МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Секція інформаційно-комунікаційних технологій

ВИПУСКНА РОБОТА

на тему:

«REST API BookReviewer з використанням Flask»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Великодний Д.В**

Студент гр. ІН-71 **Шульга С.О.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедри Довбиш А.С.

"_____" _____2021 р.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-71 спеціальності "Комп'ютерні науки" денної форми навчання Шульга Станіслава Олексійовича.

Тема: «REST API BookReviewer з використанням Flask»

Затверджена наказом по СумДУ

№_____ от _____2021 р.

Зміст пояснювальної записки: 1) огляд та аналіз існуючих аналогів; 2) постановка завдання й формулювання завдань дослідження; 3) огляд технологій для реалізації завдання; 4) розробка REST API; 5) аналіз результату.

Дата видачі завдання "_____"_____2021р.

Керівник випускної роботи _____ Великодний Д.В.

Завдання прийняв до виконання _____ Шульга С.О.

РЕФЕРАТ

Записка: 55 стор., 33 рис., 7 табл., 1 додаток, 25 джерел.

Об’єкт дослідження — актуальність онлайн продажів.

Мета роботи — розробка REST API для залишання відгуків на книги онлайн.

Методи дослідження — метод аналітично-статистичний.

Результати — розроблено REST API BookReviewer з використанням Flask. API створено на базі мови програмування Python та фреймворку Flask.

REST API, PYTHON, FLASK, CELERY, SQLALCHEMY, REDIS,
PYDANTIC, FLASK-JWT-EXTENDED, BCRYPT

ЗМІСТ

ВСТУП

Робота присвячена темі аналізу та вибору засобів реалізації REST API. Аналіз та вибір засобів реалізації є одним з найважливіших етапів при створенні будь-якого веб-сервісу.

Було обрано розробку веб-серверної архітектури. Дана модель може забезпечити швидку комунікацію між користувачем й сервером, адже для роботи використовується шаблон Model-View-Controller. Таким чином структура додатку буде легко керована, матиме можливість внесення змін до існуючої логіки та додавання нового функціоналу.

Також, підчас огляду технологій реалізації звертається увага на вибір мови програмування, фреймворку чи бібліотеки, вибір СУБД, IDE, що будуть використані розробниками та ефективність у розробці кожного з них.

Метою кваліфікаційної роботи бакалавра є розробка реалізації REST API Book Reviewer. Тематика книг, а саме технічного напрямку, була обрана через складність пошуку інформації за даною тематикою.

Сформовано головні задачі реалізації REST API BookReviewer, що повинні бути реалізовані при виконанні дипломного проекту:

- ознайомлення з предметної областю, а саме функціюванням REST API;
- ідентифікація ідеї проекту;
- розробка вимог до функціювання REST API та прототипування календарного плану;
- розробка бази даних проекту;
- розробка програмної частини;
- наповнення контентом та тестування із вхідними даними.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

Сучасні покупці стикаються із, здавалося б, нескінченними варіантами. Як результат, багато хто з користувачів відчувають розчарування у покупках, приймають швидкі рішення або відкладають рішення про ту чи іншу покупку [1].

Оскільки перевантаження перш за все уваги користувача при виборі –реальна проблема для різних галузей, як для клієнтів, так і для бізнесу. На сьогоднішній час все більше підприємств шукають рішення, щоб допомогти своїм відвідувачам обрати та вибрати ідеальний продукт.

Розглянемо декілька способів, як це можна взагалі диференціювати та покращити загальне сприйняття. Всього було протестовано найперспективніші підходи для роботи із (табл.1.1).

Таблиця 1.1 – Способи допомоги користувачам при виборі

Назва	Опис	Тезисне формування теми
Рекомендації	Можна реалізувати динамічну взаємодію користувачів на основі попередніх пошукових запитів. Перш за все, переглядів сторінок або створених списків бажань, щоб тримати їх залученими та переглядати. Це також дозволяє збільшити загальні рівні продажів, лише пропонуючи додаткові товари чи аксесуари[2].	– Збільшення залучення клієнтів; – Заохочення для перегляду інформації; – Можливості перехресного посилання; – Робота із великими об’ємами даних й машинним навчанн;

Продовження таблиці 1.1.

Назва	Опис	Тезисне формування теми
Вікторини	Нерішучість й страх до зміну стилю чи тупи подачі інформації, щодо стилю чи підходу – формують дві найбільші перешкоди для рішення про придбання[3]. Допомогти вирішити певний перелік проблем може допомогти проведення вікторини. Даний спосіб може допомогти збільшити не тільки конверсію, а й продажі.	– Добре працює в короткий проміжок часу; – Створює розважальні мікро-взаємодії із системою; – Підвищує інформованість про систему;
Інтернет-огляди	Детальні та якісна реалізація відгуків дозволяють користувачам системи сформувати краще уявлення про продукт та дослідити можливі недоліки того чи іншого товару перед тим, як оформити замовлення [4].	– Розширює методи фільтрації й сортування продуктів за сформованим рейтингом; – Пропонується детальний огляд застосувань того чи іншого товару; – Можливість сформувати всі переваги та недоліки на основі зібраної інформації; – Підвищення обережність покупців, якщо виконуються незаконні маніпуляції;

Продовження таблиці 1.1.

Назва	Опис	Тезисне формування теми
Чат в режимі реального часу	Створення справжнього та корисного досвіду чату не тільки допомагає користувачам системи знайти потрібний товар чи послугу, але й отримати певну довіру.	– Розширення методи швидкого пошуку та підбір; – Підвищення рівна обслуговуванні користувач системи;
Соціальні медіа	Приведена невелика статистика, що 67% споживачів використовують канали соціальних мереж для обслуговування споживачів. Вони використовуються для швидкого реагування на запити, але натомість 42% користувачів очікують відповіді навіть протягом години [5].	– Добре для обслуговування клієнтів в онлайн режимі; – Не ідеальне рішення для вдосконаленого прийняття рішень;

Звичайно, інформаційні технології завжди сприяють розвитку різних інновацій у різних сферах. Інновації призводять до продвинених додатків із різними функціями, покращеного зберігання великих об’ємів даних, швидшого оброблення та розповсюдження інформації. Все це змушує всі підприємства працювати значно ефективніше.

За допомогою інформаційних технологій було створено й такі радикальні зміни в певних сферах:

– інтернет-покупки стали ефективніші, ніж покупки в звичних магазинах;

– цифровий маркетинг став ефективнішим, ніж будь-яка реклама в газетах, телебаченні чи радіо;

– соціальні мережі ефективніші;

– хмарні обчислення стали значно ефективніші, ніж представлені приватні комп’ютерні мережі.

Хороші рішення в бізнесі ґрунтуються на надійному дослідженні ринку. Отже, було обрано розробити REST API для роботи із оглядами та відгуками. Тематика даної REST API – література технічного напрямку.

1.2 Аналіз аналогів

У світі сучасних технологій ми маємо безліч можливостей та безліч реалізацій одних й тих же задач. Розглянемо декілька прикладів веб-додатків, що мають модулі, що будуть реалізовані при розробки REST API BookReviewer:

– «GoodReads» [6];

– «NetGallery» [7];

– «BookDepository» [8];

На рис.1.1 представлена головна сторінка веб-сервісу по продажу книг. На даному сайті можна переглянути інформацію про будь-яку книгу, що додана до даної системи.

Розглянемо реалізацію оформлення відгуків. Як представлено на рис.1.2, користувачеві представлено мало інформації. Користувачі не мають можливості залишити розгорнуту відповідь про ту чи іншу книгу.

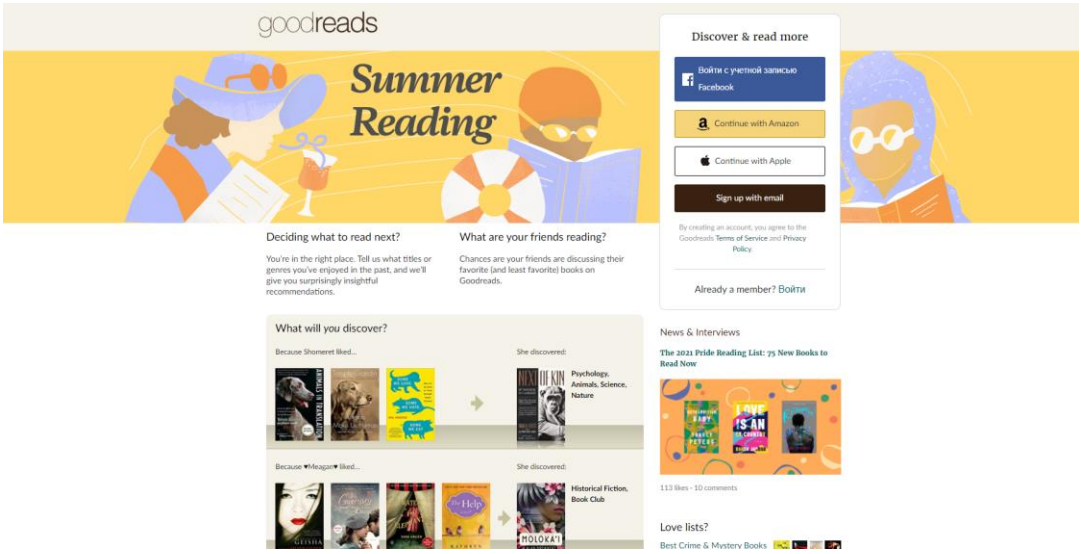

Рисунок 1.1 – Головна сторінка веб-додатку «GoodReads» [6]

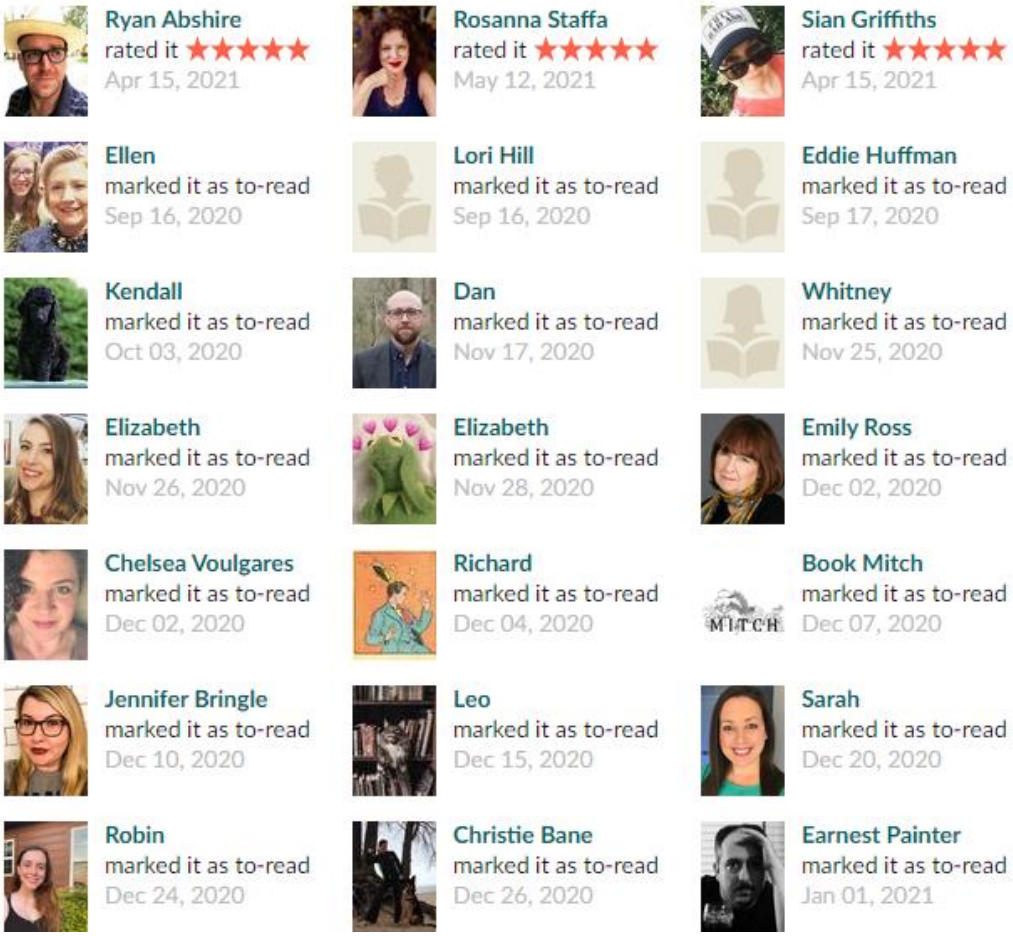

Рисунок 1.2 – Рейтинг оцінювання книги [6]

Наступний приклад – головна сторінка сайту «NetGallery» (рис.1.3). Інформація на сайті гарно згрупована за темами та напрямками, що не заважає при виборі.

