MICROSERVICES WITH SPRINGBOOT

By TAKAM TALLA LOIQUE DARIOS

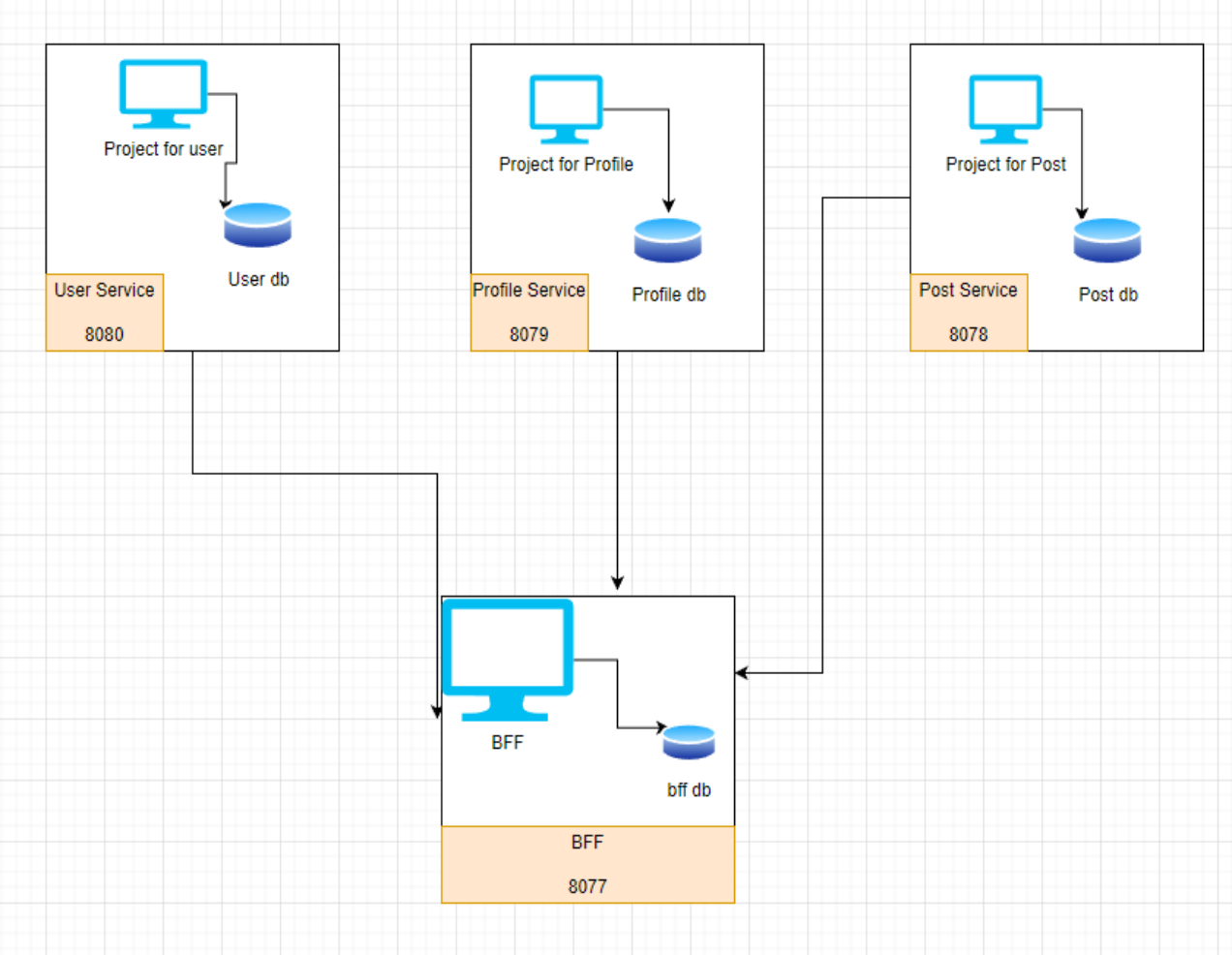
LINKEDINCLONE MICROSERVICE

In the LinkedIn clone project, the goal was to embrace the microservices architecture using Spring Boot. The idea was to decompose a monolithic system into smaller, manageable, and independently deployable services. Each microservice handles a specific piece of functionality within the platform, operating under its own database context to ensure loose coupling. This approach facilitates easier maintenance, better scalability, and allows for continuous deployment practices, enhancing the development process and product quality overall.

You are invited to read through this documentation so as to understand the project and it will also be important to read the project source codes which is found in the root of this project , immediately in the src folder.

Diagram

The diagram below visually represents the microservices architecture. It depicts each service as a separate module, with the BFF acting as the central node through which the frontend interacts with the backend services:



-The User Service connects to its dedicated user database and has an entry point exposed on port 8080.

-The Profile Service operates similarly with its own database, available on port 8079.

-The Post Service manages its datastore and is open on port 8081.

-The BFF Service aggregates the functionalities and provides a singular point of communication on port 8077, interfacing with the dedicated BFF database.

Architecture

In my architecture, I employed the **Model-View-Controller** (MVC) framework within each microservice to create a clear separation of concerns. I organized my code into several packages:

- The **config** package contains my configuration classes, which set up my application's settings, including security configurations with **SecurityConfig**.

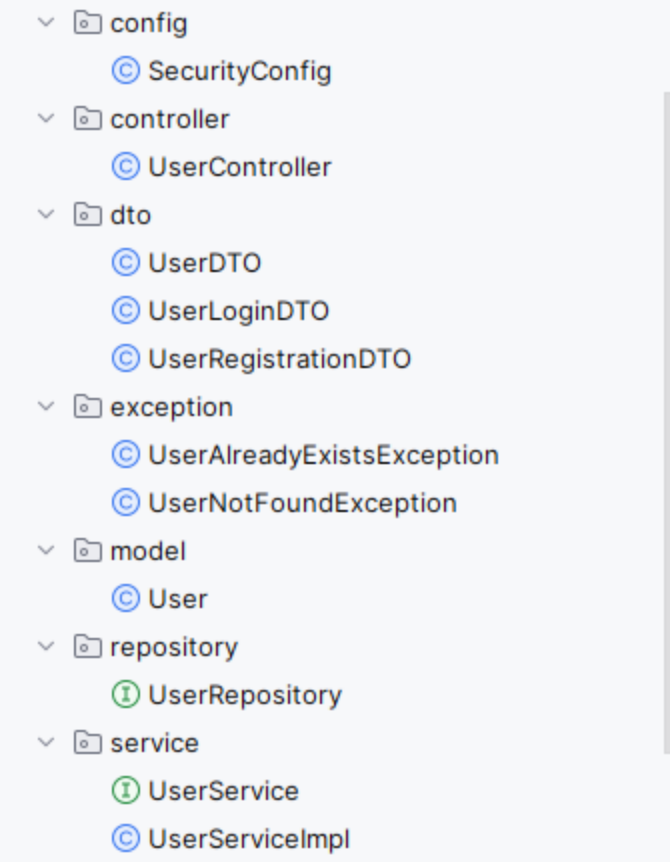
- In the **controller** package, I placed my controller classes, which act as the intermediaries handling all the incoming HTTP requests and responding accordingly.

