

ICE4016 Database

<Title >

Pledge for Report Writing

1. I will not plagiarize other students' reports or copy content from various reports.
2. I will not obtain the main content of the report through internet websites or other sources.
3. I will not manipulate the content of the report.
4. I will properly cite the sources of the documents I refer to when writing the report.
5. I will not show my report to other students before submitting it.

I solemnly pledge to maintain ethical behavior while writing reports and uphold my honor as a telecommunications engineering student.

Date: 15 December 2024

Department of Information and Communication Engineering

Name: Murad Ibrahimov

Student ID: 12244819



Contents

[1. Introduction 3](#_Toc185198961)

[2. Overview 3](#_Toc185198962)

[3. EER Diagram 9](#_Toc185198963)

[4. Database Design 11](#_Toc185198964)

[4.1 Normalization 11](#_Toc185198965)

[4.2 Constraints 13](#_Toc185198966)

[4.3 INDEX 15](#_Toc185198967)

[4.4 Transactions 16](#_Toc185198968)

[5. Logging 17](#_Toc185198969)

[6. Web Application Execution screens 17](#_Toc185198970)

[7. Locking 24](#_Toc185198971)

[Discussion 33](#_Toc185198972)

# Introduction

* **Objective**: This project focuses on designing and implementing a database-driven bookstore web application. The application supports both administrative operations and customer-facing functionalities, ensuring seamless data management and user interaction.
* **Scope**: The system enables administrators to manage books, authors, awards, warehouses, and inventory, while customers can search for, reserve, and purchase books.
* **Tools and Technologies**:
  + **Backend**: Node.js
  + **Frontend**: Handlebars.js (HBS)
  + **Database**: MySQL
  + **IDE/Tools**: MySQL Workbench, Visual Studio Code

# Overview

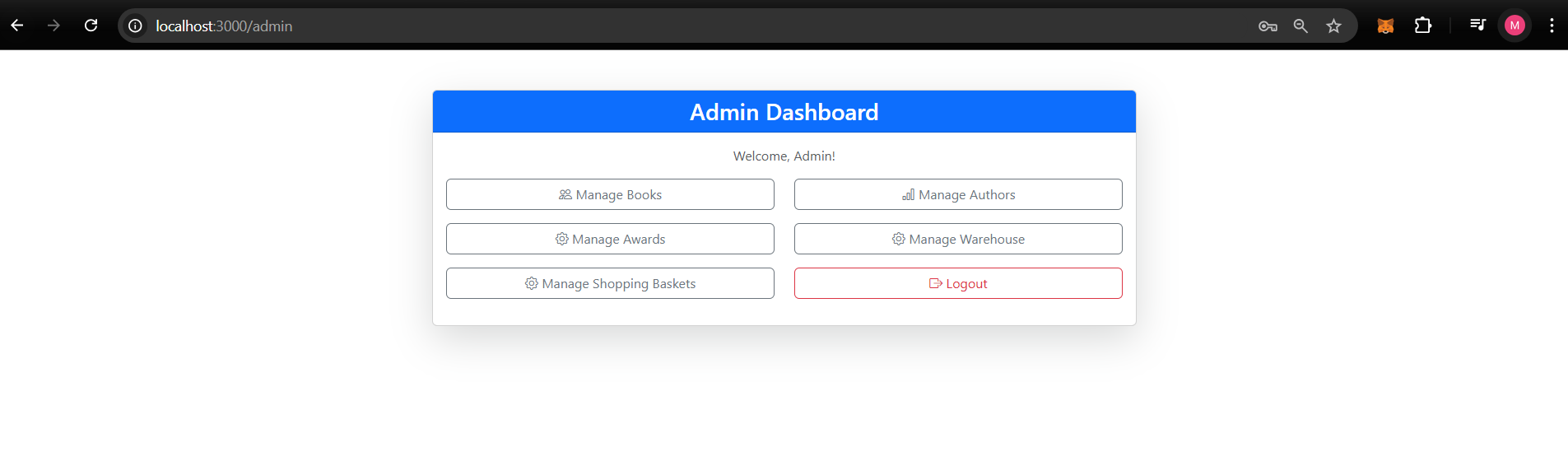
**Problem Statement**

In an increasingly digital world, bookstores need robust systems to efficiently manage inventory, customer reservations, and sales. This project addresses the need for an integrated database and web application to:

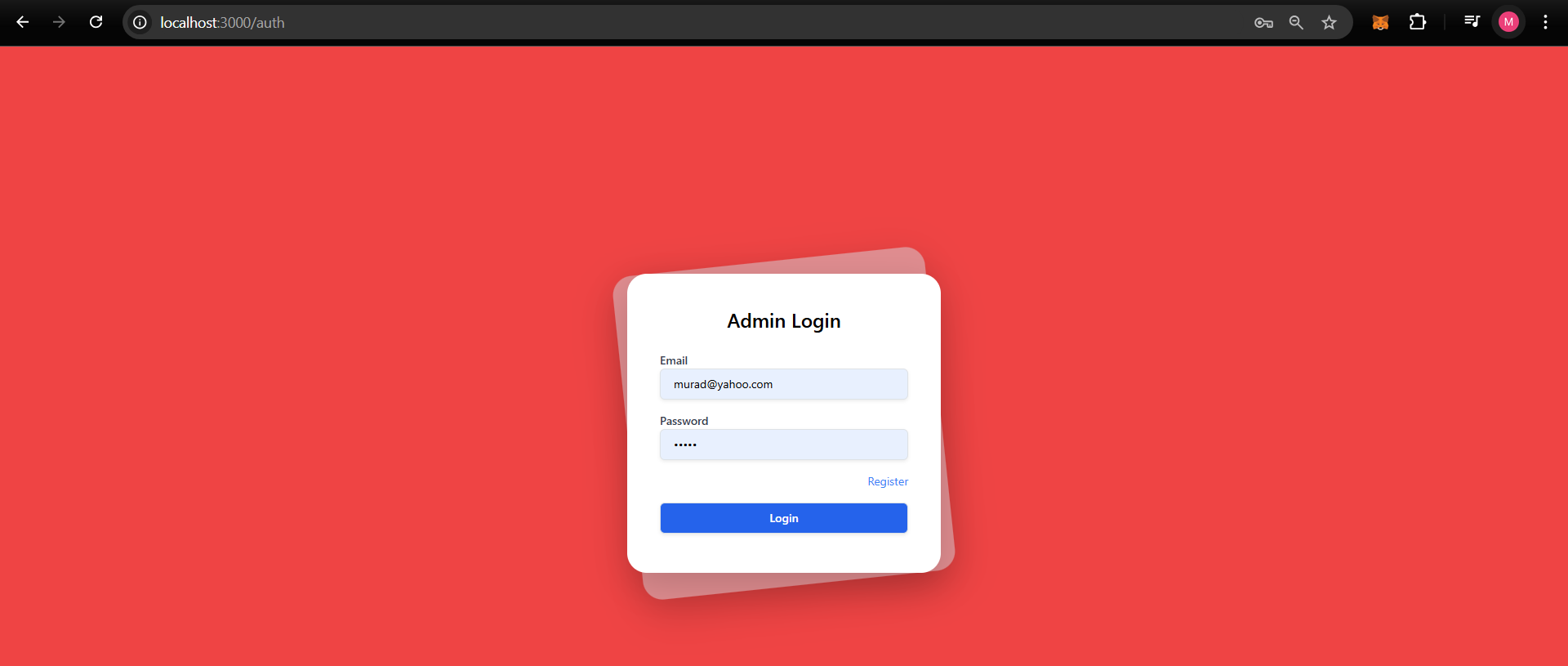
* Centralize book, author, and award management.
* Provide real-time stock updates across warehouses.
* Simplify customer interactions through a user-friendly interface for reservations and purchases.

**Functional Requirements**

1. **Admin Pages**: Admin have dashboard, which gave access to Manage Books, Manage Authors, Manage Awards, Manage Warehouses, Manage Shopping baskets, and of course, logging out.



1. **Login Authentication**: Admins must log in to access features. Admin uses /auth for logging in.



1. **Data Management**:

Major requirements of task are add, edit, and delete information about:

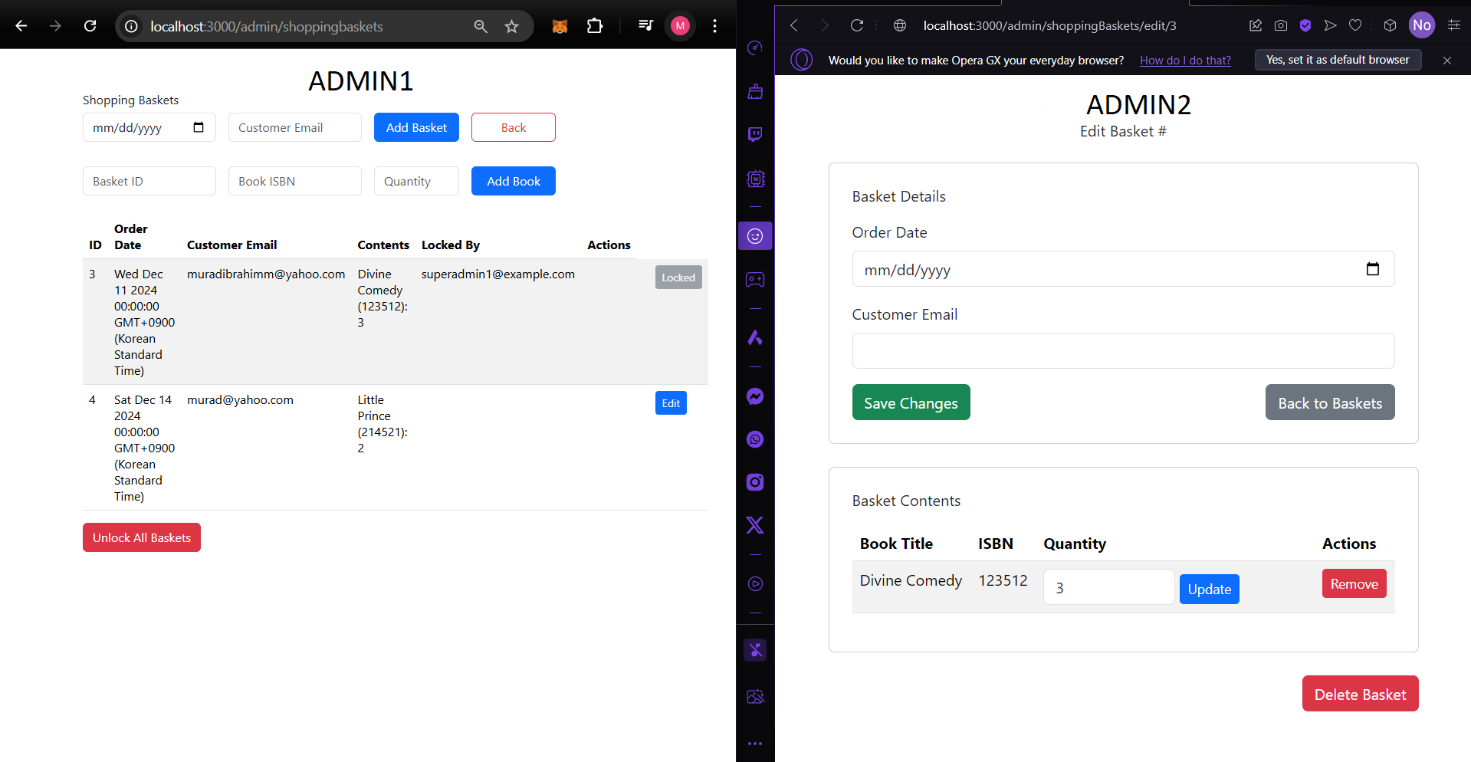
* + - Books (title, author, inventory, awards).
    - Authors (biographies, associated awards).
    - Awards (details and recipients).
    - Warehouses (location and stock levels).
    - Shopping baskets of customers(content)

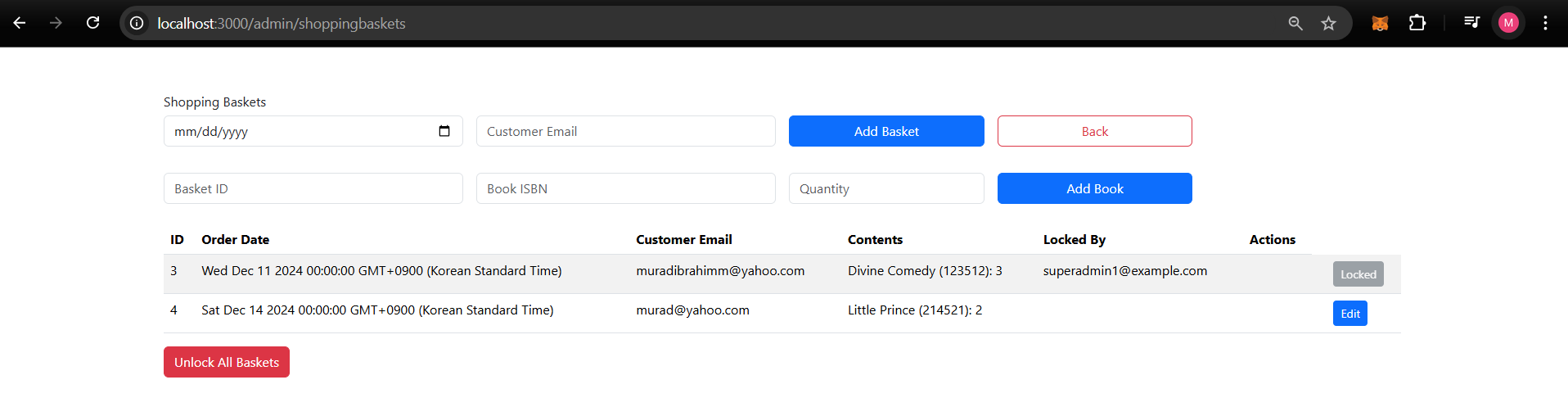
All off them will be done separately onwards.

1. **Concurrency Handling**:
   * Ensure only one admin edits a data entry at a time to prevent conflicts.

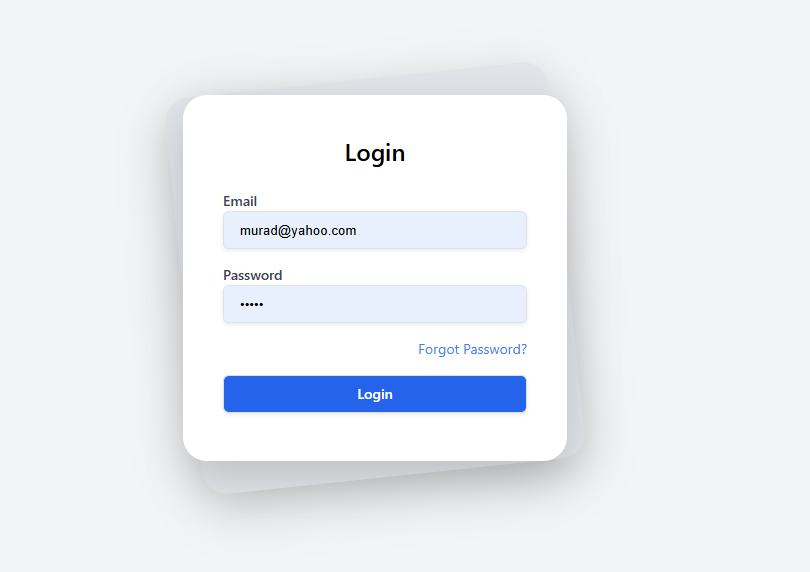
I used transactions and custom locking system to prevent one admin to edit something in the database, while other one editing.

As it seems from picture while my second admin which have [superadmin1@example.com](mailto:superadmin1@example.com) mail edits baskets, other one cannot access to this basket

