**Work Integrated Learning Programmer**

**M.Tech Software Engineering**



**SCALABLE SERVICE**

**Submitted by**

**Raghunath Reddy G**

**2023MT93051**

**2023mt93051@wilp.bits-pilani.ac.in**

Contents

[1. **BOOK EXCHANGE PLATFORM** 3](#_Toc183268558)

[**2. *Key Features:*** 3](#_Toc183268559)

[3. **Video Explanation** 4](#_Toc183268560)

[4***. Architecture design:*** 4](#_Toc183268561)

[***5. Each Micro service is internal design shown below (user service, Book services)*** 4](#_Toc183268562)

[***6. Prerequisites:-*** 5](#_Toc183268563)

[***7. Tech-stack used:*** 6](#_Toc183268564)

[***8. Flow diagram:*** 6](#_Toc183268565)

[***9. Database Design:*** 7](#_Toc183268566)

[***10. MS-SQL Database design table details*** 7](#_Toc183268567)

[***11. Postgres Database design table details for storing the book-transactions*** 8](#_Toc183268568)

[***12. Backend API implementation endpoint.*** 8](#_Toc183268569)

[***12.1 Swagger screen of the “user service”*** 9](#_Toc183268570)

[***12.2 UserService Code Repository*** 9](#_Toc183268571)

[***12.3 Backend API Book Management Service endpoints*** 9](#_Toc183268572)

[***12.4 Book-Service Code Repository*** 11](#_Toc183268573)

[***12.5 Transaction-Service Code Repository*** 12](#_Toc183268574)

[***13.API- Gateway: -*** 12](#_Toc183268575)

[***13.1 Gateway Code Repository*** 12](#_Toc183268576)

[***14. Docker Deployment*** 13](#_Toc183268577)

[***15. Steps to create Docker*** 14](#_Toc183268578)

[***10 .Frontend React application screen shot:-*** 15](#_Toc183268579)

[***10.1 Code structure:*** 15](#_Toc183268580)

[***11. Application working Screenshots*** 15](#_Toc183268581)

[**12. GitHub Repositories** (**https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development**) 20](#_Toc183268582)

[***12.1. github Backend services:-*** 20](#_Toc183268583)

[***12.2 github Frontend React application:-*** 20](#_Toc183268584)

[**13. Demonstration Video** 20](#_Toc183268585)

[***13.1 Demo video uploaded in YouTube*** 20](#_Toc183268586)

# 1. **BOOK EXCHANGE PLATFORM**

Book lovers frequently accumulate a collection of books they have read and look for other recommendations. They are always eager to explore new reading material. Traditional methods of exchanging books, such as local book swaps or lending among friends, are limited in scope and accessibility. Therefore, it is imperative to have a digital platform that can facilitate book exchanges on a larger scale. This platform should connect users with similar reading interests, enabling them to trade books easily and efficiently. This project aims to develop a full-stack web application that serves as a centralized platform for users to exchange, lend, and borrow books with other users. The platform should provide a user-friendly interface, robust search and recommendation features, and secure transaction capabilities.

Solution:

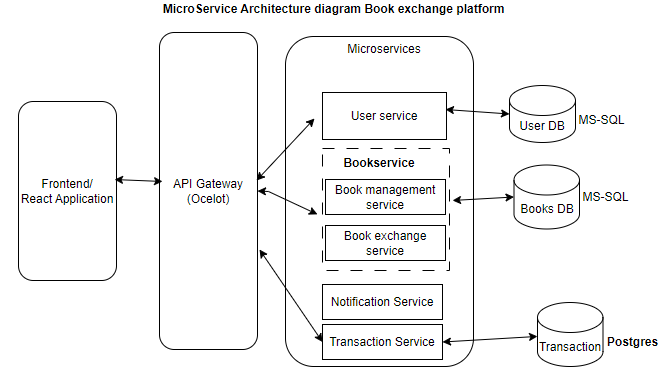
The implementation of microservices architecture is underscored by the individual deployment of each service within Docker containers, emphasizing a modular and scalable approach to application development

**Ocelot** has been used to integrate the **API Gateway application** into the system. By serving as a common point of entry, this gateway makes it easier for the many microservices to communicate with one another. Ocelot's integration improves security, simplifies the administration of service interactions, and gives external clients a single interface.

