Department of Computer Science & Engineering

Database Management System Lab

UE23CS351A

ORANGE PROBLEM 2 REPORT

Name	Yash Michael Tharappan	Yashvardhan Singh
SRN	PES1UG23CS713	PES1UG23CS716
Semester	5	5
Section	L	L
Campus	RR	RR
Department	CSE	CSE

Date: 18/11/2025

1. Abstract (Short Description)

This project implements a full-stack, web-based bookstore application. The system is designed to provide a seamless user experience for customers and a functional management interface for administrators. The primary objective is to build a deployable application that works seamlessly with a relational database, focusing on robust data management and business logic.

The application supports essential e-commerce functionalities:

- **Customer Management:** Secure user registration and login using JWT and password hashing.
- **Product Catalog:** A searchable and filterable book catalog.
- **Shopping Cart:** A persistent, user-specific shopping cart.
- **Order Processing:** A checkout system that uses stored procedures to ensure transactional integrity.
- **Reviews & Ratings:** A system for users to submit reviews, which automatically updates book ratings via database triggers/procedures.
- **Admin Dashboard:** A GUI for administrators to perform all **CRUD (Create, Read, Update, Delete)** operations on the book inventory.

The backend is built with **Node.js, Express, and TypeScript**, connecting to a **MySQL** database. The frontend is a responsive single-page application built with **React and TypeScript**.

2. User Requirement Specification (URS)

Purpose

The purpose of this project is to design and build a complete, deployable database application for an online bookstore. The application demonstrates the practical use of a relational database to manage customers, inventory, and orders in a real-world scenario.

Scope

The application manages three core entities: **Customers**, **Books**, and **Orders**.

- **Public users** can browse and search for books.
- **Authenticated customers** can manage their personal information, build a shopping cart, check out, view their order history, and post reviews.
- **Administrators** can log in to a separate dashboard to manage the book inventory (add, update, and delete books).

Detailed Description

The application functions as a full-service e-commerce site for books.

1. **Authentication:** New users can register with a name, email, address, and password. The system hashes the password (using bcryptjs) before storing it. Existing users can log in to receive a JSON Web Token (JWT), which is used to authorize all protected actions (like adding to cart or checking out).
2. **Book Catalog:** All users can view the list of books, which are pulled from the Books table. The interface (frontend/src/pages/Books.tsx) allows users to search by title/author and filter by genre.
3. **Book Details & Reviews:** Clicking a book shows a detailed page (frontend/src/pages/BookDetail.tsx) with its description, price, stock, and all customer reviews from the Reviews table (joined with the Customer table to show names). Logged-in users can submit a new review (rating and comments) from this page.
4. **Shopping Cart:** Authenticated users have a cart. When a user adds a book, the system makes an "UPSERT" (INSERT...ON DUPLICATE KEY UPDATE) call to the Cart_Items table, linking their Customer_ID to the Book_ID. Users can update quantities or remove items from the cart page (frontend/src/pages/CartPage.tsx).
5. **Checkout & Orders:** When a user proceeds to checkout (frontend/src/pages/CheckoutPage.tsx), they confirm their order. This action calls a dedicated backend endpoint (backend/src/routes/orders.ts) which invokes a stored procedure (sp_CreateOrderFromCart). This procedure is responsible for:
    - Calculating the total amount.
    - Creating a new entry in the Orders table.
    - Moving items from Cart_Items to Order_Items.
    - Decrementing the stock in the Books table.
    - Clearing the user's Cart_Items.
6. **Order History:** Users can view a list of all their past orders on the OrdersPage.tsx. This page features a complex query that joins Orders, Order_Items, and Books and uses JSON_ARRAYAGG to group all items belonging to each order.
7. **Admin Dashboard:** An admin user can access the dashboard (frontend/src/pages/AdminDashboard.tsx) to manage inventory. This interface provides a GUI for:
    - **Create:** A form to add a new book, including uploading a cover image.
    - **Read:** A grid view of all books in the Books table.
    - **Update:** An in-line form to edit a book's details (title, author, price, stock).
    - **Delete:** A button to remove a book from the Books table.

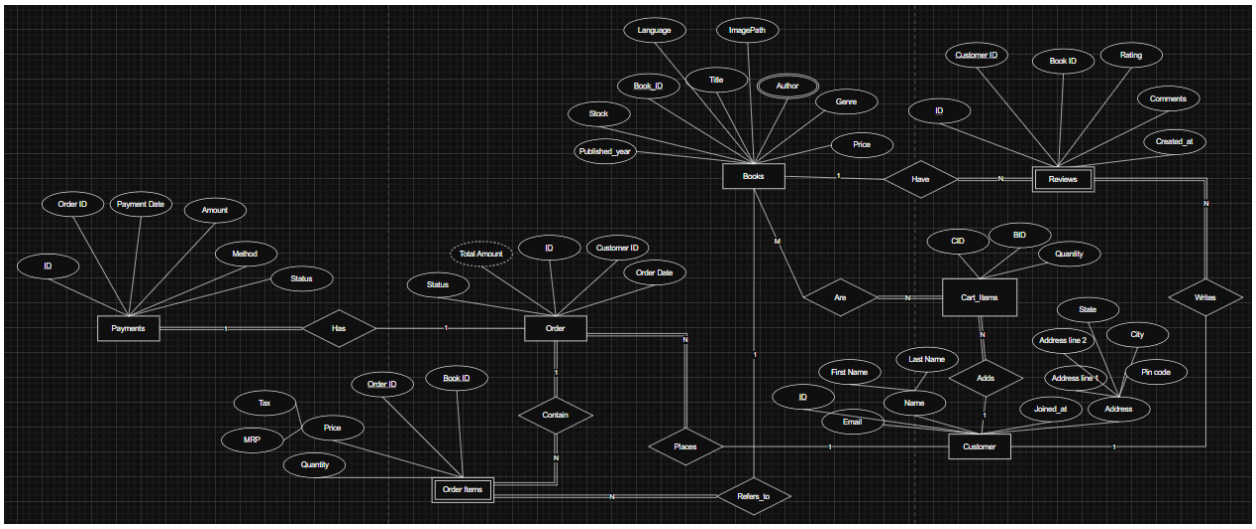
Functional Requirements

- **FR1 (Authentication):** System must allow new customers to register and existing customers to log in securely.
- **FR2 (Book Browsing):** System must display all available books with search and filter capabilities.
- **FR3 (Book Details):** System must display detailed information and all reviews for a single book.
- **FR4 (Cart Management):** System must allow authenticated users to add, update, and remove items from a persistent cart.
- **FR5 (Checkout):** System must use a stored procedure to transactionally convert a user's cart into a formal order.
- **FR6 (Order History):** System must display a list of all past orders and their details for a logged-in user.
- **FR7 (Review Submission):** System must allow authenticated users to post a rating and comment for a book.
- **FR8 (Rating Update):** System must automatically update a book's average rating when a new review is posted (via a trigger or procedure).
- **FR9 (Admin CRUD):** System must provide a GUI for administrators to Create, Read, Update, and Delete books from the inventory.

3. List of Softwares/Tools/Programming Languages

- **Backend:**
  - **Runtime:** Node.js
  - **Framework:** Express.js
  - **Language:** TypeScript
  - **Database:** MySQL
  - **Libraries:** mysql2 (DB Driver), jsonwebtoken (Auth), bcryptjs (Hashing), multer (File Uploads), zod (Validation), cors, dotenv
- **Frontend:**
  - **Framework:** React
  - **Language:** TypeScript
  - **Bundler:** Vite
  - **Libraries:** react-router-dom (Routing), axios (HTTP Client), react-context (State Management)
- **Development:**
  - nodemon (Live-reloading backend)
  - ts-node (Executing TypeScript)

4. ER Diagram



5. Relational Schema

6. DDL Commands

These are the Data Definition Language (DDL) commands used to create the database structure, extracted from SQL/Bookstore_details.sql.

```sql
create database bookstore;
use bookstore;

-- Table: Customer
-- Stores information about registered customers.
CREATE TABLE Customer (
    Customer_ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL UNIQUE,
    Address TEXT,
    Joined_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

-- (Auth.controller.ts implies this was added)
ALTER TABLE Customer ADD COLUMN Password_Hash VARCHAR(255) NOT NULL;

-- Table: Books
-- Stores details for each book available in the store.
CREATE TABLE Books (
    Book_ID INT PRIMARY KEY AUTO_INCREMENT,
    Title VARCHAR(255) NOT NULL,
    Author VARCHAR(255) NOT NULL,
    Genre VARCHAR(100),
    Published YEAR NOT NULL,
    Price DECIMAL(10, 2) NOT NULL CHECK (Price >= 0),
    Stock INT NOT NULL DEFAULT 0 CHECK (Stock >= 0)
);

ALTER TABLE Books
ADD Language VARCHAR(100) NOT NULL DEFAULT 'English';

-- (Implied by backend/src/routes/books.ts and frontend)
ALTER TABLE Books
ADD ImagePath VARCHAR(255);

-- (Implied by frontend/src/api/mock.ts and frontend/src/pages/BookDetail.tsx)
ALTER TABLE Books
```