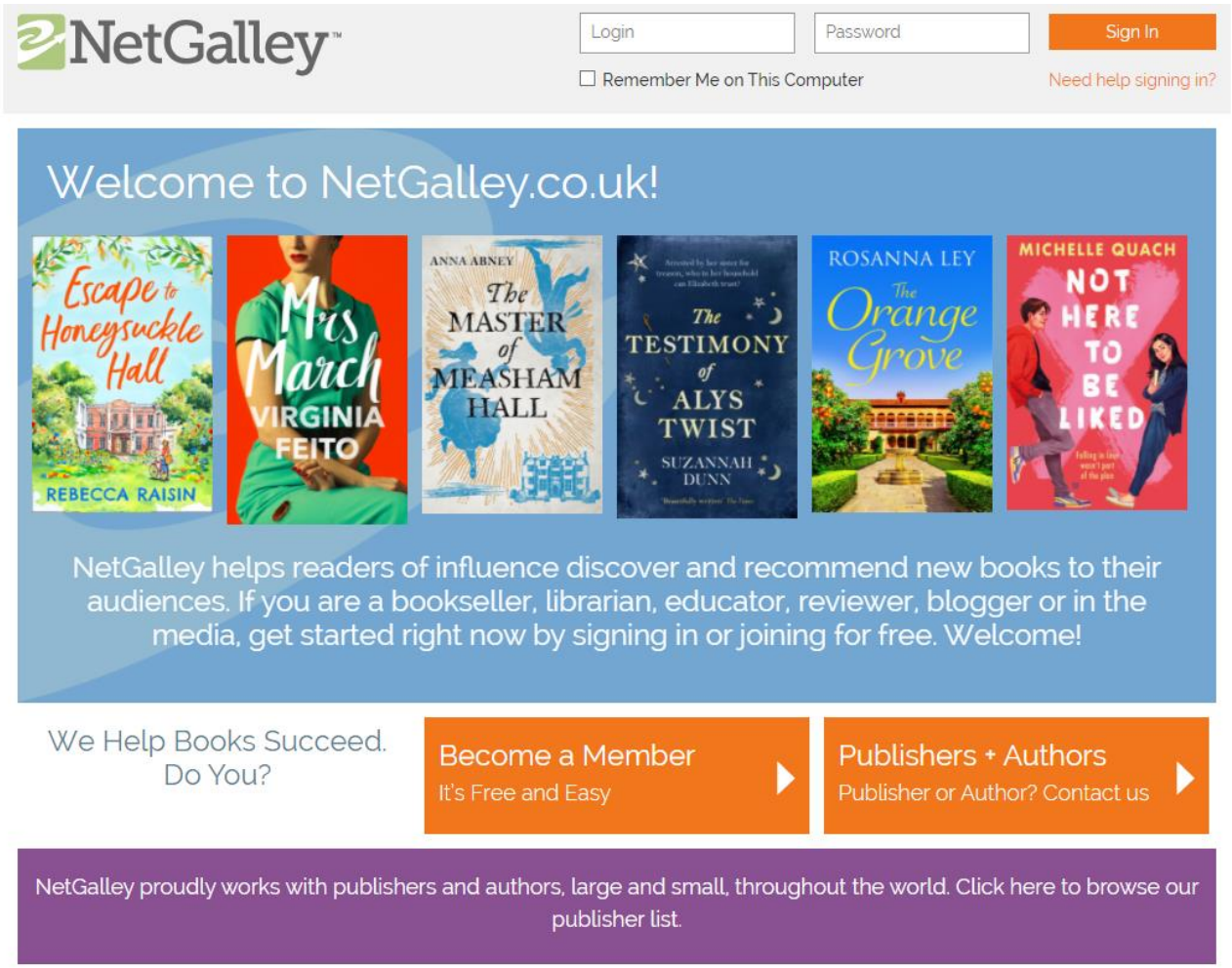

Рисунок 1.3 – Головна сторінка веб-додатку «NetGallery» [7]

На рис.1.4 зображено представлення частини інформації про книгу та середня оцінка від користувачів. Головний недолік – відсутність інформації про користувачів, що оцінили товар.

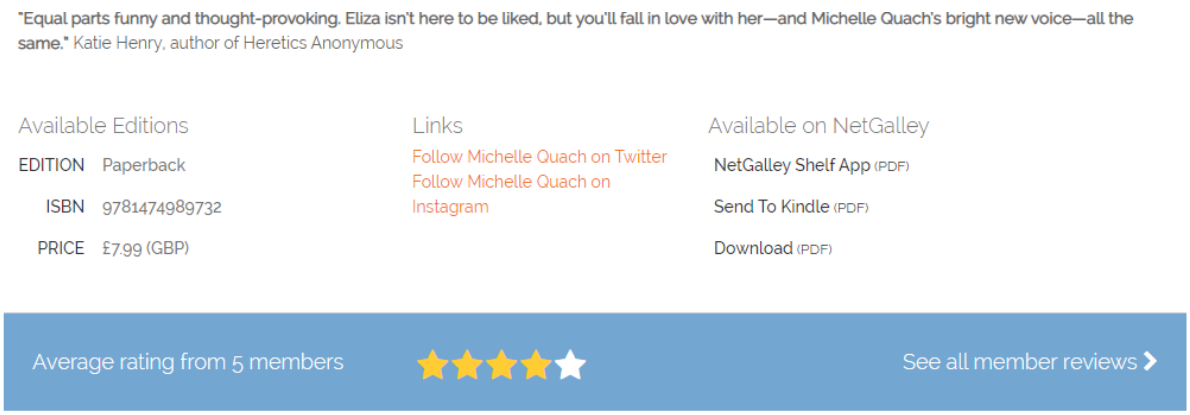

Рисунок 1.4 – Інформація про обраний товар

Останній приклад для аналізу обраної тематики – веб-додаток «BookDepository». На головній сторінці продемонстрована актуальна інформація про сезонні акції (рис.1.5).

На мою думку, на сайті представлено занадто великий діапазон можливих категорії (рис.1.6), що недоліком. Адже зайва інформація заважає користувачеві зосередитися та зробити вибір.

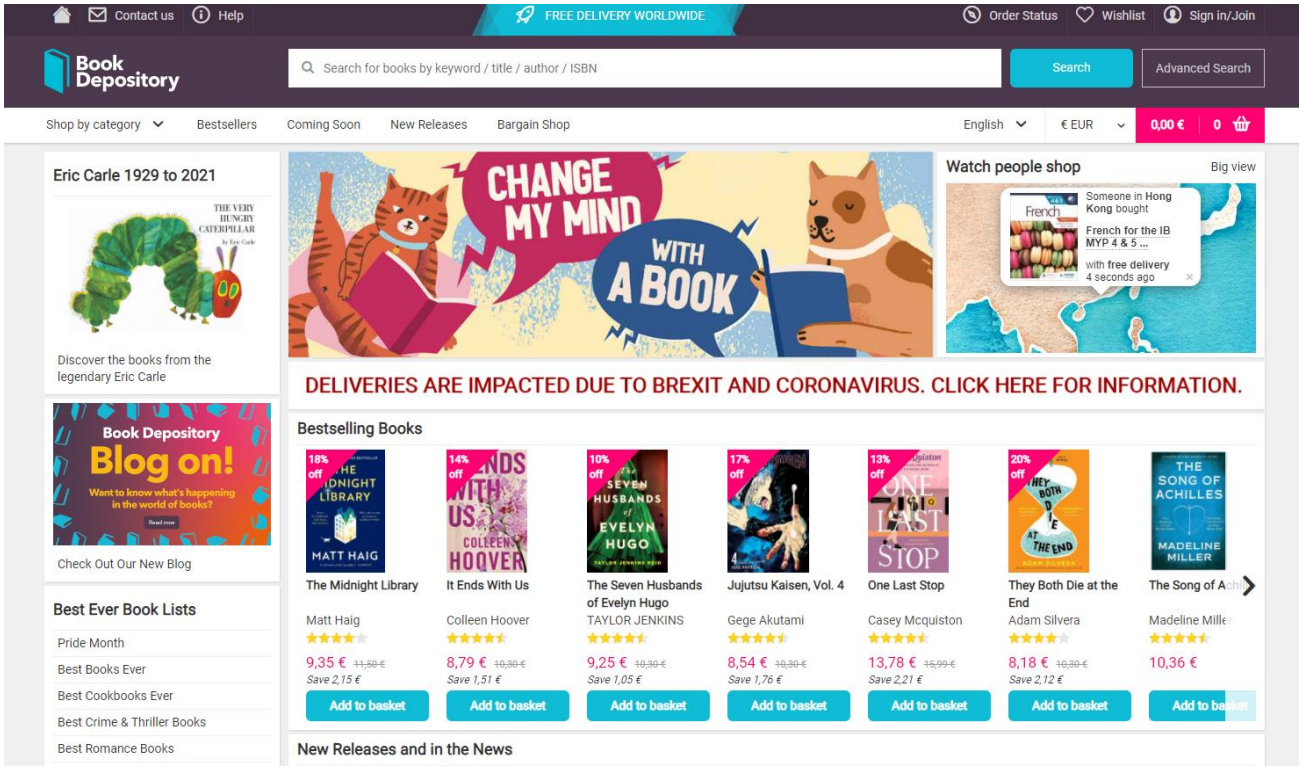

Рисунок 1.5 – Головна сторінка веб-додатку «BookDepository» [8]

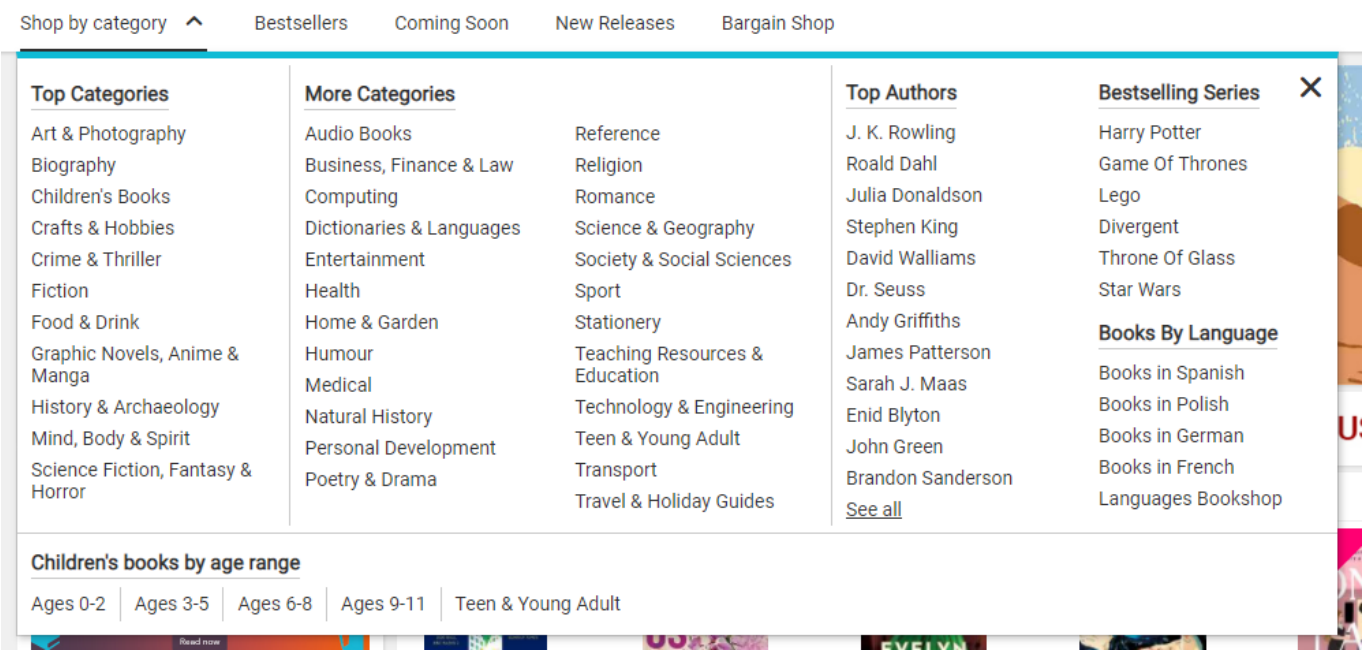

Рисунок 1.6 – Сортування за категоріями [8]

Розглянемо приклад на рис.1.7. Із позитивного, можна відмітити, що про товар присутня вся потрібна інформація, така як: опис, рейтинг, формат, автори та інше.

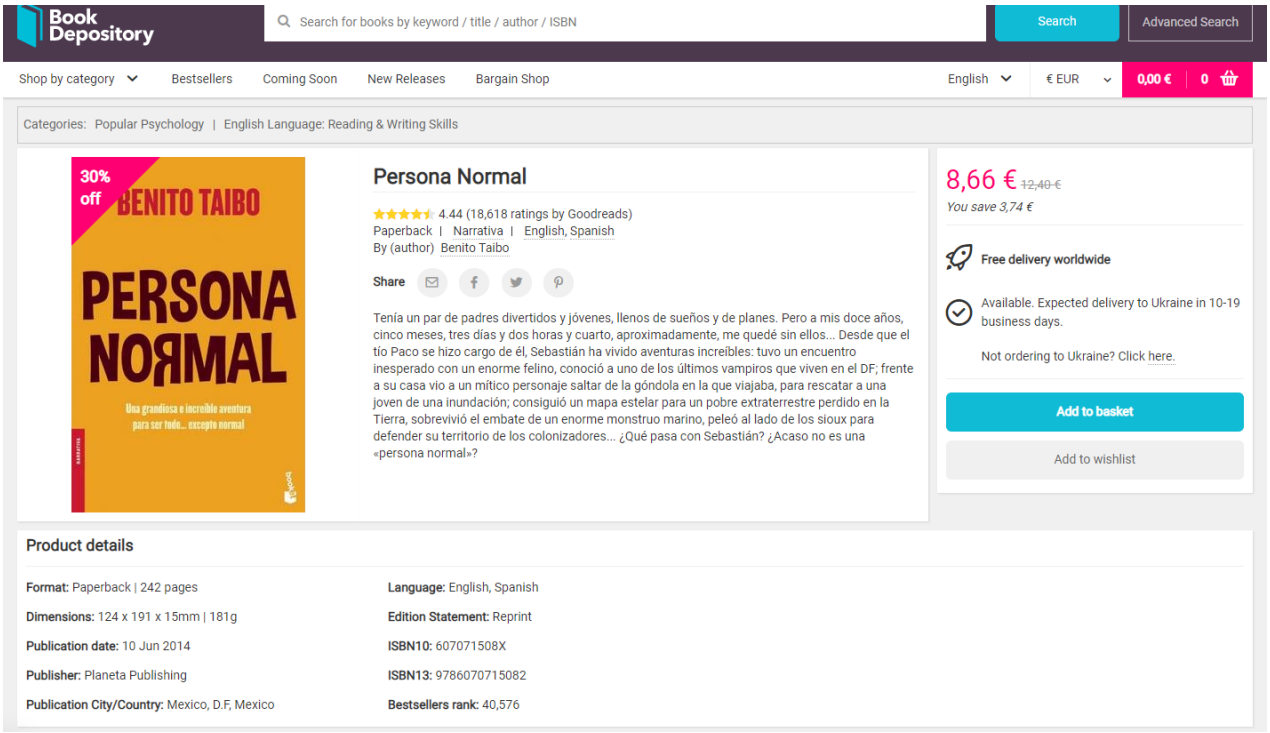

Рисунок 1.7 – Приклад інформації про товар [8]

Хоча, не дивлячи на це, на даному сайті також відсутні відгуки, та пояснення поставленої оцінки товару (рис.1.8).