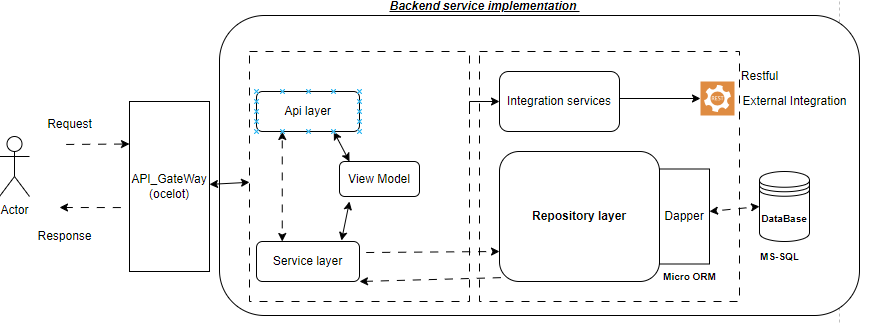
# **2. *Key Features:***

* **User Authentication**: Implement a secure user authentication system to allow users to register, log in, and manage their accounts.
* **Book Listing**: Enable users to list books they want to exchange or lend, including details such as title, author, genre, condition, and availability status.
* **Book Search:** Provide users with advanced search and filtering options to discover books based on criteria like genre, author, title, location, and availability.
* **Exchange Requests**: Allow users to send and receive exchange requests for specific books, including negotiation options for terms such as delivery method and duration.
* **Messaging System**: Implement a messaging system to facilitate user communication regarding book exchanges, including negotiation details, logistics, and scheduling. (Mock API s can be used)
* **User Profiles**: Enable users to create profiles with information about their reading preferences, favorite genres, and books they currently own or wish to acquire.
* **Transaction Management**: Provide tools for users to track the status of their exchange transactions, including pending requests, accepted exchanges, and completed transactions.

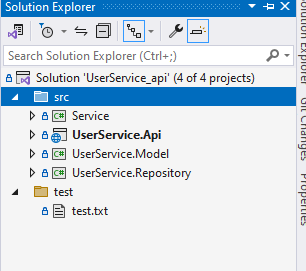
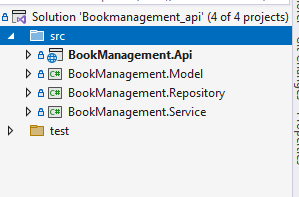
# 3. **Video Explanation**

4***. Architecture design:*** ***Book exchange platform*** 

# ***5. Each Micro service is internal design shown below (user service, Book services)***



* ***Backend api code snippet in n-layer architecture***



These microserives are *designed based on* ***business domain context & isolated within the bounder***. Each services is communicated via messaging or direct service to service call using restful template.

* **API layer:** will have the Http-Methods like Http GET,POST PUT,DELETE which is exposed as via swagger to consume
* **Service Layer: -** will contain the business logic implemented w.r.t to domain. And also it bind the data from DTO-models to Domain-Data-models.
* **Repository layer**: will have DB connection logic & all DB query and Integrated to Dapper-ORM which will connect to database and load the data to data models.
* **Model layer**: will have the POCO classes or Domain model classes which is replica of the database table which is used to bind the data from table.

# ***6. Prerequisites:-***

Below are the tools used for developing the microservice application

**1. Visual Studio Code**

Visual Studio Code (VS Code) serves as the integrated development environment (IDE) for building and managing the application code.

Downloaded and installed VS Code from https://code.visualstudio.com/.

**2. MS- SQL :Microsoft SQL**

MS SQL is relational database management system (RDBMS) used to store a data & blob images

Downloaded and installed <https://www.microsoft.com/en-in/sql-server/sql-server-downloads>

**3. PostgreSQL**

PostgreSQL is the relational database management system (RDBMS) chosen for the microservices.

Downloaded and installed PostgreSQL from https://www.postgresql.org/download/windows/.

Ensured that the PostgreSQL service is running after installation.

**4. Docker Desktop**

Docker desktop provides containerization for the microservices, helping in deployment and management.

Installed Docker Desktop from https://www.docker.com/products/docker-desktop.

Docker will enable the encapsulation of each microservice in a container for consistent and reproducible deployment.

**5. .NET 6 SDK**

The .NET 6 Software Development Kit (SDK) is crucial for building and running our .NET-based microservices.

Downloaded and installed .NET 8 SDK from https://dotnet.microsoft.com/en-us/download/dotnet/8.0.

**6. Postman Desktop**

Postman is a powerful tool for testing APIs and streamlining the development workflow.

