Software Design Specification

Section 1: Introduction

1.1 Describe the purpose of this document

This document serves as the Software Design Specification for the Book Exchange System. The purpose of this document is to provide a detailed description of the system’s overall design, architecture and functionality. The purpose is to make sure that all stakeholders, developers, testers, and managers have a clear understanding of the design approach, the components that make up the system and the interactions between each.

1.2 Describe the scope of this document

* System Architecture Overview
* Design Considerations
* Architectural Strategies
* System Design
* Graphical User Interface Design

1.3 Describe this document's intended audience

* Project Managers: Helps monitor progress and ensure that the system’s design meets the project requirements.
* Software Developers: To help them understand the overall system design and implementation details.
* System Architects: Helps validate the architecture of the system and make sure it aligns with the project’s goals and scalability requirements.
* End-users: Informs the design decisions that have been made for the user interface that will impact the overall user experience.

1.4 Identify the system/product using any applicable names and/or version numbers.

* System Name: Book Management System
* Version: 1.0
* Development Framework: Django 5.1
* Database: PostgreSQL
* Frontend Technologies: HTML
* Authentication: Django’s built-in authentication system

1.5 Provide references for any other pertinent documents such as:

• Related and/or companion documents

* Software Requirements Specification: Describes the overall description of the system, the interfaces, and all requirements.

• Prerequisite documents

* Software Requirements Specification: Describes the overall description of the system, the interfaces, and all requirements.

• Documents which provide background and/or context for this document

* Software Requirements Specification: Describes the overall description of the system, the interfaces, and all requirements.

• Documents that result from this document (e.g. a test plan or a development plan)

* Software Requirements Specification: Describes the overall description of the system, the interfaces, and all requirements.

1.6 Define any important terms, acronyms, or abbreviations

* Django: Python based web framework that runs on a web server. Follows the model-template-views architectural pattern.
* UI: Stands for user interface. It’s the point of contact between the user and the computer. This represents the front-end of the application.
* HTTP: Stands for Hypertext Transfer Protocol. It's used to transfer data over the web.
* API: Stands for Application Programming Interface. It’s the set of protocols for building and interacting the software.
* MVC: Stands for Model-View-Controller. This is the software design pattern used in developing the user interfaces.
* ORM: Stands for object relational mapping. It’s used by django to interact with the databases in an object-oriented way.

Section 2: System Overview

The Book Exchange System is an online tool made for handling a collection of books, user activities and shopping cart options. It lets users look through books, add or remove them from their online shopping cart and favorites list. The system has user login, so registered users can manage their books, add new books and write comments. The main objective is to offer an easy, user-friendly design that helps people find and organize books.

The system has the following features:

* User Authentication: Users can sign up, log in and manage their accounts.
* Book Management: Users can add new books into the system, see details about the books, and remove their books.
* Search Functionality: Users can look for books by title or author using a simple search tool.
* Comments: Users can leave comments on books.
* Shopping Cart: Users can add books to their shopping cart, take them out, and see what’s in their cart.
* Favorites: Users can mark books as favorites to easily find them later.
* Responsive UI: The web application works well on phones and other devices making it easy to use everywhere.x

The system will use a PostgreSQL database to keep track of user information, books and comments.

System Structure:

The system uses the Model-View-Controller structure which means:

* Model: This part deals with the data, like information about books, users and comments.
* View: This handles how the information is shown to the user using HTML templates.
* Controller: These are the Django views that manage web requests and work with the model.

Main Parts:

Frontend:

* The user interface is made with HTML and styled using Bootstrap.
* The frontend communicates with the backend using HTTP requests via Django views to render data and handle interactions like adding/removing books to/from the cart.

Backend:

* Built on the Django web framework, it handles routing, model definitions and the core application logic.
* Also manages authentication, authorization and session management.

Database:

* A PostgreSQL database that stores data about users, books, comments and session information.

Security:

* The system uses Django’s built-in tools for user registration, login and managing user sessions. It also uses strong methods to securely store and protect user passwords.

Scalability:

* The system is built to handle more users as they come. If the number of users increases, the application can be expanded by adding more databases.

System Workflow Example:

1. A user either logs in or signs up through the authentication page.
2. The user can then look at books, see more details and add a book to their shopping cart.
3. The user can review their cart.
4. The user can also save books to their favorites list for future use.

The system is designed to be simple for users and easy for developers to manage.

Section 3: Design Considerations

3.1 Assumptions and Dependencies

* Assumptions:
  + Users have access to Wi-Fi and a modern web browser on a suitable device to interact with the system.
  + The system assumes that users will input valid formats of data.
  + The “My Books” feature is only accessible to registered users who have logged into the system.
  + No payment processing system has been integrated at this stage.
* Dependencies:
  + Python 3.12+ and Django 5.1 for backend development
  + PostgreSQL database to store user information and book data.
  + Django administration for easy access to website data.
  + Django authentication for secure management of user accounts and sessions in the system.
  + Bootstrap 5.3 for responsive and visually appealing frontend development.

3.2 General Constraints

* The website must effectively handle multiple users simultaneously and support a minimum of 10,000 book entries.
* Images uploaded by users must comply with size and format constraints to ensure performance.

3.3 Goals and Guidelines

* Provide a new and engaging, easy-to-use interface for book browsing, uploading, and interacting.
* Ensure secure handling of user login data along with session and book management.
* Use scalable architecture to accommodate future updates such as payment processing or a rating system.

3.4 Development Methods

* Implement an agile development approach for both iterative design and testing.
* Employ Django’s built in database system for easy data management.

Section 4: ArchitecturaI Strategies

4.1 Use of a particular type of product (programming language, database, library, etc.)

* Python 3.12 and Django 5.1 for backend, server-side development.
* HTML templates utilizing Bootstrap 5.3 for modern and responsive UI design.
* Django administration’s built-in database system to manage data.
* Django authentication to securely manage user login credentials and sessions.

4.2 Reuse of existing software components to implement various parts/features of the system

* Django’s built-in administration to manage backend development.
* Django’s authentication system to manage user login/logout.
* Bootstrap for frontend design using tools such as buttons, forms, and navbar.

4.3 Future plans for extending or enhancing the software

* Integrating a gateway for payments to enable book purchases in the shopping cart.
* Implement different options for checkout as a guest user or registered user.
* Implementing a ratings/review system for a book.
* Implementing a method to set the favorites list to private or public.

4.4 User interface paradigms (or system input and output models)

* Ensure responsive design usability across different devices such as desktops, tablets, and mobile devices.
* Consistent use of Bootstrap components to maintain UI design.
* Intuitive navigation of website for ease-of-use of the features offered.

4.5 Hardware and/or software interface paradigms

* Host website on a cloud service for scalability, cost effectiveness, and high availability.
* Web browsers for user access.
* Django server for backend processing.

4.6 Error detection and recovery

* Custom error pages for various error codes like 404 or 500 errors.
* Utilize Django’s built-in error handling system.
* Log errors for debugging and future reference.
* Form validation when uploading book details.

4.7 Memory management policies

* Optimize Django’s database queries to efficiently handle large datasets.
* Implement caching for frequently accessed data.

4.8 External databases and/or data storage management and persistence

* Use Django database models for managing data like book details, user accounts, favorites, comments, “My Books”, and shopping cart items.
* Store image files in a secure location while allowing access control for them.
* Ensure reliable data persistence through consistent backups.

4.9 Distributed data or control over a network

* Not applicable in current scope.

4.10 Generalized approaches to control

* Implement role-based access control for different users (e.g. Registered users can view and delete their own books).
* Administrative access for backend management.