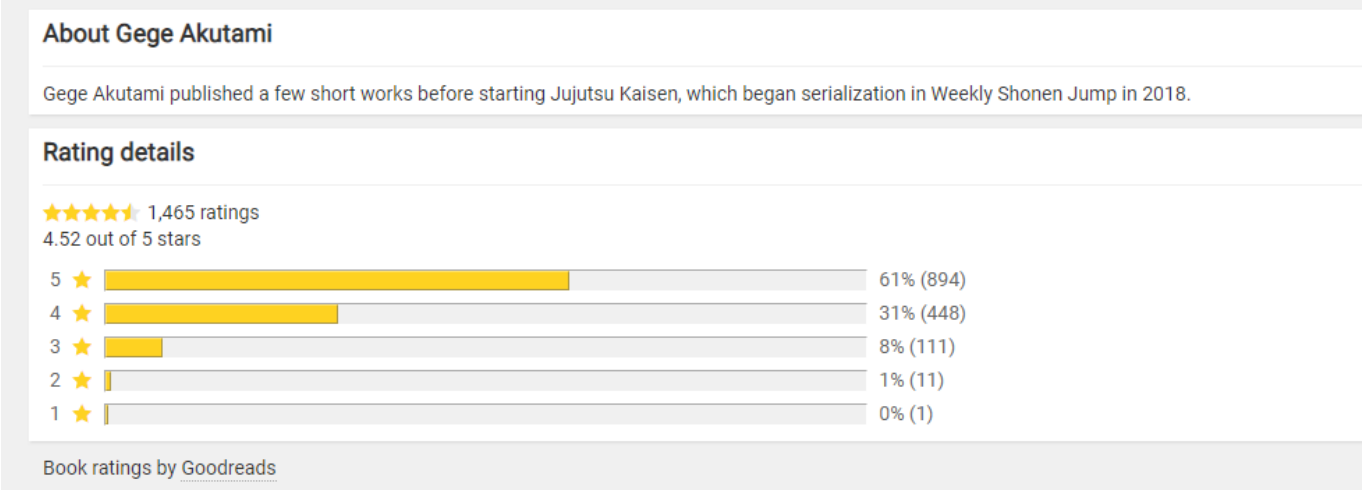

Рисунок 1.8 – Детальна інформація про оцінювання [8]

Отже, у результаті аналізу модулів, які використовуються у веб-додатках та які будуть реалізовані при розробки REST API BookReviewer, можна зробити висновок, що обрані функції реалізовані не в повній мірі.

Проаналізована інформація буде використовуватися при розробці дипломного проекту.

1.3 Постановка задачі проекту

Мета проекту – розробка REST API BookReviewer з використанням Flask. Головні задачі, що повинні бути реалізовані:

- ознайомлення з предметної областю, а саме з функціюванням REST API;
- аналіз використання;
- ідентифікація ідей проекту;
- розробка вимог до функціювання REST API;
- розробка бази даних проекту;
- розробка програмної частини;

- наповнення контентом розроблену REST API;
- тестування із вхідними даними.

Необхідно розглянути функціонал, що буде реалізовуваний REST API BookReviewer:

- авторизація, реєстрація та зміна пароля;
- додати огляд на книгу;
- отримання інформації по якійсь книзі;
- отримання всіх книг;
- отримання всіх категорій технічних книг;
- отримання всіх книг з якоїсь категорії;
- подивитися всі книги створені цим користувачем;
- поставити «лайк» книзі;
- прибрати «лайк» з книги;
- підписка на нові книги в певній категорії.

2 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ REST API

2.1 Вибір засобів реалізації

При виборі засобів реалізації дипломного проекту, перш за все, виконаємо порівняння фреймворків для розробки програмної частини. Для порівняння було обрано найпопулярніші на 2021 рік фреймворки мови програмування python – «Flask» (табл.2.1) й «Django» (табл.2.2).

Таблиця 2.1 – Переваги та недоліки «Flask»

№	Переваги	Недоліки
1	Вбудована підтримка модульного тестування [9].	Пропонує обмежену підтримку та меншу кількість спільноти порівняно з Django.
2	Використовується механізм шаблонів Jinja2.	Створення великого проекту вимагає певних попередніх знань про структуру та архітектуру.
3	Відправлення запиту RESTful.	Складне обслуговування для більших реалізацій [10].
4	Багато вичерпної документації.	Вищі витрати на обслуговування складніших систем.
5	API мають гарну форму та цілісність.	
6	Підтримує архітектуру MVC.	
7	Мінімалістична, проте потужна платформа.	

Таблиця 2.2 – Переваги та недоліки «Django»

№	Переваги	Недоліки
1	Пропонує архітектуру MVC .	Компоненти розгортаються разом, що може створити плутанину при розробці.
2	Підтримка інтерфейсних інструментів, таких як Ajax, jQuery, тощо.	Монолітна платформа
3	Підтримка декількох баз даних.	Документація часто не охоплює реальних сценаріїв.
4	Дані, змодельовані за допомогою класів Python [11].	Менше архітектурних рішень та компонентів.
5	Присутня оптимізація сайту на спеціалізованих серверах.	

У результаті аналізу можна робити висновок, що «Flask» краще підходить для розробки API, тому що розробник сам обирає підхід до розробки моделі та структури даних, тоді як «Django» краще підходить для великих проектів, де краще мати заздалегідь визначену структуру проекту.

У результаті порівняння було обрано «Flask», адже це хороший вибір, якщо потрібна полегшена кодова база, яку можливо розширити по потребі.

Наступний етап – обрання інструменту для роботи з чергами [13,14]. Розглянемо найпопулярніші рішення для поставленої задачі, а саме «Redis»(табл.2.3) та «RabbitMQ»(табл.2.4).

Таблиця 2.3 – Переваги та недоліки «Redis»

№	Переваги	Недоліки
1	Сучасна база значень ключ-значення в пам'яті.	Відсутність інструментів моніторингу та адміністрування.
2	Відмінна підтримка багатьох різних типів структури даних.	Споживання системних ресурсів під час масштабування.
3	Простота в налаштуванні та роботі	Висока ціна постійної підписки.
4	Варіантів конфігурації багато, програмування потрібно небагато.	Redis має дуже прості можливості пошуку, що означає, що він не підходить для всіх випадків використання.
5	Немає необхідності в операціях з технічного обслуговування. Після його налаштування працює бездоганно.	
6	Простий у використанні та підтримує найпоширеніші випадки використання кеш-пам’яті.	

Таблиця 2.4 – Переваги та недоліки «RabbitMQ»

№	Переваги	Недоліки
1	Добре інтегрується	Не так багато інструментів
2	Здатний надати графічний, а також табличний аналіз	Документація потребує вдосконалення, пояснюючи, як налаштувати вищезазначені функції.
3	Простота налаштування при початку роботи	Важко потрапити у внутрішні структури

Продовження таблиці 2.4.

№	Переваги	Недоліки
4	Обробка черг мертвих листів та надання гнучкості для створення власних систем мертвих листів.	Затримка закінчення черги може призвести до того, що RabbitMQ зупиниться
5	Можливість отримання точних результатів роботи.	Не включений у багато корпоративних систем

У результаті порівняння було обрано використовувати «Redis», адже він легкий у використання та має всі потрібні функції для реалізації теми дипломної роботи.

Важливий інструмент при розробці даного проекту – інструмент надсилання та отримання повідомлень [12, 15]. Отже розглянемо можливі способи реалізації, а саме «Celery» (табл.2.5) та «RQ» (рис.2.6).

Таблиця 2.5 – Переваги та недоліки «Celery»

№	Переваги	Недоліки
1	Присутнє безкоштовне програмне забезпечення.	Відсутність планів підтримки.
2	Проста установка: оскільки вона настільки проста і легка	Інтеграції можуть бути складними для практичної роботи.
3	Підтримує посередників з кількома повідомленнями.	Складнощі при використанні набору функцій.
4	Може інтегруватися з кількома веб-фреймворками.	
5	Архітектура базується на чергах завдань.	

Таблиця 2.6 – Переваги та недоліки «RQ»

№	Переваги	Недоліки
1	RQ Документація є всеосяжною.	Відсутня брокерська підтримка.
2	Модель пріоритетною черзі проста і ефективна[16].	Працює тільки на UNIX системах.
3	Простий RQ API.	Підтримує тільки Python.
4	Активними проект, стабільний.	

Отже, у результаті порівняння було обрано Celery, адже це універсальний інструмент, що має безкоштовне програмне забезпечення та може інтегруватися кількома фреймворками.

Також, окрім описаних вище інструментів, для розробки REST API Book Reviewer буде використано SQLAlchemy та Bcrypt.

SQLAlchemy – набір інструментів бази даних та реалізація об'єктно-реляційного відображення, що також називають ORM [17]. Даний інструмент також написана на Python. SQLAlchemy забезпечує узагальнений інтерфейс для створення та виконання створеної бази даних без необхідності писати оператори SQL.

Наступний інструмент – Bcrypt. Кожному відомо, що в онлайновому світі паролі відіграють важливу роль у захисті особистих даних. Виходячи і цього, дуже важливо забезпечити надійних захист паролів та авторизації у ситемі.

Отже, було обрано використовувати при розробці REST API Bcrypt. Це ключова функція виведення , яку можна розглядати як хеш-функцію [18]. Мета даного інструменту в перетворенні шматка вхідних даних у фіксований розмір, детермінований й непередбачуваний для шахраїв результат. Поширеним випадком, що також буде використовуватися при розробці REST API, є перетворення пароля в криптографічний ключ, який потім можна використовувати для безпечної автентифікації [19].

2.2. Проектування бази даних

Робота присвячена темі аналізу та вибору засобів реалізації REST API Book Reviewer. Звичайно, аналіз та вибір засобів реалізації є одним з найважливіших етапів при створенні будь-якого веб-сервісу.

Так як REST API не матиме графічного представлення, дуже важливо якісно відтворити взаємодію сутностей бази даних. Важливою частиною також є вірне іменування атрибутів кожної із таблиці, адже це дозволить полегшити розробку та експлуатацію REST API іншим розробникам у майбутньому.

Отже, було створено перелік сутностей REST API:

– categories: категорії книг;
– users: зареєстровані користувачі;
– book_reviews: відгуки про книги;
– subscription: оформлені підписки;
– reaction: додаткові реакції;
– sqllite_master: додаткова таблиця налаштувань.

Зобразимо сформовані відносини в ER-діаграмі (рис.2.1). На зображенні представленні всі відносини між таблицями, атрибути таблиці та представлення цих даних [20-21].

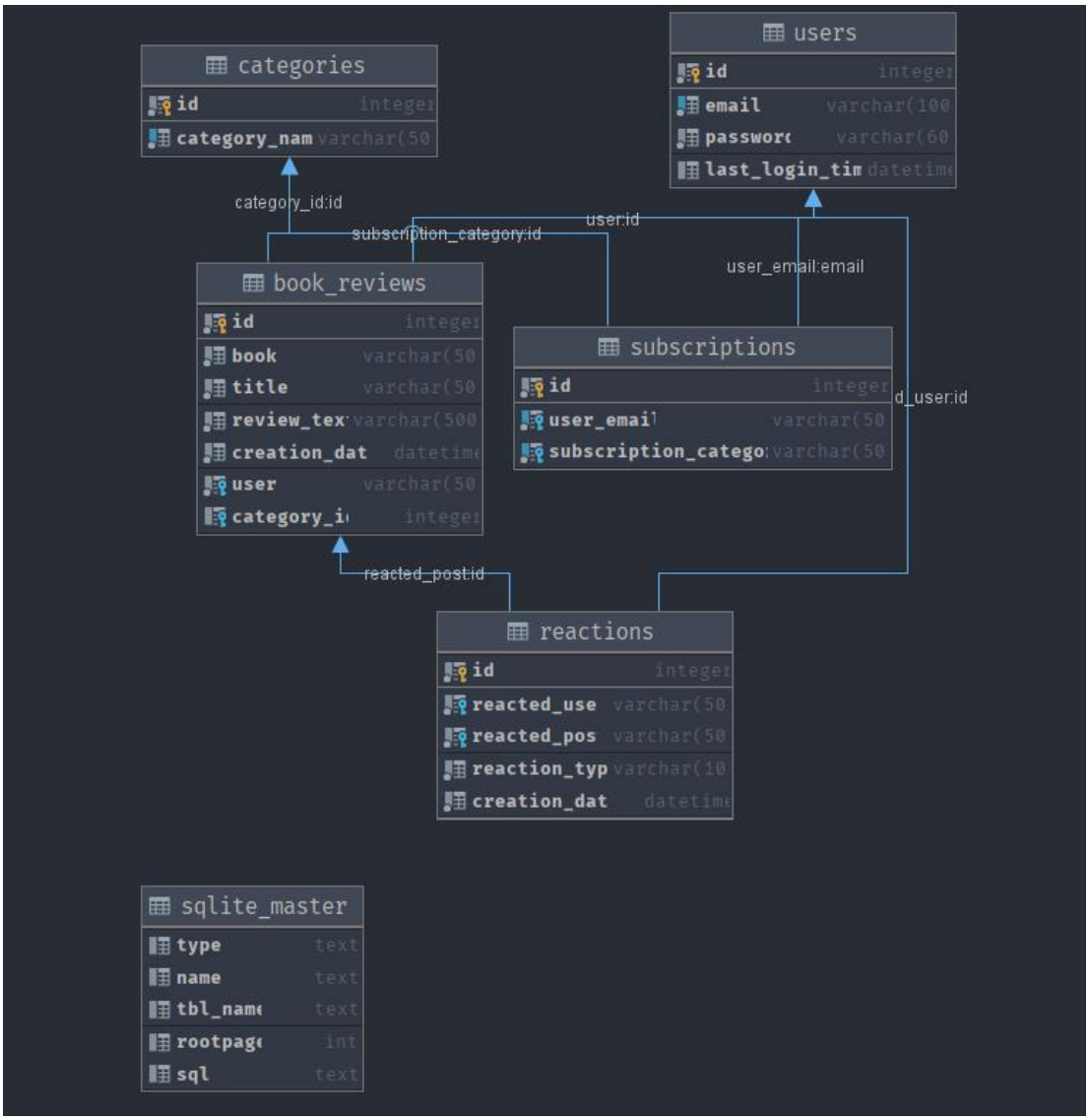

Рисунок 2.1 – REST API Book Reviewer ER-діаграма

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ REST API

3.1 Архітектура

Опис архітектури проекту є важливою частиною для розуміння того, як повинна функціонувати система. Нище приведена загальна схема роботи.