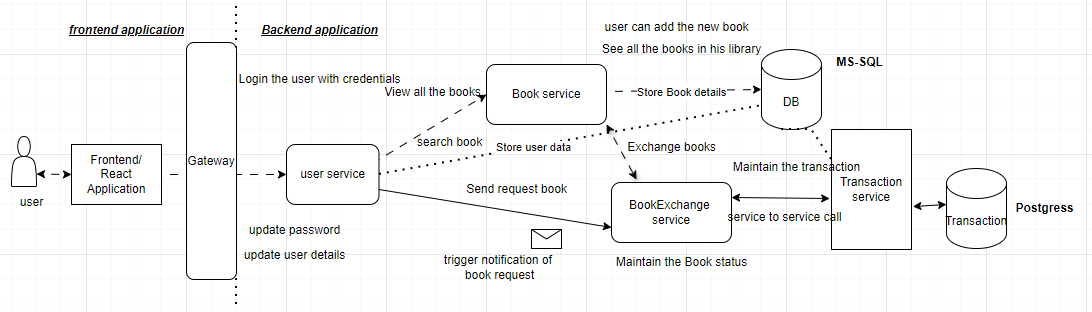
Downloaded and installed Postman Desktop from <https://www.postman.com/downloads/>.

# ***7. Tech-stack used:***

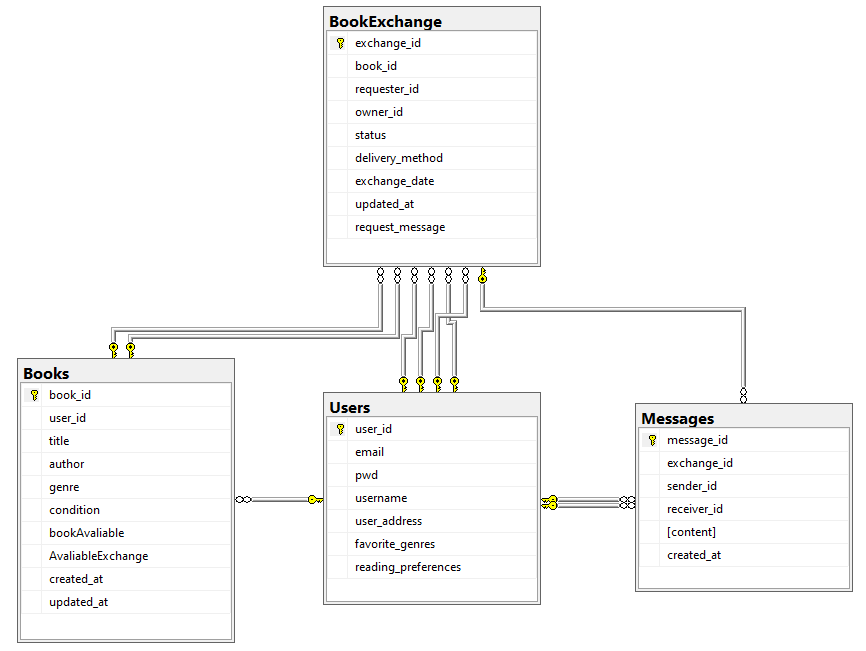
***Back-End stack: -***

* ***.NET core:***  .***Net core restful API*** service is user which is used to build the back business logic and expose it as restful service it will connect to database and pull the data using sql queries.
* ***MS SQL Database*** : is used as database to store the application user data & book exchange data , this is relational database stores the data in form of rows & columns’
* ***Postures DB*** : used to save the Transaction of the Book Exchange history
* ***Dapper***: it is ***MICRO*** ORM which connect the repository & Data model and binds the data from SQL DB to Data model.
* **API Gateway** : ocelot api gateway is used a api discovery & rate limiter and it will act a entry point for external request to api

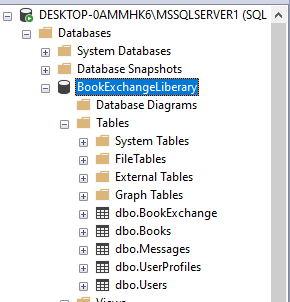
# ***8. Flow diagram:***



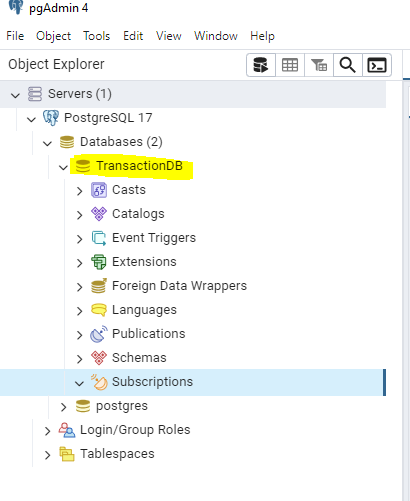
# ***9. Database Design:***



# ***10. MS-SQL Database design table details***



# ***11. Postgres Database design table details for storing the book-transactions***



# ***12. Backend API implementation endpoint.***

* ***User Story 1: User Authentication***

As a user, I want to securely register, log in, and manage my account, So that I can access and use the book exchange platform.