4.11 Concurrency and synchronization

* Django’s request handling already supports concurrent user actions.
* Synchronization handled by Django administration to prevent issues like duplicate records.

4.12 Communication mechanisms

* HTTP/HTTPS for secure client-server communication across the website.

Section 5: System Architecture

Frontend:

* 5.1 Component-1: Book Display and Search Interface
  + Displays a list of all books with publisher and options to add to shopping cart or “Favorites” list.
  + Includes a search bar on the top-left corner to locate books by their name or by publisher.
* 5.2 Component-2: Book Details Page
  + Displays detailed information about selected book when clicked by user, including book name, price, publisher, book cover, and comments.
  + Allows for user interaction through comments section.
* 5.3 Component-3: My Books Interface
  + Only accessible by registered users who have logged into the website.
  + Displays list of books the user has uploaded.
  + Includes the option to delete their own books.
* 5.4 Component-4: Favorites Page
  + Displays a list of books marked as favorites by the user.
  + Users may also unfavorite a book from their list.
* 5.5 Component-5: Shopping Cart Interface
  + Allows users to add books to their shopping cart.
  + Displays list of books users want to purchase.
  + Payment gateway not implemented at current scope.
* 5.6 Component-6: User Authentication Pages
  + Allows users with a registered account to login or logout.

Backend:

* 5.7 Component-1: User Authentication Subsystem
  + Handles user login/logout, registration, and session management through Django’s authentication system.
  + Ensures that only registered users who have logged in may access the “My Books” tab.
* 5.8 Component-2: Book Management Subsystem
  + Implements CRUD operations to manage books through Django’s administration system.
  + Allows users to upload and delete their books.
* 5.9 Component-3: Comment Management Subsystem
  + Enhance user experience by allowing users to add and view comments on books.
* 5.10 Component-4: Favorites Management Subsystem
  + Allows users to mark books as favorites to review at a later time.
* 5.11 Component-5: Shopping Cart Subsystem
  + Manages cart content for each user without the inclusion of payment processing.
* 5.12 Component-6: Search Functionality Subsystem
  + Searches through Django’s database using query filters for the specified book.
* 5.13 Component-7: Media Management Subsystem
  + Handles user uploads of book cover images and manages storage systems for uploaded images.
    - Validates specified image format and size.
  + 5.14 Component-8: Admin Panel
    - Provides administrative access and tools to manage website data such as users, books, and comments.

Section 6: Detailed System Design

### Web Application Component

#### Definition

The web application serves as the interface for users to interact with the system. It provides a visual platform (front end) and the logic to handle user requests and return appropriate responses. It is designed to meet functional requirements such as data input, visualization, and interaction and non-functional requirements like scalability and responsiveness.

#### Responsibilities

* **Frontend**:
  + Render dynamic user interfaces based on data received from the backend.
  + Handle user input, validate form data, and provide feedback to the user.
  + Ensure responsiveness for different devices (desktop, mobile, tablet).
* **Backend**:
  + Process HTTP requests and execute business logic.
  + Interact with databases to retrieve or update data.
  + Provide secure APIs for data exchange with other systems or services.

#### Constraints

* **Timing**:
  + Must respond to user requests within 2 seconds to meet usability standards.
* **Storage**:
  + Caching is required to improve performance, but caching is limited by backend database capacity.
* **Component State**:
  + A Stateless HTTP protocol is used; sessions are maintained using cookies or tokens.
* **Preconditions**:
  + Valid authentication is required to access protected resources.
* **Postconditions**:
  + Data changes a user makes are reflected in the UI in real-time.

#### Composition

* **Frontend Subcomponents**:
  + User Interface Templates (HTML/CSS/JavaScript).
  + React.js or Angular for client-side rendering and interactivity.
* **Backend Subcomponents**:
  + Django or Flask is used to handle HTTP requests and manage routes.
  + RESTful API endpoints for client-server communication.
  + Business Logic Layer is used to process inputs and execute workflows.

#### Uses/Interactions

* **Interactions with Other Components**:
  + The **Database** for storing and retrieving application data.
  + The **Authentication Service** to validate user credentials and manage sessions.
  + The **external API is used** for integrations such as payment gateways or third-party analytics.
* **Used By**:
  + End-users for interacting with the system.
  + Administrators are responsible for managing content or monitoring system health.

#### Resources

* **Managed Resources**:
  + Database connections.
  + Cache memory reduces repetitive database queries (e.g., Redis).
* **External Dependencies**:
  + Cloud hosting for deployment (e.g., AWS or Azure).
  + Frontend libraries for UI enhancements (e.g., Bootstrap, Chart.js).
* **Race Conditions/Deadlocks**:
  + Avoided by using database transaction locks and concurrency-safe programming practices.

#### Processing

* **Frontend**:
  + Load the initial HTML/JavaScript bundle from the server.
  + Use asynchronous requests (AJAX/Fetch) for dynamic updates without reloading the page.
* **Backend**:
  + Receive HTTP requests, route them based on URLs, and invoke the corresponding business logic.
  + Return responses in JSON format for API calls or render HTML templates for direct requests.
* **Concurrency**:
  + Use thread pools or asynchronous processing to handle multiple requests simultaneously.

#### Interface/Exports

* **Frontend Services**:
  + User-friendly forms for data input.
  + Charts and graphs for data visualization.
* **Backend Services**:
  + API endpoints:
    - GET /data: Fetches user-specific data.
    - POST /update: Updates user preferences.
  + Authentication middleware: Validates tokens before granting access to resources.
  + Logging and error handling: Captures critical errors and sends alerts.

**Admin Interface on Django**

• **Definition**

Django's admin interface is a built-in web-based application tool that allows developers and administrators to manage application data. It provides a dynamic dashboard for performing operations on database models registered within the application.

• **Responsibilities**

Allow administrators to directly handle application data by utilizing CRUD (create, read, update, delete) actions.

* Model customization: Offer a way to modify the admin panel’s model management and display interface.
* To guarantee that only authorized users can access the interface, enforce administrative permissions.
* Logging and reporting: Create reports on administrative actions and keep track of modifications performed via the interface.

• **Constraints**

* Authentication: Only authorized users with administrator rights are able to access the admin interface.
* Performance: Even with the big datasets, the interface depends on effective database queries to function smoothly.
* Extensibility: Django’s admin framework limits customization, although it allows for the extension of functionality through model admins and custom templates.

• **Composition**

Each model is represented by a ModelAdmin class in the admin interface, which regulates the management and display of data.

* Templates: Offer the interface’s HTML structure, which can be altered to satisfy certain design specifications.
* Coordinating registered models and their customizations, the admin site serves as the interface’s focal point.
* Forms: Offer the means of entering and validating data

• **Uses/Interactions**

* Database: Retrieves, displays, and modifies model data by interacting directly with the database.
* Authentication System: Uses Django’s authentication system to control permissions and admin user access.
* Middleware: Makes use of middleware for request processing, CSRF prevention, and session management.
* Other applications: For extra functionality like reporting, auditing, or data exporting, it might communicate with third party- programs

• **Resources**

The admins interface’s main resource for retrieving and editing data from the application’s database is the database.

* Memory: used to hold temporary information while processing requests and generating forms.
* The user interface must be rendered using templates and static files
* Concurrent Access:Preserves data integrity by utilizing database-level locking to manage possible race conditions.

• **Processing**

Using the “admin.site.register()” function to register models with the admin interface.