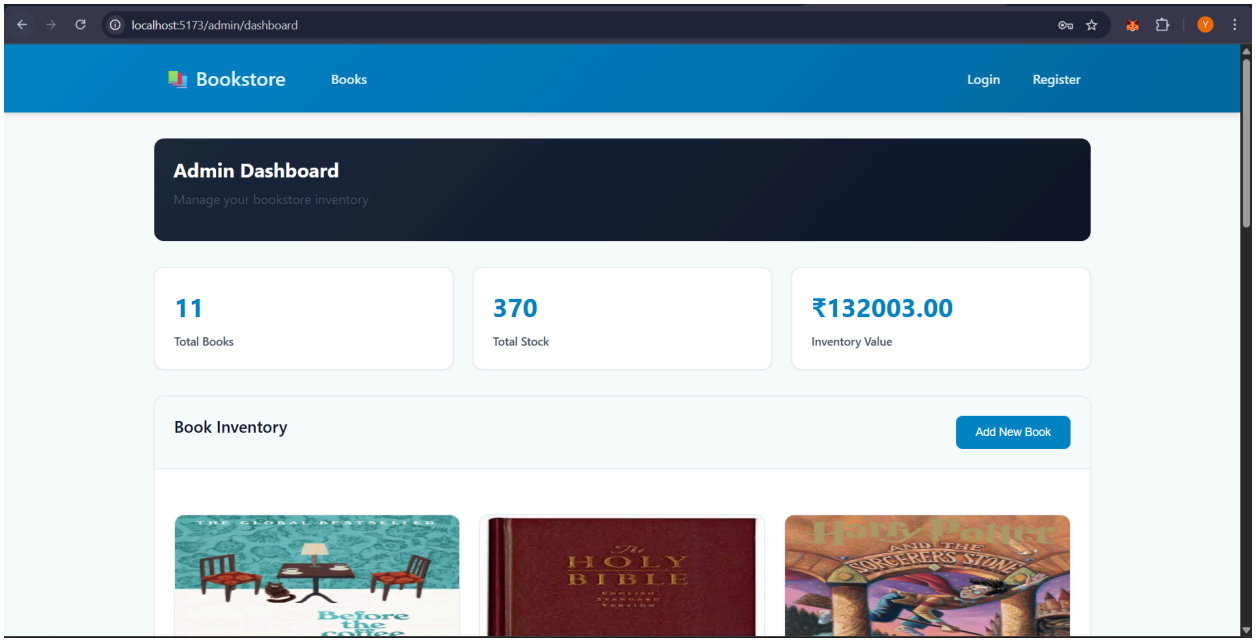
```sql
ADD Average_Rating DECIMAL(3, 2) DEFAULT 0.00;
ALTER TABLE Books
ADD Rating_Count INT DEFAULT 0;

-- Table: Orders
-- Stores the main information for each customer order.
CREATE TABLE Orders (
    Order_ID INT PRIMARY KEY AUTO_INCREMENT,
    CID INT NOT NULL,
    Order_Date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Total_Amount DECIMAL(10, 2) NOT NULL CHECK (Total_Amount >= 0),
    Status VARCHAR(50) NOT NULL DEFAULT 'Pending',
    FOREIGN KEY (CID) REFERENCES Customer(Customer_ID) ON DELETE RESTRICT
);

-- Table: Order_Items
-- A junction table that links books to specific orders
CREATE TABLE Order_Items (
    OID INT NOT NULL,
    BID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity > 0),
    Price DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (OID, BID),
    FOREIGN KEY (OID) REFERENCES Orders(Order_ID) ON DELETE CASCADE,
    FOREIGN KEY (BID) REFERENCES Books(Book_ID) ON DELETE RESTRICT
);

-- Table: Cart_Items
-- Stores details of user cart
CREATE TABLE Cart_Items (
    CID INT NOT NULL,
    BID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity > 0),
    PRIMARY KEY (CID, BID),
    FOREIGN KEY (CID) REFERENCES Customer(Customer_ID) ON DELETE CASCADE,
    FOREIGN KEY (BID) REFERENCES Books(Book_ID) ON DELETE CASCADE
);

-- Table: Reviews
-- Stores customer reviews and ratings for books.
CREATE TABLE Reviews (
    Review_ID INT PRIMARY KEY AUTO_INCREMENT,
    BID INT NOT NULL,
    CID INT NOT NULL,
```

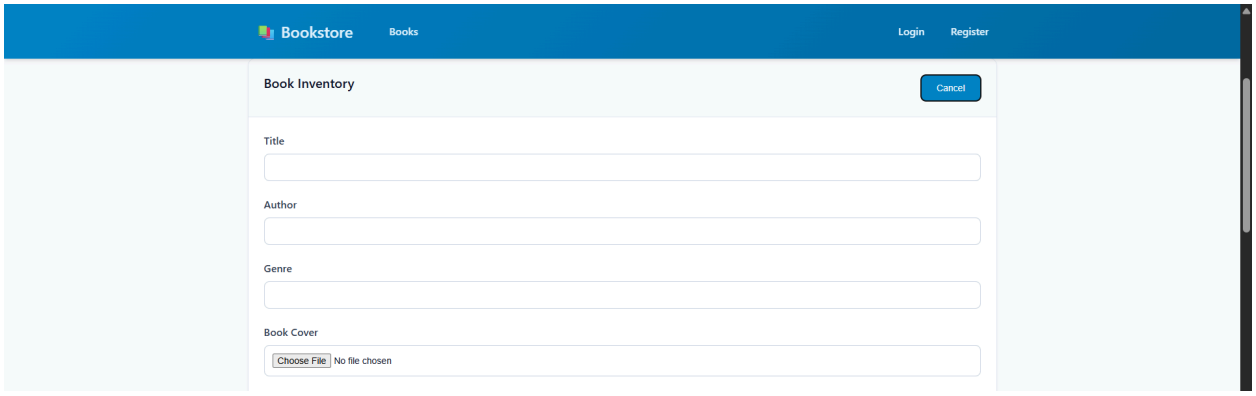
```
    Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5),
    Comments TEXT,
    Created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (BID) REFERENCES Books(Book_ID) ON DELETE CASCADE,
    FOREIGN KEY (CID) REFERENCES Customer(Customer_ID) ON DELETE CASCADE
);

-- Table: Payments
-- Stores payment details for each order.
CREATE TABLE Payments (
    Payment_ID INT PRIMARY KEY AUTO_INCREMENT,
    OID INT NOT NULL UNIQUE,
    Payment_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Amount DECIMAL(10, 2) NOT NULL,
    Method VARCHAR(50) NOT NULL,
    Status VARCHAR(50) NOT NULL DEFAULT 'Incomplete',
    FOREIGN KEY (OID) REFERENCES Orders(Order_ID) ON DELETE CASCADE
);
```

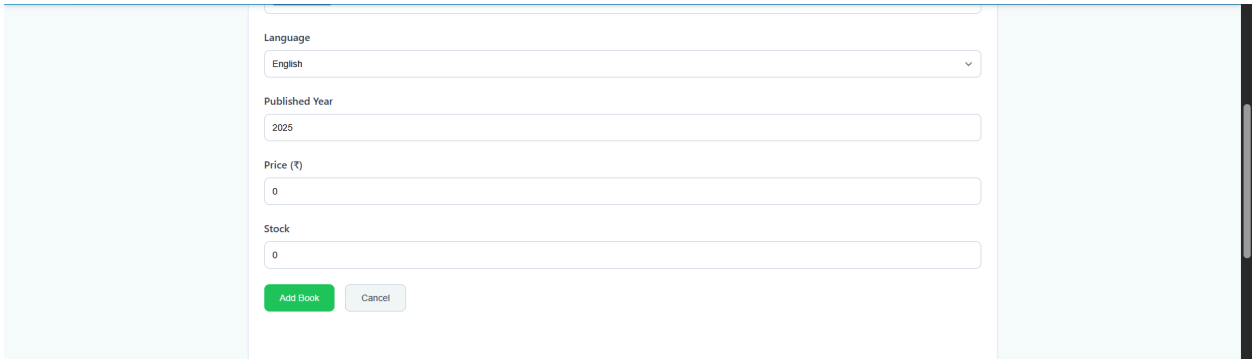
7. CRUD Operation Screenshots

This application implements all CRUD operations via a rich Graphical User Interface (GUI), primarily in the Admin Dashboard.

- **CREATE:** Screenshots of the **"Add New Book"** form in the Admin Dashboard (frontend/src/pages/AdminDashboard.tsx) are given below. This form includes fields for title, author, genre, image upload, price, and stock.