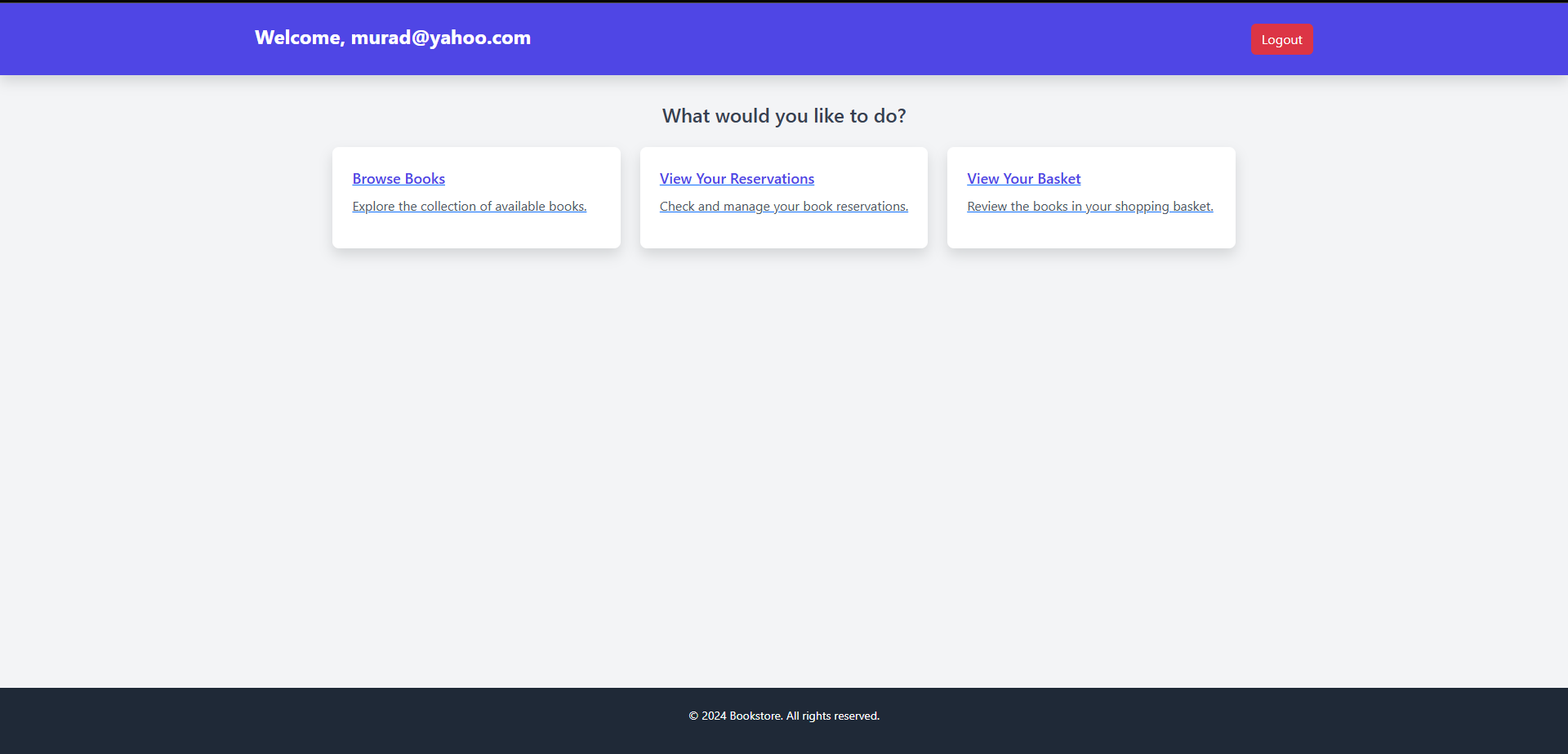




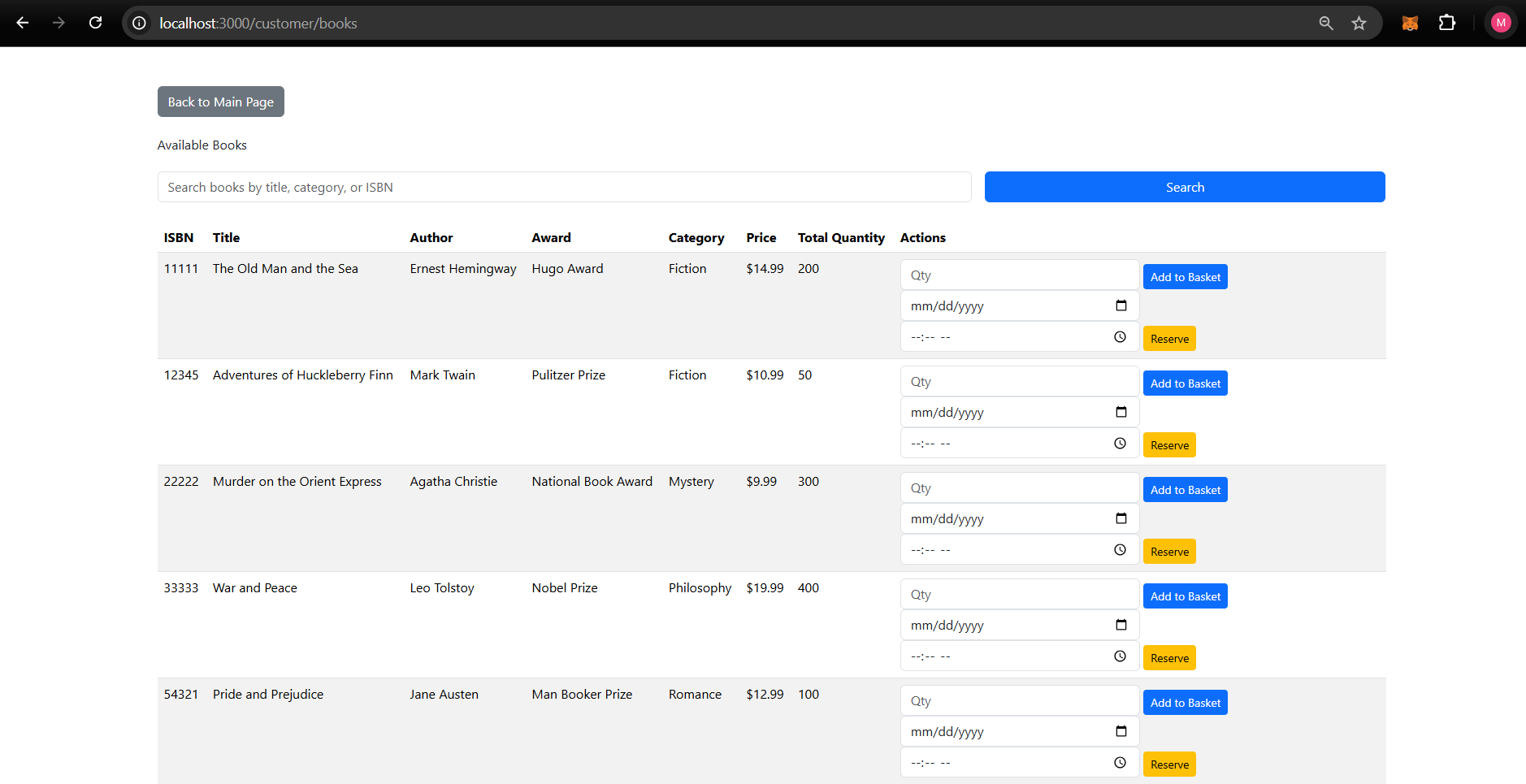
1. **Customer Pages**:

**Created separately login page for customers for logging in**:  


After logging in Customer can navigate in site by clicking to cards.

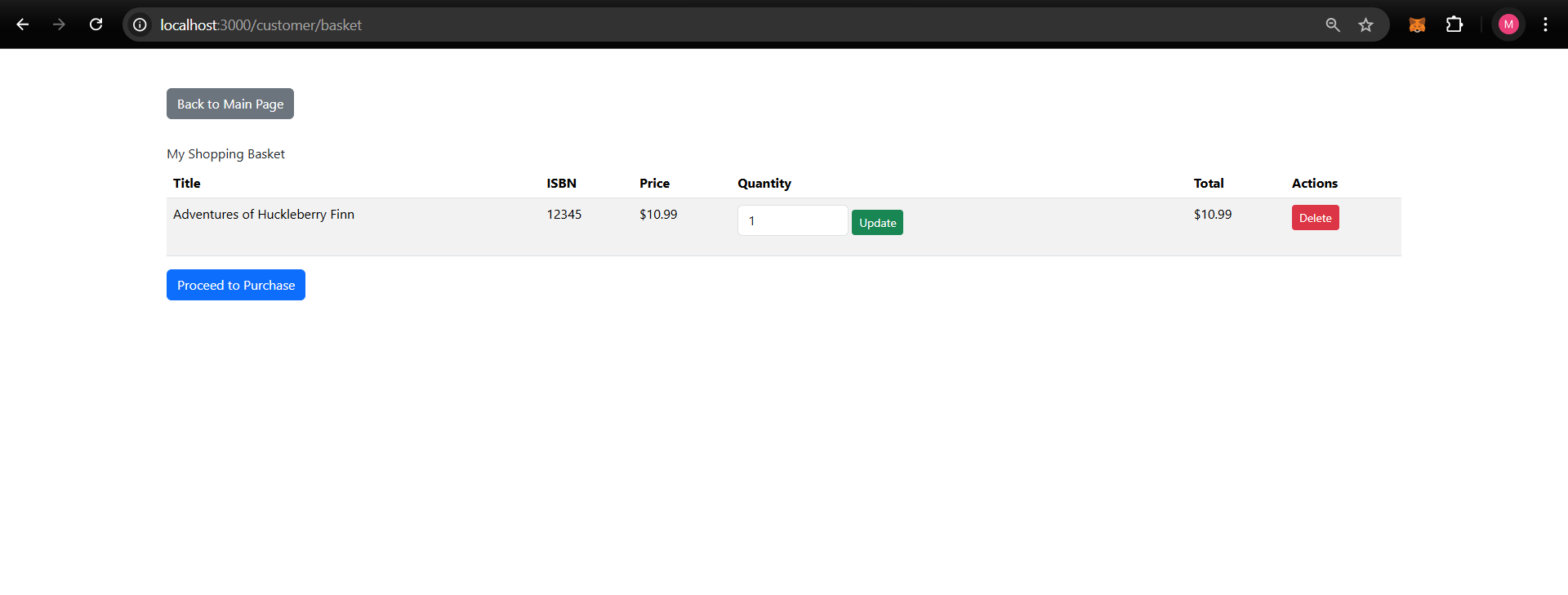


1. **Search Functionality**:



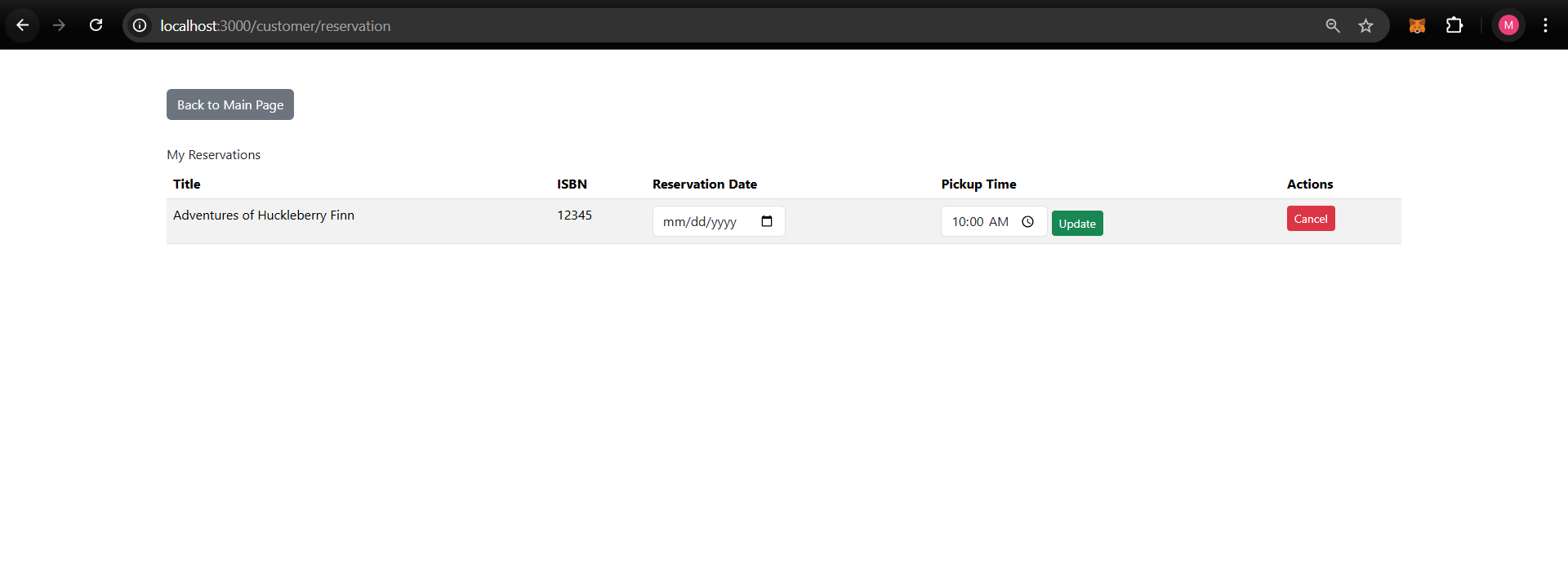
* + Search for books, authors, or awards by name.
  + View total available stock (summed across all warehouses).

1. **Shopping Basket**:



* + Add books to the shopping basket.
  + Review, edit, or remove items before purchase.

1. **Reservations**:



* + Reserve books for pickup, specifying a time slot.
  + Cancel or modify reservations as needed.
  + Prevent overlapping reservations within 10 minutes of the selected time slot.

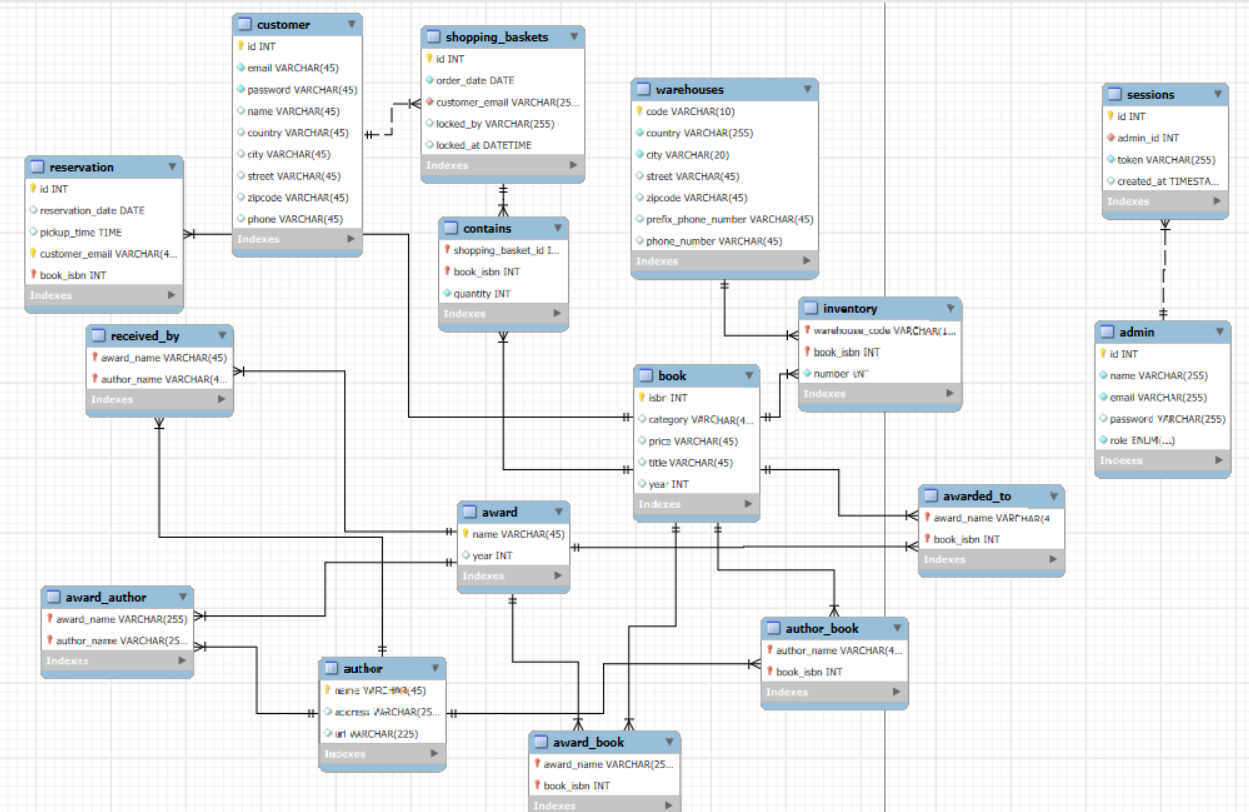
1. **Purchases**:
   * Confirm purchases and deduct inventory accordingly.
   * Disallow purchases if stock is insufficient.
   * Using My well designed User Interface, customers can directly buy the books in their baskets.

**Non-Functional Requirements**

1. **Performance**:
   * Optimized database queries using indices for faster search and retrieval.
   * Efficient caching mechanisms to reduce server load.
2. **Security**:
   * Encrypted storage of customer passwords.
   * Role-based access control for admin and customer accounts.
3. **Reliability**:
   * Transactional integrity to ensure database consistency during purchases and reservations.
   * High availability of the system through proper server and database setup.

# 3. EER Diagram

**Diagram Overview**



1. The EER diagram captures the relationships between key entities in the bookstore database. The primary entities include:

* **Customer**: Stores information about customers such as email, name, address, and phone.
* **Admin**: Manages data related to books, authors, awards, and inventory.
* **Book**: Represents details of books, including ISBN, title, category, price, and publication year.
* **Author**: Contains author details such as name, address, and URL.
* **Award**: Records awards associated with books and authors.
* **Warehouse**: Tracks inventory levels for books in different locations.
* **Shopping Basket**: Manages customer orders, including books and quantities.
* **Reservation**: Handles book reservations with pickup times.

**Attributes**

* **Customer**: id (PK), email (unique), password, name, phone, address.
* **Admin**: id (PK), name, email (unique), password, role.
* **Book**: isbn (PK), title, category, price, year.
* **Author**: name (PK), address, url.
* **Award**: name (PK), year.
* **Warehouse**: id (PK), country, city, street, zip\_code, phone.
* **Shopping Basket**: id (PK), order\_date, customer\_email (FK).
* **Reservation**: id (PK), reservation\_date, pickup\_time, customer\_email (FK), book\_isbn (FK).

**Assumptions**

1. Each book is uniquely identified by its ISBN.
2. Authors can be associated with multiple books, forming a many-to-many relationship.
3. Awards can be associated with both books and authors, creating two many-to-many relationships.
4. Warehouses store inventory for multiple books, forming a many-to-many relationship with book through inventory.
5. Customers can reserve multiple books, and each reservation must include a valid pickup time.
6. Shopping baskets are customer-specific and linked to their email addresses.

# 4. Database Design

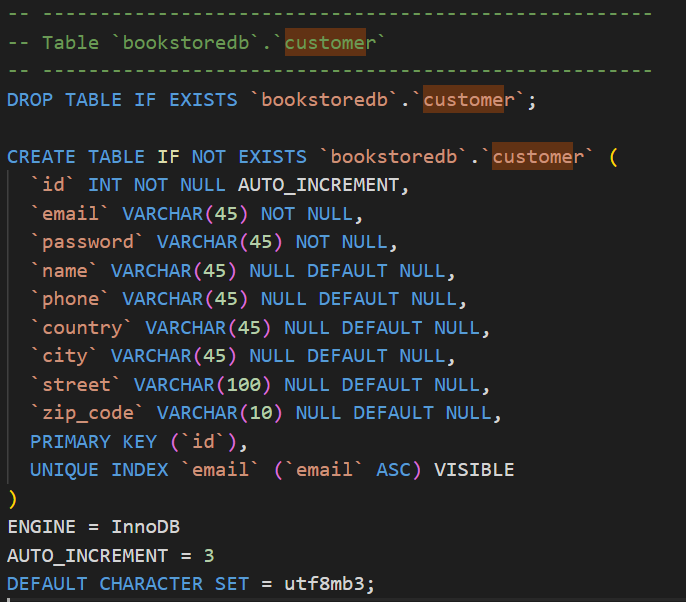
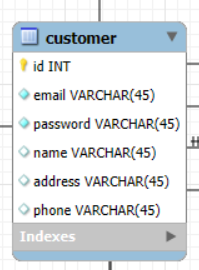
# 4.1 Normalization

1. Normalization was applied to ensure data integrity, minimize redundancy, and enhance query performance. The following steps were taken:

**First Normal Form (1NF):**

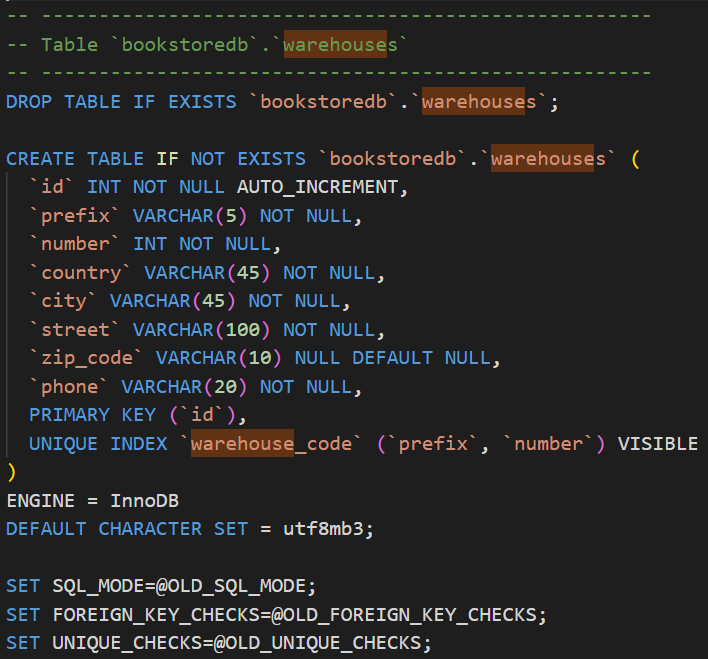
1. To achieve 1NF, all attributes were ensured to contain atomic values, and each entry in the tables was uniquely identifiable. The following actions were taken as part of this process:

* **Separation of Customer Address and Phone Numbers**: Instead of storing multiple addresses or phone numbers in a single column, these details were moved to separate tables:
  + **Customer Address Table**: Stores the addresses of customers, linked to the main Customer table through a foreign key.
  + Customer Phone Number Table: Stores phone numbers of customers, allowing multiple entries per customer if needed.



