

Microservices Architecture for an E-commerce Application

Task :

Building a Microservices Architecture for an E-commerce Application with .NET Core, Ocelot Gateway, PostgreSQL, Kafka, and Clean Architecture.

1. Introduction

This report provides an overview of the microservices architecture developed for the product and order services with auth service for authentication and authorization and API gateway. Also challenges faced on development and how I overcome those challenges.

The following sections detail the architecture, implementation, and key features of this microservices-based e-commerce solution.

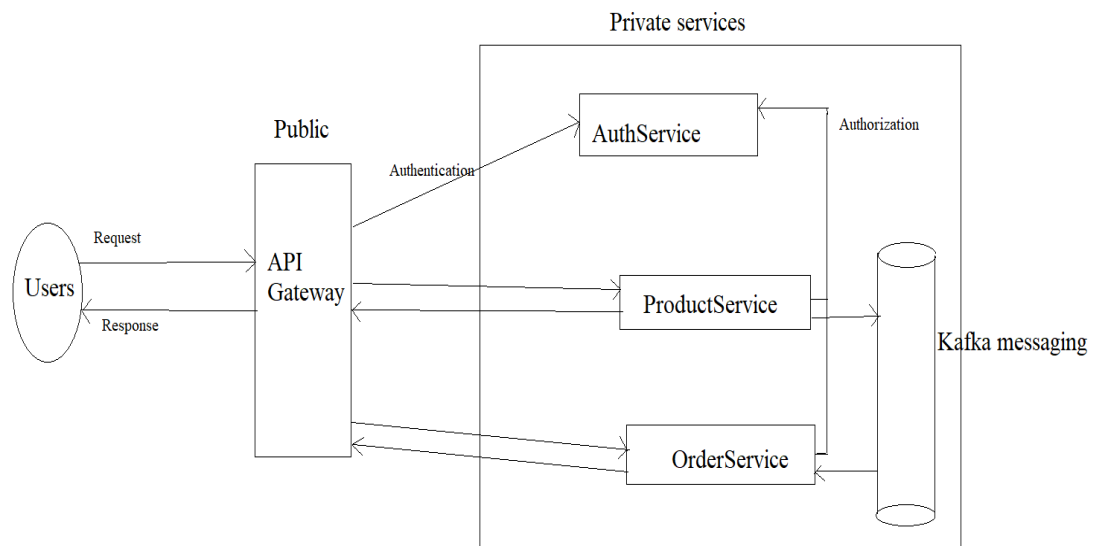


Figure 1: Project Architecture

- The application comprises two microservices: a product microservice and an order microservice. These microservices communicate with client side through an Ocelot gateway, ensuring efficient routing and management of client requests.
- Data persistence is achieved using a PostgreSQL database.
- Moreover, Kafka is employed as the message broker, facilitating seamless inter-service communication.
- Authentication and authorization is done through auth service.

2. Microservices Overview:

The developed microservices architecture consists of four main components:

1. Product Service: Responsible for managing product-related functionalities.
2. Order Service: Handles order management tasks.
3. Authentication Service: Manages user authentication and authorization using JWT token.
4. Gateway: Acts as an entry point for client applications to access the microservices.

2.1 My Preferences:

I used Auth service for both authentication and authorization because of

- Centralized security management
- Single source of trust
- Reduce duplication of code
- Language independent for authentication and authorization,

I used logging in Console, File and Seq using JSON formatter because

- Console log help while development and debugging.
- Separate new log files for each service on daily basis help to store log information for future references.
- Centralized logging using Seq helps to monitor all services from single source.

I added health check for every services because it enables proactive monitoring of service availability and performance. Health checks facilitate automated recovery mechanisms by detecting unhealthy services and triggering automated actions such as service restarts or failover to redundant instances.

I implemented “*ExceptionHandler*” to create global exception handler to handle exception and return structured response which help to me log exception also.

I added a custom authorization action filter for authorization which send request to auth service to verify whether a token is authored or not. This provide the full control over the authorization behavior throughout the system.

I used custom mapping between DTOs and Model class, because I have only one class and it was easy to write extension method rather than configuring AutoMapper for mapping profiles.