- The **service** package houses my business logic. Here, services are defined to handle the core operations and business rules of my application.

- I used the **model** package to define my domain entities that represent the data structure within my databases.

- The **dto** (Data Transfer Object) package is where I structured the data that I send across the network. This is crucial for encapsulating the data in my HTTP requests and responses.

- Lastly, the **repository** package is where I defined my Spring Data JPA repositories. These interfaces are responsible for data persistence operations, allowing for a clean abstraction of the data layer.



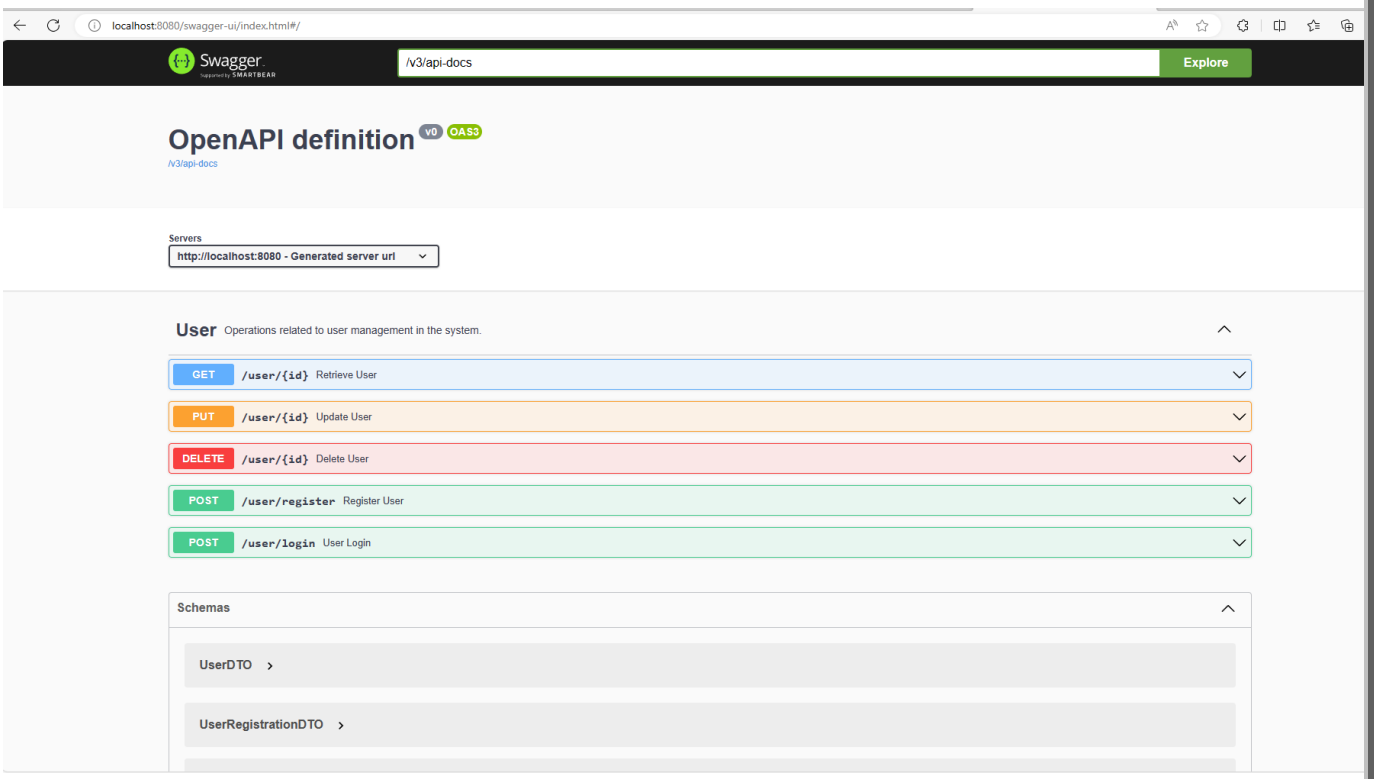
Swagger-Doc

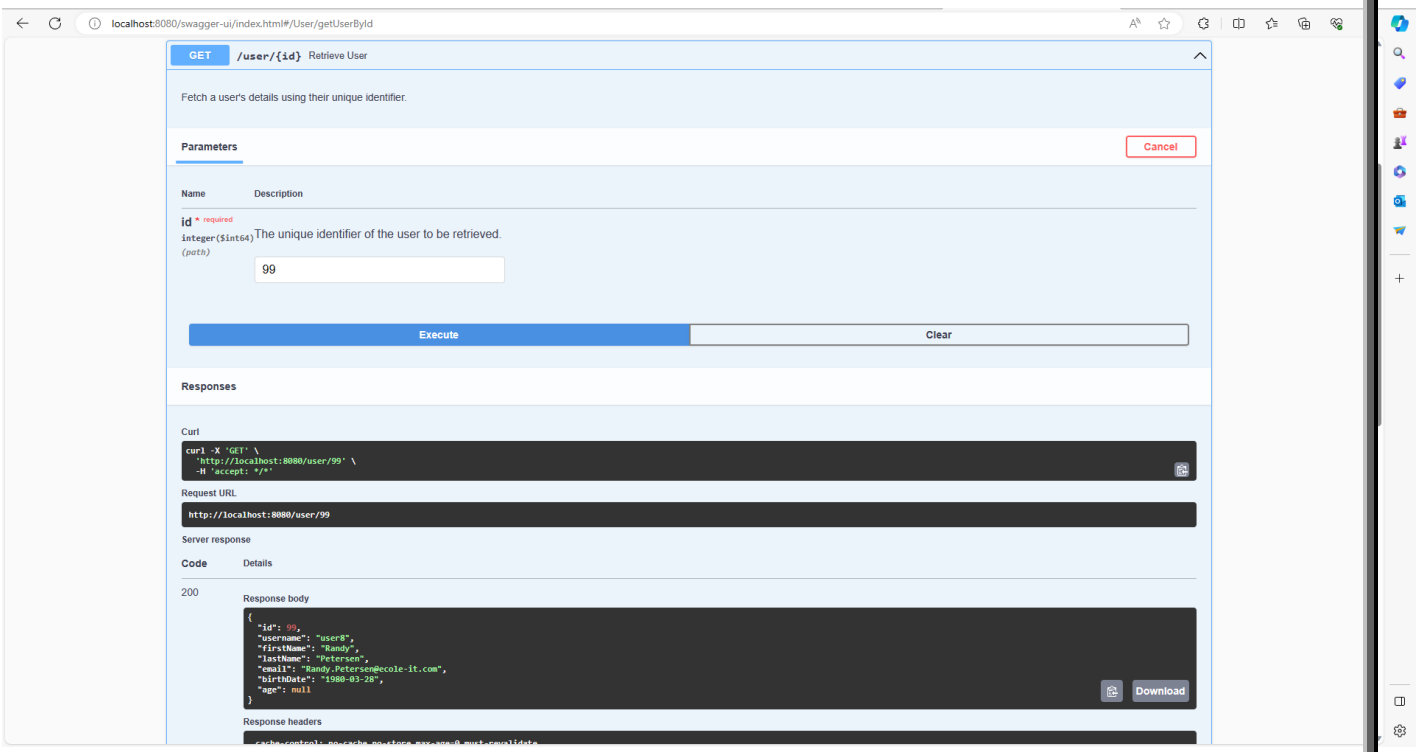
Talking of the Swagger which is being implemented on my project, it's a suite of tools that helps design, build, document, and consume RESTful web services. Swagger generates a detailed interactive documentation of my APIs, allowing for real-time testing and providing clarity on the request and response structure of each endpoint.

