

**Scalable Services Assignment**

Karuna Sharma – 2023sl93004

Pradeep Kumar A – 2023sl93017

Harshita Bijlani – 2023sl93030

Shivangi Sai – 2023sl93090

Krishnanand Bhat - 2023sl93001

**OrgSphere**

**Introduction**

The **OrgSphere Project** is a practical demonstration of building a microservices-based architecture using **Spring Boot** and **Spring Cloud**. It consists of services such as the **Organization Service**, **Employee Service**, and **Department Service**, each managing specific business functionalities while communicating seamlessly with one another. These services are deployed in a **Kubernetes environment**, leveraging features like service discovery and centralized configuration for resilient inter-service communication. The **Gateway Service** acts as a unified entry point, routing requests to the appropriate microservices, while the **Admin Service** monitors the health and performance of the system in real-time. By utilizing Kubernetes, the project ensures high availability, dynamic scaling, and efficient orchestration of all services. This implementation showcases best practices for creating scalable, maintainable, and observable cloud-native applications.

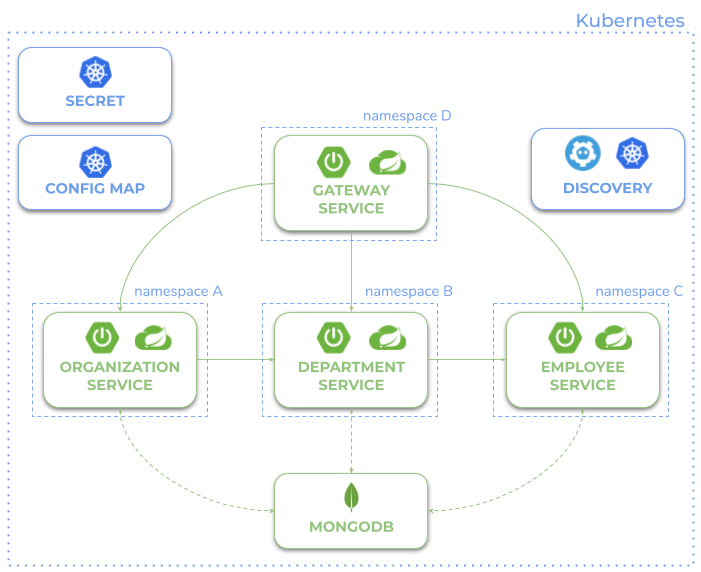
**Key Features**

* **Microservices Architecture**: Implements modular microservices, each responsible for specific business functionalities.
* **Spring Cloud Integration**: Utilizes Spring Cloud features like service discovery, centralized configuration, and load balancing to enable robust inter-service communication.
* **Kubernetes Deployment**: Demonstrates deployment of microservices on Kubernetes, leveraging its scalability and orchestration capabilities.
* **Monitoring and Administration**: Includes Spring Boot Admin to monitor the health and metrics of microservices in real-time.
* **Database Interaction**: Features CRUD operations using MongoDB for storing organizational, departmental, and employee data.

**Architecture Overview**

The architecture is composed of the following modules:

1. **Gateway Service**: A Spring Cloud Netflix Zuul-based service acting as a gateway for routing requests to appropriate microservices.
2. **Employee Service**: Handles CRUD operations for employee data stored in a MongoDB repository.
3. **Department Service**: Manages CRUD operations for departmental data and communicates with the employee service.
4. **Organization Service**: Provides CRUD operations for organizational data, interacting with both employee and department services.
5. **Admin Service**: Implements Spring Boot Admin for monitoring microservices deployed on Kubernetes.



**Implementation**

**1. Prerequisites**

Before you begin, ensure you have the following tools installed:

* **Java JDK 8/11**: To run the Spring Boot applications.
* **Maven**: For building the project.
* **Docker**: To create and manage Docker images.
* **Minikube** (or any Kubernetes cluster): To deploy and test the services.
* **kubectl**: For managing Kubernetes resources.
* React: for frontend.

**2. Build and Run the microservice locally**

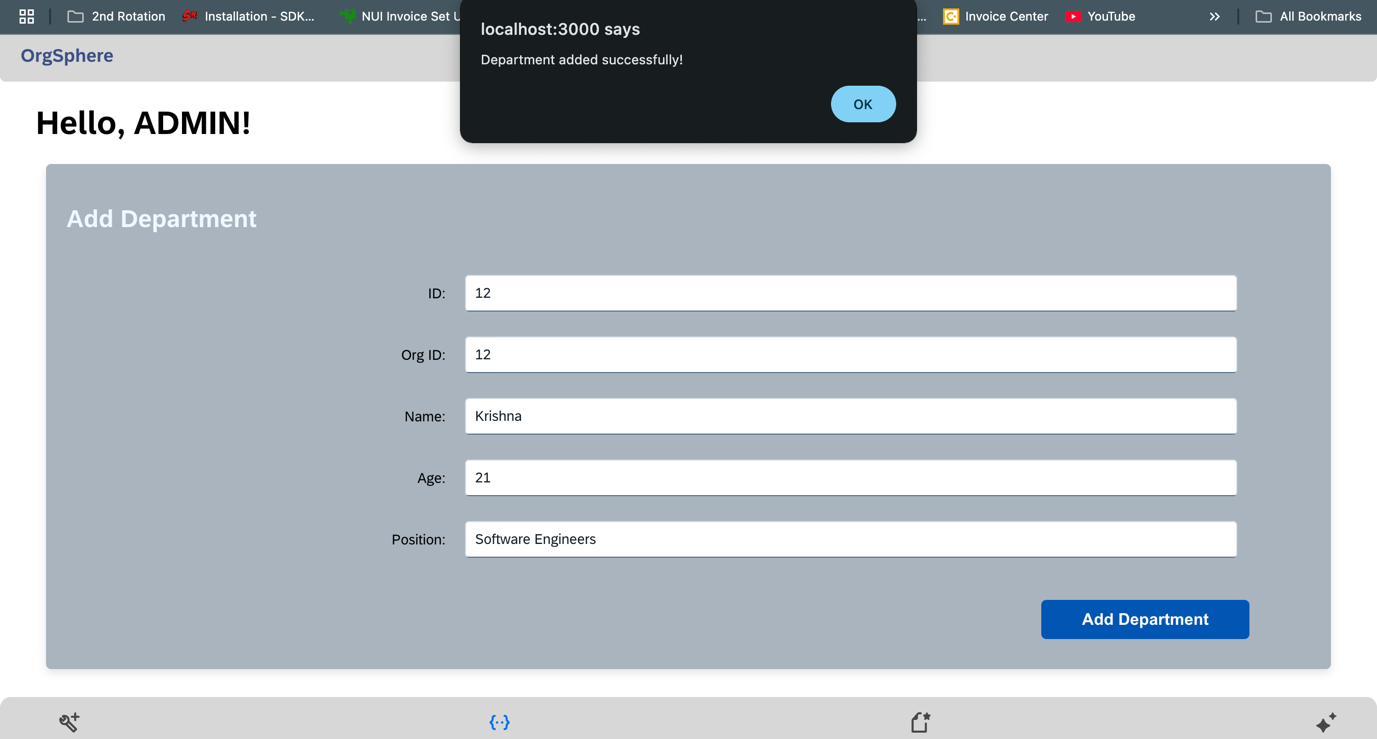
Navigate to corresponding microservice and use below mentioned code to run the microservice