3. Service Architecture:

The microservices are built using clean architecture principles, emphasizing separation of concerns and modularity. Each service follows a similar architectural pattern consisting of the following layers:

1. **Domain Layer:** This layer encapsulates the core business logic and domain models. It remains independent of any external dependencies and focuses solely on representing the business concepts.
2. **Interface Layer:** The interface layer defines contracts and interfaces for interacting with external systems or databases. It abstracts away the implementation details, facilitating flexibility and testability.
3. **Application Layer:** This layer houses the application-specific logic, including services and DTOs (Data Transfer Objects). Services orchestrate the interactions between different layers and components, while DTOs facilitate data exchange between layers and services.
4. **API Layer:** Implemented as ASP.NET Core Web APIs, the API layer exposes the microservices' functionalities to client applications. It serves as the entry point for external requests and handles communication with clients.

4. Challenges Faced:

- Difficulty in structuring the project from scratch due to the complexity of microservices architecture and adherence to Clean Architecture principles.
- Implementation challenges encountered while developing the authentication service and defining its flow within the microservices ecosystem.
- Challenges in writing unit tests for controllers, services and repositories, ensuring comprehensive test coverage while maintaining code quality and readability.
- Configuration complexities arose while setting up Kafka for both publishing and consuming messages, requiring careful consideration of topics, partitions, and serialization formats.
- Scheduled dedicated time slots outside of office hours for development and coding tasks, ensuring uninterrupted focus on the project.

5. Overcoming Challenges

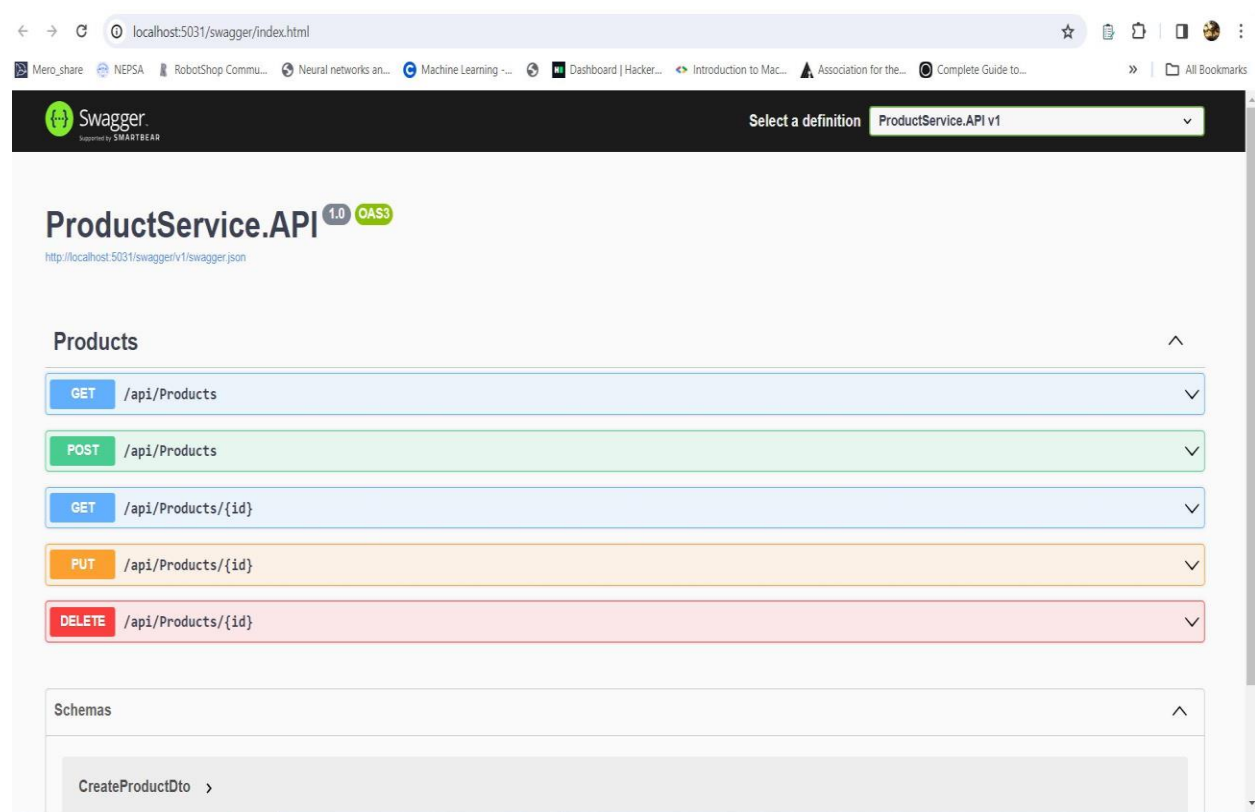
- To overcome difficulties in structuring the project, thorough planning and research were conducted initially. Defining and breaking down the requirement helped in organizing the project effectively.