1. This ensures flexibility and scalability in managing customer contact information.

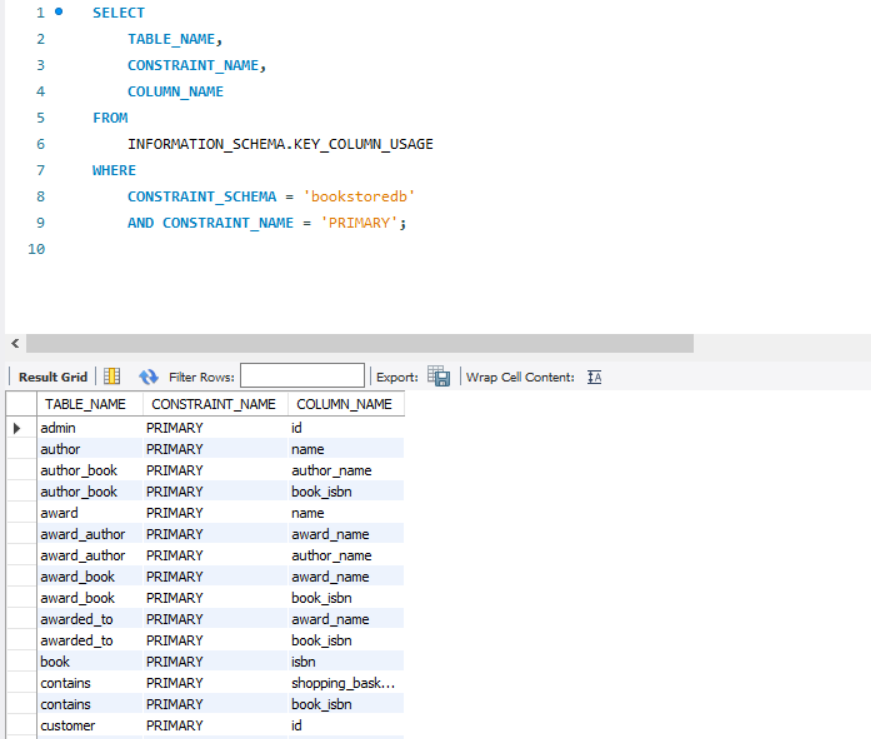
* **Separation of Warehouse Address**: The address details for each warehouse were moved to a dedicated Warehouse Address table. This prevents redundancy in case a warehouse has multiple locations or if address components need to be queried individually.
* Ware house address separating allowed multiple entries if needed.



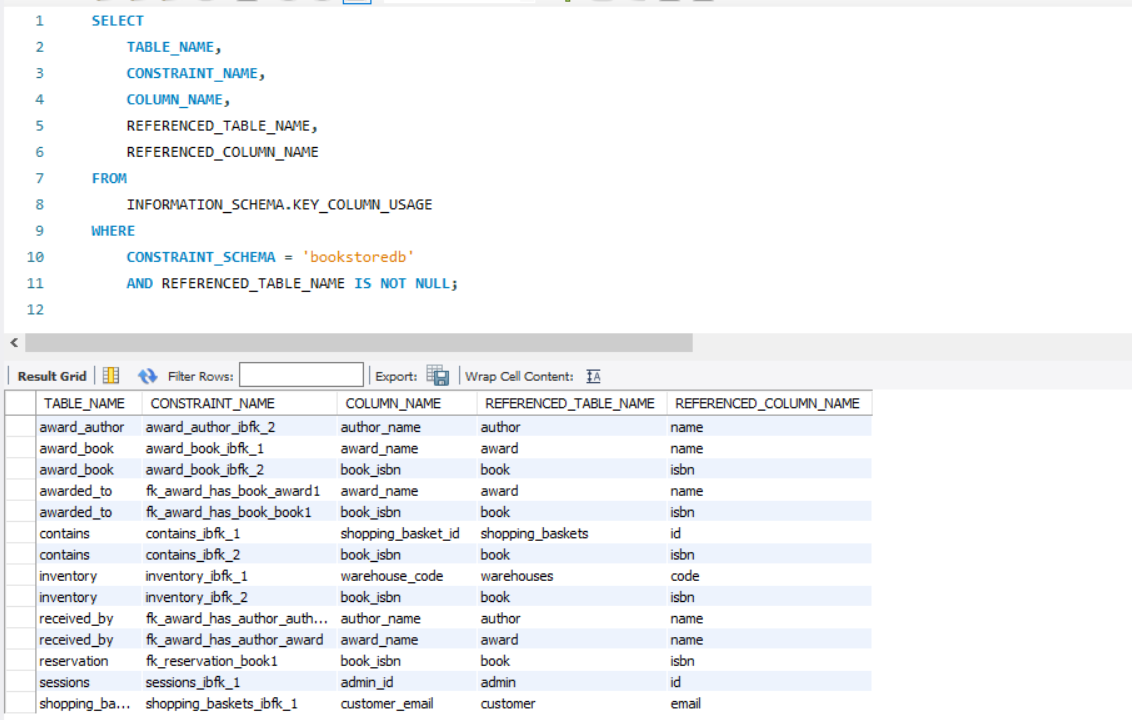
1. This approach adheres to 1NF principles by ensuring:
2. Atomicity of all attribute values.
3. No repeating groups or multivalued attributes in a single column.
4. Unique identification of rows in each table.

# 4.2 Constraints

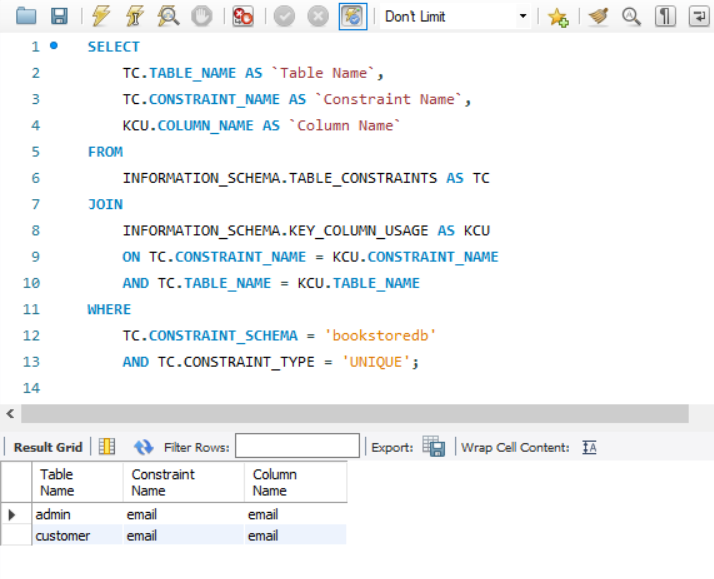
* **Primary Keys**: Ensured unique identification of records in each table.



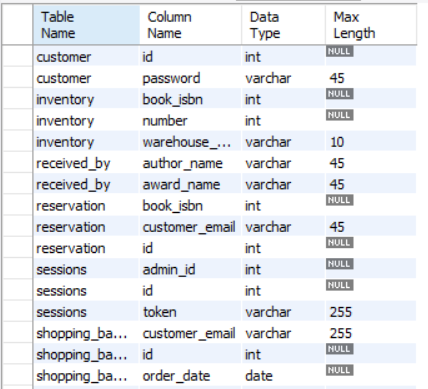
* **Foreign Keys**: Maintained referential integrity between related tables.



* **Unique Constraints**: Applied to columns like email to avoid duplication.



* **Not Null Constraints**: Applied to mandatory fields.

  and more.

# 4.3 INDEX

To improve query performance, the following indexes were implemented. All the indexes creating SQLs scripts are provided.

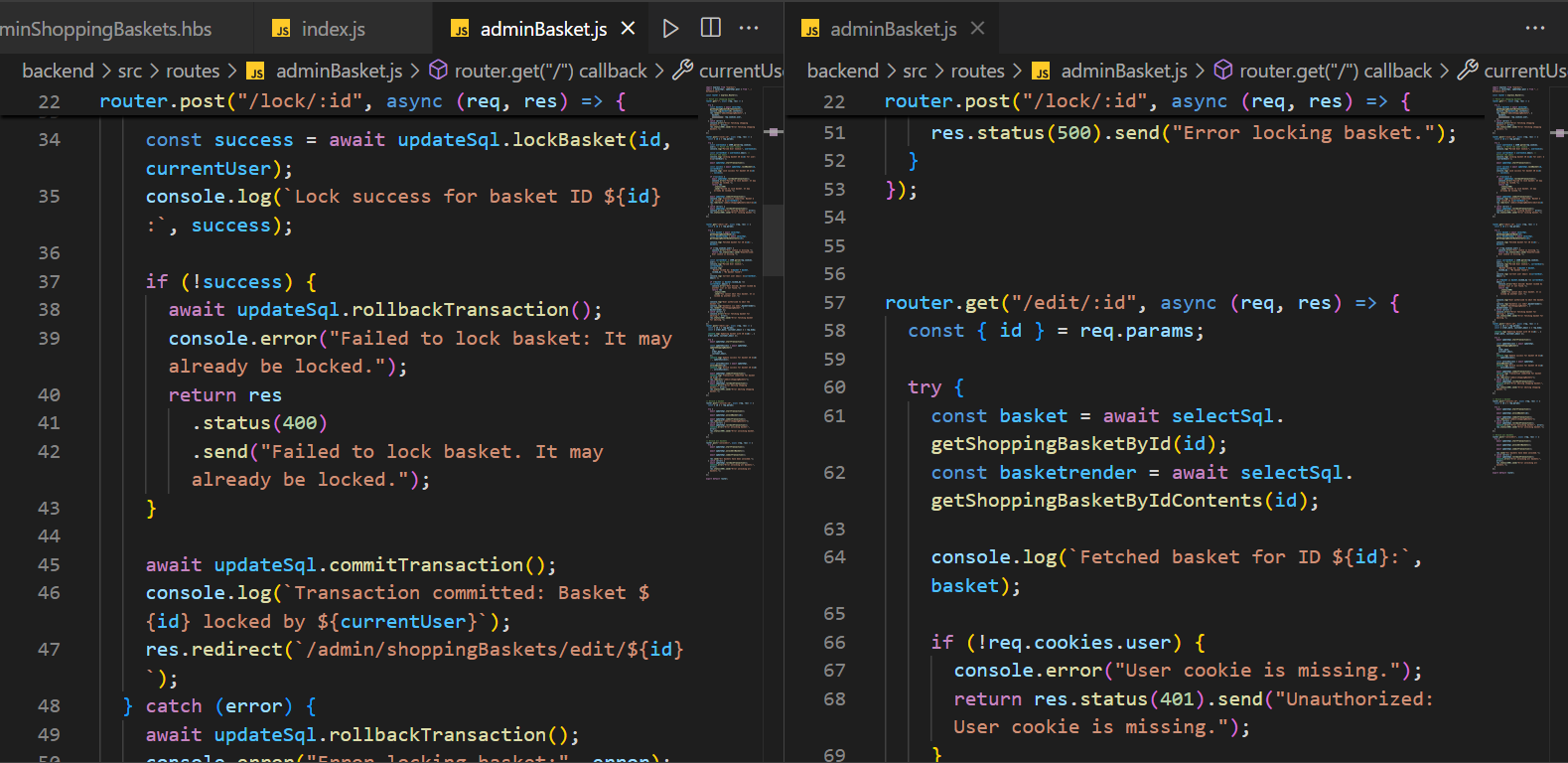
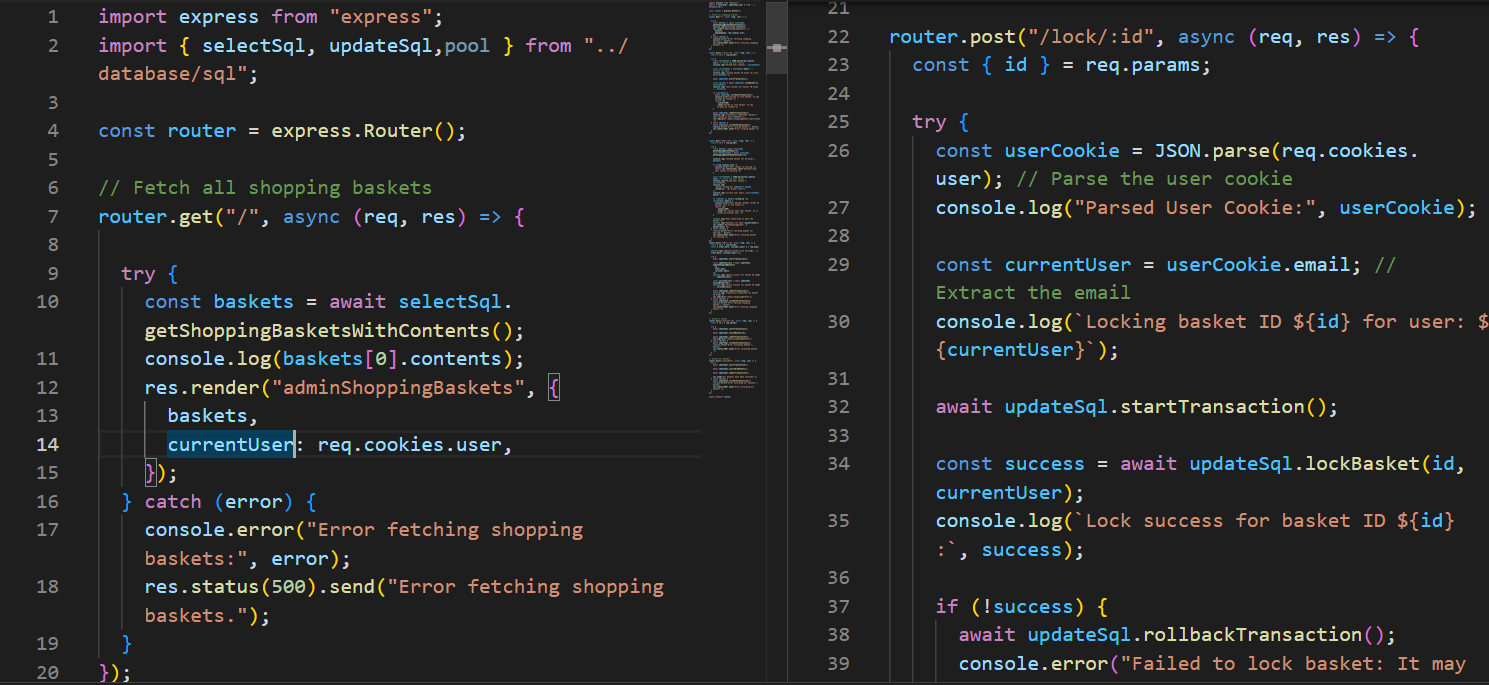
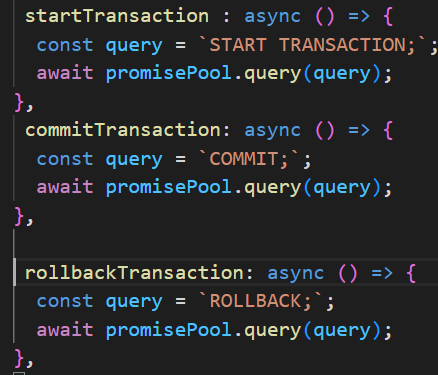
|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | **Index Name** | **Indexed Columns** | **Purpose** |
| admin | email | email | Optimize admin email lookups. |
| author\_book | book\_isbn | book\_isbn | Efficient author-book relationship. |
| customer | email | email | Improve customer email search. |
| book | PRIMARY | isbn | Unique identification of books. |
| reservation | fk\_reservation | customer\_email, book\_isbn | Quick lookup for reservations. |
| inventory | book\_isbn | book\_isbn | Fast inventory checks per book. |

Indexes ensure faster search and retrieval for critical operations, including:

* Customer login (customer.email).
* Book search by ISBN or title.
* Managing relationships between books, authors, and awards.
* Ensuring quick inventory availability checks for reservations and purchases.

# 4.4 Transactions

* **Purchase Transaction**:
  + Deduct inventory upon successful purchase.
  + Rollback transaction if stock is insufficient or payment fails.
* **Reservation Transaction**:
  + Check inventory before reserving books.
  + Rollback transaction if another reservation conflicts with the time slot.



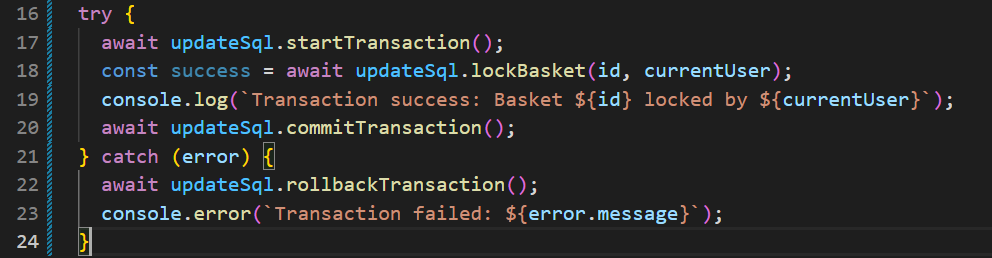
# 5. Logging

1. Logging was implemented to ensure accountability, facilitate debugging, and monitor system performance. Logs are generated for critical operations across both the admin and customer functionalities.

**Key Logging Features:**

1. **Error Logs**:
   * Captures errors during operations such as fetching, updating, or deleting records.
   * Example: Errors during database transactions (e.g., inventory updates or reservation conflicts) are logged with detailed error messages.
2. **Access Logs**:
   * Logs user activity, including login attempts and accessed pages.
   * Tracks actions performed by admins, such as adding, editing, or deleting records.
3. **Transaction Logs**:
   * Records all critical transactions (e.g., inventory deductions, reservations, and purchases).
   * Logs whether transactions were committed successfully or rolled back in case of failure.

