Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматики
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

WEB-РЕСУРС ДЛЯ ОБМІНУ КНИГАМИ

Виконала: студентка 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Каташинська А.О
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Сілагін О.В.
(прізвище та ініціали)
«07» 06 2024 р.

Рецензент: к.т.н., доцент каф. АПТ
Козачок О.М.
(прізвище та ініціали)
«07» 06 2024 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«07» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра Комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«07» 03 2024 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТЦІ
Каташинській Ангеліні Олександрівні

1. Тема роботи: WEB-ресурс для обміну книгами.
Керівник роботи: Сілагін Олексій Віталійович, к.т.н., доцент кафедри КН.
затверджені наказом вищого навчального закладу від «11» 03 2024 року № 80

2. Термін подання студентом роботи 27.05.2024р.

3. Вихідні дані до роботи:
Кількість сутностей – не менше 4;
Кількість сторінок – не менше 4;
Реляційна база даних;
Відповідність стандарту ODBC;
Дизайн інтерфейсу має бути в білому та синіх тонах, інтуїтивно зрозумілий.

4. Зміст текстової частини БДР (перелік питань які потрібно розробити):
вступ; обгрунтування доцільності розробки WEB-ресурсу для обміну книгами; аналіз відомих програмних рішень для обміну книгами; проектування WEB-ресурсу для обміну книгами; програмна реалізація WEB-ресурсу для обміну книгами; тестування роботи програми та аналіз результатів; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень):
UML-діаграма прецедентів WEB-ресурсу для обміну книгами; UML-діаграма послідовності WEB-ресурсу для обміну книгами; UML-діаграма розгортання WEB-ресурсу для обміну книгами; ER-модель предметної області WEB-ресурсу для обміну книгами; схема бази даних WEB-ресурсу для обміну книгами; схема алгоритму роботи WEB-ресурсу для обміну книгами; приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Сілагін О.В., к.т.н., доцент каф. КН		

7. Дата видачі завдання 07.03.2024

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Обгрунтування доцільності розробки WEB-ресурсу для обміну книгами	06.05.24	10.05.24	
2	Аналіз відомих програмних рішень для обміну книгами	11.05.24	15.05.24	
3	Проектування WEB-ресурсу для обміну книгами	16.05.24	19.05.24	
4	Програмна реалізація WEB-ресурсу для обміну книгами	20.05.24	24.05.24	
5	Тестування роботи програми та аналіз результатів	27.05.24	29.05.24	
6	Розробка інструкції користувача	30.05.24	03.06.24	
7	Оформлення матеріалів до захисту БДР	04.06.24	06.06.24	

Студентка _____ (підпис) **Каташинська А.О.** (прізвище та ініціали)

Керівник роботи _____ (підпис) **Сілагін О.В.** (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.4

Каташинська А.О. WEB-ресурс для обміну книгами. Бакалаврська дипломна робота зі спеціальності 122 – комп'ютерні науки, освітня програма – системи штучного інтелекту. Вінниця: ВНТУ, 2024.

Бакалаврська дипломна робота складається з 93 сторінки формату А4, на яких є 36 рисунків, 4 таблиці, список використаних джерел містить 30 найменувань.

В даній бакалаврській дипломній роботі досліджено процес розробки WEB-ресурсу для обміну книгами. Проаналізовано наявні програми-аналоги та визначено їхні переваги та недоліки. Сформовано вимоги до WEB-ресурсу та досліджено способи їх вирішення. Проведено проектування за допомогою UML-діаграм та розроблено схему алгоритму програми. Програмне забезпечення було розроблено в інтегрованому середовищі Visual Studio Code за допомогою мови програмування JavaScript та платформи Node.js. В роботі було використано бібліотеки React, Redux, Material-UI, React Hook Form для клієнтської частини та Express.js, Multer, Bcrypt для серверної. Результати тестування підтвердили, що основні функції WEB-ресурсу реалізовано.

Ключові слова: WEB-ресурс, обмін книгами, буккросинг, книга, клієнт-серверна архітектура, UML-діаграма, база даних, програмування.

ABSTRACT

Katashynska A.O. Web resource for exchanging books. Bachelor thesis on specialty 122 - computer science, educational program - artificial intelligence systems. Vinnytsia: VNTU, 2024.

The bachelor’s thesis consists of 93 pages of the A4 format, on which there are 36 drawings, 4 tables, the list of used sources contains 30 titles.

In this bachelor's thesis, the process of developing a WEB resource for exchanging books is investigated. Existing analog programs were analyzed and their advantages and disadvantages were determined. The requirements for the WEB resource were formed and ways of solving them were investigated. The design was carried out with the help of UML-diagrams and the scheme of the algorithm of the program was developed. The software was developed in the integrated Visual Studio Code environment using the JavaScript programming language and the Node.js platform. The work used React, Redux, Material-UI, React Hook Form libraries for the client part and Express.js, Multer, Bcrypt for the server part. The test results confirmed that the main functions of the WEB resource have been implemented.

Keywords: web resource, book exchange, bookcrossing, book, client-server architecture, UML diagram, relational database, programming.

ЗМІСТ

ВСТУП

Актуальність. Книги є важливим джерелом знань та мають велике значення для розвитку людства. Вони допомагають розширювати світогляд, розвивати критичне мислення та вивчати нові мови. Крім того, книги є інструментом для збереження та передачі культурної спадщини від одного покоління до іншого.

Поточний стан використання книг супроводжується рядом проблем, серед яких обмежений доступ до книг через їх накопичення на полицях власників без подальшого обігу, фінансові труднощі при придбанні нових видань, особливо рідкісних. Крім того, виробництво паперових книг має негативний вплив на екологію через споживання природних ресурсів та забруднення довкілля.

Для вирішення зазначених проблем в світі набуває популярності практика обміну книгами, відома як буккросинг. Принцип роботи буккросингу полягає в тому, що людина залишає книгу в громадському місці або передає іншому читачеві. Наступний читач може забрати книгу, прочитати, а потім продовжити її «подорож». Учасники реєструють книги на онлайн-платформі, що дозволяє відстежувати їх переміщення.

Незважаючи на переваги та популярність буккросингу, його звична форма має певні недоліки. Книги, залишені в громадських місцях, доступні лише для обмеженого кола читачів, що ускладнює обмін між особами з різних регіонів. Буккросинг у класичному вигляді не завжди надає можливість шукати конкретні книги, а відсутність контролю за якістю та збереженням книг під час обміну може призводити до їх пошкодження або втрати без можливості відстеження. Описані проблеми підкреслюють необхідність розробки WEB-ресурсу, який зміг би запропонувати більш ефективний та зручний спосіб обміну книгами.

Розробка WEB-ресурсу для обміну книгами є актуальним завданням, яке дозволить вирішити низку проблем, пов'язаних з використанням книг. Такий ресурс спростить процес обміну книгами, сприятиме популяризації читання, збереженню культурної спадщини та зменшенню негативного впливу на навколишнє середовище.

Метою дослідження є розширення функціональних можливостей WEB-ресурсу для обміну книгами.

Об’єктом дослідження є процес розробки WEB-ресурсу для обміну книгами.

Предметом дослідження є алгоритми та програмні засоби для реалізації WEB-ресурсу для обміну книгами.

Практична цінність полягає у тому, що розроблені алгоритми та методи можуть бути використані при створенні подібних WEB-ресурсів.

Задачі дослідження:

1. Обґрунтувати доцільність розробки WEB-ресурсу для обміну книгами;
2. Проаналізувати програми-аналоги, їх переваги та недоліки;
3. Спроектувати WEB-ресурс для обміну книгами;
4. Програмно реалізувати WEB-ресурс для обміну книгами;
5. Виконати тестування програми та провести аналіз отриманих результатів;

Апробація роботи. Результати досліджень було апробовано на LIII Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації Вінницького національного технічного університету (НТКП ВНТУ - 2024) м. Вінниці у 2024 р [1].

Публікації. За основними результатами досліджень бакалаврської дипломної роботи було опубліковано тези доповіді на науково-технічній конференції [1].

Впровадження. Результати, одержані в ході виконання бакалаврської дипломної роботи, а саме форми (додавання книги та виставлення книги на обмін), систему відгуків та налаштування користувача планується використати в розробках ТОВ «Намір Ентерпрайз», про що є відповідна довідка в додатку Д.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ WEB-РЕСУРСУ ДЛЯ ОБМІНУ КНИГАМИ

1.1 Огляд середовищ для реалізації програмного продукту

При розробці WEB-ресурсу для обміну книгами потрібно проаналізувати середовище, де майбутні користувачі зможуть отримати доступ до нього. Сучасні технології пропонують розробку для персональних комп’ютерів, мобільних пристроїв та WEB-браузерів. Розглянемо детальніше особливості кожного з цих середовищ.

Персональні комп'ютери (ПК) є середовищем виконання для настільних програм, розроблених для певних операційних систем. Ці програми мають широкий функціонал, оскільки не обмежені ресурсами, як мобільні пристрої. Настільні програми можуть використовувати обчислювальну потужність ПК та мати складний інтерфейс.

Мобільні пристрої, такі як смартфони та планшети, стали платформою для запуску мобільних додатків. Мобільні додатки забезпечують зручний та портативний доступ до різноманітних функцій та сервісів. Вони розробляються для певних операційних систем оптимізовані для невеликих екранів та сенсорного введення.

WEB-браузери є середовищем виконання для WEB-ресурсів. На відміну від традиційних програм, розроблених для певної операційної системи чи пристрою, вони працюють в будь-якому сучасному WEB-браузері на різних платформах: персональних комп'ютерах, мобільних пристроях, розумних телевізорах. Це забезпечує широку доступність та полегшує їх поширення [2].

Вибір оптимального середовища для розробки програмного модуля впливає на доступність, функціональність та зручність використання продукту. При порівнянні різних середовищ необхідно врахувати основні фактори. Це включає операційні системи, доступ, функціональність, продуктивність, безпеку, складність

розробки, масштабування та зручність використання. В таблиці 1.1 наведено порівняльну характеристику середовищ для розробки програмного модуля.

Таблиця 1.1 – Порівняльна характеристика середовищ для розробки програмного модуля

Характеристика	ПК	Мобільний пристрій	WEB-браузер
Операційні системи	Windows, Linux, macOS,	Android, iOS	Будь-який WEB-браузер
Доступ	Завантаження на ПК	Завантаження із магазину додатків	Не потрібно завантажувати
Функціональність	Широкий спектр	Обмежена	Залежить від браузера
Продуктивність	Висока, потужні апаратні засоби	Залежить від пристрою	Залежить від з'єднання
Безпека	Висока конфіденційність	Наявність потенційних загроз	Залежить від реалізації
Розробка	Складна	Середня	Проста
Масштабування	Легко	Складно	Легко
Зручність використання	Доступ до системних ресурсів	Портативність	Універсальність доступу

WEB-браузер є найпопулярнішим середовищем для реалізації програмного забезпечення. Він надає зручний доступ з будь-якого пристрою, підключеного до інтернету. Це надає доступність та простоту використання для всіх користувачів.

Розробка WEB-ресурсу є простішою та швидшою порівняно з створенням окремих додатків для різних операційних систем. Оновлення відбуваються на сервері та користувачі одразу отримують доступ до нової версії. Проте, функціонал може бути обмеженим через особливості WEB-браузера та доступ до апаратних ресурсів, безпека залежить від реалізації, а продуктивність від з'єднання. Також,

WEB-ресурс надає функціональність необхідну для ефективного обміну книгами. Створення фільтрації, система пошуку, особисті кабінети користувачів та інші корисні інструменти.

Отже, WEB-ресурси є оптимальним вибором завдяки своїй універсальності, простоті розробки та оновлення та масштабованості.

1.2 Аналіз відомих програмних рішень для обміну книгами

Перед розробкою WEB-ресурсу для обміну книгами необхідно провести аналіз наявних рішень на ринку. Він дозволить зрозуміти, які функції вже реалізовані, а які потребують покращення або впровадження. Крім того, надасть можливість виявити тенденції в розробці та створити продукт, що задовольнить потреби користувачів.

Bookcrossing – це сайт, який спеціалізується на створенні світової бібліотеки для підтримки концепції буккросингу. Створений Роном Хорнбеком, функціонує як платформа для об'єднання любителів книг, забезпечуючи культурний обмін та розвиток спільноти [3]. Головна сторінка описаного WEB-ресурсу зображена на рисунку 1.1.

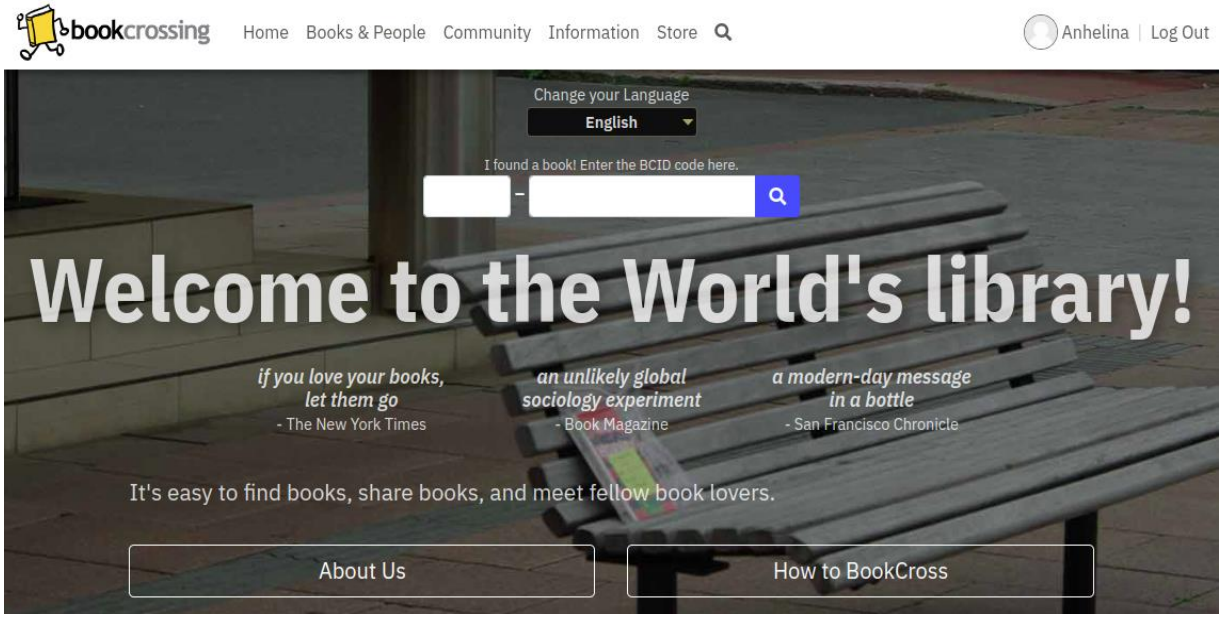

Рисунок 1.1 – Головна сторінка WEB-ресурсу Bookcrossing

Основний функціонал Bookcrossing полягає у можливості зареєструвати свої книги, залишити у публічних місцях та відстежувати їхню подальшу мандрівку. Користувачі можуть поділитися враженнями, залишивши відгук та оцінку. Платформа пропонує розширений пошук за назвою, автором та унікальним номером книги. Ще однією перевагою є інтерактивна карта з місцями, де були залишені чи знайдені книги.

Серед недоліків варто відзначити незручний інтерфейс та непередбачуваність знахідок, що не гарантує отримання бажаного видання, а стан залишених книг може бути незадовільним. Географічна обмеженість є суттєвим недоліком, тому що ефективність платформи залежить від активності спільноти у певному регіоні.

BookMooch – це сайт, який надає можливість обмінюватись книгами між користувачами по всьому світу. Його засновник Джон Бакман був розчарований кількістю книг, які були видані лише в одній країні та недоступні для інших. Маючи на меті зробити літературу доступною для всіх, незалежно від місця проживання, він створив глобальну спільноту, яка заохочує людей ділитися книгами [4]. Головна сторінка описаного WEB-ресурсу зображена на рисунку 1.2.

Рисунок 1.2 – Головна сторінка WEB-ресурсу BookMooch

Перевагами BookMooch є можливість надсилати бали благодійним організаціям для придбання необхідної літератури, що дозволяє підтримати важливі ініціативи. Зручною функцією є наявність списку бажань з сповіщенням про доступність бажаних книг. Крім того, реалізовано розширену систему фільтрації за певними параметрами, що робить процес пошуку ефективним та персоналізованим для користувачів. Однак, наявний незручний та застарілий інтерфейс, який може створювати труднощі для користувачів. Для пошуку за ISBN, потрібно купувати преміум-акаунт. Також процес реєстрації є досить деталізованим та складним для проходження.

Glodibi Marketplace – це сайт для купівлі, продажу та обміну вживаними паперовими книгами між користувачами. Метою є надання зручного майданчика для книголюбів, де вони самостійно встановлюють ціну та умови обміну книгами [5]. Головна сторінка описаного WEB-ресурсу зображена на рисунку 1.3.

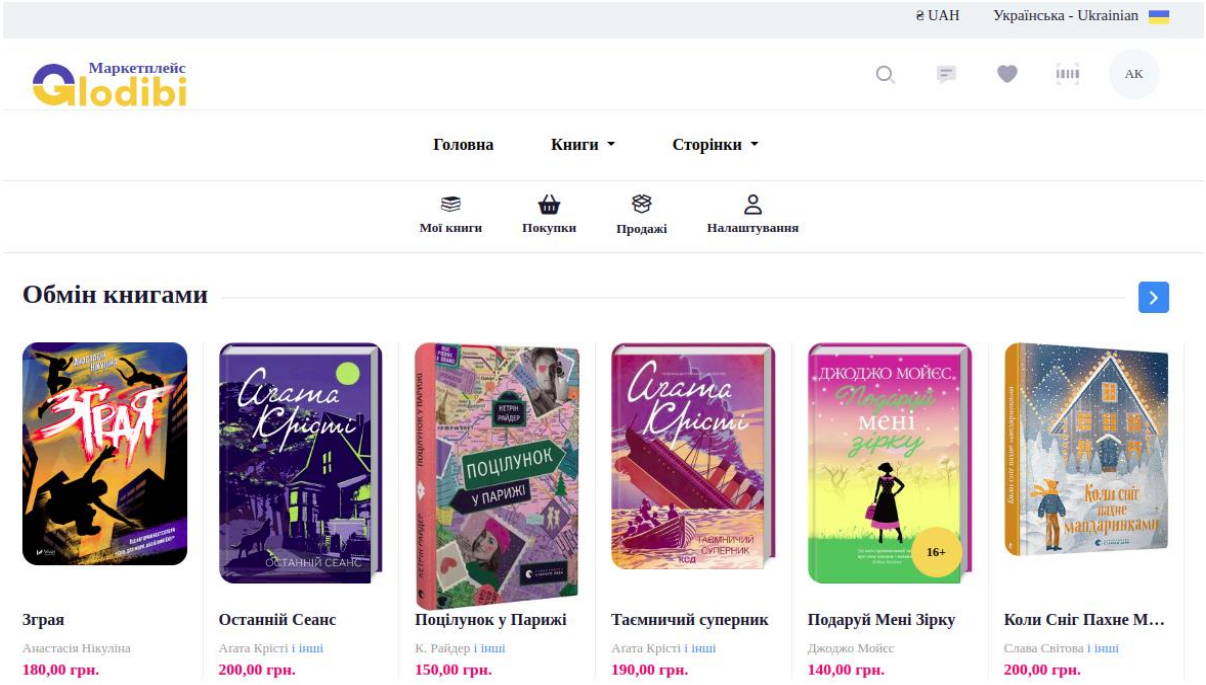

Рисунок 1.3 – Головна сторінка WEB-ресурсу Glodibi Marketplace

Основний функціонал платформи полягає у створенні оголошень про продаж або бажання обміняти книги з персональної бібліотеки. Для цього користувачеві необхідно мати обліковий запис на платформі. Оголошення можуть містити

запропоновану ціну продажу чи позначку про готовність до обміну. Крім того, існує можливість безкоштовно передавати книги іншим учасникам.

Glodibi Marketplace має низку переваг, серед яких гнучкість у виборі умов продажу чи обміну, наявність української локалізації та широкий вибір пропозицій від різних продавців з усього світу. Безкоштовна передача дозволяє популяризувати читання. Проте, варто врахувати система пошуку та фільтрації потребує доопрацювання для ефективного та швидкого знаходження потрібних книг.

PaperBackSwap – це американський сайт із великою спільнотою книголюбів, створений для безкоштовного обміну книгами. Принцип його роботи полягає в тому, що для отримання бажаного видання потрібно надіслати одну із своїх книг іншому учаснику спільноти [6]. Головна сторінка описаного WEB-ресурсу зображена на рисунку 1.4.

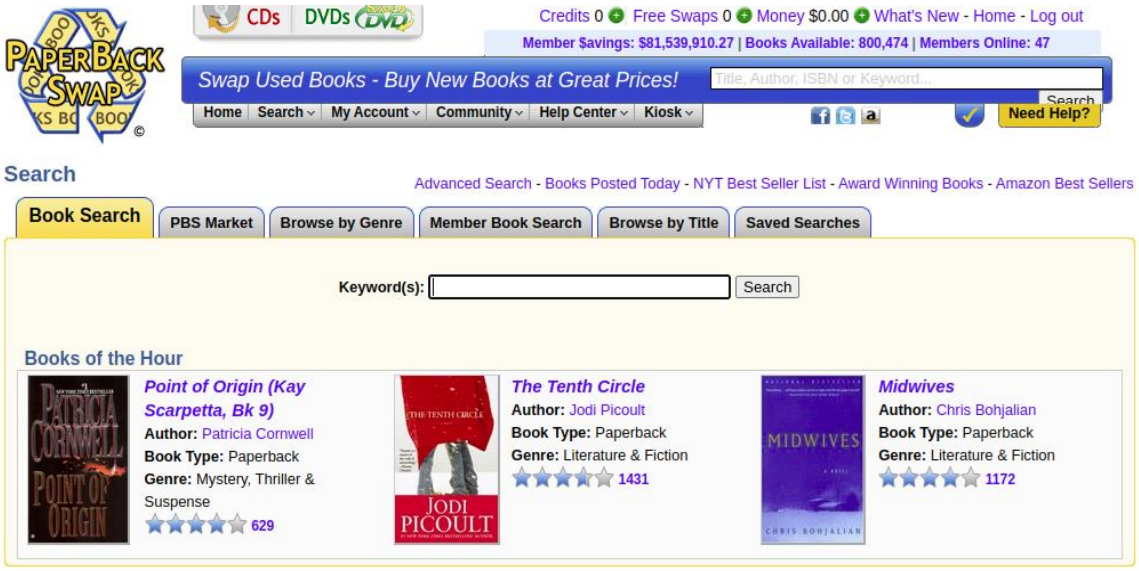

Рисунок 1.4 – Головна сторінка WEB-ресурсу PaperBackSwap

Головною перевагою PaperBackSwap є потужна система пошуку та фільтрації книг за різноманітними параметрами. Користувачі можуть шукати літературу за назвою, автором, категорією, жанром, роком публікації. Крім того, на платформі присутня можливість залишати відгуки про прочитані книги для інших учасників спільноти. Одним з недоліків є відсутня можливість обирати конкретного

користувача для обміну, адже книга надсилається тому, хто її замовив першим. Також на сайті міститься багато реклами, а інтерфейс користувача є застарілим та незручним через велику кількість надлишкової інформації та відсутність належної структуризації.

Bookswap – це британський сайт для обміну книгами на основі системи балів. Користувачі заробляють бали за надіслані книги та витрачають їх на отримання літератури від інших користувачів [7]. Головна сторінка описаного WEB-ресурсу зображена на рисунку 1.5.

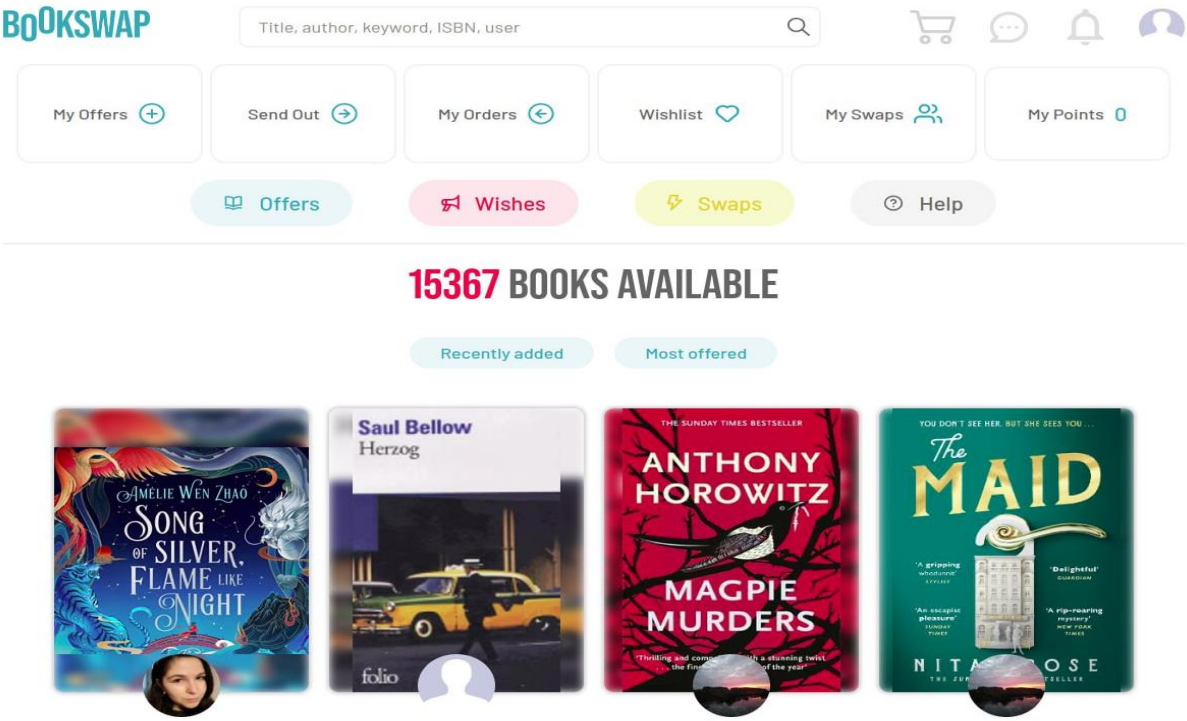

Рисунок 1.5 – Головна сторінка WEB-ресурсу Bookswap

Основними перевагами Bookswap є зручний та сучасний інтерфейс користувача, що полегшує навігацію. Також присутня можливість вибору користувача для обміну та потужна система пошуку за різними параметрами такими як, назва, автор, унікальний номер книги, користувач або ключові слова, які можуть бути пов’язані з виданням.

Одним із недоліків Bookswap є відсутність української локалізації, що створює проблеми для користувачів, які не володіють англійською мовою. Крім

того, система балів забороняє одразу здійснювати обмін, оскільки спочатку потрібно заробити достатню кількість балів шляхом відправлення власних книг. В таблиці 1.2 наведено порівняльну характеристику WEB-ресурсів для обміну книгами.

Таблиця 1.2 – Порівняльна характеристика програм-аналогів для обміну книгами

Характеристика	Glodibi Marketplace	Book Mooch	Book-swap	Bookcrossing	PaperBack Swap
Фільтрація за мовою книги та категоріями	-	+	-	+	+
Пошук книг за назвою, автором, ISBN	-	-	+	+	+
Вибір користувача для обміну	+	-	+	-	-
Інформація про стан книги	+	-	+	-	-
Можливість додати відгук про книгу	+	-	-	+	+
Наявність списку бажань	+	+	-	-	-

На основі проведеної порівняльної характеристики, можна зробити висновок, що є потреба в розробці WEB-ресурсу з повним набором функцій. Програми-аналоги надають лише обмежені можливості, що створює низку незручностей для користувачів. Створення універсального WEB-ресурсу для обміну книгами, який поєднає в собі всі необхідні функції, дозволить усунути виявлені недоліки та забезпечити зручну та ефективну роботу користувачів в межах однієї програми.

1.3 Формулювання вимог та постановка задачі

Розробка WEB-ресурсу для обміну книгами потребує аналізу вимог та постановки задач. В цьому підрозділі буде описано основний функціонал до програмного забезпечення з урахуванням потреб користувачів. Це дозволить краще провести проектування та програмну реалізацію WEB-ресурсу.

WEB-ресурс має забезпечувати зручний обмін книгами між користувачами, які прочитали або більше не потребують певні книги. Серед основних функцій, які необхідно реалізувати, можна виділити такі:

- Реєстрація та авторизація: надання можливості користувачам створювати облікові записи та авторизуватися для доступу до основного функціоналу;
- Профіль користувача: кожен користувач матиме особистий профіль, де він зможе зберігати свої дані, додавати та редагувати інформацію про свою колекцію книг;
- Список бажань: на сторінці інформації про книгу має бути можливість додати книгу до списку бажань, це допоможе швидко знаходити бажані книги та перевіряти їх наявність;
- Додавання книги на обмін: користувачі зможуть додавати книги до списку обміну, вказуючи основні характеристики книги та її стан;
- Система пошуку: можливість ефективно знаходити книги за допомогою пошуку за назвою, автором та унікальним номером книги;
- Фільтрація книг: можливість швидко знаходити книги за допомогою фільтрів за категоріями та мовою написання;
- Вибір користувача для обміну: реалізація можливості обирати певного користувача та надсилати запит на обмін;
- Адаптивність: має бути адаптивний до різних пристроїв, щоб забезпечити зручний доступ з будь-якого пристрою;
- Система відгуків: кожен користувач буде мати можливість залишати відгук про книгу, редагувати його та за потреби видаляти;

- Отримання інформації про книгу: можливість відкриття сторінки певної книги.

Також WEB-ресурс забезпечить зручний користувацький досвід для обміну книгами між користувачами. Вхідні дані завантажуватимуться з бази даних та будуть представлені в зрозумілому вигляді. У результаті розробки буде створено WEB-ресурс, який надасть можливість ефективного обміну книгами, забезпечуючи простоту та швидкість процесу обміну книг.

1.4 Висновок до розділу 1

У даному розділі було проведено обґрунтування доцільності розробки WEB-ресурсу для обміну книгами. Спершу було проаналізовано середовища для реалізації програмного забезпечення. Було обрано WEB-ресурс, як оптимальний вибір завдяки універсальності доступу, простоті розробки та оновлення, масштабованості та широким можливостям для необхідного функціоналу.

Проведено огляд наявних WEB-ресурсів для обміну книгами. Були розглянуті такі WEB-ресурси: PaperBackSwap, BookMoch, Bookcrossing, Bookswap, Globidi Marketplace. Кожен з них має свої переваги та недоліки, проте жоден з них не може повністю задовільнити потреби користувача.

Виконано постановку задачі, що включала в себе формулювання основного функціоналу розроблюваного WEB-ресурсу для обміну книгами. Користувачі матимуть особисті профілі для управління своїми книгами, можливість додавати книги на обмін із зазначенням їхнього стану, список бажань, система відгуків, функції пошуку та фільтрації за різними параметрами.

2 ПРОЕКТУВАННЯ WEB-РЕСУРСУ ДЛЯ ОБМІНУ КНИГАМИ

2.1 Обґрунтування вибору архітектури для WEB-ресурсу

Архітектура програмного забезпечення є фундаментальним аспектом розробки, оскільки вона визначає структуру, організацію та взаємодію компонентів системи. Вибір архітектури має вирішальне значення для забезпечення ефективності, масштабованості, безпеки. Для WEB-ресурсу важливо обрати архітектуру, яка дозволить досягти поставлених завдань, забезпечить гнучкість у розробці та дозволить адаптуватися до майбутніх потреб.

