📘 Book Store REST API – Project Report

# 1. Introduction

This project is a RESTful API for managing a Book Store, built using Flask, SQLAlchemy, and SQLite. The system provides endpoints to perform CRUD (Create, Read, Update, Delete) operations on books and supports features like search, filtering, and partial matches. It is lightweight, easy to extend, and follows REST principles.

# 2. Objectives

- Provide a backend API to manage books.

- Implement CRUD functionality.

- Allow searching and filtering of books by id, author.

- Handle exceptions and ensure robust error responses.

- Use SQLAlchemy ORM for database operations.

# 3. Technologies Used

- Backend Framework: Flask

- Database: SQLite (via SQLAlchemy ORM)

- Language: Python

- Libraries:

- flask – web framework

- flask\_sqlalchemy – ORM

- flasgger – API documentation (Swagger UI)

- sqlalchemy.exc – error handling

# 4. System Design

Database Model: Book

Each book record contains:

- id (Primary Key, Integer, Auto Increment)

- title (String, Required)

- author (String, Required)

- publisher (String)

- edition (String)

- language (String)

- pages (Integer)

- genre (String)

- price (Float)

- rating (Float, Default = 0.0)

- stock\_status (String: 'In Stock'/'Out of Stock')

# 5. Features

📌 CRUD Operations

1. Create – Add a new book (POST /books).

2. Read – Get all books or filter/search (GET /books).

3. Read by ID – Get a single book by ID (GET /books/id/<id>).

4. Read by Author – Get all books by (partial) author name (GET /books/author/<author>).

5. Update – Modify book details (PUT /books/id/<id>).

6. Delete – Remove a book (DELETE /books/id/<id>).

📌 Search and Filtering

- Search by id, author.

- Case-insensitive partial matches supported.

📌 Error Handling

- Returns structured JSON error messages.

- Handles SQLAlchemy errors, Integrity errors, and invalid inputs.

# 6. Example API Usage

Create a Book (POST /books)

{

"title": "Clean Code",

"author": "Robert C. Martin",

"publisher": "Prentice Hall",

"edition": "1st",

"language": "English",

"pages": 464,

"genre": "Programming",

"price": 30.5,

"rating": 4.8,

"stock\_status": "In Stock"

}

Get Books by Author (GET /books/author/Martin)

[

{

"id": 1,

"title": "Clean Code",

"author": "Robert C. Martin",

"price": 30.5,

"genre": "Programming",

"stock\_status": "In Stock"

}

]

# 7. Advantages

- Simple and lightweight API.

- Supports dynamic filtering and searching.

- Extensible for future features like user authentication, reviews, or order management.

- Well-structured error handling.

# 8. Limitations & Future Enhancements

- No authentication/authorization.

- No pagination for large datasets.

- Limited to SQLite (can be extended to MySQL/PostgreSQL).

- No frontend interface (only backend API).

- Could integrate with Swagger UI for API documentation.

# 9. Conclusion

This project demonstrates a practical implementation of a Book Store REST API using Flask and SQLAlchemy. It successfully performs CRUD operations, supports search & filtering, and ensures structured error handling. With further improvements like authentication, pagination, and frontend integration, it can evolve into a complete online book store system.