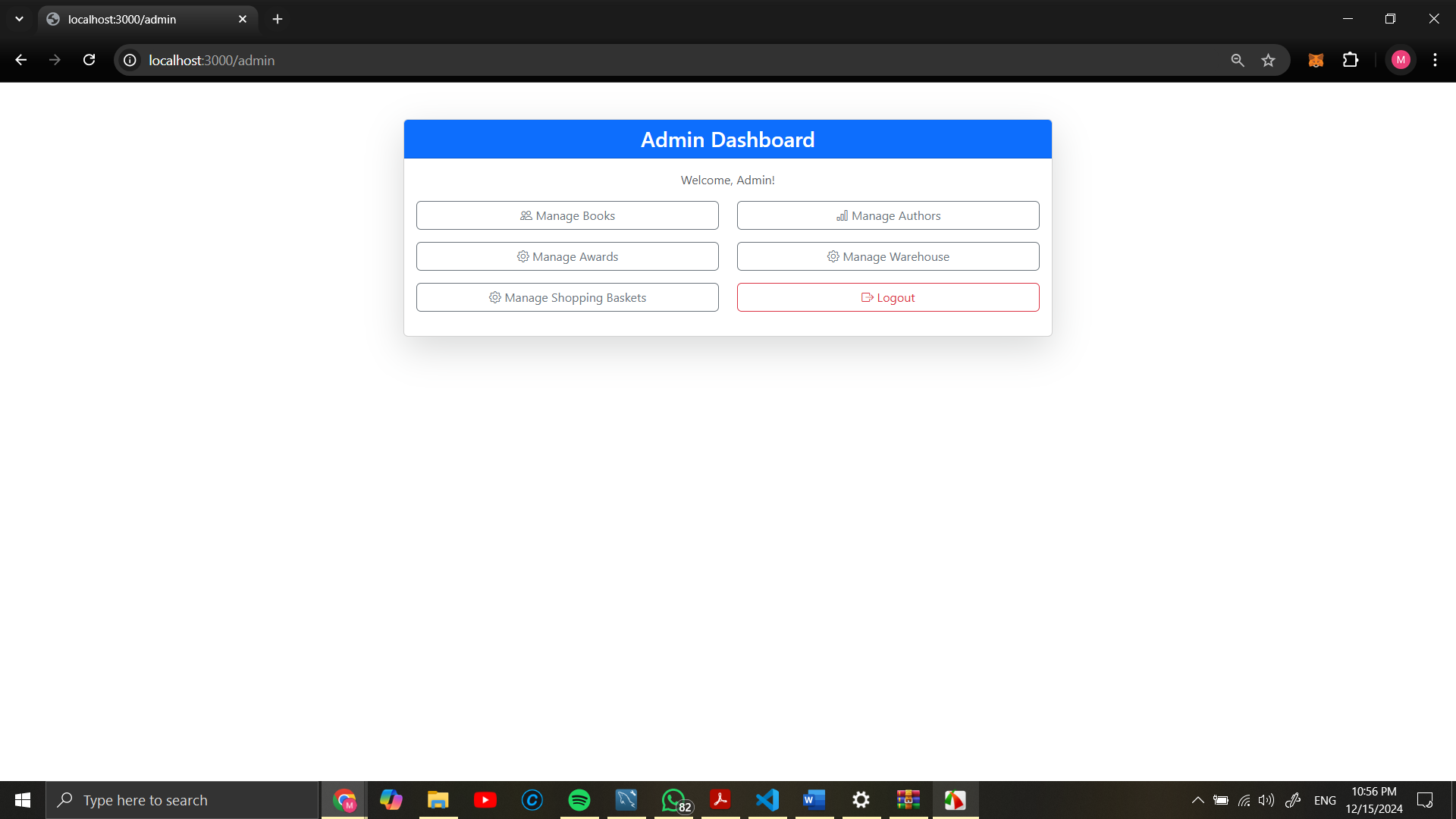
**Code Example:**

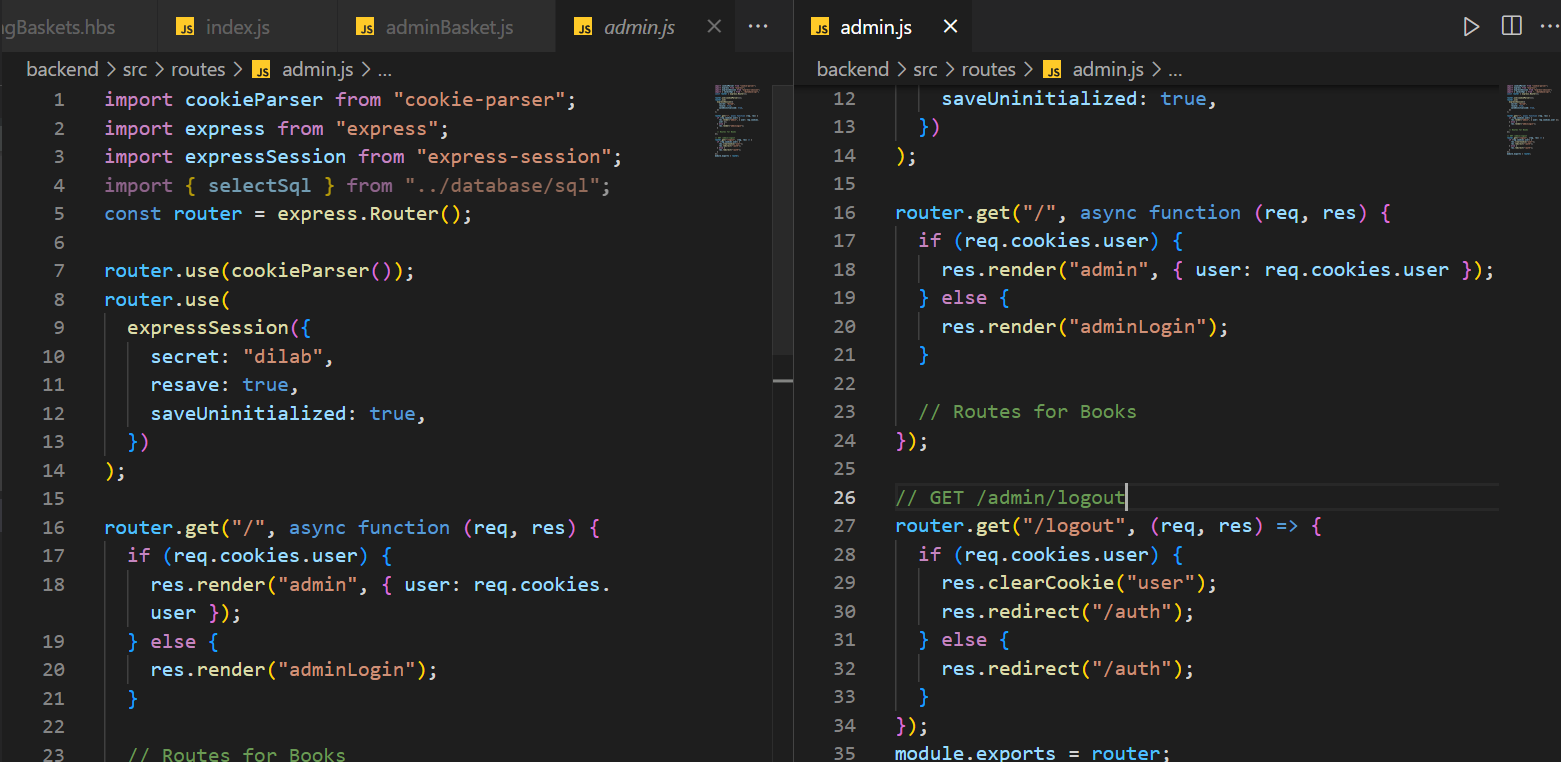


# 6. Web Application Execution screens

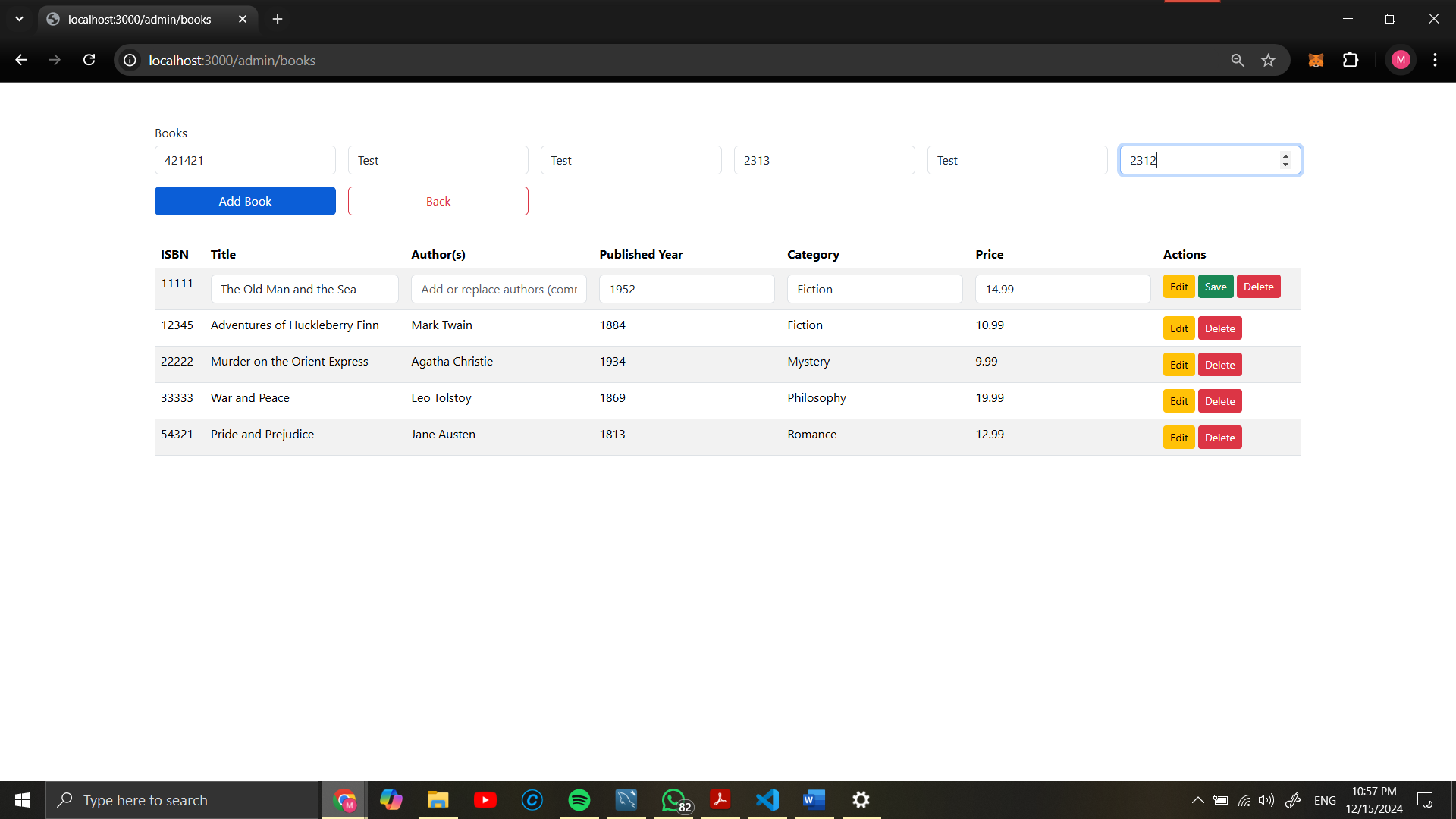
Admin Pages: Overview and screenshots of implemented functionalities.

1. **Main Admin Page**:
   * Central navigation hub for managing all sections.
   * Screenshot showcasing the layout and navigation features.

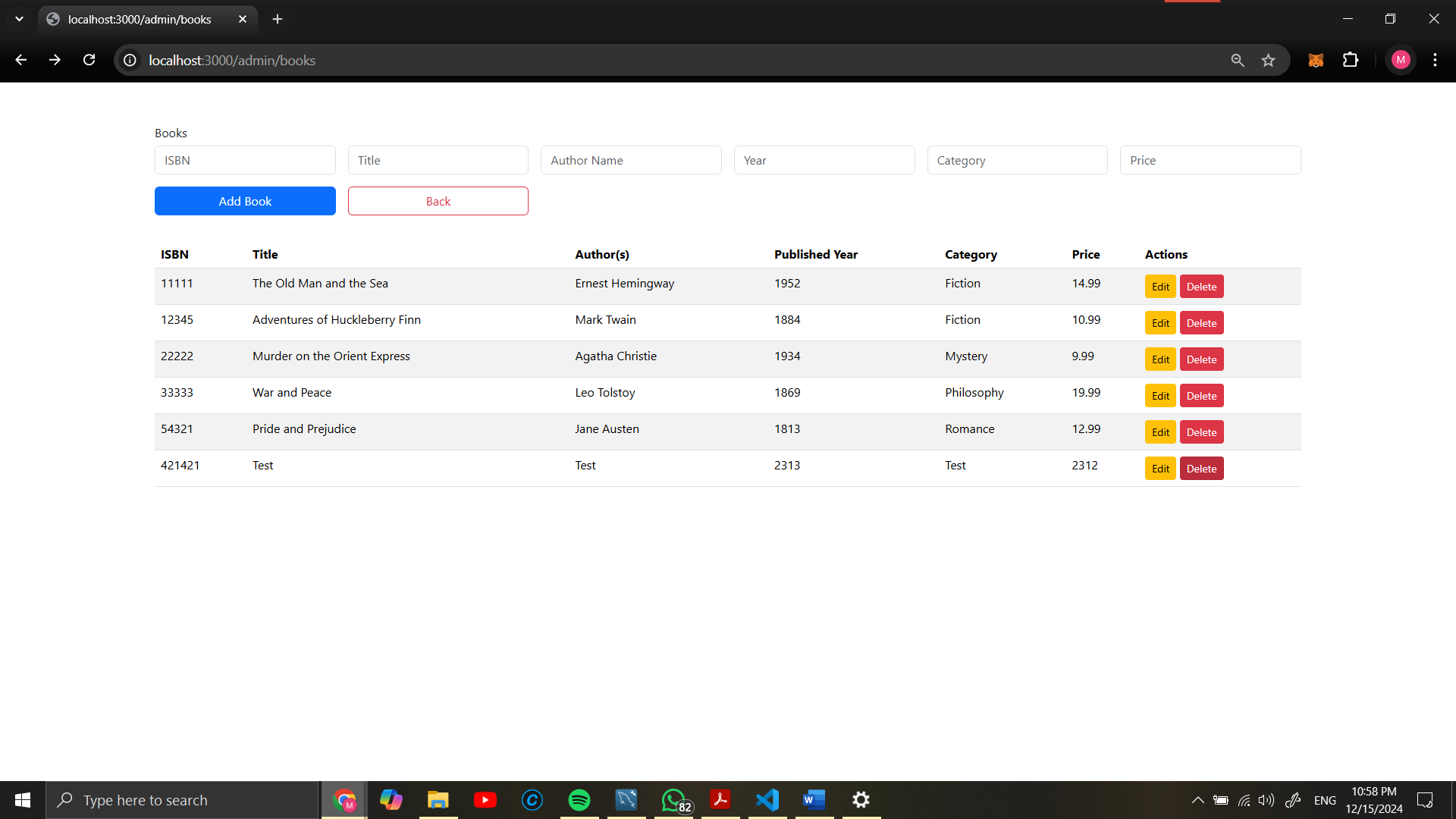




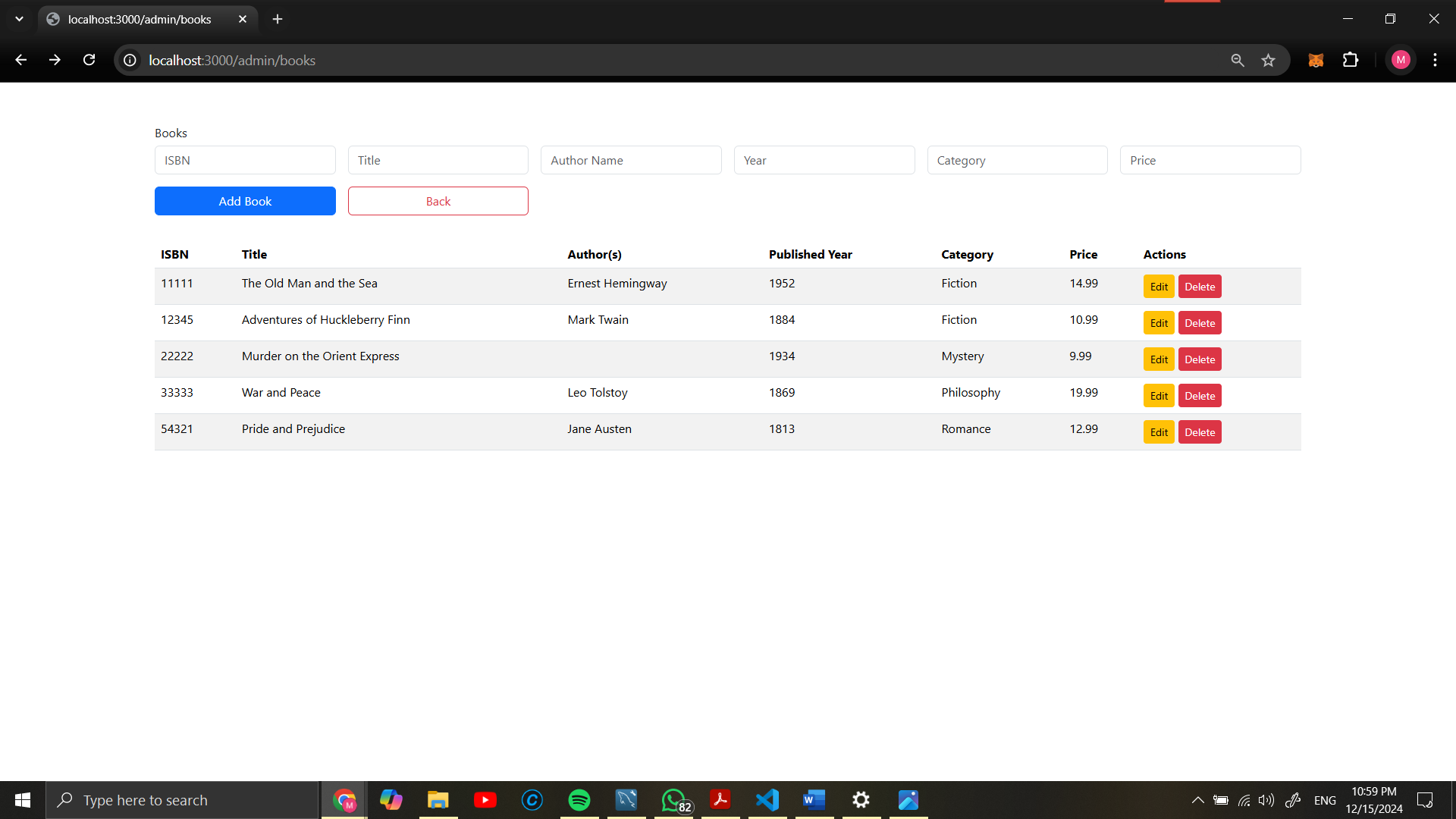
1. **Manage Books Page**:
   * Allows administrators to add, edit, and delete book records.
   * Screenshot of book management interface.