У сучасному світі розробки існує низка загальновизнаних архітектур, які широко застосовуються при створенні WEB-ресурсів. Серед них можна виділити такі як монолітна, мікросервісна, безсерверна, Model-View-Controller (MVC) та клієнт-серверна архітектури. Проаналізуємо кожну з них, щоб обрати оптимальну для нашого проекту.

MVC – це архітектурний шаблон, який широко використовується під час розробки програмного забезпечення. Він розділяє програму на три компоненти: модель (Model), представлення (View) та контролер (Controller). Використання цього шаблону у великих проектах сприяє впорядкованості структури та робить їх зрозумілими шляхом зменшення складності [8]. Розглянемо детальніше кожну складову описаної архітектури:

- Модель: відображає бізнес-логіку та відповідає за керування даними, валідацію, обробку та збереження. Модель є незалежною від представлення чи контролера і може використовуватися в різних частинах програми;
- Представлення: отримує дані та відображає їх у зрозумілому для користувача форматі, наприклад у вигляді HTML-сторінок;
- Контролер: посередник між моделлю та представленням. Він отримує запити від користувача, обробляє їх, викликає відповідні методи моделі та передає дані до представлення для відображення. Контролер також відповідає за обробку введених даних та оновлення моделі.

MVC архітектура схематично зображена на рисунку 2.1.

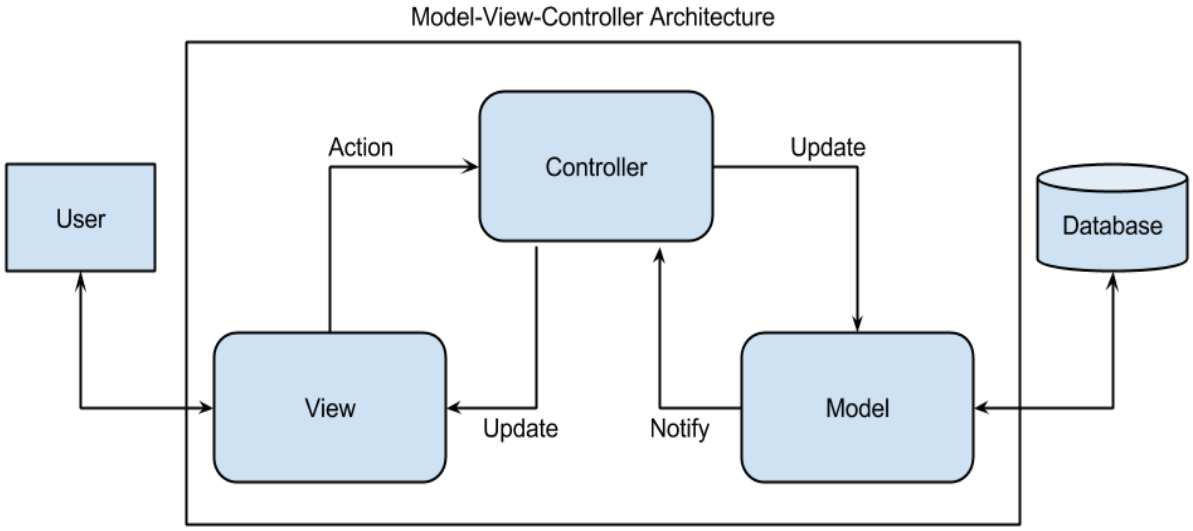

Рисунок 2.1 – Схематичне зображення MVC архітектури

Переваги MVC архітектури:

1. Структурованість: розділяє систему на три чітко визначені компоненти, що полегшує розуміння, розробку та підтримку коду;
2. Повторне використання коду: дозволяє повторно використовувати код, наприклад, однакову модель в різних представленнях;
3. Гнучкість та розширюваність: дозволяє легко додавати новий функціонал та модифікувати наявні компоненти без впливу на інші частини програми;
4. Легкість тестування: розділення компонентів полегшує модульне тестування кожного компонента окремо.

Недоліки MVC архітектури:

1. Складність: для простих проектів використання може вносити зайву складність в архітектуру;
2. Обмеження розширюваності: може виникнути потреба в додаткових компонентах архітектури, які не входять до складу MVC;
3. Залежність представлення від моделі: коли змінюється модель, то потрібно вносити зміни в представлення.

MVC архітектура є потужним підходом для розробки WEB-ресурсів. Вона забезпечує чітку структуру, однак перед вибором необхідно враховувати складність проекту та проаналізувати переваги та недоліки даної архітектури.

Монолітна архітектура є традиційним підходом до розробки програмного забезпечення, при якому весь функціонал розробляється як одна єдина система. Всі компоненти взаємодіють один з одним та розгортання відбувається на одному сервері або групі серверів [9]. Монолітна архітектура схематично зображена на рисунку 2.2.

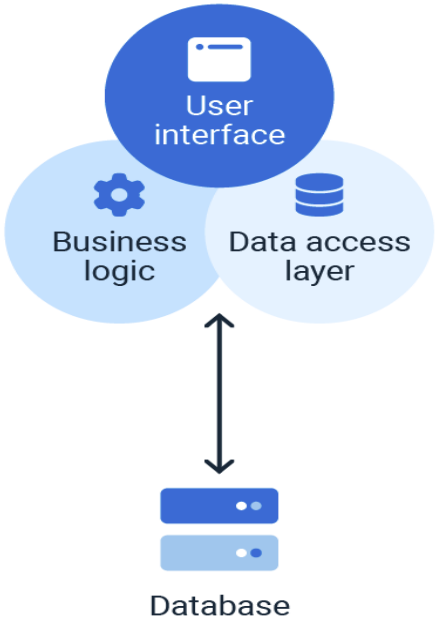

Рисунок 2.2 – Схематичне зображення монолітної архітектури

Переваги монолітної архітектури:

1. Простота розробки: оскільки всі компоненти розташовані в одному місці;
2. Легкість розгортання: потрібно розгорнути лише один модуль або додаток;
3. Ефективність: може бути ефективною для невеликих та простих систем, де немає потреби в масштабуванні окремих компонентів.

Недоліки монолітної архітектури:

1. Складність масштабування: при збільшенні розміру та складності системи, масштабування стає складнішим, оскільки необхідно масштабувати всю систему;

2. Обмежена гнучкість: зміни в одній частині можуть вплинути на інші частини, що ускладнює внесення змін та додавання нового функціоналу;
3. Складність підтримки: всі компоненти тісно пов'язані між собою, що ускладнює розуміння та внесення змін до системи.

Монолітна архітектура може бути хорошим вибором для простих та невеликих WEB-ресурсів, де немає потреби в масштабуванні окремих компонентів та де швидкість розробки є важливішою за гнучкість та масштабованість. Однак, для більш складних проектів описана архітектура може стати обмежуючим фактором, тому варто розглянути інші архітектурні підходи.

Мікросервісна архітектура представляє інноваційний підхід до створення програмного забезпечення в основі якого лежать мікросервіси та їх взаємодія. Концепція ґрунтується на принципі декомпозиції системи на дрібніші модулі, які можуть розроблятися, розгортатися та масштабуватися окремо один від одного [9]. Мікросервісна архітектура схематично зображена на рисунку 2.3.

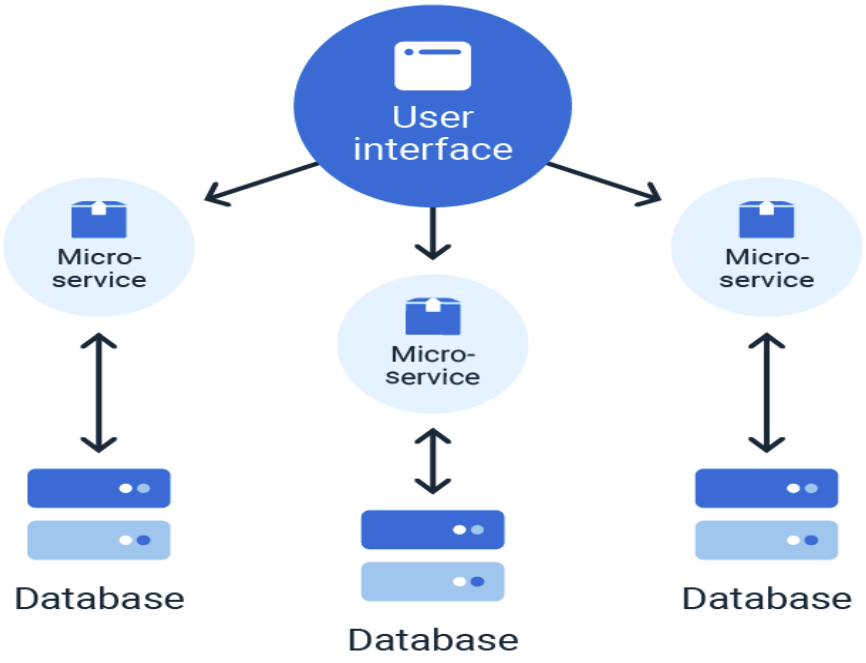

Рисунок 2.3 – Схематичне зображення мікросервісної архітектури

Переваги мікросервісної архітектури:

1. Масштабованість: мікросервіси функціонують незалежно один від одного;

2. Надійність: збій роботи одного мікросервісу впливає лише на функції, за які він відповідає та не зупиняє роботу програми повністю;
3. Технологічна різноманітність: мікросервіси можуть бути розроблені з використанням різних технологій та мов програмування, що дозволяє обирати найбільш відповідні інструменти для кожного сервісу.

Недоліки мікросервісної архітектури:

1. Складність розробки: мікросервіси потребують координації та взаємодії;
2. Витрати на інфраструктуру: розробка та обслуговування системи;
3. Складність тестування: оскільки необхідно тестувати кожен мікросервіс окремо та їх інтеграцію загалом;
4. Затримки у мережі: можуть виникати додаткові проблеми з продуктивністю, особливо при інтенсивній взаємодії між сервісами;
5. Управління даними: децентралізований підхід до управління даними в мікросервісній архітектурі може призвести до проблем з узгодженістю даних.

Мікросервісна архітектура є ефективним підходом для розробки масштабованих та гнучких WEB-ресурсів, особливо для великих проектів. Проте, вона також приносить додаткову складність та вимагає ретельного планування та управління. Перш ніж обирати цю архітектуру, необхідно врахувати особливості проекту, вимоги до продуктивності та наявні ресурси.

Безсерверна архітектура - це революційний підхід до проектування програмного забезпечення, який ґрунтується на ідеї використання хмарних обчислень та сервісів для виконання програмного коду без потреби в управлінні серверами. За такої архітектури розробники концентруються на створенні та розгортанні окремих функцій, тоді як усі питання інфраструктури, такі як масштабування та адміністрування серверів, передаються хмарному провайдеру [10].

Основними складовими безсерверної архітектури є функції, які містять бізнес-логіку та можуть викликатися через API або тригери, використання хмарних сервісів, таких як AWS Lambda, Google Cloud Functions або Azure Functions.

Особливістю цієї архітектури є те, що обчислювальні ресурси виділяються динамічно, в залежності від потреб кожної функції та оновлюються автоматично у відповідь на зміни навантаження. Безсерверна архітектура схематично зображена на рисунку 2.4.

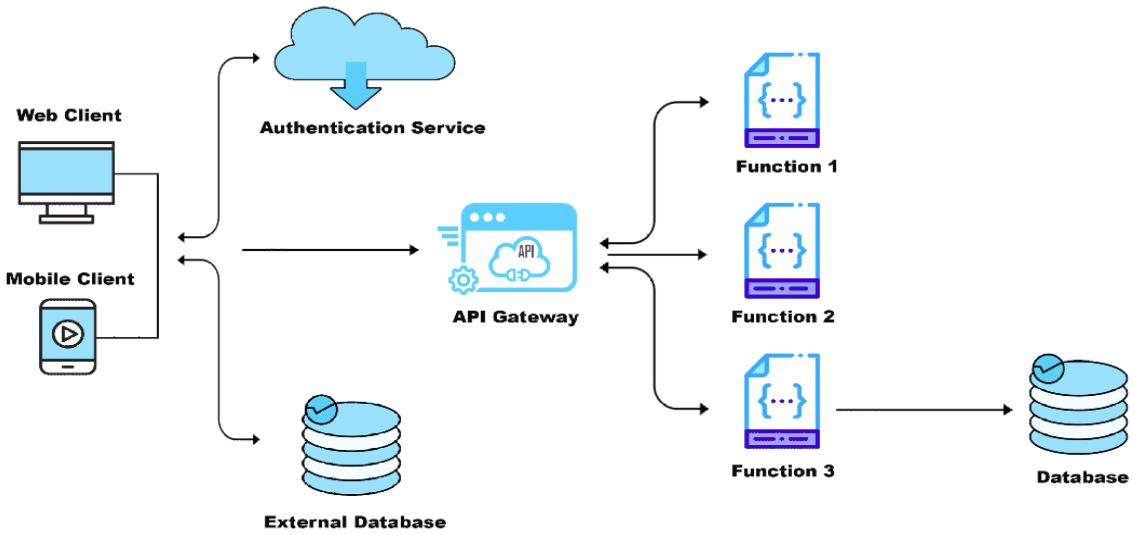

Рисунок 2.4 – Схематичне зображення безсерверної архітектури

Переваги безсерверної архітектури:

1. Оптимізація витрат: дозволяє сплачувати лише за фактично спожиті ресурси, що може суттєво скоротити витрати на інфраструктуру, особливо для додатків з нерівномірним навантаженням;
2. Автоматичне масштабування: хмарний провайдер налаштовує ресурси;
3. Продуктивність: розробники зосереджені на написанні програмного коду та не витрачають час на розгортання та масштабування продукту.

Недоліки безсерверної архітектури:

1. Залежність від провайдера: архітектура має сильну залежність від хмарного провайдера, що може спричинити втрату контролю над ресурсом;
2. Обмеження функцій: обмеження щодо тривалості виконання, обсягу пам'яті та розміру пакету може бути проблемою при розробці;
3. Складність тестування та відлагодження: може бути більш складним, оскільки код виконується в керованому середовищі провайдера;

Безсерверна архітектура є привабливим вибором для багатьох додатків, особливо з нерівномірним навантаженням. Проте, перш ніж обирати дану архітектуру, необхідно оцінити вимоги проекту, врахувати обмеження та залежності від провайдера.

Клієнт-серверна архітектура є однією з найпоширеніших архітектурних моделей для розробки програм. Вона базується на розділенні функціоналу між клієнтом та сервером [11]. Клієнт-серверна архітектура схематично зображена на рисунку 2.5.

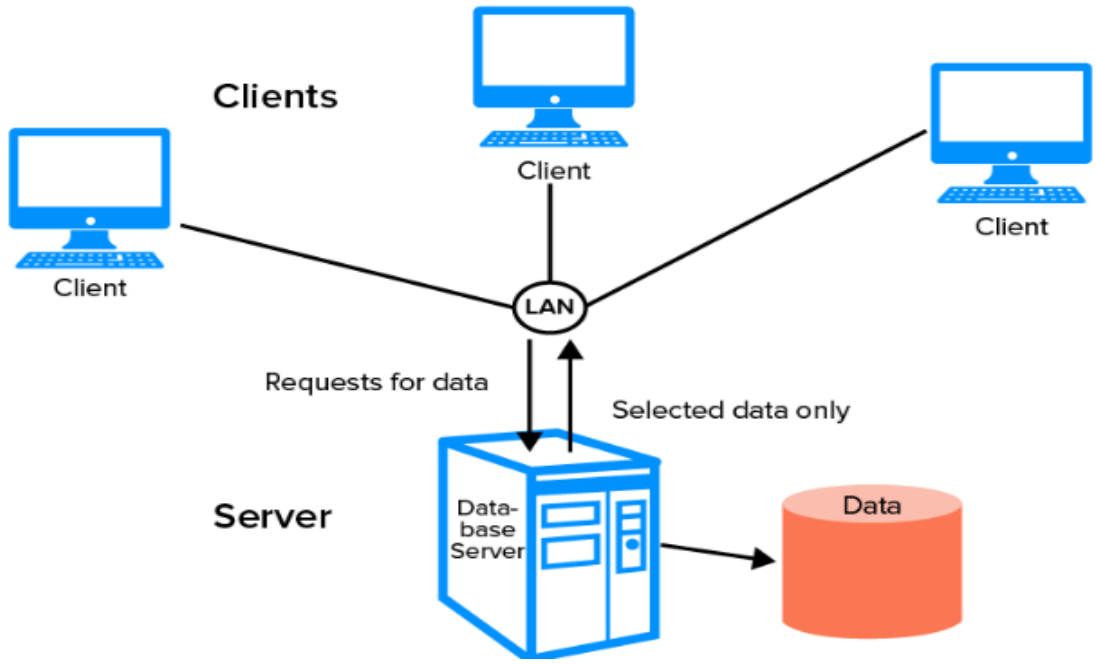

Рисунок 2.5 – Схематичне зображення клієнт-серверної архітектури

Клієнтська частина безпосередньо взаємодіє з користувачем, забезпечуючи зручний та інтуїтивно зрозумілий інтерфейс. Вона відповідає за збір введених користувачем даних, відправлення запитів до сервера та відображення отриманих від сервера даних у належному форматі.

Серверна частина обробляє запити, виконує бізнес-логіку та керує даними. Вона аналізує отримані запити, виконує необхідні операції та формує відповіді для клієнтів. Також відповідає за збереження, управління та забезпечення безпеки й цілісності даних для надання користувачам необхідних послуг та інформації.

Переваги клієнт-серверної архітектури:

1. Централізоване управління даними: забезпечує централізоване збереження та управління даними на сервері, що спрощує забезпечення цілісності, безпеки та резервного копіювання даних;
2. Масштабованість: дозволяє легко масштабувати систему як горизонтально (додавання нових серверів), так і вертикально (збільшення потужності наявних серверів) для обробки зростаючого навантаження;
3. Крос-платформеність: клієнтська частина може бути реалізована на різних платформах, що забезпечує доступність для користувачів;
4. Розподіл навантаження: дозволяє розподілити навантаження між клієнтами та сервером. Сервер може виконувати складні операції та обробку даних, що покращує продуктивність та швидкодію системи;
5. Безпека даних: Централізоване збереження даних на сервері дозволяє реалізувати надійні механізми автентифікації, авторизації та захисту даних від несанкціонованого доступу.

Недоліки клієнт-серверної архітектури:

1. Залежність від сервера: клієнти повністю залежать від сервера, якщо сервер буде недоступний, клієнти не зможуть отримати необхідні послуги та дані;
2. Проблеми з інтеграцією: при оновленні серверної частини може виникнути потреба у внесенні змін до клієнта;
3. Складність розробки та обслуговування: потребує реалізації окремих частин та протоколів комунікації між ними.

При аналізі різних варіантів архітектур для розробки було обрано клієнт-серверну архітектуру через її переваги у сфері масштабованості, безпеки та централізоване керування даними. Ця архітектура дозволить ефективно організувати роботу WEB-ресурсу, розділити функціональність між клієнтом та сервером, забезпечити надійну взаємодію з користувачами та обробку даних. Клієнт-серверна архітектура сприятиме подальшому розвитку та підтримці WEB-ресурсу для обміну книгами, дозволяючи адаптуватися до потреб та вимог.

2.2 Проектування WEB-ресурсу із застосуванням UML-діаграм

Проектування є невід’ємним етапом у процесі створення програмного забезпечення. Воно надає опис майбутнього продукту, включаючи його структуру, функціональність та взаємодію між компонентами. Для проектування WEB-ресурсу для обміну книгами буде використано UML-діаграми.

UML (Unified Modeling Language) - це стандартизована мова моделювання, яка широко застосовується в процесі розробки. UML-діаграми надають можливість представити структуру класів, послідовність взаємодій між об'єктами та багато іншого. Використання таких діаграм під час проектування забезпечує чіткість та зрозумілість для всіх учасників розробки, а також полегшує комунікацію між ними. Вони допомагають створити детальний план проекту перед початком реалізації, що дозволяє заощадити час та ресурси, а також зменшити ймовірність виникнення потенційних помилок у процесі розробки [12].

Для графічного представлення різних аспектів системи UML надає набір діаграм, серед найпоширеніших можна виділити такі:

1. Діаграма класів: відображає структуру системи, класи, їх атрибути, методи. Використовується для моделювання об'єктно-орієнтованих систем;
2. Діаграма прецедентів: відображає функціонал системи з точки зору користувача;
3. Діаграма послідовності: показує взаємодію між об'єктами програми в часі, фокусуючись на послідовності повідомлень;
4. Діаграма активності: описує потік керування в системі, включаючи розгалуження, паралельні дії та переходи між станами;
5. Діаграма компонентів: показує компоненти системи та їх залежності. Використовується для моделювання архітектури на високому рівні;
6. Діаграма розгортання: ілюструє фізичне розгортування системи, показуючи апаратне забезпечення та розподіл програмних компонентів на цих вузлах;
7. Діаграма станів: описує поведінку системи або її компонентів з точки зору станів, в яких вони можуть перебувати та переходів у відповідь на події.

Кожен з цих типів UML-діаграм має своє призначення та допомагає розглянути систему з різних точок зору. Вибір діаграм залежить від специфіки проекту, для проектування було обрано такі діаграми: прецедентів, послідовності та розгортання.

Діаграма прецедентів (Use Case Diagram) є однією з основних UML-діаграм, яка використовується для моделювання функціональних вимог до системи. Вона дозволяє визначити, що система повинна робити з точки зору кінцевих користувачів або зовнішніх систем. Діаграма прецедентів відображає взаємодію між акторами та прецедентами, які представляють основні функції, що надаються програмою [13]. Розроблена UML-діаграма прецедентів WEB-ресурсу для обміну книгами представлена на рисунку 2.6.

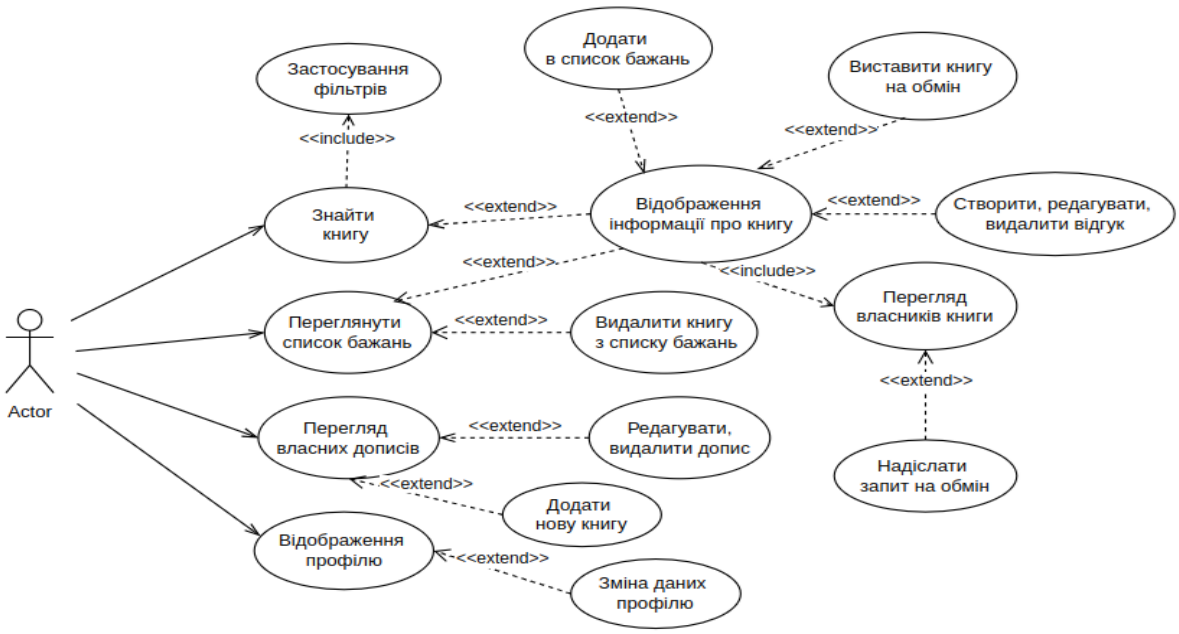

Рисунок 2.6 – UML-діаграма прецедентів WEB-ресурсу для обміну книгами

Актори (виконавці) відображають зовнішні сутності (користувачі, системи), які взаємодіють з ресурсом. Виконавці поділяються на первинні, які безпосередньо ініціюють взаємодію та вторинні - взаємодіють через первинних акторів.

Прецеденти (варіанти використання) описують функціонал системи з точки зору акторів. Вони показують послідовність дій, які виконуються програмою для досягнення певного результату та зображуються у вигляді еліпсів.

При створенні діаграми важливо дотримуватись принципу, згідно з яким прецедент повинен описувати, що треба робити, а не як. Це дозволяє зосередитися на цілях, які необхідно досягти, абстрагуючись від технічних деталей реалізації. Це допомагає визначити поведінку з точки зору користувача, сприяючи кращому розумінню вимог та спрощуючи комунікацію між замовниками та розробниками.

Зв'язки в UML-діаграмах прецедентів показують взаємодію між акторами та прецедентами, а також між самими прецедентами. Основним типом зв'язку є асоціація, яка вказує на використання прецеденту актором. Крім того, існують зв'язки розширення (extend) та включення (include). Зв'язок розширення вказує на те, що один прецедент може бути розширений функціональністю іншого прецеденту. Зв'язок включення означає, що один прецедент завжди включає в себе функціонал іншого прецеденту. Ці зв'язки допомагають організовувати діаграму, уникаючи дублювання та забезпечуючи повторне використання. Діаграма прецедентів допомагає:

- Визначити межі системи та її контекст на структурному рівні проектування;
- Сформувати загальні вимоги до поведінки системи;
- Розробити концептуальну модель системи для подальшої деталізації;
- Підготувати документацію для взаємодії замовників і користувачів системи.

Діаграма послідовності (Sequence Diagram) - це різновид UML-діаграм, призначений для моделювання взаємодії між об'єктами системи з плином часу. Вона відображає, яким чином об'єкти обмінюються повідомленнями в певній послідовності для реалізації конкретного функціоналу. Діаграма послідовності часто використовують для того, щоб показати взаємодію компонентів інтерфейсу користувача або програмного забезпечення [14]. Розглянемо основні складові діаграми послідовності:

1. Об'єкти (Objects): відображають екземпляри класів або акторів, залучених до взаємодії. Вони зображені у вигляді прямокутників з іменем та розташовуються у верхній частині діаграми;
2. Лінії життя (Lifelines): вертикальні пунктирні лінії, що відображають існування об'єкта протягом заданого проміжку часу. Кожен об'єкт на діаграмі

послідовності має свою лінію життя, яка починається від моменту створення об'єкта до моменту завершення його взаємодії;

3. Повідомлення (Messages): стрілки, які демонструють обмін даними або викликами методів між об'єктами. Повідомлення бувають синхронними (з очікуванням відповіді) або асинхронними (без очікування відповіді). Повідомлення зображуються горизонтальними стрілками, які з'єднують лінії життя об'єктів, між якими відбувається взаємодія;
4. Фокус керування (Activation boxes): прямокутники на лінії життя об'єкта, які вказують на період часу, протягом якого об'єкт активно виконує певну дію. Фокус керування відображається у вигляді вертикального прямокутника, який починається з моменту отримання повідомлення об'єктом та закінчується після завершення обробки цього повідомлення.

Представлена UML-діаграма послідовності на рисунку 2.7 показує взаємодію між користувачем та WEB-ресурсом для обміну книгами.

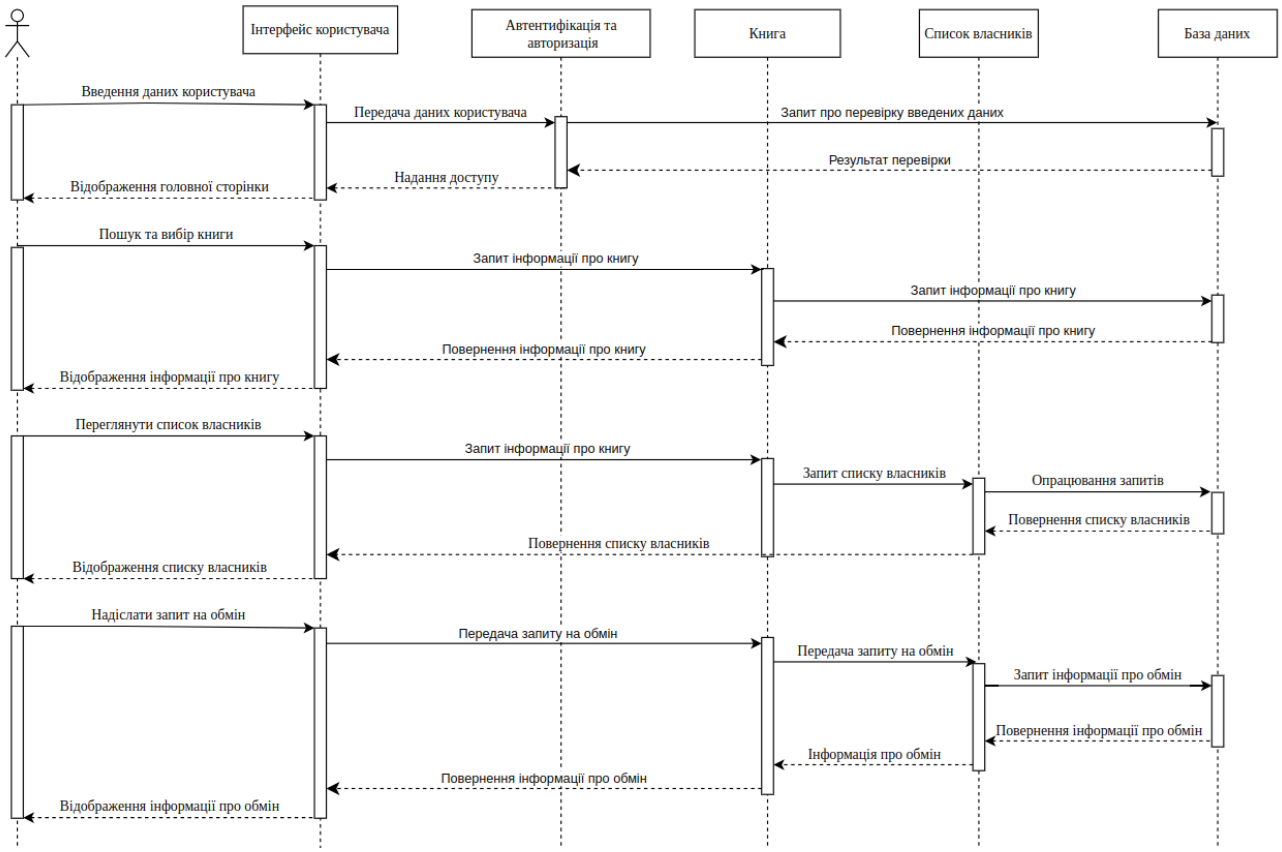

Рисунок 2.7 – UML-діаграма послідовності WEB-ресурсу для обміну книгами

Вона складається з таких елементів: інтерфейсу користувача, автентифікації та авторизації, книги, списку власників та бази даних. Кожен крок взаємодії представлений вертикальною лінією, яка йде від одного компонента до іншого, показуючи напрямок потоку даних.