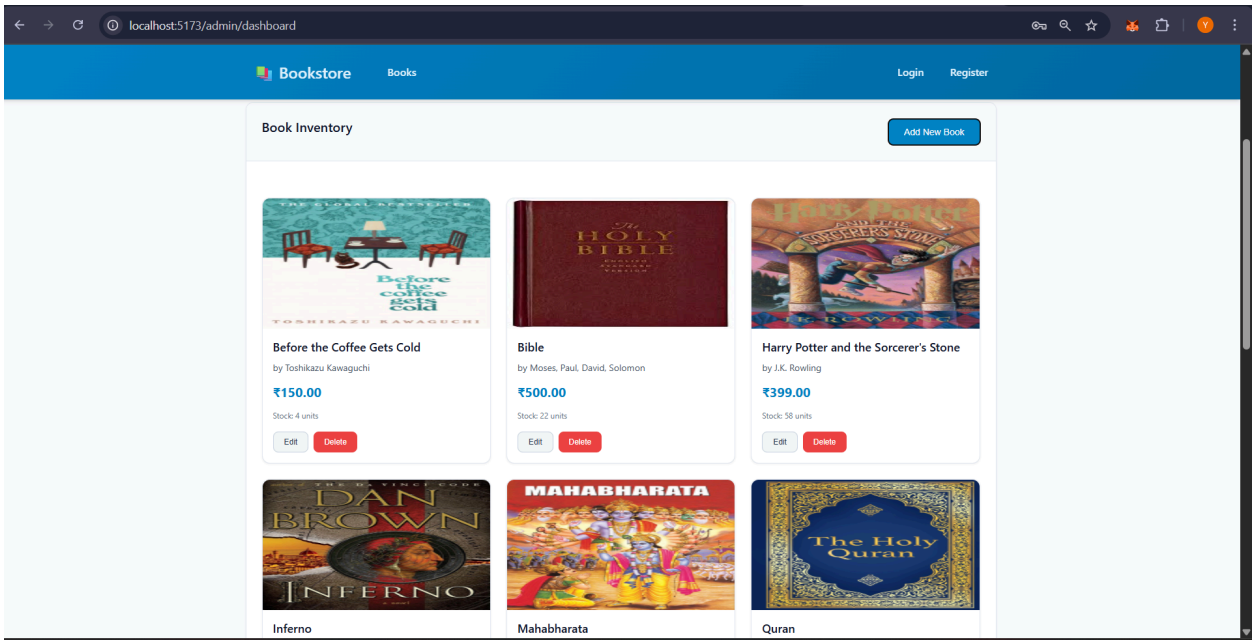

localhost:5173/admin/dashboard
Bookstore
Books
Login
Register
Admin Dashboard
Manage your bookstore inventory
11
Total Books
370
Total Stock
₹132003.00
Inventory Value
Book Inventory
Add New Book


Bookstore
Books
Login
Register
Book Inventory
Cancel
Title
Author
Genre
Book Cover
Choose File
No file chosen

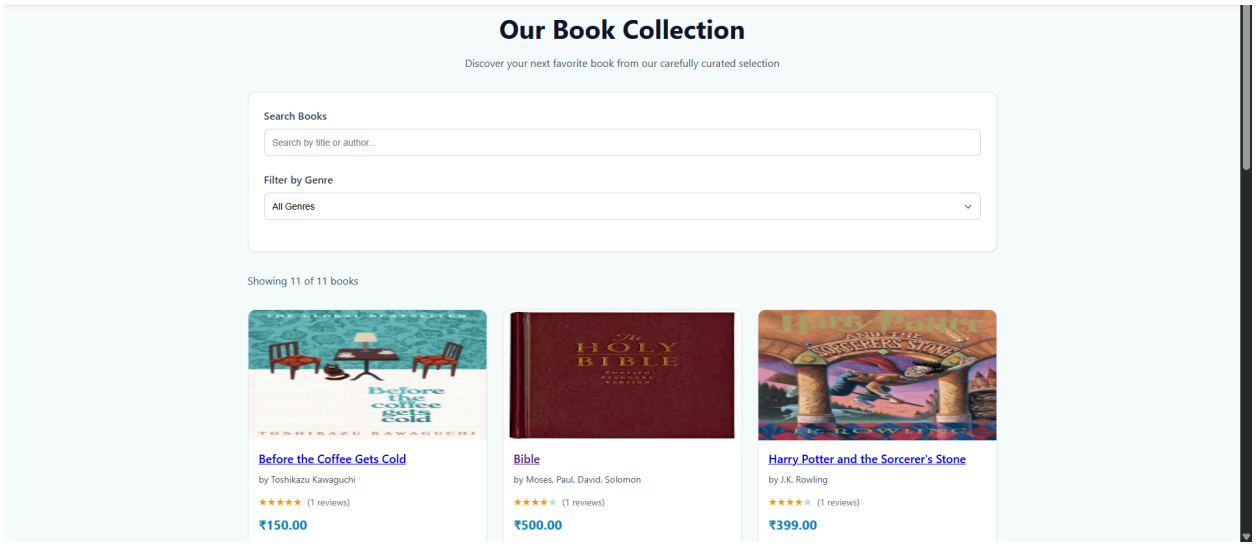

Language
English
Published Year
2025
Price (₹)
0
Stock
0
Add Book
Cancel

- **READ:** A screenshot of the **"Book Inventory"** list in the Admin Dashboard, which displays all books in a grid and an additional "Read" screenshot of the main public-facing **"Our Book Collection"** page (frontend/src/pages/Books.tsx) have been provided below.

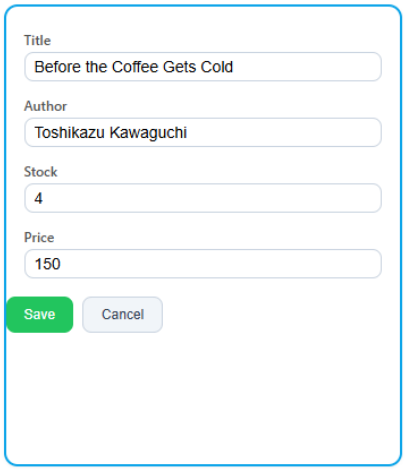
  Admin-

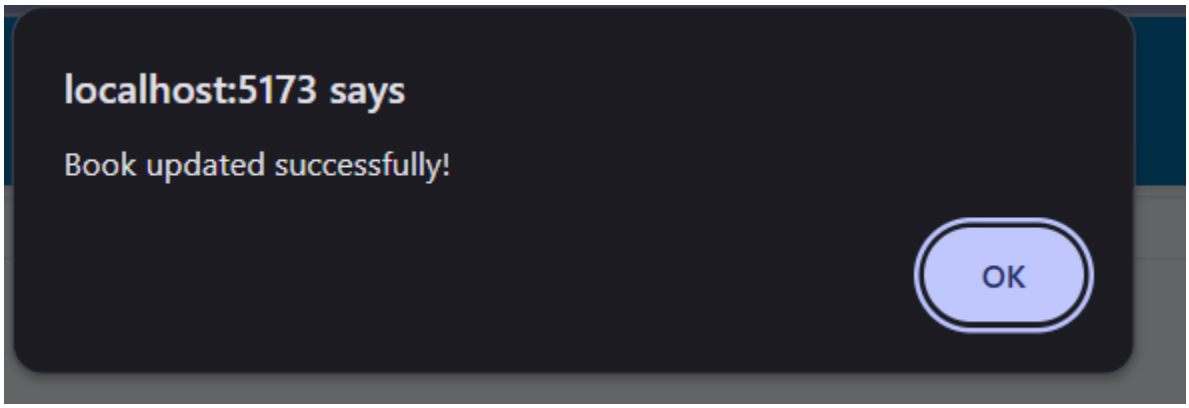


  User-



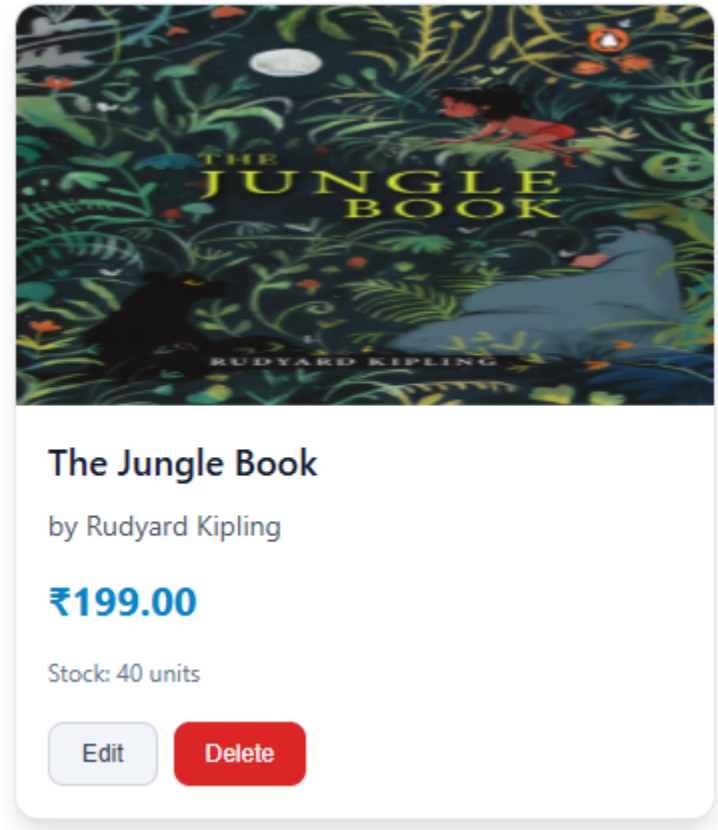
- **UPDATE:** Screenshots of the **"Edit"** interface in the Admin Dashboard is given below. Clicking "Edit" on a book converts its card into an in-line form (frontend/src/pages/AdminDashboard.tsx), allowing the admin to change its price, stock, and other details.