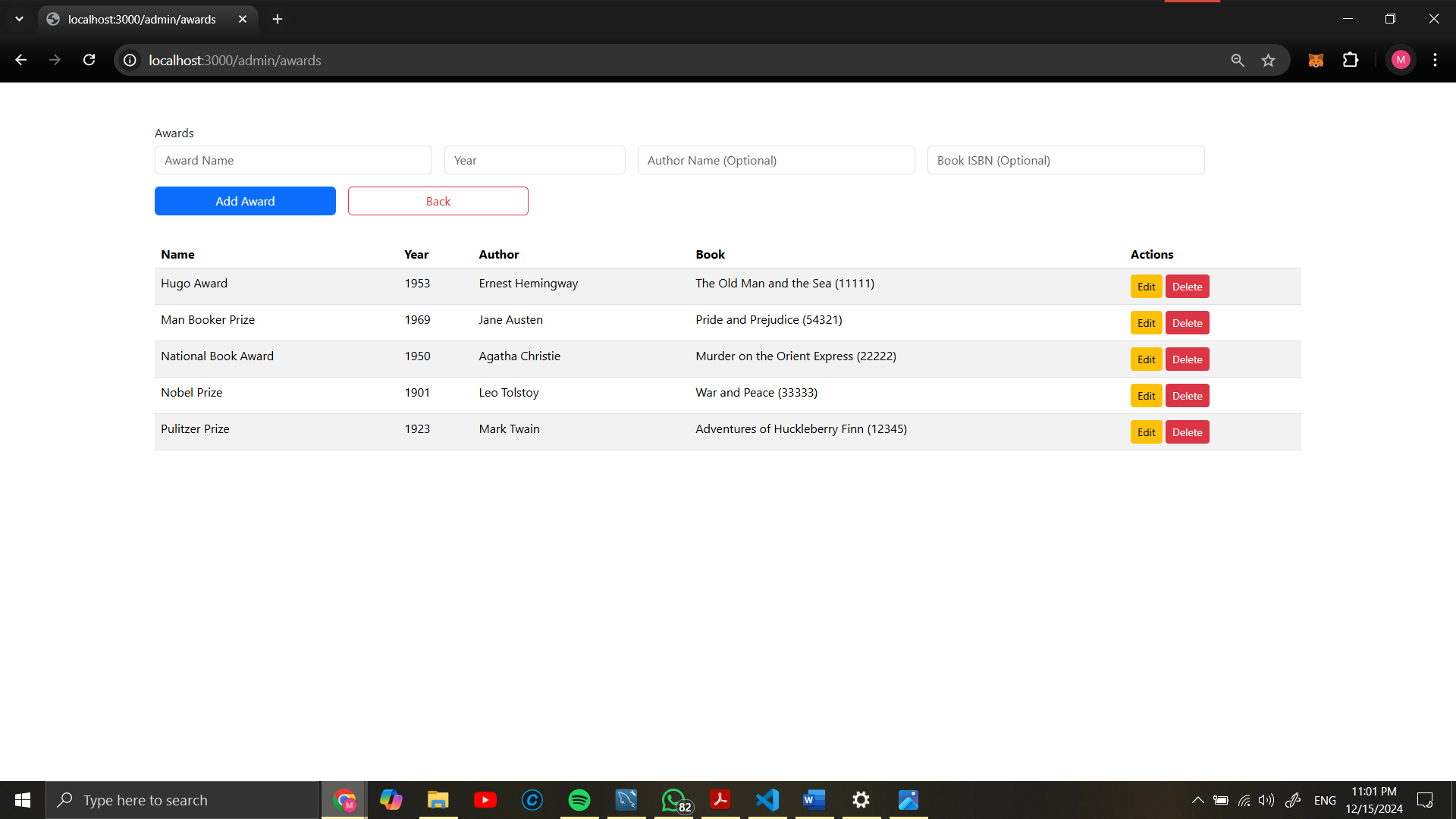
Test added



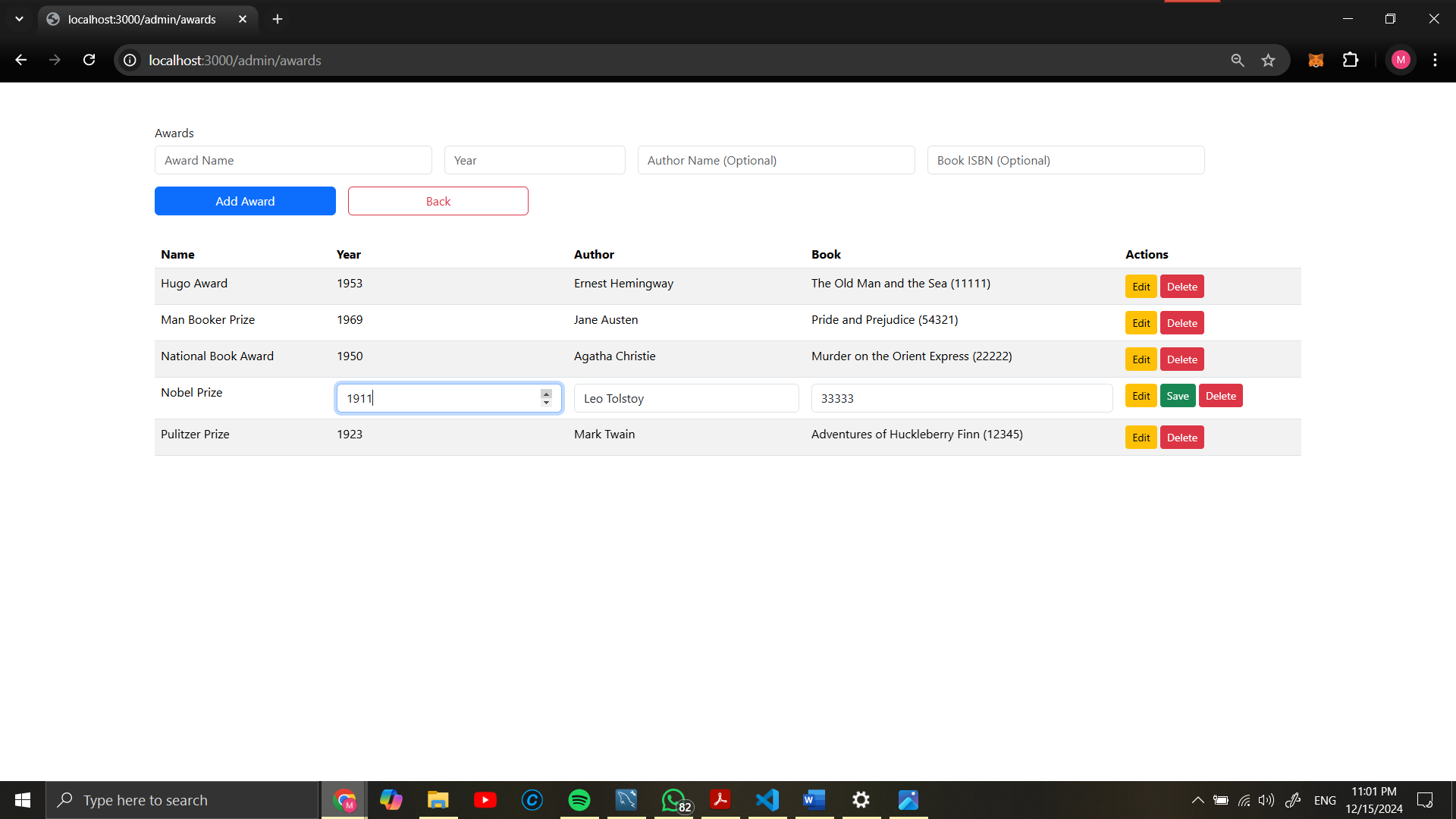
**Author of Murder on the Orient Express edited and Test book deleted:**

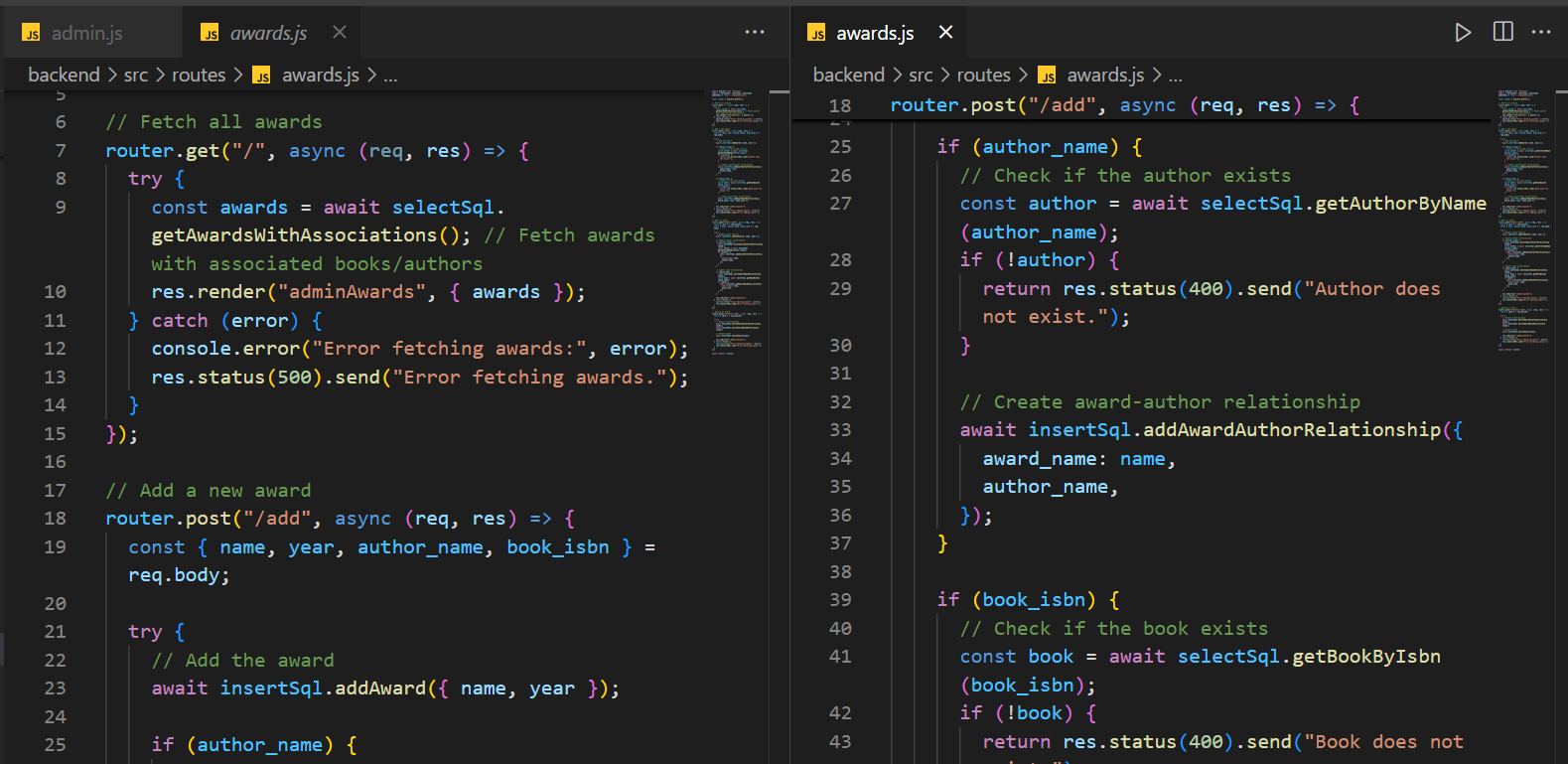


1. **Manage Awards Page**:
   * Manage award details and associate awards with books/authors.
   * Screenshot of award management page.



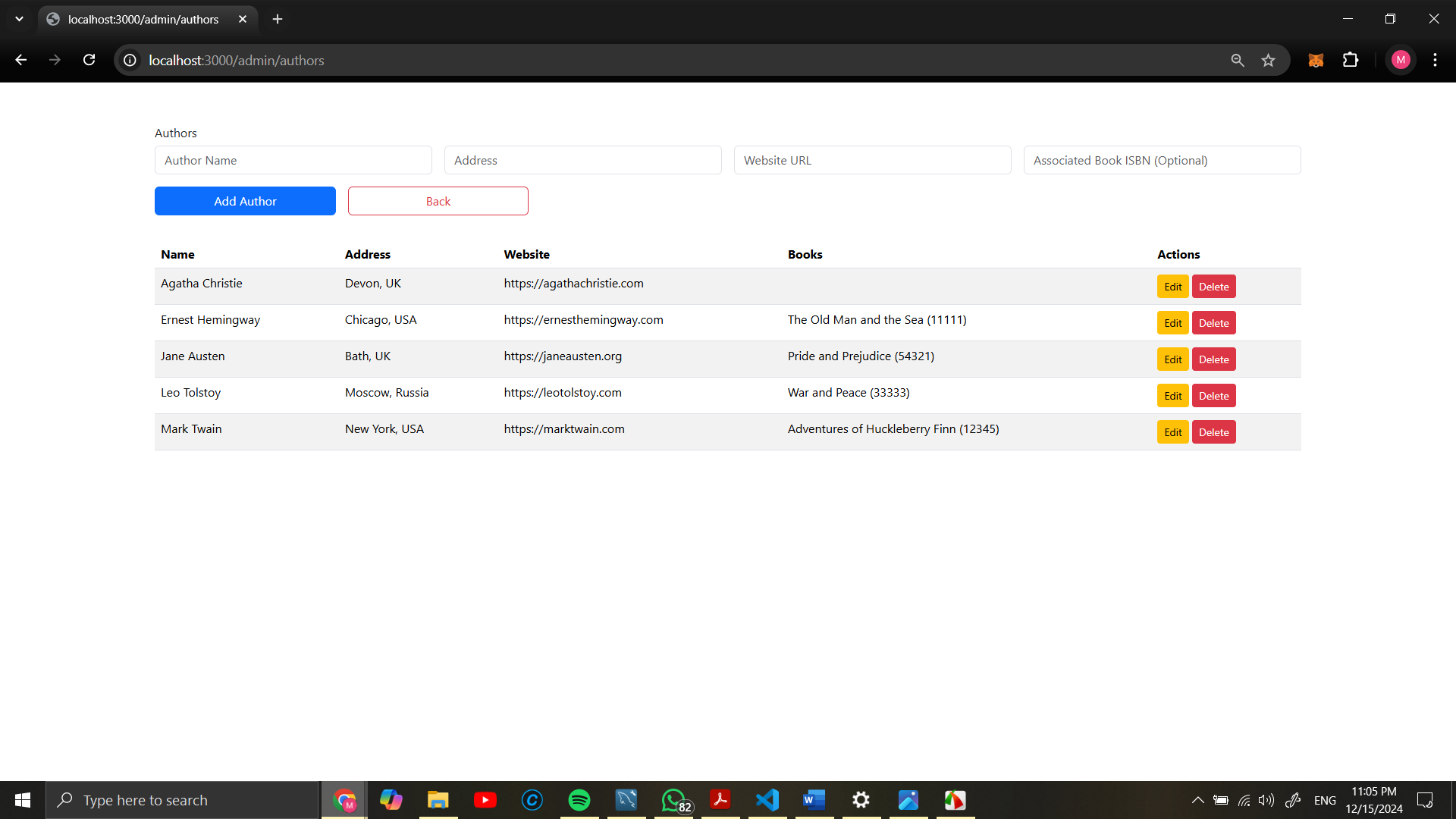
Same add, edit, delete functions are all works here as well.



**Award Management Process (this process is almost same for all of managing processes. Therefore, I will include it in report once.**

1. **Add Award**
   * **Process**:
     + Admin provides details such as name, year, author\_name, and book\_isbn through a form.
     + The POST request checks if the provided author\_name and book\_isbn exist in the database.
     + If valid:
       - The award is added to the award table.
       - Relationships (award-author, award-book) are established in their respective tables.
     + If invalid:
       - Returns an error message (e.g., "Author does not exist").
   * **Code Reference**: router.post("/add", ...) as seen in the image.
2. **Edit Award**
   * **Process**:
     + Admin updates award details via a form submission (e.g., name or year).
     + The system validates the provided inputs and updates the respective entries in the award table.
     + Relationships with author and book can also be updated as needed.
   * **Implementation**: Similar logic to the "add" process but with an UPDATE query on the database.
3. **Delete Award**
   * **Process**:
     + Admin selects an award to delete.
     + The DELETE query removes the award entry from the award table and cascades the deletion in related tables (e.g., award-author, award-book relationships).
   * **Validation**: Ensure related dependencies are handled appropriately to maintain referential integrity.
4. **Manage Authors Page**:
   * Input and manage author details.
   * Screenshot demonstrating author management functionality.