Важливо відзначити, що кожен модуль системи має свої чітко визначені завдання та взаємодіє з іншими модулями та базою даних для виконання своїх функцій. Інтерфейс користувача є центральною точкою взаємодії між користувачем та системою, забезпечуючи введення даних та відображення результатів. Розглянемо більш детально кроки та функціонал, які представлені на UML-діаграмі послідовності:

Автентифікація/авторизація користувача в системі:

1. Користувач вводить свої дані через інтерфейс користувача;
2. Введені дані передаються до модуля автентифікації та авторизації;
3. Відбувається звернення до бази даних для перевірки введених даних;
4. Результат перевірки повертається модулю, який передає відповідь про допуск користувача;
5. Інтерфейс користувача відображає інформацію про допуск користувача.

Пошук та вибір книги:

1. Користувач здійснює пошук книги через інтерфейс, застосовуючи відповідні фільтри, щоб отримати більш точні результати;
2. Запит інформації про книгу передається до модуля книги;
3. Даний модуль взаємодіє з базою даних, щоб знайти відповідну інформацію;
4. Інформація повертається модулю, який передає його в інтерфейс;
5. Отримана інформація відображається користувачу через інтерфейс.

Перегляд списку власників:

1. Користувач починає перегляд списку власників книги через інтерфейс;
2. Формуються запити на отримання інформації про книгу та список власників;
3. Дані запити передаються на опрацювання до бази даних;
4. База даних повертає список власників книги;
5. Отриманий список власників відображається користувачу через інтерфейс.

Надіслати запит на обмін:

1. Користувач через інтерфейс надсилає запит на обмін книгою;
2. Запит на обмін передається до модуля книги та списку власників;
3. Відбувається обробка запиту на обмін із залученням додаткових перевірок;
4. База даних повертає результат запиту;
5. Користувач отримує інформацію про обмін через інтерфейс користувача.

Розроблена UML-діаграма послідовності дає чітке уявлення про послідовність взаємодій між користувачем та WEB-ресурсом для обміну книгами в процесі роботи з даними, демонструючи обмін повідомленнями та виконання основних операцій.

Діаграма розгортання (Deployment Diagram) - це різновид UML-діаграм, призначений для візуалізації фізичної архітектури, тобто показує, як програмні компоненти розгортаються на апаратних вузлах. В ній використовуються тривимірні позначення для представлення вузлів системи. Діаграма розгортання дозволяє розробникам зрозуміти, як система буде розгорнута в реальному середовищі та визначити необхідні ресурси для її функціонування [15].

При створенні UML-діаграми розгортання включають такі основні елементи:

1. Вузли: фізичні або віртуальні пристрої, на яких розгортаються компоненти програми. Прикладами вузлів можуть бути сервери, робочі станції, мобільні пристрої та маршрутизатори. Кожен вузол має свої характеристики: обчислювальна потужність, обсяг пам'яті, операційна система. Вони представлені на діаграмі у вигляді прямокутників з назвою всередині. Для WEB-ресурсу вузлами є WEB-сервер, сервер додатків, сервер бази даних;
2. Артефакти: фізичні компоненти системи, такі як файли, бібліотеки, скрипти, конфігураційні файли. Вони є результатом процесу розробки та розгортаються на відповідних вузлах. Артефакти згруповані в пакети або модулі для кращої організації та управління. На діаграмі зображуються у вигляді прямокутників з назвою та позначкою «artifact»;
3. Зв'язки: показують фізичні або логічні зв'язки між вузлами та компонентами системи. Вони показують мережеві з'єднання, протоколи комунікації,

інтерфейси. Зв'язки залежно від характеру комунікації між елементами поділяють на односпрямовані та двоспрямовані.

Розглянемо головні особливості UML-діаграми розгортання:

- Фізична архітектура: надає чітке уявлення про те, на яких пристроях буде працювати система;
- Незалежність від реалізації: не відображає внутрішню логіку компонентів, а лише показує, як вони розгортаються;
- Комунікація між вузлами: відображає протоколи, інтерфейси та зв'язки;

Рівень деталізації діаграми змінюється залежно від вимог певного проекту. Для складних проектів може знадобитися декілька діаграм розгортання. Розроблена UML-діаграма розгортання WEB-ресурсу для обміну книгами представлена на рисунку 2.8.

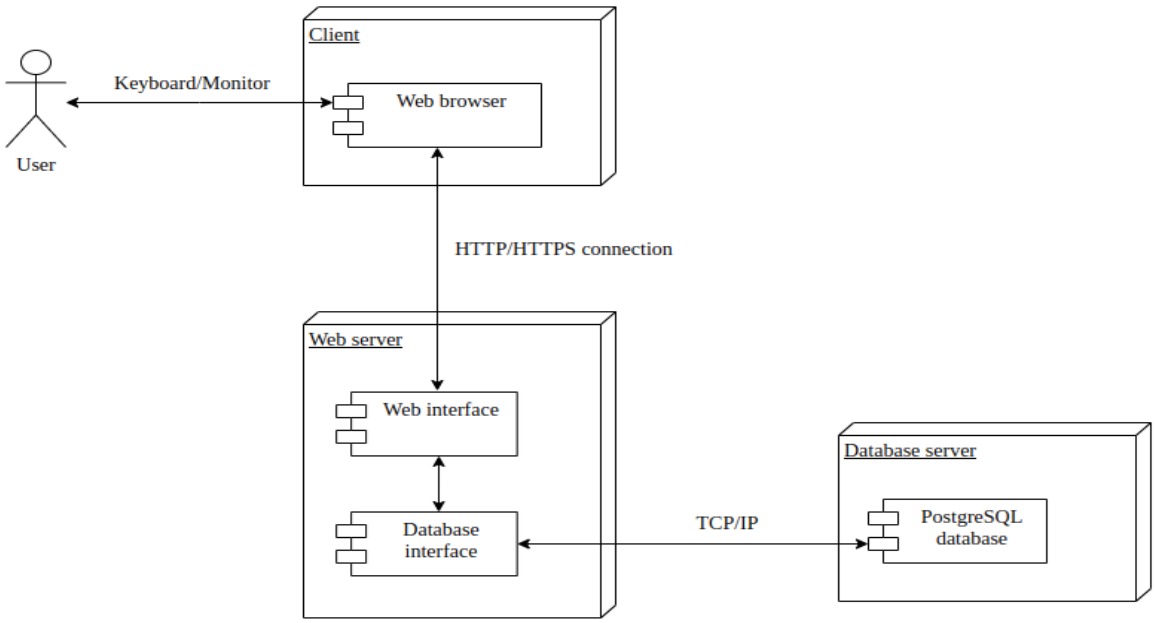

Рисунок 2.8 – UML-діаграма розгортання WEB-ресурсу для обміну книгами

Представлена діаграма розгортання відображає архітектуру WEB-ресурсу для обміну книгами, який складається з таких основних компонентів: клієнта (Client), веб-сервера (Web server) та сервера бази даних (Database server).

Користувач взаємодіє із WEB-браузером за допомогою Keyboard/Monitor. В свою чергу, WEB-браузер відправляє запити до WEB-сервера через протокол HTTP

або HTTPS, залежно від налаштувань безпеки. WEB-сервер отримує запити від клієнтів через WEB-інтерфейс та обробляє їх. Також він включає в себе інтерфейс бази даних, який відповідає за взаємодію з сервером бази даних для зберігання та отримання даних.

Сервер бази даних відповідає за зберігання та управління даними. Він включає в себе реляційну базу даних PostgreSQL та взаємодіє за допомогою протоколу TCP/IP для виконання операцій читання та запису даних.

2.3 Розробка бази даних для WEB-ресурсу

Розробка бази даних є складовою створення WEB-ресурсу для обміну книгами. При розробці бази даних необхідно враховувати модель та типи даних, структуру таблиць та безпеку даних для забезпечення правильної роботи програми.

База даних – це організований набір даних, які зберігаються та доступні в електронному вигляді. База даних використовується для зберігання, організації та управління даними. Вибір типу бази даних впливає на продуктивність, масштабованість та функціональність. Існує багато різних типів баз даних, але найбільш поширеним є поділ на дві основні категорії: реляційні (SQL) та нереляційні (NoSQL) бази даних [16].

Реляційні бази даних зберігають інформацію у вигляді структурованих таблиць з рядками (записами) та стовпцями (полями). Зв'язки між таблицями реалізуються за допомогою первинних та зовнішніх ключів, що гарантує цілісність. Для управління даними використовується мова запитів SQL.

Перевагами реляційних баз даних є структурованість, встановлення складних зв'язків, цілісність та узгодженість. Недоліками є обмежена масштабованість при обробці великих обсягів даних та необхідність попереднього визначення схеми.

Нереляційні бази даних призначені для зберігання та обробки великих обсягів неструктурованих або напівструктурованих даних. Вони не вимагають чіткого визначення схеми даних та надають можливість зберігати дані в різних форматах, таких як документи, пари ключ-значення або графи.

Перевагами нереляційних баз даних є гнучкість у зберіганні неструктурованих даних, висока масштабованість та продуктивність при роботі з великими обсягами інформації. Недоліками є відсутність чіткої схеми, обмежені можливості для забезпечення цілісності даних та складність виконання комплексних запитів.

Зважаючи на специфіку WEB-ресурсу для обміну книгами оптимальним вибором є використання реляційної бази даних. Реляційна модель даних забезпечить цілісність, узгодженість та можливість виконання складних запитів, що є важливим для програми.

ER-модель (Entity-Relationship Model) – це візуальний засіб моделювання, який використовується для відображення логічної структури бази даних. Вона дозволяє представити основні сутності предметної області, їх атрибути та зв'язки між ними [17]. Представлення ER-моделі предметної області WEB-ресурсу для обміну книгами наведено на рисунку 2.9.

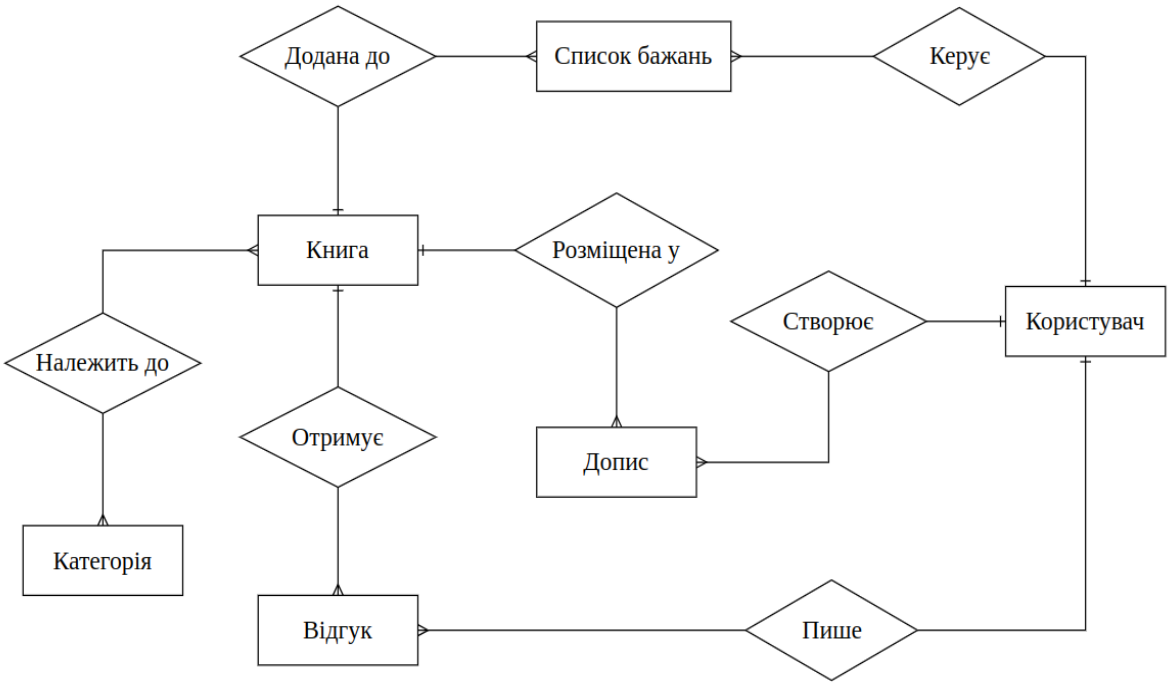

Рисунок 2.9 – ER-модель предметної області WEB-ресурсу для обміну книгами

За правилами побудови концептуальних схем предметної області у вигляді ER–структур сутності зображують позначеними прямокутниками, асоціації –

ромбами, а зв'язки між ними – ненаправленими ребрами, над якими може проставлятися ступінь зв'язку і необхідне пояснення [17].

Було спроектовано схему бази даних WEB-ресурсу для обміну книгами, яка зображена на рисунку 2.10.

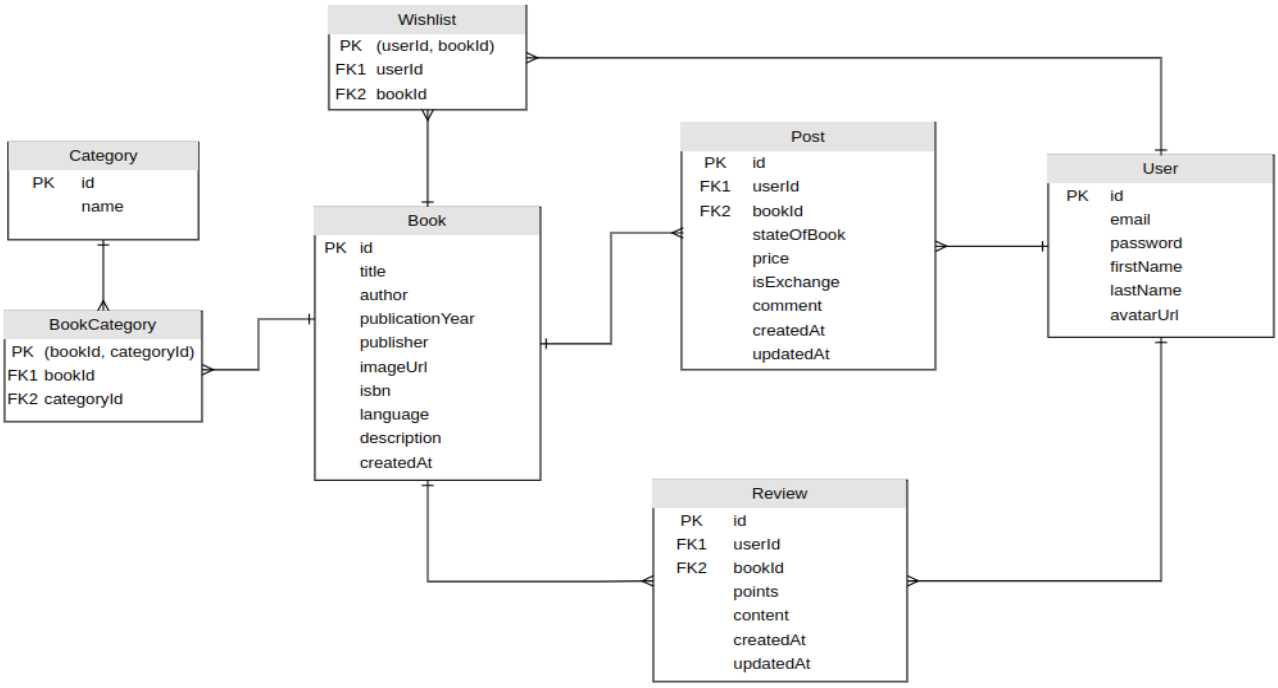

Рисунок 2.10 – Схема бази даних WEB-ресурсу для обміну книгами

Наведена схема бази даних складається з таких таблиць: category, bookCategory, wishlist, book, review, post та user. Первинні ключі позначені, як PK (Primary Key), а зовнішні ключі як FK (Foreign Key).

Для опису предметної області WEB-ресурсу для обміну книгами необхідні такі сутності: користувач, допис, список бажань, книга, відгук, категорія. Виділимо характеристики зазначених сутностей:

- користувач: ім’я, прізвище, фото, id-користувача, електронна пошта, пароль;
- допис: стан книги, ціна, можливість обміну, коментар, id-допису, дата створення та дата оновлення допису;
- список бажань: дата створення списку бажань;
- книга: назва книги, автор, рік видання, номер публікації, видавництво, обкладинка, id-книги, мова, опис;

- відгук: id-відгуку, текст, кількість балів, дата створення та оновлення відгуку;
- категорія: id-категорії, назва категорії.

Для визначення ступеню універсального відношення потрібно перерахувати всі атрибути, що описують сутності. Універсальне відношення для даної бази даних буде мати наступний вигляд: R (ім’я, прізвище, фото, id-користувача, електронна пошта, пароль, стан книги, ціна, можливість обміну, коментар, id-допису, дата створення та дата оновлення допису, дата створення списку бажань, назва книги, автор, рік видання, номер публікації, видавництво, обкладинка, id-книги, мова, опис, id-відгуку, текст, кількість балів, дата створення та дата оновлення відгуку, id-категорії, назва категорії).

Ступінь універсального відношення – 30.

Нормалізація бази даних є важливим процесом при проектуванні бази даних. Вона допомагає мінімізувати надлишковість даних, уникнути аномалій при внесенні, оновленні та видаленні даних, що зменшує ймовірність виникнення помилок.

Відповідно до літератури [18] відношення знаходиться в 1НФ, якщо на перетині кожного стовпця та кожного рядка знаходяться лише атомарні значення атрибутів і не містяться групи, що повторюються. Проаналізувавши таблиці, можна побачити, що відношення бази даних знаходяться в першій нормальній формі.

Відношення знаходиться в 2НФ, якщо виконуються обмеження 1НФ та кожен неключовий атрибут функціонально повно залежить від первинного ключа [18]. Проаналізувавши розроблену базу даних, можна зауважити, що більшість таблиць не містять складених ключів. Винятками є лише таблиці bookCategory та wishlist, проте дані таблиці не мають неключових атрибутів. Таким чином, можна стверджувати, що відношення бази даних знаходяться у другій нормальній формі.

Для приведення відношення до 3НФ необхідно забезпечити виконання вимог другої нормальної форми, а також усунути транзитивні залежності неключових атрибутів від первинного ключа. Це означає, що кожен неключовий атрибут повинен залежати лише від первинного ключа та не повинен мати залежностей від інших неключових атрибутів [18]. Оскільки в процесі аналізу таблиць розробленої

бази даних не було знайдено транзитивних залежностей неключових атрибутів, можна зробити висновок про відповідність відношень вимогам третьої нормальної форми.

2.4 Розробка схеми алгоритму роботи WEB-ресурсу для обміну книгами

Алгоритм - це скінченна послідовність чітко визначених вказівок, що описують, які дії та у якому порядку потрібно виконати для вирішення певної задачі. Існують різні способи представлення алгоритму, кожен з яких має свої особливості. Найпростішим способом є словесний опис, де кроки алгоритму описуються за допомогою природної мови та спеціальних символів. Для формального запису використовуються математичні формули. Графічна форма представлення у вигляді схем, таблиць або рисунків надає структуру алгоритму, що полегшує його розуміння та аналіз. Коли алгоритм потрібно реалізувати на комп'ютері, використовується програмна форма, яка передбачає запис алгоритму за допомогою мов програмування. Серед усіх форм представлення алгоритмів схема алгоритму є оптимальним варіантом, тому що вона забезпечує зрозумілість та легкість сприйняття [19].

В схемі алгоритму для позначення початку та кінця алгоритму використовують овал або коло, прямокутник для окремих кроків або дій, ромб вказує на умовні блоки, а паралелограм для введення або виведення даних. Стрілки з'єднують блоки в порядку їх виконання, показуючи послідовність кроків алгоритму [19].

І випадок

Алгоритм у випадку, коли користувач не зареєстрований складається з таких кроків:

Крок 1. На першому етапі відбувається запуск головного вікна;

Крок 2. Відображення форми для реєстрації;

Крок 3. Введення вхідних даних: ім’я, прізвище, пошта та пароль;

Крок 4. Якщо реєстрація успішна – перехід до кроку 5, інакше – крок 2;

Крок 5. Відображення основного меню.

II випадок

Алгоритм у випадку, коли користувач зареєстрований складається з таких кроків:

Крок 1. На першому етапі відбувається запуск головного вікна;

Крок 2. Відображення форми для входу;

Крок 3. Введення вхідних даних: пошта та пароль;

Крок 4. Якщо авторизація успішна – перехід до кроку 5, інакше – крок 2;

Крок 5. Відображення основного меню.

III випадок

Алгоритм у випадку, коли користувач обирає сторінку «Мої книги» складається з таких кроків:

Крок 1. На першому етапі відбувається запуск головного вікна;

Крок 2. Відображення форми для входу;

Крок 3. Введення вхідних даних: пошта та пароль;

Крок 4. Якщо авторизація успішна – перехід до кроку 5, інакше – крок 2;

Крок 5. Відображення основного меню;

Крок 6. Обрано сторінку «Мої книги»;

Крок 7. Завантаження книг із бази даних;

Крок 8. Користувач може додати, видалити обрану книгу.

IV випадок

Алгоритм у випадку, коли користувач обирає сторінку «Список бажань» складається з таких кроків:

Крок 1. На першому етапі відбувається запуск головного вікна;

Крок 2. Відображення форми для входу;

Крок 3. Введення вхідних даних: пошта та пароль;

Крок 4. Якщо авторизація успішна – перехід до кроку 5, інакше – крок 2;

Крок 5. Відображення основного меню;

Крок 6. Обрано сторінку «Список бажань»;

Крок 7. Відображення списку бажань;

Крок 8. Користувач може видалити книгу із списку.

Схема алгоритму роботи WEB-ресурсу для обміну книгами наведена на рисунку 2.11.

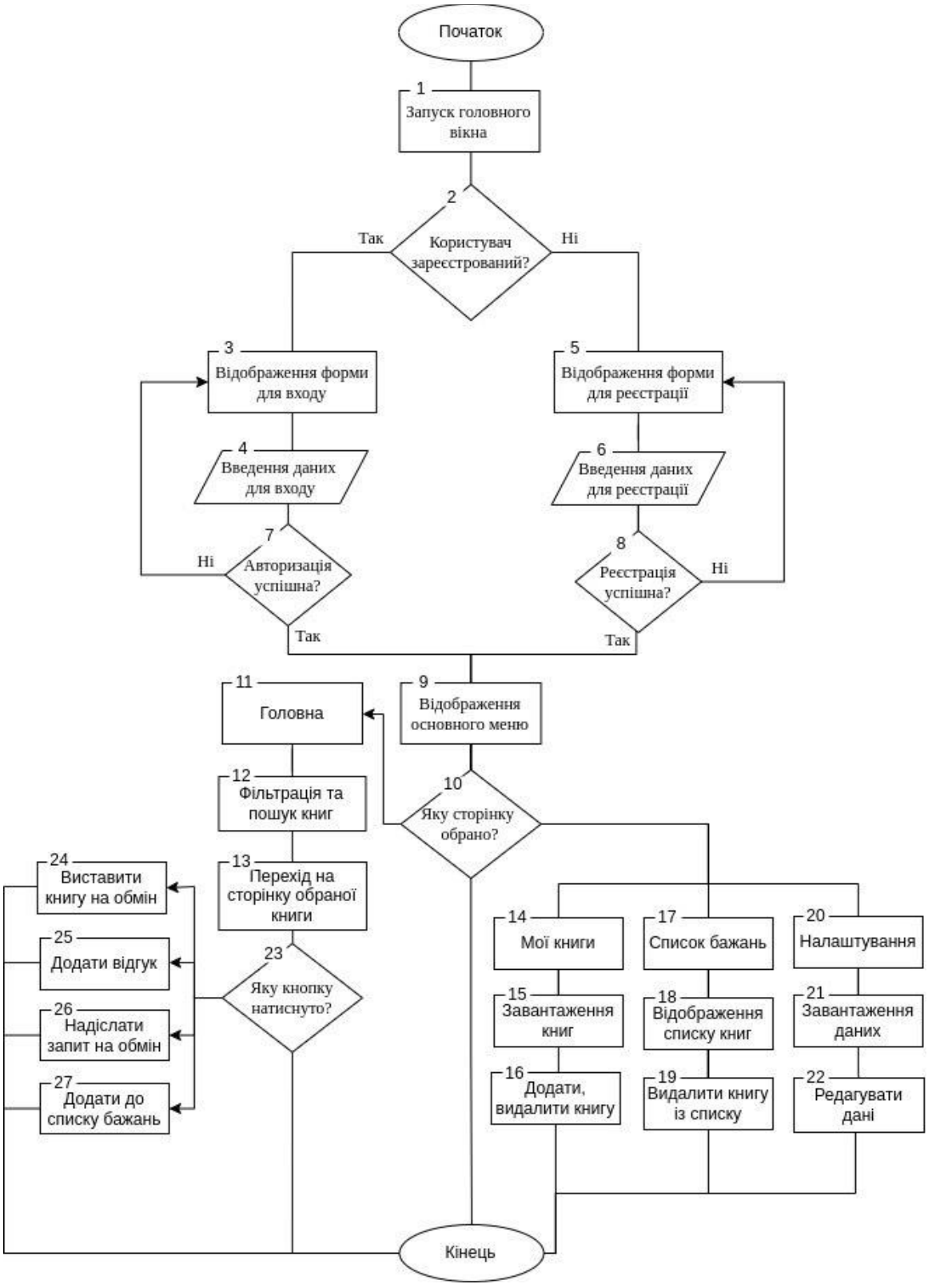

Рисунок 2.11 – Схема алгоритму роботи WEB-ресурсу для обміну книгами

V випадок

Алгоритм у випадку, коли користувач обирає сторінку «Налаштування» складається з таких кроків:

Крок 1. На першому етапі відбувається запуск головного вікна;

Крок 2. Відображення форми для входу;

Крок 3. Введення вхідних даних: пошта та пароль;

Крок 4. Якщо авторизація успішна – перехід до кроку 5, інакше – крок 2;

Крок 5. Відображення основного меню;

Крок 6. Обрано сторінку «Налаштування»;

Крок 7. Завантаження даних;

Крок 8. Редагування даних профілю.

VI випадок

Алгоритм у випадку, коли користувач обирає сторінку «Головна» складається з таких кроків:

Крок 1. На першому етапі відбувається запуск головного вікна;

Крок 2. Відображення форми для входу;

Крок 3. Введення вхідних даних: пошта та пароль;

Крок 4. Якщо авторизація успішна – перехід до кроку 5, інакше – крок 2;

Крок 5. Відображення основного меню;

Крок 6. Обрано сторінку «Головна»;

Крок 7. Фільтрація та пошук книг;

Крок 8. Перехід на сторінку обраної книги;

Крок 9. Завантаження даних;

Крок 10. Користувач натискає кнопку «Виставити книгу на обмін»;

Крок 11. Заповнення форми та відправка даних;

Крок 12. Користувач натискає кнопку «Додати відгук»;

Крок 13. Написати відгук та вказати кількість балів;

Крок 14. Натиснути на кнопку «Зберегти відгук»;

Крок 15. Користувач натискає кнопку «Надіслати запит на обмін»;

Крок 16. Відправка запиту на обмін книгою;

Крок 17. Користувач натискає кнопку «Додати до списку бажань»;

Крок 18. Відправка запиту до бази даних;

Крок 19. Відображення користувачеві повідомлення про успішне виконання запиту та повернення до головного меню.

2.5 Висновок до розділу 2

У даному розділі було проведено проектування WEB-ресурсу для обміну книгами. Було обґрунтовано вибір клієнт-серверної архітектури, яка забезпечить масштабованість, безпеку та централізоване управління даними. Ця архітектура дозволить розділити функції між клієнтом та сервером, а також сприятиме подальшому розвитку та адаптації WEB-ресурсу до потреб користувачів.

В ході проектування було розроблено UML-діаграми, зокрема діаграму прецедентів для визначення функціональних вимог, діаграму послідовності для відображення взаємодії об'єктів у часі та діаграму розгортання для представлення фізичної архітектури системи. Дані діаграми дозволили краще зрозуміти структуру та логіку роботи WEB-ресурсу.

Для організації та зберігання даних було обрано реляційну модель бази даних. У процесі проектування було визначено сутності, описано їхні характеристики та розроблено ER-модель, яка відображає структуру бази даних та зв'язки між сутностями. Також було створено схему бази даних, що включає таблиці, поля, зв'язки та обмеження.

Крім того, було розроблено схему алгоритму, яка відображає послідовність дій, процесів та взаємодій під час роботи WEB-ресурсу для обміну книгами. Ця схема охоплює різні аспекти функціонування WEB-ресурсу та є основою для подальшої реалізації програмного коду.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-РЕСУРСУ ДЛЯ ОБМІНУ КНИГАМИ

3.1 Обґрунтування вибору середовища розробки

Вибір середовища розробки є одним з етапів розробки програмного забезпечення. При виборі необхідно враховувати такі критерії, як підтримка мов програмування, функціональність редактора коду, можливості відлагодження та тестування, інтеграція з системами контролю версій. Правильно підібране середовище програмування здатне значно підвищити продуктивність розробки та якість майбутнього продукту.

Visual Studio Code (VS Code) – це безкоштовний редактор коду, розроблений компанією Microsoft, який підтримує широкий спектр мов програмування. Завдяки цьому, розробник має змогу працювати над різними проектами та використовувати різні технології в єдиному середовищі, що значно спрощує процес розробки [20].

Однією з основних переваг VS Code є потужний та інтуїтивно зрозумілий інтерфейс користувача. Він забезпечує підсвічування синтаксису, автоматичне доповнення, навігацію по коду та інші корисні функції, що полегшують роботу.

Також він має вбудований менеджер додатків, який дозволяє розширювати його функціональність за допомогою різноманітних плагінів. Завдяки великій спільноті розробників, доступний широкий вибір для додавання нових можливостей, покращення інтерфейсу та інтеграції із зовнішніми інструментами. Це забезпечує гнучкість у налаштуванні середовища розробки відповідно до потреб проекту.

Інтеграція з системами контролю версій є ще однією перевагою даного середовища. Вбудована підтримка Git дозволяє керувати репозиторіями, відстежувати зміни, створювати гілки та здійснювати коміти безпосередньо з редактора. Це значно спрощує процес роботи з версіями коду та полегшує колективну розробку в команді.

Крім того, для ефективного налагодження коду VS Code надає вбудований відлагоджувач. Він дозволяє встановлювати точки зупинки, переглядати значення

змінних, покрокове виконання коду та аналіз поведінки програми. Даний інструмент є надзвичайно корисним для виявлення та усунення помилок у коді, що сприяє розробці надійного та якісного програмного забезпечення. Головна сторінка Visual Studio Code зображена на рисунку 3.1.

