# Microservice Architecture and Documentation

**Tech stack used**

Backend: Java Spring Boot (User service and Order service)

API Gateway: Spring Cloud Gateway

Service Registry: Eureka

Configuration Management: Spring Cloud Config

Database: PostgreSQL for User & Order services

Security: OAuth 2.0 (Keycloak/Auth0)

Understanding each Microservices Individually.

**1. API Gateway**

🔹 **Purpose:**

* Acts as the **entry point** for all external requests.
* Handles **authentication, authorization, rate limiting, and request routing**.
* Ensures that clients don’t directly communicate with internal microservices.

🔹 **Tech Stack:**

* **Spring Cloud Gateway**
* Supports **JWT-based authentication** (OAuth 2.0).

🔹 **Example Functionality:**

* /api/users/\*\* → Routes requests to the **User Service**.

**2. User Service**

🔹 **Purpose:**

* Manages **user registration, login, and profile management**.
* Handles **authentication** (if no external auth provider like Keycloak is used).

🔹 **Tech Stack:**

* **Spring Boot (REST API)**
* **PostgreSQL** (for user data storage)
* **Spring Security + JWT** (for authentication).

🔹 **Example API Endpoints:**

* POST /users/register → Registers a new user.
* POST /users/login → Authenticates a user and returns a JWT token.
* GET /users/{id} → Fetches user details.

**3. Order Service**

🔹 **Purpose:**

* Manages **orders, payments, and order history**.
* Ensures that only authenticated users can place/view orders.

🔹 **Tech Stack:**

* **Spring Boot (REST API)**
* **PostgreSQL** (for order data storage).

🔹 **Example API Endpoints:**

* POST /orders/create → Creates a new order.
* GET /orders/user/{userId} → Fetches order history of a user.

**4. Service Registry**

🔹 **Purpose:**

* Allows microservices to **register themselves dynamically**.
* Enables **service discovery** so services can find and communicate with each other.

🔹 **Tech Stack:**

* **Eureka Server (Spring Cloud Netflix Eureka)**.

🔹 **How it Works:**

* **User Service & Order Service** register themselves in Eureka.
* The API Gateway uses Eureka to route requests dynamically.

**5. Configuration Service**

🔹 **Purpose:**

* Manages **centralized configuration** for all services.
* Ensures that configurations (like database URLs, security keys) are **managed in one place**.