# 7. Locking

1. **Manage Shopping Baskets**:

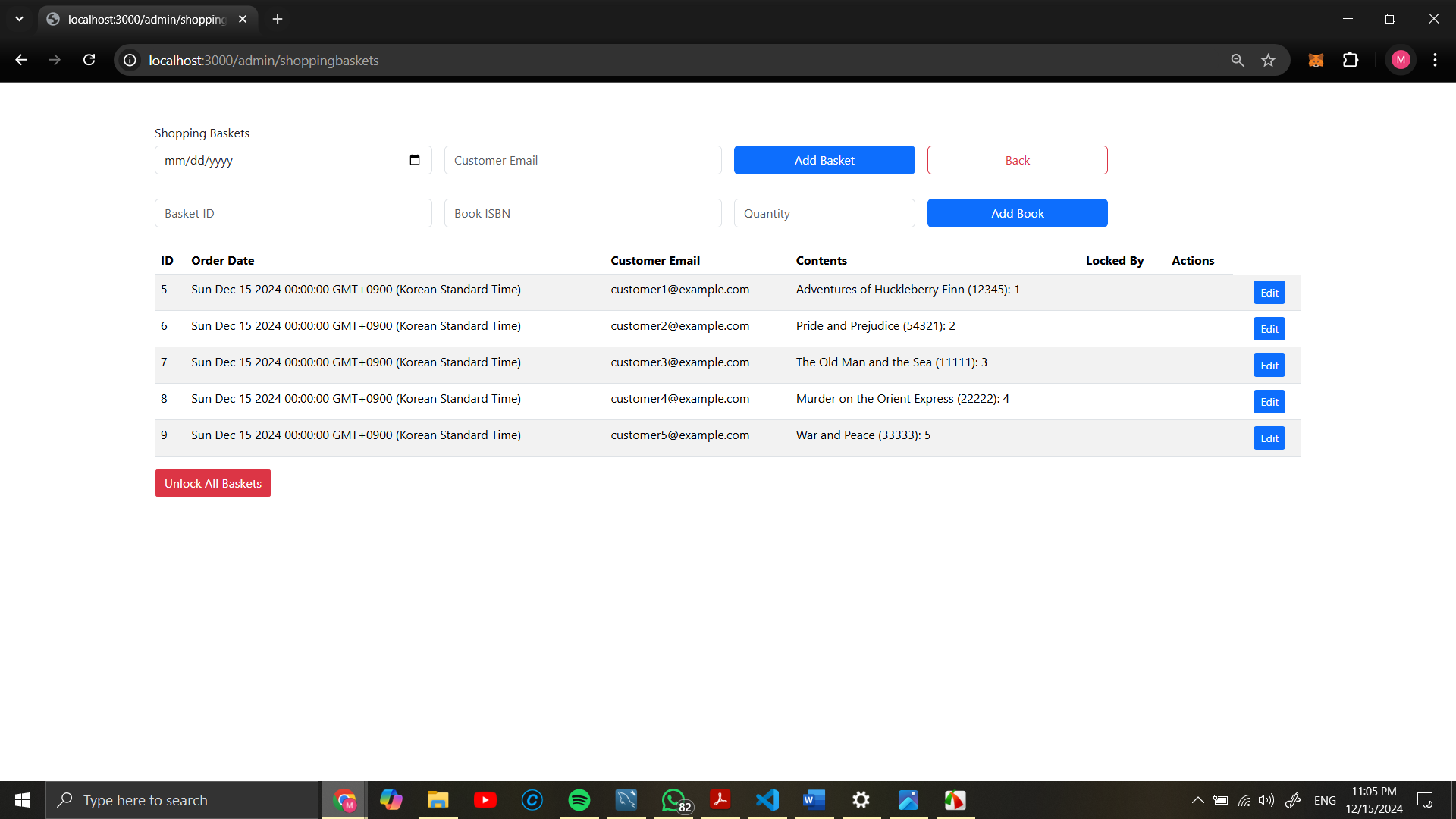
View and manage customer shopping basket contents.

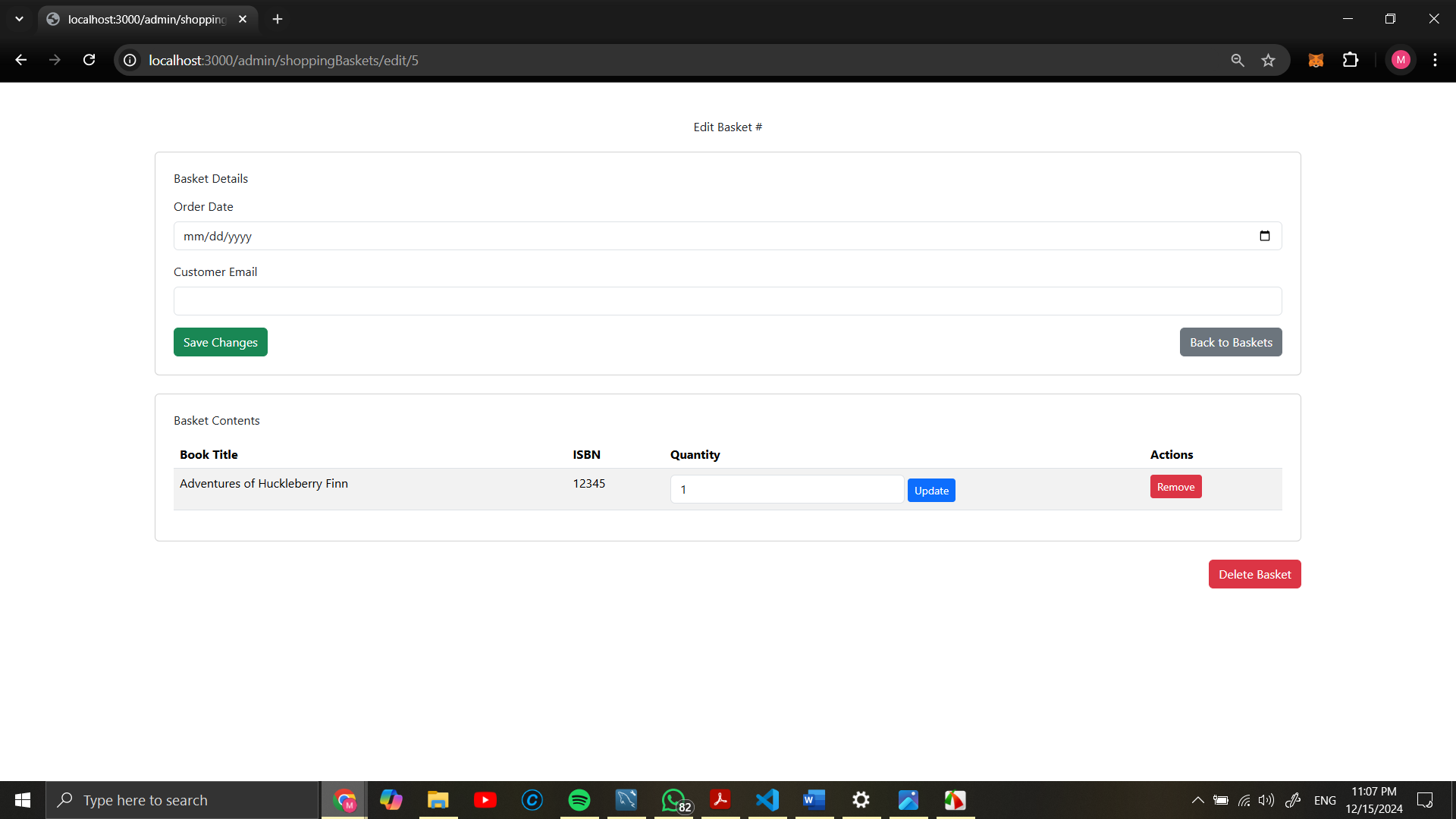
**Key Features**:

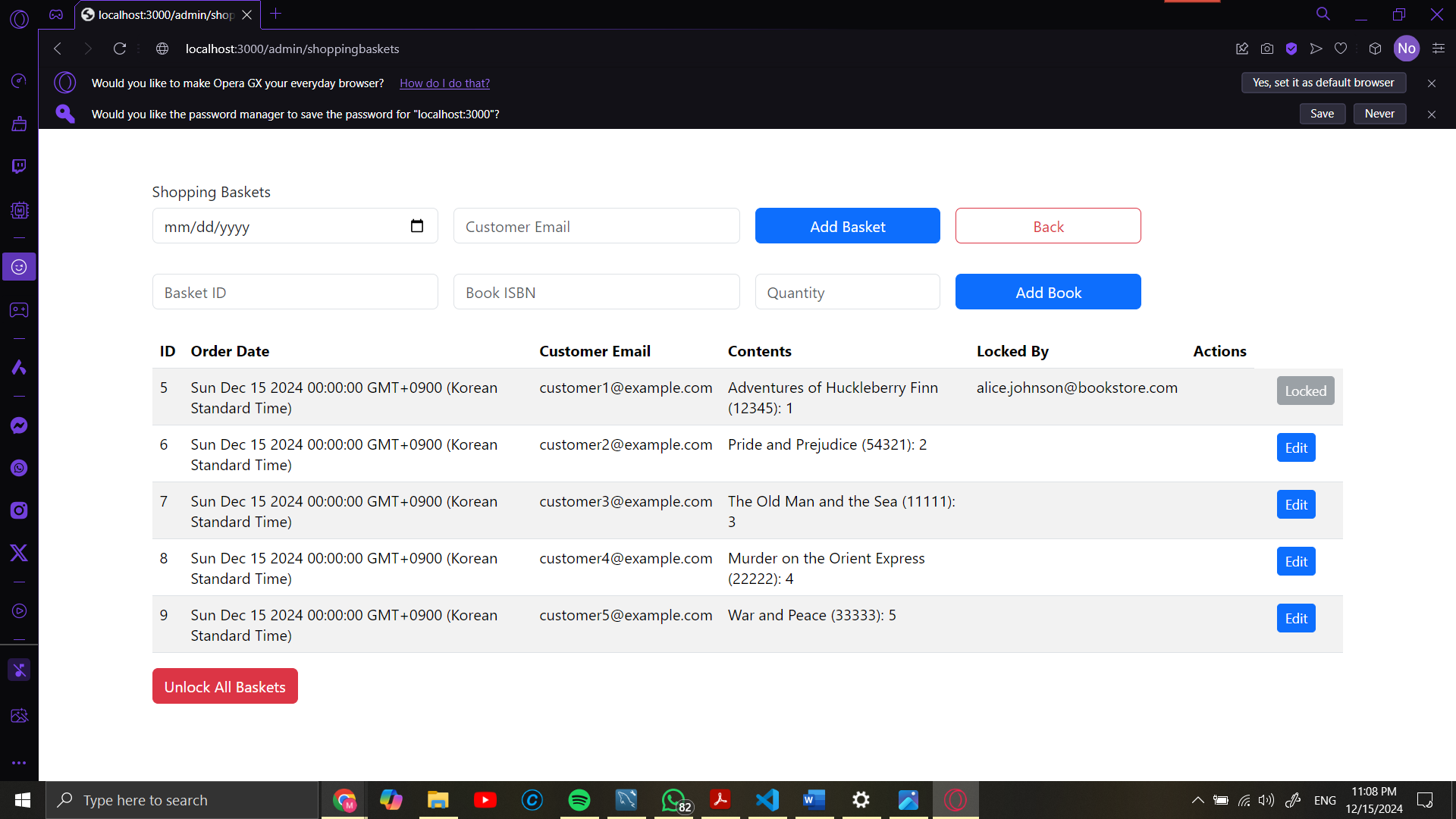
* + Locking mechanism ensures only one admin edits a basket at a time.
  + Admins can update order details and quantities.
  + Unlock all baskets feature for quick administrative recovery.

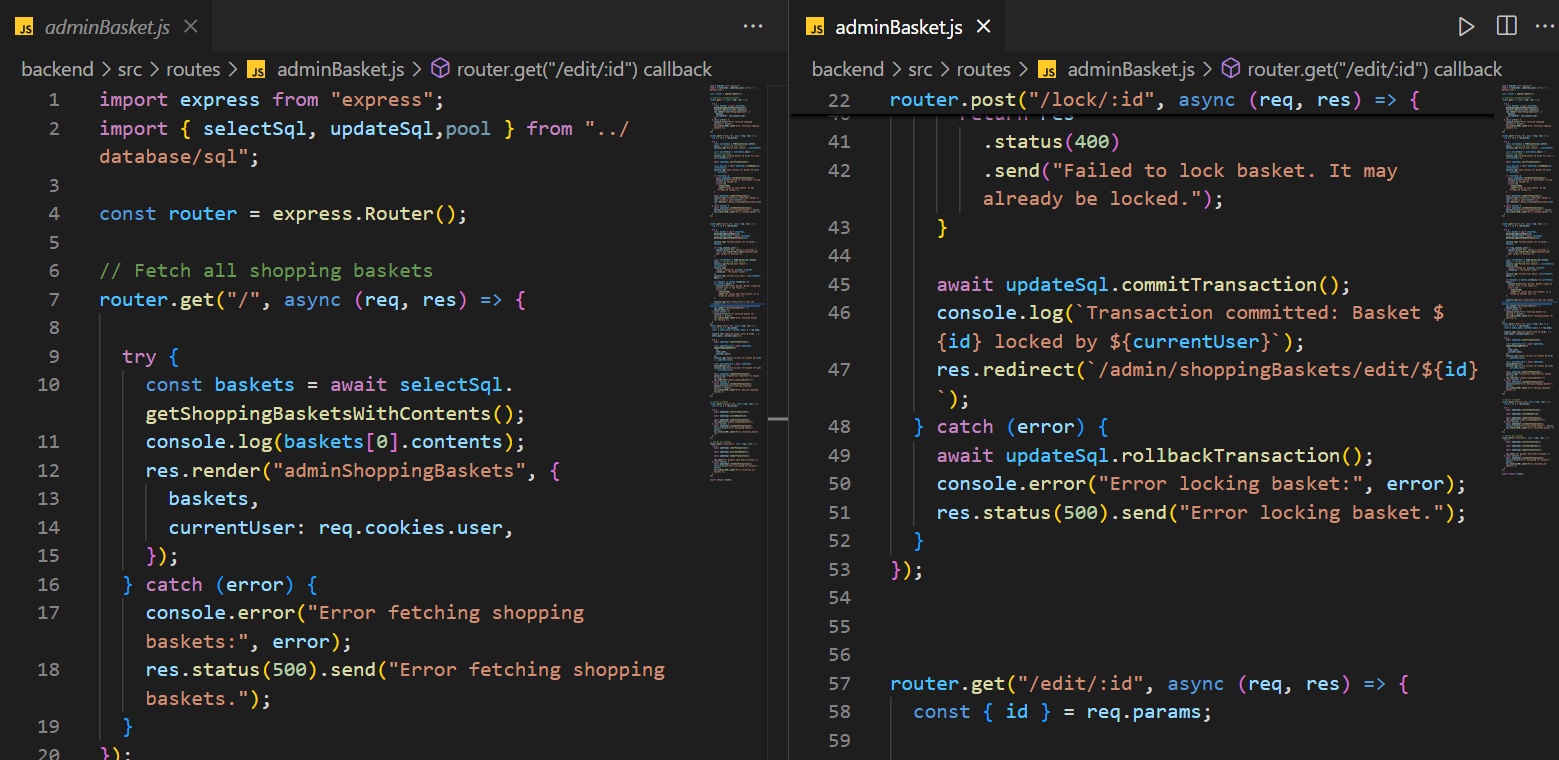
**Code Functionality**:

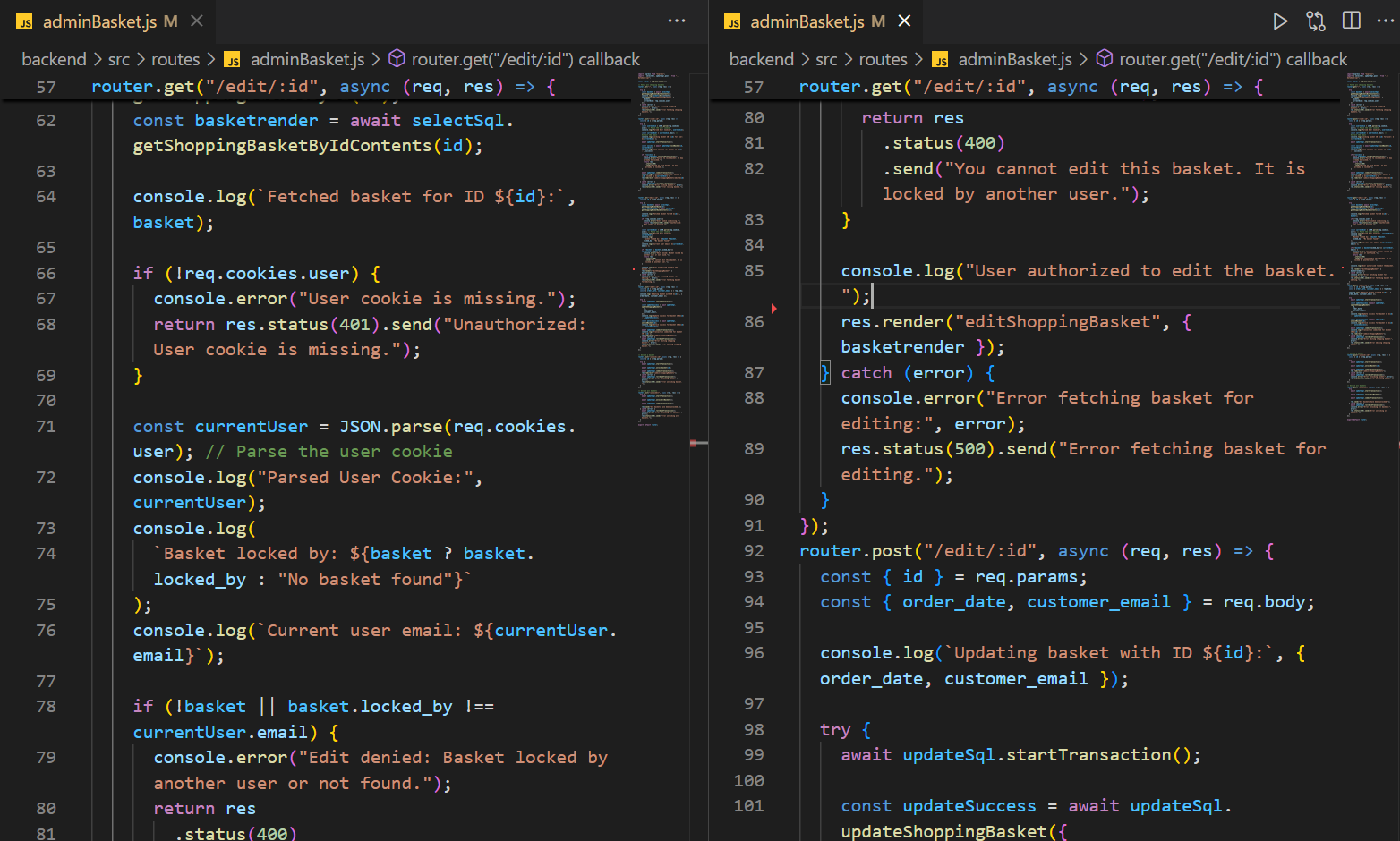
* + **Fetch Baskets**: Retrieves shopping basket data with associated book contents.
  + **Lock Basket**: Prevents simultaneous edits by locking a basket to the current admin.
  + **Edit Basket**: Allows updates to order date, customer email, and item quantities.
  + **Unlock Basket**: Releases locks for baskets after edits or errors.