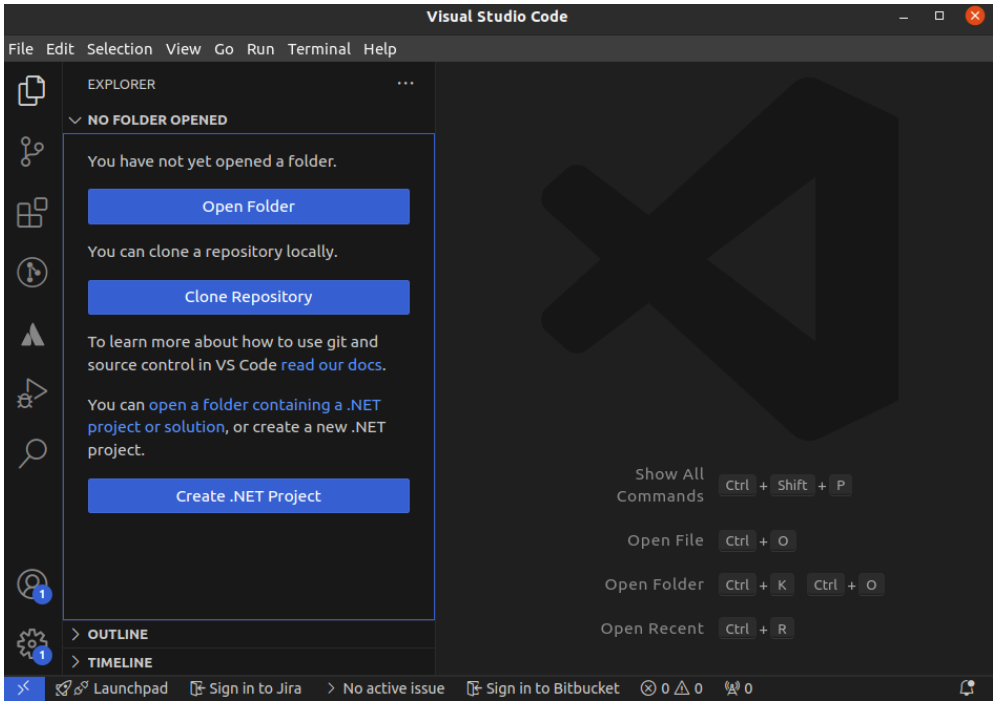

Рисунок 3.1 – Головна сторінка Visual Studio Code

Отже, Visual Studio Code обрано як середовище для розробки WEB-ресурсу для обміну книгами. Воно надає зручний та інтуїтивний інтерфейс користувача, вбудований менеджер додатків та відлагоджувач, а також інтеграцію з системами контролю версій.

3.2 Обґрунтування вибору мови та бібліотек програмування

Вибір мови програмування є важливим етапом в розробці програмного забезпечення. Для аналізу та порівняння було обрано такі популярні мови програмування: JavaScript, Python, Java, C++.

JavaScript – це високорівнева, інтерпретована, динамічно типізована мова програмування. Спочатку була розроблена для додавання інтерактивності до WEB-

сторінок, проте зараз її застосування набагато ширше, а саме застосунки, WEB-додатки, ігри, серверні сценарії, машинне навчання та мобільна розробка [21].

Однією з основних переваг JavaScript є універсальність та підтримка всіма сучасними WEB-браузерами. Вона надає можливість створення інтерактивних WEB-ресурсів за допомогою анімацій, валідації форм та обробки подій. Крім того, вона дозволяє розробку, як на клієнтській стороні, так і на серверній. Основним недоліком JavaScript є відсутність статичної типізації, що призводить до помилок на етапі виконання, проблеми безпеки, через виконання на стороні клієнта та нестабільність стандартів між різними браузерами, що ускладнює підтримку сумісності коду.

Python – це високорівнева, об’єктно-орієнтована мова програмування загального призначення. Вона має динамічну типізацію та автоматичне керування пам'яттю, що полегшує процес написання коду. Python широко використовується у WEB-розробці, машинному навчанні, обчисленнях та під час аналізу даних [22].

Однією з головних переваг Python є його простота та зручність у використанні, що робить його популярним серед новачків. Він підтримує кросплатформеність, кілька парадигм програмування, включаючи процедурне, функціональне та об'єктно-орієнтоване. Недоліком є повільна швидкість виконання в порівнянні з компільованими мовами. Динамічна типізація ускладнює виявлення помилок до виконання програми. Також він потребує багато оперативної пам'яті, що є проблемою для великих проектів.

Java – це об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems. Її розробка була спрямована на створення платформонезалежної мови програмування з високим рівнем безпеки. Вона широко використовується для розробки WEB-додатків, мобільних додатків та корпоративних систем [23].

Однією з головних переваг є її платформонезалежність. Це означає, що програми можуть працювати на будь-якій платформі, що підтримує Java Virtual Machine (JVM). Вбудовані механізми безпеки контролю доступу та шифрування роблять її безпечною мовою для розробки мережевих додатків. Одним із недоліків Java є низька продуктивність у порівнянні з мовами, які компілюються

безпосередньо в машинний код. Виконання програм на Java є повільнішим через додатковий рівень абстракції JVM. Вона має складний синтаксис та присутня складність при налаштуванні середовища розробки. Крім того, великий розмір скомпільованих файлів та вимоги до пам'яті ускладнюють розробку великих проектів та вимагають додаткових зусиль для оптимізації.

C++ – це мова програмування загального призначення, яка є розширенням мови C. Вона підтримує як процедурне, так і об'єктно-орієнтоване програмування, що дозволяє створювати складні програми. Крім того, мова надає низькорівневий доступ до пам'яті та можливість роботи з апаратним забезпеченням [23].

Однією з основних переваг C++ є висока продуктивність. Вона забезпечує високий рівень контролю над апаратними ресурсами, що робить її корисною для системного програмування. Крім того, має можливості для роботи з об'єктами, шаблонами, бібліотеками, що забезпечує гнучкість та модульність коду. Однак C++ вважається складною мовою для навчання. Вона вимагає ретельного керування пам'яттю та відстеження ресурсів, що може призвести до помилок, витоків пам'яті та інших проблем з безпекою. В таблиці 3.1 наведено порівняльну характеристику мов програмування.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Характеристика	JavaScript	Python	Java	C++
Швидкодія	+	-	+	+
Зрозумілість коду	+	+	+	+
Динамічна типізація	+	-	-	-
Підтримка ООП	-	+	+	+
Платформонезалежність	+	+	+	-
Велика кількість бібліотек	+	+	+	+
Розробка WEB-ресурсів	+	+	-	-

Проаналізувавши різні мови програмування та їхні особливості, для розробки WEB-ресурсу для обміну книгами було обрано JavaScript. Ця мова дозволяє

створювати як серверну частину з використанням Node.js, так і клієнтську частину для браузера, застосовуючи сучасні бібліотеки та фреймворки для ефективної розробки.

Для розробки клієнтської частини було використано React. React - це популярна JavaScript бібліотека із відкритим вихідним кодом для створення користувацьких інтерфейсів. Основними перевагами є компонентний підхід, віртуальний DOM, використання JSX та екосистема додаткових бібліотек. Він забезпечує високу продуктивність, підтримку та можливість створювати масштабовані користувацькі інтерфейси [24]. Для управління станом WEB-ресурсу використовувалась бібліотека Redux, яка створена на основі Flux. Схема роботи Redux зображена на рисунку 3.2.

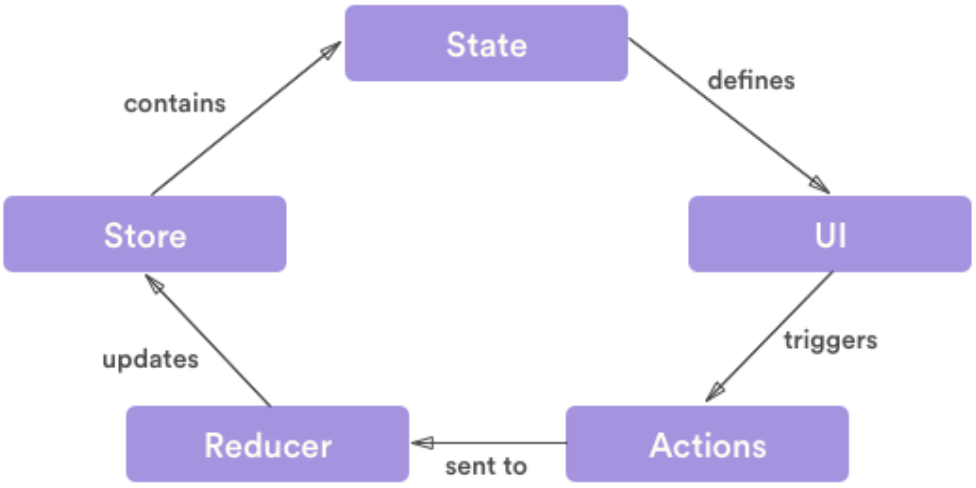

Рисунок 3.2 – Схема роботи Redux для управління станом WEB-ресурсу

Як бачимо з наведеного рисунку, об'єкт store зберігає єдиний стан (state) усього WEB-ресурсу. Користувацький інтерфейс (UI) підписується на цей стан та оновлюється при його змінах. Зміни стану відбуваються виключно через відправлення дій (actions). Дії викликаються спеціальними функціями-створювачами на основі взаємодій користувача або інших подій в програмі. Reducer є чистою функцією, яка обробляє відправлені дії та попередній стан, повертаючи новий стан. Таким чином, Redux забезпечує однонаправлений потік

даних та передбачувану поведінку стану, полегшуючи розробку програмного забезпечення.

Для зручності розробки було використано бібліотеку redux-logger. Вона виконує логування стану у консолі браузера. Це дозволяє легко переглядати послідовність виконаних дій, попередні та нові стани, що полегшує налагодження програми. Приклад роботи логування за допомогою redux-logger зображено на рисунку 3.3.

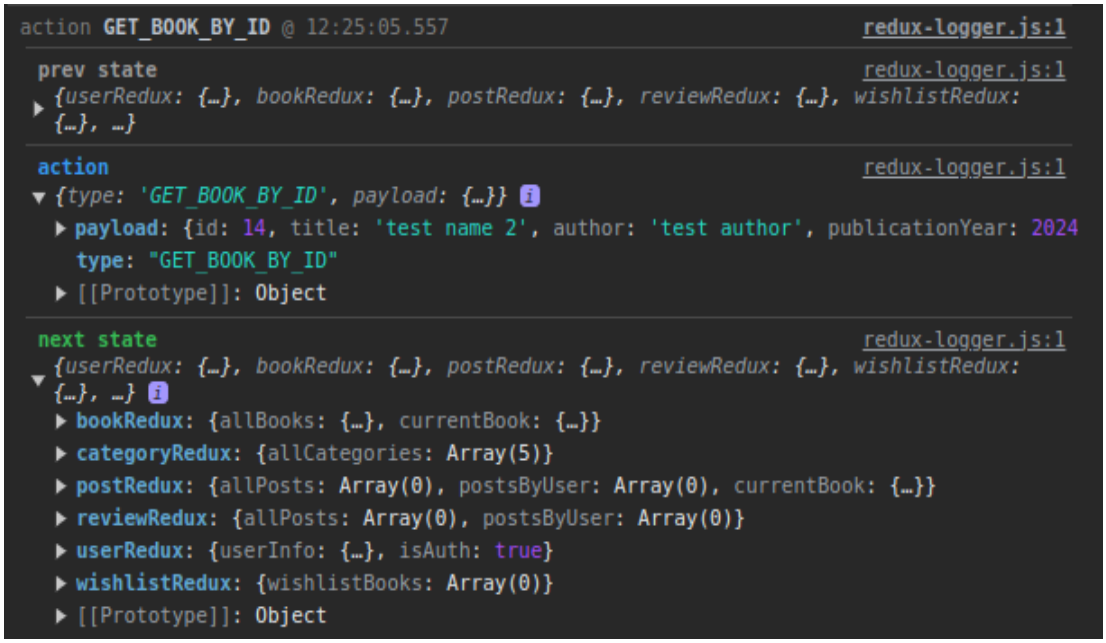

Рисунок 3.3 – Приклад роботи логування за допомогою redux-logger

Material-UI - це популярна бібліотека компонентів для React, яка надає широкий вибір компонентів. Це включає кнопки, тексти, таблиці, картки, модальні вікна та багато інших. Кожен компонент уже стилізований та готовий до використання, що спрощує розробку та дозволяє зосередитися на функціональності програми [25].

Material-UI надає можливість налаштування компонента за допомогою пропсів, що дозволяє розробникам адаптувати їх під свої потреби та дизайн. Крім того, бібліотека пропонує систему тем, яка дозволяє легко змінювати кольори та стилі. Ще однією перевагою є активна спільнота розробників та постійне оновлення. Використання Material-UI дозволяє швидко створювати стильні та

функціональні інтерфейси, що робить її ідеальним вибором для розроблюваного WEB-ресурсу.

Була використана бібліотека React Hook Form, яка надає легкий спосіб керування формами. Використання хуків дозволяє поєднати логіку форм у функціональні компоненти. Вона інтегрується з бібліотекою Yup, яка містить безліч функцій для валідації даних, а також дозволяє створювати нові, як зображено на рисунку 3.4.

```
yup.addMethod(yup.string,'hasLowerCase',function (number) {
  return this.test({
    name: 'hasLowerCase',
    message: 'password.hasLowerCase',
    params: {
      number,
    },
    test: (value = '') => /[a-z]/.test(value),
  });
});
```

Рисунок 3.4 – Створення нового методу для перевірки даних

Серверна частина була розроблена за допомогою гнучкого фреймворку для Node.js, а саме Express.js. Він забезпечує мінімалістичну структуру для розробки серверної логіки, дозволяючи легко керувати маршрутизацією, обробляти запити та відповіді, а також інтегруватися з різними middleware для обробки даних, автентифікації, логування та інших завдань [26].

Express.js скорочує час розробки API. Крім того, він добре інтегрується з іншими популярними технологіями, що робить його універсальним інструментом для сучасної WEB-розробки. Для обробки завантаження файлів на сервері було використано Multer, middleware для Node.js. Multer значно спрощує отримання файлів з HTML форм та дозволяє зберігати їх на сервері або в базі даних.

Multer працює з формами типу «multipart/form-data», які використовуються для завантаження файлів. Вона дозволяє налаштовувати місце зберігання файлів,

їх імена, а також обмеження на розмір та тип. На рисунку 3.5 представлено налаштування Multer.

```
const multer = require('multer');

const storage = multer.diskStorage({
  destination(cb) {
    cb(null,'images/')
  },
  filename(file,cb) {
    cb(null,new Date().toISOString() + '-' + file.originalname)
  }
})

const types = ['image/png','image/jpeg','image/jpg']

const fileFilter = (file,cb) => {
  if (types.includes(file.mimetype)) {
    cb(null,true)
  }
  else {
    cb(null,false)
  }
}

module.exports = multer({ storage,fileFilter })
```

Рисунок 3.5 – Налаштування Multer для завантаження зображень

В представленому коді спочатку визначається місце зберігання файлів (storage) з вказівкою директорії та формату імені файлів. Потім задається фільтр файлів (fileFilter), який приймає зображення лише з певними типами. У кінці експортується конфігурація Multer з налаштуваннями зберігання та фільтрування файлів.

Для підвищення безпеки при реєстрації та вході користувачів використовуються бібліотеки bcrypt та jsonwebtoken. Під час реєстрації пароль користувача хешується за допомогою bcrypt та зберігається у безпечному вигляді. А jsonwebtoken (JWT) використовується для створення та перевірки токенів автентифікації. Після успішного входу сервер генерує токен JWT, який містить інформацію про користувача та підписується секретним ключем. Токен відправляється клієнту й використовується для перевірки автентифікації

користувача при наступних запитах до сервера. Реалізація функцій реєстрації та входу користувача зображена на рисунку 3.6.

```
async registration(req,res,next) {
  const { firstName,lastName,email,password } = req.body
  const candidate = await User.findOne({ where: { email } })
  if (candidate) {
    return next(ApiError.badRequest('Користувач з такою поштою вже існує'))
  }
  const hashPassword = await bcrypt.hash(password,5)
  const user = await User.create({ firstName,lastName,email,password: hashPassword })
  const token = generateJwt(user.id,user.email)
  return res.json({ token,id: user.id })
}

async login(req,res,next) {
  const { email,password } = req.body
  const user = await User.findOne({ where: { email } })
  if (!user) {
    return next(ApiError.internal('Користувач не знайдений'))
  }
  let comparePassword = bcrypt.compareSync(password,user.password)
  if (!comparePassword) {
    return next(ApiError.internal('Перевірте пошту або пароль'))
  }
  const token = generateJwt(user.id,user.email)
  return res.json({ token,id: user.id })
}
```

Рисунок 3.6 – Реалізація функцій реєстрації та входу користувача

При реєстрації проводиться перевірка на наявність користувача з вказаною електронною поштою, хешується пароль за допомогою bcrypt, створюється новий користувач та генерується для нього JWT токен. При вході перевіряється наявність користувача, далі порівнюється введений пароль з хешованим паролем, який зберігається у базі даних, якщо введені дані правильні, то створюється токен.

3.3 Обґрунтування вибору СУБД та ORM

Вибір системи управління базами даних (СУБД) є важливим етапом при розробці. Проаналізуємо дві реляційні бази даних: PostgreSQL та MySQL.

PostgreSQL є потужною, об'єктно-реляційною СУБД з відкритим вихідним кодом. Вона підтримує розширені типи даних, включаючи JSON, XML та просторові дані. PostgreSQL відома відповідністю стандартам SQL та можливістю роботи з великими обсягами даних, а також підтримує ACID-транзакції [27].

MySQL – це ще одна популярна реляційна система управління базою даних. Вона відома своєю простотою, продуктивністю та широким поширенням. MySQL підтримує реплікацію та кластеризацію, робить її гнучкою для різних сценаріїв використання [28].

PostgreSQL пропонує кращу підтримку складних транзакцій та відповідність стандартам SQL. Також надійність та консистентність даних є важливими факторами і тут вона також має перевагу. Проаналізувавши описані СУБД, було обрано PostgreSQL через її масштабованість, відповідність стандартам, надійність та продуктивність.

ORM (Object-Relational Mapping) – це методологія програмування, що дозволяє перетворювати дані між об'єктами та реляційними базами даних. Вона полегшує роботу, тому що розробники будуть використовувати об'єкти замість SQL-запитів [29]. Схема роботи ORM, яка використана при розробці зображена на рисунку 3.7.

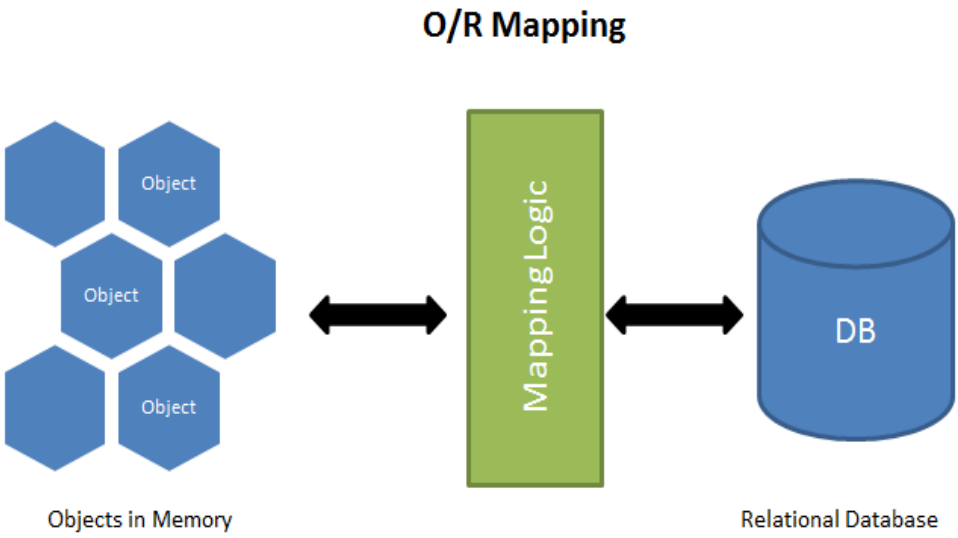

Рисунок 3.7 – Схема роботи ORM (Object-Relational Mapping)

Sequelize - це популярна ORM для Node.js, яка підтримує роботу з різними СУБД. Вона забезпечує зручний інтерфейс для роботи з ними за допомогою об'єктів та моделей. Sequelize було обрано завдяки здатності легко інтегруватися з PostgreSQL. Особливостями є підтримка асоціацій, автоматичне створення таблиць

та вбудована валідація даних [30]. Приклад створення моделі користувача зображено на рисунку 3.8.

```
const sequelize = require('../db')
const { DataTypes } = require('sequelize')

const User = sequelize.define('user',{
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  email: {
    type: DataTypes.STRING,
    unique: true
  },
  password: {
    type: DataTypes.STRING
  },
  firstName: {
    type: DataTypes.STRING
  },
  lastName: {
    type: DataTypes.STRING
  },
  avatarUrl: {
    type: DataTypes.STRING,
    allowNull: true,
  },
})
```

Рисунок 3.8 – Приклад створення моделі користувача

У наведеному коді створюється модель User з використанням Sequelize. Вона описує структуру таблиці, де кожна властивість відповідає стовпцю з вказаним типом даних та додатковими параметрами, такими як первинний ключ, унікальність. Ця модель буде відображена як таблиця в базі даних при синхронізації.

3.4 Тестування роботи програми та аналіз результатів

Тестування є важливою складовою розробки WEB-ресурсу для обміну книгами, що дозволяє виявити та виправити помилки, перевірити відповідність функціоналу вимогам та забезпечити належну роботу програми.

Для реєстрації користувачу потрібно вказати ім’я, прізвище, пошту та пароль, а для входу лише пошту та пароль. Модальне вікно для реєстрації користувача зображено на рисунку 3.9, а для входу на рисунку 3.10.

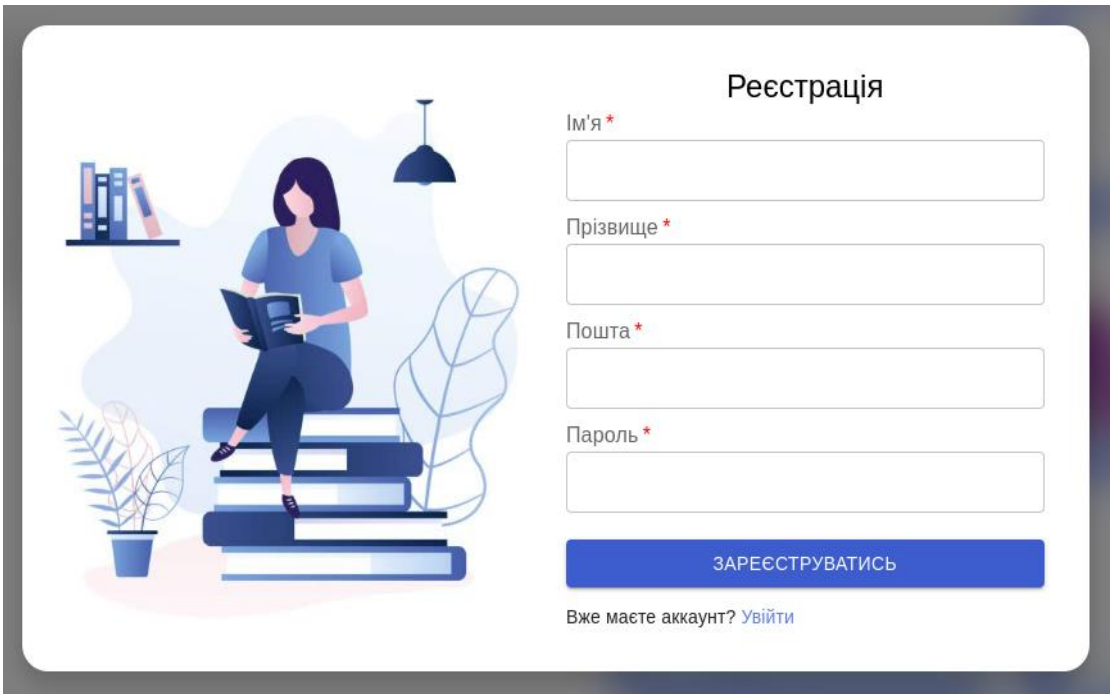

Рисунок 3.9 – Модальне вікно для реєстрації користувача

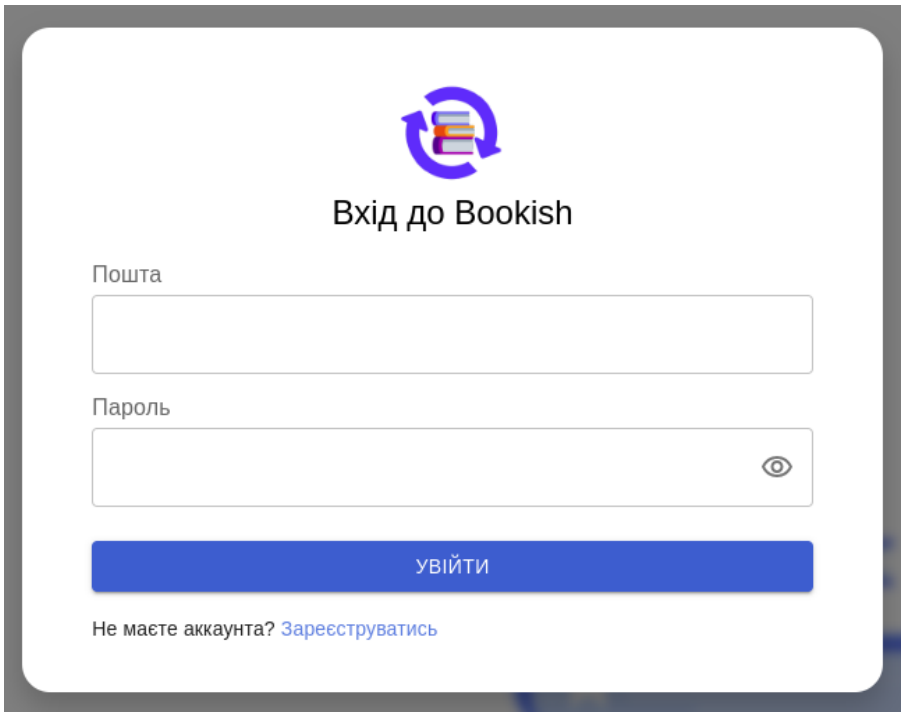

Рисунок 3.10 – Модальне вікно для входу користувача

Після успішного входу відображається головна сторінка, де користувач має змогу бачити список книг, який можна фільтрувати за категоріями та мовою видання. Крім того, реалізовано зручний пошук за назвою, автором або унікальним ідентифікатором книги. Головна сторінка WEB-ресурсу для обміну книгами зображена на рисунку 3.11.

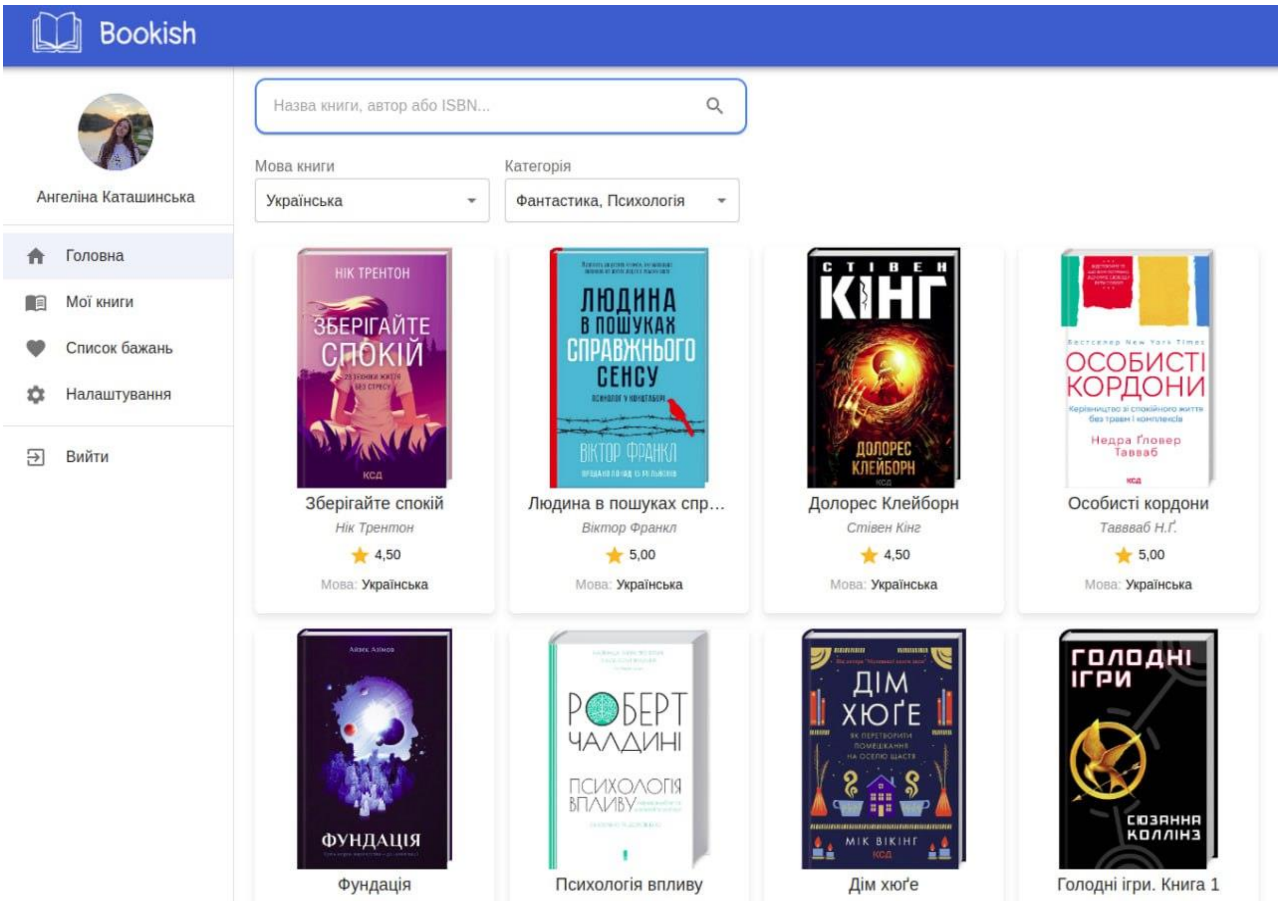

Рисунок 3.11 – Головна сторінка WEB-ресурсу для обміну книгами

Натиснувши на обрану книгу, користувач переходить на окрему сторінку (рис. 3.12), де відображаються важливі деталі книги: назва, автор, видавництво, рік випуску, мова, унікальний номер та опис. Окрема секція показує список користувачів, готових обміняти чи продати цю книгу із вказаними ними умовами. В користувача також є можливість переглянути коментар власника книги та натиснути кнопку «Надіслати запит». В разі успішного надсилання запиту на обмін книгою користувач побачить повідомлення (рис. 3.13). А власник книги отримає повідомлення (рис. 3.14) на вказану електронну пошту від WEB-ресурсу.