Acceptance Criteria:

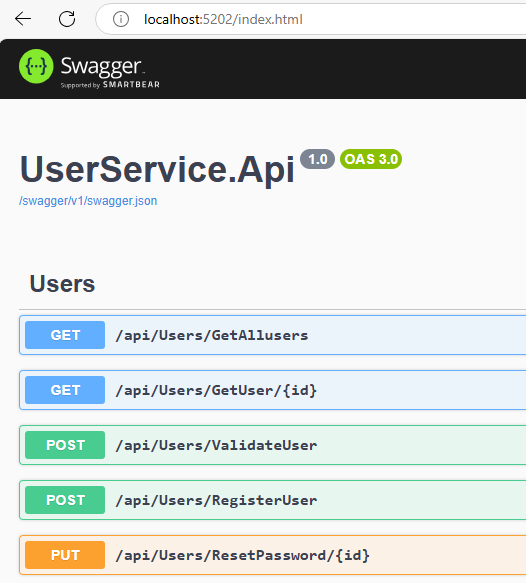
The platform must allow users to register with a valid email and password.

Passwords must be stored securely using encryption.

Users should be able to reset their password via a password recovery system.

Users should be able to log out from their account.

|  |  |  |  |
| --- | --- | --- | --- |
| **Entity** | **Http Method** | **URI** | **Description** |
| Users | GET | **/api/Users/GetAllusers** | **Get all the users list** |
| GET | **/api/Users/GetAllusers/${userid}** | **Search the User by ID** |
| POST | **/api/Users/RegisterUser** | **Add the new User** |
| PUT | **/api/users/resetPassword/ ${id}** | **Update the user password w.r.t userID** |
| DELETE | **/api/users/${id}** | **Deleted the user** |

***12.1 Swagger screen of the “user service”*** 

# ***12.2 UserService Code Repository***

The source code for the User Microservice is available on GitHub. You can access the code repository from the below link:

GitHub Repository: [UserMicroservice](https://github.com/raghunathreddy/user-service-api) (https://github.com/raghunathreddy/user-service-api)

# ***12.3 Backend API Book Management Service endpoints***

* ***User Story 2: Book Listing***

As a user, I want to list books that I want to exchange or lend, So that others can browse and request the books I offer.

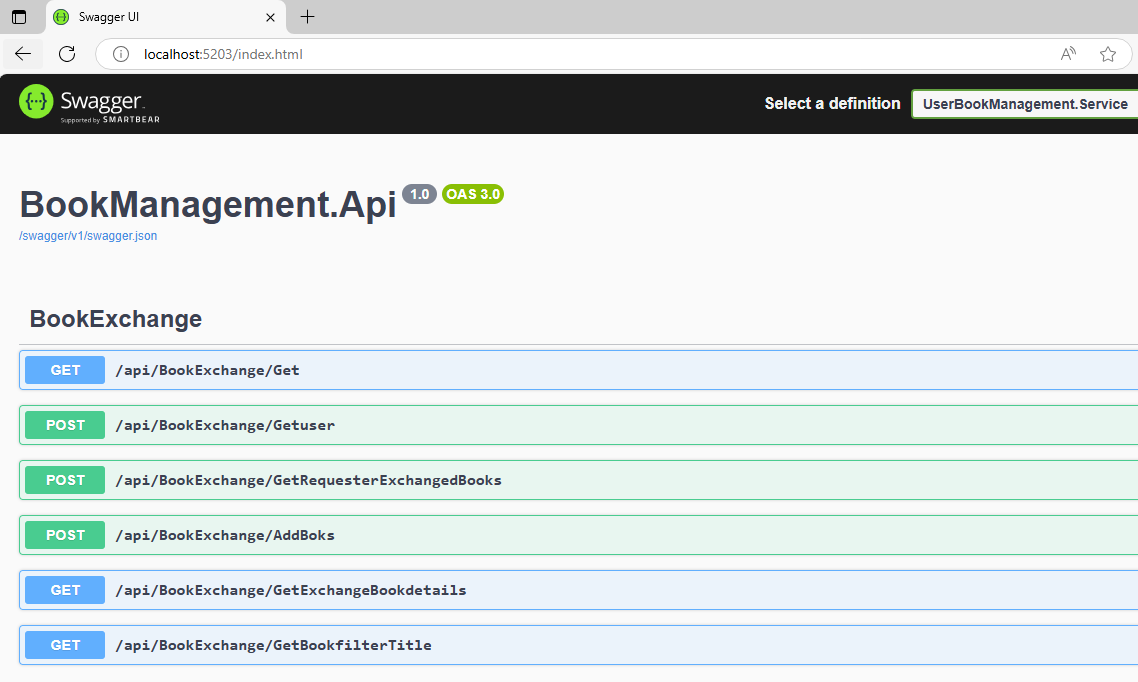
Acceptance Criteria:

Users should be able to add a book to their list by providing details such as title, author, genre, condition, and availability status.

