Project Overview

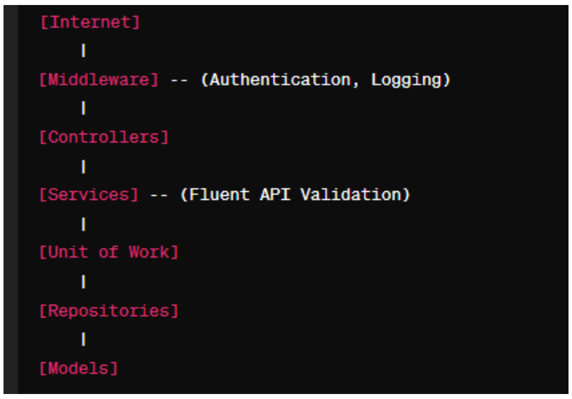
The BookRenter application is under development, aiming to provide a platform for renting books. As a developer, my role involves developing RESTful APIs for specific functionalities.

Project Architecture Overview:

In the ongoing process of refactoring the codebase, the project architecture is being reorganized to adhere to common application development patterns. The primary patterns being implemented are the Repository Pattern and the Service Layer Pattern. The current structure of the application layers is as follows:

1. **Controllers:**
    - The Controllers layer serves as the entry point for handling user requests and contains application logic.
    - Responsibilities include accepting user input data, validating input, and passing data to the corresponding service layer.
2. **Services:**
    - Act as the bridge between Controllers and Repositories, orchestrating data validation, business logic, and data persistence operations.
    - Fluent API Validation Integration: Services now leverage Fluent API Validation for comprehensive data validation and business rule enforcement. This integration allows for more readable, maintainable, and chainable validation logic that is executed before any business logic or database interactions.
3. **Unit of Work:**
    - The Unit of Work component serves as a cohesive unit comprising repositories, facilitating transactions and ensuring data consistency.
    - Responsibilities include managing the lifecycle of repository instances and coordinating multiple repository operations within a single transaction context.
4. **Repositories:**
    - The Repositories layer is responsible for interacting with database models and executing database operations.
    - Each repository is associated with a specific database table, encapsulating data access logic and providing methods for CRUD (Create, Read, Update, Delete) operations.
5. **Models:**
    - The Models layer consists of Laravel model files representing database tables and defining relationships between entities.

- Each model represents a distinct database table and encapsulates data access and manipulation methods.



```
[Internet]
    |
[Middleware] -- (Authentication, Logging)
    |
[Controllers]
    |
[Services] -- (Fluent API Validation)
    |
[Unit of Work]
    |
[Repositories]
    |
[Models]
```

Observations and Granularity:

Upon reviewing the architecture, the granularity of the layers is currently aligned in a 1:1 relationship as follows:

- Each database table corresponds to a single model, reflecting the entity structure within the database.
- Each model is associated with a dedicated repository, encapsulating data access logic specific to that entity.
- Every repository is linked to a Unit of Work, ensuring transactional integrity and coordination among related repository operations.
- Each Unit of Work corresponds to a service, orchestrating business logic operations and interacting with repositories.
- Finally, each service is associated with a controller, establishing a direct mapping between service methods and controller actions.

Endpoint Details

- **POST /Auth/login:** Provides authentication for users to log in.
- **GET /api/Book/{id}:** Retrieves information about a specific book by its ID.
- **PUT /api/Book/{id}:** Updates information about a specific book.
- **GET /api/SearchBook:** Allows users to search for books based on a search term.
- **GET /api/Book:** Retrieves a list of all available books.
- **POST /api/Book:** Allows the addition of a new book to the system.
- **GET /api/Cart:** Retrieves the list of items in the user's shopping cart.
- **POST /api/Cart:** Adds a book to the user's shopping cart.
- **DELETE /api/Cart/{bookId}:** Removes a specific book from the user's shopping cart.
- **GET /api/Inventory/{bookId}:** Retrieves inventory information for a specific book.