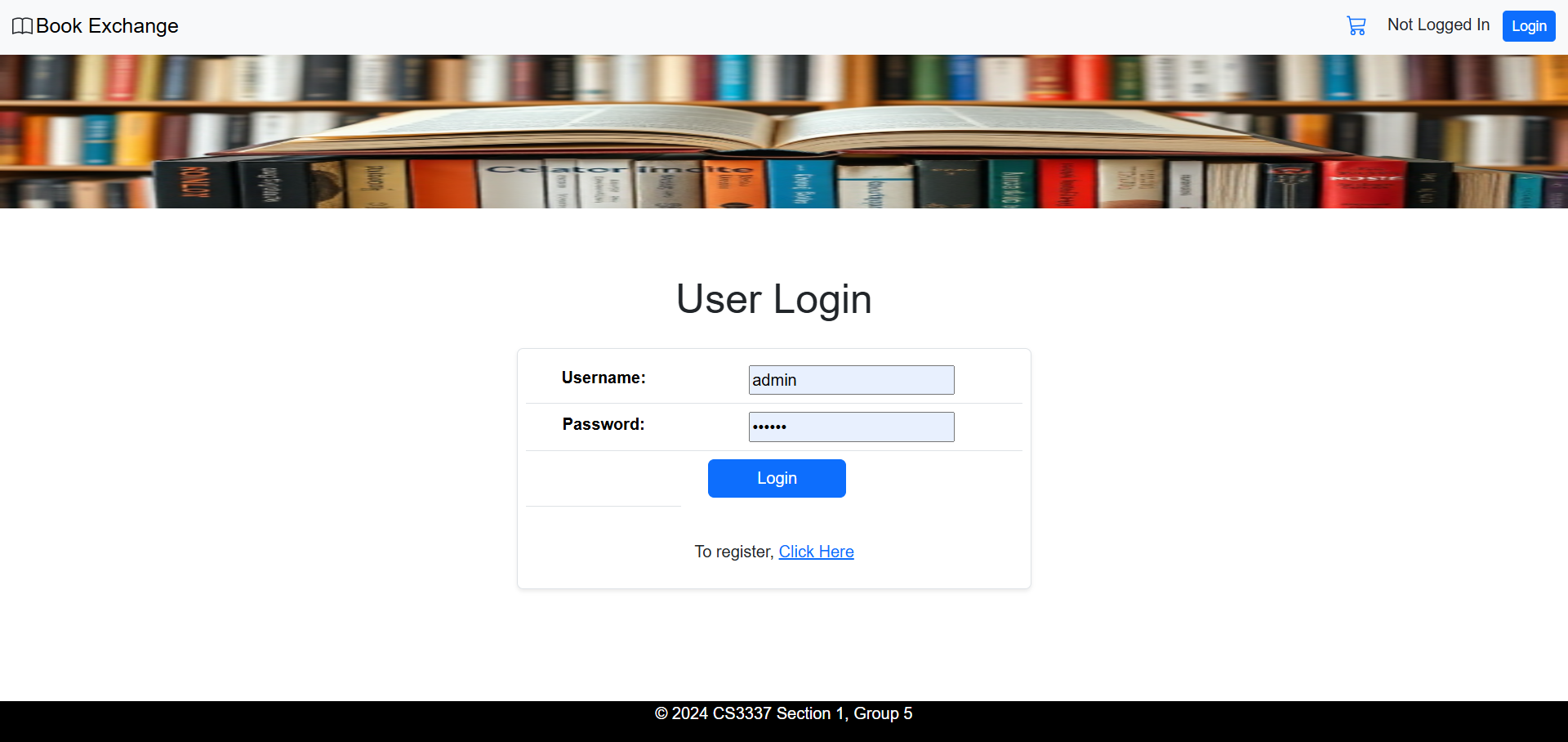
* Request Handling: Handles HTTP requests to render forms, get data, and carry out operations like record saving and deletion.
* The process of retrieving and formatting data for tabular or form views is known as “data rendering”
* Validation: Before making changes to the database, input data is verified using form validators and the Django model.
* Logging: Keeps track of administrator modification for auditing and troubleshooting

• **Interface/Exports**

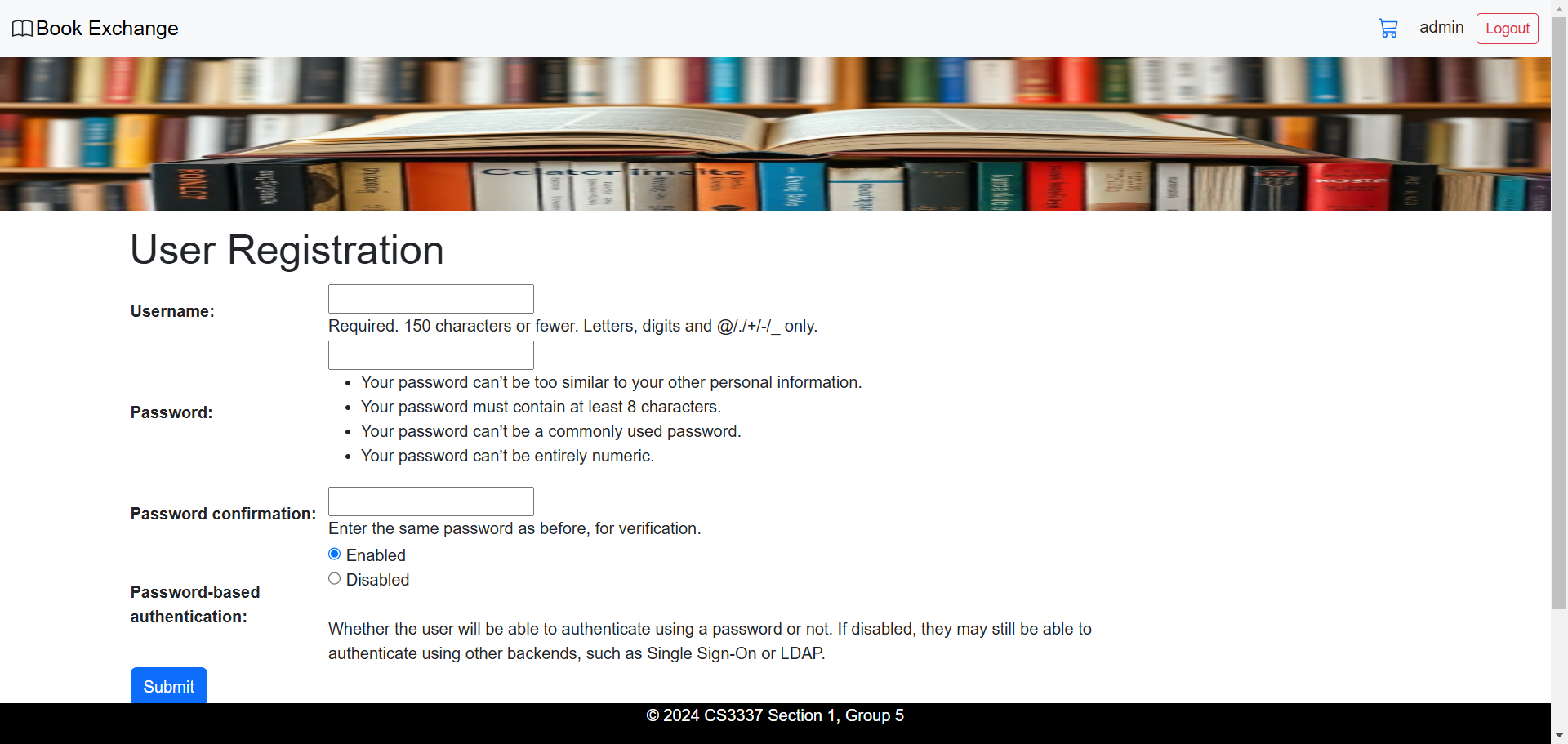
1. **Model Management** 
   1. Automatically generates interfaces for managing registered models
   2. Exported Components
      1. Add/Edit forms: based on model fields
      2. List views: display tabular data
      3. Detail Views: displays detailed information
2. **Authentication and Permissions** 
   1. Built-in authentication for secure access
   2. Permissions
      1. Assign model-level permissions: Can add, Can change
      2. Role-Based access control for different admin users
3. **Logging and Monitoring** 
   1. Tracks admin activity (e.g., login/logout events, changes made to models).
   2. Logs are accessible for auditing purposes.