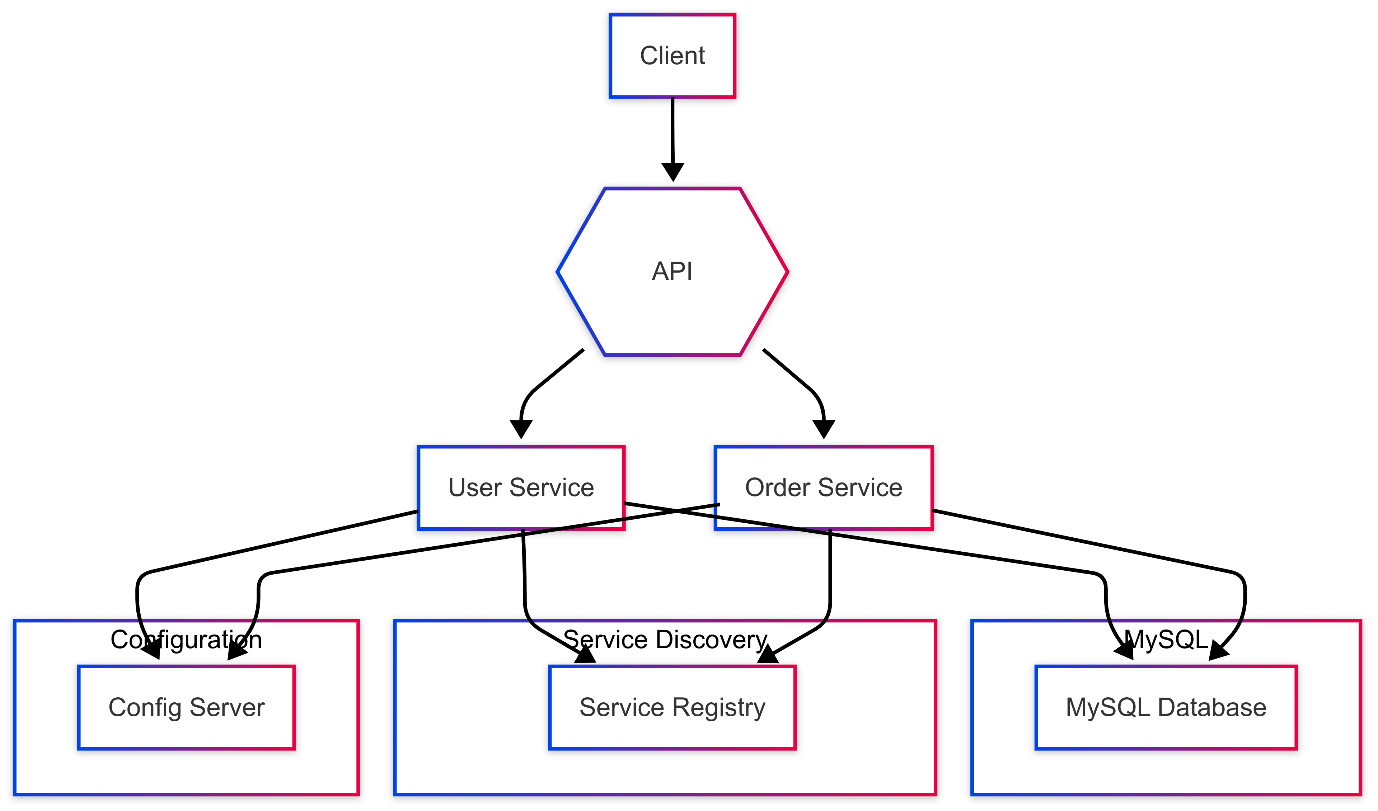
🔹 **Tech Stack:**

* **Spring Cloud Config Server**.

🔹 **How it Works:**

* Stores **configuration properties** in Git or a central database.
* Microservices fetch configuration from this service during startup.

High level design for the project



**Client:** Interacts with the system through the API Gateway.

**API Gateway:** Acts as the entry point, routing requests to the appropriate service.

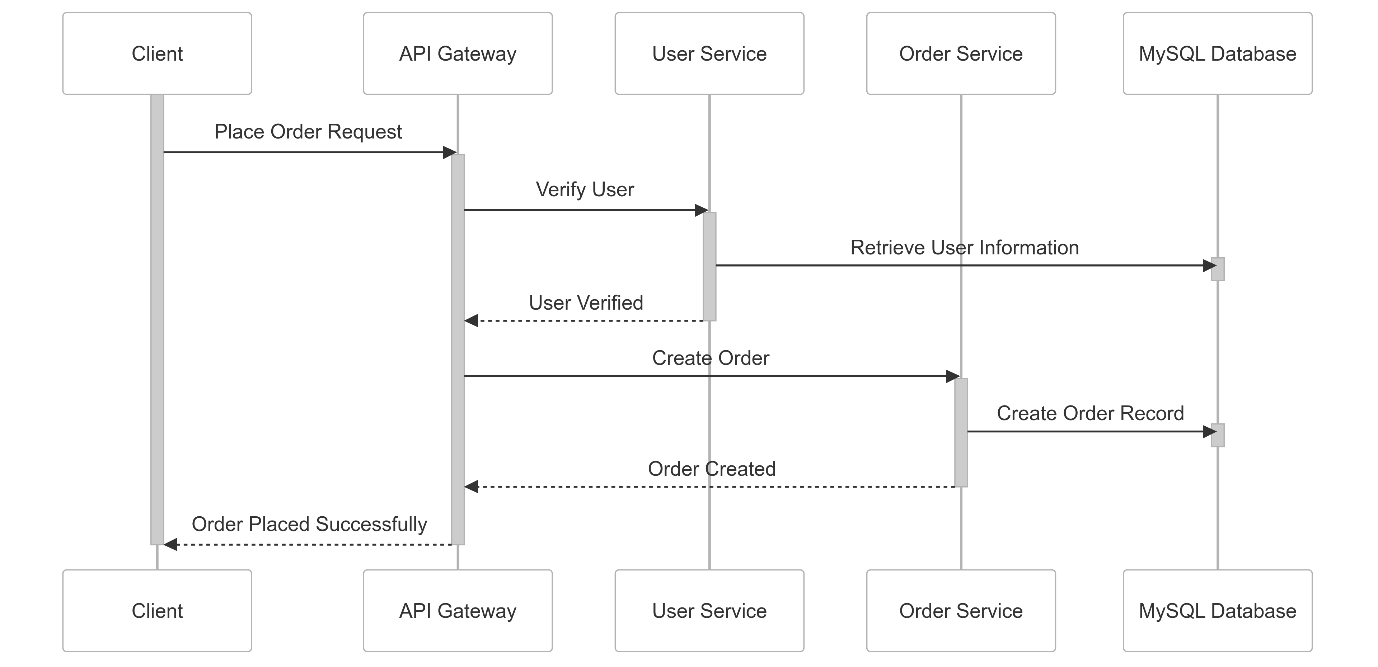
**User Service:** Handles user-related operations, interacting with the database and service registry.