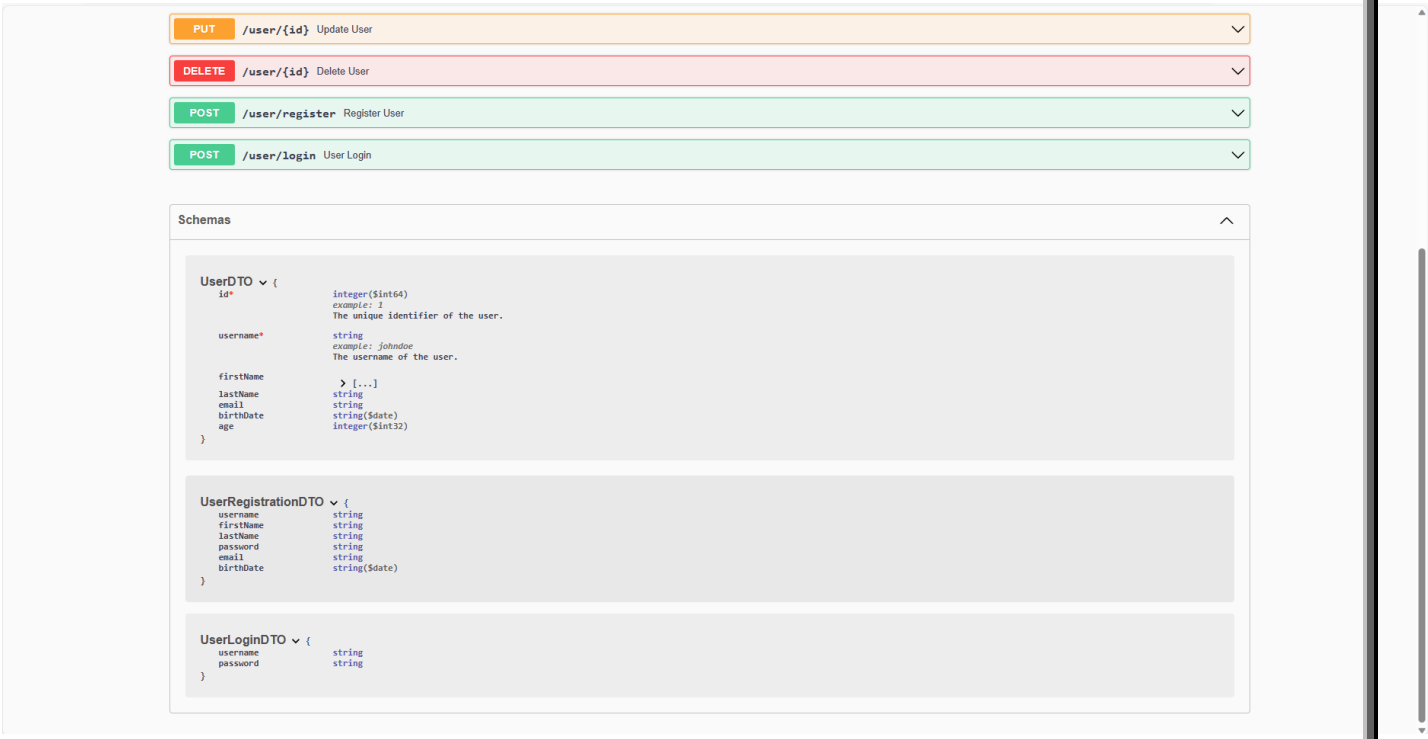
Installing the dependency for swagger is pretty easy to do as you just need to copy paste the code below and reload maven.

```xml
<!-- Swagger dependencies -->
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.0.2</version>
</dependency>
```