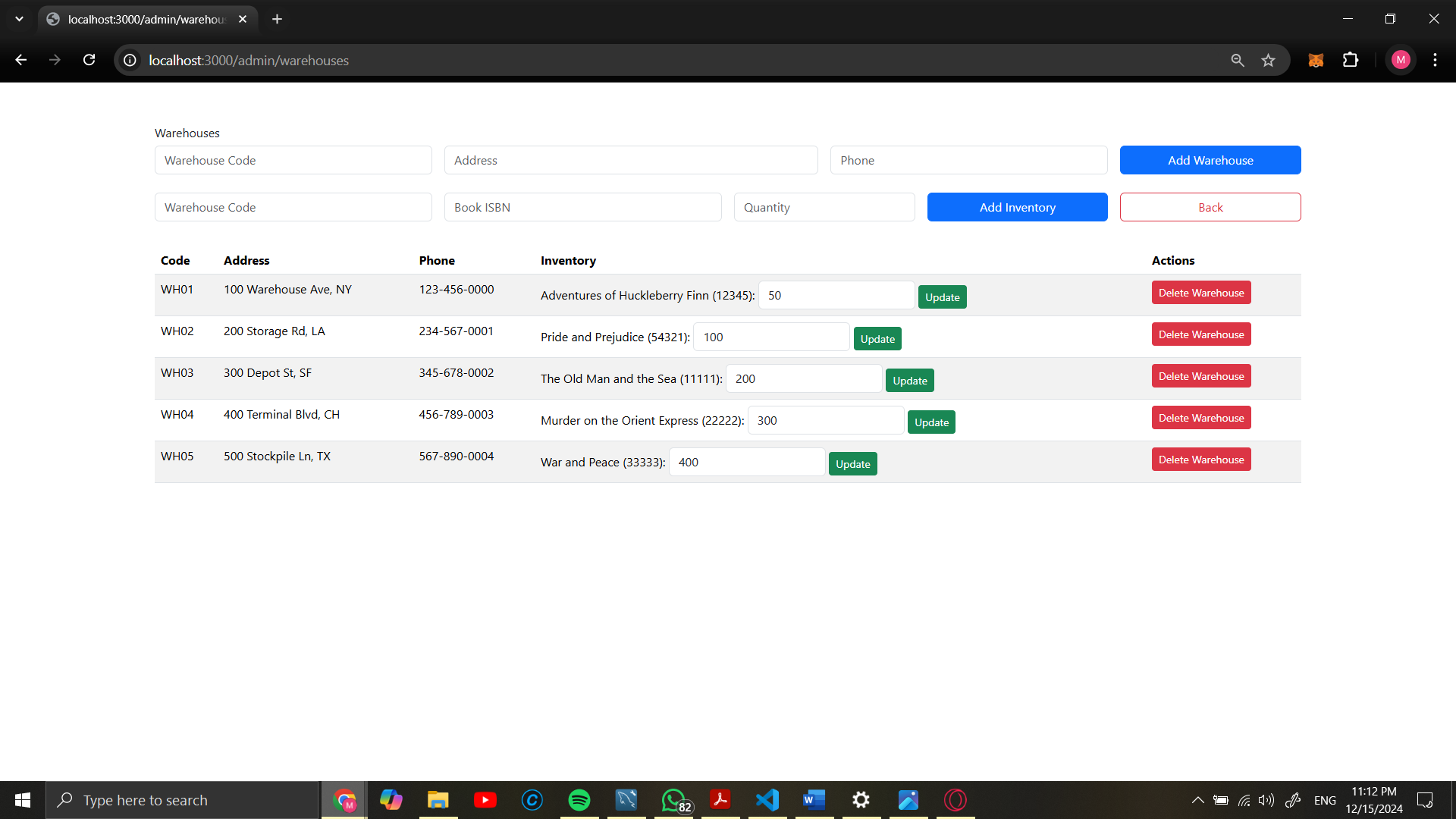






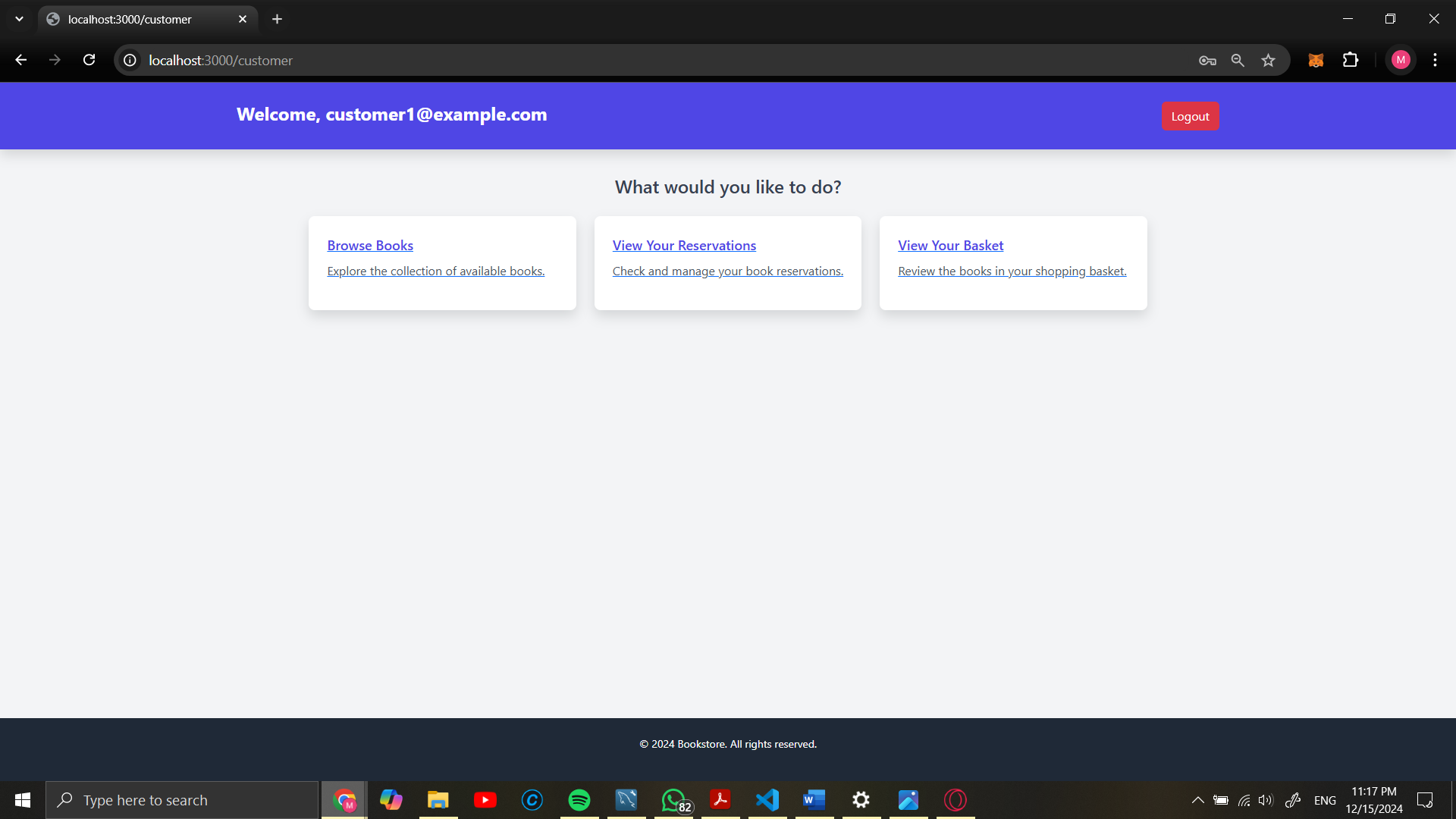


1. **Manage Warehouses Page**:
   * Track and manage warehouse inventory and locations.
   * Screenshot of warehouse management interface.



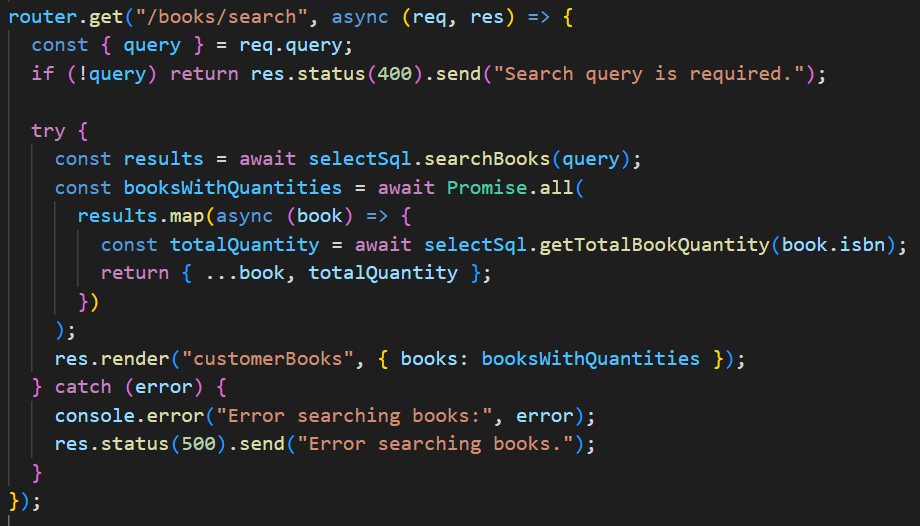
**Customer Pages**

1. **Overview and Screenshots of Implemented Functionalities:**
2. **Main Page for Navigation**:
   * Customer homepage with links to key features.
   * Screenshot showcasing the navigation options.

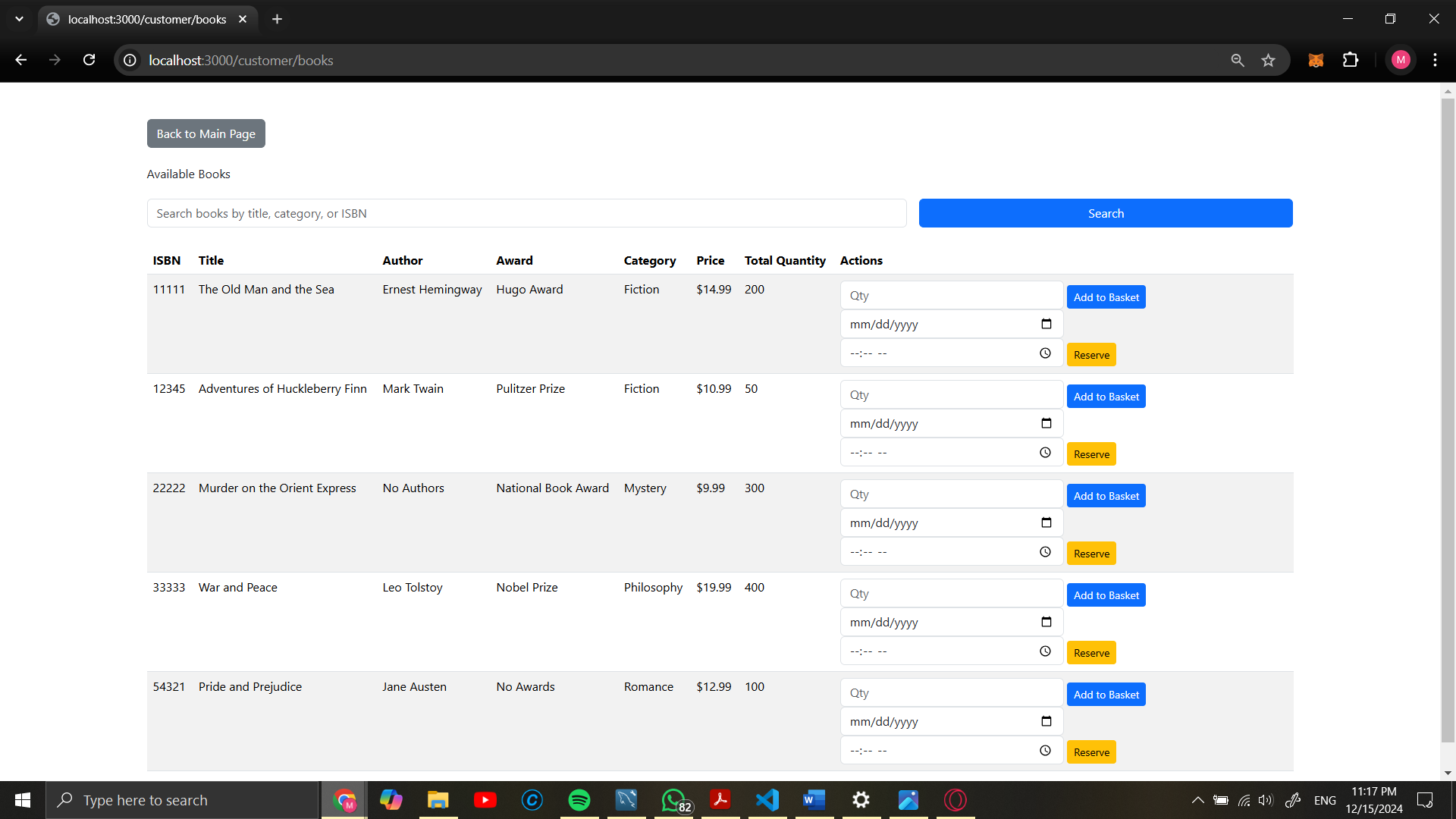


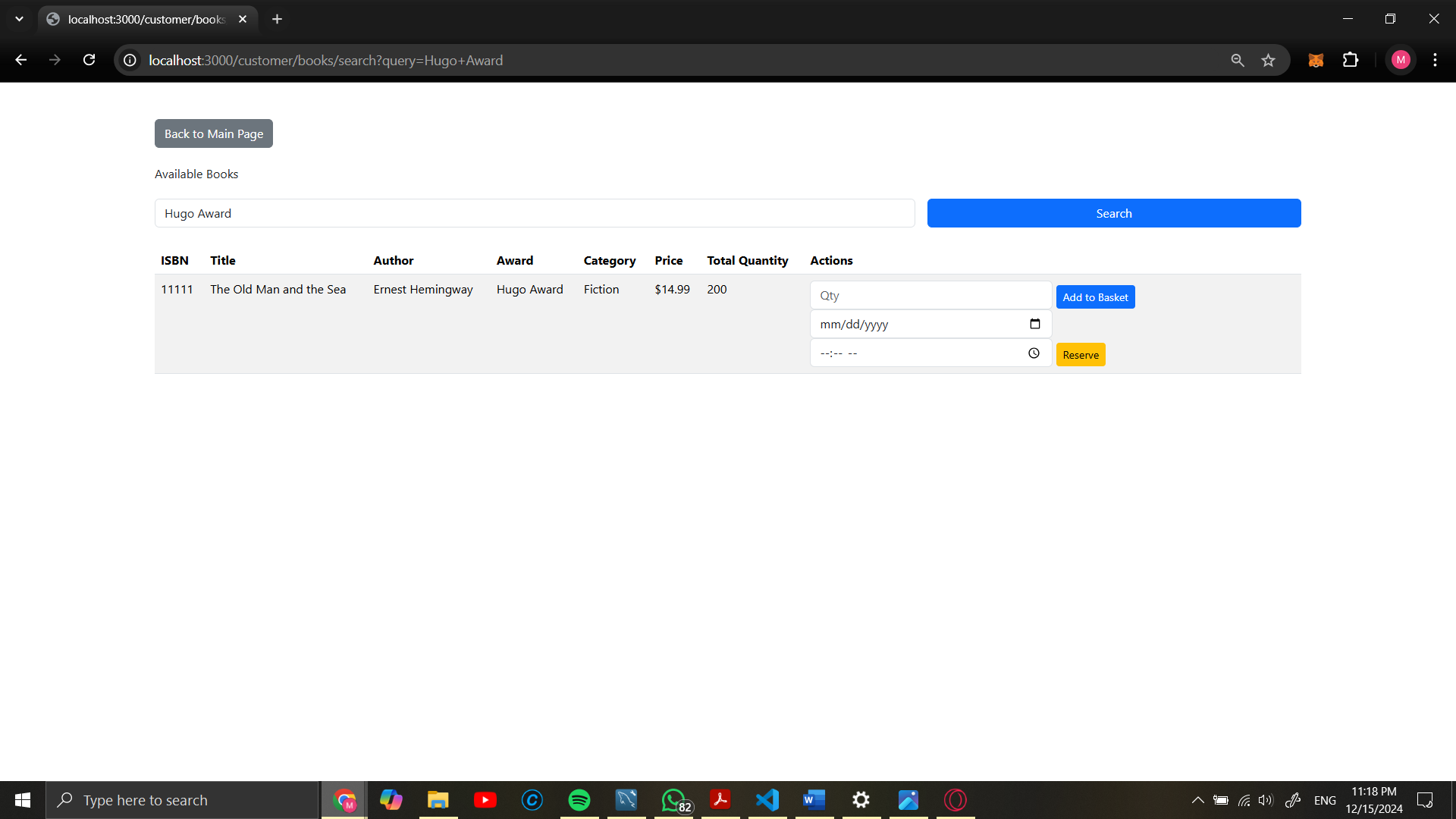
1. **Book Searching Page**:

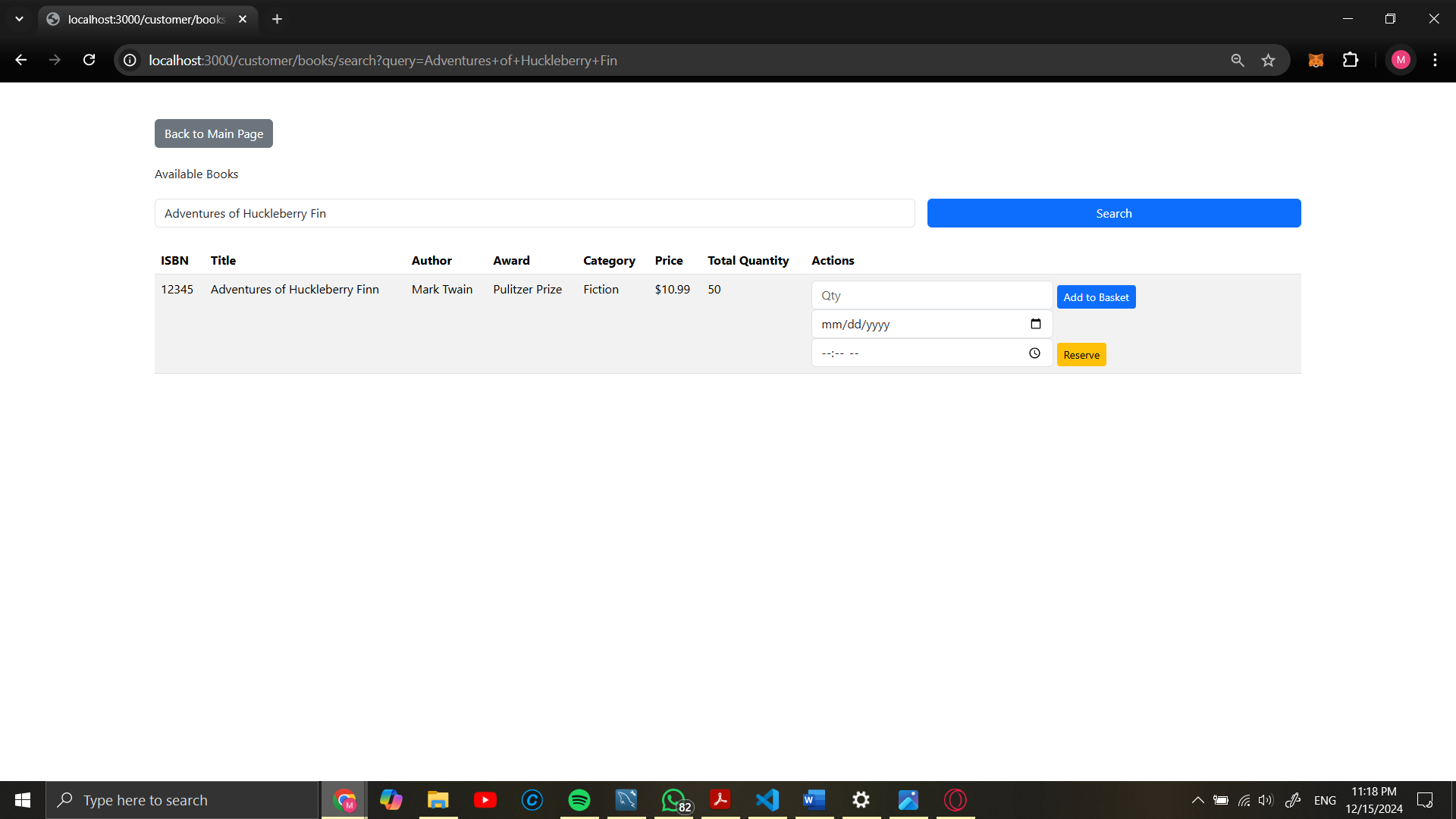
* Customers can search for books using keywords, including:
  + **Title**: Book titles are matched.
  + **Author**: Search results include books by specific authors.
  + **Award**: Filters books that have received awards.
* **Search Process**:
  + A GET request is triggered when a user enters a search query in the search bar.
  + The query is processed, and results are fetched from the database.
  + Results are enriched with total quantities aggregated from all warehouses to display real-time availability.
* **Interface**:
  + Users see search results in a tabular format.
  + For each book, availability, title, author, and awards are displayed.
  + If the book is in stock, users can add it to their basket or reserve it.
  + Out-of-stock books are indicated with a message.
* Screenshot showcasing search results.