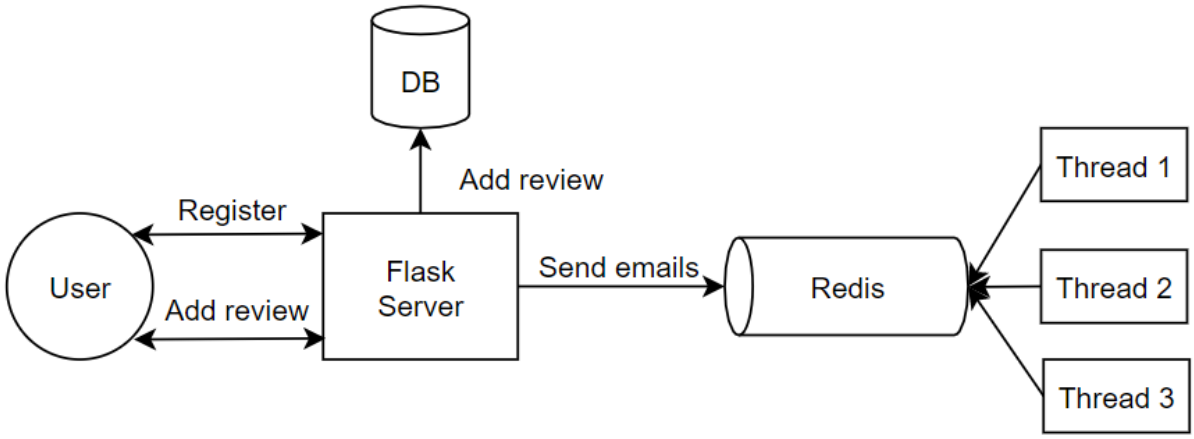

Рисунок 3.1 – Загальна схема роботи Flask, Redis та Celery:

– User: Користувач, що звертається до API. Користувач звертається до сервера за допомогою таких інструментів як браузер, Postman/Insomnia.

– Flask application: API, що відповідає на всі запити клієнтів. Це можуть буди запити на читання, запис, модифікацію або видалення даних. Якщо запит клієнта стосується лише його самого, клієнт отримає синхронну відповідь. Якщо запит пов’язаний із задачами, які можуть зайняти певний час, наприклад відправка електронних листів користувачам, що підписані на певні категорії відгуків, то Flask відправить цю задачу на опрацювання в Redis.

– DB: основна база даних, з якою взаємодіє API, у нашому випадку SQLite3.

– Redis: сховище структур даних в пам'яті, що використовується в якості бази даних, кеша і брокера повідомлень. Redis надає такі структури даних, як

рядки, хеші, списки, множини, сортовані множини [22]. У випадку з BookReviewer, буде використовуватися як брокер повідомлень.

– Celery: це реалізація черги завдань для веб-додатків Python, яка використовується для асинхронного виконання роботи поза циклом HTTP-запит-відповідь [23]. Черги завдань використовуються як механізм розподілу роботи між потоками або машинами. Входом черзі завдань є одиниця роботи, яка називається завданням. Виділені робочі процеси постійно стежать за чергами завдань в пошуках нової роботи. Celery обмінюється повідомленнями, зазвичай використовуючи брокер для посередництва між клієнтами і робочими процесами. Щоб ініціювати завдання, клієнт додає повідомлення в чергу, а брокер потім доставляє це повідомлення робочому процесу. Система Celery може складатися з декількох робочих і брокерів, що забезпечує високу доступність і горизонтальне масштабування [24]. В нашому випадку, Celery виконуватиме функцію асинхронної відправки повідомлень клієнтам сервісу.

3.2 Використання REST API

Для використання REST API BookReviewer та перегляду реалізації перш за все потрібно виконати запуск Celery (рис.3.2) та запуск Redis.

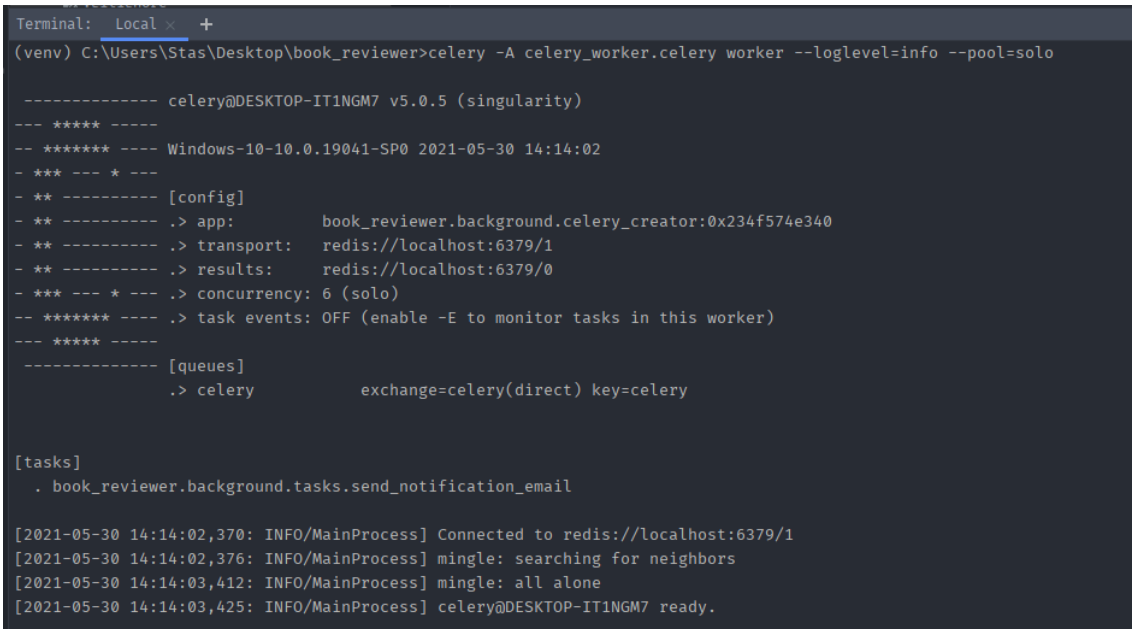

Рисунок 3.2 – Запуск Celery

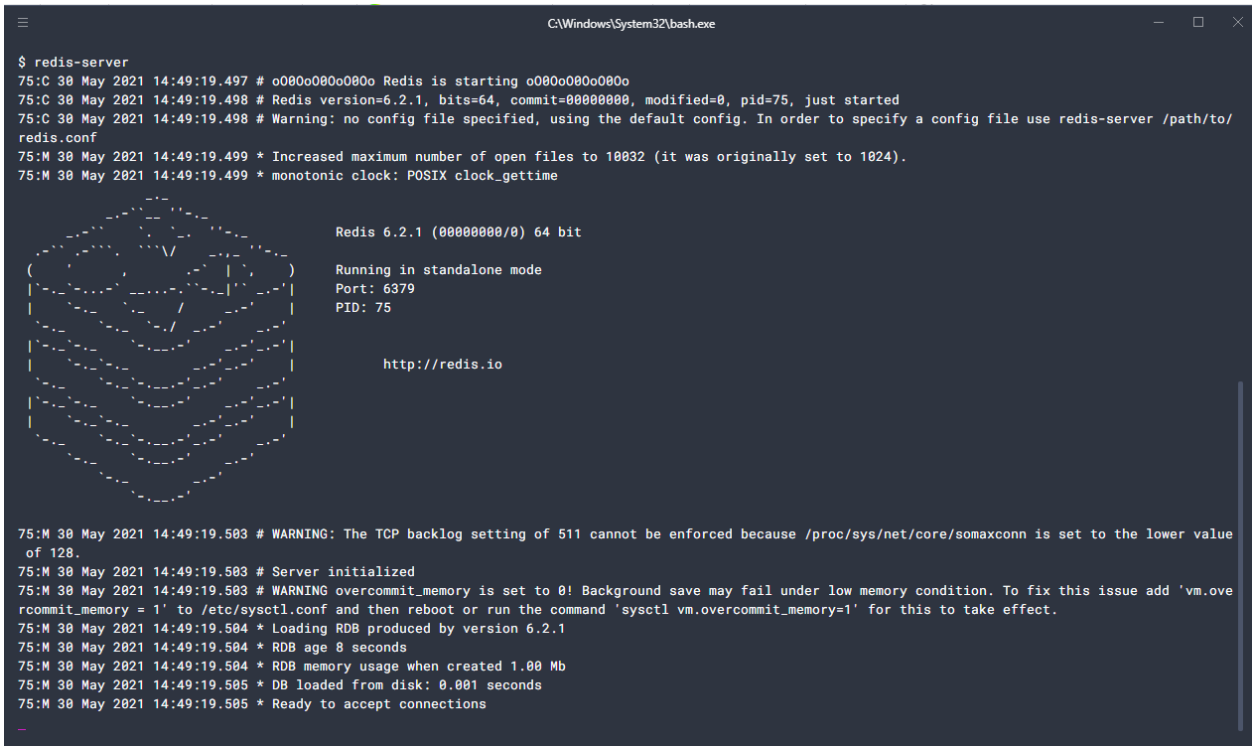

Рисунок 3.3 – Запуск Redis

Розгляне приклади використання REST API. На рис.3.3 представлений JSON із даними для реєстрації користувача. Справа ми можем побачити повідомлення,

що виводить статус нашого запиту. У даному випадку реєстрація користувача була успішною.

На даному етапі присутня валідація, що перевіє отримані дані від користувача. Прикладом є спроба зареєструватися із електронною поштою, що вже є у базі даних REST API (рис.3.4).

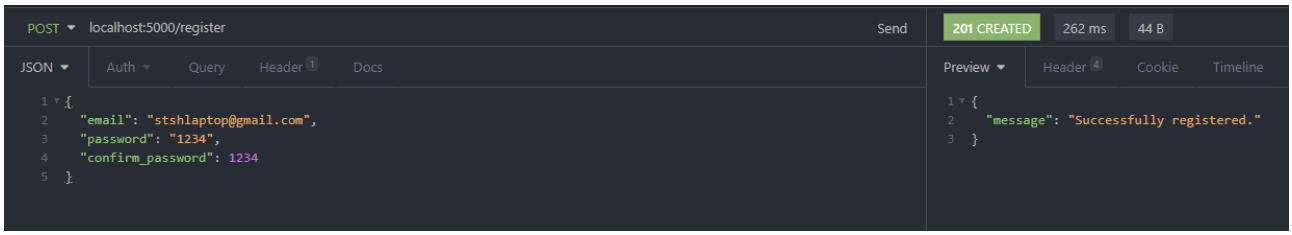

Рисунок 3.3 – Успішна реєстрація користувача

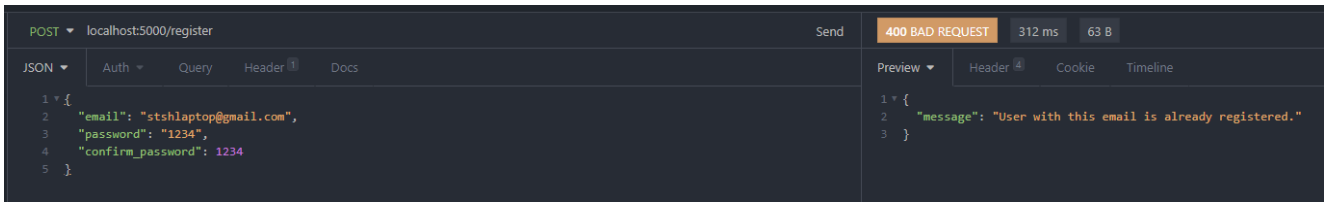

Рисунок 3.4 – Не успішна реєстрація користувача

Після реєстрації, переходимо до авторизації користувача. Для успішної авторизації передається електронна адреса та пароль (рис.3.5). Результатом є access та refresh токени.

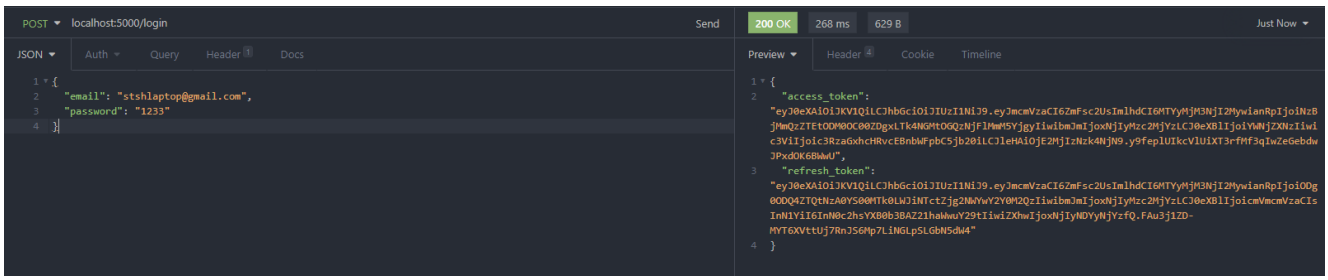

Рисунок 3.5 – Успішна авторизація користувача

На рис.3.6 представлено отримання нового access токена використовуючи refresh токен.