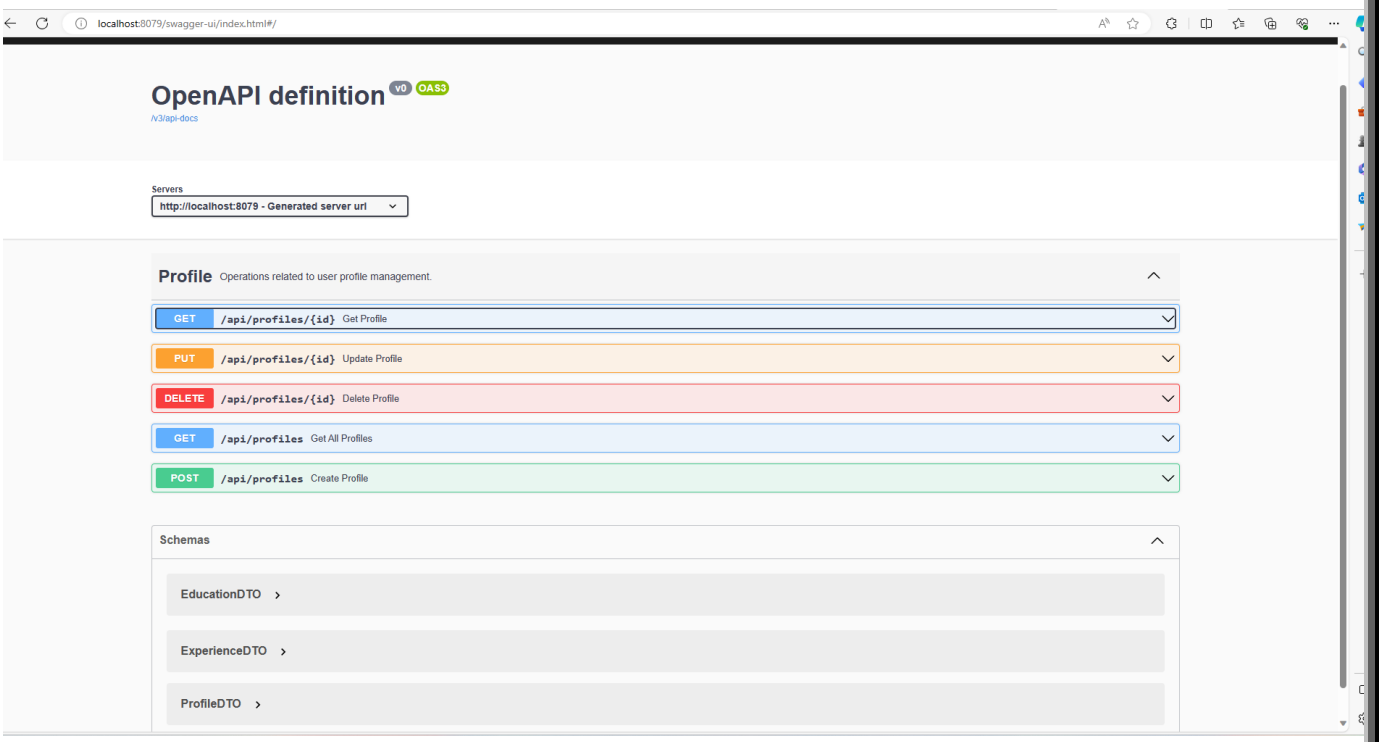
- For the **UserService** , the Swagger UI details endpoints for user registration, login, and profile management, ensuring seamless user experiences.