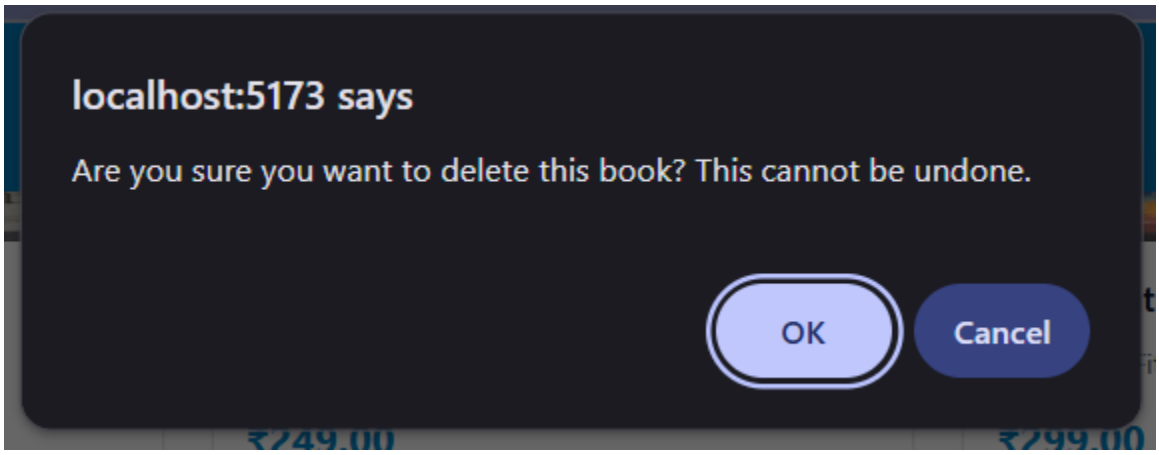




After updating: (Notice **Price** change)

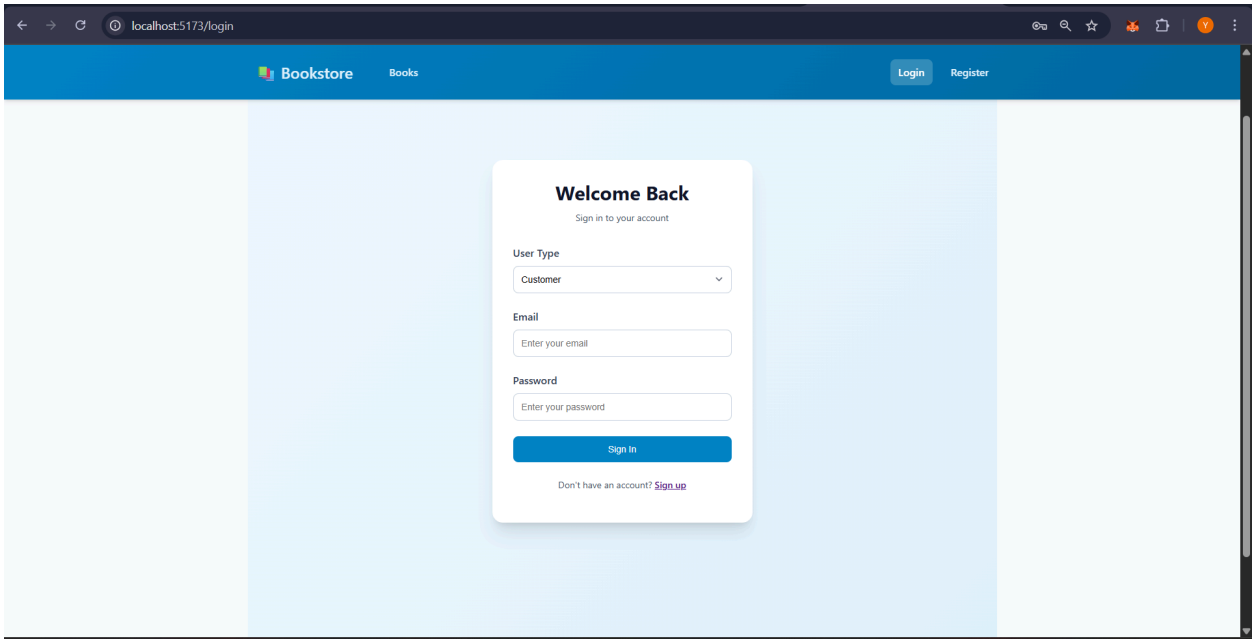
- **DELETE:** A screenshot of the Admin Dashboard showing the **"Delete"** button on a book card, along with the JavaScript window.confirm() dialog that appears to prevent accidental deletion are given below:



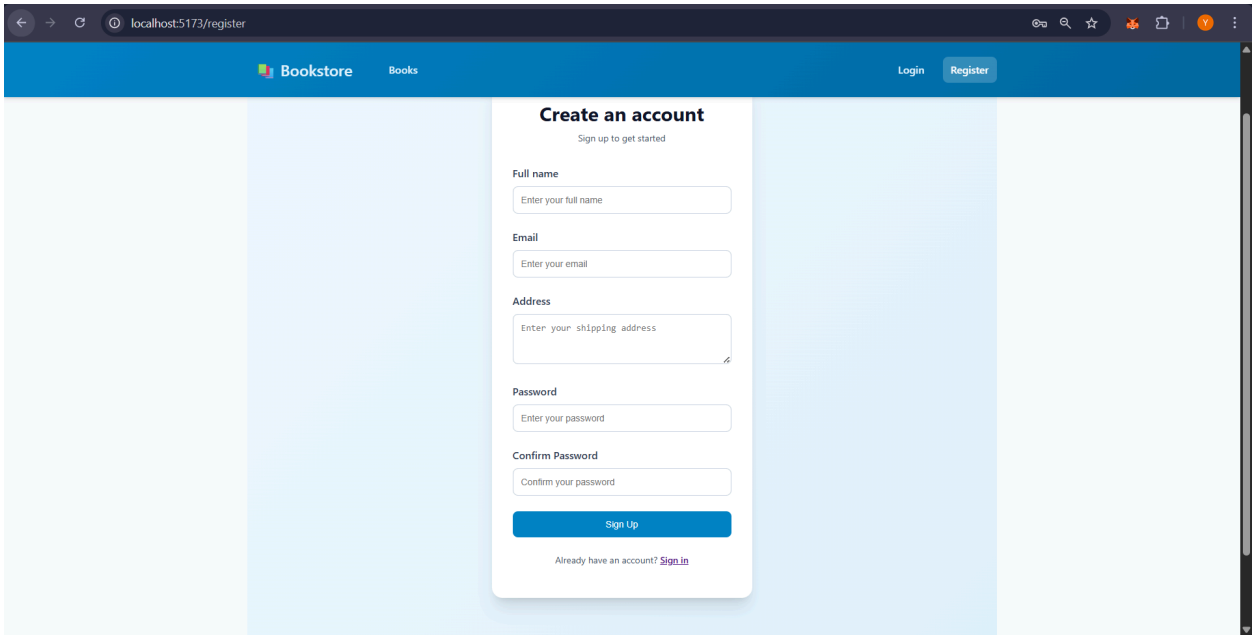


9. List of Functionalities/Features (and Screenshots)

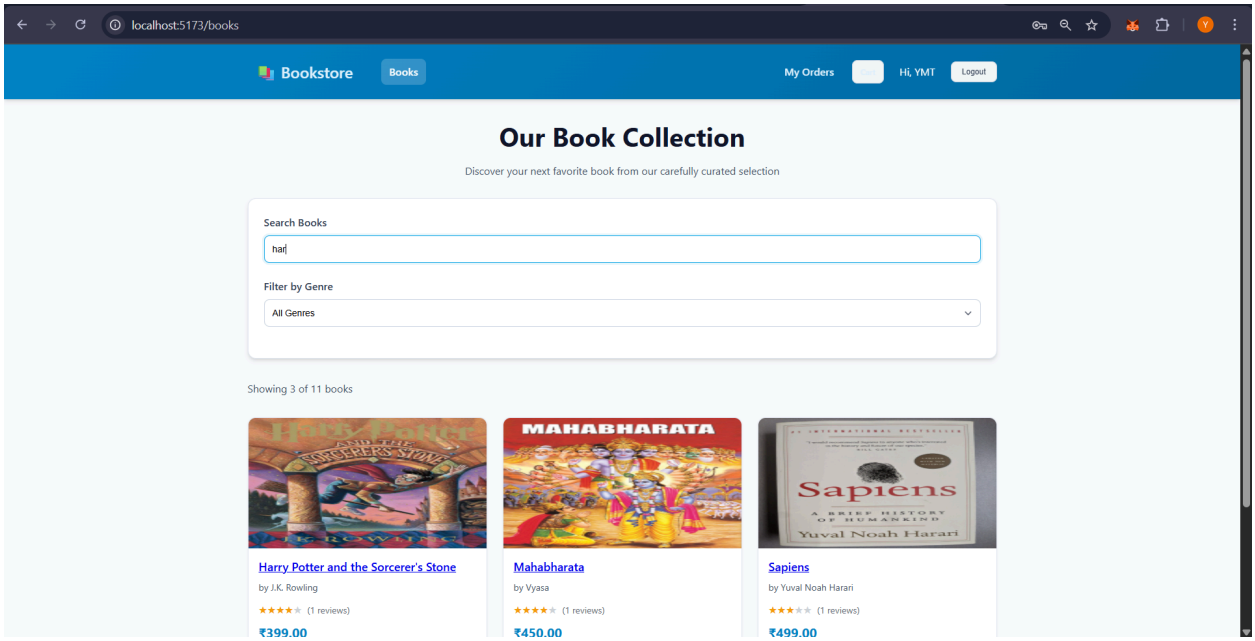
- **Functionality 1: User Authentication (Login & Register)**
  - frontend/src/pages/Login.tsx (Login form)



