Online Bookstore API - Project Documentation

Demo Video: https://youtu.be/cb7grv0Z2NE
Live API Documentation: https://online-bookstore-yqif.onrender.com/api-docs/

1. Project Proposal

1.1 Problem Description

The modern digital marketplace has transformed how consumers purchase books, with online bookstores becoming the preferred method for book discovery and acquisition. This project implements a comprehensive RESTful API for an online bookstore that demonstrates proper backend architecture, efficient data management, and scalable API design principles.

Key Problems Addressed:

- Need for a robust RESTful API architecture for e-commerce systems
- Requirement for efficient book inventory management with CRUD operations
- User authentication and session management with JWT tokens
- In-memory shopping cart implementation for session-based operations
- Comprehensive order management system with status tracking
- API documentation and testing capabilities

1.2 Proposed Solution

We have developed a comprehensive RESTful API for an online bookstore using Node.js and MongoDB. The solution features a clean separation of concerns through well-defined API endpoints, robust authentication system, and efficient data management.

Core Components:

- **RESTful API Architecture**: Clean, standardized endpoints following REST principles
- **Book Management System**: Complete CRUD operations for book inventory
- **User Authentication**: JWT-based authentication with secure password hashing
- **Shopping Cart System**: Session-based in-memory cart functionality
- **Order Management**: Complete order lifecycle with status tracking
- **API Documentation**: Interactive Swagger UI for testing and documentation

Key Features:

- User registration and authentication with JWT tokens
- Complete book CRUD operations
- Session-based shopping cart with add/remove/update operations

- Order placement with shipping address and status management
- Interactive API documentation with Swagger UI
- Scalable MongoDB database integration
- Environment-based configuration management

1.3 Technologies Used

Backend Technologies:

- **Node.js**: Server-side JavaScript runtime environment
- **Express.js**: Fast, unopinionated web framework for Node.js
- **MongoDB**: NoSQL database for scalable data storage
- **Mongoose**: Elegant MongoDB object modeling for Node.js

Authentication & Security:

- **JWT (jsonwebtoken)**: Secure token-based authentication
- **bcrypt**: Password hashing and salting library
- **cors**: Cross-Origin Resource Sharing middleware

API Documentation:

- **Swagger UI**: Interactive API documentation and testing interface
- **swagger-jsdoc**: JSDoc comments to Swagger specification

Development Tools:

- **nodemon**: Development server with auto-restart
- **dotenv**: Environment variable management
- **Git**: Version control system
- **npm**: Package management

Deployment:

- **Render**: Cloud platform for backend deployment
- **MongoDB Atlas**: Cloud MongoDB hosting service

1.4 Project Timeline and Milestones

Phase 1: Project Setup and Planning

- Initialize project repository
- Set up development environment
- Create project structure
- Design database schema
- Create UML diagrams

Phase 2: Backend Development

- Implement core classes (Book, User, ShoppingCart)

- Set up MongoDB connection
- Create database models
- Implement user authentication
- Develop book management APIs

Phase 3: Frontend Development

- Create user interface components
- Implement book browsing functionality
- Develop shopping cart interface
- Create user registration/login forms
- Implement purchase workflow

Phase 4: Integration and Testing

- Integrate frontend with backend APIs
- Perform comprehensive testing
- Fix bugs and optimize performance
- Implement error handling

Phase 5: Documentation and Deployment

- Complete project documentation
- Prepare deployment configuration
- Final testing and quality assurance
- Project presentation preparation

1.5 Division of Responsibilities

Simeon Azeh: Backend Developer

- Database design and implementation
- Server-side class development
- API development and testing
- User authentication system
- Database integration
- API integration between frontend and backend
- Session management implementation
- System testing and debugging

Damilare Azeez: Frontend Developer

- User interface design and implementation
- Client-side JavaScript development
- Responsive design implementation
- User experience optimization
- Frontend testing

- Shopping cart functionality
- Purchase workflow implementation

Lina IRATWE : Documentation and Quality Assurance

- Project documentation creation
- UML diagram development
- Testing and quality assurance
- Code review and optimization
- Deployment preparation