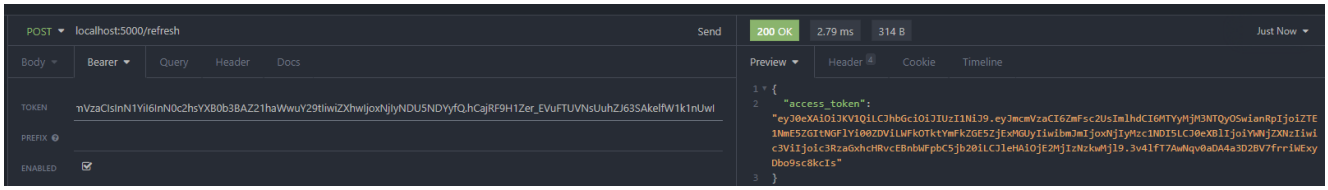

Рисунок 3.6 – Отримання refresh токену

Також користувач має можливість змінити пароль. Для цього потрібно передати таку інформацію: старий та новий паролі, та підтвердження нового паролю.

Розглянемо декілька прикладів. На рис.3.7 представлений варіант із вірно веденими даними.

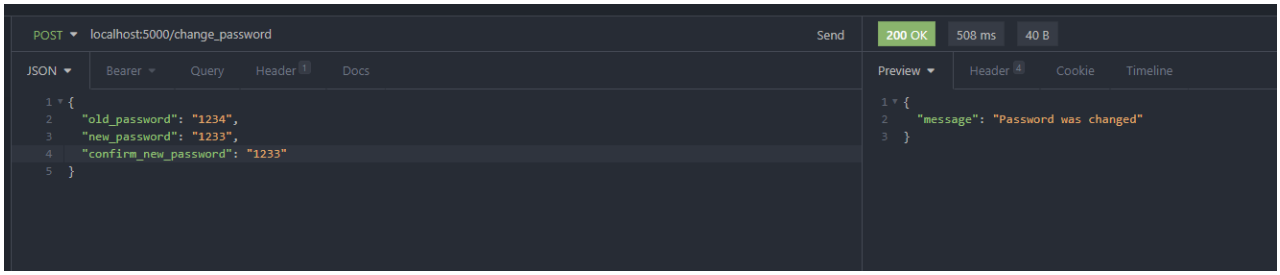

Рисунок 3.7 – Варіант зміни паролю із вірно веденими даними

Якщо користувач дублює інформацію, тобто пароль фактично не змінюється або він намагається передати не валідні значення, то зміни паролю не відбувається. Користувач отримує повідомлення про статус помилки (рис.3.8-9).

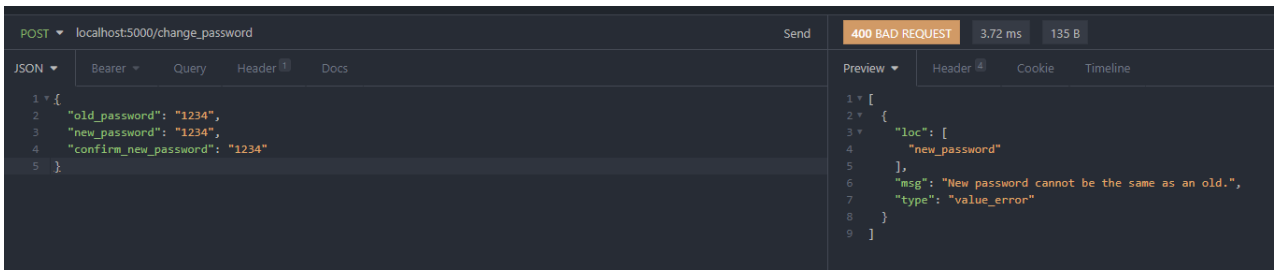

Рисунок 3.8 – Варіант зміни паролю із не вірно веденими даними (повторенням старого паролю)

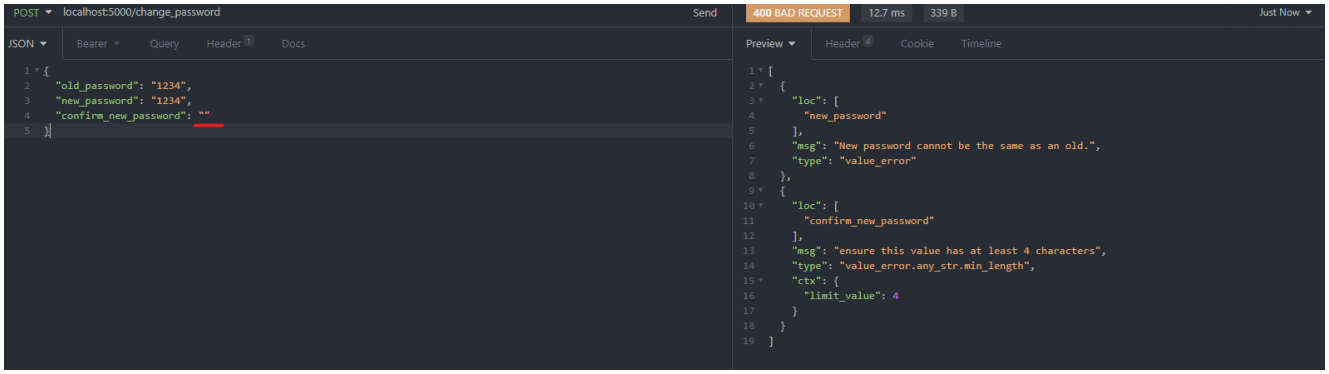

Рисунок 3.9 – Варіант зміни паролю із не вірно веденими даними (не валідні значення)

Вірно зареєструвавшись та авторизувавшись у системі користувач може переглянути всі відгуки за певною категорією книг (рис.3.10). Відгук представляє з себе назву книги, категорію до якої вона відноситься, дату створення відгуку, ідентифікаційний, заголовок та сам відгук.

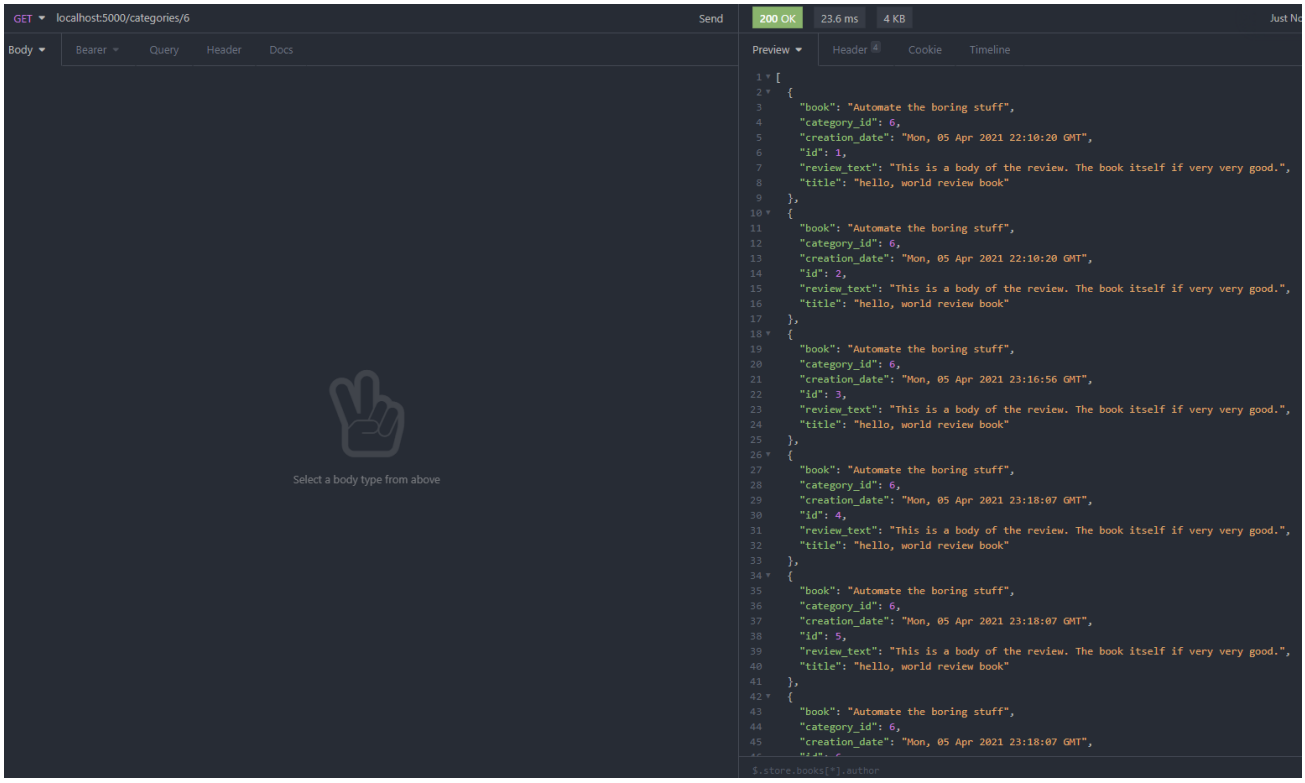

Рисунок 3.10 – Перегляд відгуки за певною категорією книг

Розглянемо всі можливі категорії на рис.3.11. На даному етапі розробки REST API BookReviewer додано всього 10 категорій, наприклад system design, project management, programing та інше.

Рисунок 3.10 – Категорії книг

Можна переглянути всі відгуки не залежно від категорії (рис.3.11). Відбувається відображення із використання пагінації.

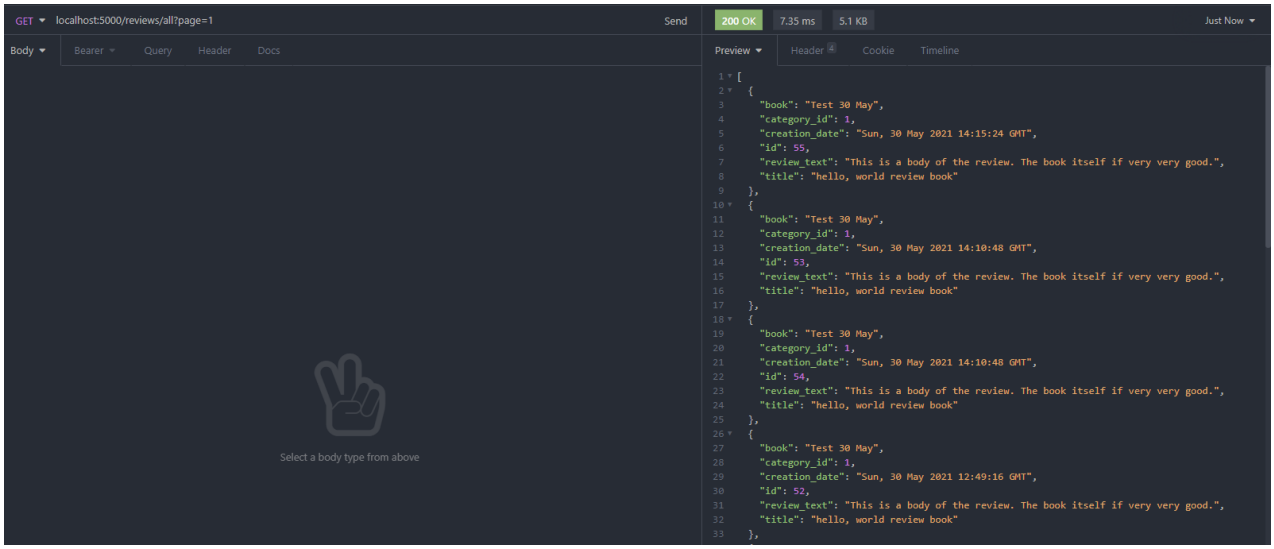

Рисунок 3.11 – Відображення із використання пагінації

Також присутня додаткова можливість перегляду, як всіх відгуків залишеним користувачем (рис.3.12), так й від інших (рис.3.13).

