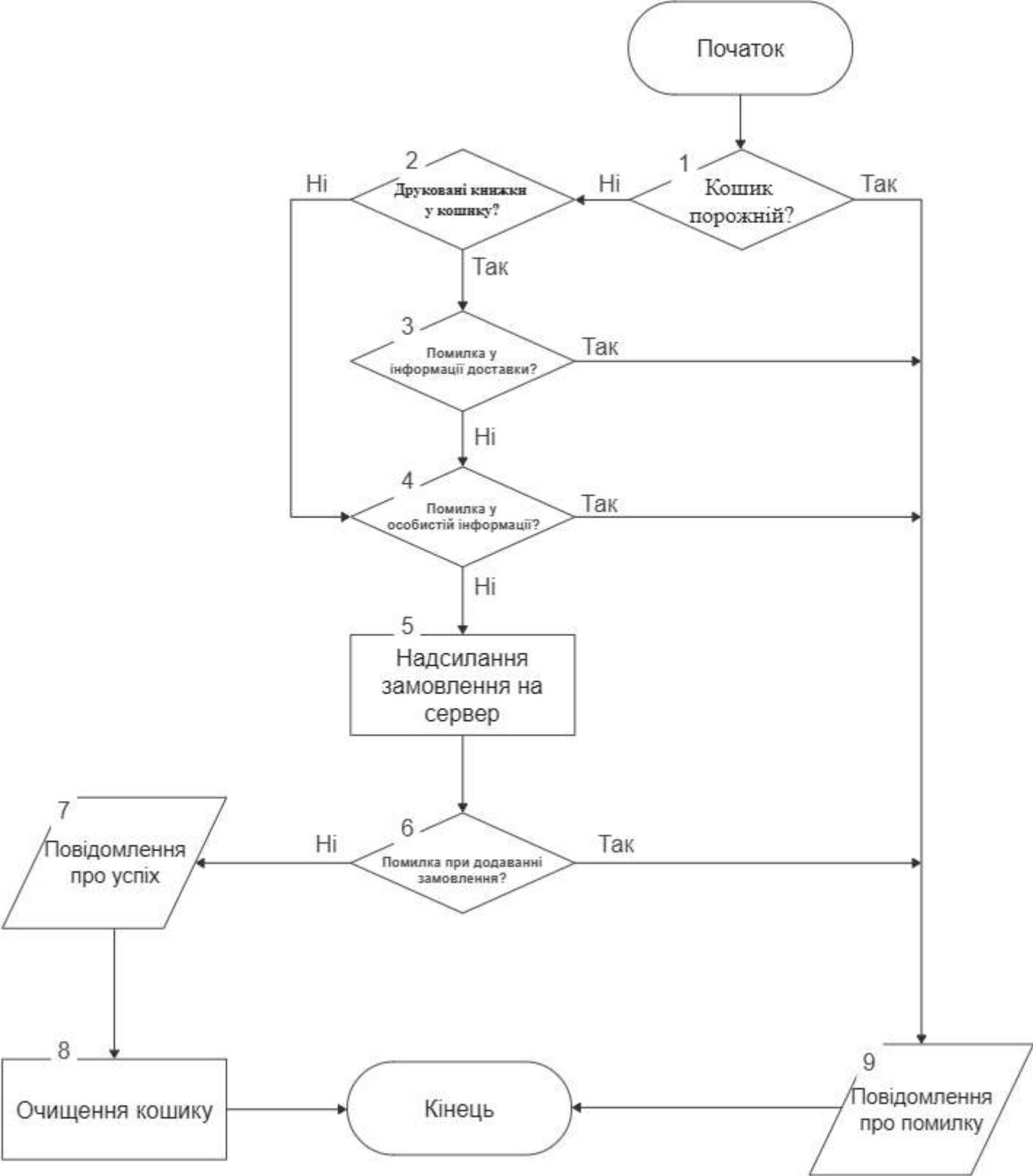
```

```

    console.log(query);
    let response = await fetch(query);
    let data = await response.json();
    var length = data.length;
    const search_row = document.querySelector('.items__row');
    for (const item of data) {
        response = await fetch(
`http://localhost:2210/image?imgName=${item.image}`);
        let blob = await response.blob();
        let image = URL.createObjectURL(blob);
        search_row.innerHTML += ` <div class="item"> <div
class="item__content"> <a class="item__href"