Each book listing must have a unique ID associated with a user’s profile.

Users should be able to edit or delete book listings at any time.

The book listing must be displayed in the user's profile and searchable by others.



|  |  |  |  |
| --- | --- | --- | --- |
| **Entity** | **Http Method** | **URI** | **Description** |
| **ExchangeBook**  **Service** | GET | **/api/BookEchange/Get** | **Get all the Books for Exchange** |
| POST | **/api/BookEchange/GetUser** | **Search the Books available w.r.t UserID** |
| POST | **/api/ BookEchange /AddBooks** | **Add the New books for available for Exchange** |
| GET | /api/ BookEchange /GetBookFilterByTitle | **Search the Books available for Exchange w.r.t Title** |
| POST | **/api/ BookEchange /GetRequestExchangeBooks** | **Raise a request for books exchange** |

**User Story 3: Book Search**

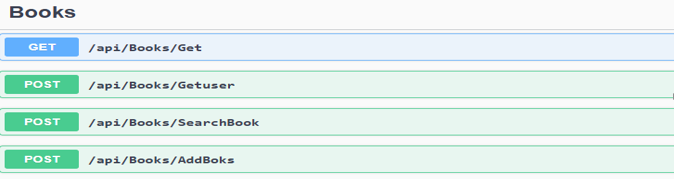
As a user, I want to search for books based on criteria such as title, author, genre, and location,

Acceptance Criteria:

The platform must provide a search bar where users can enter keywords like title, author, or genre.

The platform should allow users to filter search results by availability status, genre, and location.

Users must be able to view detailed information about a book (title, author, condition, etc.) when clicking on a search result. The search results should be paginated or load incrementally to handle large datasets.



|  |  |  |  |
| --- | --- | --- | --- |
| **Entity** | **Http Method** | **URI** | **Description** |
| Books  Service | GET | **/api/Books/Get** | **Get all the Books in library /store** |
| POST | **/api/ Books /GetUser** | **Search the Books available w.r.t UserID** |
| POST | **/api/ Books /SearchBook** | **Search book w.r.t title , gener,** author |
| POST | **/api/ Books /AddBooks** | **Add the book in this list** |

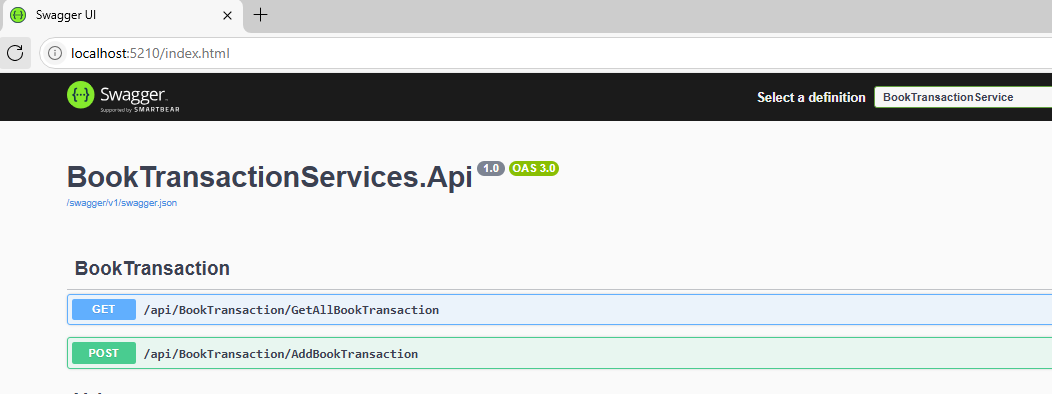
# ***12.4 Book-Service Code Repository***

The source code for the BookExcahnage Microservice is available on GitHub. You can access the code repository from the below link:

GitHub Repository: [Book-Microservice](https://github.com/raghunathreddy/book-service) (https://github.com/raghunathreddy/book-service)

***Transaction Services :-*** this service store the data of the transaction of the Book Exchanged between the users , this transaction history is used for analytics