Section 7: GraphicaI User Interface Design

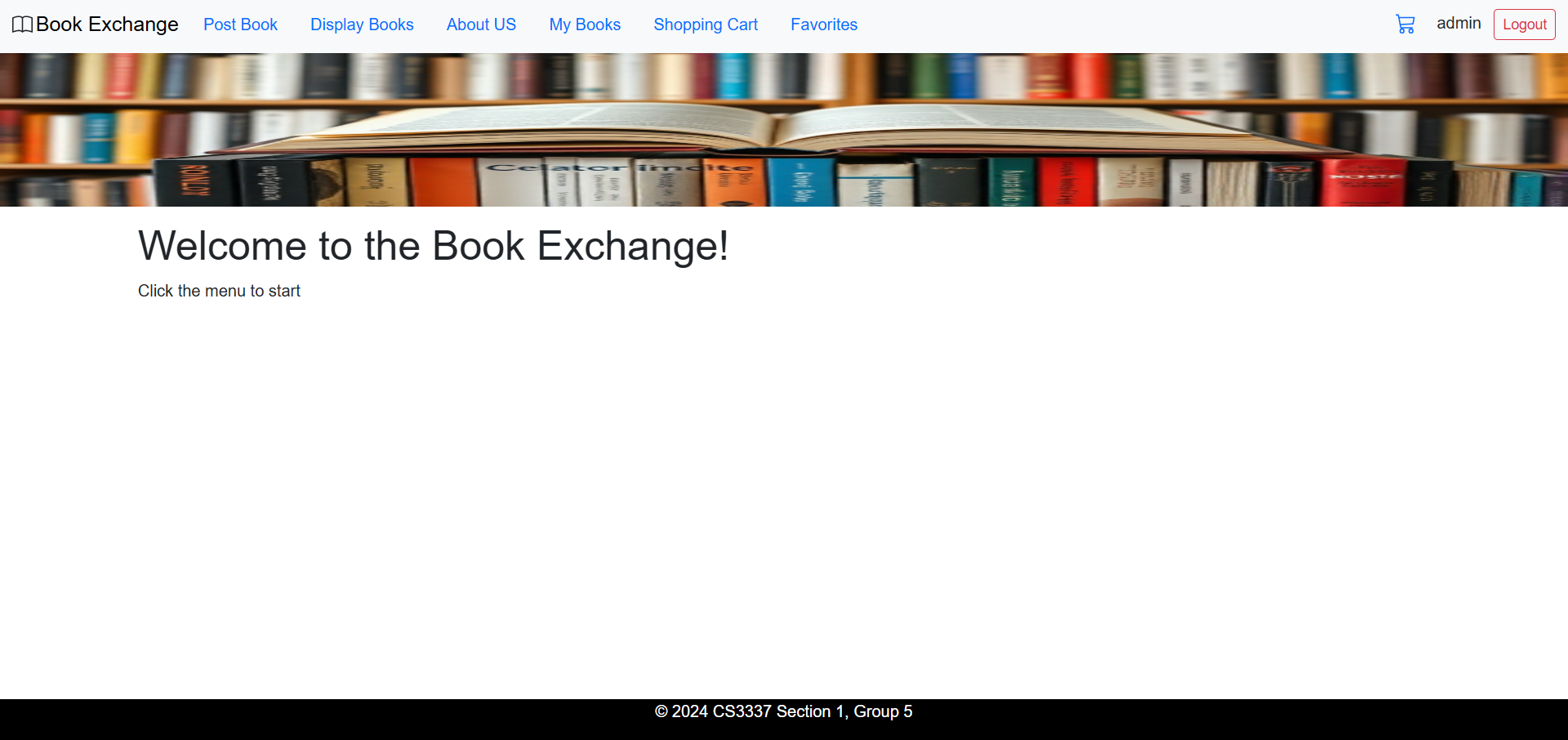
* **User Login Portal**
  + Login screen will have the user sign in with their username and password. There is also a link to register for first time users. The shopping cart is also made available for unregistered users.



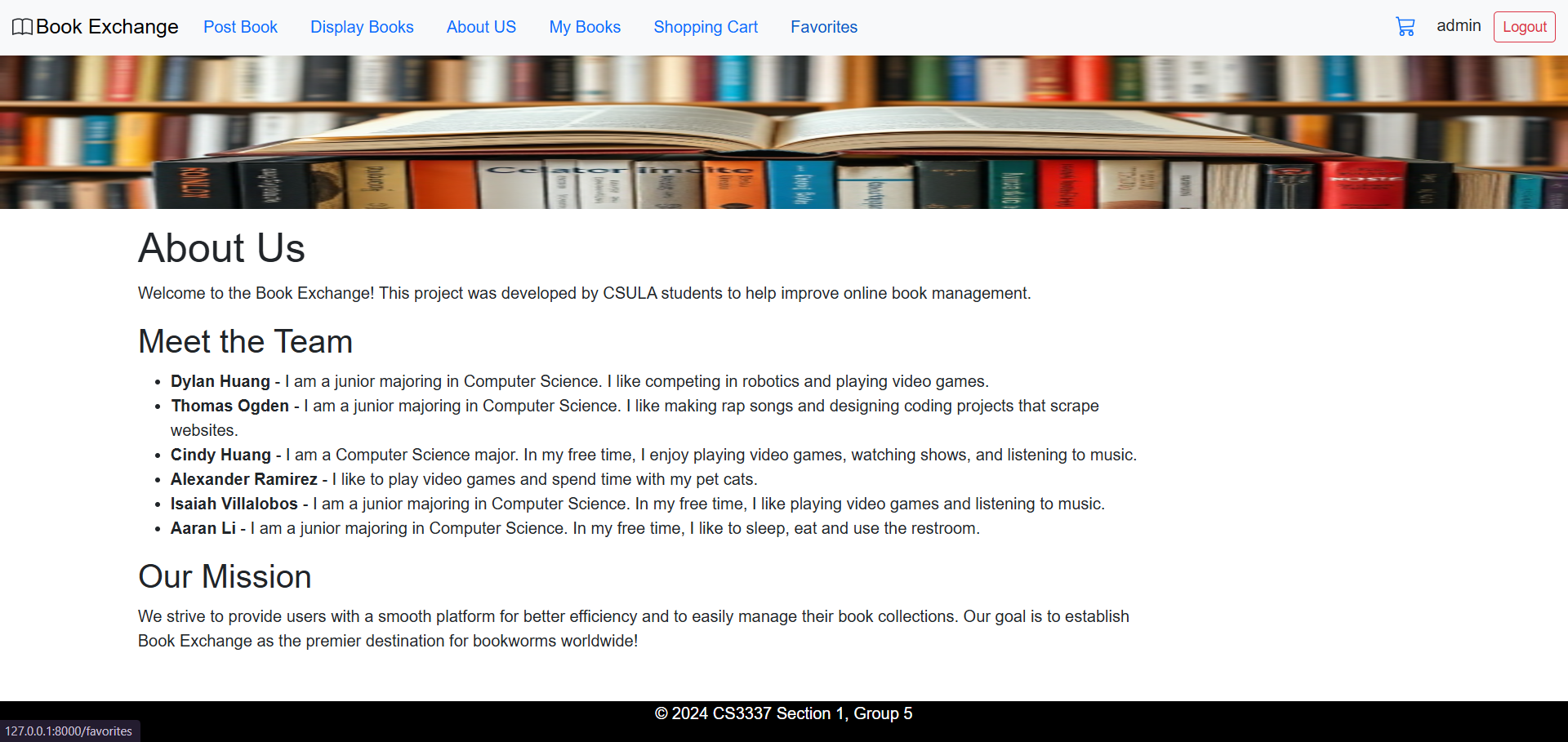
* **Registration**
  + The registration page will have users create a unique username and password with the option to enable or disable the password-based authentication.