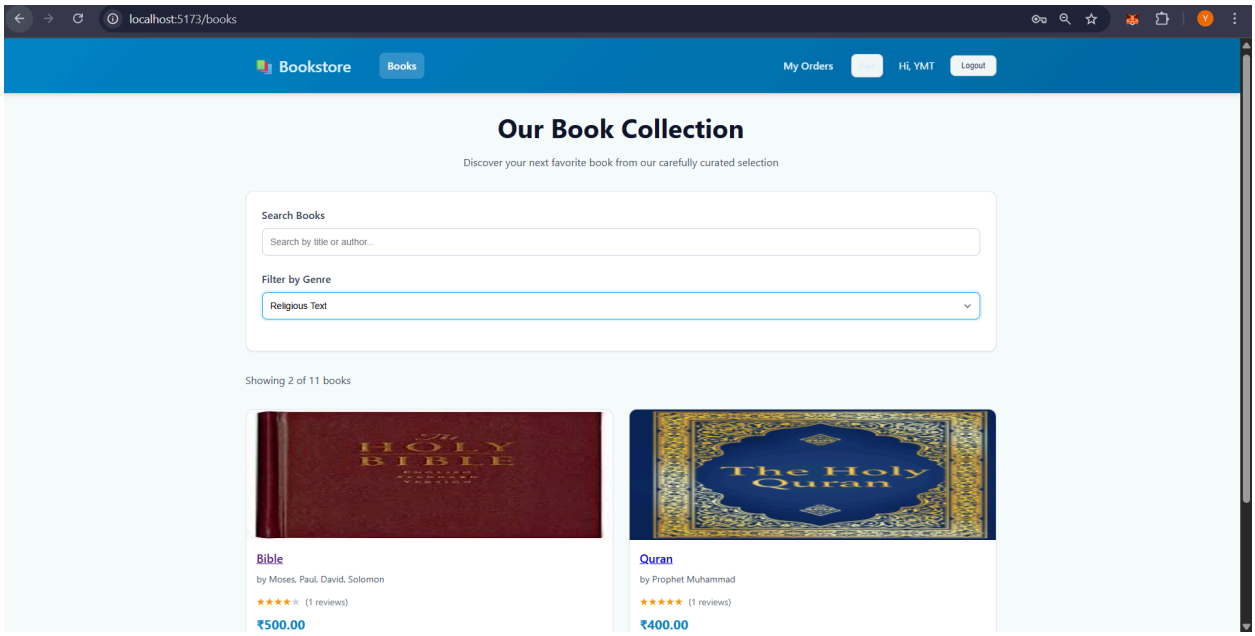
  - frontend/src/pages/Register.tsx (Registration form)



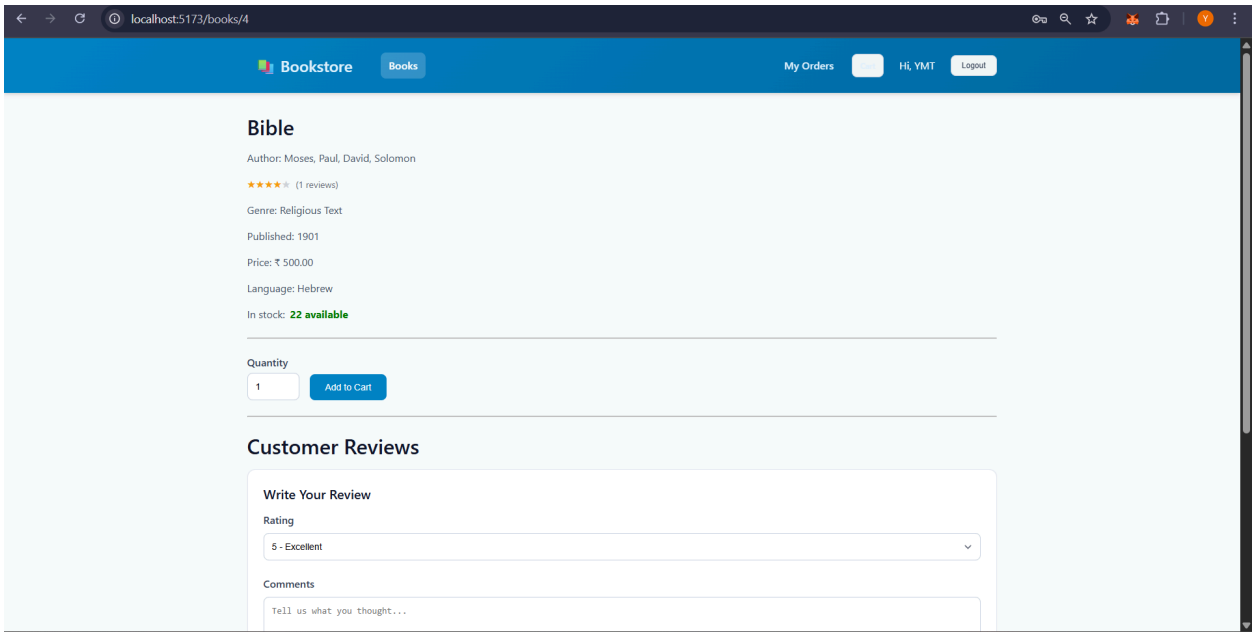
- **Functionality 2: Book Catalog (Browse, Search, Filter)**
  - frontend/src/pages/Books.tsx, showing the book grid with the "Search by title" and



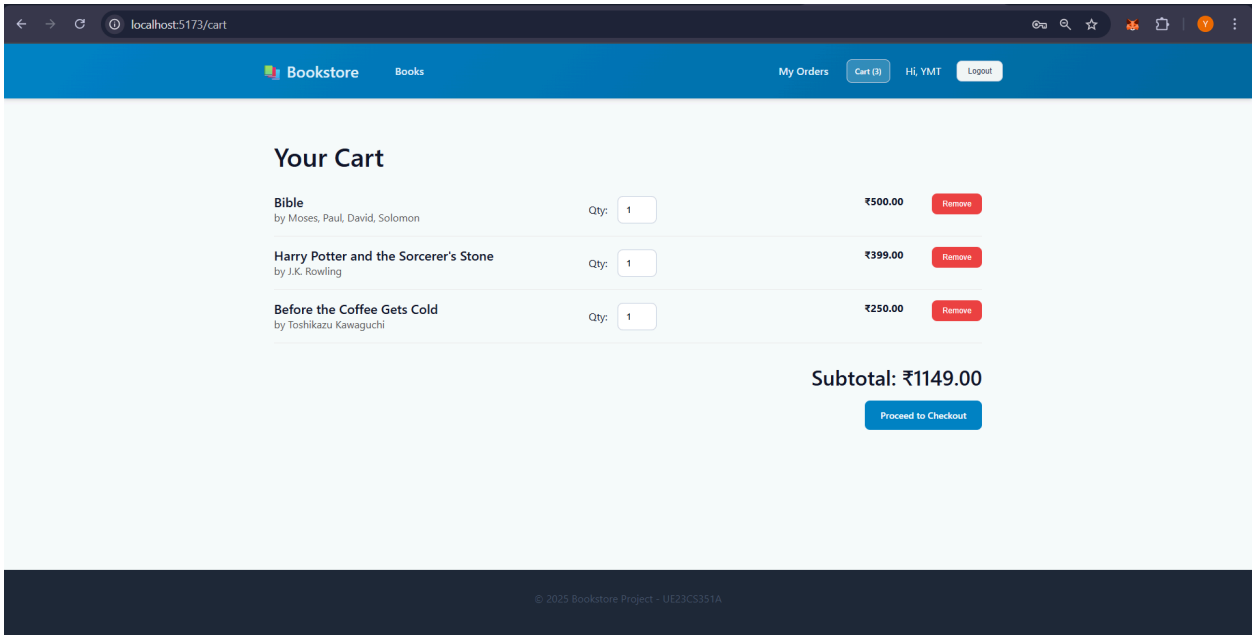
  - "Filter by Genre"



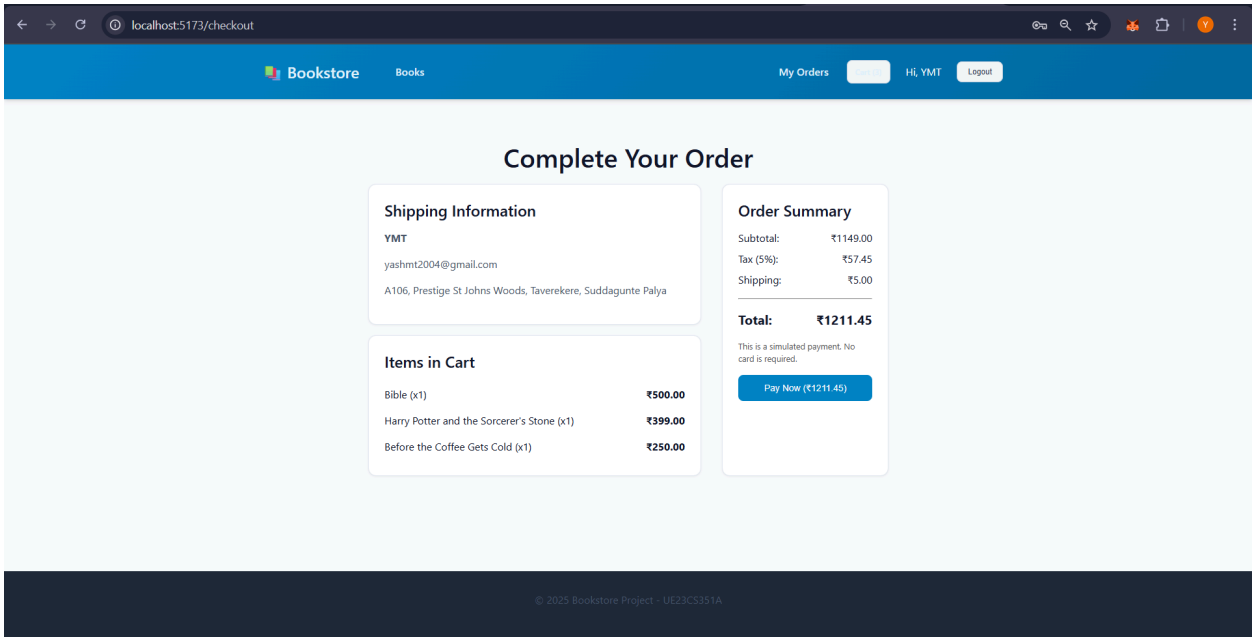
- **Functionality 3: Book Details & Review System**
  - frontend/src/pages/BookDetail.tsx, showing the book's details, the list of existing reviews, and the "Write Your Review" form for logged-in users