- **GET /api/Inventory:** Retrieves a list of all available inventory items.
- **POST /api/Inventory:** Adds a new inventory item to the system.
- **POST /User:** Allows the registration of a new user(**Just added to testing purpose of different roles)**

Functionality Details

- **Search a Book:** Enables users to search for a book by its name or author.
- **Add a Book to Cart:** Allows adding a book to the cart with specific conditions and error handling.
- **Checkout Books in Cart:** Describes the checkout process and actions taken upon successful checkout.

Unit Testing

Additionally, thorough testing coverage has been ensured by implementing unit tests for the APIs, repositories, and services using xUnit. These tests validate the functionality and behaviour of the system components, ensuring robustness and reliability throughout the development process.

Technologies Used

- ASP.NET 8 WebApi
- .NET 8
- MSSQL
- Jwt-based authentication and role-based authorization
- Default logging by dotnet
- Swagger OpenAPI
- Health Checks
- Fluent Api
- XUnit

Features

- Clean Code Architecture
- RESTful Design
- Entity Framework Core - db First
- Repository Pattern - Generic

- Unit of Work
- Identity with JWT Authentication
- Role-based Authorization
- Custom Exception Handling Middlewares
- Swagger UI
- Operator Overloading (Instead of AutoMapper)
- Fluent Validation
- Unit Testing

Project Structure

- Middlewares
- Controllers
- Services
- Repositories
- Entity
- Tests

To set up and run the BookRenter project, follow these steps:

1. **Clone the Project:**
    - Clone the BookRenter project repository from the GitHub to your local machine. You can find the project on GitHub at the following URL: BookRenter GitHub Repository.

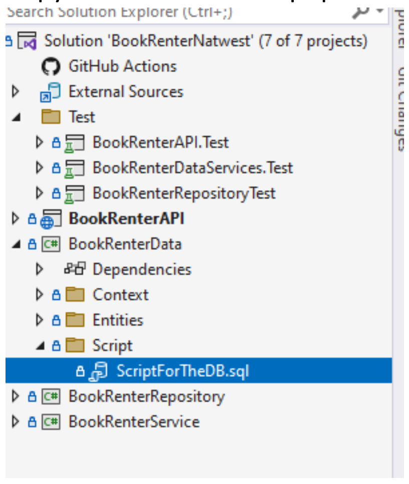
2. **Open the Project:**
    - Open the project directory in your preferred Integrated Development Environment (IDE) like Visual Studio or Visual Studio Code.
3. **Update Connection String:**
    - Navigate to the "appsettings.json" file in the project directory.
    - Locate the "ConnectionStrings" section.
    - Replace the existing connection string with the provided connection string:

"ConnectionStrings": { "BookRenterDatabase": "Server=LAPTOP-3KRDMGLE;Database=BookRenterDb;User Id=sa;Password=Admin@123;Encrypt=True;TrustServerCertificate=True;" }

- Replace **<Your_Server_Name>** with your SQL Server instance name.

4. **Execute the SQL Script:**

    - Open your preferred SQL Server management tool (e.g., SQL Server Management Studio).
    - Copy the entire SQL script provided for database setup.



Search Solution Explorer (Ctrl+;)

Solution 'BookRenterNatwest' (7 of 7 projects)
- GitHub Actions
- External Sources
- Test
  - BookRenterAPI.Test
  - BookRenterDataServices.Test
  - BookRenterRepositoryTest
- BookRenterAPI
- BookRenterData
  - Dependencies
  - Context
  - Entities
  - Script
    - ScriptForTheDB.sql
- BookRenterRepository
- BookRenterService