---

2. Software Documentation

2.1 System Architecture

The Online Bookstore follows a three-tier architecture pattern:

Presentation Layer (Frontend):

- HTML/CSS/JavaScript client interface
- Responsive web design for multiple devices
- Interactive user interface components
- Client-side form validation

Business Logic Layer (Backend):

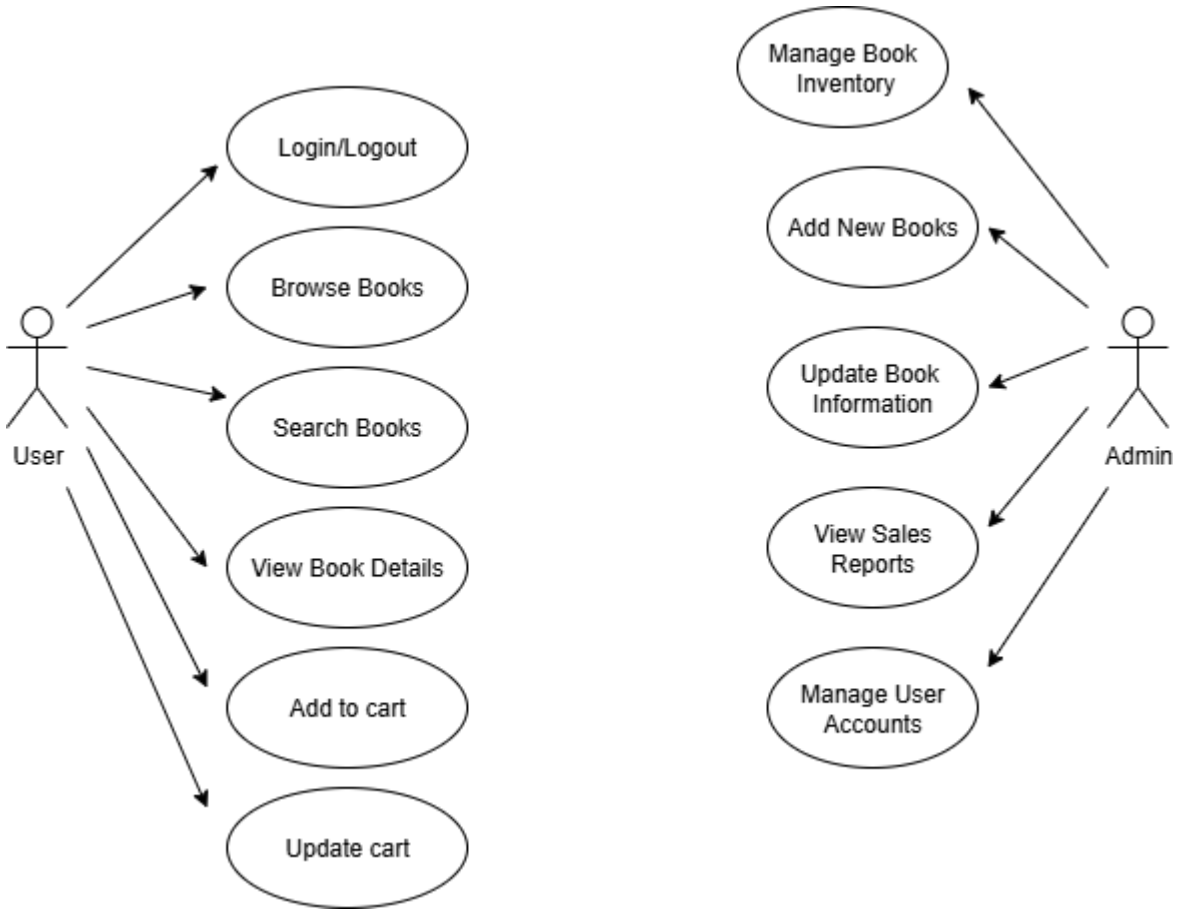
- Node.js/Express.js server application
- Object-oriented class implementations
- API endpoints for data operations
- Authentication and session management

Data Access Layer (Database):

- MongoDB database for persistent storage
- Mongoose ODM for data modeling
- Efficient indexing for search operations
- Data validation and constraints