href="./item.html?id=${item.book_id}">  <div id="\${item.book_id}"
class="item__content__name"> ${item.name} </div> </a> <div
class="item__content__row"> <div class="item__content__collum">
<div class="item__content__price"> ${item.price} грн </div> </div>
${item.quantity > 0 ? ` <button class="button__buy"> Купити
</button>` : ` <div class="text-error center">Немає в
наявності</div> <button class="button__buy display_none"> Купити
</button>`} </div> </div> </div> `;
        length = length - 1;
    };
    if(length = 1){ resolve(); }
});
}

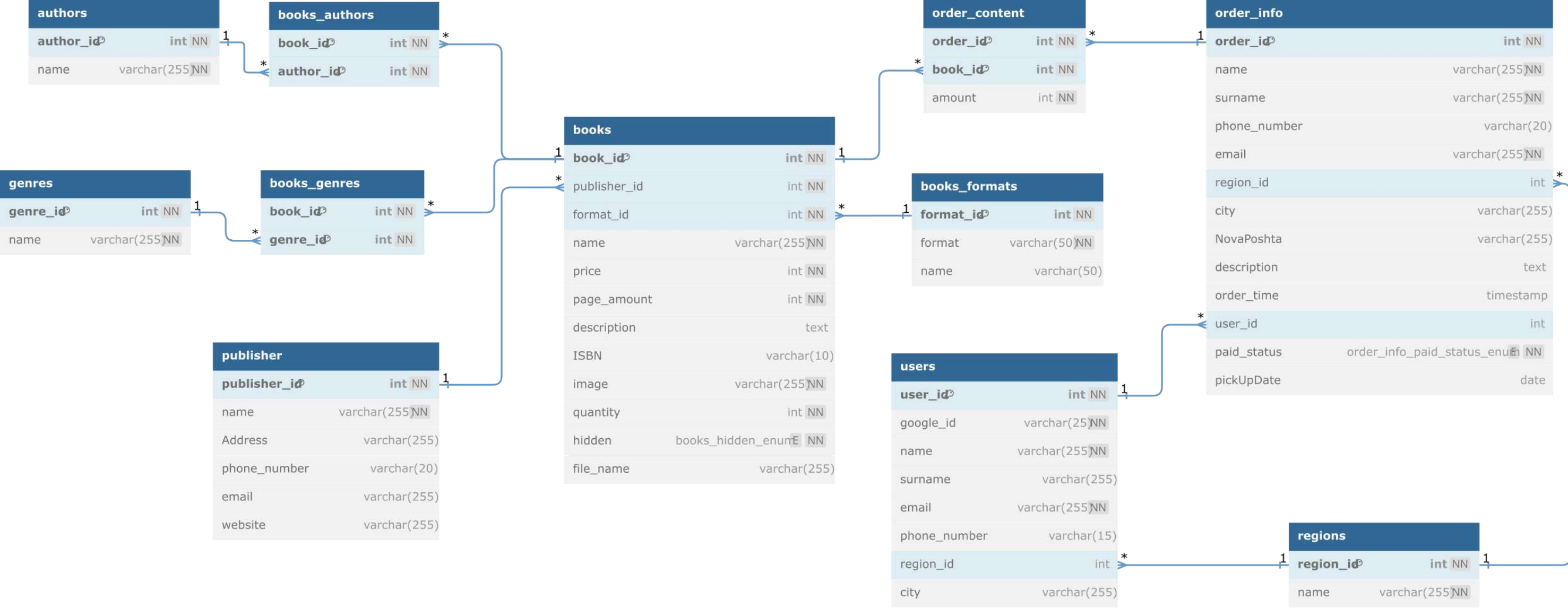
```

Алгоритм оформлення замовлення



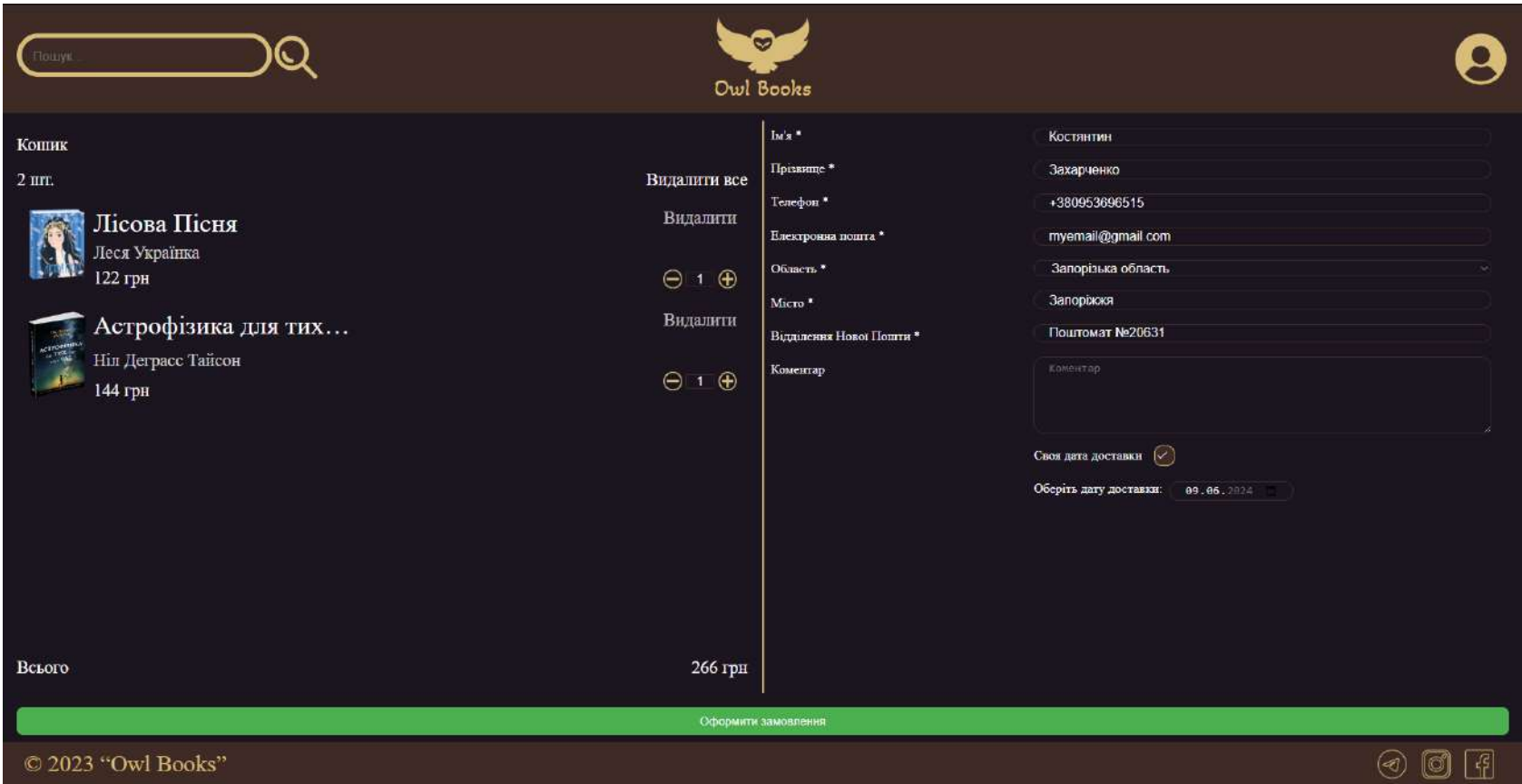
					13.02070849.00040 ПЛ1								
					Розробка комп'ютерної системи інтернет-магазину книгарні				Лит.		Маса	Масшт.	
Зам.	Лист.	№ докум.	Підп.	Дата									
Розроб.		Захарченко К.Р.											
Перев.		Хохлов М.М.											
Т.контр.									Лист 1		Листів 1		
					Алгоритм оформлення замовлення				НУ «Запорізька політехніка» КНТ-520				
Н.контр.		Польська О.В.											
Затв.		Кудерметов Р.К.											