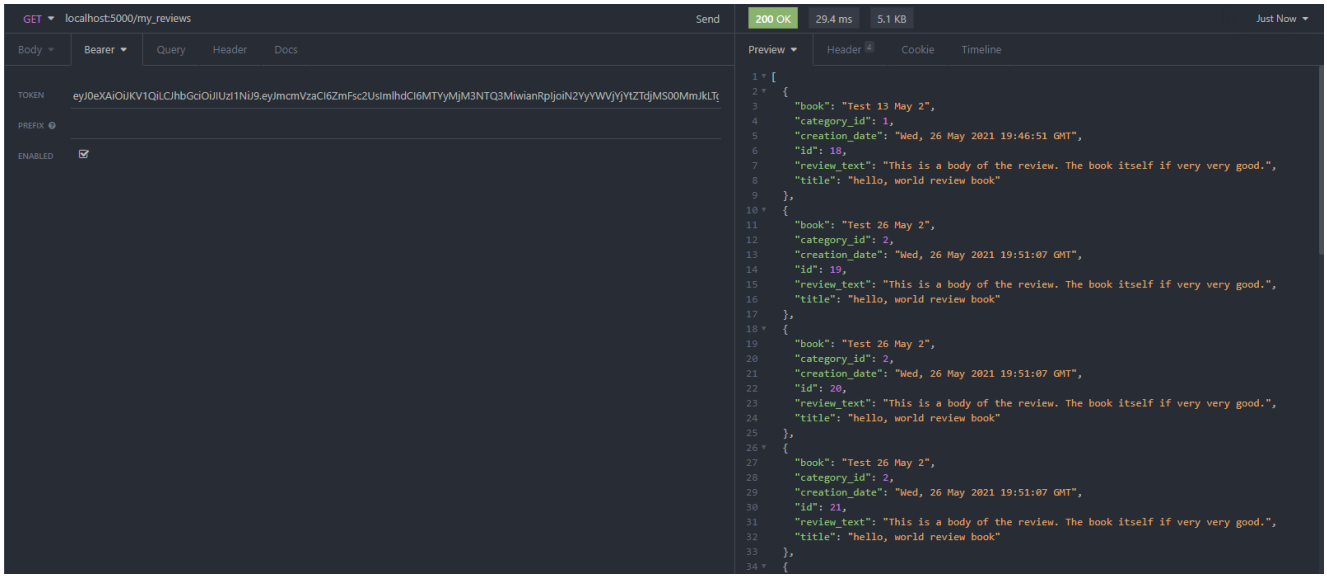

Рисунок 3.12 – Власні відгуки

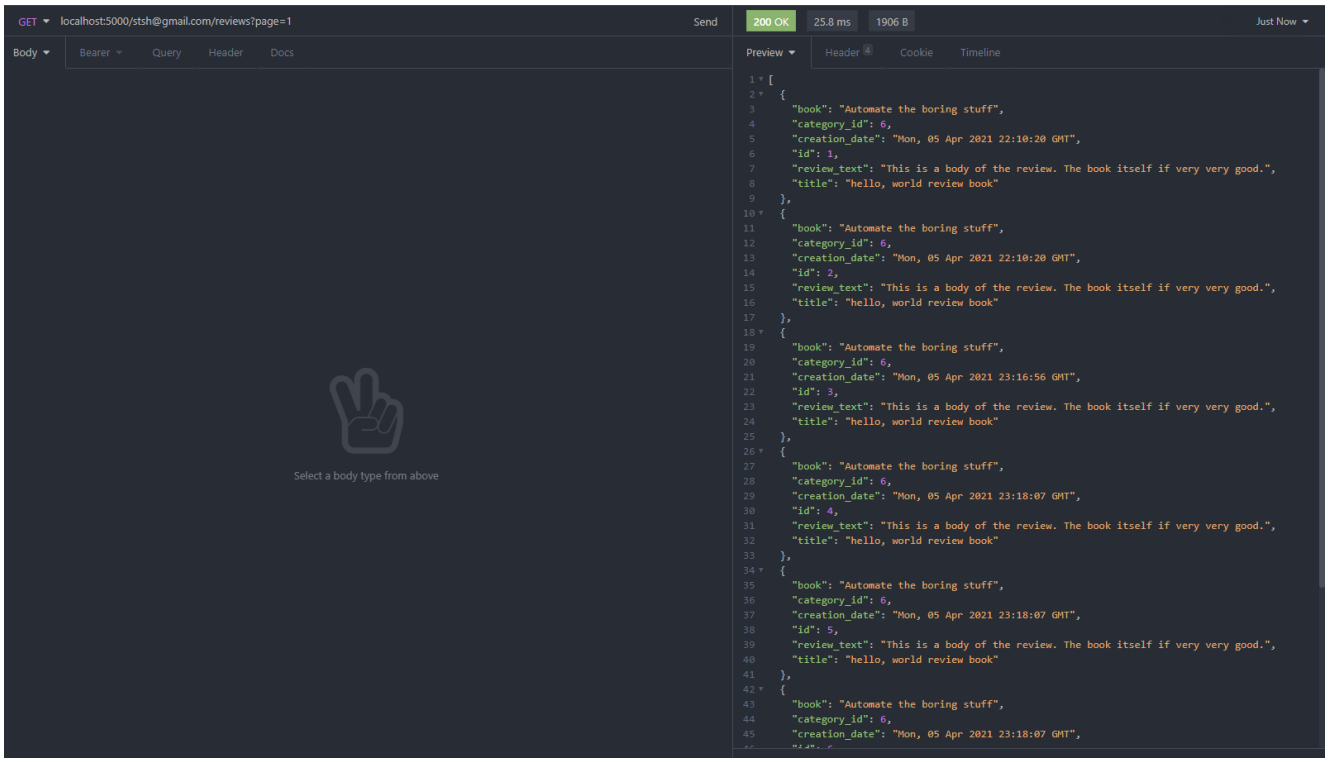

Рисунок 3.13 – Відгуки іншого користувача

Крім фільтрація за користувачем, присутня фільтрація з книгою, а саме за її назвою (рис.3.14).

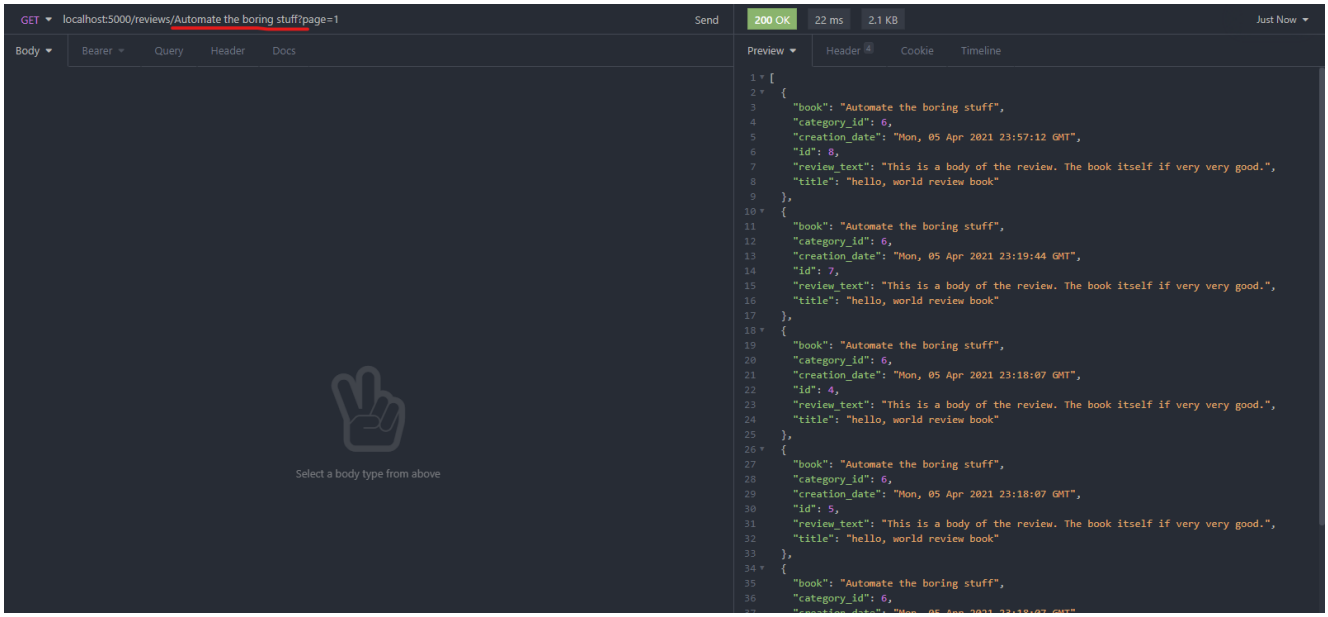

Рисунок 3.13 – Відгуки за певною книгою

Для того щоб оцінити певний відгук потрібно передати ідентифікаційний номер обраного відгуку (рис.3.14). За такою же системою оцінювання можна прибрати використовуючи вже не POST метод, а DELETE (рис.3.15).

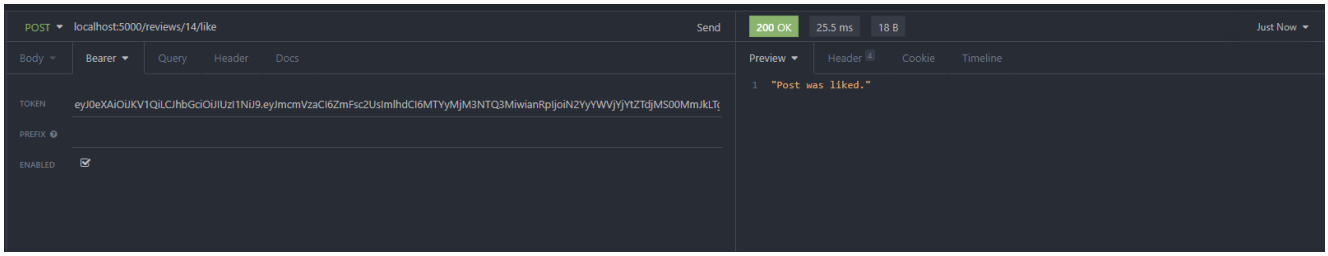

Рисунок 3.14 – Приклад додавання оцінювання

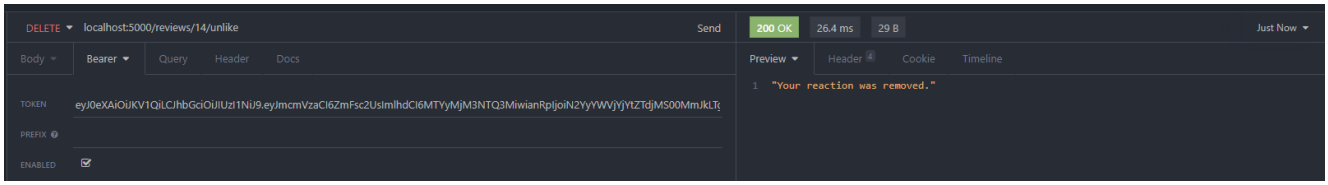

Рисунок 3.14 – Приклад зняття оцінювання

Звичайно користувач має можливість підписатися на категорію для отримання нової актуальної інформації. Інформація отримується за допомогою отримання повідомлень у системі.

Приклад представлений на рис.3.15. У списку відображається не сама назва, а ідентифікаційний номер обраної категорії.

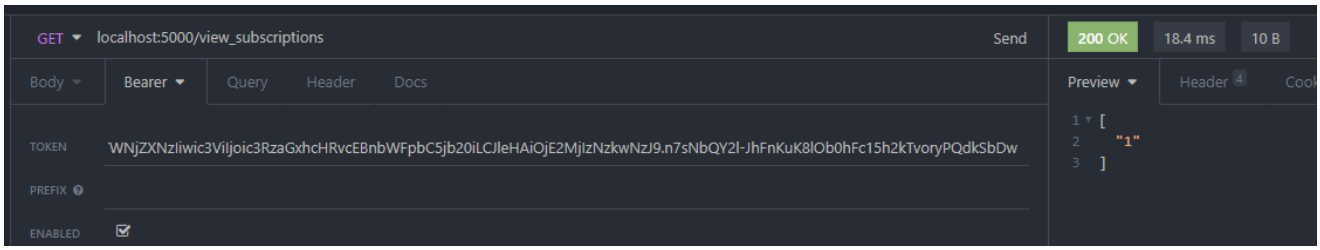

Рисунок 3.15 – Приклад списку підписок на категорії

Приклад оформлення підписки на рис.3.16. Дана функція реалізована через POST метод. Користувач отримує повідомлення про результат виконаних дій.

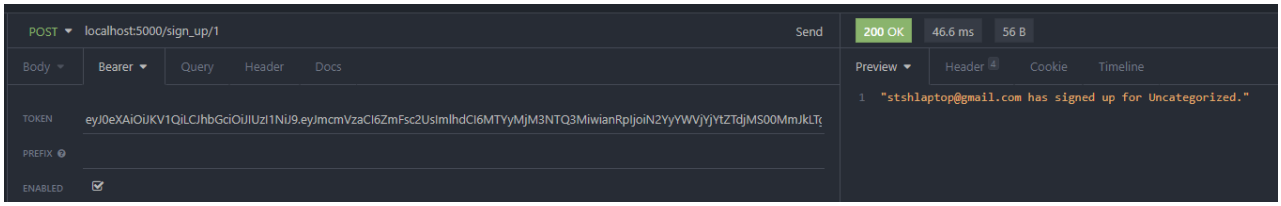

Рисунок 3.16 – Оформлення підписки

Присутня також функція для видалення певної категорії із підписки (рис.3.17). Виконавши відписку від категорії, користувач не буде отримувати повідомлення за даним напрямком.

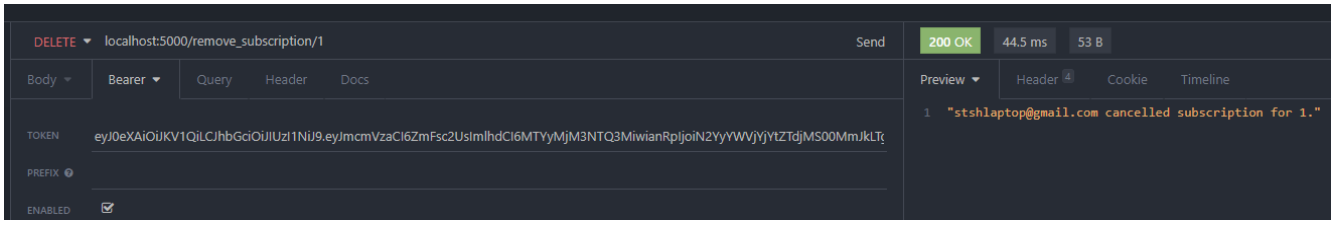

Рисунок 3.17 – Оформлення відписки

Одна із найголовніших можливостей користувача – додавання відгуків на книги. Розглянемо декілька прикладів. Безпомилковий запит на додавання відгуку, що представлений на рис.3.17.

Звичайно, будь-який користувач може припустися помилки та невірно заповнити дані. На ці випадки реалізована перевірка: перевірка на валідність значення категорії та заповнення всіх обов'язкових атрибутів JSON (рис.3.18-19).