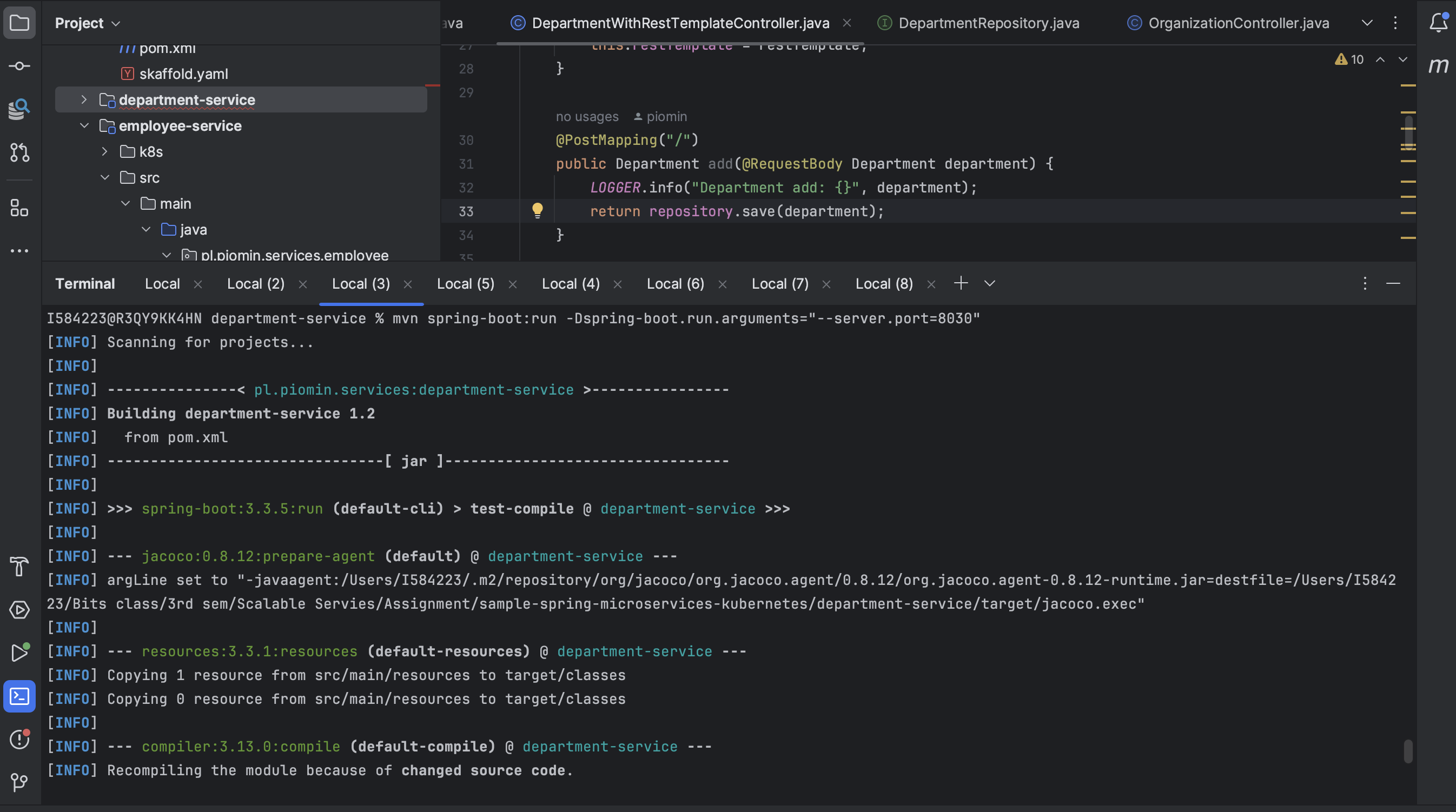
2.1 Department Service

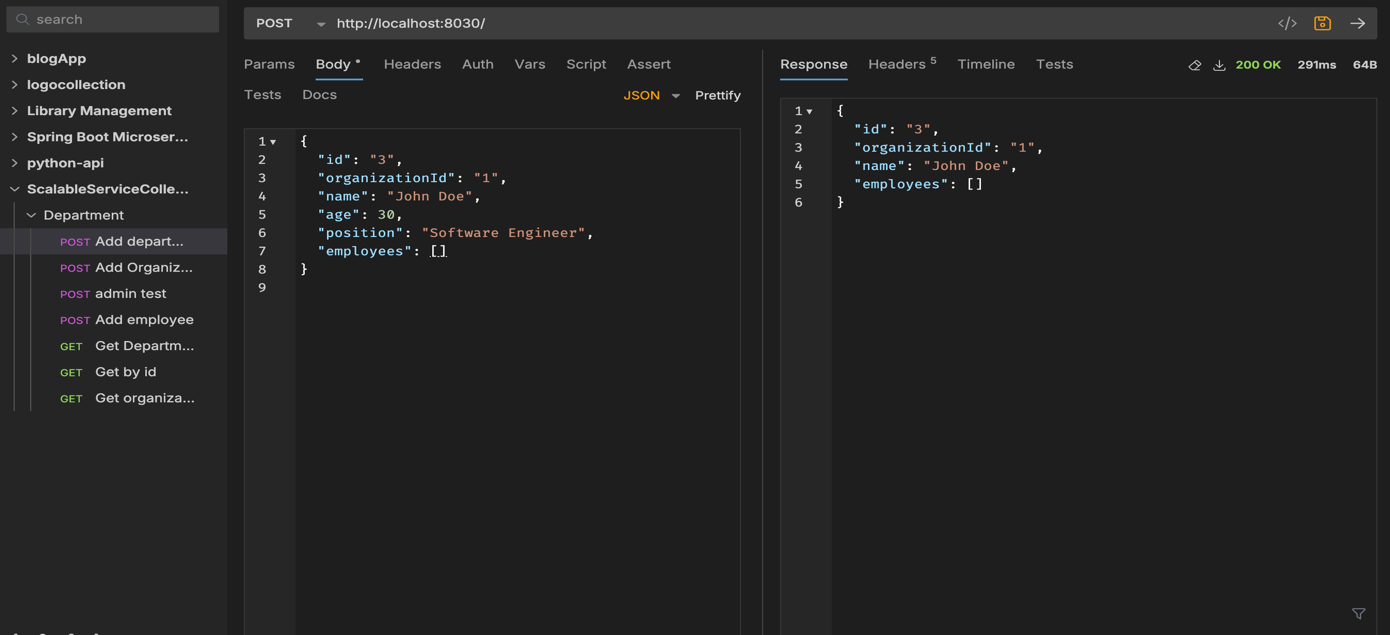
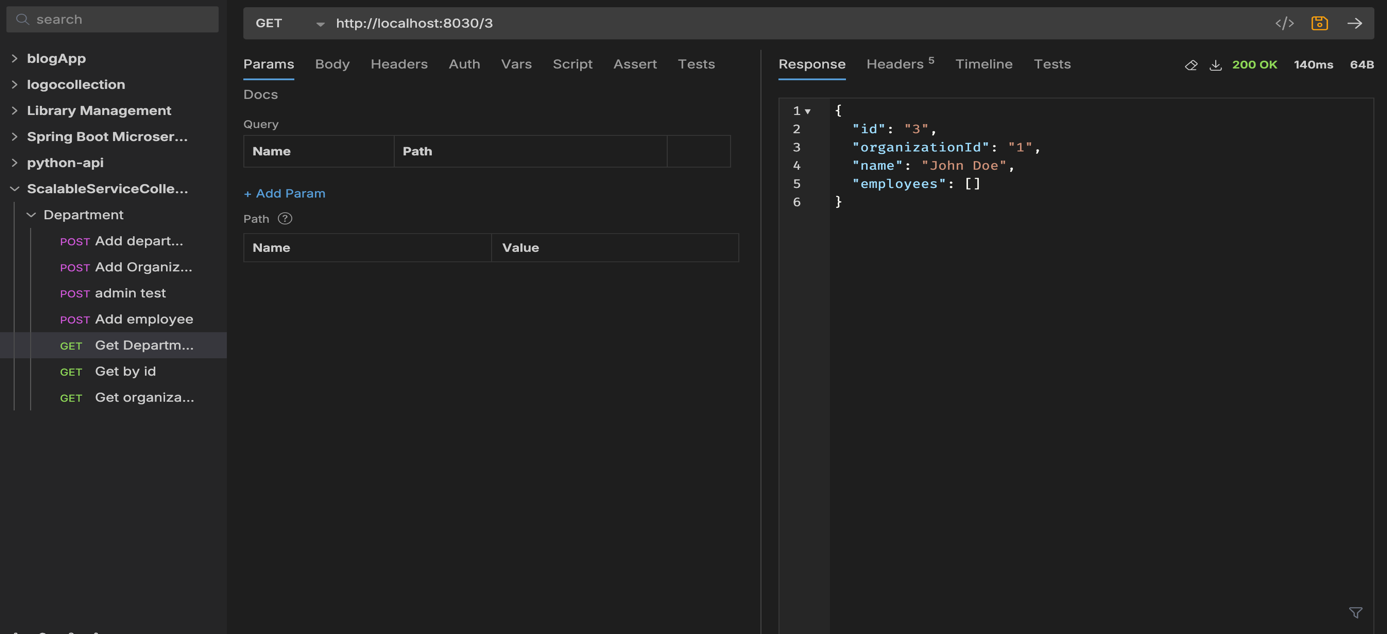
This contains about department method