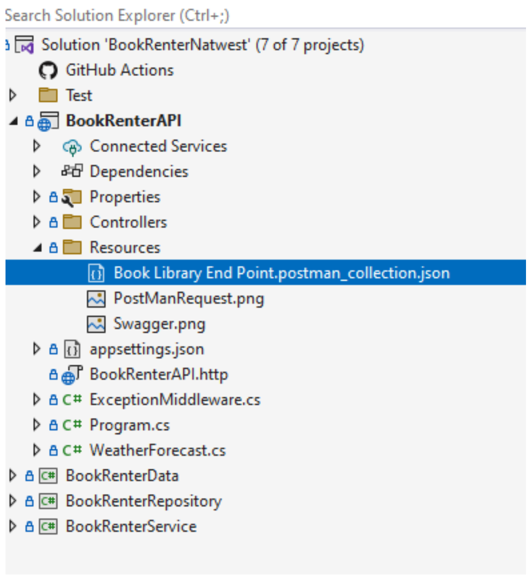
    - Execute the script in the management tool to create the database schema, populate it with initial data, and establish foreign key constraints.
    - Ensure that the script runs successfully without any errors.
    - I set up the entire database by running a script that creates tables, inserts initial data, and establishes foreign key constraints. I chose not to use Entity Framework migration for this because the process of seeding the data is quite complex. This setup simplifies running the project for you..

5. **Run the Project:**

    - Build the solution in your IDE to ensure all dependencies are resolved.
    - Start the BookRenter application by running the project from your IDE.
    - Verify that the application starts without any errors.

6. **Test Endpoints with Postman:**

    - Import the provided Postman request JSON file into your Postman application.



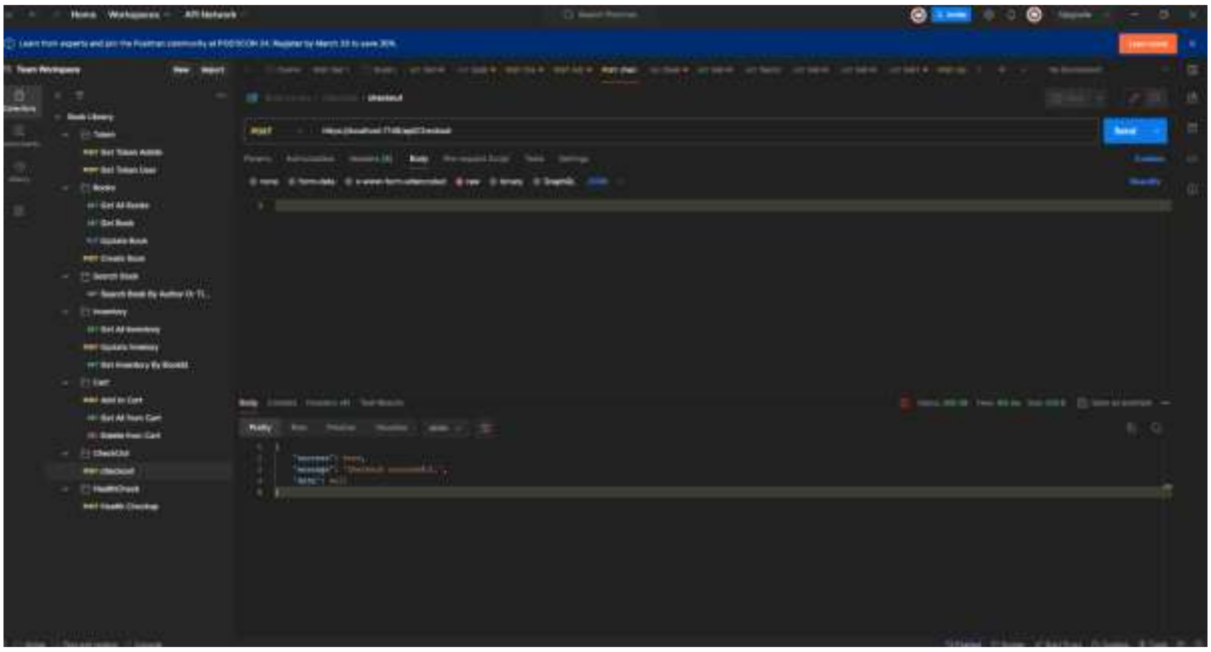
- Use the imported requests to test various endpoints of the BookRenter application.
- Verify that the endpoints function correctly and return expected responses.

In the application, there are two types of user accounts: admin and regular user. Admin users have full access to all features and functionalities within the app. On the other hand, regular users have limited capabilities, allowing them to search for books, manage their cart, and proceed with checkout.