2.2 UML Diagrams

2.2.1 Class Diagram



2.2.2 Detailed Class Relationships

Inheritance Relationships:

- No inheritance in this design (composition over inheritance principle)

Association Relationships:

- User → ShoppingCart: One-to-One (Each user has one active cart)
- User → Order: One-to-Many (Each user can have multiple orders)
- ShoppingCart → CartItem: One-to-Many (Cart contains multiple items)
- Order → OrderItem: One-to-Many (Order contains multiple items)
- CartItem → Book: Many-to-One (Multiple cart items can reference same book)
- OrderItem → Book: Many-to-One (Multiple order items can reference same book)

Composition Relationships:

- ShoppingCart *-- CartItem: CartItems cannot exist without a ShoppingCart
- Order *-- OrderItem: OrderItems cannot exist without an Order
- User *-- Address: Address is part of User entity

2.2.2 Use Case Diagram in detail

Use Case	Description	Preconditions	Postconditions
Register Account	Customer creates a new account	System is accessible	Account created, confirmation sent
Login/Logout	Customer authenticates/ends session	Valid credentials	Session established/terminated
Browse Books	Customer views book catalog	System is accessible	Book list displayed
Search Books	Customer searches for specific books	System is accessible	Search results displayed
View Book Details	Customer views detailed book information	Book exists in system	Book details displayed
Add to Cart	Customer adds books to shopping cart	User logged in, book available	Item added to cart
View Cart	Customer views cart contents	User logged in	Cart contents displayed
Update Cart	Customer modifies cart items	User logged in, items in cart	Cart updated
Checkout	Customer completes purchase	Items in cart, valid payment	Order created, payment processed
View Order History	Customer views past orders	User logged in	Order history displayed

2.2.4 Use Case Relationships

Include Relationships (<<include>>):

- Browse Books → View Book Details
- Search Books → View Book Details
- Add to Cart → Login (user must be authenticated)
- View Cart → Login (user must be authenticated)
- Update Cart → Login (user must be authenticated)
- Checkout → Login (user must be authenticated)
- View Order History → Login (user must be authenticated)
- All Admin use cases → Login (admin must be authenticated)

Extend Relationships (<<extend>>):

- Add New Books → Manage Book Inventory (extends inventory management)
- Update Book Information → Manage Book Inventory (extends inventory management)

- Checkout → View Cart (often starts from viewing cart)

Inheritance Relationships:

- Admin inherits authentication capabilities from base User actor
- Both Customer and Admin can perform Login/Logout

2.2.5 System Boundaries and Actors

Primary Actors:

- **Customer**: External user who purchases books
- **Admin**: Internal user who manages the system

Secondary Actors (External Systems):

- Payment Gateway (for processing payments)
- Email Service (for notifications)
- Inventory Management System (for stock updates)

System Boundary: The Online Bookstore System encompasses all use cases within the dashed boundary, representing the scope of the software system being developed.

2.3 API Documentation

Base URL: `http://localhost:5000` (Development) / `https://online-bookstore-yqif.onrender.com` (Production)
Interactive Documentation: Available at `/api-docs` endpoint with Swagger UI

2.3.1 User Endpoints

POST /api/users

- **Description:** Register a new user account

Request Body:
{ "name": "string", "email": "string", "password": "string"}

- **Response:** `{ success: boolean, message: string, user: object }`

POST /api/users/login

- **Description:** Authenticate user login

Request Body:
{ "email": "string", "password": "string"}

- **Response:** `{ success: boolean, message: string, token: string, user: object }`

2.3.2 Book Management Endpoints

GET /api/books

- **Description:** Retrieve all books
- **Response:** `{ books: array, count: number }`

POST /api/books

- **Description:** Add new book to inventory
- **Authentication:** Required (JWT token)

Request Body:
{ "title": "string", "author": "string", "price": "number", "description": "string", "stock": "number"}

-
- **Response:** `{ success: boolean, message: string, book: object }`

2.3.3 Shopping Cart Endpoints

POST /api/cart/add

- **Description:** Add book to cart (session-based)