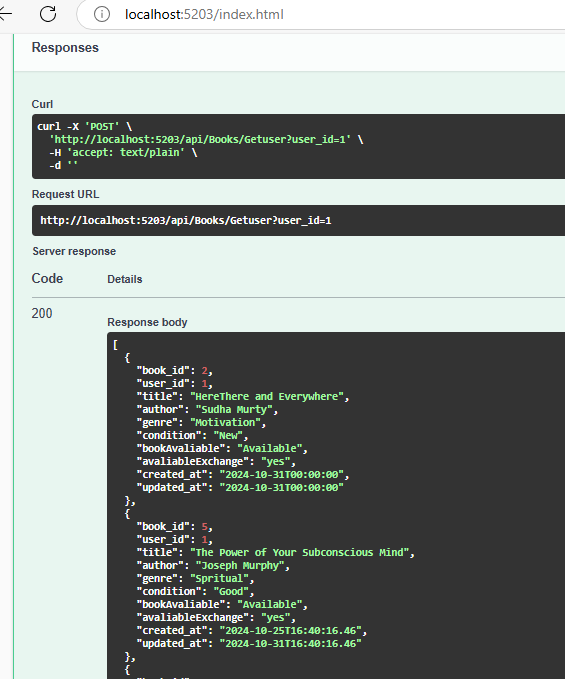
***Transaction-Service api:-***



# ***12.5 Transaction-Service Code Repository***

GitHubRepository: <Transaction-Microservice> (<https://github.com/raghunathreddy/transaction-service>)

***13.2 Below are the books API Request /response example***



# ***13.API- Gateway: -***

Gateway is software pattern that sits in front of an application programming interface (API) or group of microservices to facilitate requests and delivery of data and services, all the api services i.e. user service, book-service and Transaction-service are accessed via api-gateway

***Ocelot*** *is an open-source API Gateway for .NET* It offers a straightforward method for managing, aggregating, and securing APIs and is made to operate with the.NET Core platform. Ocelot allows communication between microservices in a distributed architecture and serves as a central point of entry for receiving HTTP requests.

***Ocelot offers following features:-***

1. ***Simple to Set Up and Use*:** Ocelot is easy to configure, especially if you’re already familiar with ASP.NET Core. The configuration is typically done through a ocelot.json file where you define routing, authentication, and other settings.

**2.** ***Routing:-*** Ocelot provides advanced routing capabilities such as path matching, request transformation, and routing to different services based on paths or headers.

*3.* ***Load Balancing****:* Ocelot can automatically distribute requests across multiple instances of a service, enabling load balancing for high availability and scalability.

***4. Request Aggregation****:* You can aggregate multiple backend service responses into a single response. This is useful in microservices environments where the client needs data from multiple services.

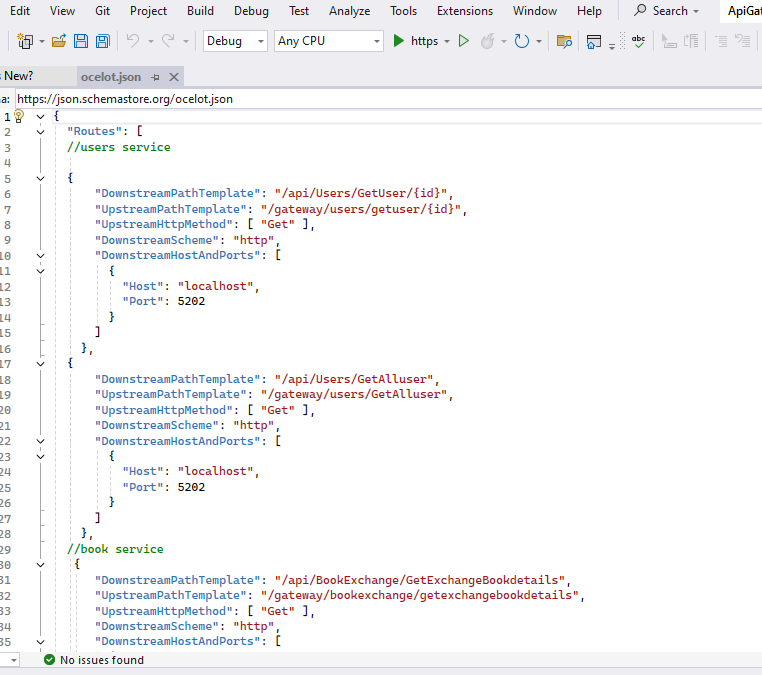
***5. Rate Limiting:*** Built-in support for controlling the rate of requests to microservices.

