Authentication Microservice – Onion Architecture & Repository Pattern

Overview

This project is a simple authentication system developed in .NET 8, following the Onion Architecture and Repository Pattern to ensure scalability, maintainability, and separation of concerns.
The service supports user registration, login, and JWT-based authentication for microservice-based applications.

Project Structure

The solution is organized into four main layers, following Onion Architecture principles:

```
1.AuthService
  AuthService.Domain
       Entities/
       Exceptions/
2.AuthService.Application
      Common/
      DTOs/
      Interfaces/
      Settings/
      Services/
      Validators/
3.AuthService.Infrastructure
      Data/
      Repositories/
      Migrations/
      Security/
4.AuthService.API
      Extentions/
      Controllers/
      Filters/
      Logs/
      Middleware/
      Program.co
      appsettings.json
```

Layer Responsibilities

- **Domain** – Defines core business models, entities, and interfaces.
- **Application** – Contains business logic, use cases, and service implementations.
- **Infrastructure** – Handles database connectivity, repository implementations, and migrations.
- **API** – Exposes REST API endpoints for external clients.

Technologies Used

- **.NET 8** – Main framework
- **Entity Framework Core** – ORM for database interactions
- **SQL Server** – Database
- **JWT (JSON Web Token)** – Authentication mechanism
- **FluentValidation** – Request validation
- **Swagger (Swashbuckle)** – API documentation
- **Serilog** – Logging

API Endpoints

Authentication

Method	Endpoint	Description	Auth Required
POST	/api/auth/register	Registers a new user	No
POST	/api/auth/login	Logs in and returns JWT	No

API Testing

You can test the API using:

- **Swagger UI** – Available at https://localhost:7124/swagger
- **Postman Collection** – blob/master/AuthService.postman_collection.json

Running the Project

1. Clone the Repository

```
git clone https://github.com/FFAzeez/AuthService
cd AuthService
```

2. Update Database Connection String

In AuthService.API/appsettings.json, update:

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=AuthService;User
Id=;Password=;TrustServerCertificate=True;"
}
```

3. Apply Migrations

```
cd src/AuthService.Infrastructure
dotnet ef database update --project AuthService.Infrastructure
--startup-project ../AuthService.API
```

5. Then Run Update Database

```
dotnet ef database update --startup-project
../AuthService.API/AuthService.API.csproj
```

4. Run the Service

```
cd ../AuthService.API
dotnet run
```

Logging

This service uses **Serilog** to log:

- API requests & responses
- Authentication attempts
- Database queries
- Errors and exceptions

Logs are written to:

- Console (during development)
- File (in `/logs` directory)

Repository Pattern

- **Interfaces** are defined in `Domain.Interfaces`
- **Implementations** are in `Infrastructure.Repositories`
- All database operations go through repositories, ensuring loose coupling

Security

- Passwords are hashed before saving to the database
- JWT tokens include expiry times and secure claims

- Middleware validates tokens on protected endpoints

Deliverables

- GitHub Repository: https://github.com/FFAzeez/AuthService
- Swagger UI: Available after running the API