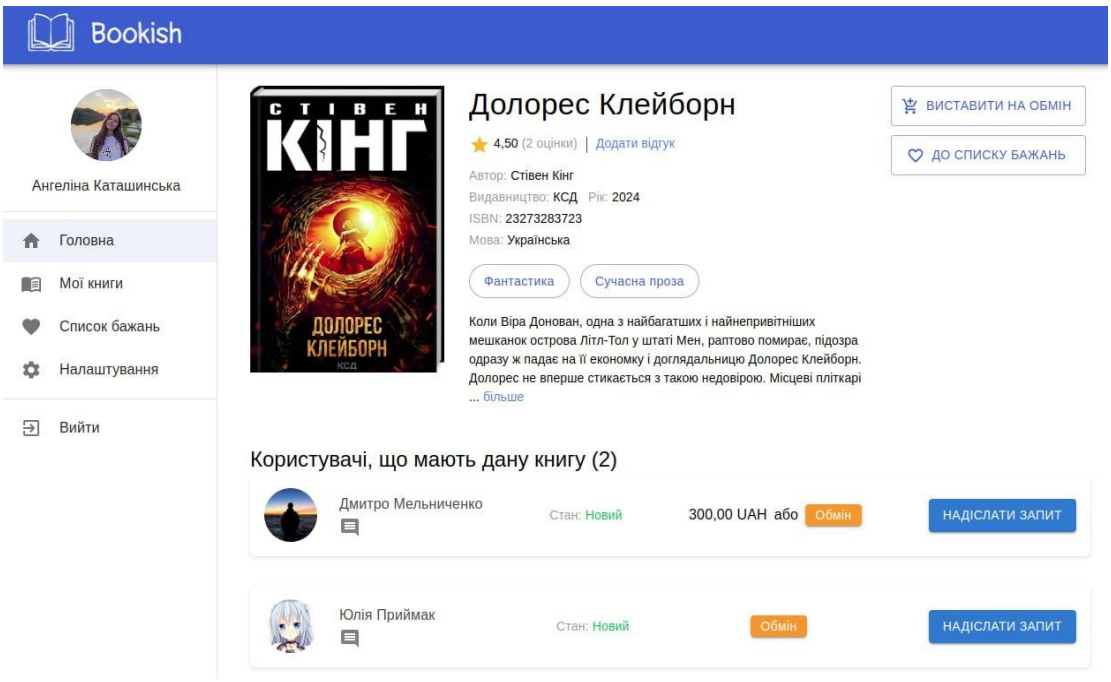

Рисунок 3.12 – Сторінка книги

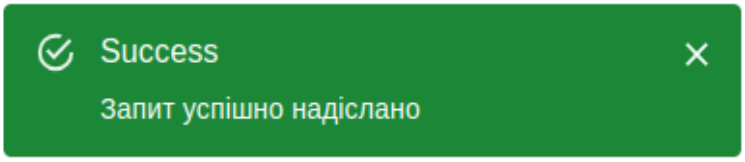

Рисунок 3.13 – Повідомлення про успішне надсилання запиту

Рисунок 3.14 – Повідомлення про запит на обмін книгою

Нижче зображена секція відгуків (рис. 3.15), яка дозволяє ознайомитися з відгуками та оцінкою книги від інших користувачів. А також можна залишити свій відгук (рис. 3.16), редагувати або видалити його за потреби.

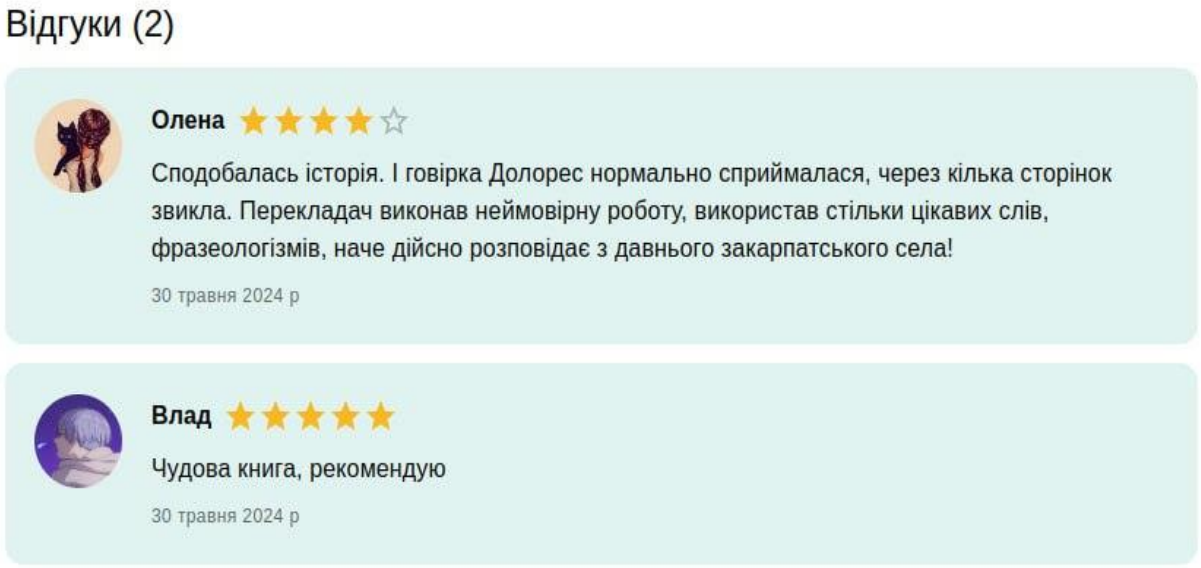

Рисунок 3.15 – Відгуки користувачів

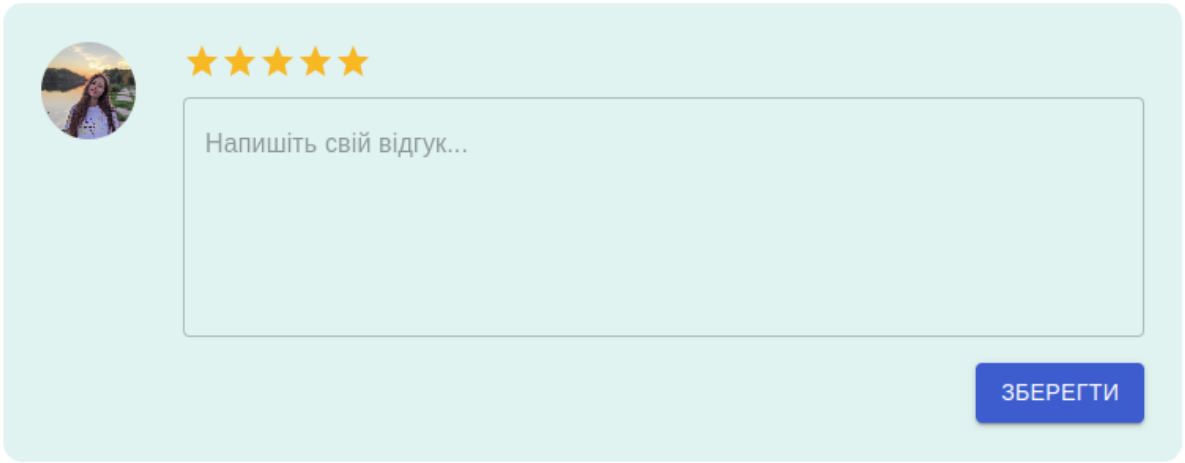

Рисунок 3.16 – Створення відгуку

Також на сторінці книги розміщені кнопки для додавання книги до списку бажань або виставлення її на обмін. Виставлення книги на обмін чи продаж зображено на рисунку 3.17. Виставлені книги відображені в книжковій поличці, яка знаходиться на сторінці «Мої книги», яка зображена на рисунку 3.18.

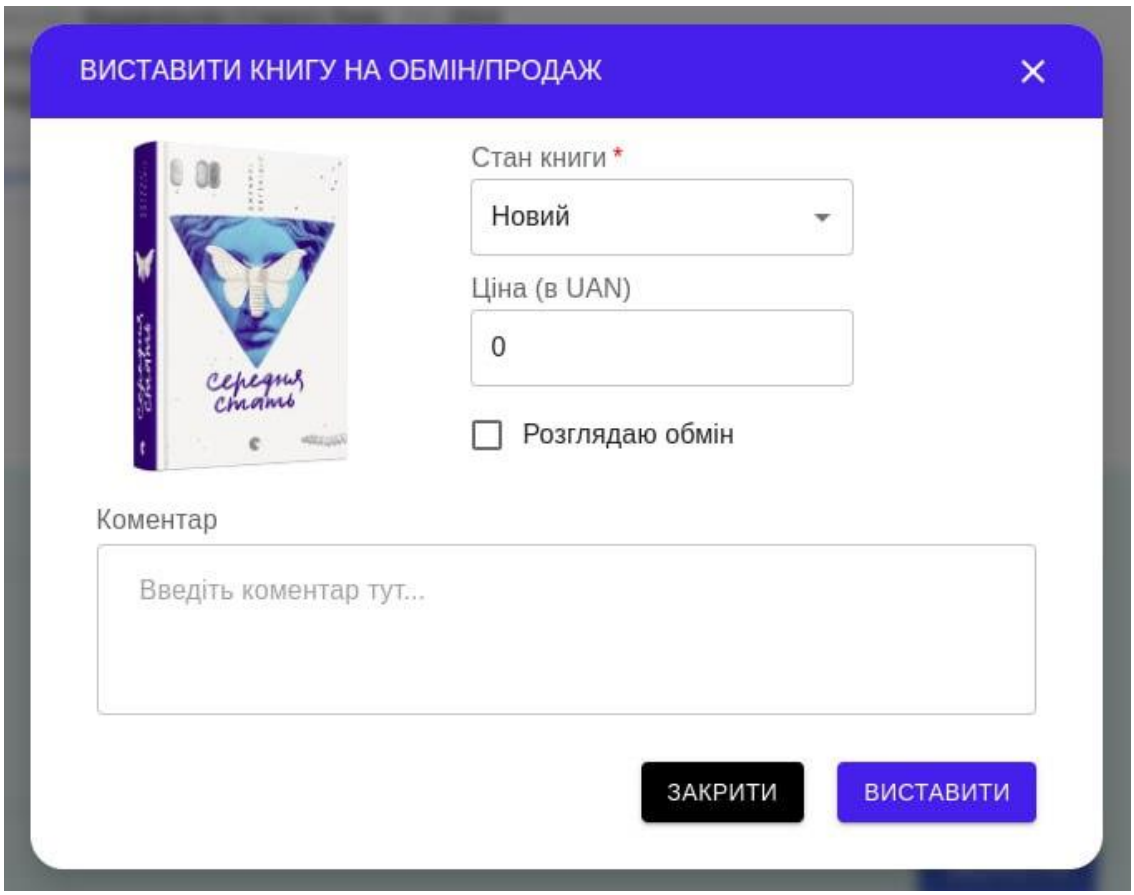

Рисунок 3.17 – Виставлення книги на обмін

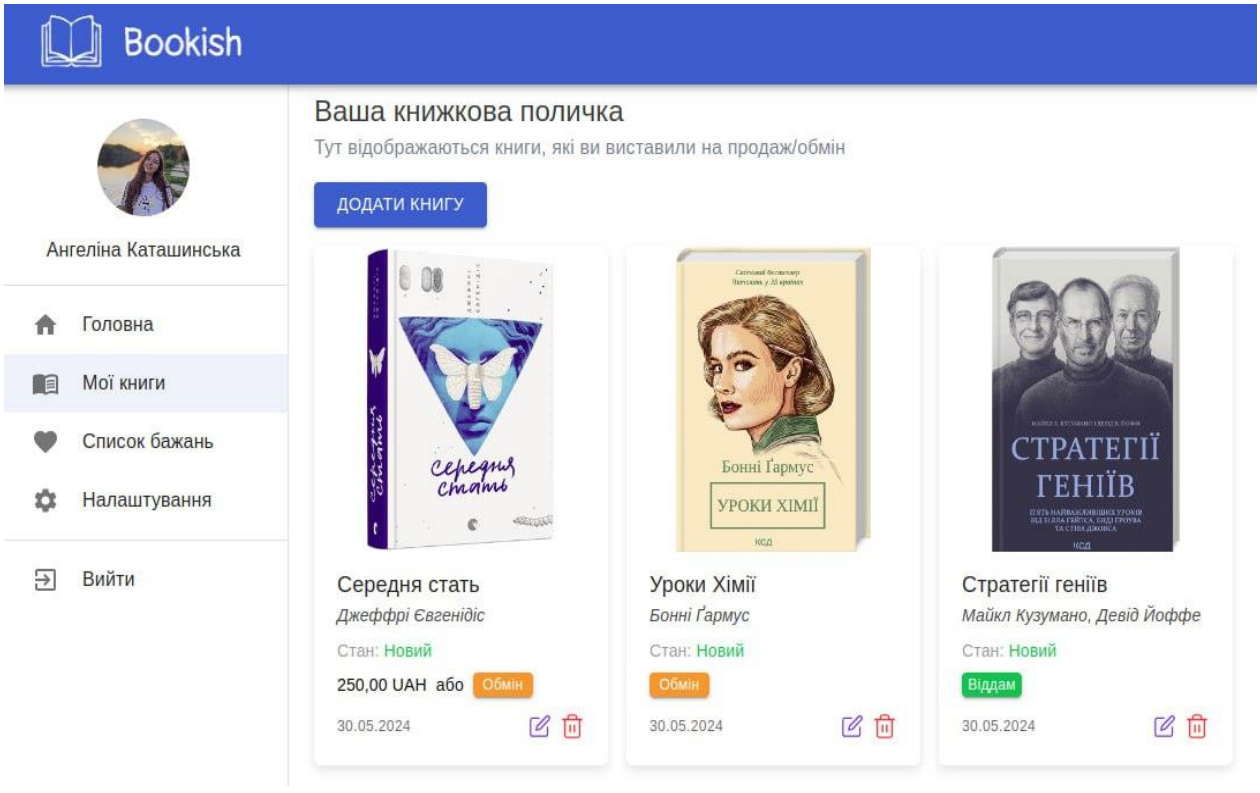

Рисунок 3.18 – Сторінка «Мої книги»

Якщо потрібної книги немає в базі даних WEB-ресурсу для обміну книгами, користувач має можливість додати її самостійно. На наведеному вище зображенні видно кнопку «Додати книгу» при натисканні якої відкривається форма для додавання нової книги (рис. 3.19). Після заповнення необхідних даних нову книгу буде додано до загального списку книг та автоматично виставлено на обмін.

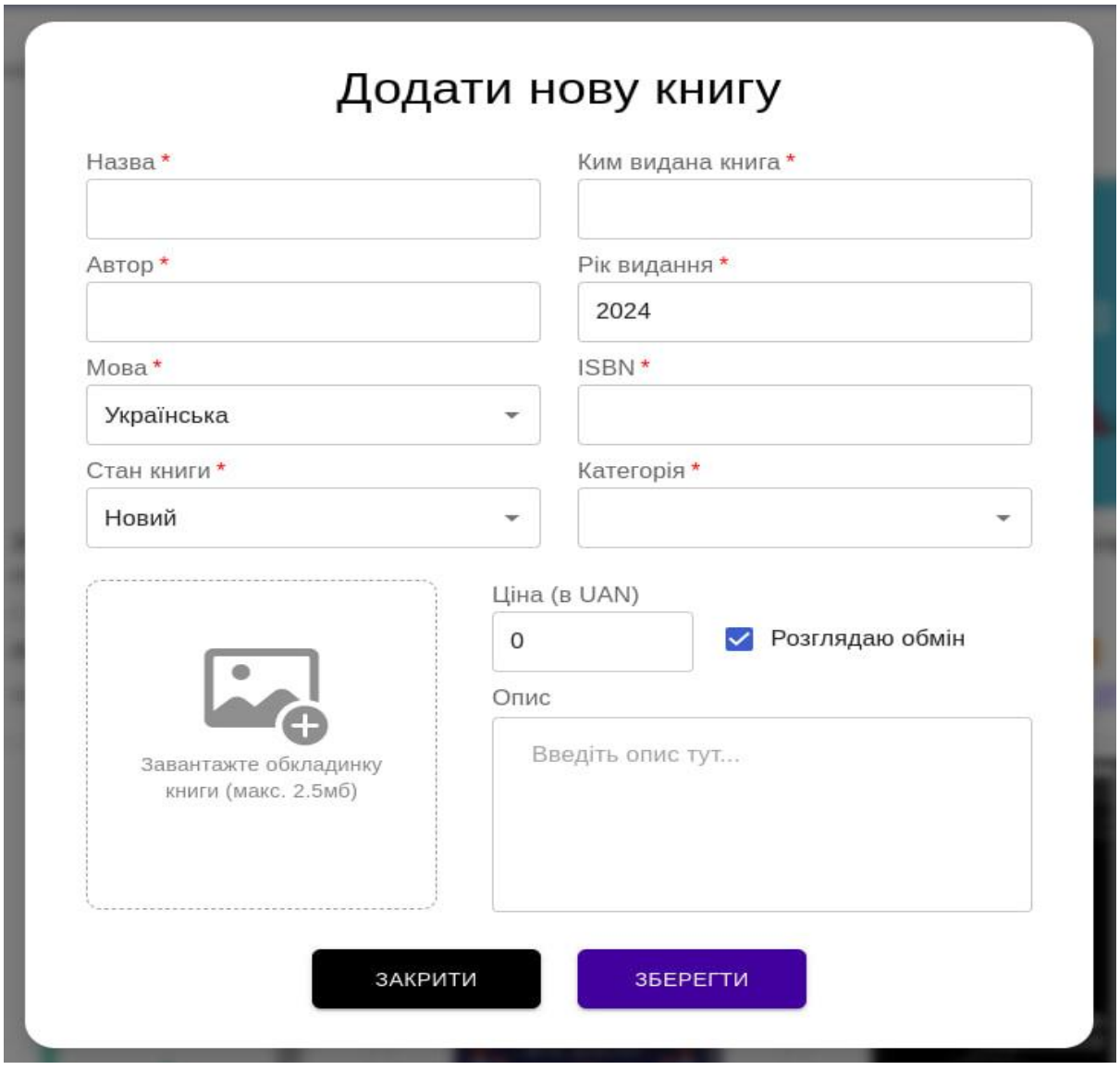

Рисунок 3.19 – Додавання нової книги

Також користувач має можливість налаштувати свій персональний профіль. На сторінці «Налаштування» розміщена форма для оновлення даних користувача, яка зображена на рисунку 3.20. Крім того, можна завантажити фото профілю або залишити його за замовчуванням.

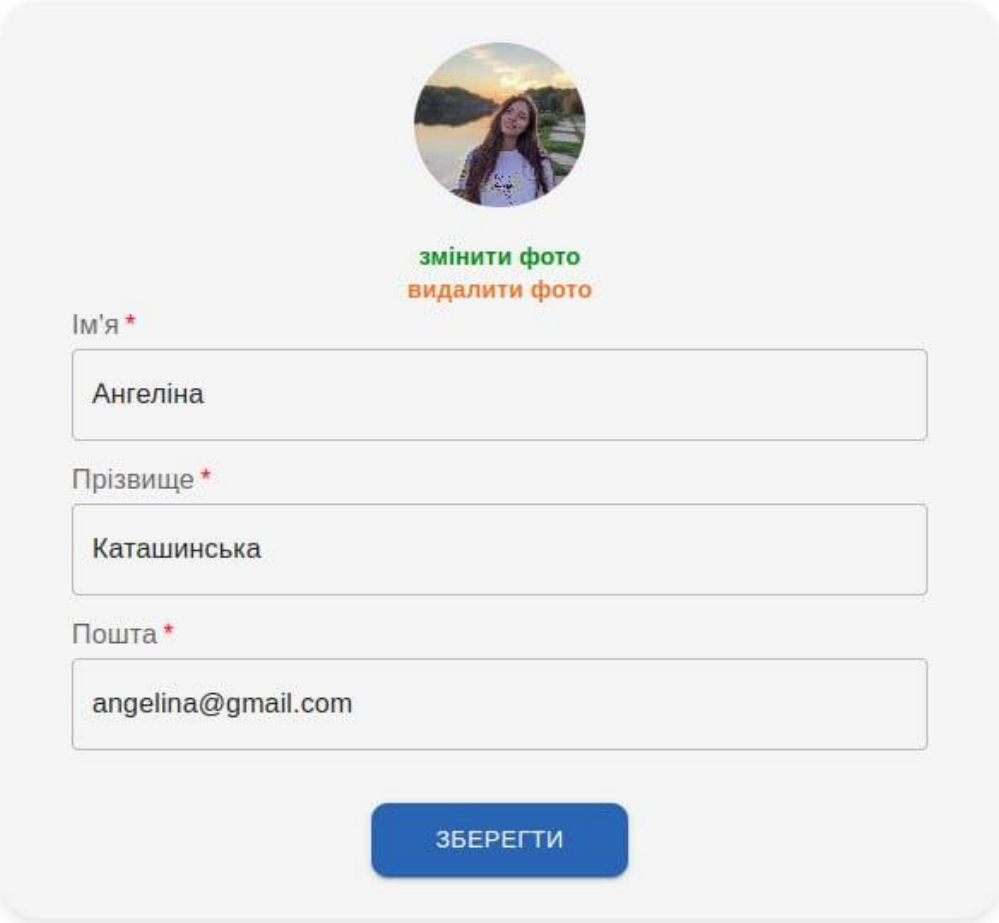

Рисунок 3.20 – Форма для оновлення даних користувача

Під час тестування WEB-ресурсу для обміну книгами було проведено реєстрацію для десятьох користувачів, кожен з яких виставив по три книги на обмін та надіслав кілька запитів іншим користувачам. Було ретельно перевірено функціональність фільтрації та пошуку за різними параметрами, систему відгуків, список бажань, а також налаштування профілю користувача, включаючи завантаження та збереження фото. Окрім цього, декілька раз тестувалася можливість редагування та видалення усіх можливих записів, а також правильність переадресації посилань та роботи кнопок. Додатково, була перевірена валідація форм для введення даних, щоб гарантувати достовірність та безпеку інформації, яку вводять користувачі. Проведене тестування показало, що функціонал WEB-ресурсу працює належним чином та повністю відповідає визначеним вимогам.

Порівняльна характеристика розробленого WEB-ресурсу для обміну книгами з програмами-аналогами наведено в таблиці 3.2.

Таблиця 3.2 – Порівняльна характеристика розробленого WEB-ресурсу

Характеристика	Globidi Marketplace	Book Mooch	Bookswap	PaperBack Swap	Розроблений WEB-ресурс
Фільтрація за мовою книги та категоріями	-	+	-	+	+
Пошук книг за назвою, автором, ISBN	-	-	+	+	+
Вибір користувача для обміну	+	-	+	-	+
Інформація про стан книги	+	-	+	-	+
Можливість додати відгук про книгу	+	-	-	+	+
Наявність списку бажань	+	+	-	-	+

Отже, за результатами таблиці можна побачити, що розроблений WEB-ресурс для обміну книгами має розширений функціонал по відношенню до програм-аналогів, що свідчить про те, що мету дослідження було досягнуто.

3.5 Висновок до розділу 3

У цьому розділі було детально розглянуто та проаналізовано основні складові програмної реалізації WEB-ресурсу для обміну книгами. Спочатку було обґрунтовано вибір Visual Studio Code як середовища розробки, що надає сучасний та зручний інтерфейс для створення програмного забезпечення.

Було виконано порівняльний аналіз популярних мов програмування, таких як JavaScript, Python, Java та C++. З огляду на потреби проекту, було прийнято рішення використовувати мову програмування JavaScript. Серед бібліотек для клієнтської частини було обрано React, Redux, Material UI та React Hook Form, що забезпечують ефективну розробку користувацького інтерфейсу, керування станом додатку та додаткові функціональні можливості. Для розробки серверної частини було використано фреймворк Express, бібліотеки Multer для обробки завантаження файлів та Bcrypt для безпечного шифрування паролів.

При обґрунтуванні вибору системи управління базою даних було розглянуто PostgreSQL та MySQL. З урахуванням вимог до масштабованості, надійності, продуктивності, відповідності стандартам SQL та функціональності було обрано PostgreSQL. Для полегшення взаємодії з базою даних на рівні коду було застосовано об'єктно-реляційне відображення (ORM) Sequelize.

Тестування допомогло перевірити роботу основних функціональних можливостей, виявити та виправити помилки, що забезпечило стабільну роботу програми. Крім того, було проведено порівняльну характеристику розробленого WEB-ресурсу з програмами-аналогами. Було розширено функціональні можливості, а саме: можливість вибору користувача для обміну, ефективний пошук за назвою, автором, унікальним номером книги, фільтрація за категоріями, інформація про стан книги, наявність списку бажань та система відгуків. З цього можна зробити висновок, що мету дослідження було досягнуто.

ВИСНОВКИ

Під час виконання бакалаврської дипломної роботи було розроблено WEB-ресурс для обміну книгами.

Всі задачі поставлені для реалізації WEB-ресурсу для обміну книгами, були виконані в повному обсязі, а саме:

- обґрунтовано доцільність розробки WEB-ресурсу для обміну книгами;
- проаналізовано програми-аналоги, їх переваги та недоліки;
- спроектовано WEB-ресурс для обміну книгами;
- програмно реалізовано WEB-ресурс для обміну книгами;
- виконано тестування програми та проведено аналіз отриманих результатів.

Було розглянуто різні варіанти середовища для реалізації програмного продукту та обрано WEB-браузер, як оптимальне рішення. На основі проведеного аналізу програм-аналогів поставлено задачі та визначено основні вимоги до функціоналу.

Під час проектування було обґрунтовано вибір архітектури, розроблено ER-модель та схему бази даних. Також було наведено UML-діаграми, які допомогли визначити та описати основні функції програми та процес взаємодії користувача з нею. Крім того, було розроблено схему алгоритму роботи WEB-ресурсу.

Також було обрано середовище розробки, мову та бібліотеки програмування, систему управління базою даних та ORM. Крім того, проведено тестування розробленого WEB-ресурсу, яке підтвердило відповідність заданим вимогам.

Мета дослідження, яка полягала в розширенні функціональних можливостей, досягнута за рахунок того, що у порівнянні з аналогами, розроблений WEB-ресурс надає можливість вибору користувача для обміну. Крім того, пропонує ефективний пошук та зручну систему фільтрації для швидкого знаходження книг, список бажань, можливість додати відгук та переглянути інформацію про стан книги.

Отже, всі поставлені задачі було виконано, а мету роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Каташинська А. О., Сілагін О. В. Розробка WEB-ресурсу для обміну книгами. LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024), Вінниця, 2024. [Електронний ресурс]. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20547

2. Web, Mobile or Desktop App [Електронний ресурс]. – Режим доступу: https://whatfix.com/blog/desktop-application/

3. Bookcrossing [Електронний ресурс]. – Режим доступу: https://www.bookcrossing.com/

4. BookMooch [Електронний ресурс]. – Режим доступу: http://bookmooch.com/

5. Glodibi Marketplace web resourse [Електронний ресурс]. – Режим доступу: https://marketplace.glodibi.com/

6. PaperBackSwap web resourse [Електронний ресурс]. – Режим доступу: https://www.paperbackswap.com/index.php

7. Bookswap [Електронний ресурс]. – Режим доступу: https://bookswap.co.uk/

8. Характеристика MVC архітектури [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Модель-вид-контролер

9. Monolithic vs Microservices Architecture [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/

10. Особливості безсерверної архітектури [Електронний ресурс]. – Режим доступу: https://www.datadoghq.com/knowledge-center/serverless-architecture/

11. Огляд клієнт-серверної архітектури [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/client-server-model/

12. Unified Modeling Language (UML) [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/

13. UML-діаграма прецедентів (Use case diagram) [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/use-case-diagram/

14. UML-діаграма послідовності [Електронний ресурс]. – Режим доступу: https://www.maxzosim.com/sequence-diagrams/
15. UML-діаграма розгортання [Електронний ресурс]. – Режим доступу: https://creately.com/guides/deployment-diagram-tutorial/
16. What is a database [Електронний ресурс]. – Режим доступу: https://www.simplilearn.com/tutorials/dbms-tutorial/what-is-a-database
17. Entity-Relationship model [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model
18. Нормалізація бази даних [Електронний ресурс]. – Режим доступу: https://web.posibnyky.vntu.edu.ua/fitki/10savchuk_organizaciya_bazdanih_znan/gl_43.html
19. Поняття, властивості, форми подання алгоритму [Електронний ресурс]. – Режим доступу: https://romanov.in.ua/11-3/
20. Середовище розробки Visual Studio Code [Електронний ресурс]. – Режим доступу: https://code.visualstudio.com/docs
21. JavaScript [Електронний ресурс]. – Режим доступу: https://learn.javascript.ua/intro
22. Python tutorial [Електронний ресурс]. – Режим доступу: https://docs.python.org
23. Difference between C++ and Java languages [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/cpp-vs-java/
24. React documentation [Електронний ресурс]. – Режим доступу: https://react.dev/
25. Material-UI [Електронний ресурс]. – Режим доступу: https://mui.com/material-ui/
26. Express.js [Електронний ресурс]. – Режим доступу: https://expressjs.com/
27. PostgreSQL [Електронний ресурс]. – Режим доступу:
28. MySQL database [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/
29. What is an Object-relational mapping (ORM) [Електронний ресурс]. – Режим доступу: https://www.prisma.io/dataguide/types/relational/what-is-an-orm
30. Sequelize documentation [Електронний ресурс]. – Режим доступу: https://sequelize.org/

ДОДАТКИ

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: WEB-ресурс для обміну книгами

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 96,96% Схожість 3,04%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Озеранський В.С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Каташинська А.О.

Керівник роботи Сілагін О.В

ДОДАТОК Б
ІНСТРУКЦІЯ КОРИСТУВАЧА

Для реєстрації користувачу потрібно вказати ім’я, прізвище, пошту та пароль, а для входу пошту та пароль. Модальне вікно для реєстрації користувача із заповненими даними зображено на рисунку Б.1, а для входу на рисунку Б.2.

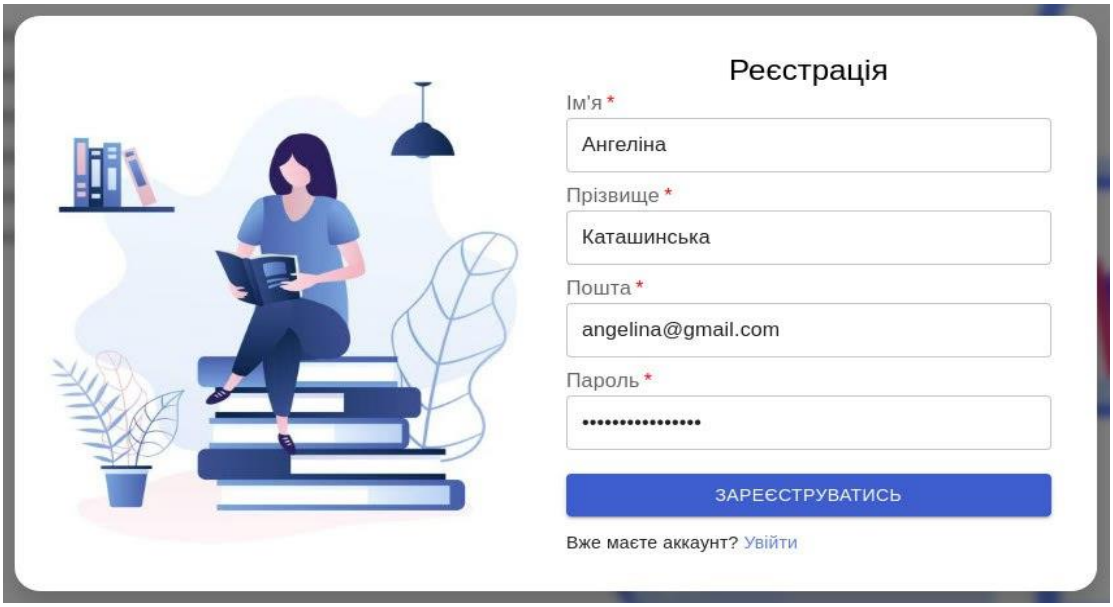

Рисунок Б.1 – Модальне вікно для реєстрації користувача

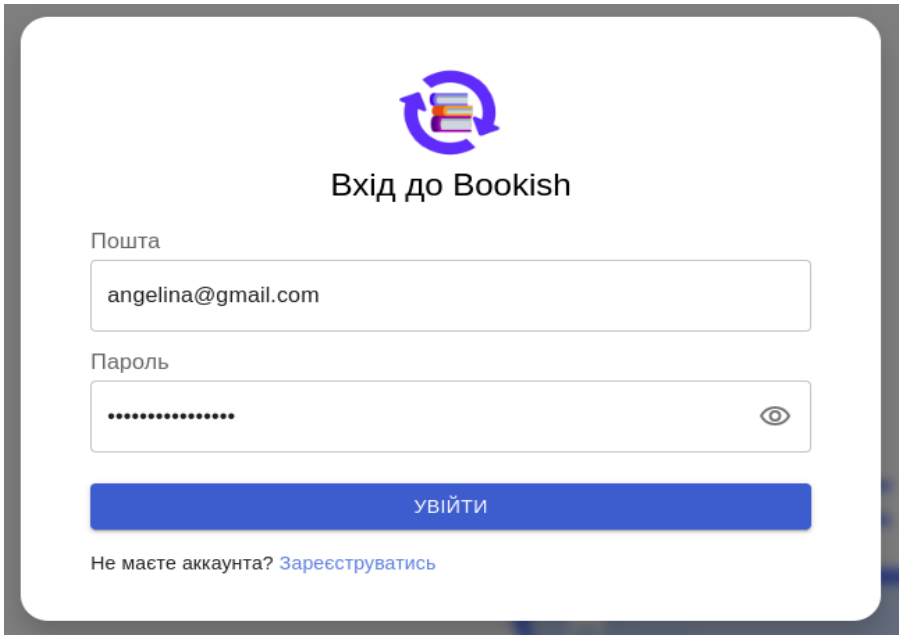

Рисунок Б.2 – Модальне вікно для входу користувача

Після входу відображається головна сторінка (рис. Б.3) із книгами, які можна фільтрувати за категоріями та мовою. Також реалізовано пошук за різними параметрами. Натиснувши на обрану книгу, відбувається перехід на сторінку книги (рис. Б.4).