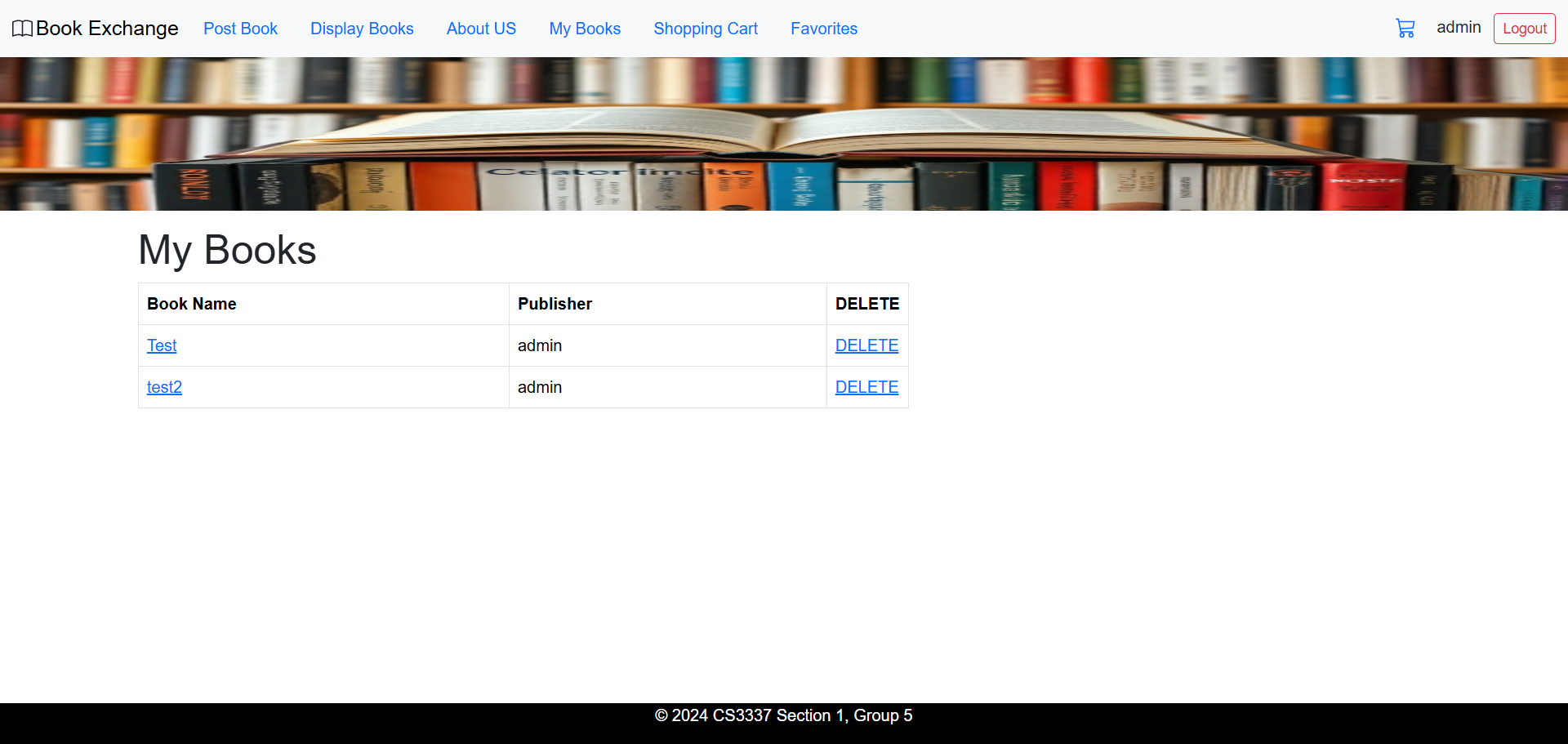
* **Home Page**
  + Displays a welcome message for all users with a message indicating click the menu to begin their experience.



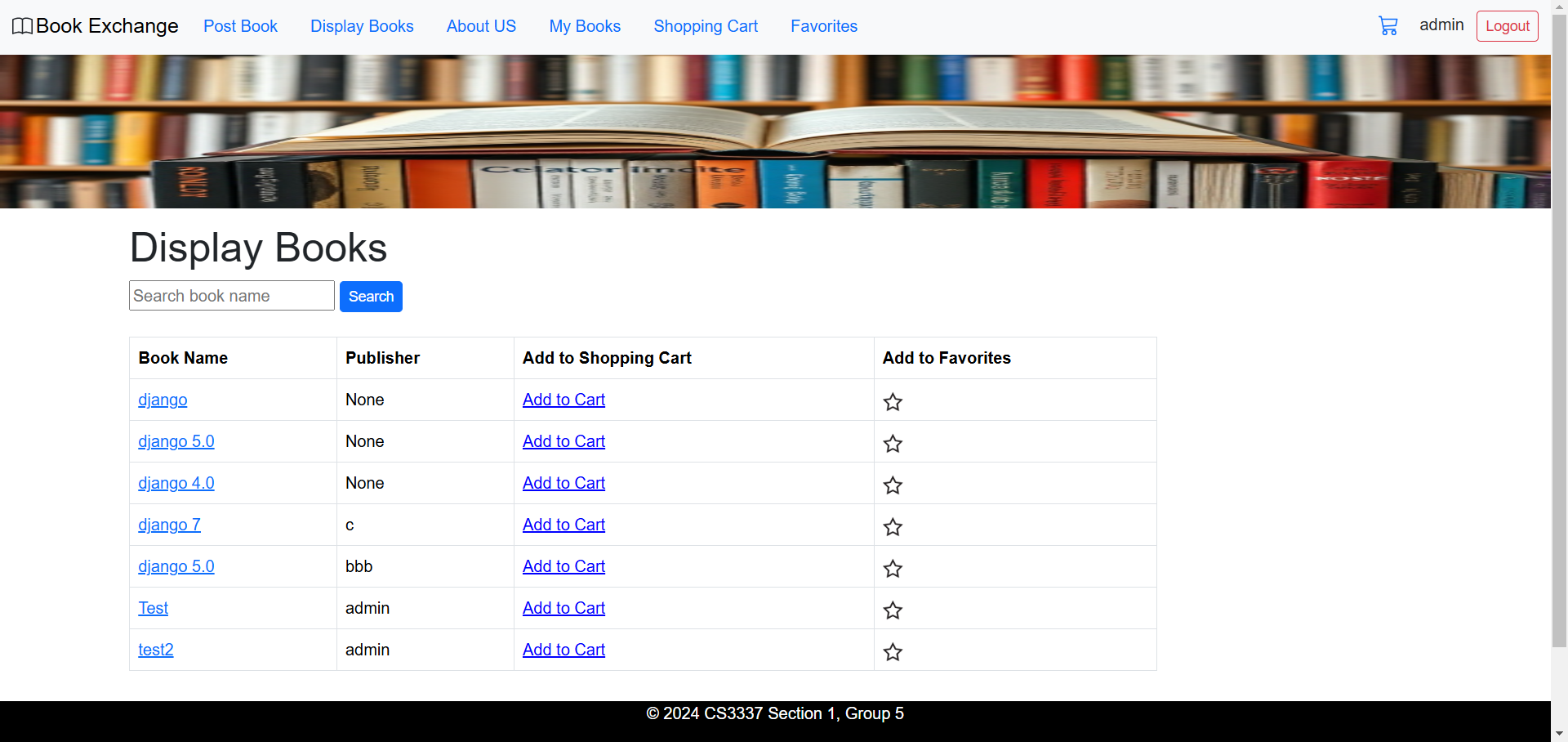
* **About Us**
  + A brief introduction of the development team and the goal for creating this website.