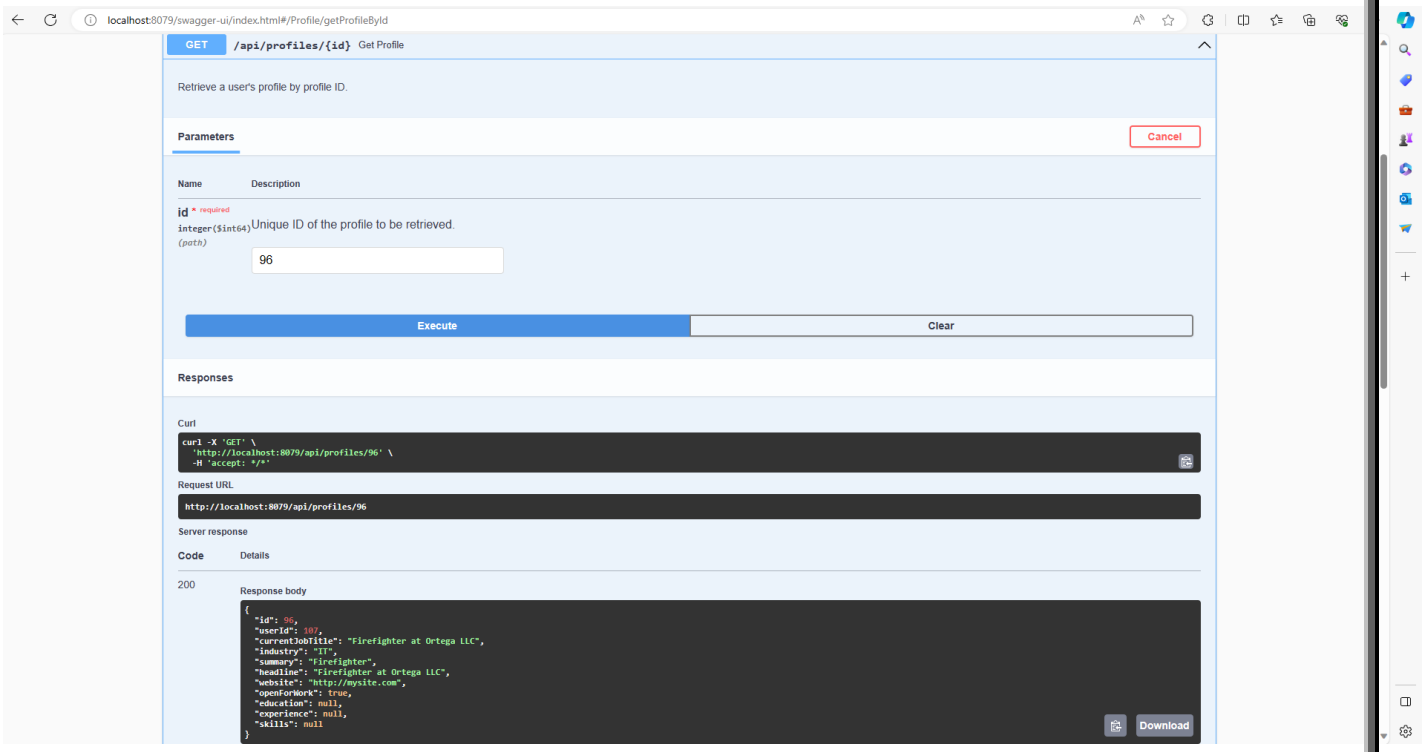


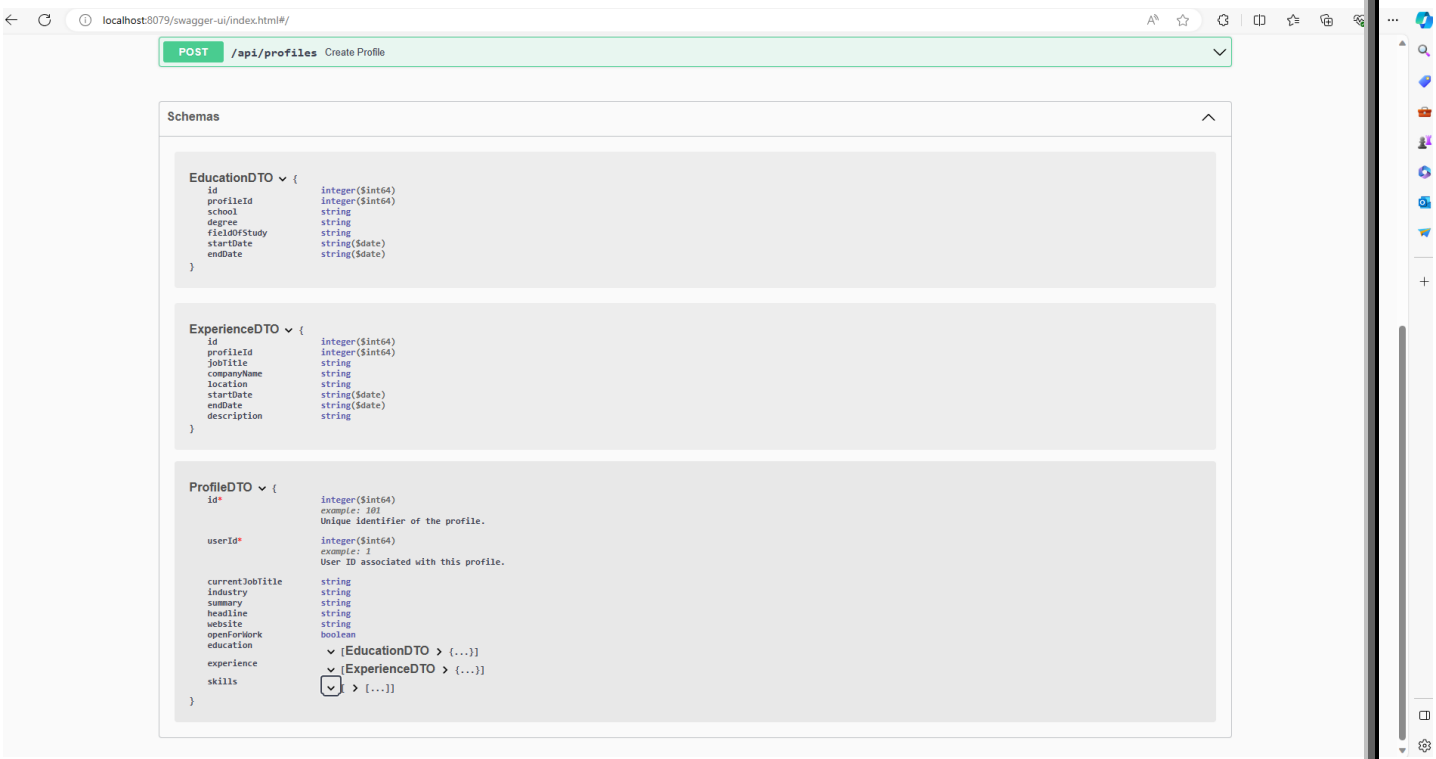

localhost:8080/swagger-ui/index.html#/User/getUserById
GET /user/{id} Retrieve User
Fetch a user's details using their unique identifier.
Parameters
Cancel
Name
Description
id * required
integer($int64)
(path)
The unique identifier of the user to be retrieved.
99
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8080/user/99' \
-H 'accept: */*'
Request URL
http://localhost:8080/user/99
Server response
Code
Details
200
Response body
{
"id": 99,
"username": "user8",
"firstName": "Randy",
"lastName": "Petersen",
"email": "Randy.Petersen@ecole-it.com",
"birthDate": "1980-03-28",
"age": null
}
Download
Response headers


PUT /user/{id} Update User
DELETE /user/{id} Delete User
POST /user/register Register User
POST /user/login User Login
Schemas
UserDTO {
id* integer($int64)
example: 1
The unique identifier of the user.
username* string
example: johndoe
The username of the user.
firstName > [...]
lastName string
email string
birthDate string($date)
age integer($int32)
}
UserRegistrationDTO {
username string
firstName string
lastName string
password string
email string
birthDate string($date)
}
UserLoginDTO {
username string
password string
}

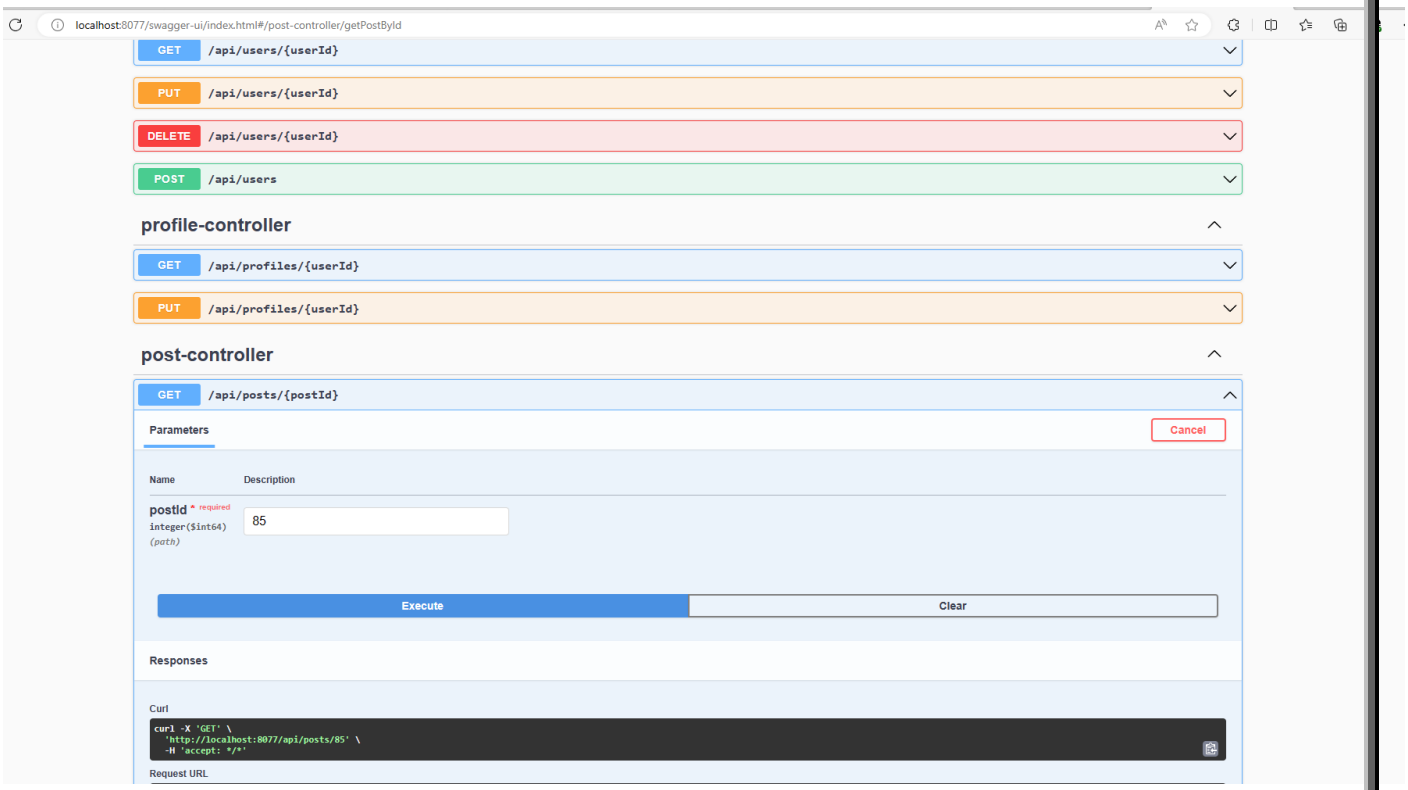
- The **ProfileService** Swagger documentation lists operations to create, update, and retrieve user profiles, reflecting professional details with precision.



