# Book Inventory Management System

**Author:** Sai Vikas Goli

**Project Overview:**

The Book Inventory System is a web-based application designed to manage and track book inventory for libraries, bookstores, or personal collections. This system provides users with the ability to easily add, filter, export book details, including titles, authors, genres, publication years, page counts, ISBNs, prices, and available quantities. The application aims to streamline the management of book data, making it more efficient and accessible.

**Objectives**

* **User-Friendly Interface**: Create a simple and intuitive front-end interface that allows users to interact with the system seamlessly.
* **Robust Database Management**: Utilize a MySQL database to store and retrieve book information, ensuring data integrity and reliability.
* **Dynamic Backend**: Develop a Node.js backend to handle API requests, manage database connections, and process business logic for book operations.
* **CRUD Functionality**: Implement full Create, Read, Update, and Delete (CRUD) functionality for managing books within the inventory.
* **Scalability**: Design the system with scalability in mind, allowing for future enhancements such as user authentication, search capabilities, and advanced reporting features.

**Technologies Used**

* **Frontend**: HTML, CSS , and JavaScript for building a responsive user interface.
* **Backend**: Node.js with Express framework to create a RESTful API.
* **Database**: MySQL for structured data storage and management. (MySQL Workbench)
* **Dependencies**: express, mysql2, and body-parser for handling requests and managing database connections. Bootstrap and Font-Awesome icons for enhanced CSS
* **IDE**: Microsoft Visual Studio Code.

**Step 1: Database Schema Design**

Created a Database called **book\_inventory** and created a **books** table to store book details:

Here’s the Relational database Schema.

CREATE DATABASE book\_inventory;

CREATE TABLE books (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

author VARCHAR(255) NOT NULL,

genre VARCHAR(100),

publicationYear INT,

pages INT,

isbn VARCHAR(13),

price DECIMAL(10, 2),

quantity INT DEFAULT 1

);

**Inserted Some raw data into books table**

INSERT INTO books (title, author, genre, publicationYear, pages, isbn, price, quantity)

VALUES

('To Kill a Mockingbird', 'Harper Lee', 'Classic', 1960, 281, '9780061120084', 7.99, 12),

('1984', 'George Orwell', 'Dystopian', 1949, 328, '9780451524935', 9.99, 8),

('Pride and Prejudice', 'Jane Austen', 'Romance', 1813, 279, '9781503290563', 6.99, 15),

('The Catcher in the Rye', 'J.D. Salinger', 'Classic', 1951, 214, '9780316769488', 8.99, 10),

('The Hobbit', 'J.R.R. Tolkien', 'Fantasy', 1937, 310, '9780547928227', 10.99, 5);

**Step 2: Backend Development with Node.js and MySQL**

**The backend of the Book Inventory System is built using Node.js and Express. It provides a RESTful API that allows the frontend to interact with the MySQL database to manage book inventory.**

1. **Setup for Node.js Project**:

mkdir book-inventory

cd book-inventory

npm init -y

npm install express mysql2 body-parser

1. **Created app.js file:**

The main entry point of the application that sets up the server and middleware.

1. **Created a Model Folder and added book.js file:**

Handles database connection and book-related queries using MySQL.

1. **Created a Controller Folder and added bookController.js file:**

Contains the business logic for handling book-related operations such as:

* addBooks()
* getBooks()
* exportBooks()

1. **Created a Routes folder and added route,js file:**

Defines the API endpoints for book operations.

**Step 3: Frontend Development with HTML, CSS and JavaScript**

1. **Created a Views Folder and added home.html file:**

The home page that serves as the main entry point for the application.

1. **Added addBook.html file in the views folder:**

The page contains a form for adding a new book to the inventory and filter function to filter books based on Genre and Author.

1. **Added a style.css file in the views folder:**

The stylesheet for styling the web pages.

1. **Added a script.js file in the views folder:**

JavaScript for handling form submissions and making API calls.

**Design Decisions**

1. **Modular Code Structure**: The application is organized into controllers, models, and routes, following the MVC pattern. This enhances code readability and maintainability by separating concerns.
2. **Asynchronous Programming**: Utilizing asynchronous programming in API calls on the frontend improves user experience by allowing the UI to remain responsive while waiting for server responses.

**Challenges Faced**

1. **CORS Issues**: Encountered Cross-Origin Resource Sharing (CORS) issues when making API calls from the frontend. This was resolved by configuring appropriate headers on the server.
2. **Data Validation**: Ensuring proper data validation on both the frontend and backend was crucial to prevent incorrect data entries. Implementing validation rules required careful consideration of the types of data users might input.

**Conclusion**

This documentation outlines the structure and functionalities of both the backend and frontend of the Book Inventory System and Design decisions and challenges faced. The backend provides a robust API for managing book data, while the frontend offers an intuitive interface for users to interact with the system.