- Addressing challenges related to the implementation of the authentication service involved searching over internet, going through multiple blog post and took help from Chatgpt.
- For unit testing I took reference from my other task and also read multiple blog post in internet.
- Kafka Configuration is most challenging task for me, I never configured Kafka by myself before this, and I took reference for couple of YouTube videos and Chatgpt.
- Also configuring Publisher and Consumer is challenging but I took help from internet and run docker compose file for Kafka, which I downloaded from internet.

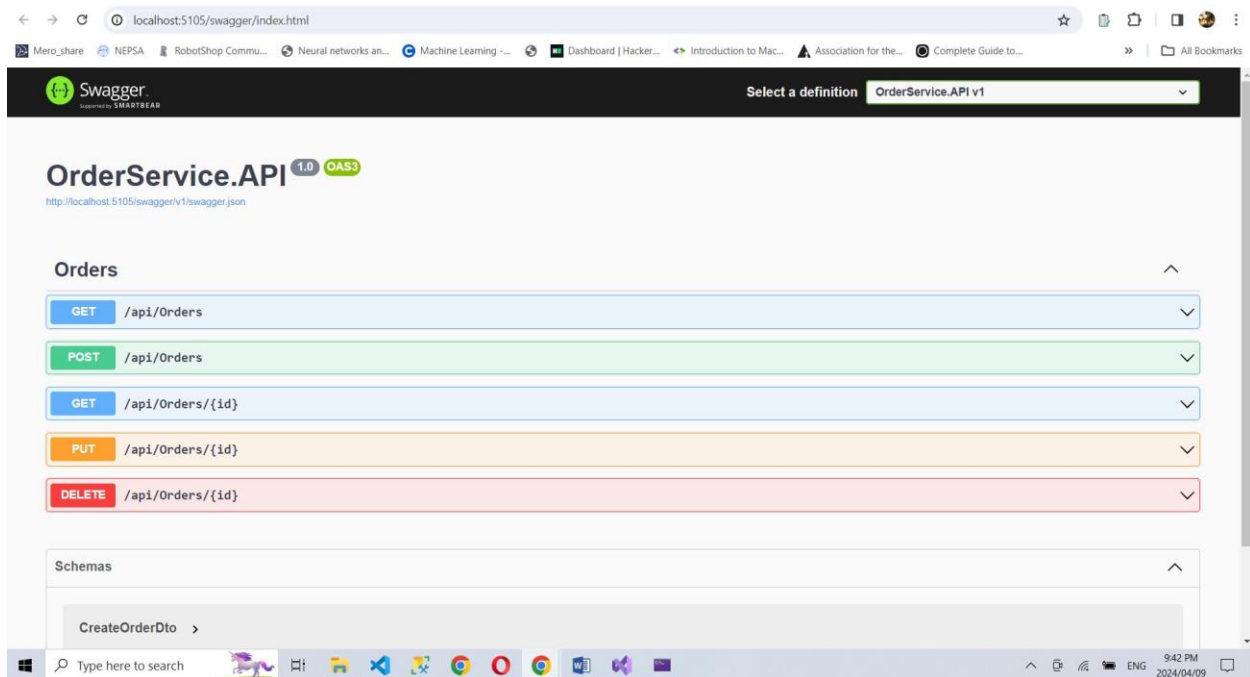
I try my best to setup Kafka as messaging system, I manage to write code through different blog post but not able to run Kafka using docker compose file. I download zookeeper and Kafka locally and tried to install but faced lots of problems specifically some configuration problems.