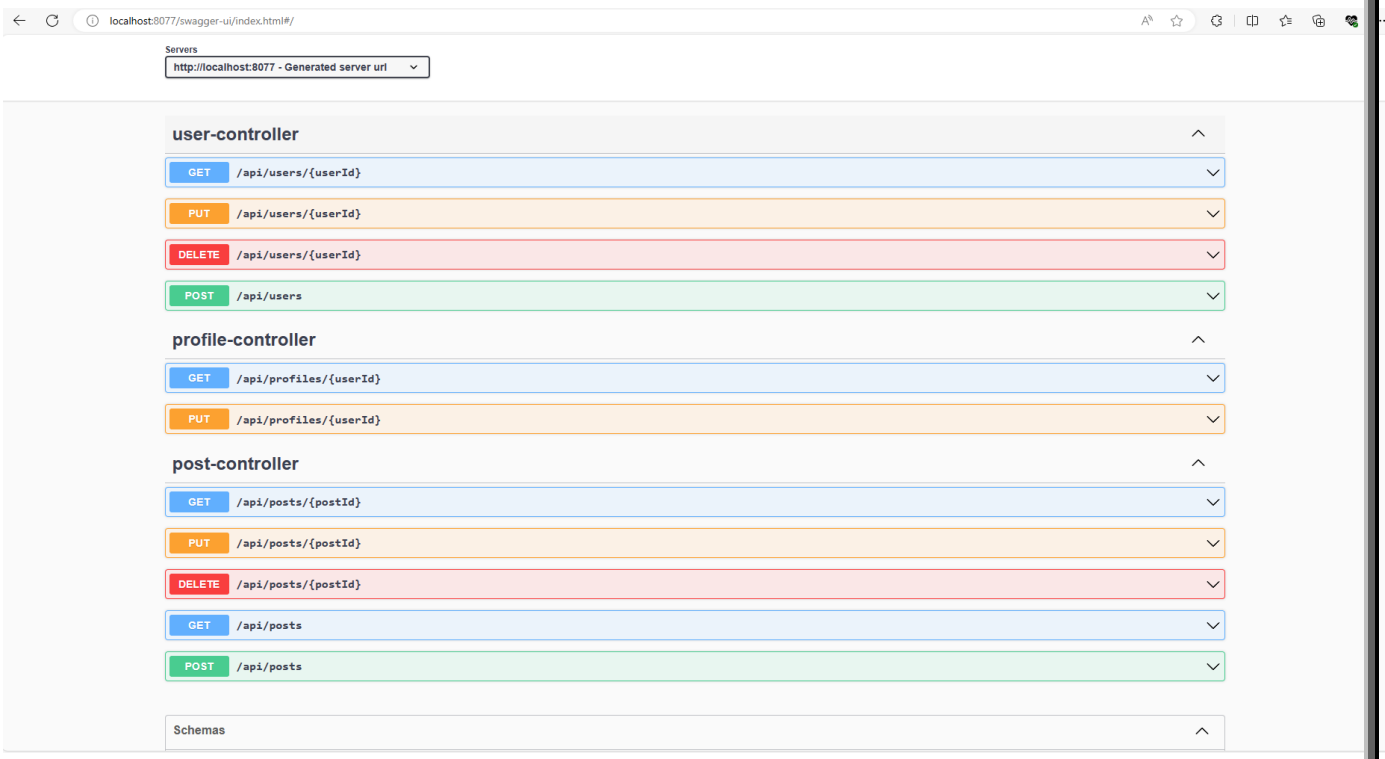

localhost:8079/swagger-ui/index.html#/Profile/getProfileById
GET /api/profiles/{id} Get Profile
Retrieve a user's profile by profile ID.
Parameters
Cancel
Name
Description
id * required
integer($int64)
(path)
Unique ID of the profile to be retrieved.
96
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8079/api/profiles/96' \
-H 'accept: */*'
Request URL
http://localhost:8079/api/profiles/96
Server response
Code
Details
200
Response body
{
"id": 96,
"userId": 107,
"currentJobTitle": "Firefighter at Ortega LLC",
"industry": "IT",
"summary": "Firefighter",
"headline": "Firefighter at Ortega LLC",
"website": "http://mysite.com",
"openForWork": true,
"education": null,
"experience": null,
"skills": null
}
Download