Request Body:
{ "bookId": "string", "quantity": "number"}

-
- **Response:** `{ success: boolean, message: string, cart: object }`

DELETE /api/cart/remove/:bookId

- **Description:** Remove item from cart
- **Parameters:** `bookId` (path parameter)
- **Response:** `{ success: boolean, message: string, cart: object }`

PUT /api/cart/update/:bookId

- **Description:** Update item quantity in cart
- **Parameters:** `bookId` (path parameter)

Request Body: { "quantity": "number"}

- **Response:** `{ success: boolean, message: string, cart: object }`

DELETE /api/cart/clear

- **Description:** Clear entire cart
- **Response:** `{ success: boolean, message: string }`

- **GET /api/cart/summary**
- **Description:** Get cart items and totals
- **Response:** `{ items: array, totalPrice: number, itemCount: number }`

2.3.4 Order Management Endpoints

POST /api/orders

- **Description:** Place a new order
- **Authentication:** Required (JWT token)

Request Body:
{ "items": [{ "book": "bookId", "quantity": "number" }], "shippingAddress": "string"}

-
- **Response:** `{ success: boolean, message: string, order: object }`

GET /api/orders

- **Description:** Get all orders (admin functionality)
- **Authentication:** Required (JWT token)
- **Response:** `{ orders: array }`

GET /api/orders/user/:userId

- **Description:** Get orders for specific user
- **Authentication:** Required (JWT token)
- **Parameters:** `userId` (path parameter)
- **Response:** `{ orders: array }`

PUT /api/orders/:orderId/status

- **Description:** Update order status
- **Authentication:** Required (JWT token)
- **Parameters:** `orderId` (path parameter)

Request Body:
{ "status": "pending|processing|shipped|delivered|cancelled"}

- **Response:** `{ success: boolean, message: string, order: object }`

2.4 Database Schema

2.4.1 User Collection

{

_id: ObjectId,

name: String (required),

email: String (required, unique),

password: String (required, hashed with bcrypt),

createdAt: Date (default: Date.now),

updatedAt: Date (default: Date.now)

}

Indexes:

- `email` (unique index for authentication)

2.4.2 Book Collection

{

_id: ObjectId,

title: String (required),

author: String (required),

price: Number (required, min: 0),

description: String,

stock: Number (required, min: 0),

createdAt: Date (default: Date.now),

updatedAt: Date (default: Date.now)

}

Indexes:

- `title` (text index for search functionality)
- `author` (index for author-based queries)

2.4.3 Cart (In-Memory, Session-Based)

// Stored in server memory per session

```
{
  sessionId: String,
  items: [{
    book: Book (populated object),
    quantity: Number (required, min: 1)
  }],
  lastUpdated: Date
}
```

Note: Cart data is not persisted in the database. It exists only in server memory during the user session.

2.4.4 Order Collection

```
{
  _id: ObjectId,
  user: ObjectId (ref: 'User', required),
  items: [{
    book: ObjectId (ref: 'Book', required),
    quantity: Number (required, min: 1)
  }],
  totalPrice: Number (required, min: 0),
  shippingAddress: String (required),
  status: String (
    enum: ['pending', 'processing', 'shipped', 'delivered', 'cancelled'],
    default: 'pending'
  ),
  createdAt: Date (default: Date.now),
  updatedAt: Date (default: Date.now)
```

}

Indexes:

- `user` (index for user-specific order queries)
- `status` (index for status-based filtering)
- `createdAt` (index for date-based sorting)

2.5 Setup and Installation Instructions

2.5.1 Prerequisites

- **Node.js** (version 14.x or higher)
- **MongoDB** (local installation or MongoDB Atlas account)
- **npm** (Node Package Manager)
- **Git** (for version control)

2.5.2 Installation Steps

1. Clone the Repository

git clone https://github.com/Simeon-Azeh/online-bookstore.git

cd online-bookstore/backend

2. Install Dependencies

npm install

3. Environment Configuration Create a `.env` file in the `backend` folder:

PORT=5000

MONGODB_URI=your_mongodb_connection_string

JWT_SECRET=your_jwt_secret_key

NODE_ENV=development

4. Start the Application

Development mode (with auto-restart):

npm start

Or directly with Node.js:

node server.js

5. Access the Application

- **API Base URL:** http://localhost:5000
- **Interactive API Documentation:** http://localhost:5000/api-docs
- **Live Demo:** https://online-bookstore-yqif.onrender.com/api-docs/