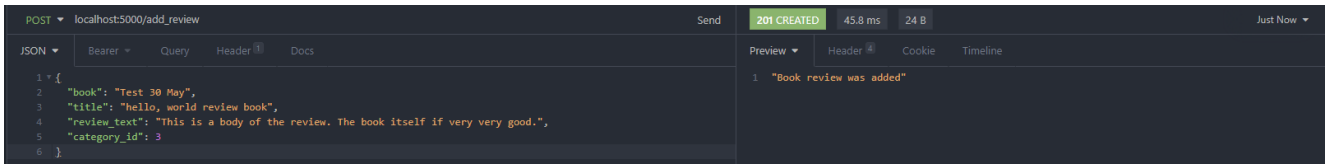

Рисунок 3.17 – Приклад вірного запиту на додавання відгуку

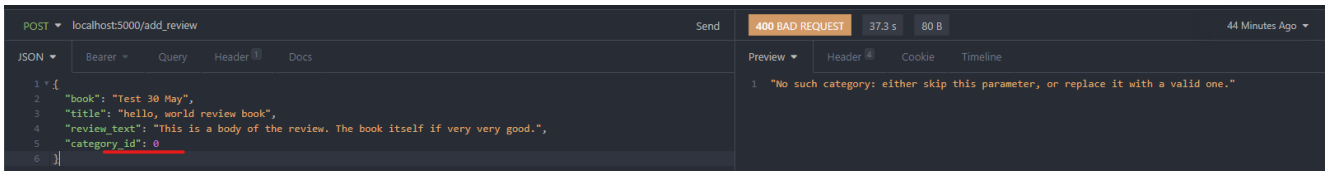

Рисунок 3.18 – Приклад відправлення невірних значень (невірне значенні атрибуту)

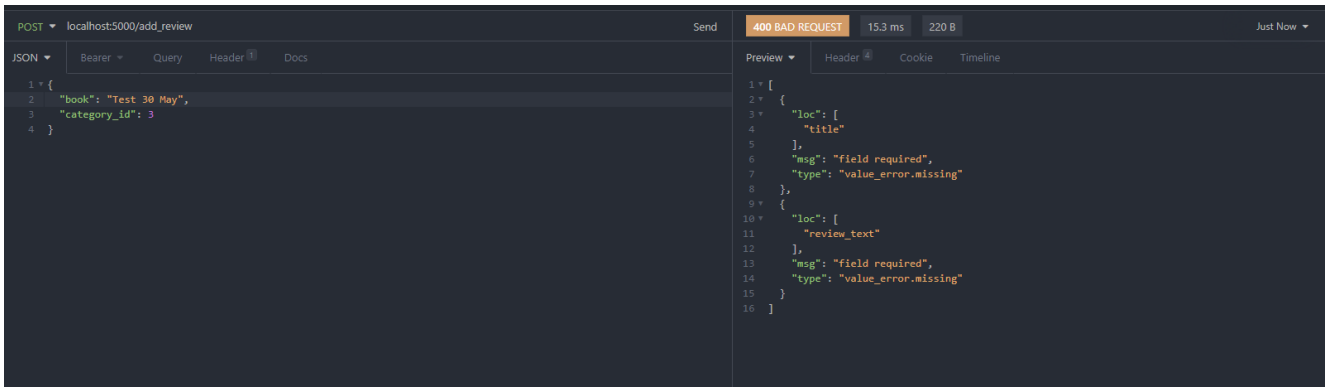

Рисунок 3.19 – Приклад відправлення невірних значень (відсутність обов'язкового поля)

ВИСНОВКИ

Під час виконання випускної роботи було зроблено детальний огляд технологій для розробки REST API для написання відгуків на книги онлайн.

Було проаналізовано декілька сучасних мов програмування та популярних фреймворків, що підходять для створення веб-сервісу У результаті роботи було виконано ознайомлення з предметної областю, а саме функціюванням REST API. Ідентифікація ідеї проекту допомогла сформулювати загальне уявлення про проект.

У ході виконання роботи було сформвано перелік вимог до функціювання REST API та прототипування календарного плану. Було виконано розробку бази даних та програмної частини проекту.

Під час тестування, додаток було наповнено контентом та вхідними даними. Результатом роботи є REST API для роботи з великою кількістю даних та написання відгуків на книги онлайн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 10 Problems That Ecommerce Business Faces and Solutions [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://acquire.io/blogs/problems-solutions-ecommerce-faces (дата звернення: 05.05.2021);

2. How Useful Are These 7 Methods in Helping People Choose? [Електронний ресурс]. – 2020. – Режим доступу до ресурсу:https://zoovu.com/blog/how-useful-are-these-7-methods-in-helping-people-choose/ (дата звернення: 05.05.2021);

3. The Importance of The Information Technology In Business Today [Електронний ресурс]. – 2020. – Режим доступу до ресурсу:https://www.business2community.com/tech-gadget/importance-information-technology-business-today-01393380 (дата звернення: 05.05.2021);

4. Market Research [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.questionpro.com/blog/what-is-market-research (дата звернення: 05.05.2021);

5. I. Elnabarawy, D. C. Wunsch, A. M. Abdelbar, in Proceedings of the International Joint Conference on Neural Networks (Institute of Electrical and Electronics Engineers Inc., 2016), vols. 2016-October, pp. 2986–2991.

6. GoodReads [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.goodreads.com (дата звернення: 05.05.2021);

7. NetGallery [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.netgalley.co.uk (дата звернення: 05.05.2021);

8. BookDepository [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.bookdepository.com (дата звернення: 05.05.2021);

9. Flask vs Django: What's the Difference Between Flask and Django? [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.guru99.com/flask -django.html (дата звернення: 05.05.2021);

10. Python Web Development: Django vs. Flask | The Ultimate Dilemma [Електронний ресурс]. – 2021. – Режим доступу до ресурсу:

https://www.planeks.net/python-web-development-django-vs-flask-the-ultimate-dilemma/ (дата звернення: 05.05.2021);

11. Python for web development [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.imaginarycloud.com/blogs/flask-vs-django/ (дата звернення: 05.05.2021);

12. Python Celery Review [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://blog.iron.io/python-celery-pros-cons-and-review/ (дата звернення: 05.05.2021);

13. When to use RabbitMQ [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.cloudamqp.com/blogs/when-to-use-rabbitmq-or-apache-kafka.html (дата звернення: 05.05.2021);

14. Celery й RabbitMQ [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.devacademy.ru/article/ochered-soobschenij-i-asinhronnyie-zadachi-s-pomoschyu-celery-i-rabbitmq/ (дата звернення: 05.05.2021);

15. Redis Queue (RQ) [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.full-stackpython.com/redis-queue-rq.html (дата звернення: 05.05.2021);

16. Redis [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.sfappworks.com/blogs/redis/ (дата звернення: 05.05.2021);

17. The Python SQL Toolkit and Object Relational Mapper [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.sqlalchemy.org (дата звернення: 05.05.2021);

18. SQLAlchemy [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.full-stackpython.com/sqlalchemy.html (дата звернення: 05.05.2021);

19. Bcrypt Step by Step [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://qvault.io/cryptography/bcrypt-step-by-step/ (дата звернення: 05.05.2021);

20. BCrypt to hash passwords [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://danboterhoven.medium.com/why-you-should-use-bcrypt-to-hash-passwords-af330100b861 (дата звернення: 05.05.2021);

21. Entity Relationship Diagram (ERD) [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.smart-draw.com/entity-relationship-diagram/ (дата звернення: 05.05.2021);

22. H. K. Al-Masree, Extracting Entity Relationship Diagram (ERD) From Relational Database Schem. International Journal of Database Theory and Application. 8, 15–26 (2015).

23. Redis [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://redis.io (дата звернення: 05.05.2021);

24. Celery [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://www.fullstackpython.com/celery.html (дата звернення: 05.05.2021);

25. Introduction to Celery [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.celeryproject.org/en/stable/getting-started/introduction.html#id7 (дата звернення: 05.05.2021);

ДОДАТОК А. КОД РЕАЛІЗАЗІЇ

__init__.py

```
from flask import Flask
from flask_jwt_extended import JWTManager
from flask_bcrypt import Bcrypt
from .models import db
from book_reviewer.background.tasks import mail
from .config import Config
from book_reviewer.background.celery_utils import init_celery

jwt_manager = JWTManager()
bcrypt = Bcrypt()


def create_app(config_class=Config, **kwargs):
    app = Flask(__name__)
    app.config.from_object(config_class)
    if kwargs.get("celery"):
        init_celery(kwargs.get("celery"), app)

    db.init_app(app)
    bcrypt.init_app(app)
    jwt_manager.init_app(app)
    mail.init_app(app)

    from book_reviewer.auth.routes import auth
    from book_reviewer.book_reviews.routes import book_reviews
    from book_reviewer.categories.routes import categories
    from book_reviewer.errors.handlers import errors
    app.register_blueprint(auth)
    app.register_blueprint(book_reviews)
    app.register_blueprint(categories)
    app.register_blueprint(errors)

    return app
```

celery_creator.py

```
from celery import Celery

def make_celery(app_name=__name__):
    backend = "redis://localhost:6379/0"
    broker = backend.replace("0", "1")
    return Celery(app_name, backend=backend, broker=broker)


celery = make_celery()
```

celery_utils.py

```
def init_celery(celery, app):
    celery.conf.update(app.config)
    TaskBase = celery.Task

    class ContextTask(TaskBase):
        def __call__(self, *args, **kwargs):
            with app.app_context():
                return TaskBase.__call__(self, *args, **kwargs)
celery.Task = ContextTask
```

tasks.py

```
from flask_mail import Mail, Message
from .celery_creator import celery

mail = Mail()


@celery.task()
def send_notification_email(users_list: list, review_category: str):
    """Sends an email to a list of users, subscribed to some review category."""
    for user in users_list:
        msg = Message('New review added!', sender='shulga.s1337@gmail.com', recipients=user.split())
        msg.body = f"""Hello, {user},
```

```
New book review was added to {review_category}
"""
    mail.send(msg)
```

dto.py

```
from pydantic import BaseModel, Field
from typing import Optional


class CreateReviewDto(BaseModel):
    book: str = Field(min_length=5, max_length=50)
    title: str = Field(min_length=5, max_length=50)
    review_text: str = Field(max_length=500)
    category_id: Optional[int] = 0
```

book_reviewer/routes.py

```
from flask import Blueprint, request, jsonify
from flask_jwt_extended import jwt_required, get_jwt_identity
from pydantic import ValidationError
from book_reviewer.utils import json_body_required
from .dto import CreateReviewDto
from .service import (add_review, all_reviews_for_book, all_reviews_for_all_books, all_reviews_made_by_user,
                      like_review, unlike_review, sign_up_for_email_notifications,
                      unsubscribe_from_email_notifications, view_my_subscriptions)


book_reviews = Blueprint('book_reviews', __name__)


@book_reviews.route('/add_review', methods=['POST'])
@json_body_required
@jwt_required()
def add_new_review():
    try:
        review_creation_data = CreateReviewDto.parse_obj(request.json)
        result = add_review(review_creation_data, email=get_jwt_identity())
```

```
        return jsonify(result), 201

    except ValidationError as e:
        return e.json(), 400

    except Exception as e:
        return jsonify(str(e)), 400


@book_reviews.route("/reviews/<string:book>", methods=['GET'])
@jwt_required(optional=True)
def view_all_reviews_for_a_book(book):
    page_num = int(request.args.get('page', default=1))
    result = all_reviews_for_book(book_name=book, page_number=page_num)

    return jsonify(result), 200


@book_reviews.route("/reviews/all", methods=['GET'])
@jwt_required(optional=True)
def view_all_reviews_for_all_books():
    page_num: int = int(request.args.get('page', default=1))
    result = all_reviews_for_all_books(page_num)

    return jsonify(result), 200


@book_reviews.route("/<string:user>/reviews", methods=['GET'])
@jwt_required(optional=True)
def view_all_reviews_under_user(user):
    page_num = int(request.args.get('page', default=1))
    result = all_reviews_made_by_user(user, page_num)

    return jsonify(result), 200


@book_reviews.route("/my_reviews", methods=['GET'])
@jwt_required()
def view_reviews_under_current_user():
    page_num = int(request.args.get('page', default=1))
```

```
    result = all_reviews_made_by_user(get_jwt_identity(), page_num)

    return jsonify(result), 200


@book_reviews.route("/reviews/<int:review_id>/like", methods=['POST'])
@jwt_required()
def like_a_review(review_id):
    result = like_review(review_id, get_jwt_identity())
    return jsonify(result), 200


@book_reviews.route("/reviews/<int:review_id>/unlike", methods=['DELETE'])
@jwt_required()
def unlike_a_review(review_id):
    result = unlike_review(review_id, get_jwt_identity())
    return jsonify(result), 200


@book_reviews.route("/sign_up/<int:category_id>", methods=['POST'])
@jwt_required()
def sign_up_for_notifications(category_id):
    return jsonify(sign_up_for_email_notifications(category_id, get_jwt_identity())), 200


@book_reviews.route("/remove_subscription/<int:category_id>", methods=['DELETE'])
@jwt_required()
def remove_email_notifications(category_id):
    return jsonify(unsubscribe_from_email_notifications(category_id, get_jwt_identity())), 200


@book_reviews.route("/view_subscriptions", methods=['GET'])
@jwt_required()
def view_subscriptions():
    return jsonify(view_my_subscriptions(get_jwt_identity())), 200
```