Command:

$cd department-service   
$mvn clean install  
$mvn spring-boot:run  
  
Screenshot:





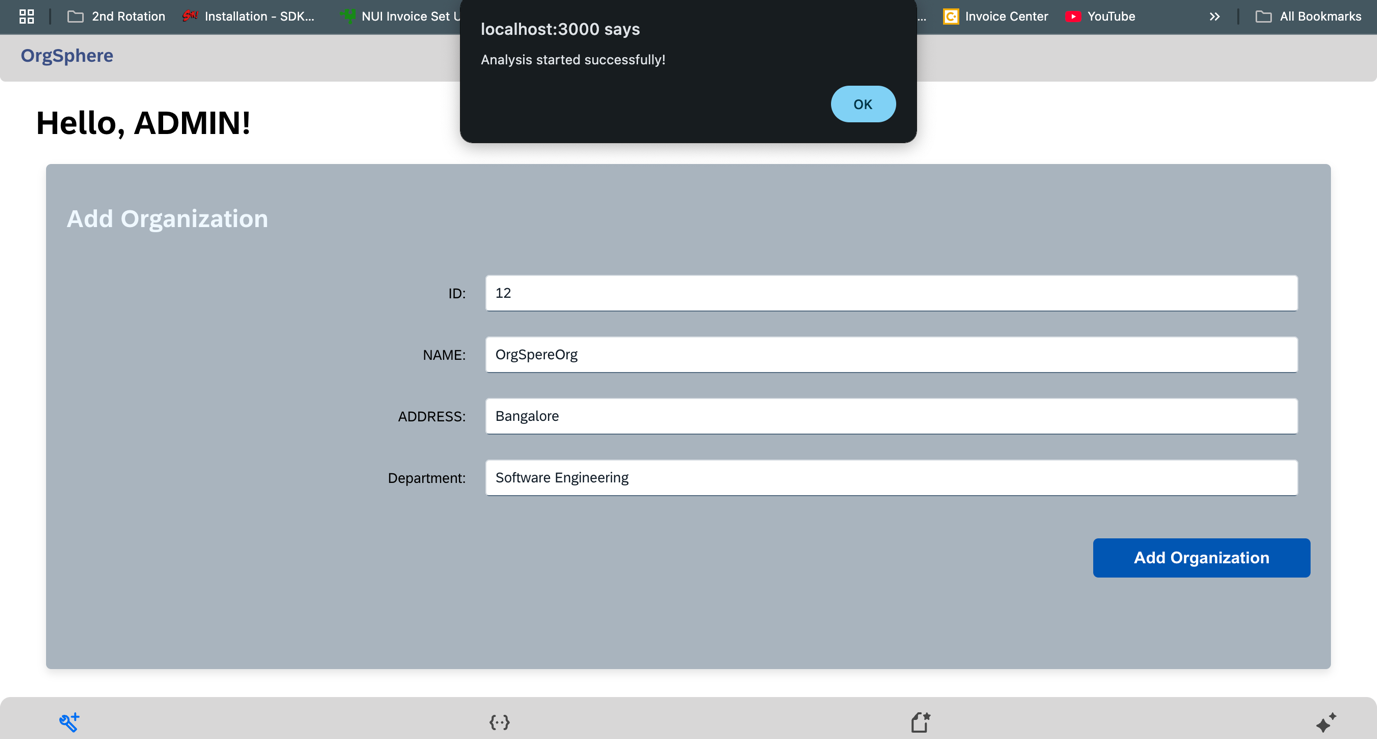
**2.2 Organization Service**

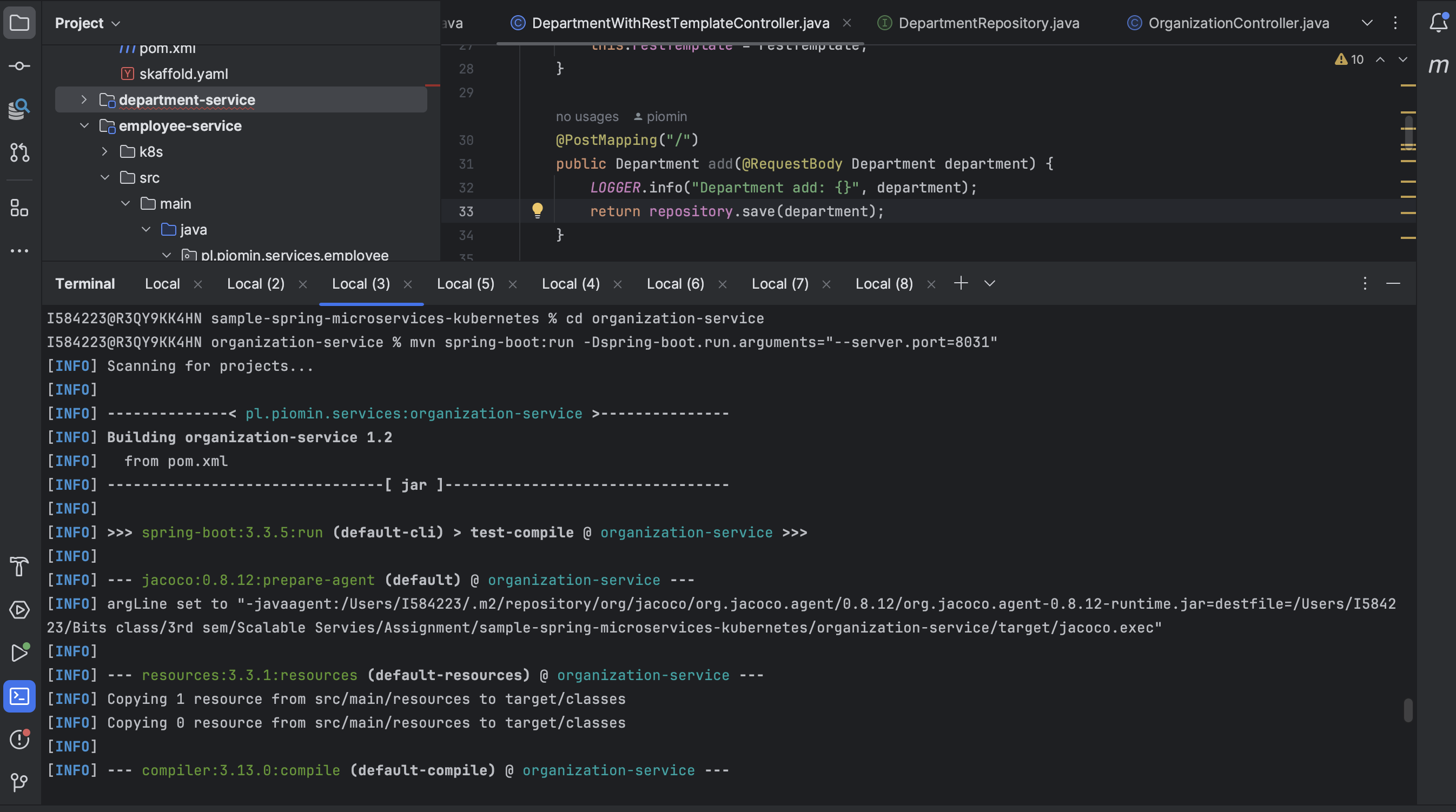
This contains about organization method

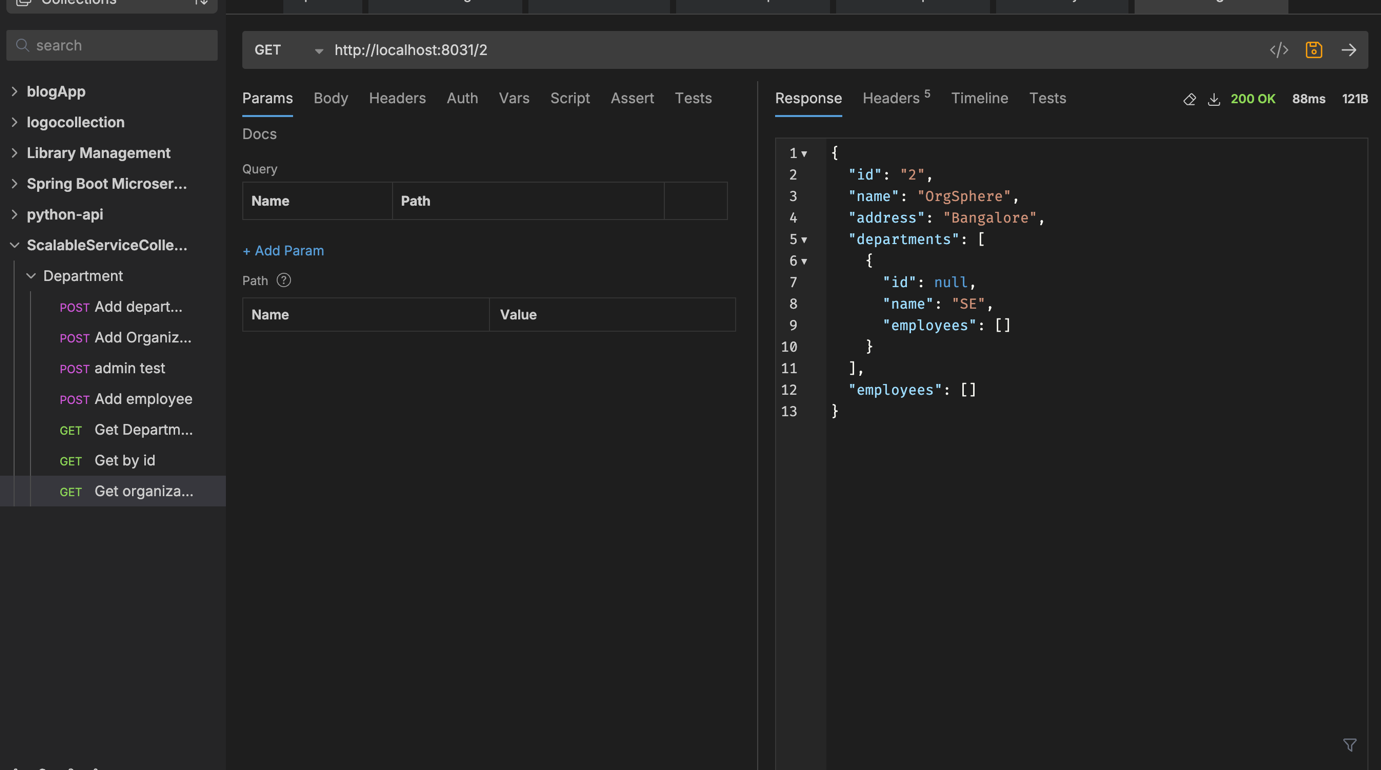
Command:

$cd organization-service   
$mvn clean install  
$mvn spring-boot:run

Screenshot:







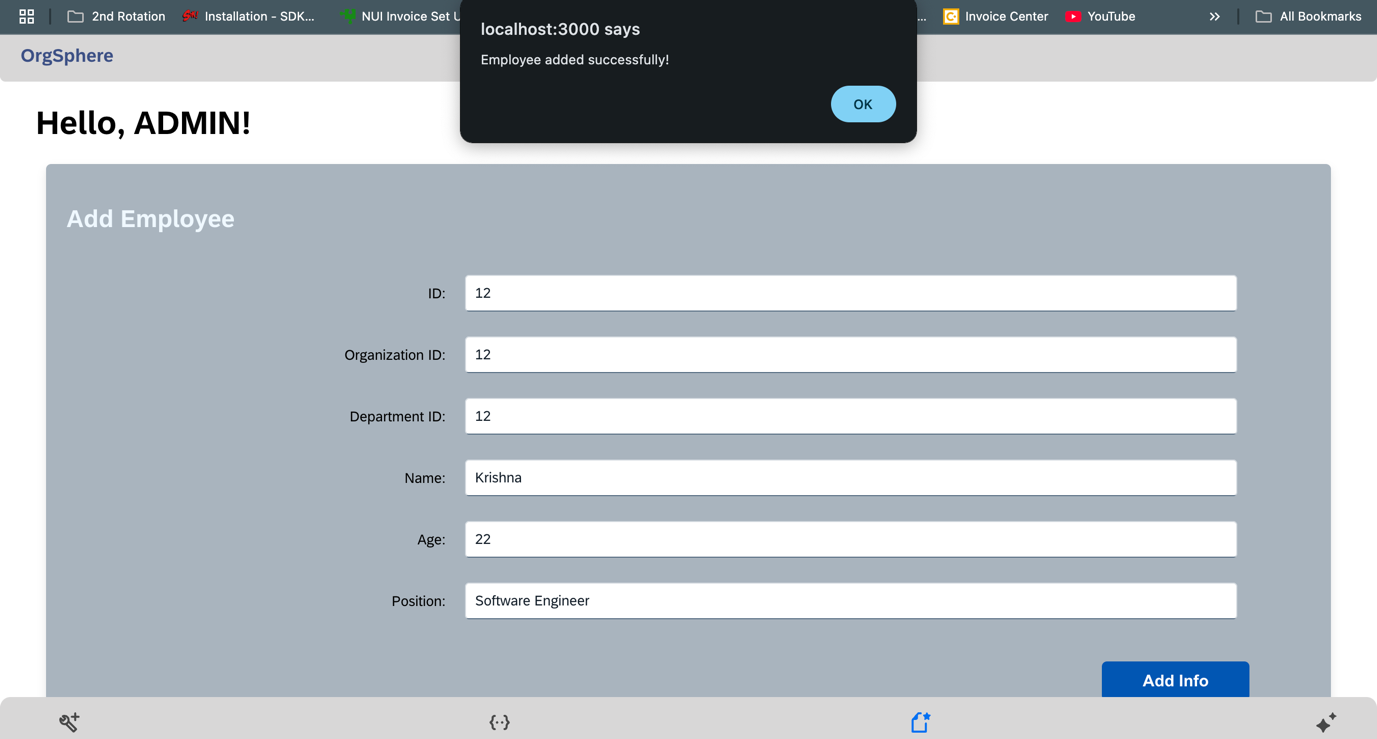
**2.3 Employee Service**

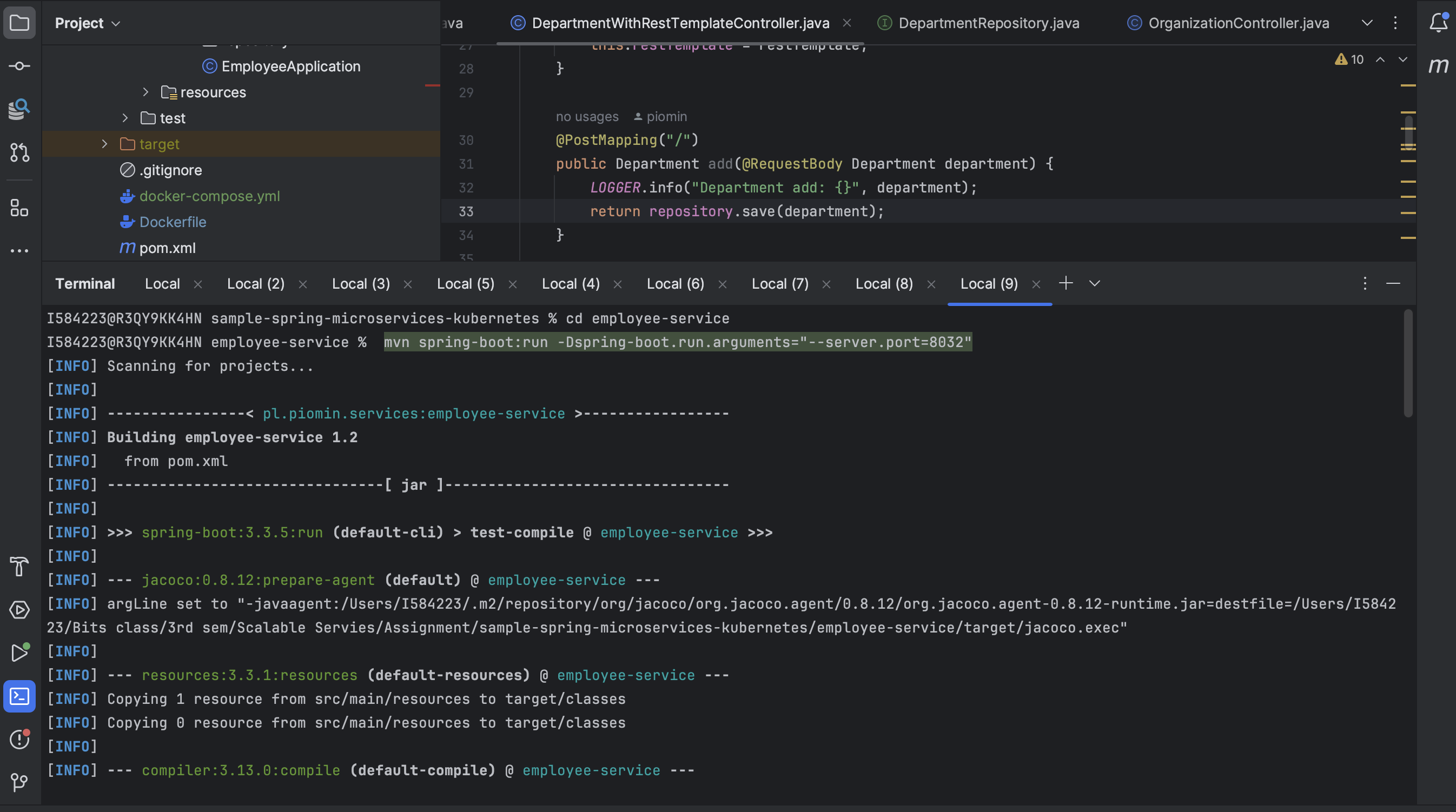
This contains about employee method

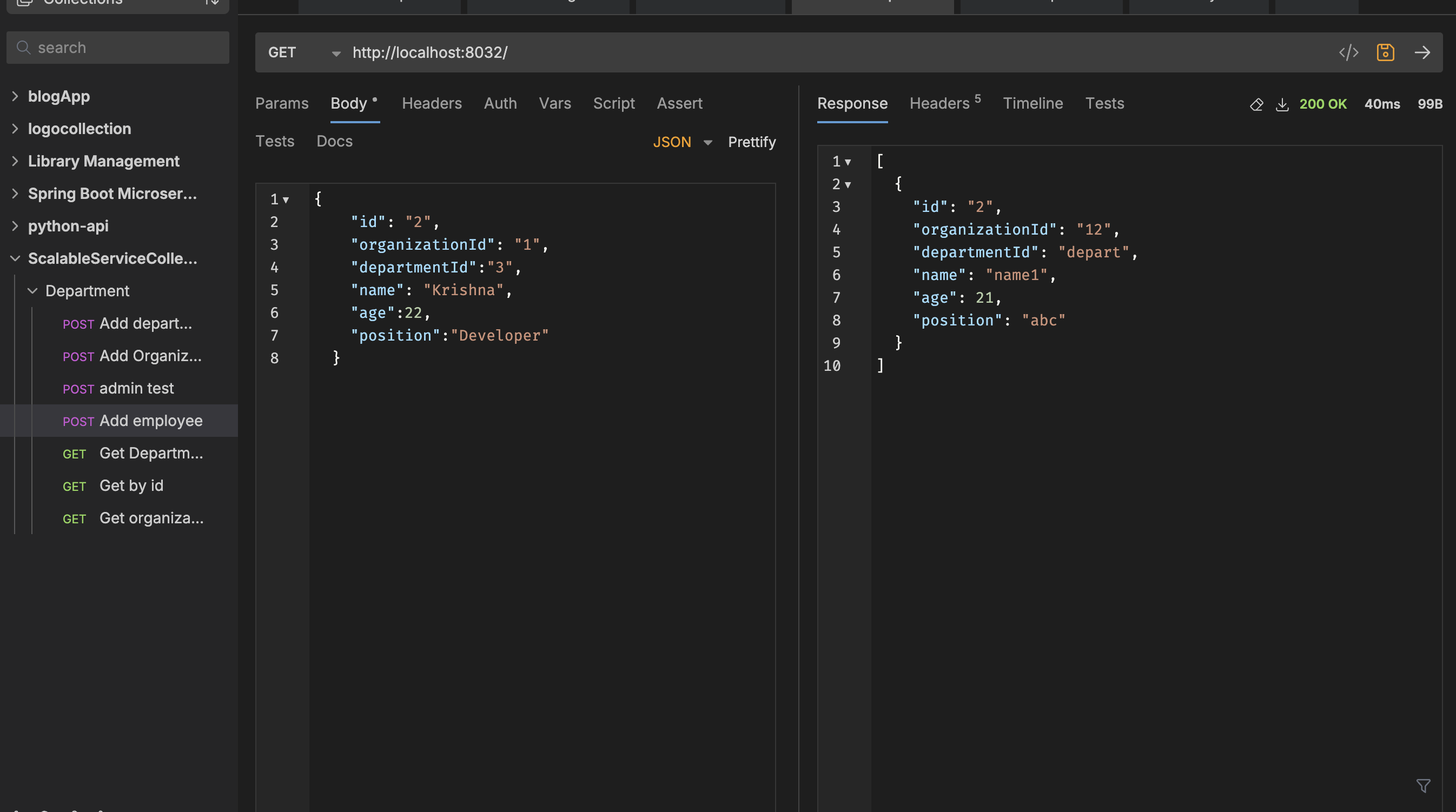
Command:

$cd employee-service   
$mvn clean install  
$mvn spring-boot:run

Screenshots:







**3. Creating Docker Images**

To deploy the services in Kubernetes, create Docker images for each service.