2.5.3 Project Structure

```
online-bookstore/
├── backend/
│   ├── models/             # MongoDB models
│   │   ├── User.js
│   │   ├── Book.js
│   │   └── Order.js
│   ├── routes/             # API route handlers
│   │   ├── users.js
│   │   ├── books.js
│   │   ├── cart.js
│   │   └── orders.js
│   ├── middleware/         # Custom middleware
│   │   ├── auth.js
│   │   └── validation.js
│   ├── config/             # Configuration files
│   │   └── database.js
│   ├── swagger/            # API documentation
```

```
│   │   └── swagger.yaml
│   ├── .env                # Environment variables
│   ├── package.json
│   └── server.js           # Application entry point
├── frontend/               # Frontend files (optional)
│   ├── index.html
│   ├── css/
│   └── js/
└── README.md
```

2.5.4 Environment Variables

Variable	Description	Example
`PORT`	Server port number	`5000`
`MONGODB_URI`	MongoDB connection string	`mongodb://localhost:27017/bookstore`
`JWT_SECRET`	Secret key for JWT token signing	`your-secret-key-here`
`NODE_ENV`	Environment mode	`development` or `production`

2.5.5 Frontend Integration

If you have a frontend application:

Open the frontend files:

```
 # Navigate to frontend directory
cd ../frontend


# Open index.html in browser or use a local server
# Example with Python:
python -m http.server 3000
# Example with Node.js:
npx serve .
```

Configure API endpoints:

- Ensure the backend API is running at `http://localhost:5000`
- Update frontend JavaScript files to point to correct API endpoints
- Handle CORS if frontend and backend are on different ports

2.5.6 Testing the API

Using Swagger UI (Recommended):

1. Navigate to http://localhost:5000/api-docs
2. Use the interactive interface to test endpoints
3. Authenticate by registering a user and using the login endpoint

Using curl or Postman:

```
# Register a user

curl -X POST http://localhost:5000/api/users \

  -H "Content-Type: application/json" \

  -d '{"name":"John Doe","email":"john@example.com","password":"password123"}'



# Login

curl -X POST http://localhost:5000/api/users/login \

  -H "Content-Type: application/json" \

  -d '{"email":"john@example.com","password":"password123"}'



# Get all books

curl -X GET http://localhost:5000/api/books
```

2.5.7 Deployment

Development Deployment:

```
npm start
```

Production Deployment (Render/Heroku):

1. **Environment Setup:**
    - Set all environment variables in deployment dashboard
    - Use MongoDB Atlas for database (set MONGODB_URI)
    - Set NODE_ENV to "production"
2. **Security Considerations:**
    - Disable Swagger UI in production (check NODE_ENV)
    - Use strong JWT_SECRET
    - Enable HTTPS in production
    - Implement rate limiting for API endpoints
3. **Current Deployment:**
    - Live API: https://online-bookstore-yqif.onrender.com
    - Documentation: https://online-bookstore-yqif.onrender.com/api-docs/

2.5.8 Troubleshooting

Common Issues:

1. **MongoDB Connection Error:**
    - Verify MongoDB is running locally or check Atlas connection string
    - Ensure network access is configured for MongoDB Atlas
    - Check firewall settings
2. **JWT Authentication Issues:**
    - Verify JWT_SECRET is set in environment variables
    - Check token format in Authorization header: `Bearer <token>`
    - Ensure token hasn't expired
3. **CORS Issues:**
    - CORS is configured in the application
    - Check if frontend and backend URLs are correct
    - Verify allowed origins in CORS configuration
4. **Port Conflicts:**
    - Change PORT in .env file if 5000 is occupied
    - Kill existing processes: `lsof -ti:5000 | xargs kill -9`

Support Resources:

- GitHub Repository: https://github.com/Simeon-Azeh/online-bookstore
- API Documentation: Available at `/api-docs` endpoint
- Demo Video: https://youtu.be/cb7grv0Z2NE