# E-commerce Microservices Project

## Project Overview

This project is a Microservices-based E-commerce Application built using Java Spring Boot. Each functionality (Authentication, Product Management, Order Management, etc.) is separated into individual microservices to ensure scalability, maintainability, and flexibility.

The system uses an API Gateway to handle all client requests, route them to appropriate services, and manage cross-cutting concerns like authentication and CORS.

## Architecture

- Auth Service: User Registration, Login, JWT Authentication

- Product Service: Product Catalog Management

- Order Service: Order Creation, Order Management

- API Gateway: Centralized Entry Point for All Requests

Each service has its own database (Database per Service Pattern). Services communicate through REST APIs (synchronous communication).

## Tech Stack

|  |  |
| --- | --- |
| Layer | Technology |
| Language | Java 17 |
| Framework | Spring Boot 3.x |
| Authentication | JWT (JSON Web Tokens) |
| API Gateway | Spring Cloud Gateway |
| Database | PostgreSQL / MySQL (configurable) |
| Build Tool | Maven |
| Service Communication | REST APIs |
| Security | Spring Security |
| Development Tools | IntelliJ IDEA, Postman |

## Microservices Breakdown

### 1. Auth Service

Endpoints:

- POST /api/auth/register — Register new users

- POST /api/auth/login — Authenticate users and issue JWT tokens

Features:

- JWT Token generation and validation

- Password encryption using BCrypt

- Role-based Authorization

### 2. Product Service

Endpoints:

- POST /api/products — Create a new product

- GET /api/products — List all products

- GET /api/products/{id} — Get product by ID

Features:

- Add, Read, Update, Delete (CRUD) operations

- Validations for input data

- Authentication required for write operations

### 3. Cart Service

Endpoints:

POST /api/cart/add — Add a product to the user's cart

DELETE /api/cart/remove — Remove a product from the cart

PUT /api/cart/update — Update quantity of an item in the cart

GET /api/cart — Retrieve the current user's cart

Features:

Associates cart with the authenticated user

Allows product addition, removal, and quantity modification

Validates product existence using Product Service

Ensures user authentication via JWT

Persists cart data in a dedicated cart and cart\_items schema

### 4. Order Service

Endpoints:

- POST /api/orders — Place a new order

- GET /api/orders — Get user orders

Features:

- Order creation linked with products

- Fetching user-specific orders

- Authentication and Authorization for order placement

### 5. API Gateway

Responsibilities:

- Routes requests to respective microservices

- JWT Token forwarding and validation

- Centralized CORS policy configuration

- Load balancing (future scope)

Gateway Routes Examples:  
spring:  
 cloud:  
 gateway:  
 routes:  
 - id: auth-service  
 uri: lb://AUTH-SERVICE  
 predicates:  
 - Path=/api/auth/\*\*  
   
 - id: product-service  
 uri: lb://PRODUCT-SERVICE  
 predicates:  
 - Path=/api/products/\*\*  
   
 - id: order-service  
 uri: lb://ORDER-SERVICE  
 predicates:  
 - Path=/api/orders/\*\*

**5. Eureka Server (Service Discovery)**

Responsibilities:

* Registers all microservices (Auth, Product, Cart, Order)
* Enables service-to-service communication without hardcoded URLs
* Provides health checks and heartbeat monitoring
* Centralizes service location data for load-balanced routing

Configuration Snippet (example):

spring:

application:

name: eureka-server

server:

port: 8761

eureka:

client:

register-with-eureka: false

fetch-registry: false

Service Registration Example (in a microservice):

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

## Setup Instructions

1. Clone the Repositories  
 git clone https://github.com/your-username/ecommerce-microservices.git  
  
2. Database Setup  
 - Create separate databases for:  
 - Auth Service (e.g., ecommerce\_auth\_db)  
 - Product Service (e.g., ecommerce\_product\_db)  
 - Order Service (e.g., ecommerce\_order\_db)

- Cart Service (e.g., ecommerce\_cart\_db)  
 - Update each service's application.properties or application.yml with database credentials.  
  
3. Run Each Service  
 - Start the Auth Service:  
 cd auth-service  
 mvn spring-boot:run  
 - Start the Product Service:  
 cd product-service  
 mvn spring-boot:run  
- Start the Cart Service:  
 cd cart-service  
 mvn spring-boot:run   
- Start the Order Service:  
 cd order-service  
 mvn spring-boot:run  
- Start the API Gateway:  
 cd api-gateway  
 mvn spring-boot:run  
- Start the Eureka Servcer:  
 cd eureka-server  
 mvn spring-boot:run  
  
4. Test APIs Using Postman  
 - First, register/login using the Auth Service.  
 - Use the JWT token in Authorization headers (Bearer token) when accessing Product or Order services.

## Security Implementation

- Passwords are stored in hashed format (using BCrypt).

- JWT is validated at the API Gateway and passed downstream.

- Role-based Access can be expanded (Admin, User, Seller roles).

## Future Enhancements

- Add Payment Service.

- Implement Rate Limiting and Circuit Breaker using Resilience4J.

- Deploy services using Docker and manage using Kubernetes.

- Integrate Distributed Tracing using Zipkin or Sleuth.

## Contributing

Pull requests are welcome. For major changes, please open an issue first to discuss what you would like to change.