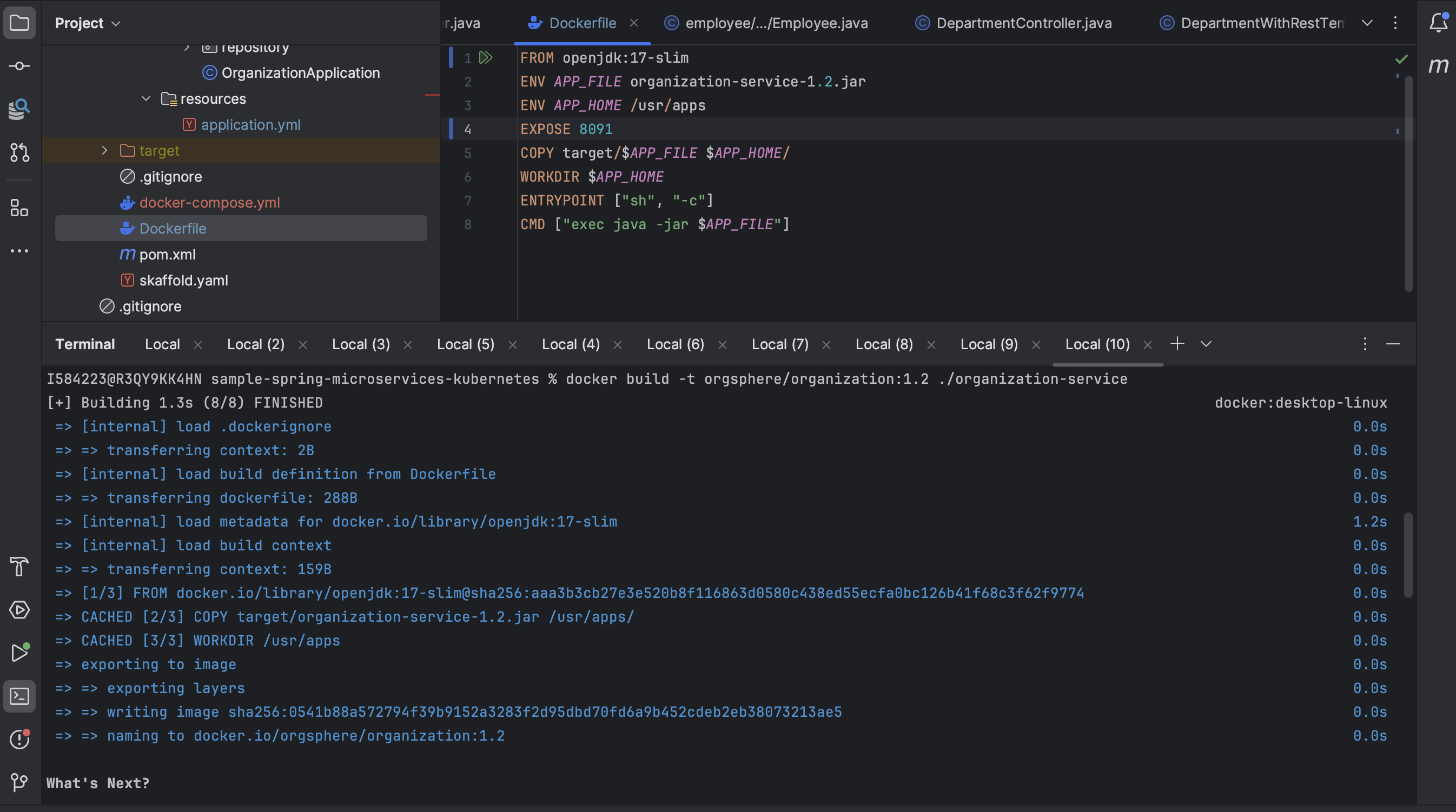
**Step 3.1: Build Docker Images**

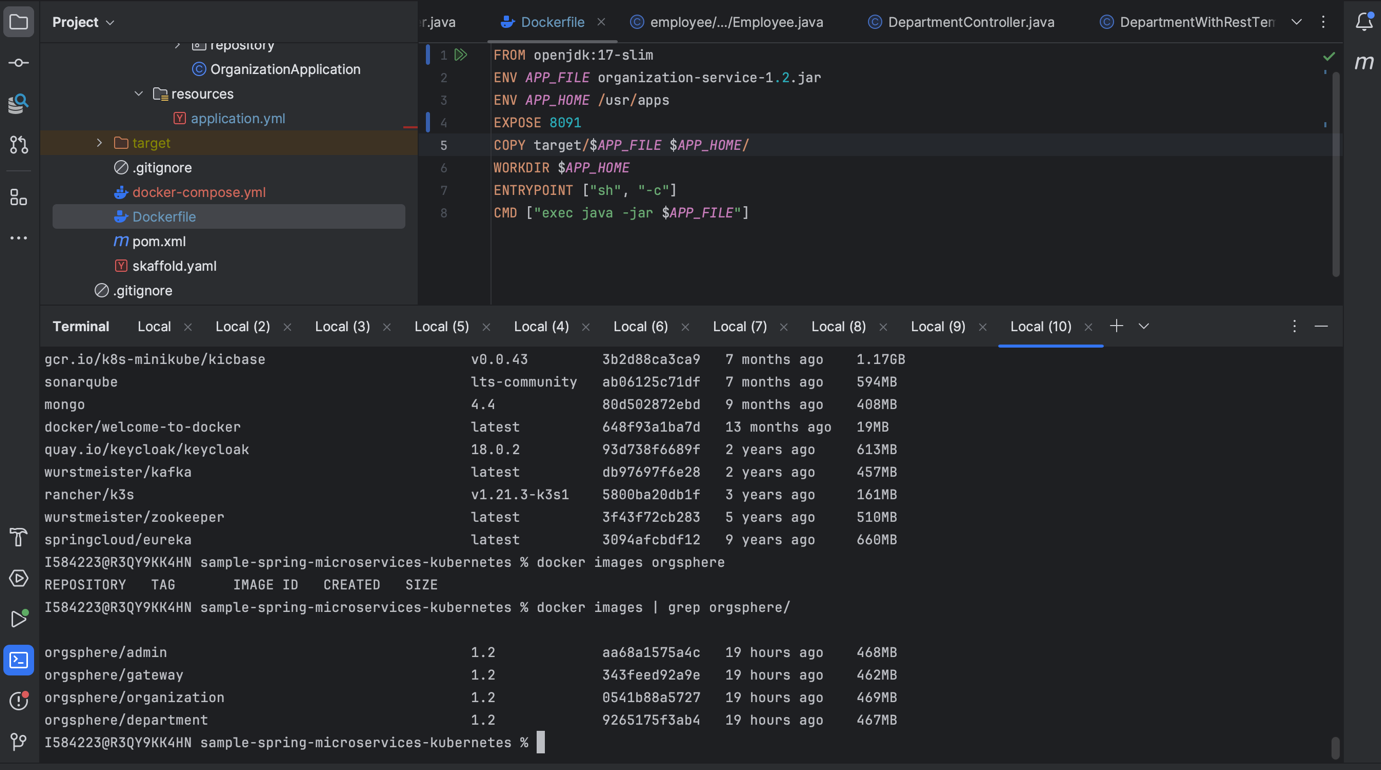
Navigate to each service directory and create a Docker image:

Using command:

cd {service\_name}

docker build -t OrgSphere/{service\_name}:1.0 .





**4. Setting Up Kubernetes**

Deploy the application on Kubernetes using **Minikube** or any Kubernetes cluster.

**Step 4.1: Start Minikube**

Start Minikube with sufficient resources:

Command:

$minikube start --vm-driver=virtualbox --memory='4000mb'



**Step 4.2: Push Docker Images to Minikube**

If using Minikube, load your images into the Minikube environment:

Command

minikube image load orgsphere/{service\_name}:1.0

Repeat for all services.

