# Project Title: Bookstore Management System

**Objective:** Develop a full-featured RESTful API for a bookstore management system that can later integrate seamlessly with a front-end application.

**Features:**

1. User Authentication and Profiles

- **Sign Up:** Users can register using their email, password, and name. Validate email uniqueness.
- **Sign In:** Authenticate users and provide JWT on successful sign-in.
- **Profile:** Authenticated users can view and update their profiles.

2. Books CRUD with Advanced Features

- **Basic CRUD:** As previously mentioned.
- **Book Reviews:** Authenticated users can leave reviews and ratings for books.
- **Search & Filter:** Allow users to search books by title or author and filter by ratings.

3. Shopping Cart

- Authenticated users can add books to their cart, update the quantity, or remove books from the cart.
- View cart contents, with book details and total price.

4. Error Handling and Validation

- - Comprehensive error handling for all routes.
- - Input validation for all API endpoints using libraries like `jo`i or `express-validator`.

**Detailed Steps:**

1. Set Up & Database Integration

- Initialize Node.js project and install necessary packages.
- Connect to MongoDB and define User, Book, and Cart schemas and models using Mongoose.

2. User Authentication and Profile Management

- Create routes for registration, login, and user profile management.
- Implement password hashing using libraries like `bcryp`t.

- Secure routes using JWT middleware.

3. Books CRUD & Advanced Features

- Implement routes for book CRUD operations.
- Develop endpoints for adding reviews to books. Each review should contain a comment and a rating (1-5 stars).
- Implement a search and filter mechanism for books.

4. Shopping Cart Functionality

- Create routes for adding books to cart, updating quantity, viewing the cart, and deleting books from the cart.
- Calculate the total price for the books in the cart.

5. Error Handling & Input Validation

- Handle common errors like missing routes, invalid inputs, and unauthorized access.
- Validate all user inputs to prevent malicious or unintended requests.

6. Testing & Documentation

- Test all routes with Postman or similar tools.
- Document each API endpoint, detailing request methods, parameters, and expected responses.

**Evaluation Criteria (Expanded):**

1. Functional Completeness

Ensure all the described features are implemented and working as expected.

2. Integration Readiness

API should be designed in a way that makes it easy to integrate with a front-end application. Consider practices like CORS setup for cross-origin requests.

3. Security

- Proper use of password hashing and secure JWT implementation.
- Validation of user inputs to prevent potential attacks.

4. Code and Database Design

- Efficient and organized code structure.
- Proper schema design ensuring data integrity and optimized queries.

5. Testing and Documentation

- Comprehensive testing of each route.
- Detailed documentation for future developers or teams.

**Indicative API Response:**

**User Authentication and Profiles**

1. Sign Up

- **Endpoint:** `POST /users/signup`
- **Input:** `{ email: "user@example.com", password: "userPassword", name: "UserName" }`
- **Output:** `{ success: true, message: "User registered successfully." }`

2. Sign In

- **Endpoint:** `POST /users/signin`
- **Input:** `{ email: "user@example.com", password: "userPassword" }`
- **Output:** `{ success: true, token: "JWT_TOKEN", userId: "userId" }`

3. Profile View and Update

- **Endpoint:** `GET /users/profile` and `PUT /users/profile`
- **Input (for PUT):** `{ name: "UpdatedName" }`
- **Output:** `{ success: true, user: { email: "user@example.com", name: "UserName" } }`
  Books CRUD with Advanced Features

4. Add Book

- **Endpoint:** `POST /books`
- **Input:** `{ title: "BookTitle", author: "AuthorName", ISBN: "1234567890", price: 19.99, quantity: 10 }`
- **Output:** `{ success: true, message: "Book added successfully.", bookId: "bookId" }`

5. Get All Books

- **Endpoint:** `GET /books`

- **Output:** `{ success: true, books: [ { /* book data */ }, ... ] }`

6. Get Single Book

- **Endpoint:** `GET /books/:bookId`
- **Output:** `{ success: true, book: { /* book data */ } }`

7. Update Book

- **Endpoint:** `PUT /books/:bookId`
- **Input:** `{ title: "NewTitle", price: 22.99 }`
- **Output:** `{ success: true, message: "Book updated successfully." }`

8. Delete Book

- **Endpoint:** `DELETE /books/:bookId`
- **Output:** `{ success: true, message: "Book deleted successfully." }`

9. Add Review to Book

- **Endpoint:** `POST /books/:bookId/reviews`
- **Input:** `{ comment: "Great book!", rating: 5 }`
- **Output:** `{ success: true, message: "Review added successfully." }`

10. Search Books

- **Endpoint:** `GET /books/search?query=BookTitle`
- **Output:** `{ success: true, books: [ { /* matching books */ }, ... ] }`
- Shopping Cart Functionality

11. Add Book to Cart

- **Endpoint:** `POST /car`t
- **Input:** `{ bookId: "bookId", quantity: 2 }`
- **Output:** `{ success: true, message: "Book added to cart." }`

12. Get Cart Contents

- **Endpoint:** `GET /car`t
- **Output:** `{ success: true, cart: { /* cart data */ } }`

13. Update Book Quantity in Cart

- **Endpoint:** `PUT /cart/:bookId`
- **Input:** `{ quantity: 3 }`
- **Output:** `{ success: true, message: "Cart updated successfully." }`

14. Delete Book from Cart

- **Endpoint:** `DELETE /cart/:bookId`
- **Output:** `{ success: true, message: "Book removed from cart." }`

Remember, all the routes except for Sign Up and Sign In should have JWT middleware for user authentication. This will ensure that only authenticated users can access those routes.

This is a high-level design, and the actual implementation might require more detailed responses, handling corner cases, and providing more metadata in the response, like pagination information for long lists of books.