# **CPSC 449 Web Backend Engineering**

**Assignment 1**

**Name: Sai Sirisha Surapaneni**

**Email:** [**siri23@csu.fullerton.edu**](mailto:siri23@csu.fullerton.edu)

1. **Introduction**

The Flask Library Management System is a web application designed to streamline the management of books within a library. The system provides a RESTful API interface for interacting with the database, allowing users to perform CRUD operations on book records.

1. **Technologies Used**

The Flask Library Management System is built using the following technologies:

* Flask: A lightweight and extensible web framework for Python, providing tools and utilities for building web applications.
* MySQL: A relational database management system used for storing and managing book records and related data.
* Python: A versatile programming language used for backend development, data manipulation, and business logic implementation.
* HTML/CSS: Frontend technologies used for designing the user interface and styling the s
* Postman/HTTPie: Tools used for testing API endpoints and verifying their functionality during the development and testing phases.
* Swagger: A tool used for documenting and testing APIs. Swagger provides a user-friendly interface for exploring and interacting with the API endpoints.

1. **Configuration**

* Database Connection: Update the database configuration parameters (host, username, password, database name) in the Flask application code (app.py).

A screen shot of a computer

Description automatically generated

* Environment Variables: Use environment variables to store sensitive information, such as database credentials.

1. **Database Schema**

* Tables: The database schema includes a books table with columns for id, title, author, genre, published\_year, and availability.

A screenshot of a computer program

Description automatically generated

* Entries: The books table contains records with suitable data types for id, title, author, genre, published\_year, and availability columns.



* Relationships: No explicit relationships are defined in the schema, as the books table is standalone

1. **API Requests/Responses**
2. **GET** /books: Retrieve all books from the database.

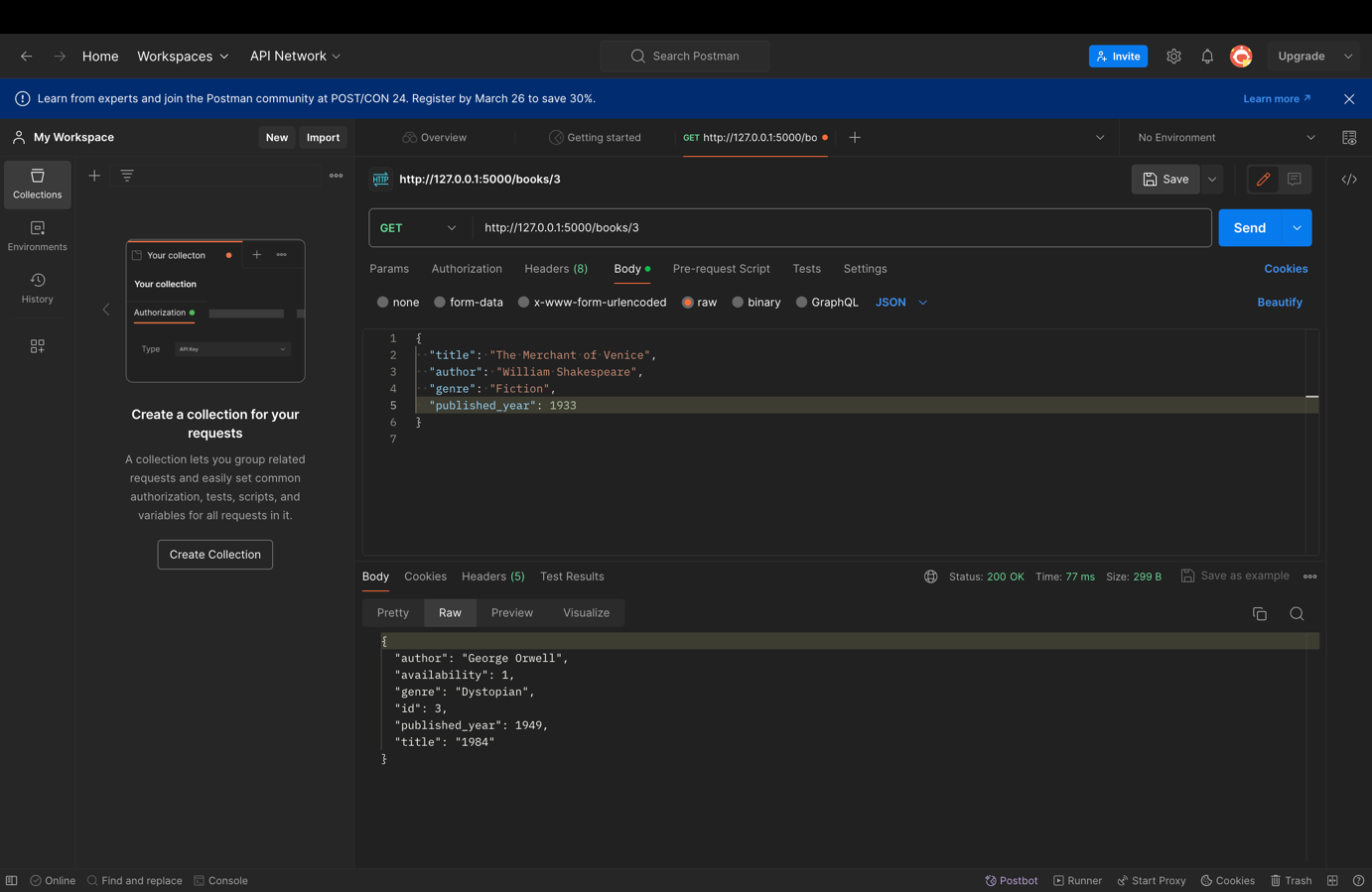
Send a GET request to http://127.0.0.1:5000/books using your preferred tool (Postman, HTTPie, etc.). You should receive a JSON response containing all books in the database.