# ***13.1 Gateway Code Repository***

The source code for the Api-Gateway is available on GitHub. You can access the code repository from the below link:

GitHub Repository: [API-Gateway](https://github.com/raghunathreddy/api-gateway-ocelot/tree/main/ApiGateway) (https://github.com/raghunathreddy/api-gateway-ocelot/tree/main/ApiGateway)

***Gateway Mapping File***

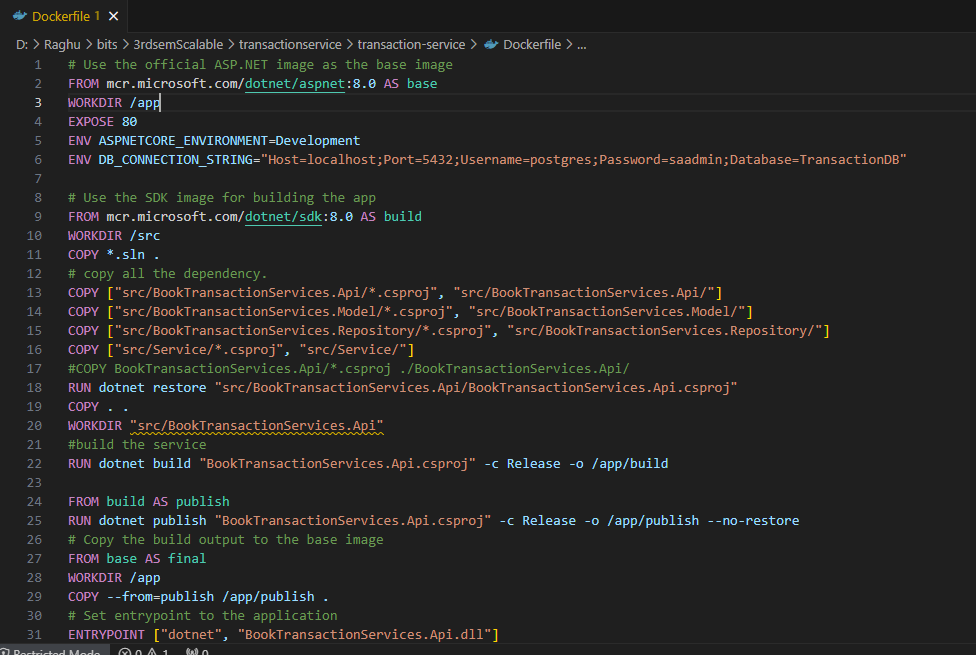


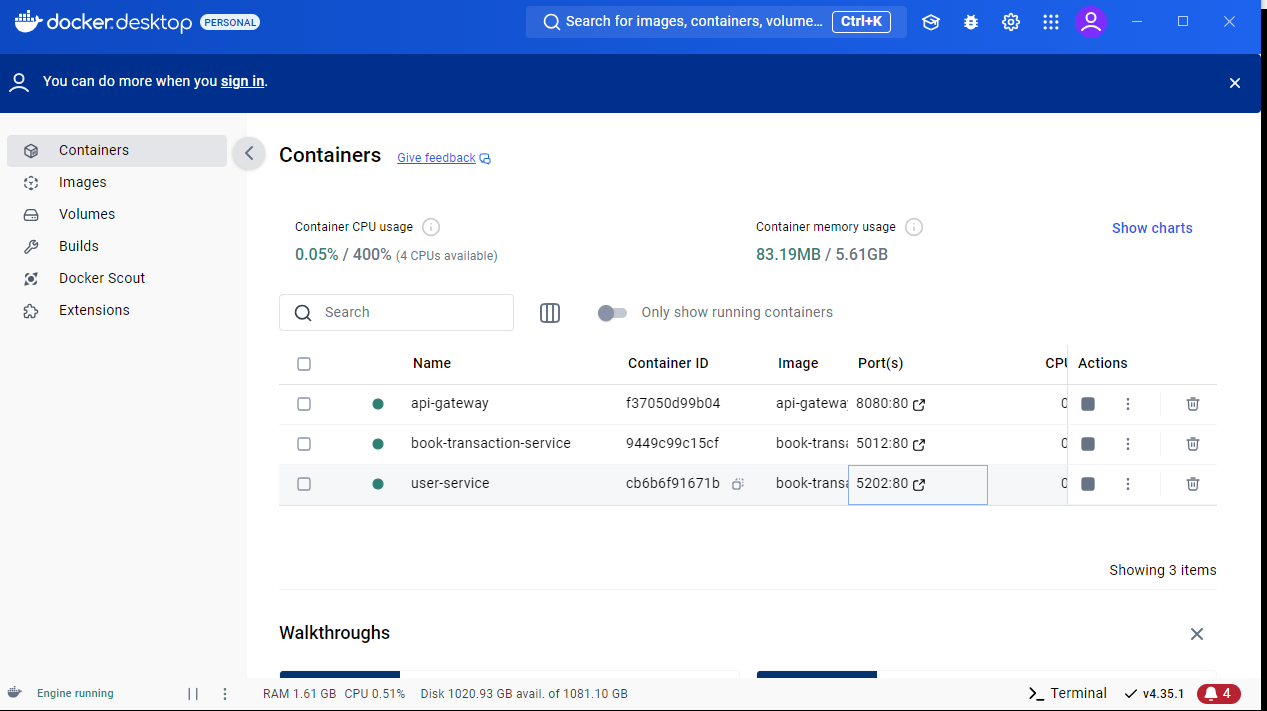
# ***14. Docker Deployment***

Docker provides Applications and their dependencies can be packaged into containers using a standardized and lightweight method offered by Docker. By encapsulating the runtime environment, these containers guarantee consistency across the development, testing, and production phases.   
Docker is essential to enabling a seamless and effective deployment process in our.NET 6 e-commerce microservices application.   
Among the main benefits of adopting Docker are   
**1. Consistency**: Make sure that the testing, development, and production environments are all the same.   
**2. Isolation**: To avoid dependency conflicts, execute microservices in separate containers.   
**3. Scaling**: Adapt to demand by smoothly scaling individual services.   
**4. Streamlined Integration**: Use Docker Compose or Kubernetes to make it easier to integrate heterogeneous microservices.   
**5. Portability**: The ability to move between environments is made possible via containers.

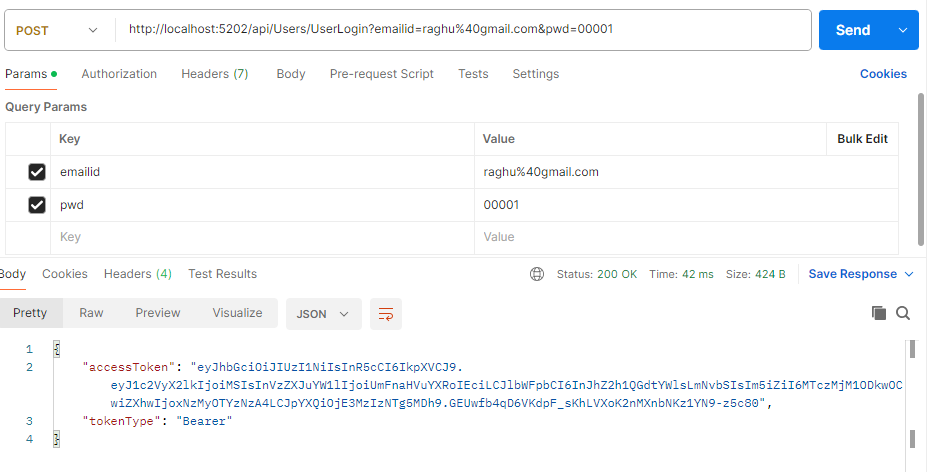
# ***15. Steps to create Docker***

For each microservice, User, Book-Service, and Transaction, create a Dockerfile specifying the necessary dependencies and configurations.





# ***Jwt Authentication***



# **12. GitHub Repositories** ([**https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development**](https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development))

# ***12.1. github Backend services:-***

* ***UserService API*** *:*-

[***https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development/UserService\_api***](https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development/UserService_api)

* ***BookService API:-***

[***https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development/Bookmanagement\_api***](https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development/Bookmanagement_api)

# ***12.2 github Frontend React application:-***

* ***React Application:-***[***https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development/frontend/book-exchange-liberary/client***](https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development/frontend/book-exchange-liberary/client)

# **13. Demonstration Video**

* [***https://github.com/raghunathreddy/BITS-Software-Engineering/blob/main/3rdsem/fullstack-development/document/FSAD\_Raghunath\_2023MT93051.mp4***](https://github.com/raghunathreddy/BITS-Software-Engineering/blob/main/3rdsem/fullstack-development/document/FSAD_Raghunath_2023MT93051.mp4)

# ***13.1 Demo video uploaded in YouTube***

***https://www.youtube.com/watch?v=rNr2a8\_wAAI***

**14. Documentation & Architecture diagram:-**

* [***https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development/document***](https://github.com/raghunathreddy/BITS-Software-Engineering/tree/main/3rdsem/fullstack-development/document)