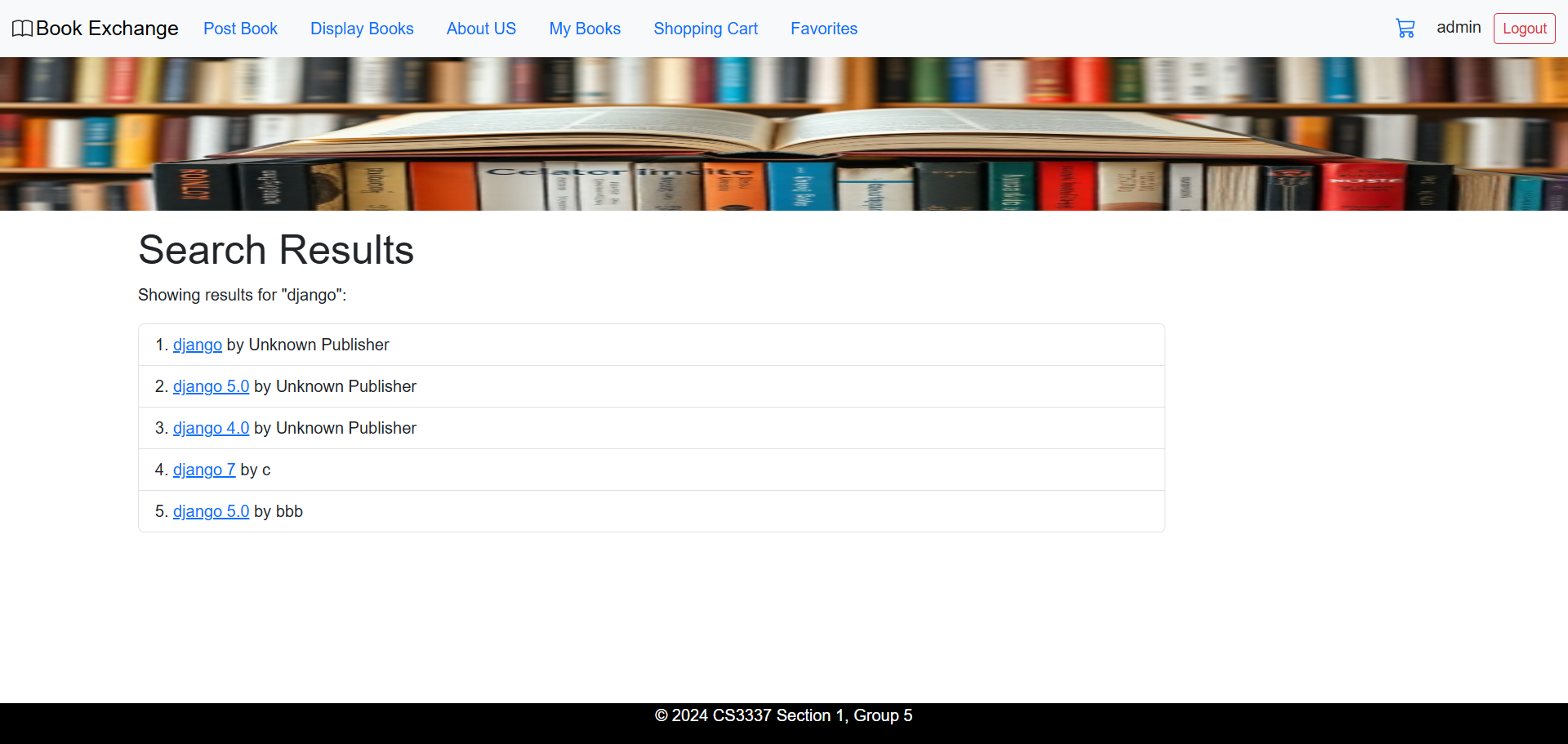
* **My Books**
  + A page only accessible by registered users who have logged in. Displays all books that have been published by the user with the option to delete these books.



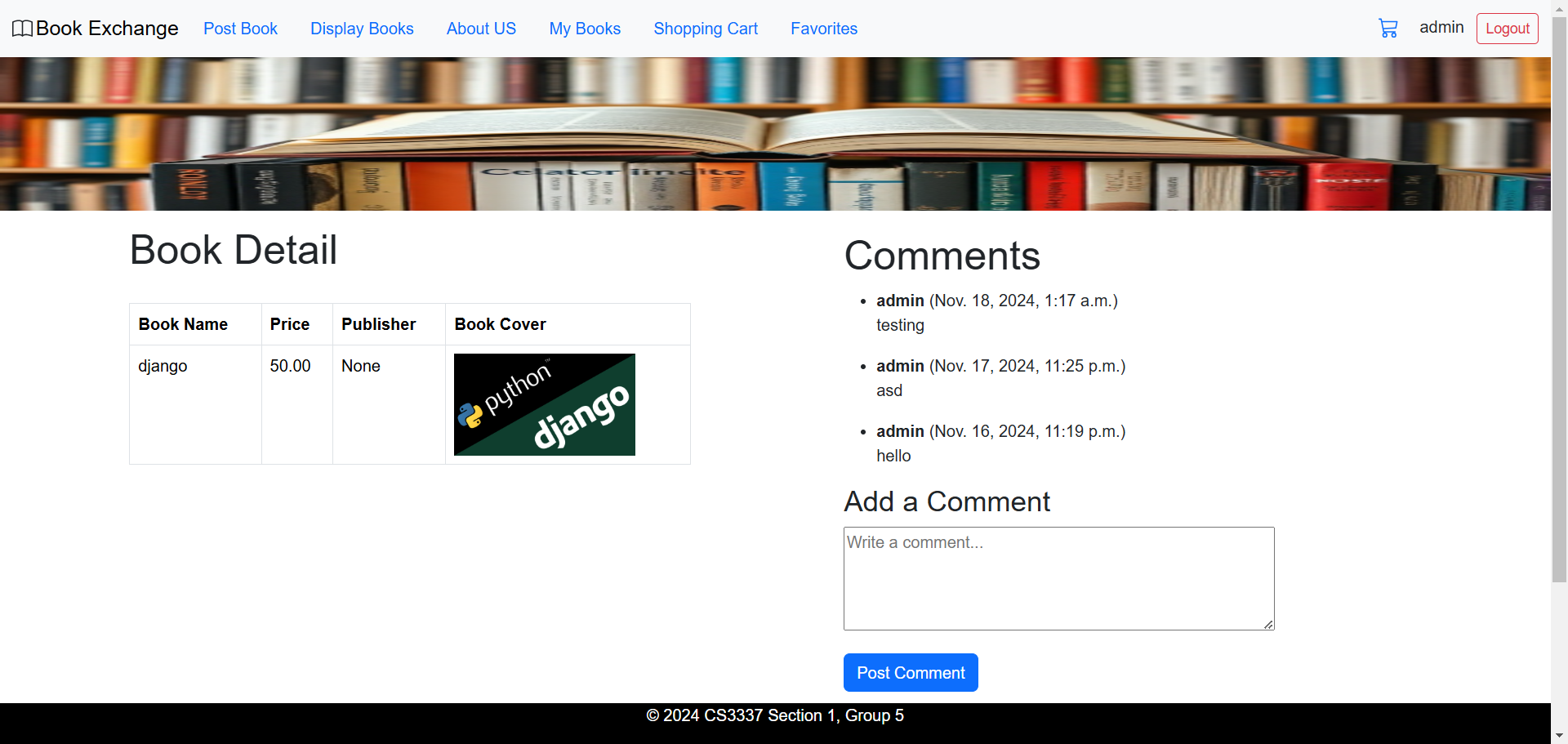
* **Display Books**
  + Displays all books in the system with publisher name and options to add to shopping cart or to a favorites list. The search function is also available here to search for a book by its name or by its publisher.