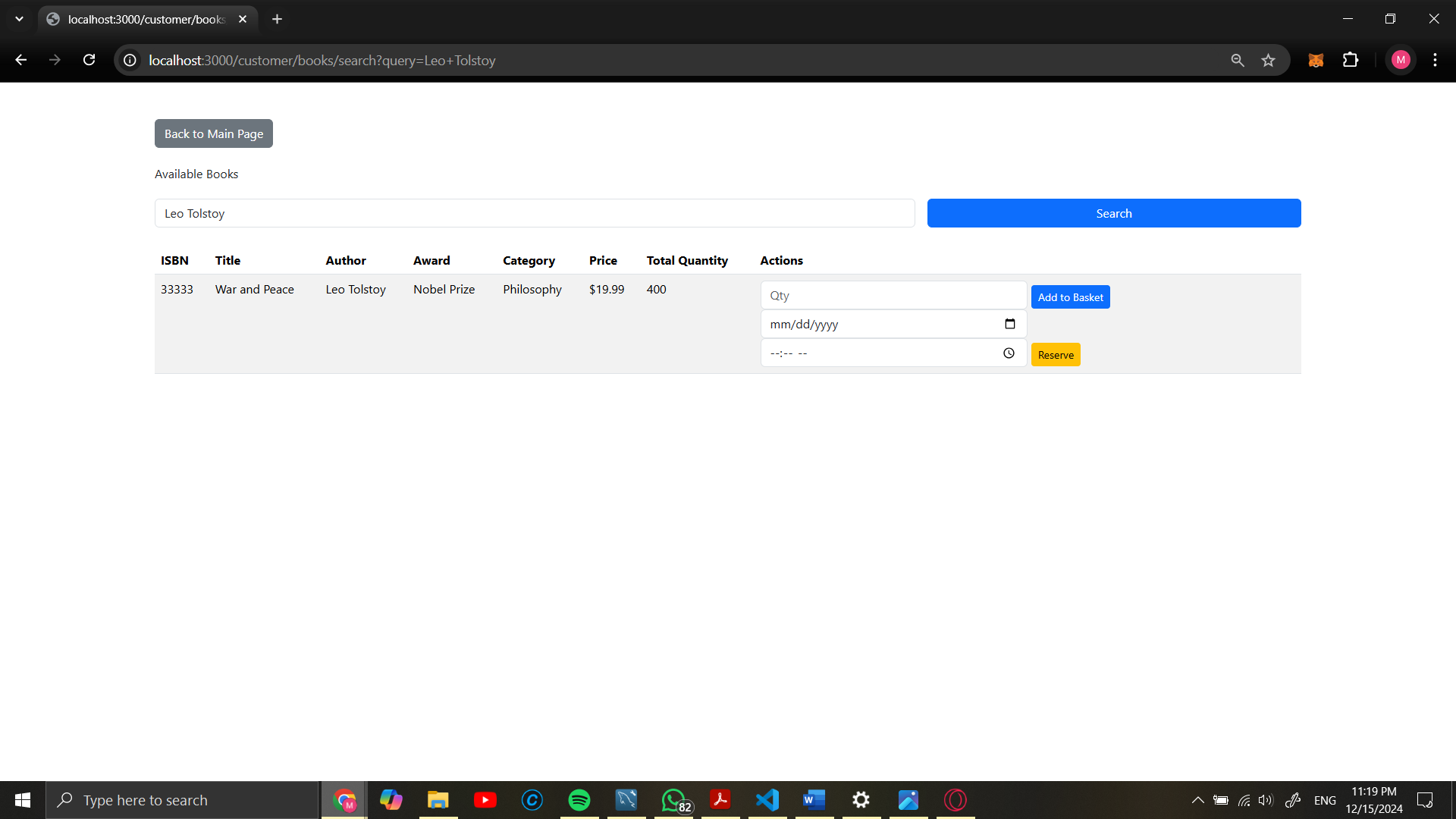


Screenshot displaying search results and filters.

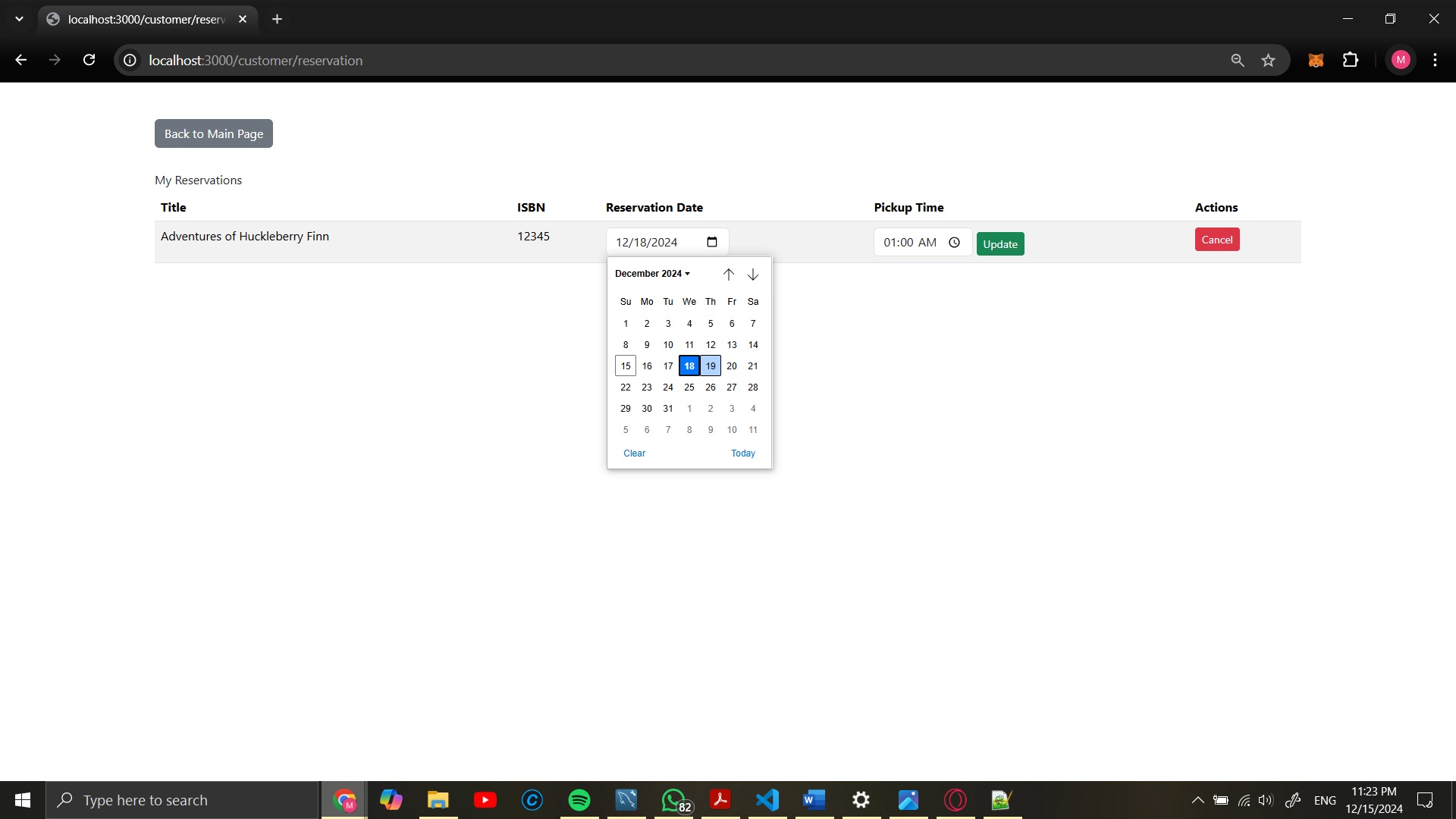




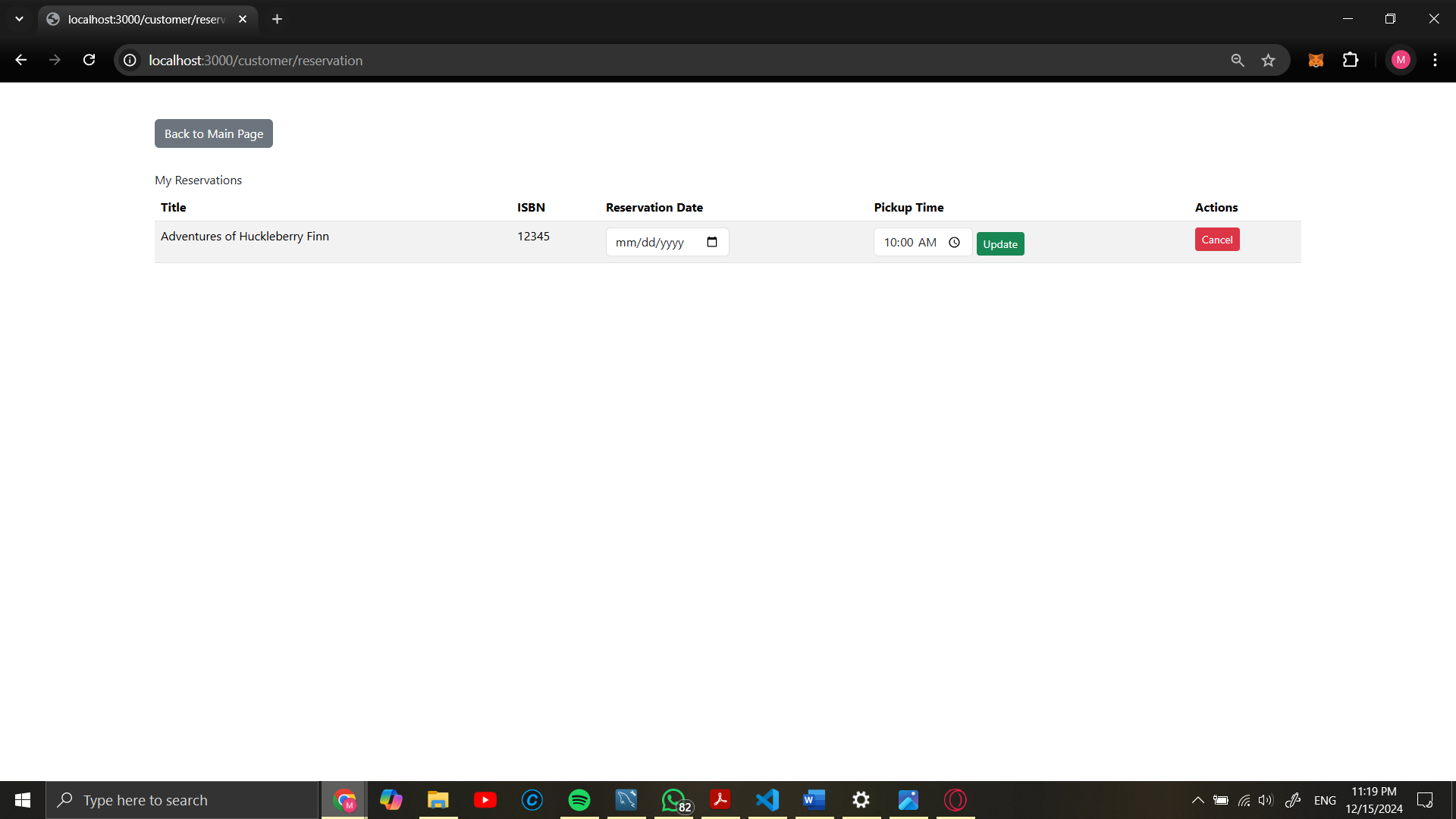


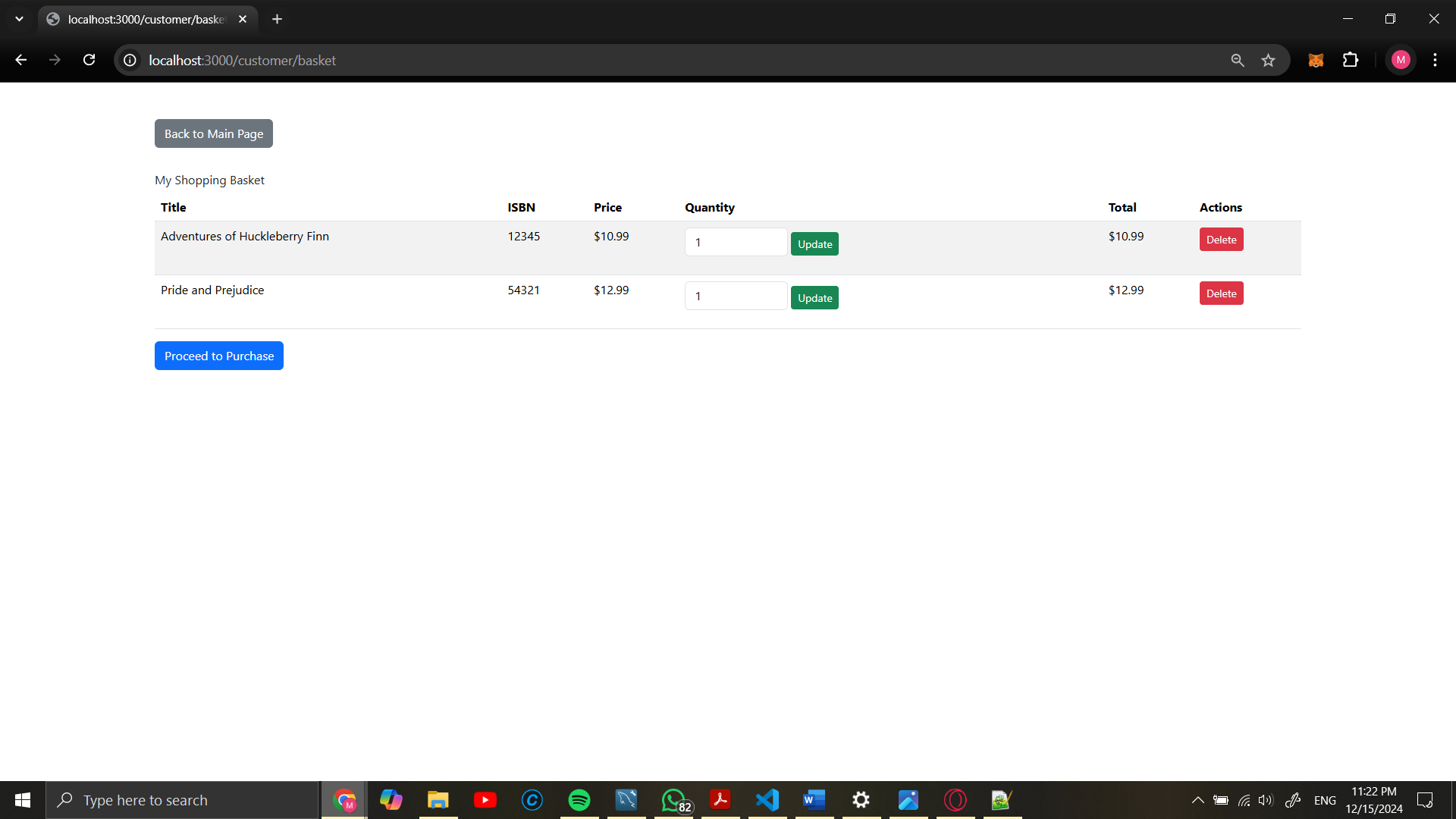


1. **Reservations Basket**:
   * View, create, modify, or cancel book reservations.
   * Screenshot showing reservation management.



1. **Shopping Basket**:
   * Add, view, edit, or purchase books.
   * Screenshot demonstrating the shopping basket interface.





# Discussion

The bookstore database project successfully demonstrates the design and implementation of a robust database-driven web application. By addressing critical functionalities for administrators and customers, the system ensures efficient data management, real-time inventory tracking, and seamless user interaction. Features such as book searching, reservations, and shopping baskets provide a comprehensive solution for digital bookstore operations.

Through the application of normalization techniques, proper indexing, and transactional integrity, the database design guarantees optimized performance, data accuracy, and reliability. The project showcases effective use of modern technologies, including Node.js, MySQL, and Handlebars.js, to deliver a user-friendly interface and a scalable backend system.

Future enhancements can include advanced features such as personalized recommendations, automated email notifications, and an enhanced reporting dashboard for admins to analyze sales and inventory trends. Overall, this project serves as a strong foundation for a fully functional and scalable online bookstore system.