book_reviewer/service.py

```
from ..models import db, User, BookReview, BookReviewSchema, ReviewCategory, Reaction, Subscription
from .dto import CreateReviewDto
```

```
from book_reviewer.background.tasks import send_notification_email

ITEMS_PER_PAGE = 20


def add_review(review: CreateReviewDto, email: str) -> str:
    """Adds a new book review, calls background function in case there are users to be notified"""
    user = User.query.filter_by(email=email).first_or_404()
    existing_categories = {review.id: review.category_name for review in ReviewCategory.query.all()}

    if review.category_id not in existing_categories.keys():
        raise Exception("No such category: either skip this parameter, or replace it with a valid one.")

    book_review = BookReview(
        book=review.book,
        title=review.title,
        review_text=review.review_text,
        category_id=review.category_id,
        user=user.id
    )
    db.session.add(book_review)
    db.session.commit()

    users_to_notify = Subscription.query.filter_by(subscription_category=review.category_id).all()
    user_emails = [user.user_email for user in users_to_notify]

    if user_emails:
        send_notification_email.delay(users_list=user_emails,
review_category=existing_categories[review.category_id])
        return 'Book review was added'

    return 'Book review was added'


def all_reviews_for_book(book_name: str, page_number: int = 1) -> list:
    """Returns all reviews for a given book name, ordered by creation date descending"""
    books = BookReview.query.filter_by(book=book_name).order_by(BookReview.creation_date.desc()).paginate(
        page=page_number,
        per_page=ITEMS_PER_PAGE,
        error_out=False).items
```

```
        books_list = [BookReviewSchema.from_orm(book).dict() for book in books]

        return books_list


    def all_reviews_for_all_books(page_number: int = 1) -> list:
        """Returns all reviews for all books, ordered by creation date descending"""
        books = BookReview.query.order_by(BookReview.creation_date.desc())
        paginated_books = books.paginate(page=page_number, per_page=ITEMS_PER_PAGE, error_out=False)
        serialized_books_list = [BookReviewSchema.from_orm(book).dict() for book in paginated_books.items]

        return serialized_books_list


    def all_reviews_made_by_user(user_email: str, page_number: int = 1) -> list:
        """Returns all reviews made by certain user"""
        user = User.query.filter_by(email=user_email).first_or_404()
        reviews_made_by_user = user.user_reviews.paginate(page=page_number, per_page=ITEMS_PER_PAGE,
error_out=False).items

        serialized_reviews = [BookReviewSchema.from_orm(review).dict() for review in reviews_made_by_user]

        return serialized_reviews


    def like_review(review_id: int, user_email: str) -> str:
        """Marks review as 'liked' for certain user"""
        review = BookReview.query.filter_by(id=review_id).first_or_404()
        user_id = User.query.filter_by(email=user_email).first().id

        reaction_exists = Reaction.query.filter_by(reacted_post=review.id, reacted_user=user_id).first()

        if not reaction_exists:
            reaction = Reaction(reacted_user=user_id, reacted_post=review.id, reaction_type='like', )
            db.session.add(reaction)
            db.session.commit()
            return 'Post was liked.'

        return 'You have already liked this post.'
```

```
    def unlike_review(review_id: int, user_email: str) -> str:
        """Removes 'like' reaction from a review"""
        user_id = User.query.filter_by(email=user_email).first().id
        reaction = Reaction.query.filter_by(reacted_post=review_id, reacted_user=user_id).first()

        if reaction:
            db.session.delete(reaction)
            db.session.commit()
            return 'Your reaction was removed.'

        return 'You have not reacted to this post.'


    def sign_up_for_email_notifications(category_id: int, email: str) -> str:
        """
        Adds user to the list of subscribers under a certain category of reviews,
        so that user gets an email notification whenever review is added"""
        category = ReviewCategory.query.filter_by(id=category_id).first_or_404()
        is_already_subscribed = Subscription.query.filter_by(user_email=email,
subscription_category=category.id).first()

        if not is_already_subscribed:
            subscription = Subscription(user_email=email, subscription_category=category.id)
            db.session.add(subscription)
            db.session.commit()
            return f'{email} has signed up for {category.category_name}.'

        return f'{email} is already signed up for {category.category_name}.'


    def unsubscribe_from_email_notifications(category_id: int, email: str) -> str:
        """Removes users subscription for a given category"""
        subscription = Subscription.query.filter_by(user_email=email, subscription_category=category_id).first()
        if subscription:
            db.session.delete(subscription)
            db.session.commit()
            return f'{email} cancelled subscription for {subscription.subscription_category}.'

        return f'{email} is not subscribed for {category_id}.'
```

```
def view_my_subscriptions(email: str) -> [list, str]:
    """Lists all subscriptions under given user, if there are any"""
    subscriptions = Subscription.query.filter_by(user_email=email).all()
    if subscriptions:
        subscriptions_list = [subscriptions.subscription_category for subscriptions in subscriptions]
        return subscriptions_list

    return f'{email} does not have any subscriptions'
```

categories / routes.py

```
from flask import Blueprint, request, jsonify
from flask_jwt_extended import jwt_required
from .service import all_categories_for_books, all_reviews_under_certain_category


categories = Blueprint('categories', __name__)


@categories.route("/categories", methods=['GET'])
@jwt_required(optional=True)
def view_all_categories_for_books():
    result = all_categories_for_books()
    return jsonify(result), 200


@categories.route("/categories/<int:category_id>", methods=['GET'])
@jwt_required(optional=True)
def view_all_reviews_under_given_category(category_id):
    page_num = int(request.args.get('page', default=1))
    result = all_reviews_under_certain_category(category_id, page_num)

    return jsonify(result), 200
```

categories / service.py

```
from ..models import BookReviewSchema, ReviewCategory, ReviewCategorySchema

ITEMS_PER_PAGE = 20


def all_categories_for_books() -> list:
    """Returns all category names and ids"""
    categories = ReviewCategory.query.all()
    category_names_list = [ReviewCategorySchema.from_orm(category).dict() for category in categories]

    return category_names_list


def all_reviews_under_certain_category(category_id: int, page_number: int = 1) -> list:
    """Returns all book reviews under given category_id"""
    needed_category = ReviewCategory.query.filter_by(id=category_id).first_or_404()
    reviews = needed_category.reviews.paginate(page=page_number, per_page=ITEMS_PER_PAGE).items

    reviews_under_category = [BookReviewSchema.from_orm(review).dict() for review in reviews]

    return reviews_under_category
```

auth/ routes.py

```
from flask import Blueprint, jsonify, request
from flask_jwt_extended import get_jwt_identity, create_access_token, jwt_required
from pydantic import ValidationError
from .dto import UserRegisterDto, UserLoginDto, UserChangePwdDto
from .service import register_user, login_user, change_pwd
from book_reviewer.utils import json_body_required

auth = Blueprint('auth', __name__)


@auth.route('/register', methods=["POST"])
@json_body_required
def register():
```

```
    try:
        user = UserRegisterDto.parse_obj(request.json)
        register_user(user)
        return jsonify(message='Successfully registered.'), 201

    except ValidationError as e:
        return e.json(), 400

    except Exception as e:
        return jsonify(message=str(e)), 400


@auth.route("/login", methods=["POST"])
@json_body_required
def login():
    try:
        user_data = UserLoginDto.parse_obj(request.json)
        tokens = login_user(user_data)
        return jsonify(tokens), 200

    except ValidationError as e:
        return e.json(), 400

    except Exception as e:
        return jsonify(message=str(e)), 400


@auth.route("/refresh", methods=["POST"])
@jwt_required(refresh=True)
def refresh():
    identity = get_jwt_identity()
    access_token = create_access_token(identity=identity)
    return jsonify(access_token=access_token), 200


@auth.route("/change_password", methods=['POST'])
@jwt_required()
@json_body_required
def change_password():
    try:
```

```
        user_data = UserChangePwdDto.parse_obj(request.json)
        change_pwd(user_data, email=get_jwt_identity())
        return jsonify(message='Password was changed'), 200

    except ValidationError as e:
        return e.json(), 400

    except Exception as e:
        return jsonify(str(e)), 400
    return jsonify(result), 200
```

auth/ service.py

```
from typing import Union
from flask_jwt_extended import create_refresh_token, create_access_token
from sqlalchemy.exc import IntegrityError
from .dto import UserRegisterDto, UserLoginDto, UserChangePwdDto
from book_reviewer.models import db, User
from book_reviewer import bcrypt


def register_user(user: UserRegisterDto) -> None:
    hashed_password = bcrypt.generate_password_hash(user.password).decode('utf-8')
    user = User(email=user.email, password=hashed_password,)
    try:
        db.session.add(user)
        db.session.commit()

    except IntegrityError:
        raise Exception('User with this email is already registered.')


def login_user(user: UserLoginDto) -> Union[dict, str]:
    try:
        db_user = User.query.filter_by(email=user.email).first()
        if not db_user:
            raise Exception('No such login.')

        if not bcrypt.check_password_hash(db_user.password, user.password):
```

```
            raise Exception("Error: either login or password is incorrect.")

        response = {
            'access_token': create_access_token(identity=user.email),
            'refresh_token': create_refresh_token(identity=user.email)
        }
        return response

    except Exception as e:
        return str(e)


def change_pwd(user: UserChangePwdDto, email) -> None:
    db_user = User.query.filter_by(email=email).first_or_404()

    if not bcrypt.check_password_hash(db_user.password, user.old_password):
        raise Exception("Error: incorrect old password")

    db_user.password = bcrypt.generate_password_hash(user.new_password).decode('utf-8')
    db.session.commit()
```

errors/ handlers.py

```
from flask import Blueprint, jsonify

errors = Blueprint('errors', __name__)


@errors.app_errorhandler(404)
def error_404(error):
    return jsonify('Could not find what you are looking for'), 404


@errors.app_errorhandler(405)
def error_405(error):
    return jsonify('The method is not allowed for the requested endpoint'), 405


@errors.app_errorhandler(500)
```

```
def error_500(error):
    return jsonify('Server error happened'), 500
```

config.py

```
import os
import datetime


class Config:
    JWT_SECRET_KEY = os.environ.get('DIPLOMA_KEY')
    SQLALCHEMY_DATABASE_URI = 'sqlite:///book_reviewer.db'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    JWT_ACCESS_TOKEN_EXPIRES = datetime.timedelta(hours=1)
    JWT_REFRESH_TOKEN_EXPIRES = datetime.timedelta(days=1)
    MAIL_SERVER = 'smtp.googlemail.com'
    MAIL_PORT = 587
    MAIL_USE_TLS = True
    MAIL_USERNAME = os.environ.get('EMAIL_USER')
    MAIL_PASSWORD = os.environ.get('EMAIL_PASS')


class TestConfig:
    JWT_SECRET_KEY = 'test_secret_key'
    SQLALCHEMY_DATABASE_URI = 'sqlite://'
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    JWT_ACCESS_TOKEN_EXPIRES = datetime.timedelta(minutes=15)
    JWT_REFRESH_TOKEN_EXPIRES = datetime.timedelta(hours=1)
    return jsonify('Server error happened'), 500
```

models.py

```
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
from pydantic import BaseModel, constr

db = SQLAlchemy()
```

```
class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(100), nullable=False, unique=True)
    password = db.Column(db.String(60), nullable=False)
    last_login_time = db.Column(db.DateTime())

    user_reviews = db.relationship('BookReview', backref='users', lazy='dynamic')

    def __repr__(self):
        return f"User('{self.id}','{self.email}', '{self.last_login_time}')"


class BookReview(db.Model):
    __tablename__ = 'book_reviews'
    id = db.Column(db.Integer, primary_key=True)
    book = db.Column(db.String(50), nullable=False)
    title = db.Column(db.String(50), nullable=False)
    review_text = db.Column(db.String(500), nullable=False)
    creation_date = db.Column(db.DateTime, nullable=False, default=datetime.now())

    user = db.Column(db.String(50), db.ForeignKey('users.id'), nullable=False)
    category_id = db.Column(db.Integer, db.ForeignKey('categories.id'), default=1)

    def __repr__(self):
        return f"BookReview('{self.title}', '{self.review_text}','{self.category_id}', '{self.creation_date}')"


class ReviewCategory(db.Model):
    __tablename__ = 'categories'
    id = db.Column(db.Integer, primary_key=True)
    category_name = db.Column(db.String(50), unique=True, nullable=False)

    reviews = db.relationship('BookReview', backref='categories', lazy='dynamic')

    def __repr__(self):
        return f"ReviewCategory('{self.id}', '{self.category_name}')"
```

```
class Subscription(db.Model):
    __tablename__ = 'subscriptions'
    id = db.Column(db.Integer, primary_key=True)
    user_email = db.Column(db.String(50), db.ForeignKey('users.email'), nullable=False)
    subscription_category = db.Column(db.String(50), db.ForeignKey('categories.id'), nullable=False)
    __table_args__ = (db.UniqueConstraint('user_email', 'subscription_category'),)

    def __repr__(self):
        return f"Subscription('{self.user_email}', '{self.subscription_category}')"


class Reaction(db.Model):
    __tablename__ = 'reactions'
    id = db.Column(db.Integer, primary_key=True)
    reacted_user = db.Column(db.String(50), db.ForeignKey('users.id'), nullable=False)
    reacted_post = db.Column(db.String(50), db.ForeignKey('book_reviews.id'), nullable=False)
    reaction_type = db.Column(db.String(10), nullable=False)
    creation_date = db.Column(db.DateTime, nullable=False, default=datetime.now())

    def __repr__(self):
        return f"Reaction('{self.id}', '{self.reacted_user}', '{self.reacted_post}',\
            '{self.reaction_type}' ,'{self.creation_date}')"


class BookReviewSchema(BaseModel):
    id: int
    book: str
    title: str
    review_text: str
    category_id: int
    creation_date: datetime

    class Config:
        orm_mode = True


class ReviewCategorySchema(BaseModel):
    id: int
    category_name: str
```

```
    class Config:
        orm_mode = True
```

utils.py

```
from functools import wraps
from flask import request, jsonify, abort, make_response


def json_body_required(original_function):
    @wraps(original_function)
    def wrapper(*args, **kwargs):
        if not request.json:
            abort(make_response(jsonify(message="Missing request body"), 400))
        return original_function(*args, **kwargs)
    return wrapper
```