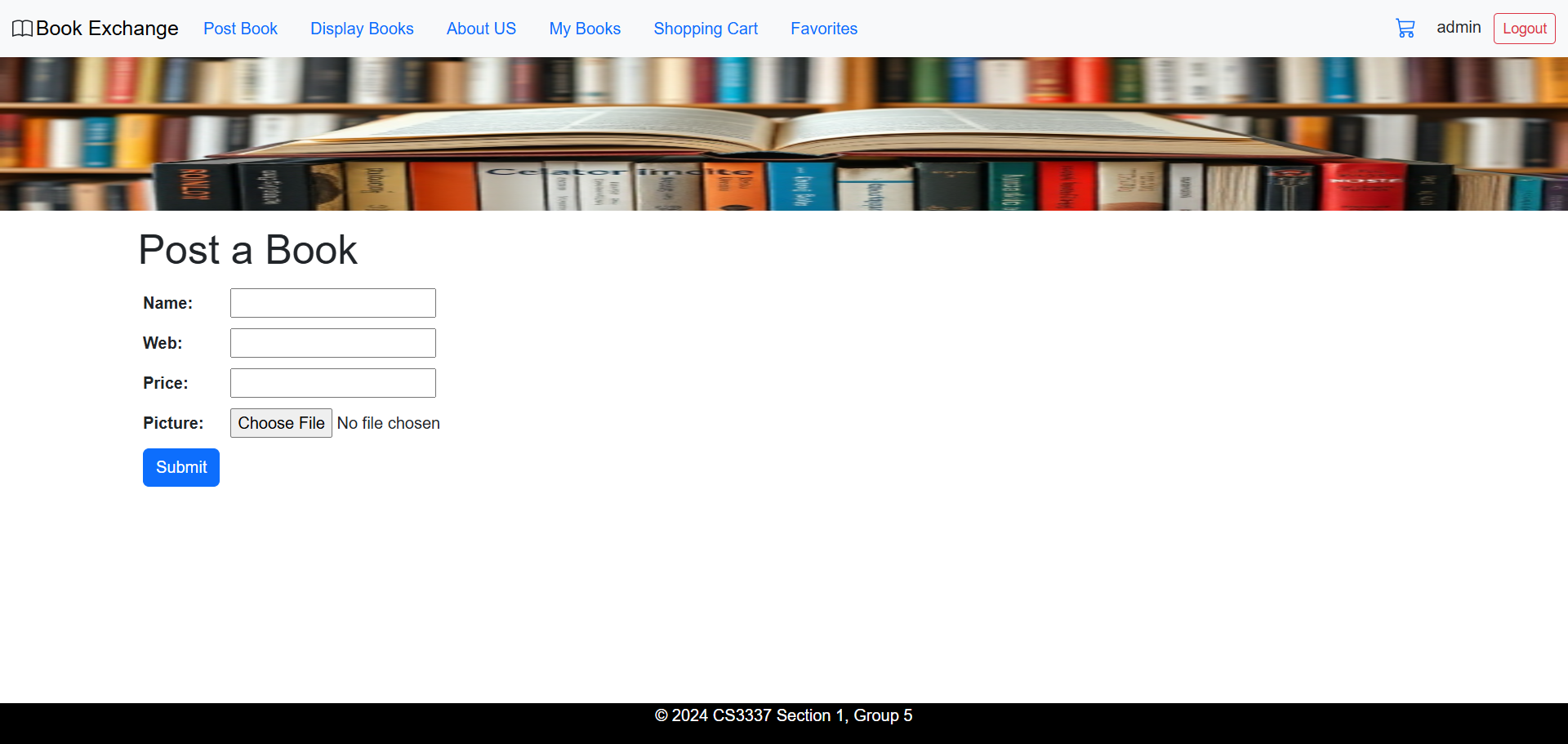
* **Search Results**
  + Search results are displayed in an ordered list followed by the publisher name, if known.



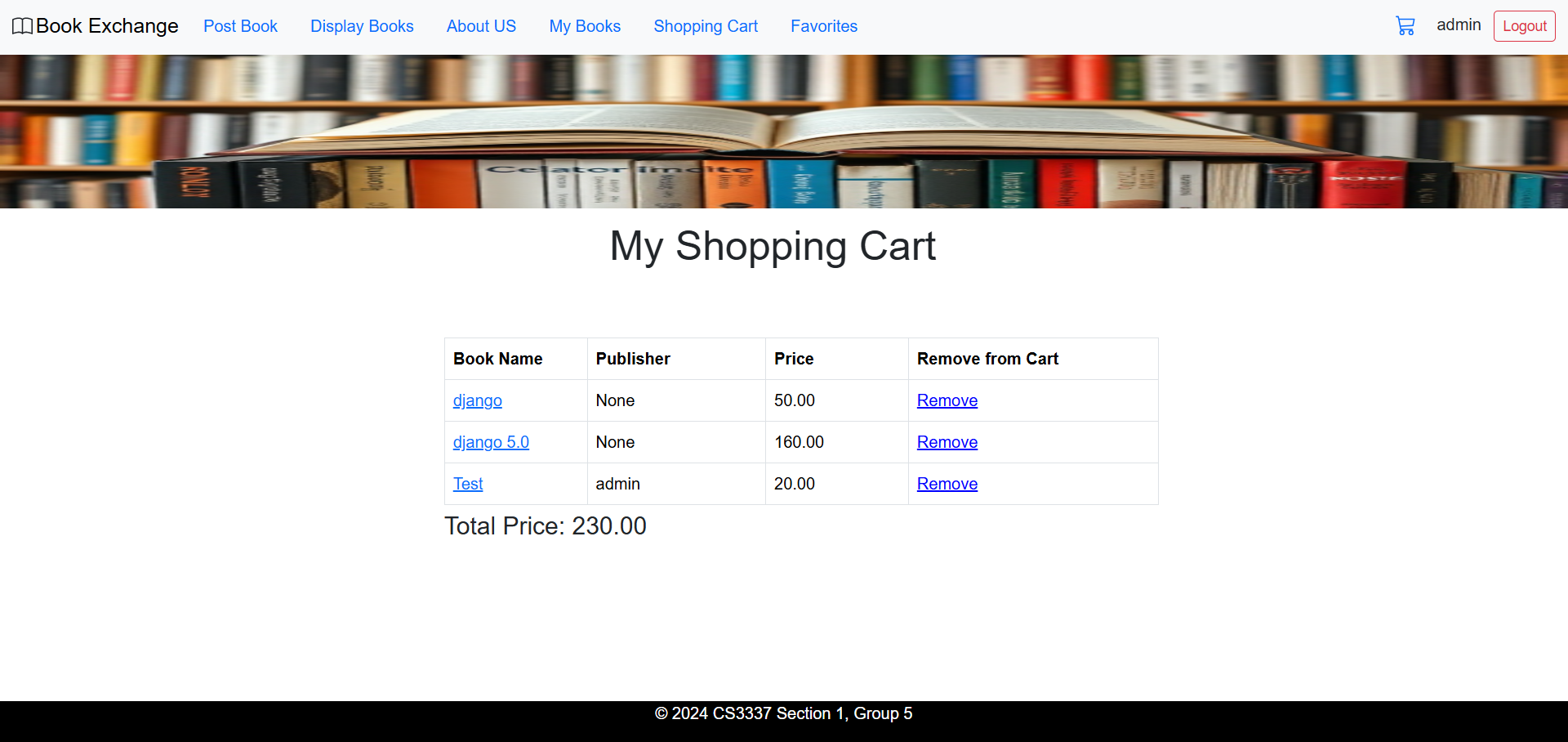
* **Book Details**
  + Displays details for the book that has been selected, which includes book name, book price, publisher, and book cover. To the right, there are also comments displayed. The user has the option to post their own comments as well.