Screenshots

Product service

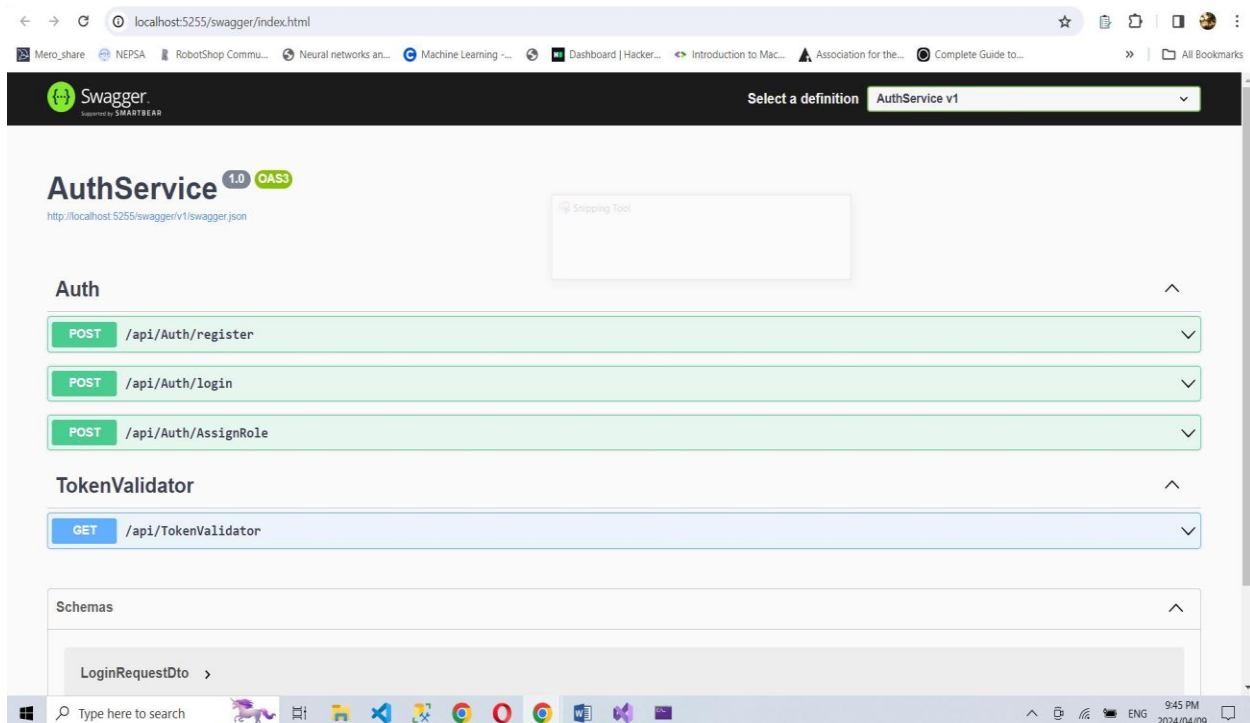


Order service



The image shows the Swagger UI for the OrderService.API v1. The browser address bar displays 'localhost:5105/swagger/index.html'. The Swagger logo is in the top left, and a dropdown menu in the top right shows 'Select a definition' with 'OrderService.API v1' selected. The main heading is 'OrderService.API' with version '1.0' and 'OAS3' tags. Below this, the URL 'http://localhost:5105/swagger/v1/swagger.json' is shown. The 'Orders' section is expanded, showing five endpoints: GET /api/Orders, POST /api/Orders, GET /api/Orders/{id}, PUT /api/Orders/{id}, and DELETE /api/Orders/{id}. The 'Schemas' section is also expanded, showing 'CreateOrderDto'.

Auth Service



The image shows the Swagger UI for the AuthService v1. The browser address bar displays 'localhost:5255/swagger/index.html'. The Swagger logo is in the top left, and a dropdown menu in the top right shows 'Select a definition' with 'AuthService v1' selected. The main heading is 'AuthService' with version '1.0' and 'OAS3' tags. Below this, the URL 'http://localhost:5255/swagger/v1/swagger.json' is shown. The 'Auth' section is expanded, showing three endpoints: POST /api/Auth/register, POST /api/Auth/login, and POST /api/Auth/AssignRole. The 'TokenValidator' section is also expanded, showing one endpoint: GET /api/TokenValidator. The 'Schemas' section is expanded, showing 'LoginRequestDto'.

Seq

The screenshot displays the Seq application interface, which is used for viewing and analyzing log data. The main window shows a list of events with timestamps and descriptions. The sidebar on the right contains filters for signals, queries, variables, and history.