**Order Service:** Manages orders, interacting with the database, user service, and service registry.

**MySQL Database:** Stores data for both User and Order services.

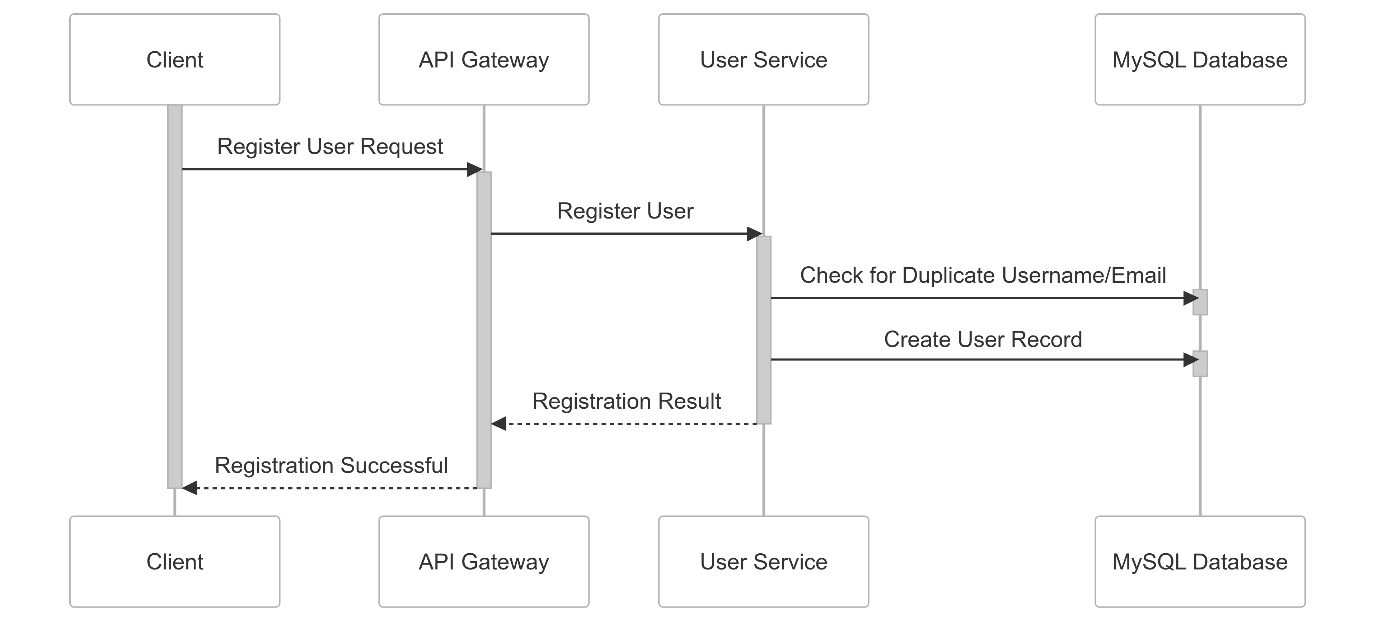
**Service Registry:** Enables dynamic service discovery.