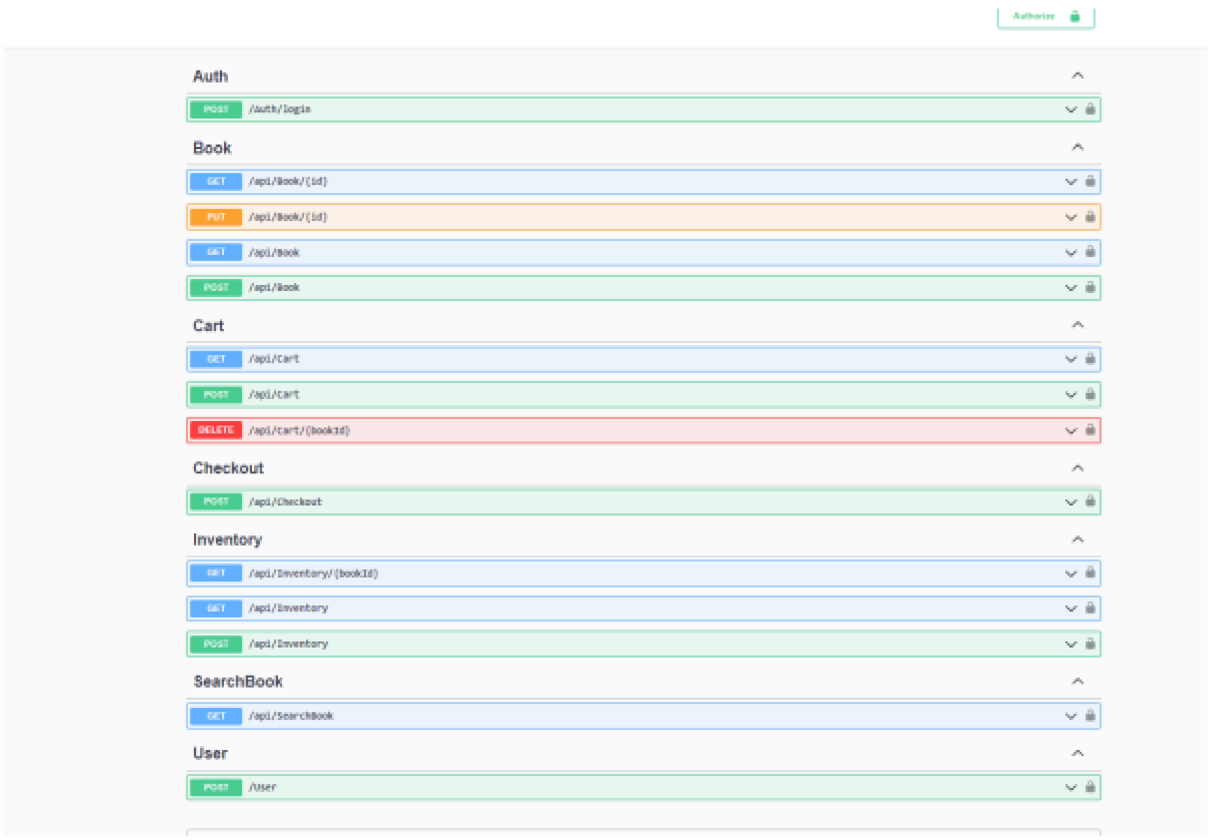
For testing or demonstration, use the following credentials:

- Administrator Access: Username: admin Password: test
- Regular User Access: Username: user Password: user

By following these steps, you'll successfully set up and run the BookRenter project, execute the database script, and configure the connection string for database access. Additionally, you'll be able to test the endpoints using the provided Postman requests.



```
{
    "success": true,
    "message": "Checkout successful.",
    "data": null
}
```



Auth

POST /auth/login

Book

GET /api/Book/{id}

PUT /api/Book/{id}

GET /api/Book

POST /api/Book

Cart

GET /api/Cart

POST /api/Cart

DELETE /api/Cart/{bookId}

Checkout

POST /api/Checkout

Inventory

GET /api/Inventory/{bookId}

GET /api/Inventory

POST /api/Inventory

SearchBook

GET /api/SearchBook

User

POST /user