Відношення таблиць
бази даних системи

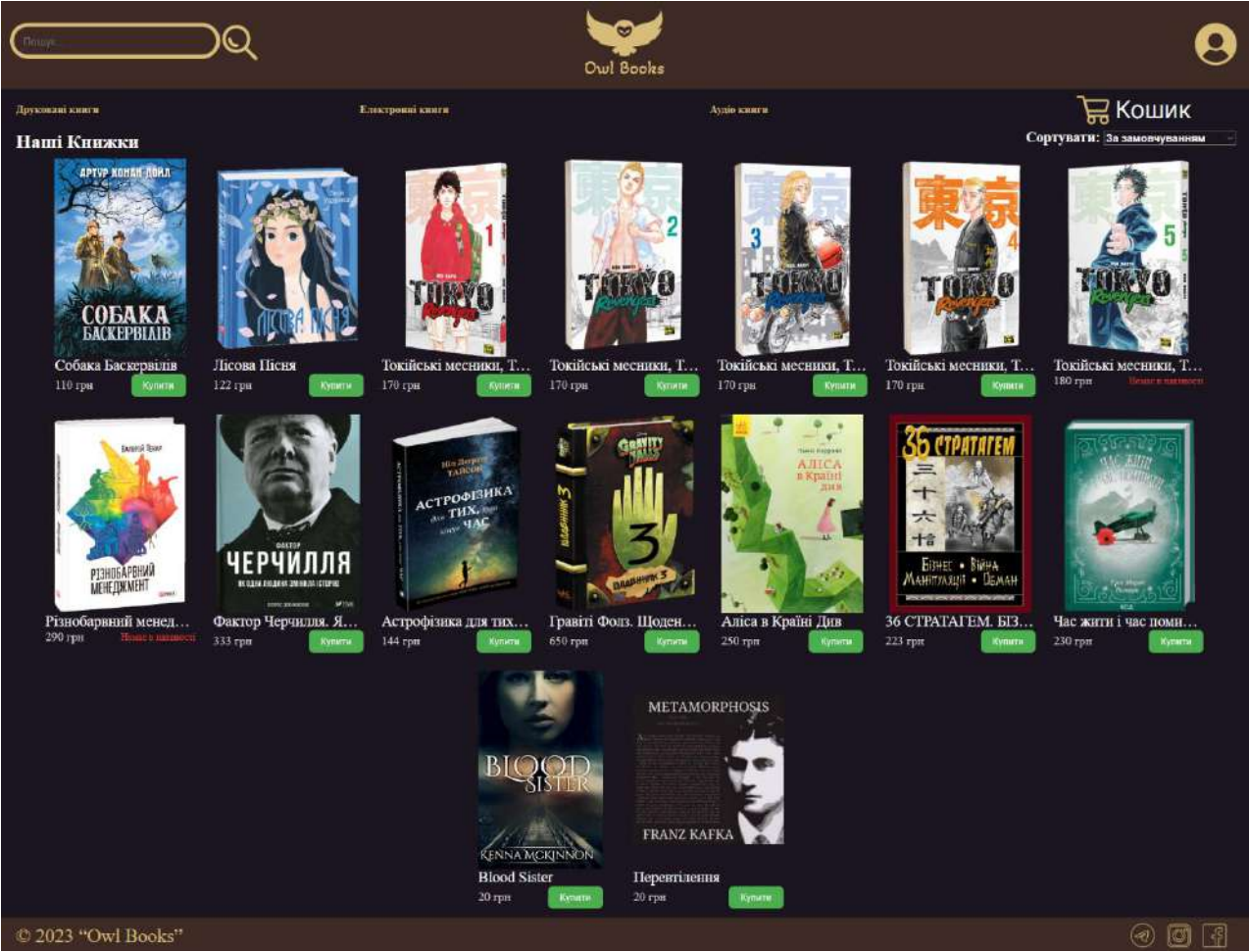


						13.02070849.00040 ПЛ2									
						Розробка комп'ютерної системи інтернет-магазину книгарні				Лист.		Маса	Масшт.		
Зам.	Лист.	№ докум.	Підп.	Дата						Лист 1		Листів 1			
Розроб.		Захарченко К.Р.													
Перев.		Хохлов М.М.													
Т.контр.															
Н.контр.		Польська О.В.				Відношення таблиць бази даних системи				НУ «Запорізька політехніка» КНТ-520					
Затв.		Кудерметов Р.К.													

Сторінки сайту системи
інтернет-магазину книгарні



Сторінка оформлення замовлення



Головна сторінка

						13.02070849.00040 ПЛЗ							
						Розробка комп'ютерної системи інтернет-магазину книгарні			Лист.		Маса	Масшт.	
Зам.	Лист.	№ докум.	Підп.	Дата					Лист 1		Листів 1		
Розроб.		Захарченко К.Р.											
Перев.		Хохлов М.М.											
Т.контр.													
Н.контр.		Польська О.В.				Сторінки сайту системи інтернет-магазину книгарні			НУ «Запорізька політехніка» КНТ-520				
Затв.		Кудерметов Р.К.											