Events Log:

- 08 Apr 2024 21:56:28.497 Request finished HTTP/1.1 GET http://localhost:5105/api/Orders - 200 null application/json; charset=utf-8 5713.3156ms
- 08 Apr 2024 21:56:28.458 Executed endpoint 'OrderService.API.Controllers.OrdersController.Get (OrderService.API)'
- 08 Apr 2024 21:56:28.450 Executed action 'OrderService.API.Controllers.OrdersController.Get (OrderService.API)' in 5510.3161ms
- 08 Apr 2024 21:56:28.388 Executing OkObjectResult, writing value of type 'System.Linq.Enumerable+SelectListIterator`2[OrderService.Domain.Entities.Order, OrderService.Dom...'
- 08 Apr 2024 21:56:28.357 Retrieved all orders successfully.
- 08 Apr 2024 21:56:28.347 Order Service Called successfully for GetAllAsync service method
- 08 Apr 2024 21:56:28.032 Executed DbCommand (81ms) [Parameters=[], CommandType=Text, CommandTimeout=30] SELECT o."Id", o."CreatedBy", o."CreatedDate", o."IsDelet...
- 08 Apr 2024 21:56:25.117 Request Recived and Forwarding to Repository
- 08 Apr 2024 21:56:25.113 GET request received for all orders.
- 08 Apr 2024 21:56:22.924 Route matched with {action = "Get", controller = "Orders"}. Executing controller action with signature System.Threading.Tasks.Task`1[Microsoft.AspNet...
- 08 Apr 2024 21:56:22.849 Executing endpoint 'OrderService.API.Controllers.OrdersController.Get (OrderService.API)'
- 08 Apr 2024 21:56:22.829 Failed to determine the https port for redirect.
- 08 Apr 2024 21:56:22.784 Request starting HTTP/1.1 GET http://localhost:5105/api/Orders - null null
- 08 Apr 2024 21:55:58.347 Request finished HTTP/1.1 GET http://localhost:5105/swagger/v1/swagger.json - 200 null application/json; charset=utf-8 454.7983ms
- 08 Apr 2024 21:55:57.892 Request starting HTTP/1.1 GET http://localhost:5105/swagger/v1/swagger.json - null null
- 08 Apr 2024 21:55:57.641 Request finished HTTP/1.1 GET http://localhost:5105/_vs/browserLink - 200 null text/javascript; charset=UTF-8 640.2707ms
- 08 Apr 2024 21:55:57.409 Request finished HTTP/1.1 GET http://localhost:5105/_framework/aspnetcore-browser-refresh.js - 200 13768 application/javascript; charset=utf-8 3...
- 08 Apr 2024 21:55:57.090 Request starting HTTP/1.1 GET http://localhost:5105/_framework/aspnetcore-browser-refresh.js - null null
- 08 Apr 2024 21:55:57.002 Request starting HTTP/1.1 GET http://localhost:5105/_vs/browserLink - null null
- 08 Apr 2024 21:55:56.934 Request finished HTTP/1.1 GET http://localhost:5105/swagger/index.html - 200 null text/html; charset=utf-8 746.2047ms
- 08 Apr 2024 21:55:56.210 Request starting HTTP/1.1 GET http://localhost:5105/swagger/index.html - null null
- 08 Apr 2024 21:55:54.904 Content root path: E:\Projfilo_Projects\E_Shope\OrderService\OrderService.API
- 08 Apr 2024 21:55:54.901 Hosting environment: Development
- 08 Apr 2024 21:55:54.898 Application started. Press Ctrl+C to shut down.
- 08 Apr 2024 21:55:54.880 Now listening on: http://localhost:5105
- 08 Apr 2024 21:55:49.820 Starting web application
- 08 Apr 2024 21:54:33.944 Request finished HTTP/2 GET https://localhost:7279/ - 404 null application/problem+json 622.6766ms
- 08 Apr 2024 21:54:33.851 requestId: 0HN2NSP293RFV:00000001, previousRequestId: No PreviousRequestId, message: Error Code: UnableToFindDownstreamRouteError Mess...
- 08 Apr 2024 21:54:33.845 requestId: 0HN2NSP293RFV:00000001, previousRequestId: No PreviousRequestId, message: DownstreamRouteFinderMiddleware setting pipeline ar...

Sidebar:

- SIGNALS:** None, @Level1, Errors, Warnings, Exceptions, + 2 more
- QUERIES:**
- VARIABLES:**
- HISTORY:**

Footer: 2024.2.11240 Individual License, 9:57 PM, 2024/04/08