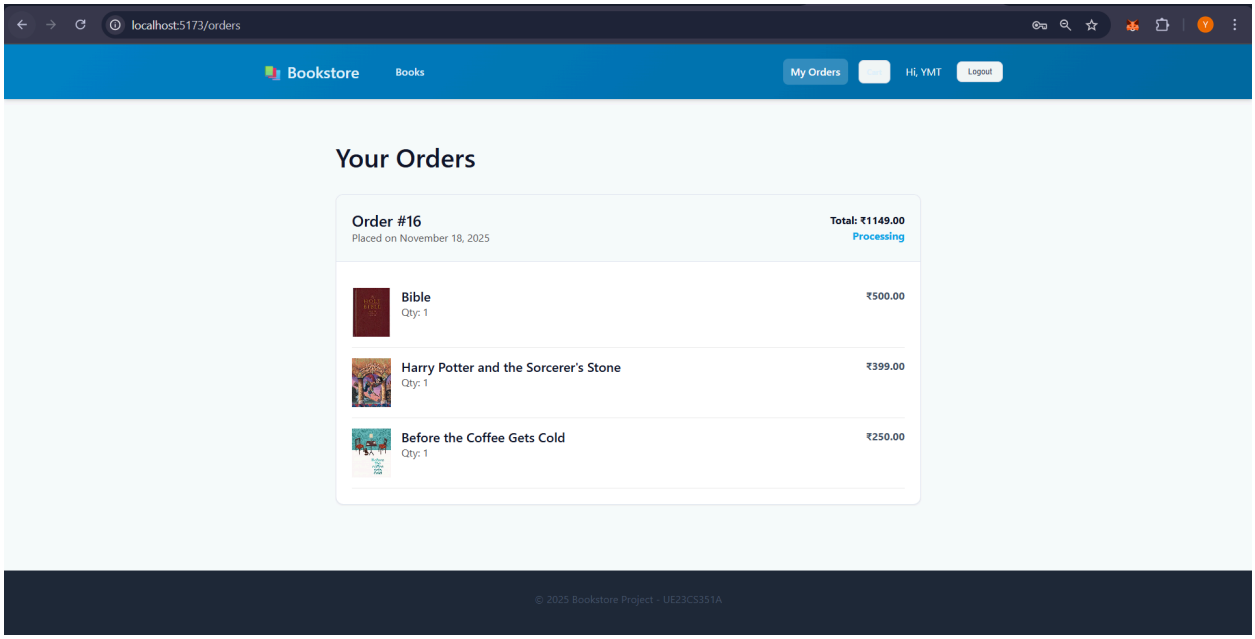
- **Functionality 4: Shopping Cart Management**
  - frontend/src/pages/CartPage.tsx, showing a list of items, quantity inputs, and the "Proceed to Checkout" button



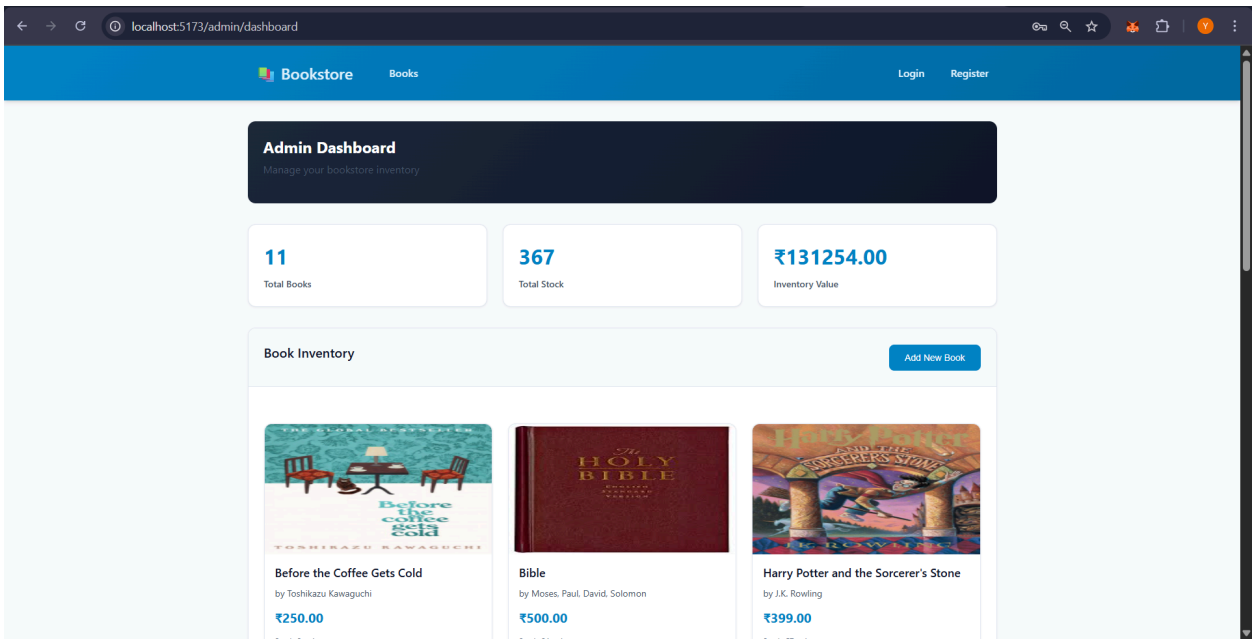
- **Functionality 5: Checkout & Order Placement**
  - frontend/src/pages/CheckoutPage.tsx, showing the final order summary with subtotal, tax, and total, along with the "Pay Now" button



- **Functionality 6: Order History**
  - frontend/src/pages/OrdersPage.tsx, displaying a list of past orders, each with its date, total amount, status, and a nested list of items



- **Functionality 7: Admin Inventory Management (CRUD)**
    - frontend/src/pages/AdminDashboard.tsx, showing the inventory grid, stats cards ("Total Books," "Total Stock"), and the "Add New Book" form.



9. Triggers, Procedures/Functions, Nested query, Join, Aggregate queries

Stored Procedure

- **Name:** sp_CreateOrderFromCart
- **Description:** This procedure is called during checkout. It transactionally converts a customer's cart into a permanent order. Its logic includes:
    1. Reading all items from Cart_Items for the given CID.
    2. Calculating the Total_Amount based on Books.Price and Cart_Items.Quantity.
    3. Inserting a new record into the Orders table.
    4. Copying all cart items into the Order_Items table.
    5. Decrementing the Stock in the Books table for each item.
    6. Deleting all items from the Cart_Items table for the CID.
- SQL:

```
SQL
DELIMITER $$
CREATE PROCEDURE sp_CreateOrderFromCart(IN in_CID INT)
BEGIN
```

```
    -- 1. Declare variables
    DECLARE new_OrderID INT;
    DECLARE total_Order_Amount DECIMAL(10, 2);
    DECLARE stock_Check_Failed INT DEFAULT 0;

    -- 2. Start the transaction
    START TRANSACTION;

    -- 3. Check stock levels BEFORE creating the order
    SELECT 1 INTO stock_Check_Failed
    FROM Cart_Items ci
    JOIN Books b ON ci.BID = b.Book_ID
    WHERE ci.CID = in_CID AND ci.Quantity > b.Stock
    LIMIT 1;

    -- 4. If stock check failed, roll back and send an error
    IF stock_Check_Failed = 1 THEN
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: Not enough stock for
one or more items.';
    ELSE
        -- 5. Calculate the total amount
        SELECT SUM(b.Price * ci.Quantity) INTO total_Order_Amount
        FROM Cart_Items ci
        JOIN Books b ON ci.BID = b.Book_ID
        WHERE ci.CID = in_CID;

        -- 6. Create the main Order record
        INSERT INTO Orders (CID, Total_Amount, Status)
        VALUES (in_CID, total_Order_Amount, 'Processing'); -- Set status to
'Processing'

        -- 7. Get the ID of the new order we just created
        SET new_OrderID = LAST_INSERT_ID();

        -- 8. Copy items from Cart_Items to Order_Items
        INSERT INTO Order_Items (OID, BID, Quantity, Price)
        SELECT new_OrderID, ci.BID, ci.Quantity, b.Price
        FROM Cart_Items ci
        JOIN Books b ON ci.BID = b.Book_ID
        WHERE ci.CID = in_CID;

        -- 9. Update the stock in the Books table
        UPDATE Books b
```

```
        JOIN Cart_Items ci ON b.Book_ID = ci.BID
        SET b.Stock = b.Stock - ci.Quantity
        WHERE ci.CID = in_CID;

        -- 10. Clear the user's cart
        DELETE FROM Cart_Items WHERE CID = in_CID;

        -- 11. Create a 'Complete' payment record
        INSERT INTO Payments (OID, Amount, Method, Status)
        VALUES (new_OrderID, total_Order_Amount, 'Simulated Card', 'Complete');

        -- 12. Commit the transaction
        COMMIT;
    END IF;

END$$

DELIMITER ;
```