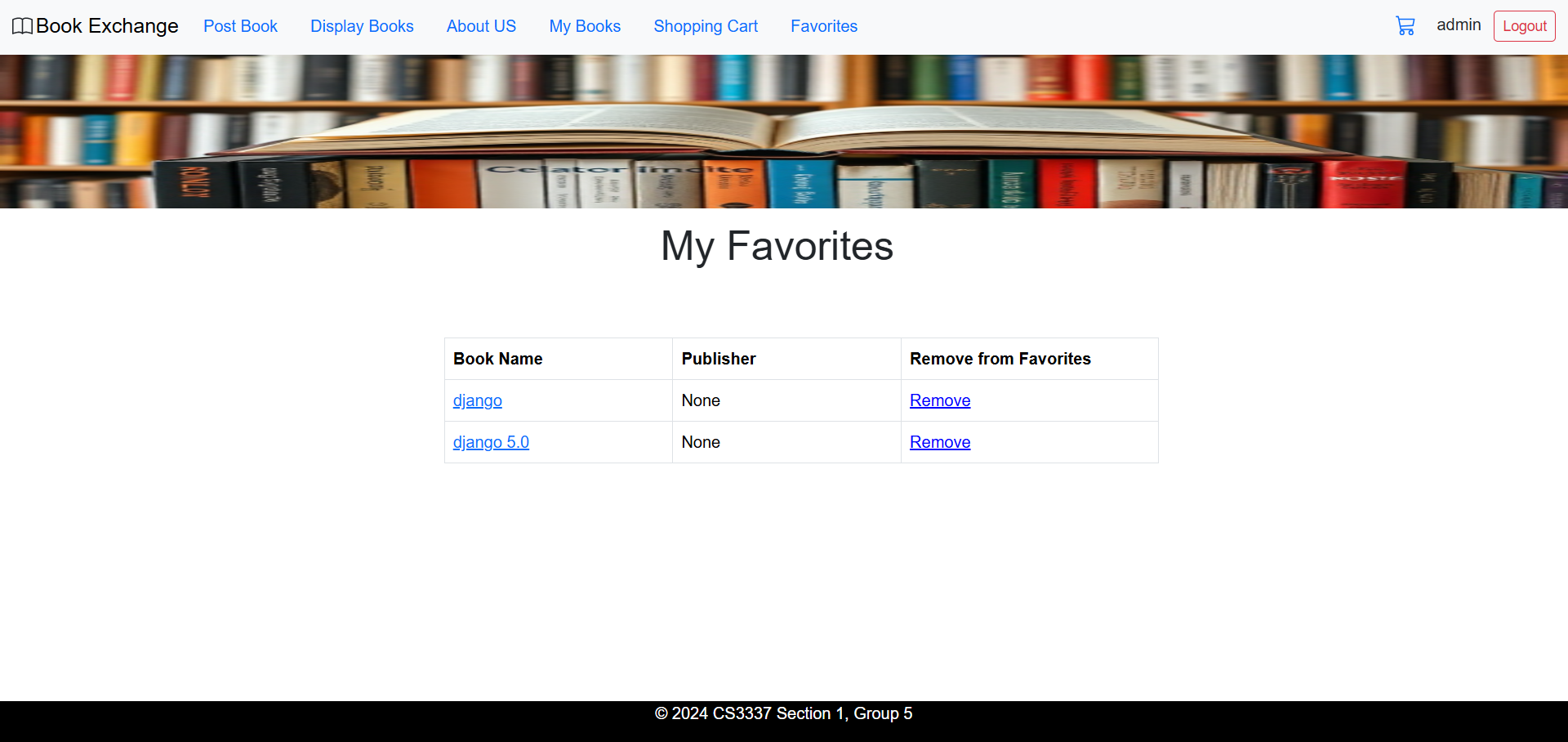
* **Post Book**
  + Displays a form for the user to post their own book. It asks the user for the book name, a website link, the book price, and a valid .JPG image for the book cover.



* **Shopping Cart**
  + Shopping cart displays all books a user has added into their cart to review and purchase, though a payment gateway has yet to be developed at the current scope. Displays total price of the cart below and gives users the option to remove a book from their shopping cart. The shopping cart is accessible via the navigation bar through the “Shopping Cart” tab or the shopping cart icon in the top-right corner of the web page.



* **Favorites**
  + Displays a user’s list of favorite books with the publisher name. There is also an option to remove a book from favorites.



Section S: GIossary

An ordered list of defined terms and concepts used throughout the document.

1. **API**: Stands for Application Programming Interface. It’s the set of protocols for building and interacting the software.
2. **Backend**: The part of a software system that makes an application work and is not usually visible or accessible to a user of the system.
3. **Bootstrap**: A free, open-source framework that helps web developers create responsive websites and web apps.
4. **CRUD (Create, Read, Update, Delete)**: The four basic operations for managing and creating data in computer programming.
5. **Database**: An organized collection of structured information, or data, typically stored electronically in a computer system.
6. **Django**: Python based web framework that runs on a web server. Follows the model-template-views architectural pattern.
7. **Frontend**: A software interface (such as a graphical user interface) designed to enable user-friendly interaction with a computer by a user.
8. **HTTP**: Stands for Hypertext Transfer Protocol. It's used to transfer data over the web.
9. **Interface**: A defined set of rules and methods that describe how different software components or systems can interact and exchange data with each other.
10. **MVC**: Stands for Model-View-Controller. This is the software design pattern used in developing the user interfaces.
11. **ORM**: Stands for object relational mapping. It’s used by django to interact with the databases in an object-oriented way.
12. **Subsystem**: A distinct, self-contained part of a larger software system that performs a specific set of functions.
13. **UI**: Stands for user interface. It’s the point of contact between the user and the computer. This represents the front-end of the application.