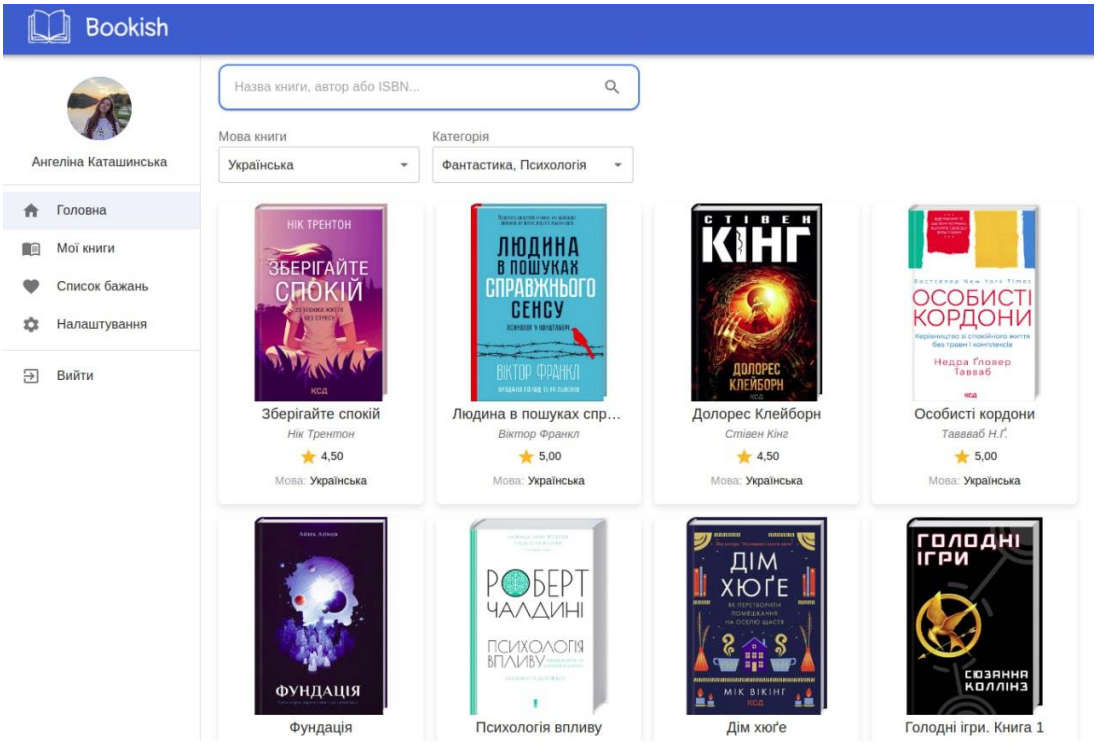

Рисунок Б.3 – Головна сторінка WEB-ресурсу для обміну книгами

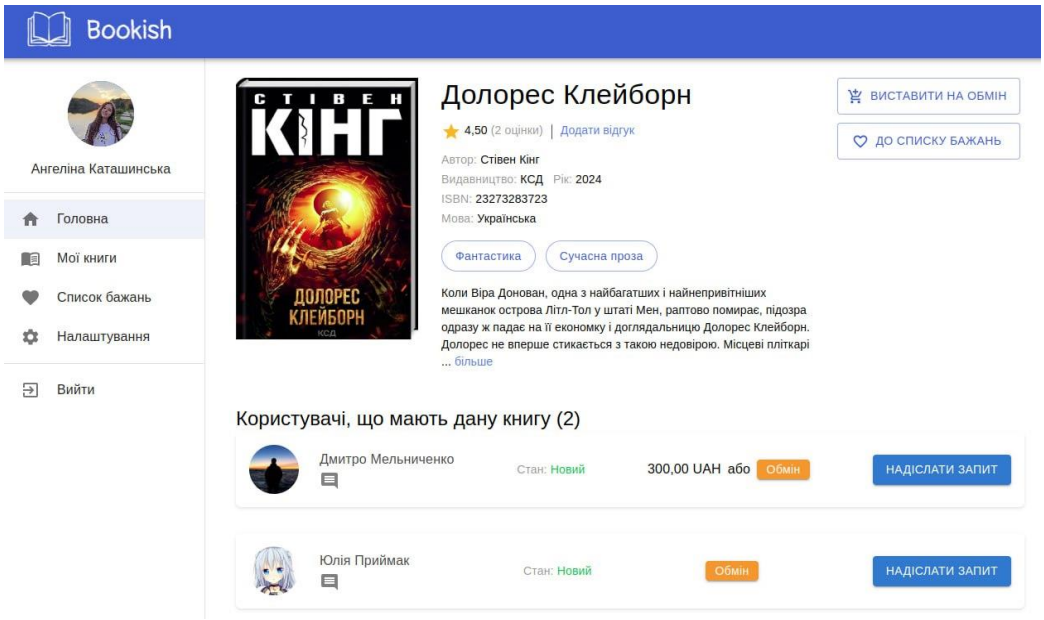

Рисунок Б.4 – Сторінка книги

Окрема секція показує список користувачів, готових обміняти чи продати книгу із вказаними умовами. В разі надсилання запиту на обмін книгою власник книги отримає повідомлення (рис. Б.5) на вказану електронну пошту від WEB-ресурсу.

Рисунок Б.5 – Повідомлення про запит на обмін книгою

Секція відгуків (рис. Б.6), яка дозволяє ознайомитися з відгуками та оцінкою книги від інших користувачів, а також можна залишити свій відгук (рис. Б.7).

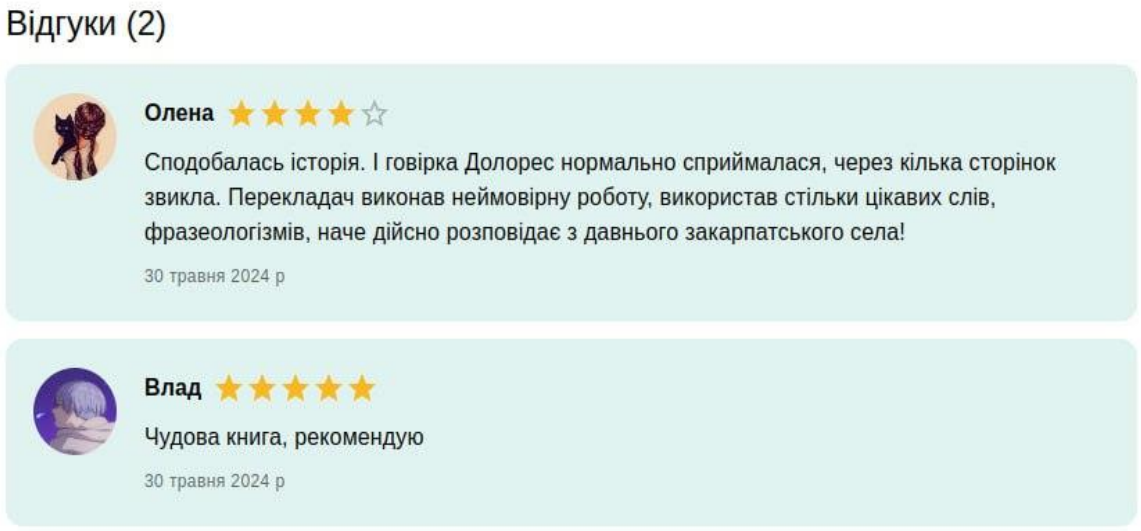

Рисунок Б.6 – Відгуки користувачів

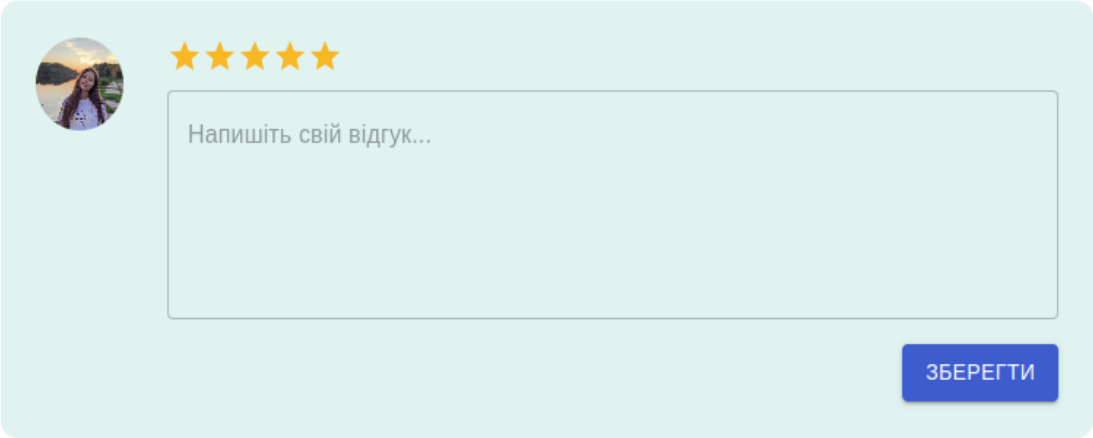

Рисунок Б.7 – Створення відгуку

Також на сторінці книги розміщені кнопки для додавання книги до списку бажань або виставлення її на обмін. Книги додані до списку бажань відображені на сторінці «Список бажань». Виставлення книги на обмін зображено на рисунку Б.8. Виставлені книги відображені на сторінці «Мої книги» (рис. Б.9). Якщо потрібної книги немає в базі даних є можливість додати її самостійно. Для цього потрібно натиснути «Додати книгу» після чого відкриється форма (рис. Б.10).

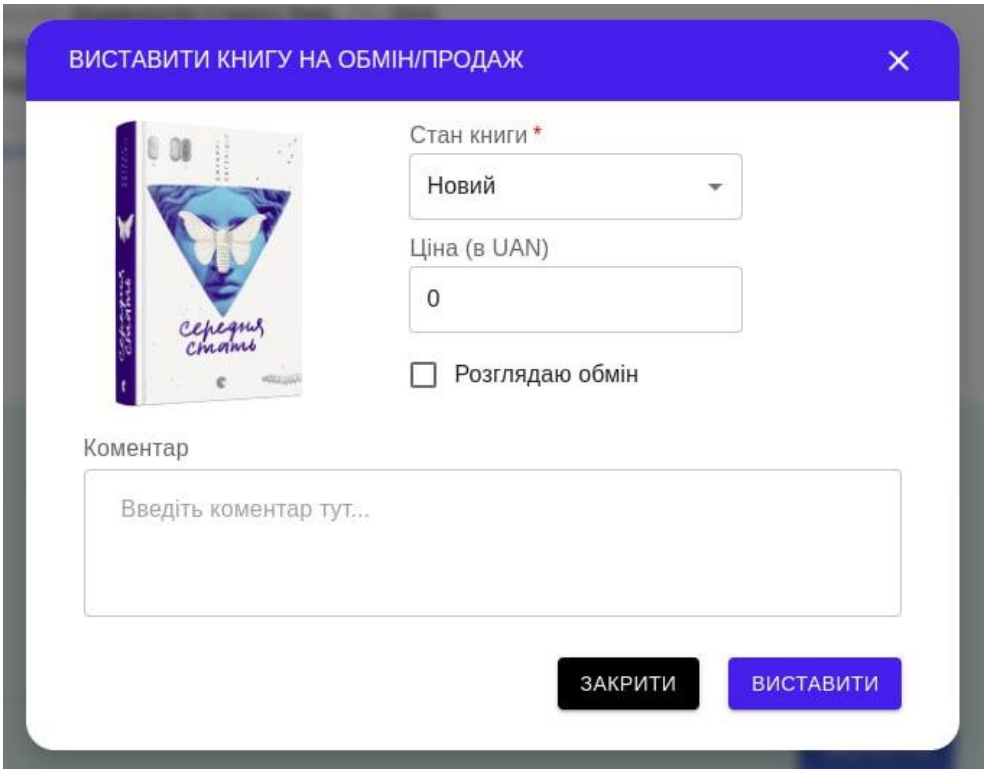

Рисунок Б.8 – Виставлення книги на обмін

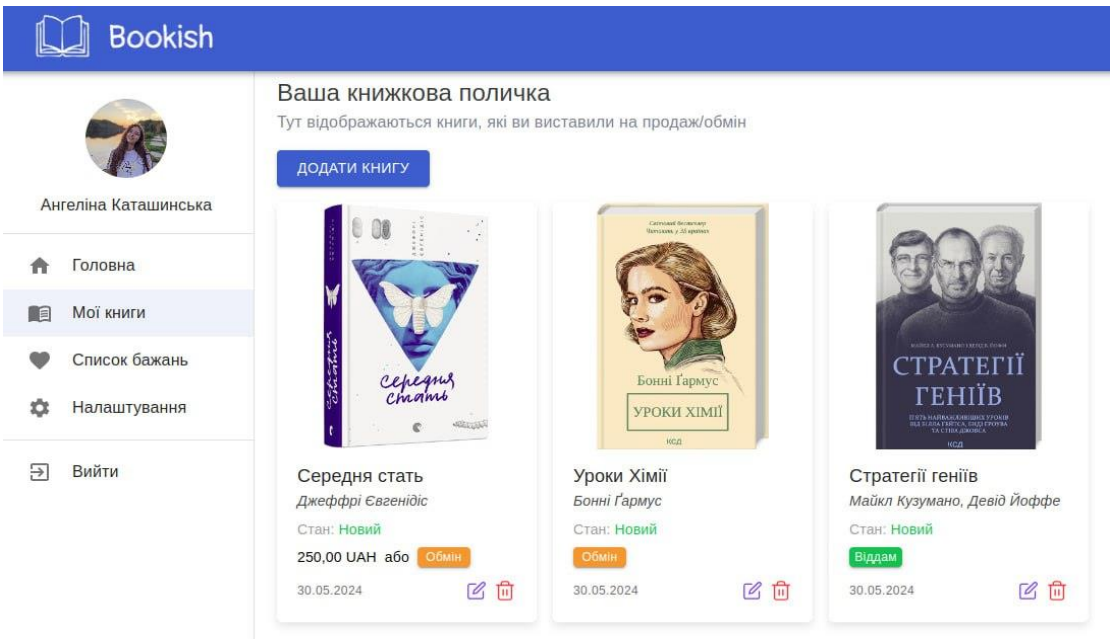

Рисунок Б.9 – Сторінка «Мої книги»

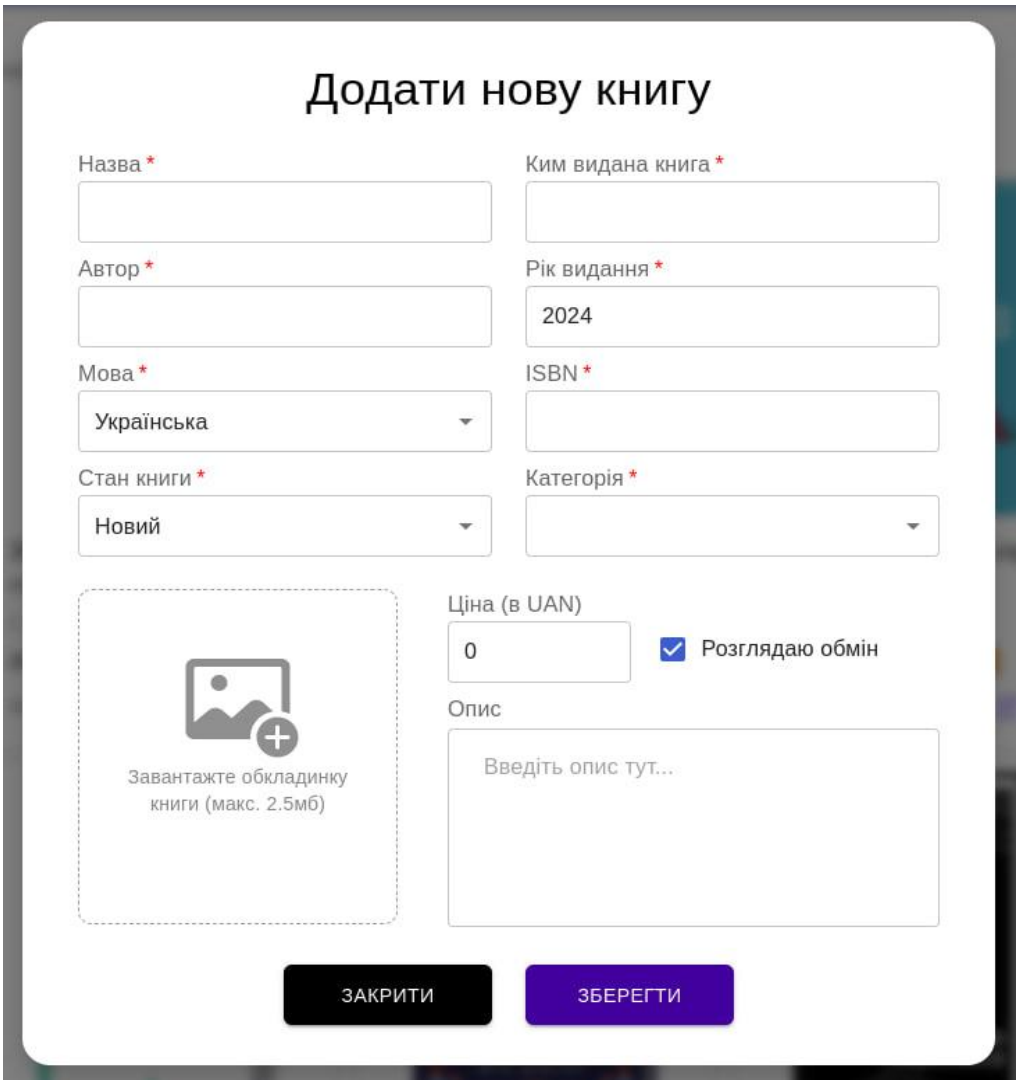

Рисунок Б.10 – Додавання нової книги

Після заповнення необхідних даних (рис. Б.11) нову книгу буде додано до загального списку книг та автоматично виставлено на обмін. Поля відмічені зірочкою є обов'язковими до заповнення, а також необхідно завантажити обкладинку книги та вказати умови для обміну. Для додавання книги потрібно натиснути на кнопку «Зберегти», а якщо потрібно скасувати додавання натиснути «Закрити».

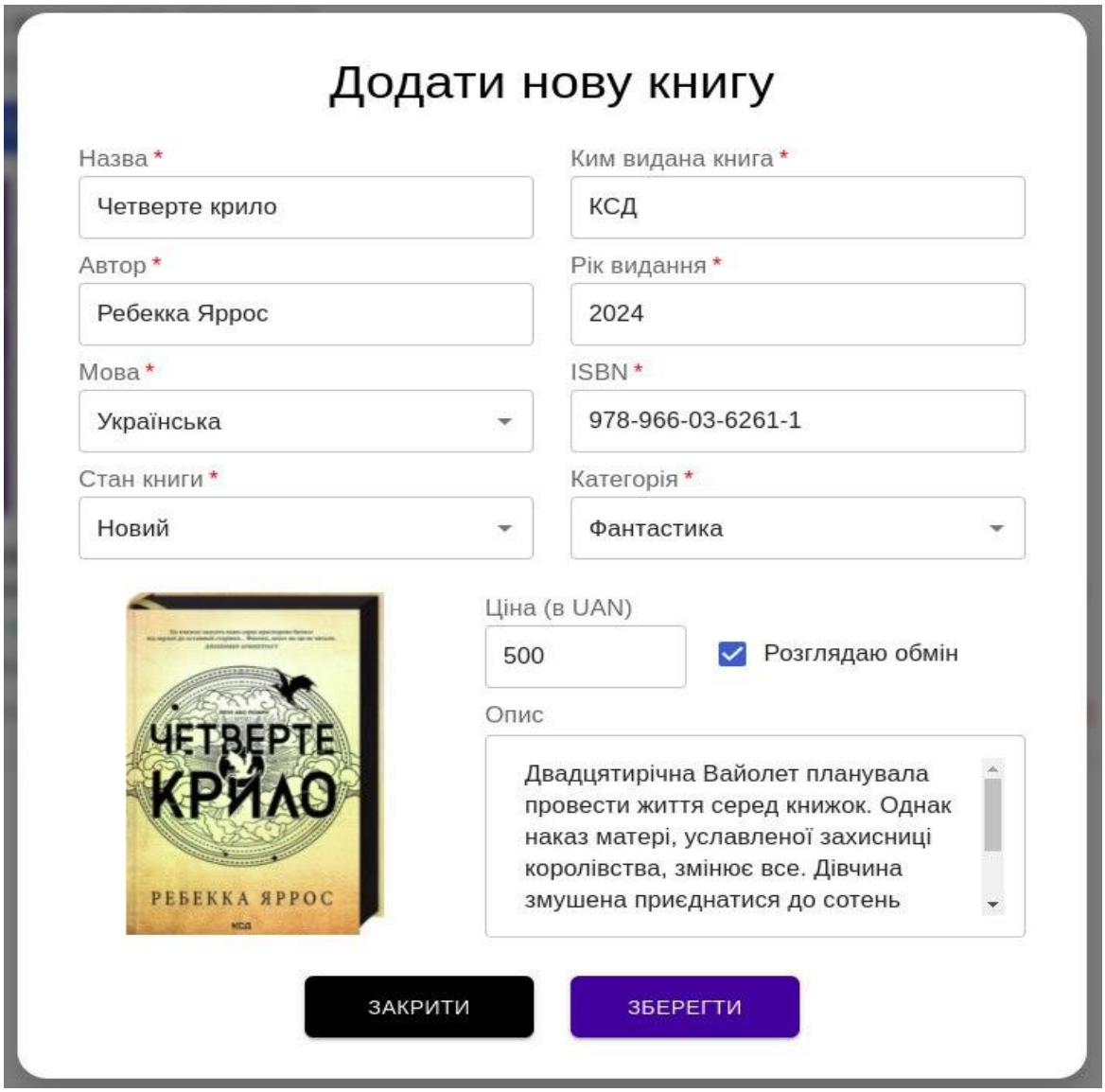

Рисунок Б.11 – Приклад заповнених даних для додавання книги

Також користувач має можливість налаштувати свій персональний профіль. На сторінці «Налаштування» розміщена форма для оновлення даних користувача, яка зображена на рисунку Б.12. Крім того, можна завантажити фото профілю,

натиснувши «Змінити фото» або залишити його за замовчуванням. Також можна видалити фото, якщо натиснути «Видалити фото». Після оновлення інформації про користувача потрібно натиснути на кнопку «Зберегти».