Trigger

- **Name:** trg_UpdateBookRating
- **Description:** This trigger fires AFTER INSERT ON Reviews. It recalculates the average rating and total rating count for the BID referenced in the new review and updates the Average_Rating and Rating_Count columns in the Books table.
- SQL:

```
SQL
DELIMITER $$

CREATE TRIGGER trg_UpdateBookRating
AFTER INSERT ON Reviews
FOR EACH ROW
BEGIN
    DECLARE avg_rating DECIMAL(3, 2);
    DECLARE rating_count INT;

    -- Calculate the new average and count for the specific book
    SELECT
        AVG(Rating),
        COUNT(Rating)
```

```
    INTO
        avg_rating,
        rating_count
    FROM Reviews
    WHERE BID = NEW.BID; -- 'NEW' refers to the row just inserted

    -- Update the Books table with the new values
    UPDATE Books
    SET
        Average_Rating = avg_rating,
        Rating_Count = rating_count
    WHERE Book_ID = NEW.BID;

END$$

DELIMITER ;
```

Join Query

- **Use Case:** Fetching all items in a user's cart with their full book details.
- **File:** backend/src/controllers/cart.controller.ts

SQL:

```sql
SELECT b.Book_ID, b.Title, b.Author, b.Price, b.Stock,
ci.Quantity
FROM Cart_Items ci
JOIN Books b ON ci.BID = b.Book_ID
WHERE ci.CID = ?
```

Aggregate Query

- **Use Case:** Fetching a user's order history, with all items for each order grouped into a JSON array.
- **File:** backend/src/routes/orders.ts

SQL:

```sql
SELECT
     o.Order_ID,
     o.Order_Date,
     o.Total_Amount,
     o.Status,
     JSON_ARRAYAGG(
       JSON_OBJECT(
         'Book_ID', b.Book_ID,
         'Title', b.Title,
         'ImagePath', b.ImagePath,
         'Quantity', oi.Quantity,
         'Price', oi.Price
       )
     ) AS items
   FROM Orders o
   JOIN Order_Items oi ON o.Order_ID = oi.OID
   JOIN Books b ON oi.BID = b.Book_ID
   WHERE o.CID = ?
   GROUP BY o.Order_ID
   ORDER BY o.Order_Date DESC
```

- **Aggregate Function Used:** JSON_ARRAYAGG()

Nested Query (Subquery)

- **Use Case:** Finding all customers who have reviewed a specific book (e.g., 'The Jungle Book').
- **Description:** This is a conceptual query that fits the schema and requirement.

SQL:

```sql
SELECT Name, Email
   FROM Customer
   WHERE Customer_ID IN (
```

```
    SELECT CID
    FROM Reviews
    WHERE BID = (
      SELECT Book_ID
      FROM Books
      WHERE Title = 'The Jungle Book'
    )
  );
```

10. Code Snippets for Invocation

Invoking a Stored Procedure (Checkout)

- **File:** backend/src/routes/orders.ts

Snippet: This TypeScript code calls the sp_CreateOrderFromCart procedure when a user hits the /checkout endpoint.

TypeScript

```
router.post('/checkout', protect, async (req, res) => {
  const customerId = req.user?.id;
  if (!customerId) {
    return res.status(401).json({ message: 'Not authorized' });
  }

  try {
    // Call the stored procedure
    await pool.query('CALL sp_CreateOrderFromCart(?)', [customerId]);

    res.status(201).json({ message: 'Order placed successfully!' });

  } catch (error: any) {
    // Check for our custom stock error
    if (error.sqlState === '45000') {
      return res.status(400).json({ message: error.sqlMessage });
    }
    console.error('Checkout error:', error);
    res.status(500).json({ message: 'Server error during checkout.' });
  }
});
```

Invoking a Join Query (Get Cart)

- **File:** backend/src/controllers/cart.controller.ts

Snippet: This code executes the JOIN query to fetch a user's cart contents.

TypeScript

```
export const getCart = async (req: Request, res: Response) => {
  const customerId = req.user?.id;
  if (!customerId) {
    return res.status(401).json({ message: 'Not authorized' });
  }

  try {
    // This is a JOIN query as required by your project
    const [items]: any = await pool.query(
      `SELECT b.Book_ID, b.Title, b.Author, b.Price, b.Stock, ci.Quantity
       FROM Cart_Items ci
       JOIN Books b ON ci.BID = b.Book_ID
       WHERE ci.CID = ?`,
      [customerId]
    );
    res.json(items);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

Invoking an INSERT (which fires a Trigger)

- **File:** backend/src/routes/reviews.ts
- Snippet: This code inserts a new review. The database trigger trg_UpdateBookRating would fire automatically after this INSERT succeeds.

TypeScript

```
router.post('/:bookId', protect, async (req, res) => {
  const { bookId } = req.params;
  const { rating, comments } = req.body;
  const customerId = req.user?.id;
```

```
  // ...
  try {
    await pool.query(
      'INSERT INTO Reviews (BID, CID, Rating, Comments) VALUES (?, ?, ?, ?)',
      [bookId, customerId, rating, comments]
    );
    // The trigger (sp_UpdateBookRating) will automatically update the book's
average
    res.status(201).json({ message: 'Review added successfully!' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
});
```

11. SQL Queries (.sql file content)

SQL

```
-- --- 1. SCHEMA SETUP ---
CREATE DATABASE IF NOT EXISTS bookstore;
USE bookstore;

-- --- 2. TABLE DEFINITIONS (DDL) ---

-- Table: Customer
CREATE TABLE Customer (
    Customer_ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL UNIQUE,
    Address TEXT,
    Joined_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
ALTER TABLE Customer
ADD COLUMN Password_Hash VARCHAR(255) NOT NULL
AFTER Email;

-- Table: Books
CREATE TABLE Books (
    Book_ID INT PRIMARY KEY AUTO_INCREMENT,
```

```
    Title VARCHAR(255) NOT NULL,
    Author VARCHAR(255) NOT NULL,
    Genre VARCHAR(100),
    Published YEAR NOT NULL,
    Price DECIMAL(10, 2) NOT NULL CHECK (Price >= 0),
    Stock INT NOT NULL DEFAULT 0 CHECK (Stock >= 0)
);
ALTER TABLE Books
ADD Language VARCHAR(100) NOT NULL DEFAULT 'English';
ALTER TABLE Books
ADD COLUMN ImagePath VARCHAR(255) DEFAULT '/uploads/default.jpg';
ALTER TABLE Books
ADD COLUMN Average_Rating DECIMAL(3, 2) NOT NULL DEFAULT 0.00,
ADD COLUMN Rating_Count INT NOT NULL DEFAULT 0;

-- Table: Orders
CREATE TABLE Orders (
    Order_ID INT PRIMARY KEY AUTO_INCREMENT,
    CID INT NOT NULL,
    Order_Date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Total_Amount DECIMAL(10, 2) NOT NULL CHECK (Total_Amount >= 0),
    Status VARCHAR(50) NOT NULL DEFAULT 'Pending',
    FOREIGN KEY (CID) REFERENCES Customer(Customer_ID) ON DELETE RESTRICT
);

-- Table: Order_Items
CREATE TABLE Order_Items (
    OID INT NOT NULL,
    BID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity > 0),
    Price DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (OID, BID),
    FOREIGN KEY (OID) REFERENCES Orders(Order_ID) ON DELETE CASCADE,
    FOREIGN KEY (BID) REFERENCES Books(Book_ID) ON DELETE RESTRICT
);

-- Table: Reviews
CREATE TABLE Reviews (
    Review_ID INT PRIMARY KEY AUTO_INCREMENT,
    BID INT NOT NULL,
    CID INT NOT NULL,
    Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5),
    Comments TEXT,
    Created_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```sql
    FOREIGN KEY (BID) REFERENCES Books(Book_ID) ON DELETE CASCADE,
    FOREIGN KEY (CID) REFERENCES Customer(Customer_ID) ON DELETE CASCADE
);

-- Table: Payments
CREATE TABLE Payments (
    Payment_ID INT PRIMARY KEY AUTO_INCREMENT,
    OID INT NOT NULL UNIQUE,
    Payment_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Amount DECIMAL(10, 2) NOT NULL,
    Method VARCHAR(50) NOT NULL,
    Status VARCHAR(50) NOT NULL DEFAULT 'Incomplete',
    FOREIGN KEY (OID) REFERENCES Orders(Order_ID) ON DELETE CASCADE
);

-- Table: Cart_Items
CREATE TABLE Cart_Items (
    CID INT NOT NULL,
    BID INT NOT NULL,
    Quantity INT NOT NULL DEFAULT 1 CHECK (Quantity > 0),
    Added_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (CID, BID),
    FOREIGN KEY (CID) REFERENCES Customer(Customer_ID) ON DELETE CASCADE,
    FOREIGN KEY (BID) REFERENCES Books(Book_ID) ON DELETE CASCADE
);