A screenshot of a computer

Description automatically generated

1. **GET** /books/{id}: Retrieve a specific book by ID.

Send a GET request to http://127.0.0.1:5000/books/{id} (replace {id} with the ID of an existing book) to retrieve details of a specific book. Ensure you replace {id} with an actual ID from the database.



**Figure 1 GET/success.**

A screenshot of a computer

Description automatically generated

**Figure 2 GET/error**

1. **POST** /books: Add a new book to the database.

Send a POST request to http://127.0.0.1:5000/books with JSON data containing details of a new book (title, author, genre, published\_year). Verify that the book is added successfully by checking the response message.

A screenshot of a computer

Description automatically generated

1. **PUT** /books/{id}: Update an existing book by ID.

Send a PUT request to http://127.0.0.1:5000/books/{id} (replace {id} with the ID of an existing book) with JSON data containing updated details of the book (title, author, genre, published\_year). Verify that the book is updated successfully by checking the response message.

A screenshot of a computer

Description automatically generated

**Figure 3 PUT/success**

A screenshot of a computer

Description automatically generated

**Figure 4 PUT/error**

1. **DELETE** /books/{id}: Delete a book from the database by ID.

Send a DELETE request to http://127.0.0.1:5000/books/{id} (replace {id} with the ID of an existing book) to delete the book from the database. Verify that the book is deleted successfully by checking the response status code and message.

A screenshot of a computer

Description automatically generated

**Figure 5 DELETE/success**

A screenshot of a computer

Description automatically generated

**Figure 6 DELETE/error**

Access the application in your web browser at [http://127.0.0.1:5000](http://127.0.0.1:5000/).

A screenshot of a computer

Description automatically generated

**Figure 7 Application WebPage**

Additionally, one can add books using the Add\_Book button.

A computer screen with a white background

Description automatically generated

1. **SWAGGER INTEGRATION**

The Flask Library Management System API application integrates Swagger for enhanced API documentation and testing. Swagger offers a user-friendly interface to explore and interact with endpoints, simplifying understanding of available endpoints, parameters, and responses for developers.

A screenshot of a computer

Description automatically generated

**Figure 8 Swagger Documentation**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **GITHUB Repo**

Repo Link: <https://github.com/sirishaa03/Library-Management-System>

This repository hosts the documentation for the Flask Library Management System, an application built to manage book records efficiently. The documentation provides scomprehensive guidance on setting up, configuring, and utilizing the system effectively.