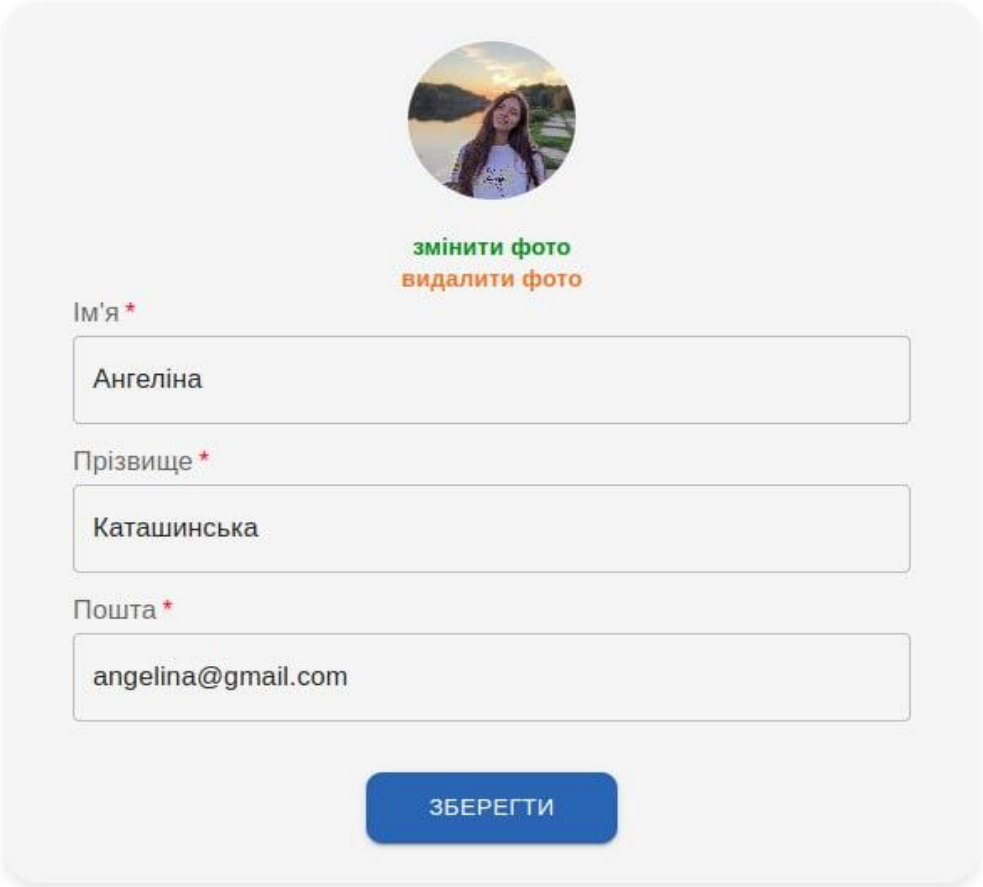

Рисунок Б.12 – Форма для оновлення даних користувача

ДОДАТОК В

ЛІСТИНГ ПРОГРАМИ

```
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import PrivateRoutes from './components/PrivateRoutes/PrivateRoutes';
import HomeLayout from './components/HomeLayout/HomeLayout';
import { HomePage, BookPage, MyBooksPage, SettingsPage,
WishlistPage, LandingPage} from './pages';

const App = () => {
  return (
    <Router>
      <Routes>
        <Route element={<PrivateRoutes />}>
          <Route element={<HomeLayout />}>
            <Route path="/home" element={<HomePage />} />
            <Route path="/myBooks" element={<MyBooksPage />} />
            <Route path="/settings" element={<SettingsPage />} />
            <Route path="/book/:bookId" element={<BookPage />} />
            <Route path="/wishlist" element={<WishlistPage />} />
          </Route>
        </Route>
        <Route path="/" element={<LandingPage />} exact />
      </Routes>
    </Router>
  );
};

export default App;

import React, { useEffect, useState } from 'react';
import { Box } from '@mui/material';
import { useDispatch, useSelector } from 'react-redux';
import { getAllBooks } from '../redux/book/actions';
import styles from './HomePage.styles';
import { CategoryFilter, BooksList, SearchBar, SelectWithoutForm as Select } from './components';

const languagesOptions = ['Українська', 'Англійська'];

const HomePage = () => {
  const dispatch = useDispatch();
  const books = useSelector((state) => state.bookRedux?.allBooks?.rows);
  const [searchTerm, setSearchTerm] = useState('');
  const [selectedCategories, setSelectedCategories] = useState([]);
  const [language, setLanguage] = useState('Українська');
  const [selectedLanguages, setSelectedLanguages] = useState([]);

  const handleLanguageFilter = (languages) => {
    setSelectedLanguages(languages);
```

```
  };
  const handleChangeLanguage = (e) => {
    setLanguage(e.target.value);
  };

  const handleSearch = (e) => {
    setSearchTerm(e.target.value);
    console.log('Введений пошуковий термін:', e.target.value);
  };

  useEffect(() => {
    dispatch(getAllBooks({ searchTerm, categories: selectedCategories }));
  }, [dispatch, searchTerm, selectedCategories]);

  return (
    <Box sx={styles.container}>
      <SearchBar handleSearch={handleSearch} searchTerm={searchTerm} />
      <Box sx={styles.filterContainer}>
        <Select
          inputLabel='Мова книги' options={languagesOptions} value={language}
          handleChange={handleChangeLanguage} sx={styles.languageSelect}/>
        <CategoryFilter
          required={false} selectedCategories={selectedCategories}
          setSelectedCategories={setSelectedCategories} boxStyle={styles.categoryFilter} />
      </Box>
      {books?.length ? <BooksList books={books} /> : 'На жаль, по вашому запиту книг не знайдено'}
    </Box>
  );
};

export default HomePage;

const AddBookModal = ({ isModalOpen, handleClose, bookId, postId, isExchangeInitial, bookPrice,
postComment }) => {
  const [isExchange, setIsExchange] = useState(true);
  const [cover, setCover] = useState('');
  const [file, setFile] = useState(null);
  const [error, setError] = useState(false);
  const getSchemaValues = useGetSchemaValues(addBookFormValidation);
  const [selectedCategories, setSelectedCategories] = useState([]);

  const initialValues = useMemo(() => ({
    comment: postComment || '',
    price: 0,
    title: '',
    author: '',
    publicationYear: 2024,
    publisher: '',
    isbn: '',
    stateOfBook: 'Новий',
    description: '',
    language: 'Українська'
```

```
}), [postComment]);

const model = useMemo(() => getSchemaValues(initialValues), [getSchemaValues, initialValues]);

const dispatch = useDispatch();
const { showErrorSnackbar, showSuccessSnackbar } = useSnackbar();

const handleCheckboxChange = (event) => {
  setIsExchange(event.target.checked);
};

const handleSubmitAddBook = async (data) => {
  try {
    const formData = new FormData();
    formData.append('cover', file);
    formData.append('isExchange', isExchange);
    formData.append('isAddPost', true);
    formData.append('categoryIds', selectedCategories.join(','));
    for (const [key, value] of Object.entries(data)) {
      formData.append(key, value);
    }
    const response = await dispatch(createBook(formData));
    showSuccessSnackbar('Нову книгу додано');
  } catch (error) {
    showErrorSnackbar('Щось пішло не так...');
  }
};

const handleUpload = (imageUrl, blob, file) => {
  setCover(imageUrl);
  setFile(file)
};

const handleDeleteImage = () => {
  setCover('');
};

return (
  <Modal
    handleClose={handleClose}
    isOpen={isModalOpen}
  >
    <Box>
      <Box>
        <Typography variant='h4'> Додати нову книгу </Typography>
        <FormProvider model={model} validationSchema={addBookFormValidation}
          onSubmit={handleSubmitAddBook} updateModel='always'>
          <Box>
            <Box>
              <Stack>
                <CustomTextField name='title' inputTitle='Назва' variant='outlined'/>
                <CustomTextField name='author' inputTitle='Автор' variant='outlined'/>
```

```
          <CustomSelect name='language' inputTitle='Мова' options={languageOptions} />
        <CustomSelect name='stateOfBook' inputTitle='Стан книги' options={stateOfBookOptions}/>
            </Stack>
            <Stack>
              <CustomTextField name='publisher' inputTitle='Ким видана книга' variant='outlined'/>
              <CustomTextField name='publicationYear' inputTitle='Рік видання' variant='outlined'/>
              <CustomTextField name='isbn' inputTitle='ISBN' variant='outlined'/>
             <CategoryFilter selectedCategories={selectedCategories}
            setSelectedCategories={setSelectedCategories} />
            </Stack>
          </Box>
          <Box>
            <Box>
              <UploadButton handleUpload={handleUpload} name='cover'>
                {!cover && (
                  <>
                    <CardMedia component="img" image={defaultUploadImage} alt='Обкладинка'
                    />
                    <Typography variant='body2'>
                      Завантажте обкладинку книги
                      (макс. 2.5мб)
                    </Typography>
                  </>
                )}
                {error ? (
                  <Typography variant='body2'>
                    Завантажте будь-ласка обкладинку книги
                  </Typography>
                ) : null}
              </UploadButton>
            </Box>
            <Stack>
              <Box>
                <CustomTextField name='price' inputTitle='Ціна (в UAN)' variant='outlined'/>
              </Box>
              <FormControlLabel control={
                <Checkbox checked={isExchange} onChange={handleCheckboxChange}/>
              } label="Розглядаю обмін" />
              <CustomTextField name='description' placeholder="Введіть опис тут..."
                inputTitle='Опис' variant='outlined' required={false}
                multiline minRows={5} maxRows={5}/>
            </Stack>
          </Box>
          <FormControls currentItem={true} handleClose={handleClose} />
        </Box>
      </FormProvider>
    </Box>
  </Box>
 </Modal>
);
};
```

```
yup.addMethod(yup.string, 'hasLowerCase', function() {
 return this.test({
   name: 'hasLowerCase',
   message: 'password.hasLowerCase',
   test: (value = '') => /[a-z]/.test(value),
 });
});

yup.addMethod(yup.string, 'hasDigit', function() {
 return this.test({
   name: 'hasDigit',
   message: 'password.hasDigit',
   test: (value = '') => /\d/.test(value),
 });
});

export const rootReducer = combineReducers({
 userRedux: userReducer,
 bookRedux: bookReducer,
 postRedux: postReducer,
 reviewRedux: reviewReducer,
 wishlistRedux: wishlistReducer,
 categoryRedux: categoryReducer,
});

const composeEnhancers = compose;
const store = createStore(
 rootReducer,
 composeEnhancers(applyMiddleware(thunk, logger))
);

import React, { useEffect, useState } from 'react';
import { GET_ALL_BOOKS, GET_BOOK_BY_ID, CREATE_BOOK } from '../action-types';
import axios from '../../utils/API';
import { BOOK } from '../../core/_consts/api_url';

export const createBook = (payload) => async (dispatch) => {
  try {
    const { data } = await axios.post(BOOK.createBook, payload);
    dispatch({ type: CREATE_BOOK, payload: data });
  } catch (error) {
    console.error('error', error);
  }
};

export const getAllBooks = (payload) => async (dispatch) => {
  try {
    const { data } = await axios.post(BOOK.getAllBooks, payload);
    dispatch({ type: GET_ALL_BOOKS, payload: data });
  } catch (error) {
    console.error('error', error);
  }
```

```
};

export const getBookById = (bookId) => async (dispatch) => {
  try {
    const { data } = await axios.get(`${BOOK.getBookById}${bookId}`);
    dispatch({ type: GET_BOOK_BY_ID, payload: data });
  } catch (error) {
    console.error('error', error);
  }
};

import {
    CREATE_USER, GET_ALL_BOOKS, GET_BOOK_BY_ID, LOGIN,
   CREATE_BOOK, DELETE_REVIEW, CREATE_REVIEW, UPDATE_REVIEW
} from '../../redux/action-types';

const initState = {
    allBooks: [],
    currentBook: {}
};

const bookReducer = (state = initState, action) => {
    const { type, payload } = action;

    switch (type) {
        case CREATE_BOOK:
            return { ...state };
        case GET_ALL_BOOKS:
            return { ...state, allBooks: payload };
        case GET_BOOK_BY_ID:
            return { ...state, currentBook: payload };
        case CREATE_REVIEW:
            return {
                ...state,
                currentBook: {...state.currentBook, reviews: [payload, ...state.currentBook.reviews] }
            };
        case UPDATE_REVIEW:
            const reviewIndex = state.currentBook.reviews.findIndex(review => review.id === payload.id);
            if (reviewIndex !== -1) {
                const updatedReviews = [
                    ...state.currentBook.reviews.slice(0, reviewIndex),
                    payload, ...state.currentBook.reviews.slice(reviewIndex + 1)
                ];

                return {
                    ...state,
                    currentBook: { ...state.currentBook, reviews: updatedReviews }
                };
            } else {
                return state;
            }
        case DELETE_REVIEW:
```

```
            const updatedReviews = state.currentBook.reviews.filter(review =>
            review.id !== Number(payload.id));
            console.log('UPDATED_REVIEW', updatedReviews)
            return {
               ...state,
               currentBook: { ...state.currentBook, reviews: updatedReviews
               }
            };
         default:
            return state;
      }
};

export default bookReducer;

const sequelize = require('../db');
const { DataTypes } = require('sequelize');

const User = sequelize.define('user', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  email: { type: DataTypes.STRING, unique: true },
  password: { type: DataTypes.STRING },
  firstName: { type: DataTypes.STRING },
  lastName: { type: DataTypes.STRING },
  avatarUrl: { type: DataTypes.STRING, allowNull: true, },
});

const Book = sequelize.define('book', {
  id: { type: DataTypes.INTEGER, primaryKey: true },
  title: { type: DataTypes.STRING },
  author: { type: DataTypes.STRING },
  publicationYear: { type: DataTypes.INTEGER },
  publisher: { type: DataTypes.STRING },
  imageUrl: { type: DataTypes.STRING, allowNull: true },
  isbn: { type: DataTypes.STRING },
  description: { type: DataTypes.TEXT, allowNull: true },
  language: { type: DataTypes.STRING, allowNull: true }
}, {});

const Post = sequelize.define('post', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  price: { type: DataTypes.FLOAT, },
  isExchange: { type: DataTypes.BOOLEAN, },
  comment: { type: DataTypes.STRING, allowNull: true },
  stateOfBook: { type: DataTypes.STRING, allowNull: true, },
  userId: { type: DataTypes.INTEGER, allowNull: false, },
  bookId: { type: DataTypes.INTEGER, allowNull: false, },
});

const Review = sequelize.define('review', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  content: { type: DataTypes.TEXT, allowNull: true },
```

```
  points: { type: DataTypes.INTEGER, allowNull: false,
  validate: { min: 1, max: 5 }
  },
  userId: { type: DataTypes.INTEGER, allowNull: false,
    references: { model: User, }
  },
  bookId: { type: DataTypes.INTEGER, allowNull: false,
    references: { model: Book, }
  }
}, {
  indexes: [
    { unique: true, fields: ['userId', 'bookId'] }
  ]
});

const Wishlist = sequelize.define('wishlist', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true
  },
  userId: { type: DataTypes.INTEGER, allowNull: false,
    references: { model: User, key: 'id' }
  },
  bookId: { type: DataTypes.INTEGER, allowNull: false,
    references: { model: Book, key: 'id' }
  }
}, {
  indexes: [
    {
      unique: true,
      fields: ['userId', 'bookId']
    }
  ]
});

const Category = sequelize.define('category', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true, },
  name: { type: DataTypes.STRING, allowNull: false,
  },
}, { timestamps: false });

const BookCategory = sequelize.define('bookCategory', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true, },
  bookId: { type: DataTypes.INTEGER, allowNull: false,
    references: { model: Book, key: 'id', },
  },
  categoryId: { type: DataTypes.INTEGER, allowNull: false,
    references: { model: Category, key: 'id', },
  },
}, {
  timestamps: false,
});

// Додавання зв’язків між таблицями
```

```
Book.belongsToMany(Category, { through: BookCategory, foreignKey: 'bookId',});
Category.belongsToMany(Book, { through: BookCategory, foreignKey: 'categoryId',});
BookCategory.belongsTo(Book, { foreignKey: 'bookId' });
BookCategory.belongsTo(Category, { foreignKey: 'categoryId' });

User.belongsToMany(Book, { through: Wishlist, foreignKey: 'userId' });
Book.belongsToMany(User, { through: Wishlist, foreignKey: 'bookId' });
Wishlist.belongsTo(User, { foreignKey: 'userId' });
Wishlist.belongsTo(Book, { foreignKey: 'bookId' });

User.hasMany(Post, { foreignKey: 'userId' });
Book.hasMany(Post, { foreignKey: 'bookId' });
Post.belongsTo(User, { foreignKey: 'userId' });
Post.belongsTo(Book, { foreignKey: 'bookId' });

Book.hasMany(Review, { foreignKey: 'bookId' });
User.hasMany(Review, { foreignKey: 'userId' });
Review.belongsTo(User, { foreignKey: 'userId' });
Review.belongsTo(Book, { foreignKey: 'bookId' });

const multer = require('multer');
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
     cb(null, 'images/');
  },
  filename: (req, file, cb) => {
     cb(null, new Date().toISOString() + '-' + file.originalname);
  },
});
const types = ['image/png', 'image/jpeg', 'image/jpg'];
const fileFilter = (req, file, cb) => {
  if (types.includes(file.mimetype)) {
     cb(null, true);
  } else {
     cb(null, false);
  }
};

module.exports = multer({ storage, fileFilter });

const ApiError = require('../error/ApiError');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { User, Basket } = require('../models/models');

const generateJwt = (id, email, role) => {
 return jwt.sign({ id, email },
  process.env.SECRET_KEY,
  { expiresIn: '24h' }
 );
};
```

```
class UserController {
 async registration(req, res, next) {
  const { firstName, lastName, email, password } = req.body;
  const candidate = await User.findOne({ where: { email } });
  if (candidate) {
   return next(ApiError.badRequest('Користувач з такою поштою вже існує'));
  }
  const hashPassword = await bcrypt.hash(password, 5);
  const user = await User.create({ firstName, lastName, email, password: hashPassword });
  const token = generateJwt(user.id, user.email);
  return res.json({ token, id: user.id, });
 }

 async login(req, res, next) {
  const { email, password } = req.body;
  const user = await User.findOne({ where: { email } });

  if (!user) {
   return next(ApiError.internal('Користувач не знайдений'));
  }

  let comparePassword = bcrypt.compareSync(password, user.password);
  if (!comparePassword) {
   return next(ApiError.internal('Перевірте пошту або пароль'));
  }

  const token = generateJwt(user.id, user.email);
  return res.json({ token, id: user.id });
 }

 async check(req, res, next) {
  const token = generateJwt(req.user.id, req.user.email);
  return res.json({ token });
 }

 async getUserById(req, res) {
  const { id } = req.query;
  const user = await User.findOne({
   where: { id },
   attributes: {
    exclude: ['password', 'role']
   }
  });
  return res.json(user);
 }

 async updateUserInfo(req, res, next) {
  try {
   const user = await User.findByPk(req.body.id);
   let avatarUrl = null;

   if (req.file) {
```

```
      const filePath = req.file.path;
      avatarUrl = `http://localhost:5000/${filePath}`;
    }

    for (const key in req.body) {
      user[key] = req.body[key];
    }

    if (user.avatarUrl && avatarUrl) {
      user.avatarUrl = avatarUrl;
    }

    const updatedUserInfo = await user.save();
    return res.json(updatedUserInfo);
  } catch (e) {
    next(ApiError.badRequest(e.message));
  }
 }
}

module.exports = new UserController();

const uuid = require('uuid');
const path = require('path');
const { Sequelize } = require('sequelize');
const fs = require('fs');
const nodemailer = require('nodemailer');
const { google } = require('googleapis');
const ApiError = require('../error/ApiError');
const config = require('./config'); // Імпортуємо конфігурацію

const OAuth2 = google.auth.OAuth2;
const OAuth2_client = new OAuth2( config.clientId, config.clientSecret);

OAuth2_client.setCredentials({ refresh_token: config.refreshToken });
const accessToken = OAuth2_client.getAccessToken();
const transporter = nodemailer.createTransport({
   service: 'gmail',
   auth: {
      type: 'OAuth2',
      user: config.user,
      clientId: config.clientId,
      clientSecret: config.clientSecret,
      refreshToken: config.refreshToken,
      accessToken
   }
});

class EmailController {
   async sendEmail(req, res, next) {
      try {
         const { recipientEmail, senderFullname, senderEmail, bookName } = req.body;
```

```
            const mailOptions = {
                from: `Bookish service <${config.user}>`,
                to: recipientEmail,
                subject: 'Запит на обмін книгою',
                html: `
                    <!DOCTYPE html>
                    <html>
                    <head>
                        <meta charset="UTF-8" />
                        <title>Запит на обмін книгою</title>
                        <style>${fs.readFileSync('path/to/styles.css', 'utf8')}</style>
                    </head>
                    <body>
                        <div class="container">
                            <div class="header">
                                <h3>Вітаємо!</h3>
                            </div>
                            <div class="content">
                                <p>Ви отримали запит на обмін книгою "${bookName}", від користувача
                                        ${senderFullname}</p>
                                <p>Для обговорення деталей обміну напишіть на вказану електронну пошту:
                                        ${senderEmail}</p>
                                <div class="footer">
                                    <p>З повагою, Bookish</p>
                                </div>
                                <div class="book-image">
                                    <img src={booksImage} alt="Книги" />
                                </div>
                            </div>
                        </div>
                    </body>
                    </html>
                `
            };

            transporter.sendMail(mailOptions, (error, info) => {
                if (error) {
                    res.status(500).send('Щось пішло не так');
                } else {
                    res.status(200).send('Email sent successfully');
                }
            });
        } catch (e) {
            next(ApiError.badRequest(e.message));
        }
    }
}

module.exports = new EmailController();
```

ДОДАТОК Г

ГРАФІЧНА ЧАСТИНА

«WEB-РЕСУРС ДЛЯ ОБМІНУ КНИГАМИ»

Виконала: студентка 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Каташинська А.О
(прізвище та ініціали)

Керівник: к.т.н, доцент каф. КН
Сілагін О.В.
(прізвище та ініціали)
«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