**Config Server:** Centralizes configuration management for all services.

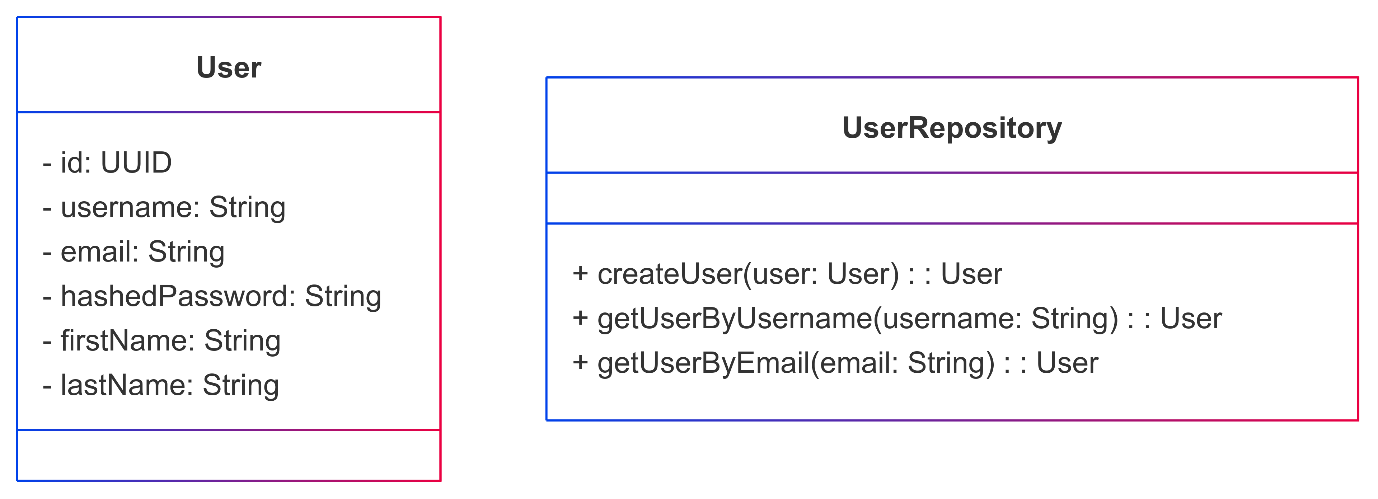
Sequential diagram:

User service

Sequential diagram.