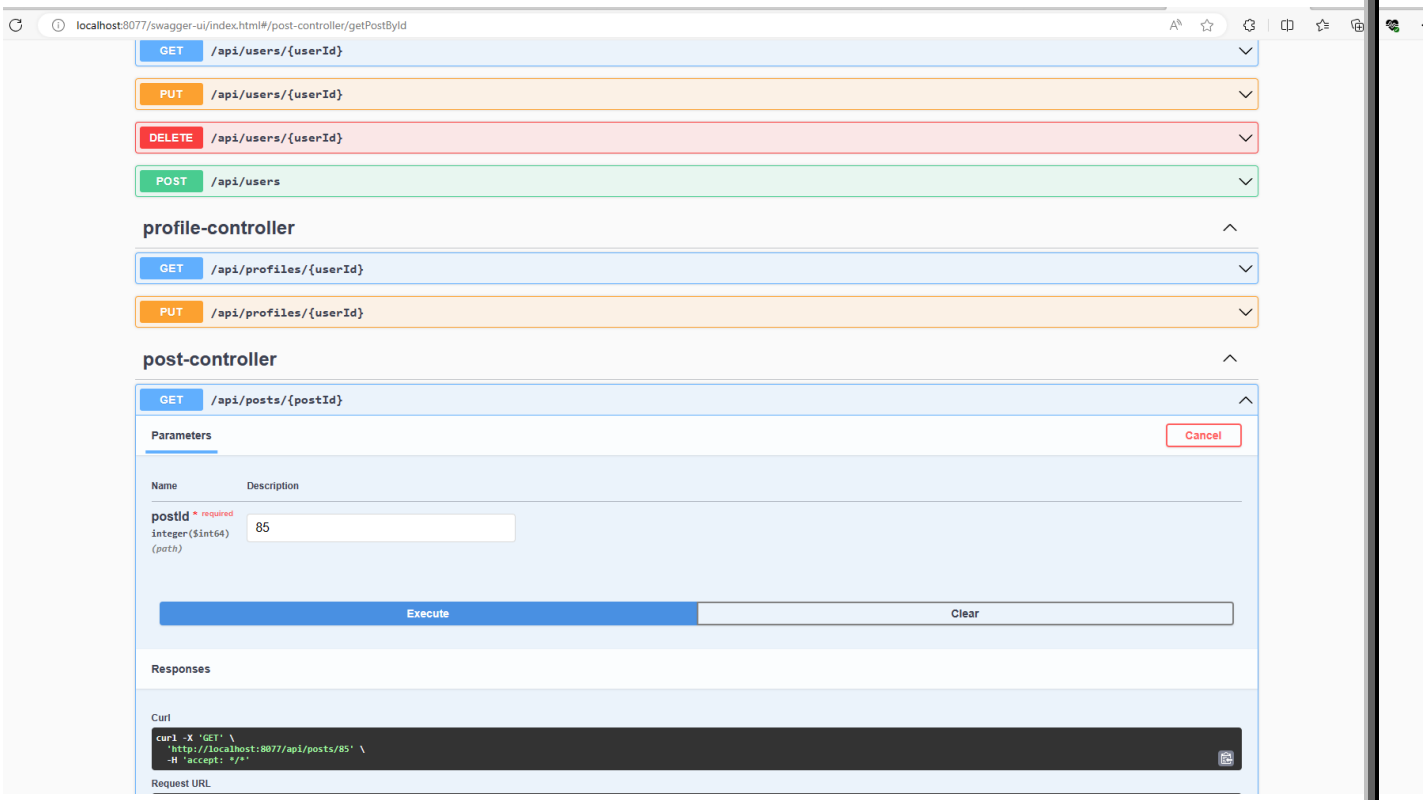

localhost:8079/swagger-ui/index.html#/
POST /api/profiles Create Profile
Schemas
EducationDTO {
id integer($int64)
profileId integer($int64)
school string
degree string
fieldOfStudy string
startDate string($date)
endDate string($date)
}
ExperienceDTO {
id integer($int64)
profileId integer($int64)
jobTitle string
companyName string
location string
startDate string($date)
endDate string($date)
description string
}
ProfileDTO {
id* integer($int64)
example: 101
Unique identifier of the profile.
userId* integer($int64)
example: 1
User ID associated with this profile.
currentJobTitle string
industry string
summary string
headline string
website string
openForWork boolean
education [EducationDTO {...}]
experience [ExperienceDTO {...}]
skills [...]
}

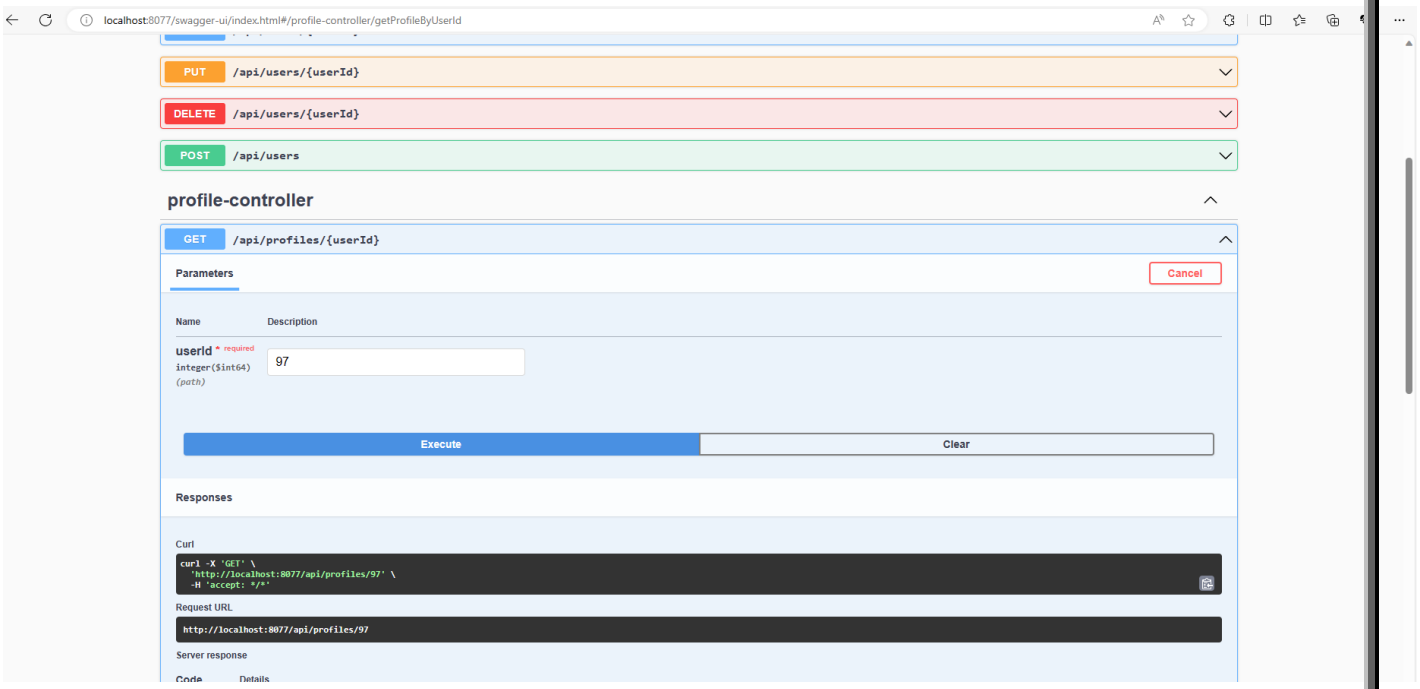
- Swagger for the **PostService** outlines the APIs for posting content, commenting, and liking posts, vital for user engagement on the platform.