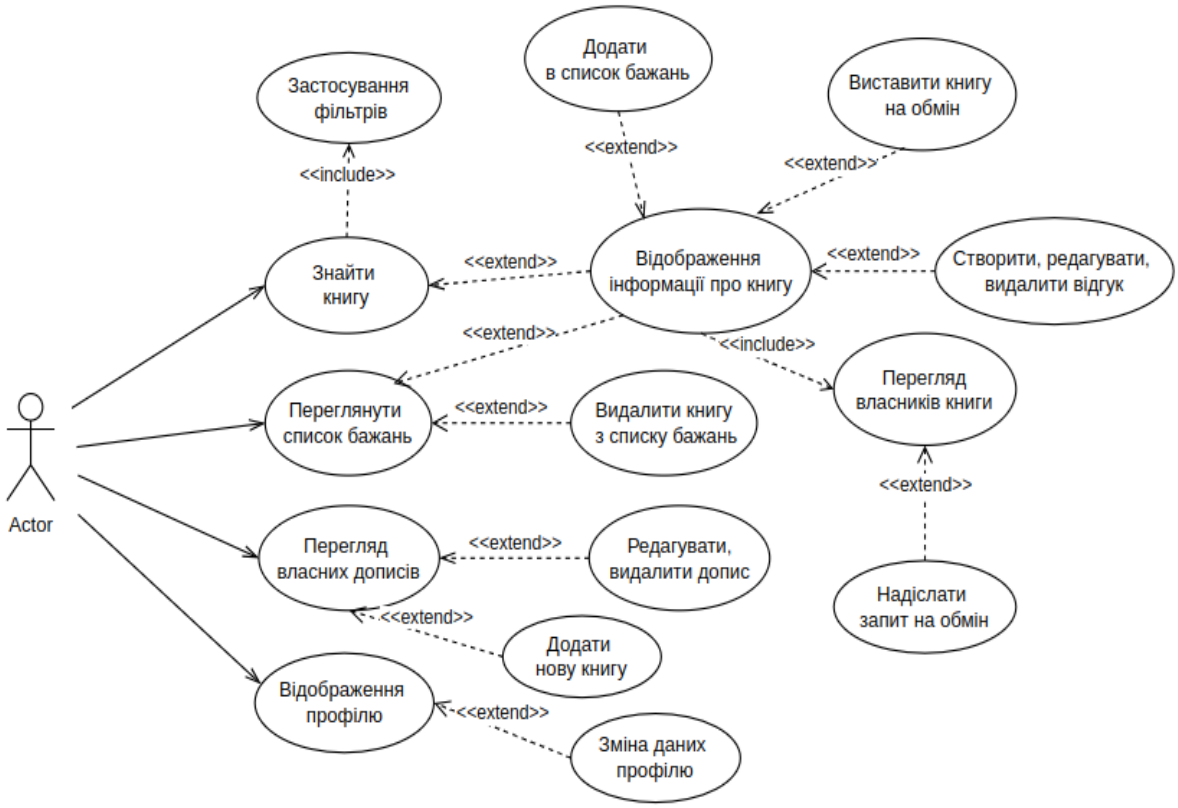

Рисунок Г.1 – UML-діаграма прецедентів WEB-ресурсу для обміну книгами

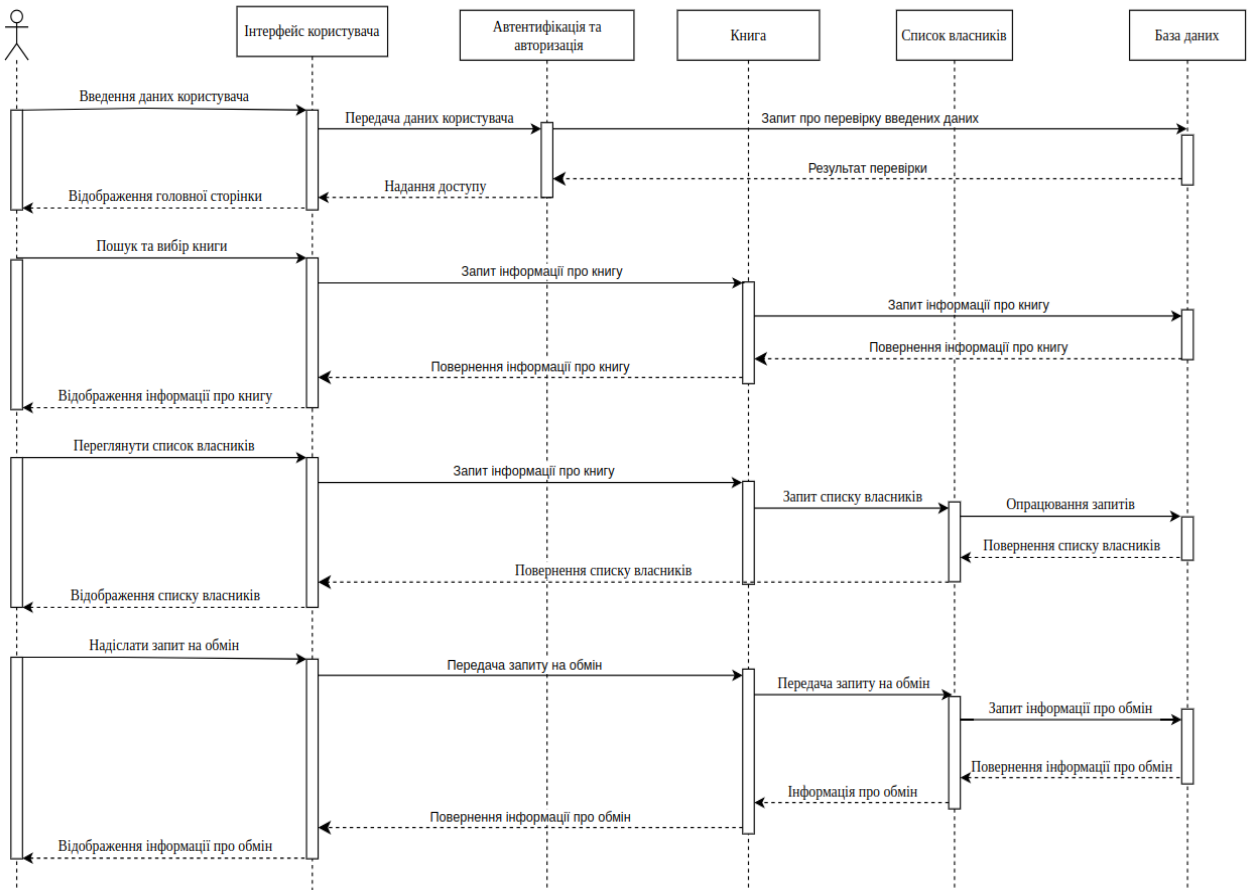

Рисунок Г.2 – UML-діаграма послідовності WEB-ресурсу для обміну книгами

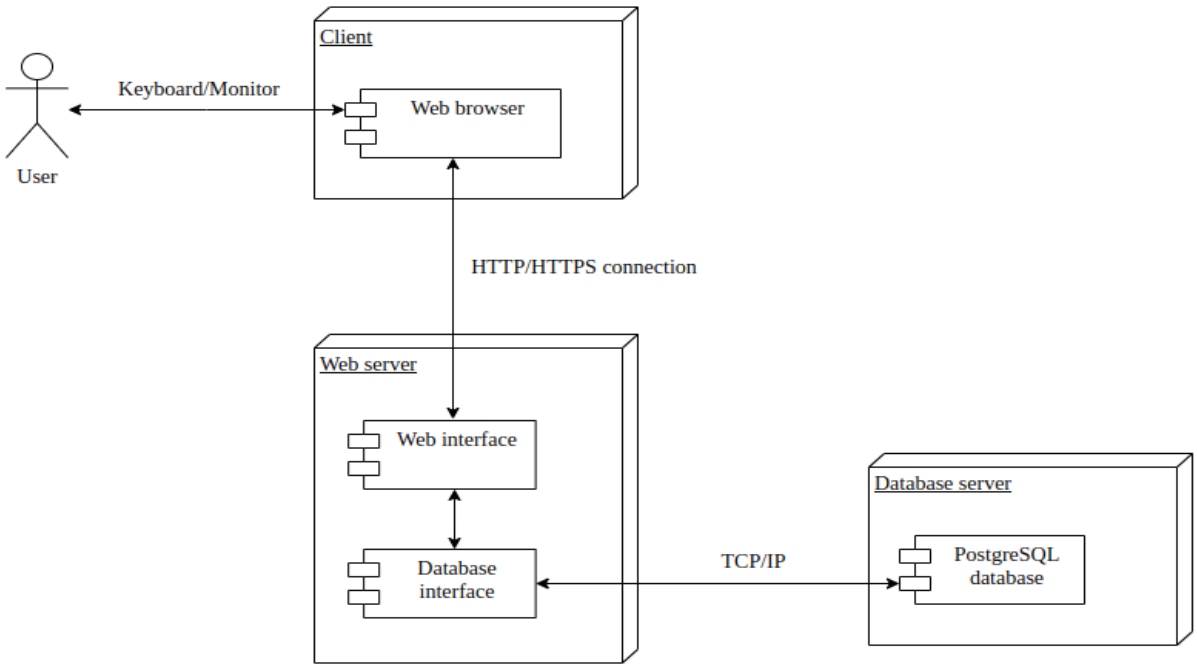

Рисунок Г.3 – UML-діаграма розгортання WEB-ресурсу для обміну книгами

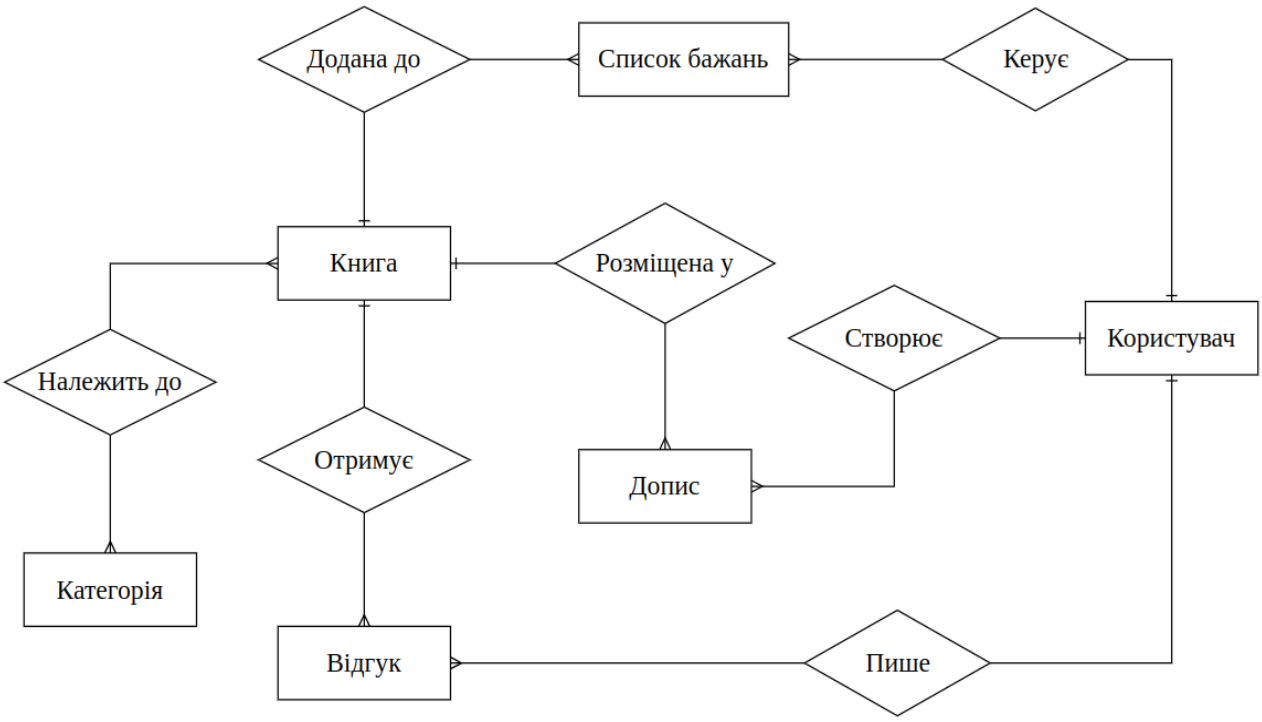

Рисунок Г.4 – ER-модель предметної області WEB-ресурсу для обміну книгами

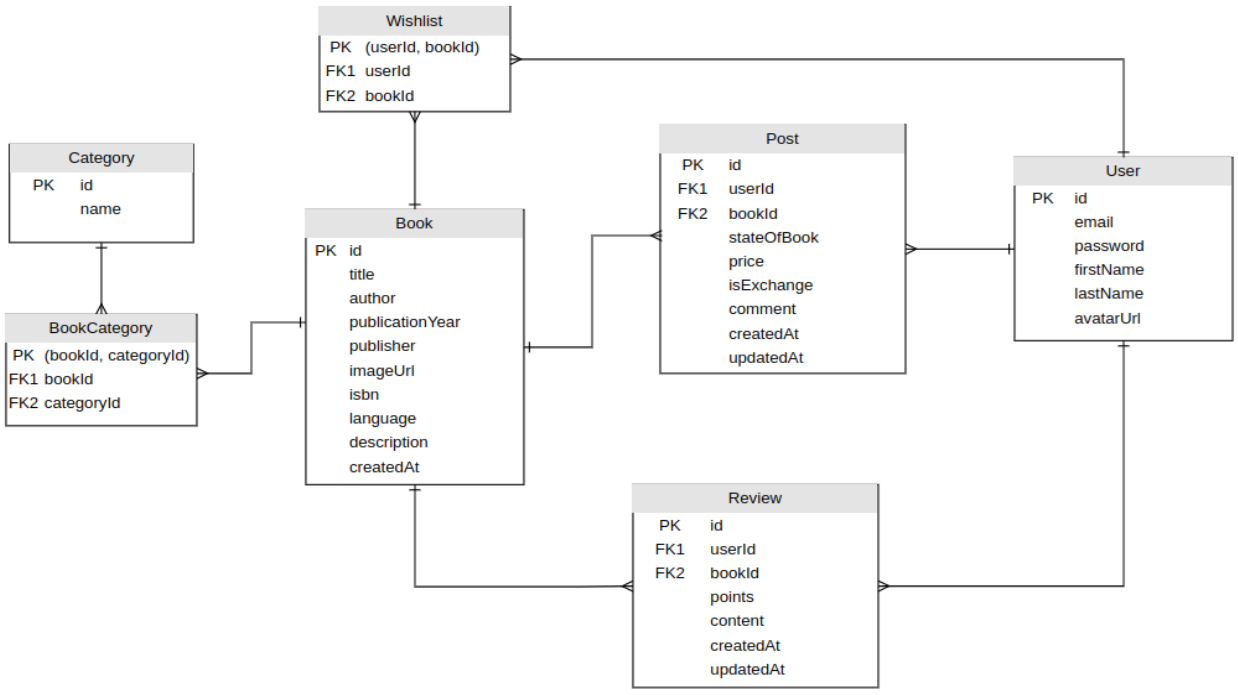

Рисунок Г.5 – Схема бази даних WEB-ресурсу для обміну книгами

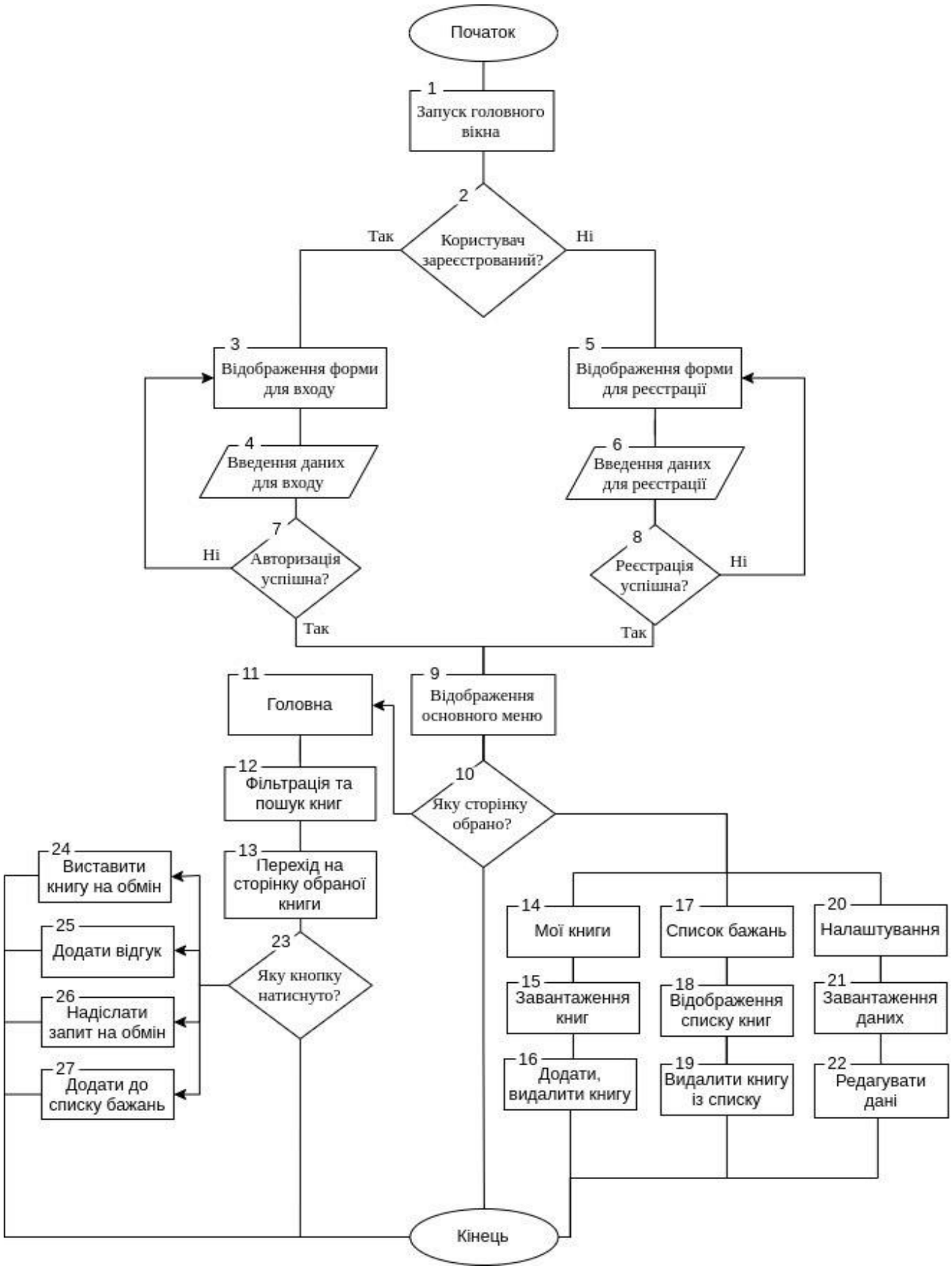

Рисунок Г.6 – Схема алгоритму роботи WEB-ресурсу для обміну книгами

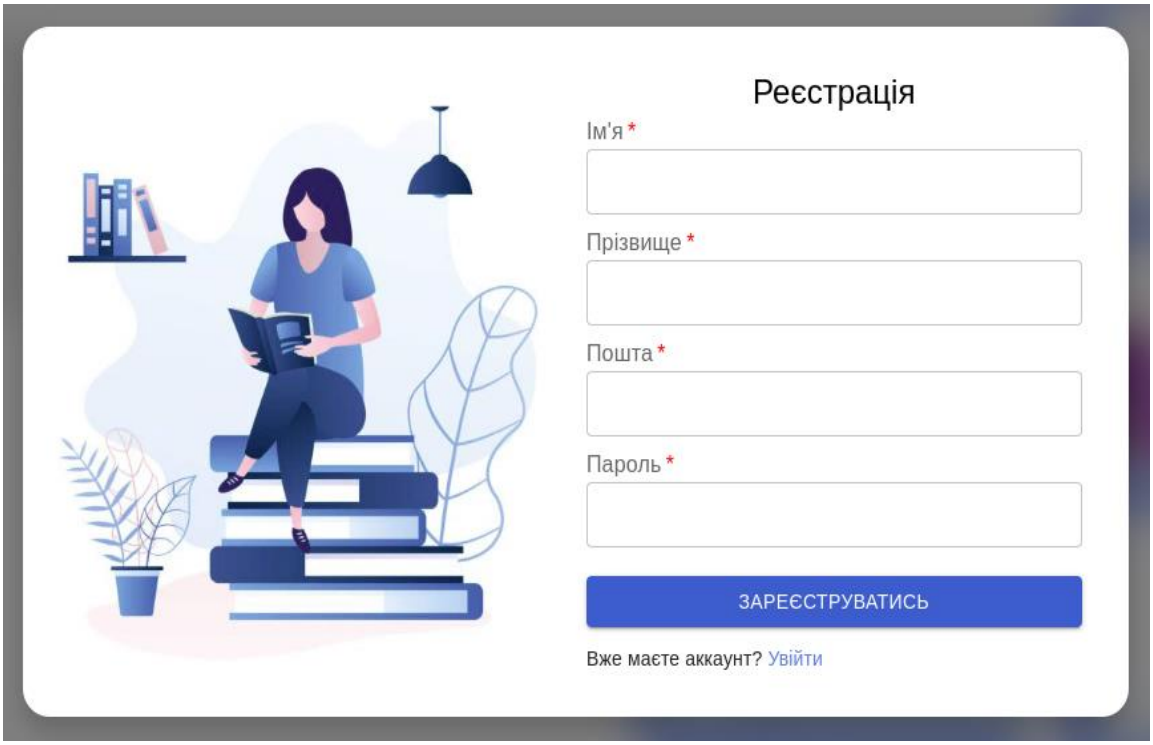

Рисунок Г.7 – Модальне вікно для реєстрації користувача

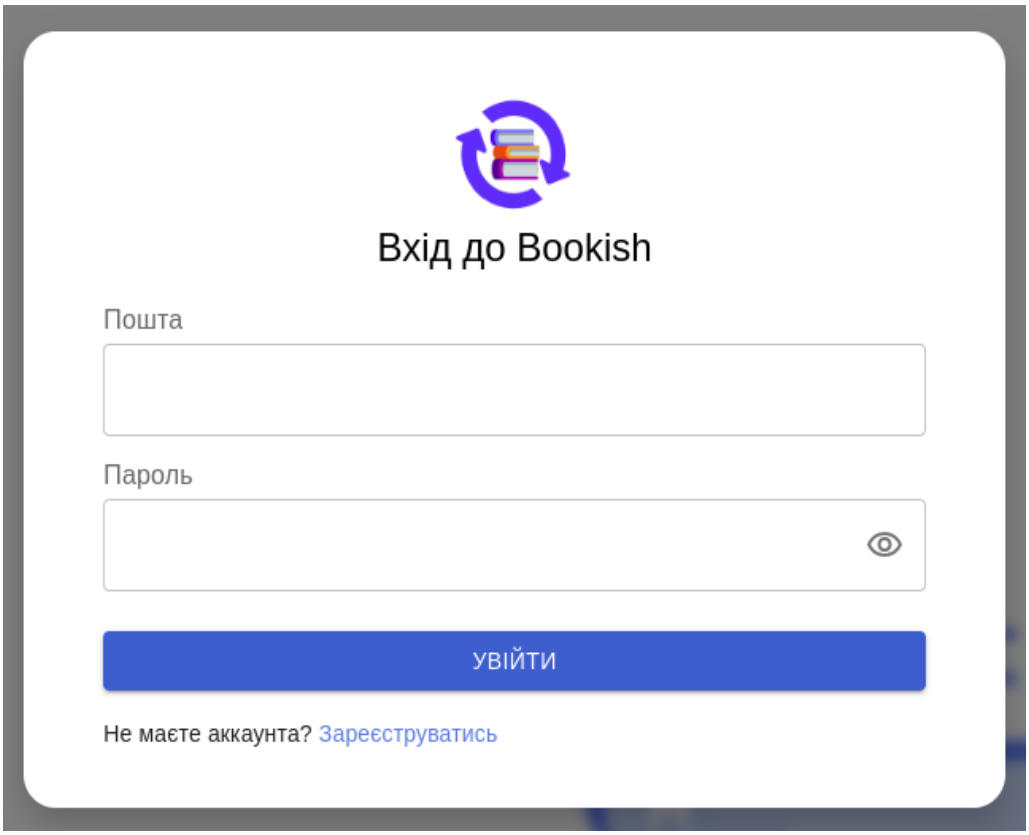

Рисунок Г.8 – Модальне вікно для входу користувача

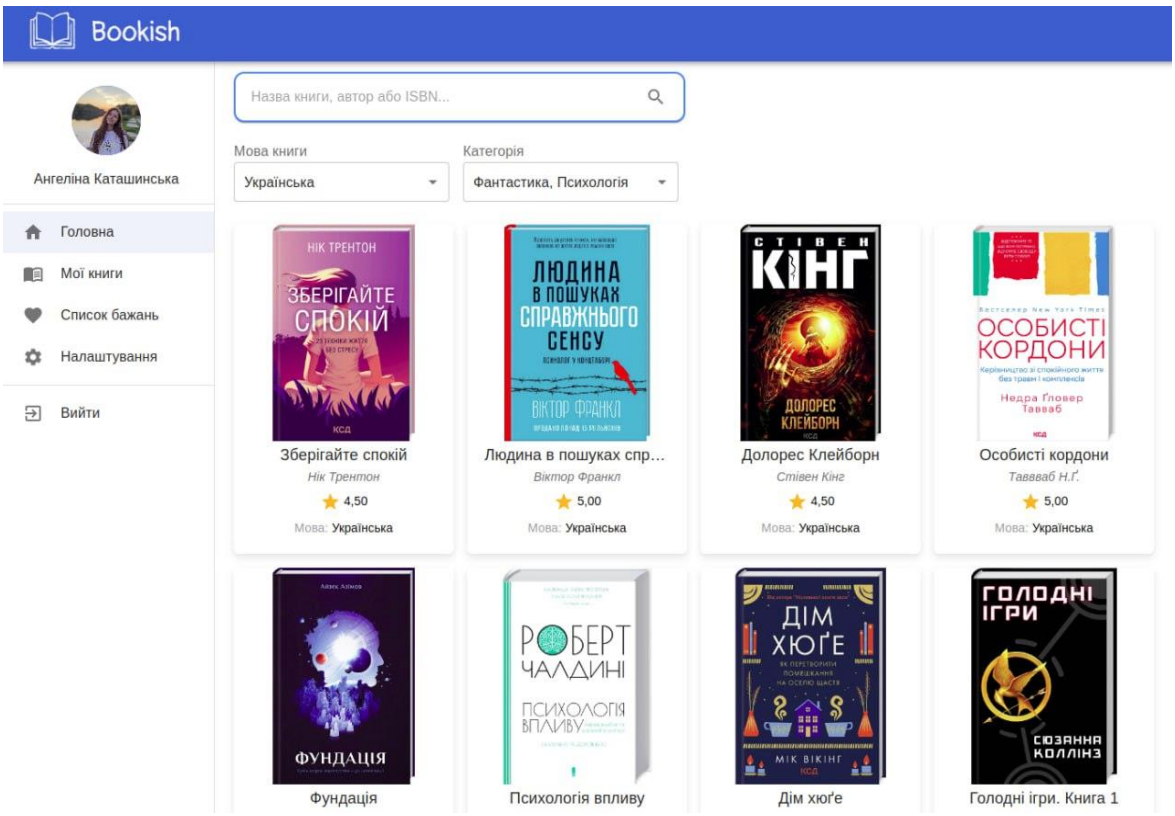

Рисунок Г.9 – Головна сторінка WEB-ресурсу для обміну книгами

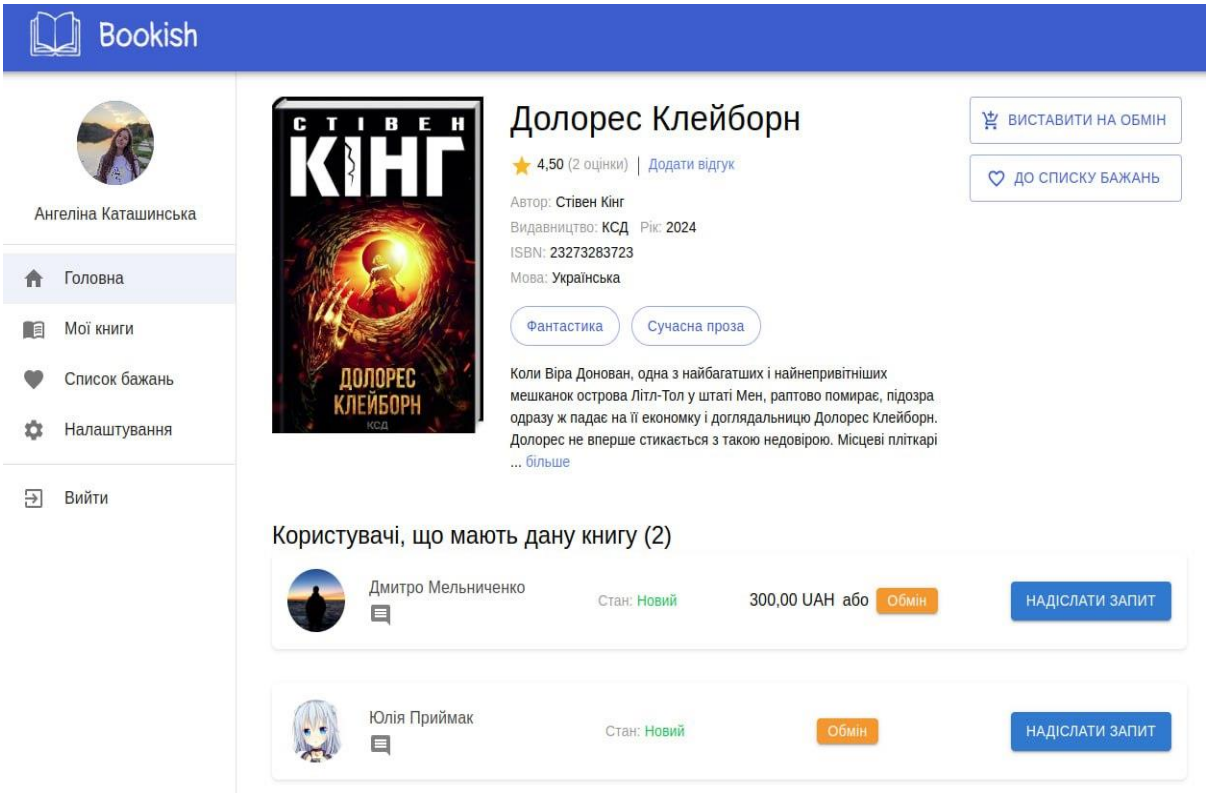

Рисунок Г.10 – Сторінка книги

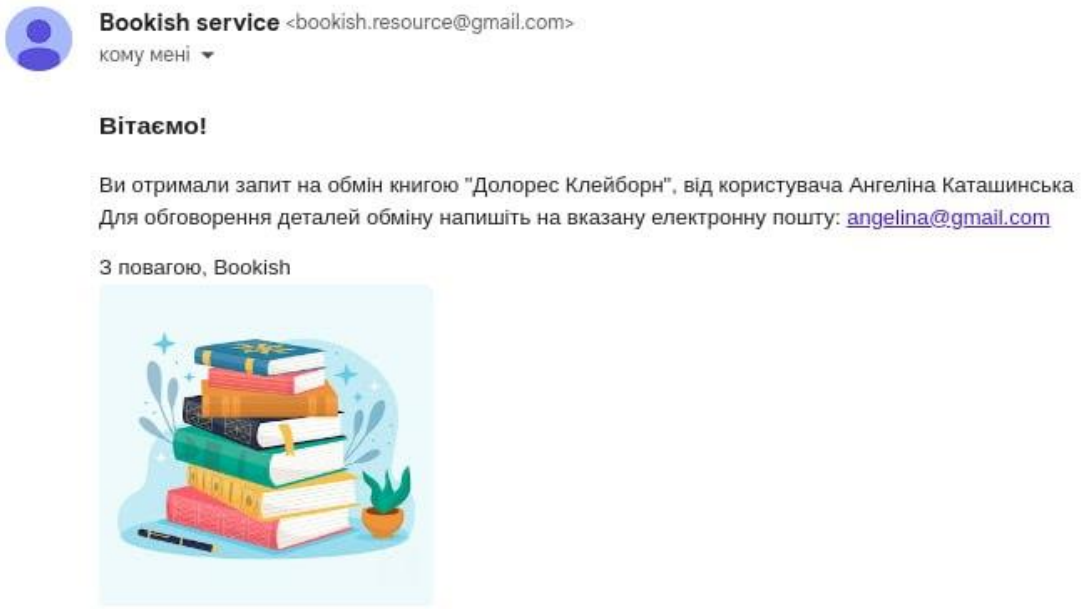

Рисунок Г.11 – Повідомлення про запит на обмін книгою

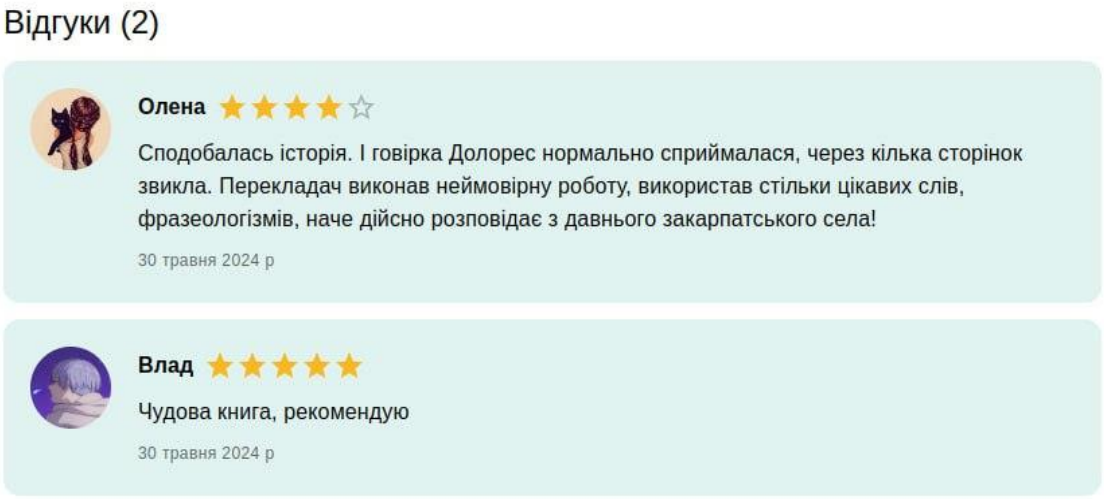

Рисунок Г.12 – Відгуки користувачів

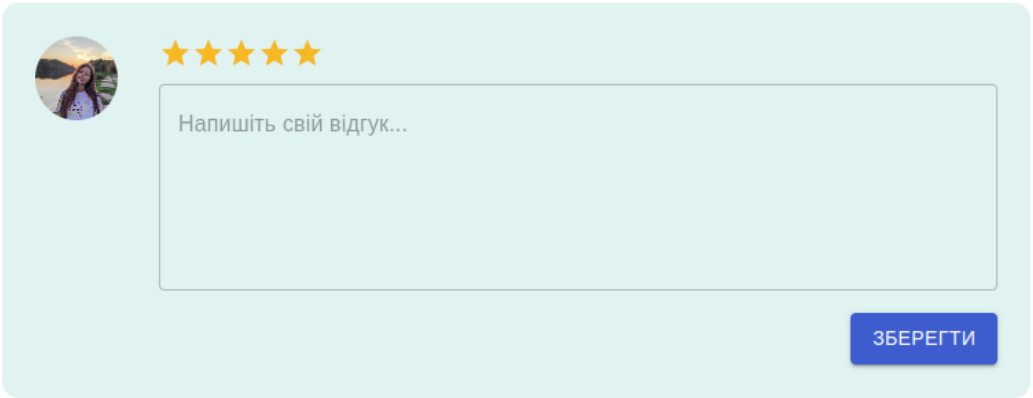

Рисунок Г.13 – Створення відгуку

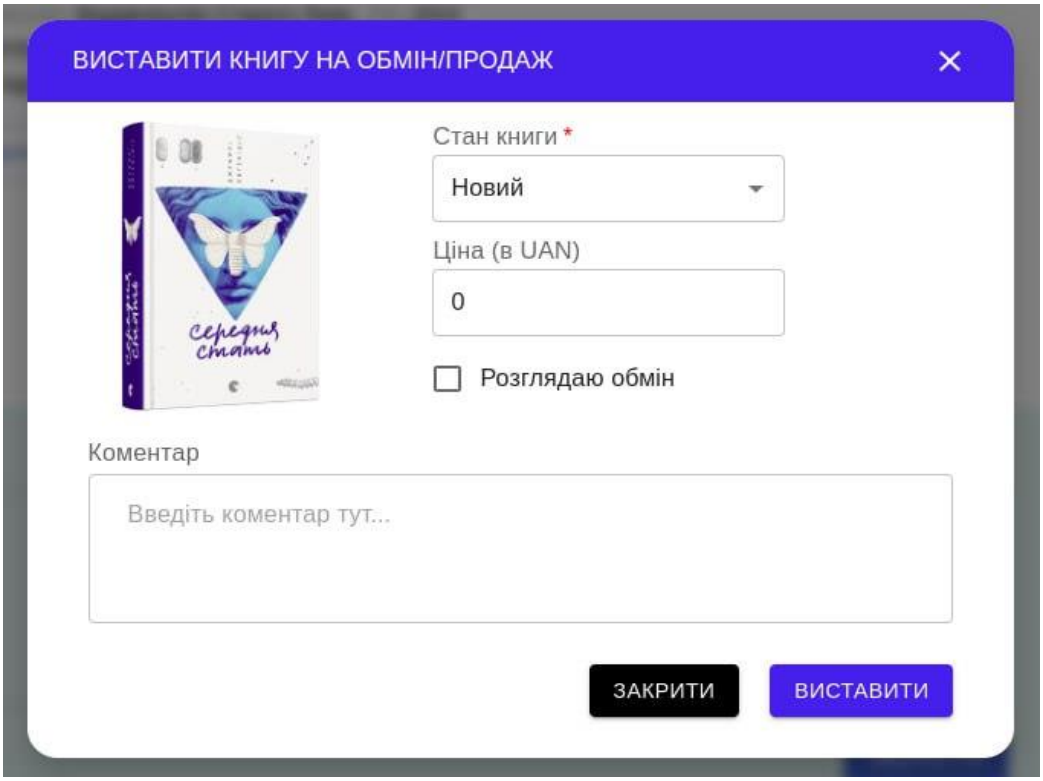

Рисунок Г.14 – Виставлення книги на обмін

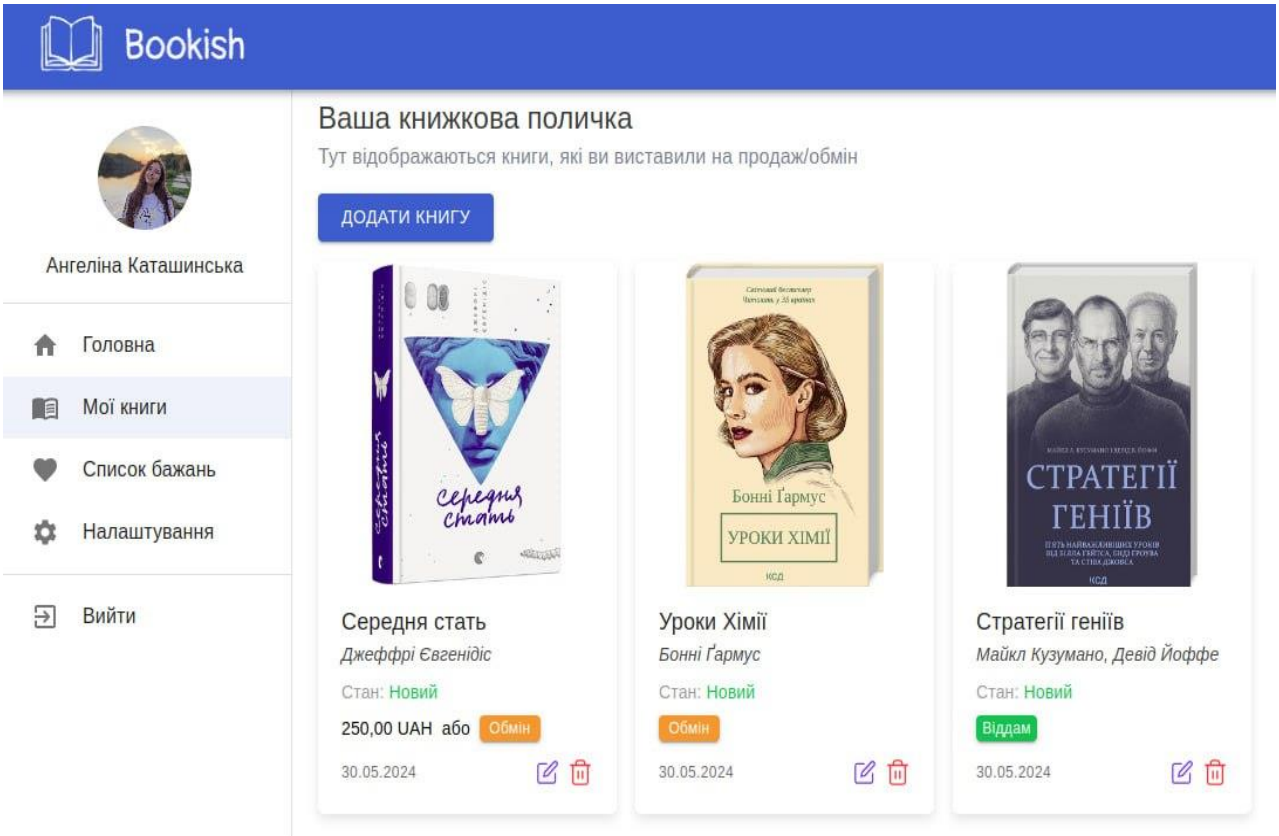

Рисунок Г.15 – Сторінка «Мої книги»

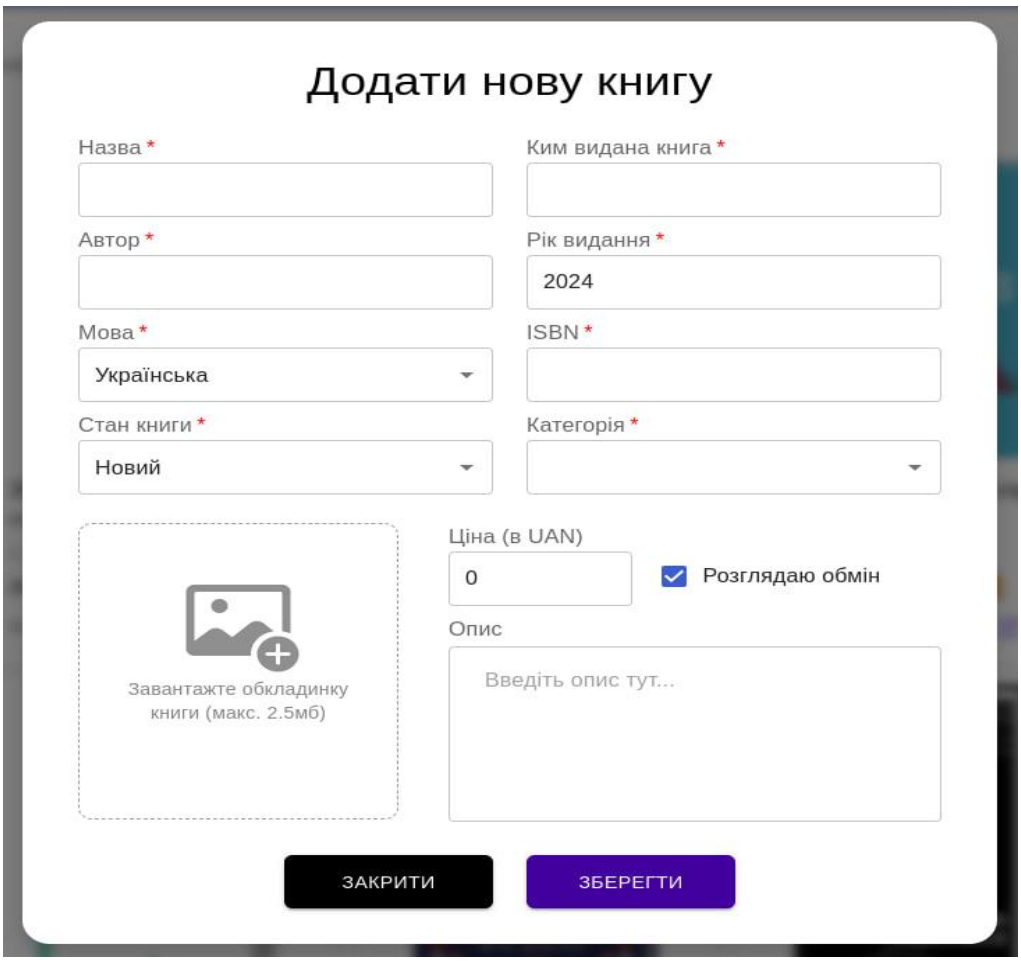

Рисунок Г.16 – Додавання нової книги

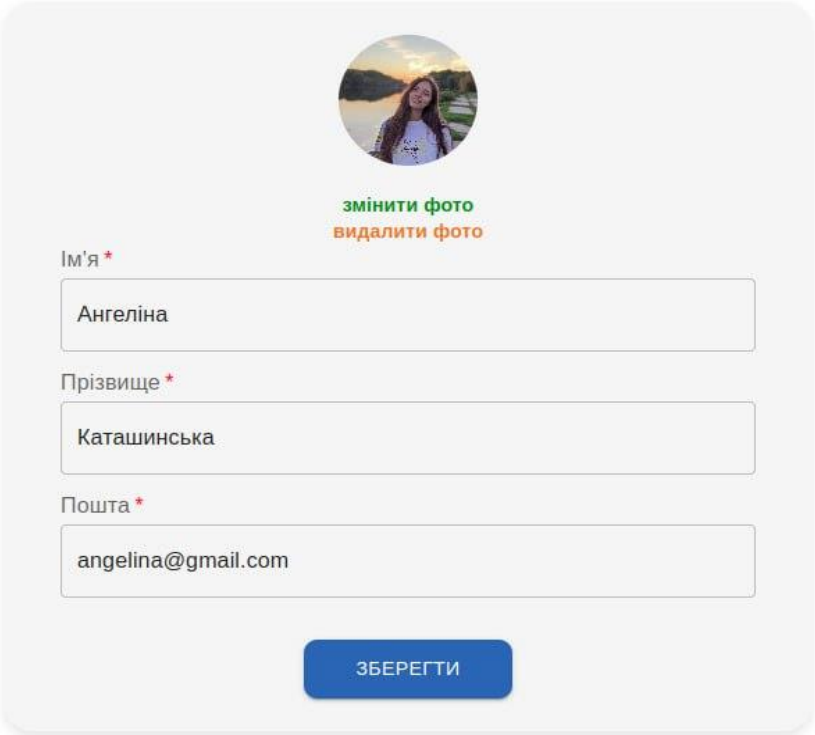

Рисунок Г.17 – Форма для оновлення даних користувача

ДОДАТОК Д
ДОВІДКА ПРО ВПРОВАДЖЕННЯ

ТОВ «Намір Ентерпрайз»
23310, Вінницька обл., Вінницький р-н., місто Гнівань., вул. Соборна, будинок 66
Тел +38 097 066 5320
email: namir.dev@gmail.com, https://namir.pro/

ДОВІДКА

Дана Каташинській Ангеліні Олександрівні в тому, що результати, одержані нею в процесі виконання бакалаврської дипломної роботи «WEB-ресурс для обміну книгами», а саме: форми (додавання нової книги та виставлення книги на обмін), систему відгуків та налаштування користувача планується використати в розробках ТОВ «Намір Ентерпрайз».

Генеральний директор
ТОВ «Намір Ентерпрайз» підпис Знаміровський А. А.