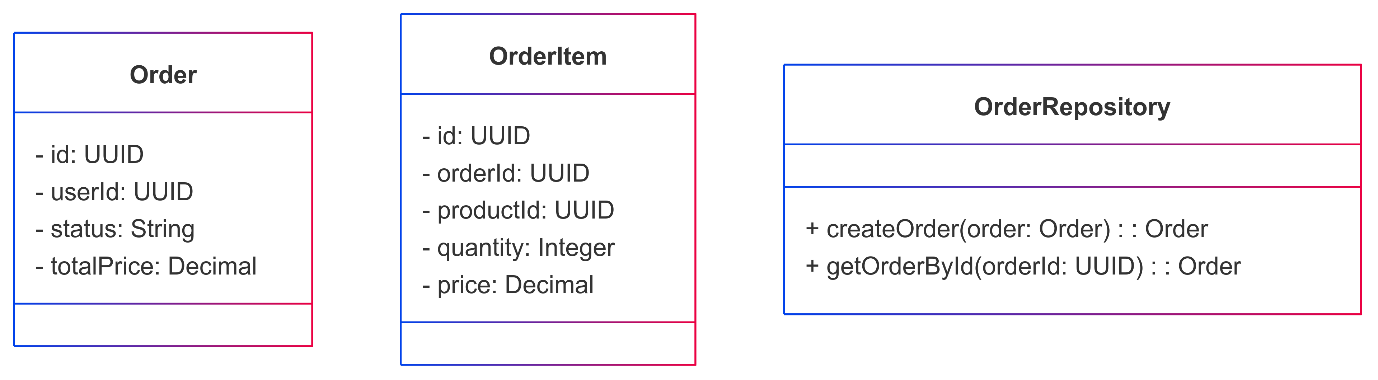


Class diagram

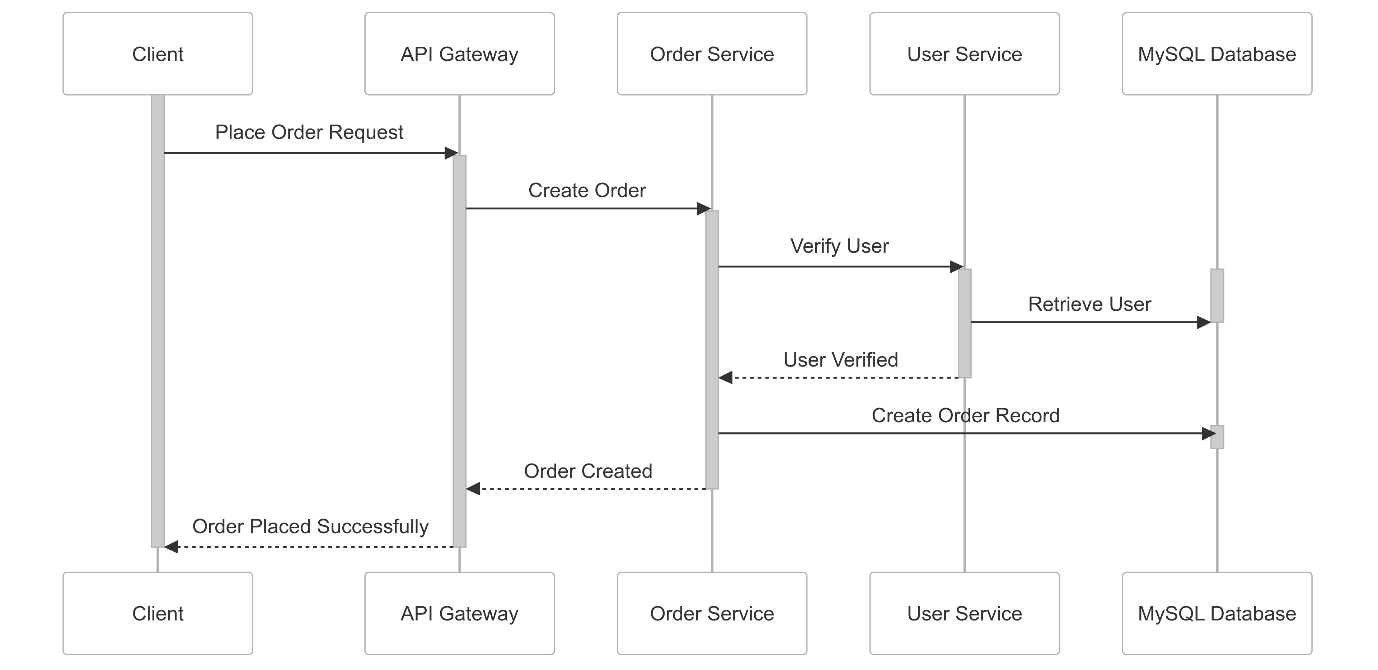


Order service

Class diagram:

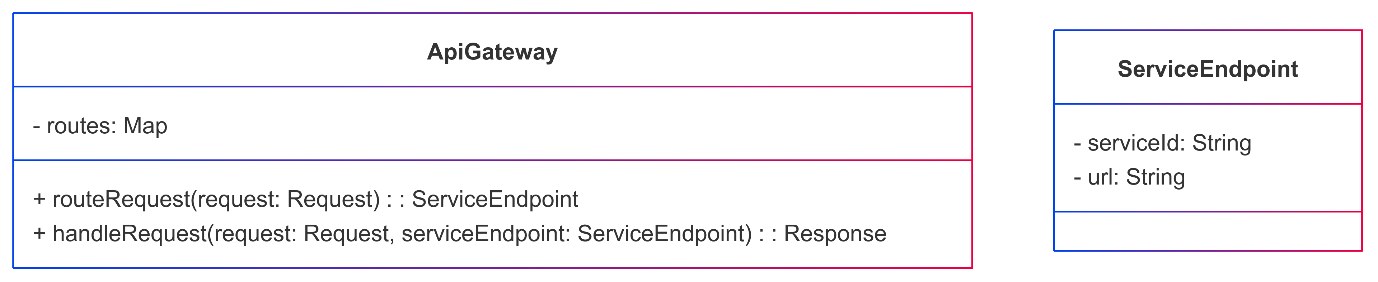


Sequential diagram:

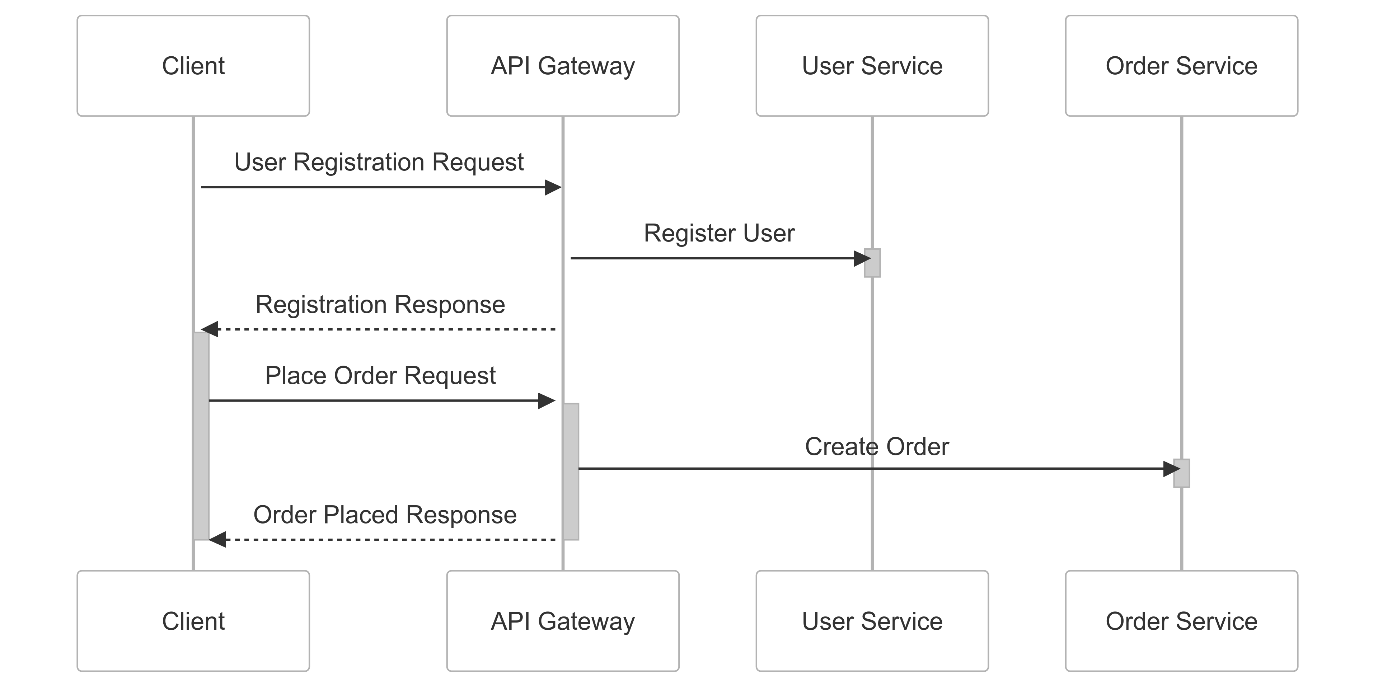


API gateway

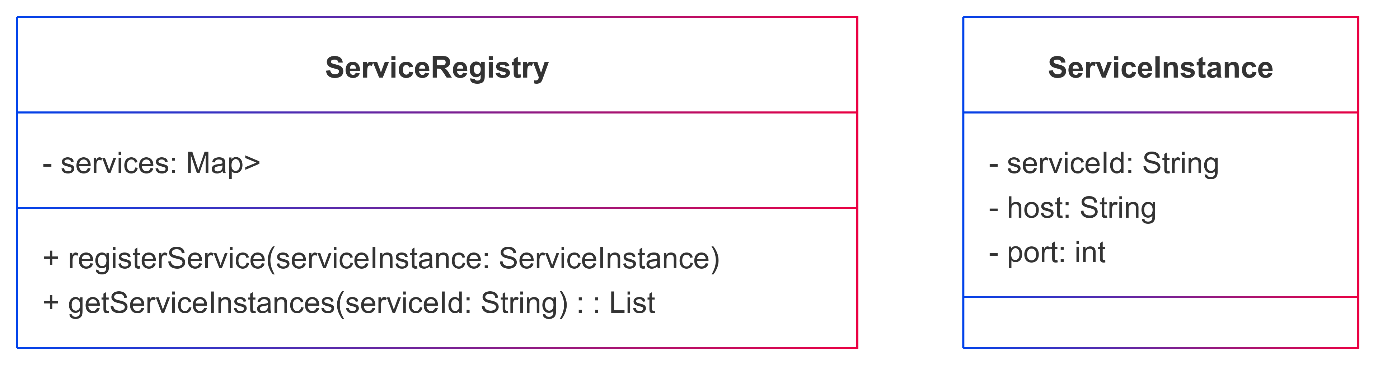
Class diagram



Sequential diagram



Service Registry:



Config.  