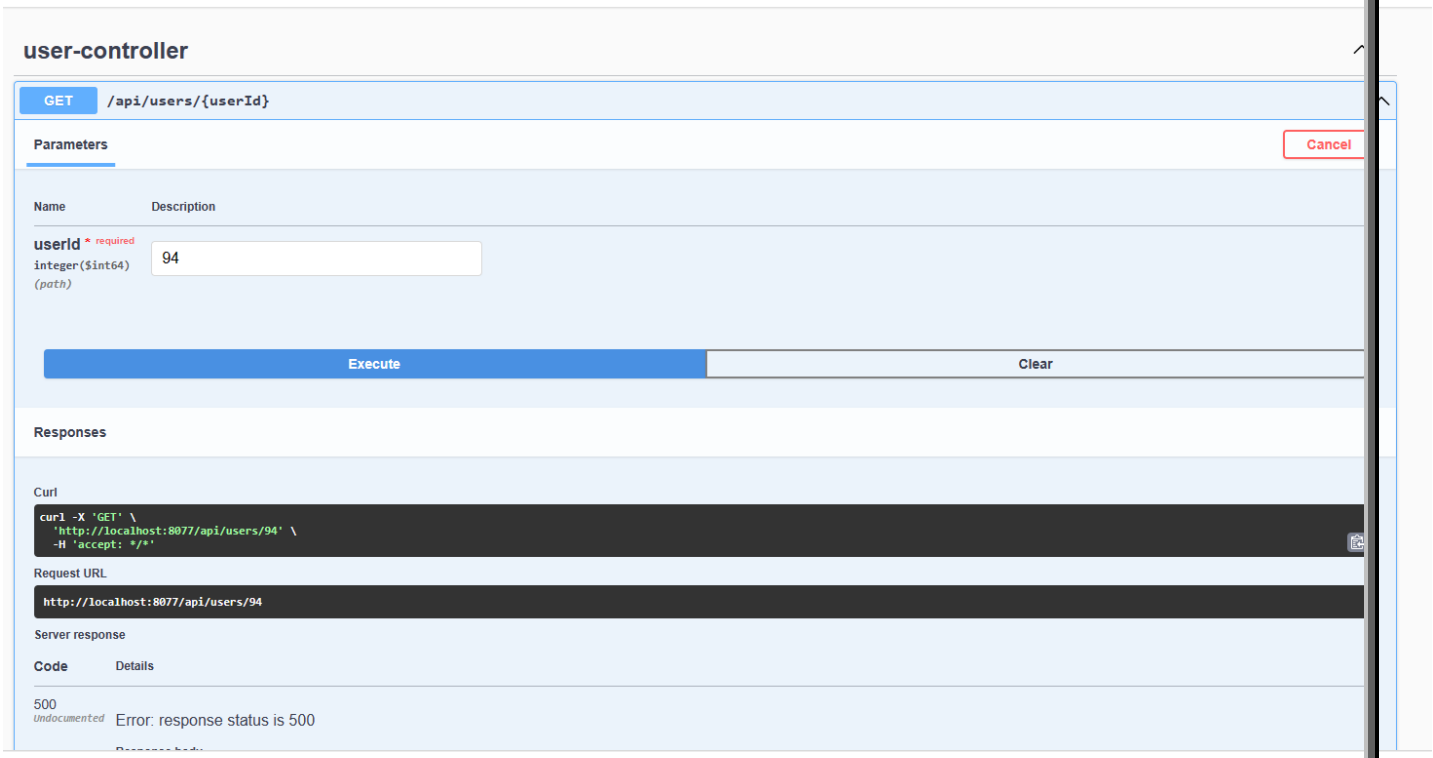
-For the **BFF (Backend for Frontend) Service** in my project, the Swagger UI plays a crucial role in aggregating and documenting the APIs from User, Profile, and Post Services. It provides a unified interface for frontend developers, making it easier to understand and interact with the various backend functionalities. The BFF Swagger UI consolidates the endpoints from all services, offering a comprehensive view of the APIs and streamlining the frontend-backend integration process. This centralized documentation enhances developer experience and expedites the development workflow.


localhost:8077/swagger-ui/index.html#/
Servers
http://localhost:8077 - Generated server url
user-controller
GET /api/users/{userId}
PUT /api/users/{userId}
DELETE /api/users/{userId}
POST /api/users
profile-controller
GET /api/profiles/{userId}
PUT /api/profiles/{userId}
post-controller
GET /api/posts/{postId}
PUT /api/posts/{postId}
DELETE /api/posts/{postId}
GET /api/posts
POST /api/posts
Schemas


localhost:8077/swagger-ui/index.html#/post-controller/getPostById
GET /api/users/{userId}
PUT /api/users/{userId}
DELETE /api/users/{userId}
POST /api/users
profile-controller
GET /api/profiles/{userId}
PUT /api/profiles/{userId}
post-controller
GET /api/posts/{postId}
Parameters
Cancel
Name
Description
postId * required
integer($int64)
(path)
85
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8077/api/posts/85' \
-H 'accept: */*'
Request URL


localhost:8077/swagger-ui/index.html#/profile-controller/getProfileByUserId
PUT /api/users/{userId}
DELETE /api/users/{userId}
POST /api/users
profile-controller
GET /api/profiles/{userId}
Parameters
Cancel
Name
Description
userId * required
integer($int64)
(path)
97
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8077/api/profiles/97' \
-H 'accept: */*'
Request URL
http://localhost:8077/api/profiles/97
Server response
Code
Details


localhost:8077/swagger-ui/index.html#/
PUT /api/posts/{postId}
DELETE /api/posts/{postId}
GET /api/posts
POST /api/posts
Schemas
UserDTO {
id integer($int64)
username string
email string
firstName string
lastName string
password string
}
ProfileDTO {
userId integer($int64)
firstName string
lastName string
email string
phoneNumber string
headline string
summary string
industry string
currentJobTitle string
website string
openForWork boolean
}
PostDTO {
id integer($int64)
userId integer($int64)
title string
content string
creationDate string($date-time)
}


user-controller
GET
/api/users/{userId}
Parameters
Cancel
Name
Description
userId * required
integer($int64)
(path)
94
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:8077/api/users/94' \
-H 'accept: */*'
Request URL
http://localhost:8077/api/users/94
Server response
Code
Details
500
Undocumented
Error: response status is 500

THANKS FOR READING