-- --- 3. ADVANCED LOGIC (PROCEDURES & TRIGGERS) ---

-- Stored Procedure: Create Order From Cart
DELIMITER $$
CREATE PROCEDURE sp_CreateOrderFromCart(IN in_CID INT)
BEGIN
    -- 1. Declare variables
    DECLARE new_OrderID INT;
    DECLARE total_Order_Amount DECIMAL(10, 2);
    DECLARE stock_Check_Failed INT DEFAULT 0;

    -- 2. Start the transaction
    START TRANSACTION;

    -- 3. Check stock levels BEFORE creating the order
    SELECT 1 INTO stock_Check_Failed
    FROM Cart_Items ci
```

```
    JOIN Books b ON ci.BID = b.Book_ID
    WHERE ci.CID = in_CID AND ci.Quantity > b.Stock
    LIMIT 1;

    -- 4. If stock check failed, roll back and send an error
    IF stock_Check_Failed = 1 THEN
        ROLLBACK;
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: Not enough stock for
one or more items.';
    ELSE
        -- 5. Calculate the total amount
        SELECT SUM(b.Price * ci.Quantity) INTO total_Order_Amount
        FROM Cart_Items ci
        JOIN Books b ON ci.BID = b.Book_ID
        WHERE ci.CID = in_CID;

        -- 6. Create the main Order record
        INSERT INTO Orders (CID, Total_Amount, Status)
        VALUES (in_CID, total_Order_Amount, 'Processing'); -- Set status to
'Processing'

        -- 7. Get the ID of the new order we just created
        SET new_OrderID = LAST_INSERT_ID();

        -- 8. Copy items from Cart_Items to Order_Items
        INSERT INTO Order_Items (OID, BID, Quantity, Price)
        SELECT new_OrderID, ci.BID, ci.Quantity, b.Price
        FROM Cart_Items ci
        JOIN Books b ON ci.BID = b.Book_ID
        WHERE ci.CID = in_CID;

        -- 9. Update the stock in the Books table
        UPDATE Books b
        JOIN Cart_Items ci ON b.Book_ID = ci.BID
        SET b.Stock = b.Stock - ci.Quantity
        WHERE ci.CID = in_CID;

        -- 10. Clear the user's cart
        DELETE FROM Cart_Items WHERE CID = in_CID;

        -- 11. Create a 'Complete' payment record
        INSERT INTO Payments (OID, Amount, Method, Status)
        VALUES (new_OrderID, total_Order_Amount, 'Simulated Card', 'Complete');
```

```
        -- 12. Commit the transaction
        COMMIT;
    END IF;

END$$
DELIMITER ;

-- Trigger: Update Book Rating on New Review
DELIMITER $$
CREATE TRIGGER trg_UpdateBookRating
AFTER INSERT ON Reviews
FOR EACH ROW
BEGIN
    DECLARE avg_rating DECIMAL(3, 2);
    DECLARE rating_count INT;

    -- Calculate the new average and count for the specific book
    SELECT
        AVG(Rating),
        COUNT(Rating)
    INTO
        avg_rating,
        rating_count
    FROM Reviews
    WHERE BID = NEW.BID; -- 'NEW' refers to the row just inserted

    -- Update the Books table with the new values
    UPDATE Books
    SET
        Average_Rating = avg_rating,
        Rating_Count = rating_count
    WHERE Book_ID = NEW.BID;

END$$
DELIMITER ;


-- --- 4. DATA POPULATION (DML) ---

-- Add Customers (Total 10)
-- Note: Passwords must be set via the application's hashing (bcrypt) logic.
INSERT INTO Customer (Name, Email, Address) VALUES
('Yashvardhan Singh', 'abc@email.com', '123 Street, Bengaluru, Karnataka'),
('Yash Tharappan', 'def@email.com', '456 Road, Bengaluru, Karnataka'),
```

```
('Vibhav K', 'hij@email.com', '789 Lane, Bengaluru, Karnataka'),
('Ravi Kumar', 'ravi@email.com', '101 Main St, Delhi, India'),
('Ananya Sharma', 'ananya@email.com', '202 Park Rd, Mumbai, India'),
('Karan Mehta', 'karan@email.com', '303 Lakeview, Pune, India'),
('Sneha Gupta', 'sneha@email.com', '404 Hilltop, Chennai, India'),
('Aditya Singh', 'aditya@email.com', '505 Garden St, Kolkata, India'),
('Isha Reddy', 'isha@email.com', '606 Riverside, Hyderabad, India'),
('Vikram Patel', 'vikram@email.com', '707 Downtown, Jaipur, India');

-- Add Books (Total 10)
INSERT INTO Books (Title, Author, Genre, Published, Price, Stock, Language,
ImagePath) VALUES
('The Jungle Book', 'Rudyard Kipling', 'Children''s Fiction', 1901, 199.00, 40,
'English', '/uploads/1.jpg'),
('The Immortals of Meluha', 'Amish Tripathi', 'Mythological Fiction', 2010,
299.00, 50, 'English', '/uploads/2.jpg'),
('Mahabharata', 'Vyasa', 'Epic', 1951, 450.00, 30, 'Sanskrit',
'/uploads/3.jpg'),
('Bible', 'Moses, Paul, David, Solomon', 'Religious Text', 1901, 500.00, 25,
'Hebrew', '/uploads/4.jpg'),
('Quran', 'Prophet Muhammad', 'Religious Text', 1924, 400.00, 20, 'Arabic',
'/uploads/5.jpg'),
('Harry Potter and the Sorcerer''s Stone', 'J.K. Rowling', 'Fantasy', 1997,
399.00, 60, 'English', '/uploads/6.jpg'),
('The Alchemist', 'Paulo Coelho', 'Fiction', 1988, 249.00, 45, 'English',
'/uploads/7.jpg'),
('Sapiens', 'Yuval Noah Harari', 'Non-Fiction', 2011, 499.00, 35, 'English',
'/uploads/8.jpg'),
('Inferno', 'Dan Brown', 'Thriller', 2013, 399.00, 50, 'English',
'/uploads/9.jpg'),
('The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 1925, 299.00, 40,
'English', '/uploads/10.jpg');

-- Add Orders (Total 9)
INSERT INTO Orders (CID, Total_Amount, Status) VALUES
(1, 648.50, 'Shipped'),
(2, 280.75, 'Delivered'),
(1, 870.00, 'Pending'),
(3, 520.00, 'Delivered'),
(4, 350.50, 'Shipped'),
(5, 610.00, 'Pending'),
(6, 430.25, 'Delivered'),
(7, 720.00, 'Shipped'),
(8, 295.50, 'Pending'),
```

```
(9, 480.75, 'Delivered');

-- Add Order_Items
INSERT INTO Order_Items (OID, BID, Quantity, Price) VALUES
(1, 1, 1, 299.00),
(1, 2, 1, 349.50),
(2, 4, 1, 280.75),
(3, 3, 1, 320.00),
(3, 5, 1, 550.00),
(4, 6, 1, 399.00),
(4, 7, 1, 121.50),
(5, 8, 2, 249.00),
(6, 9, 1, 399.00),
(6, 10, 1, 31.25),
(7, 2, 1, 299.00),
(7, 5, 1, 181.75);

-- Add Reviews
INSERT INTO Reviews (BID, CID, Rating, Comments) VALUES
(1, 1, 5, 'An absolutely fantastic and thought-provoking read! Highly
recommend.'),
(4, 2, 4, 'A very compelling story with rich characters. Enjoyed it
thoroughly.'),
(1, 2, 4, 'Good book, interesting concept.'),
(2, 3, 5, 'Loved the mythological twist!'),
(3, 4, 4, 'Great insights on history and philosophy.'),
(5, 5, 5, 'A classic must-read.'),
(6, 6, 4, 'Engaging and thrilling!'),
(7, 7, 5, 'Inspirational story, beautifully written.'),
(8, 8, 3, 'Interesting concept but a bit slow.'),
(9, 9, 5, 'Absolutely gripping thriller!');

-- Add Payments
INSERT INTO Payments (OID, Amount, Method, Status) VALUES
(1, 648.50, 'Credit Card', 'Completed'),
(2, 280.75, 'UPI', 'Completed'),
(3, 870.00, 'Credit Card', 'Pending'),
(4, 520.00, 'UPI', 'Completed'),
(5, 610.00, 'Debit Card', 'Pending'),
(6, 430.25, 'Credit Card', 'Completed'),
(7, 720.00, 'UPI', 'Completed'),
(8, 295.50, 'Debit Card', 'Pending'),
(9, 480.75, 'Credit Card', 'Completed');
```

```sql
-- --- 5. DATA INITIALIZATION ---
-- Backfill Average_Rating and Rating_Count for the initial data
-- (The trg_UpdateBookRating trigger will handle all new reviews automatically)
UPDATE Books b
JOIN (
    SELECT
        BID,
        AVG(Rating) AS avg_r,
        COUNT(Rating) AS count_r
    FROM Reviews
    GROUP BY BID
) AS stats ON b.Book_ID = stats.BID
SET
    b.Average_Rating = stats.avg_r,
    b.Rating_Count = stats.count_r;
```

13. GitHub Repo Link

https://github.com/YMTEA-05/Bookstore.git