**Step 4.3: Deploy MongoDB**

Go to the k8s directory and apply MongoDB manifests:

Command:

kubectl apply -f mongo-secret.yaml

kubectl apply -f mongo-configmap.yaml

kubectl apply -f mongo-deployment.yaml

kubectl apply -f mongo-service.yaml

**Step 4.4: Deploy the Services**

Apply the YAML configurations for each service:

Command:

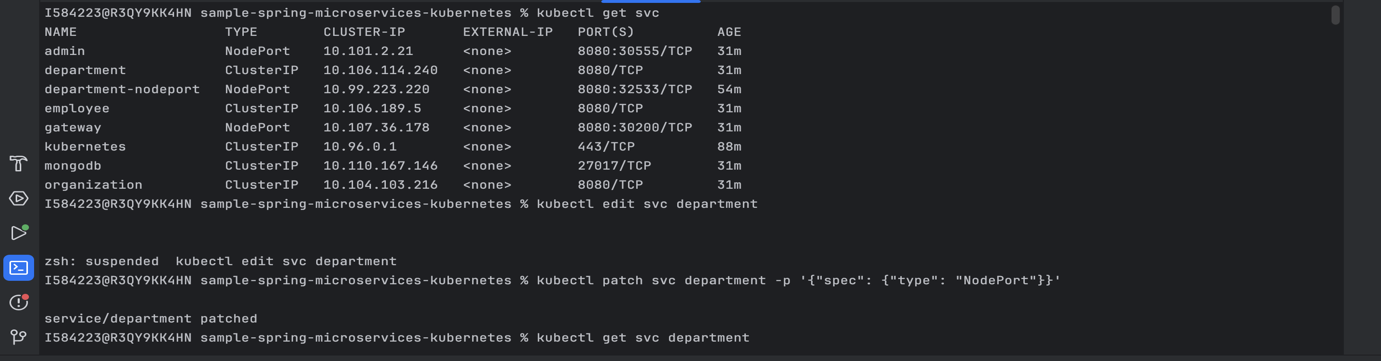
kubectl apply -f employee-service.yaml

kubectl apply -f organization-service.yaml

kubectl apply -f department-service.yaml

kubectl apply -f gateway-service.yaml

kubectl apply -f admin-service.yaml

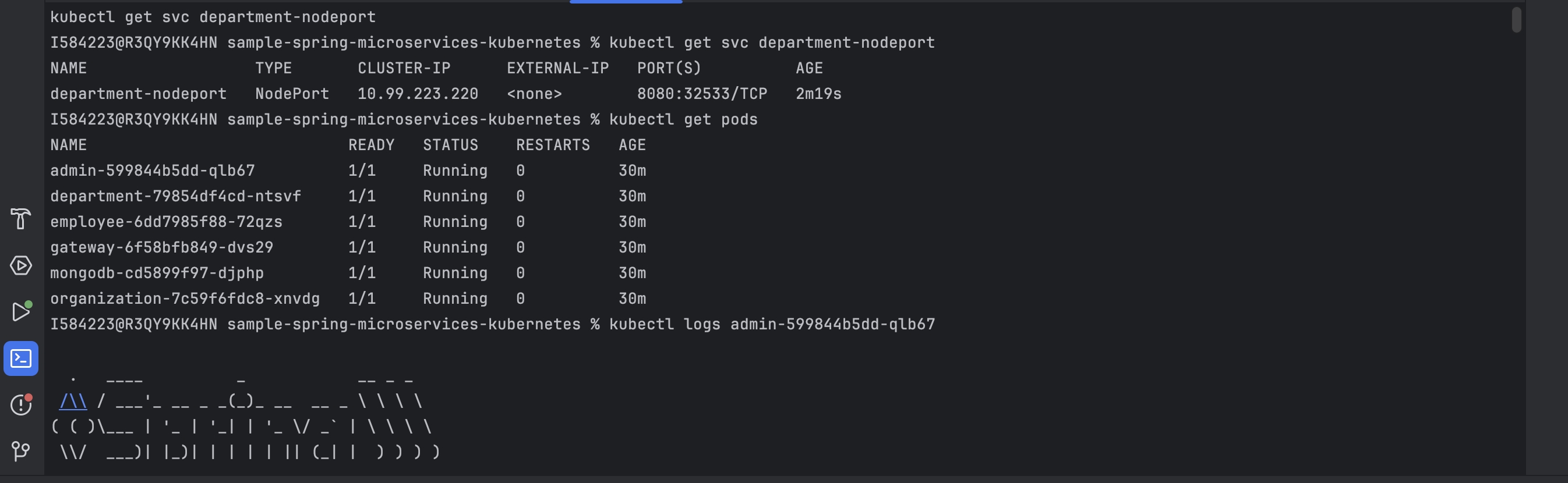


**Step 4.5: Check Pods**

Verify that all pods are running:

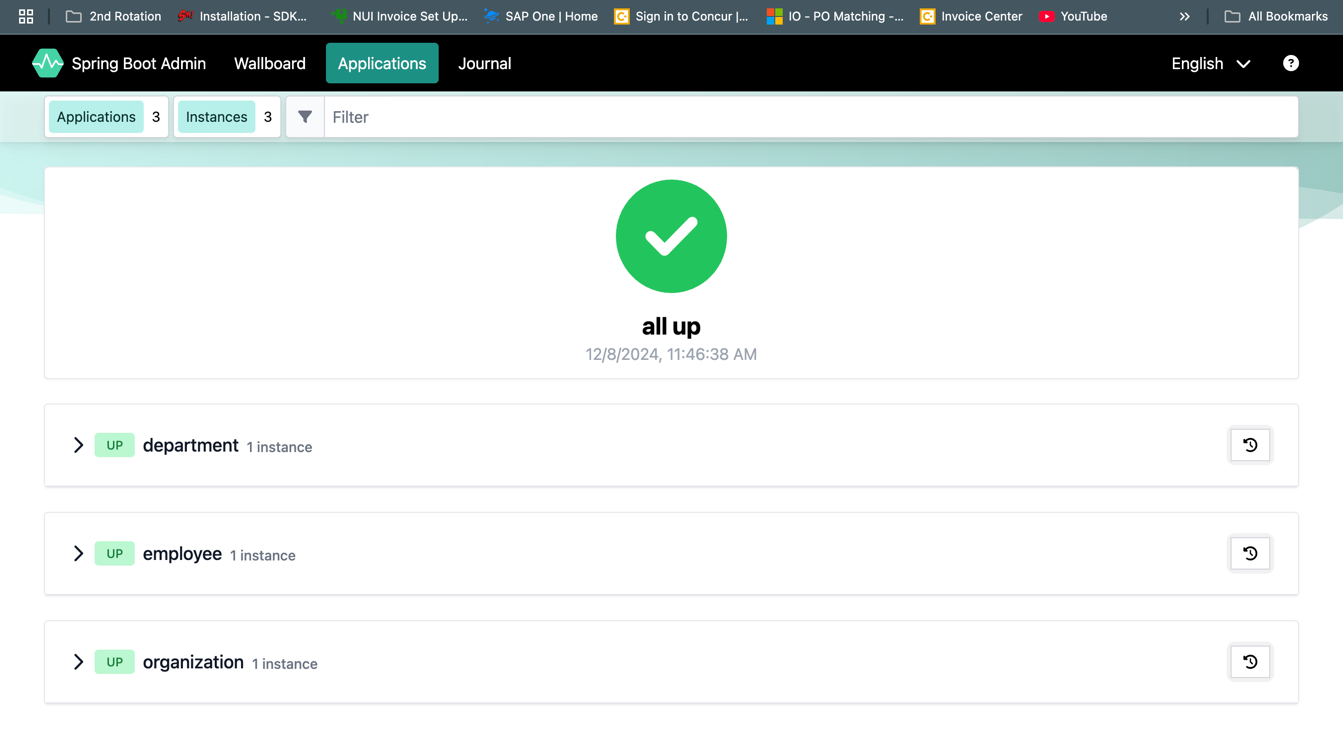
Command:

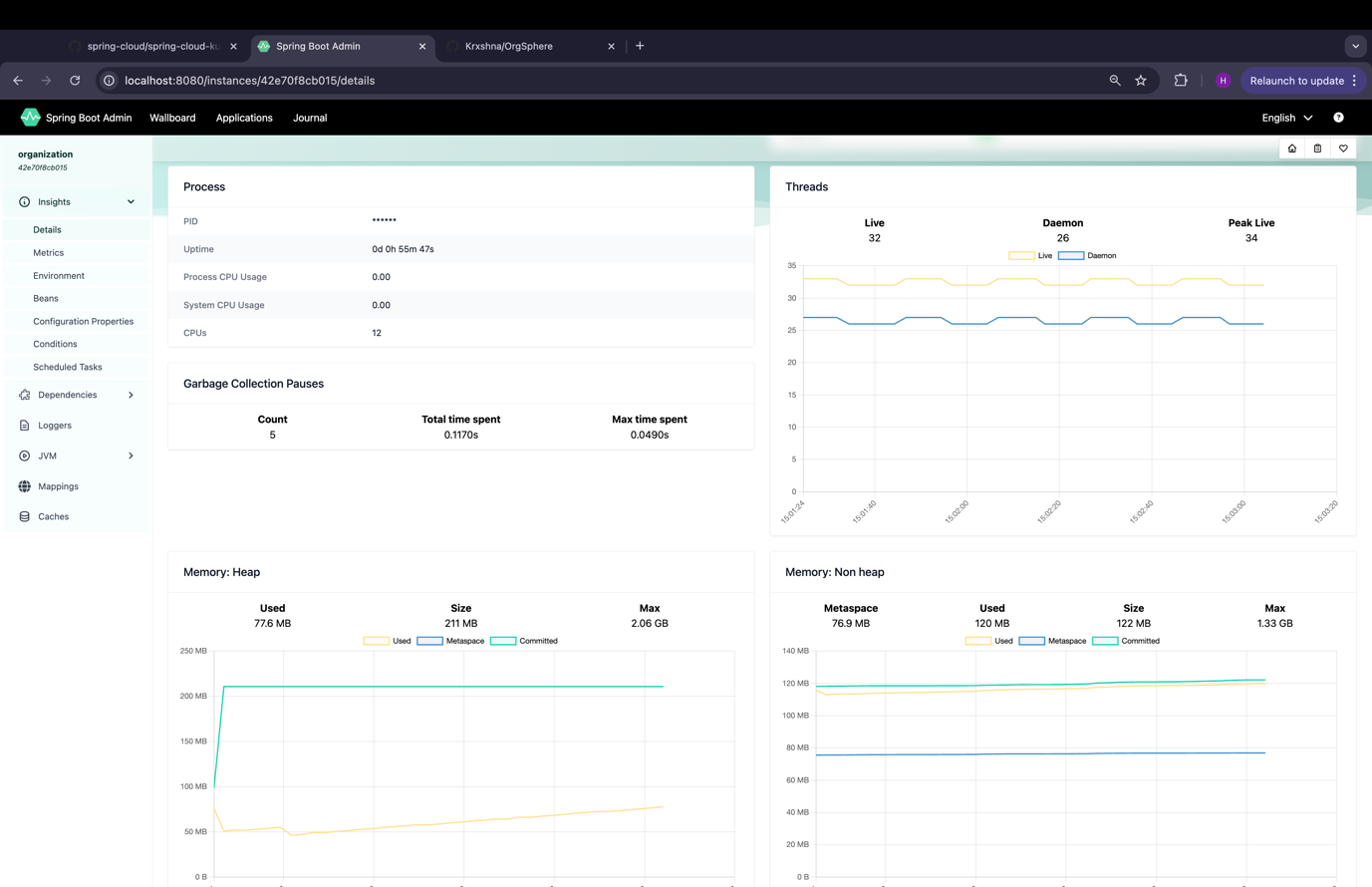
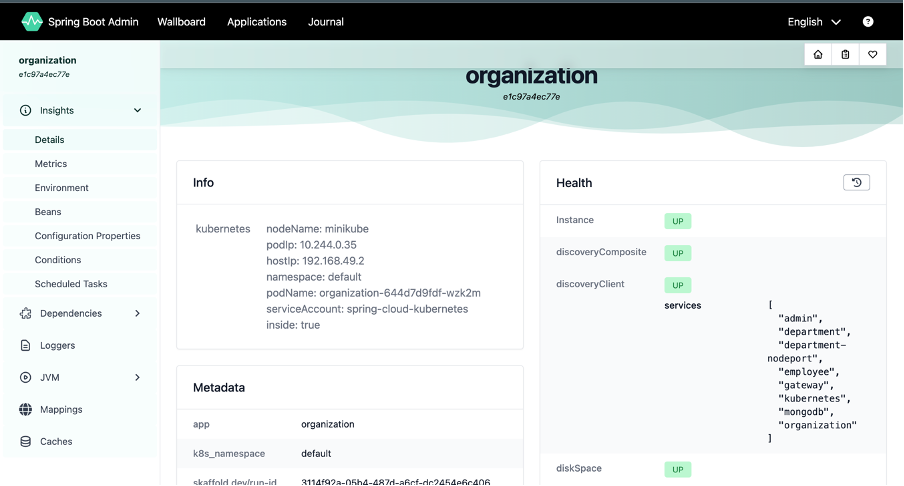
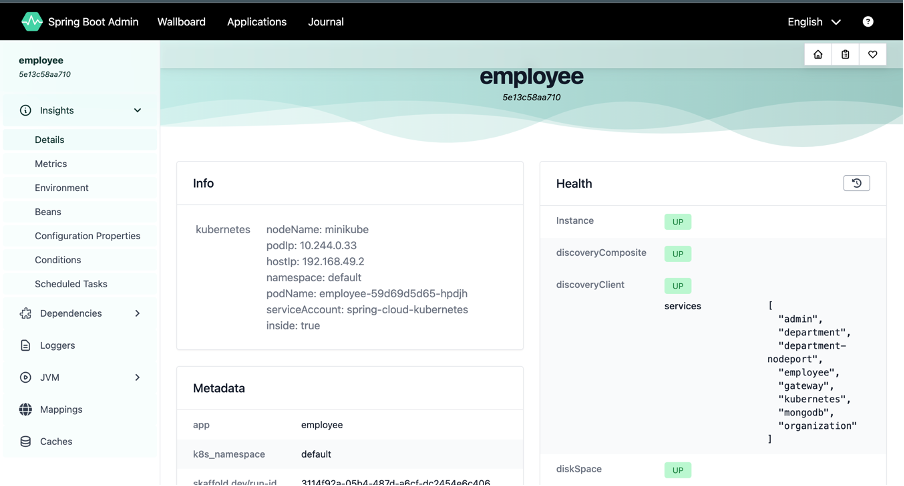
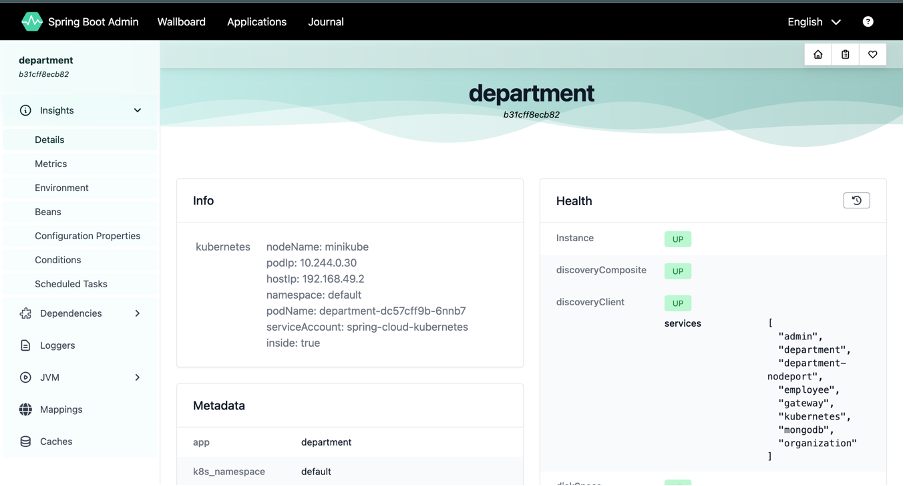
kubectl get pods



**5. Testing the Application in Kubernetes**

We can see the service are running.





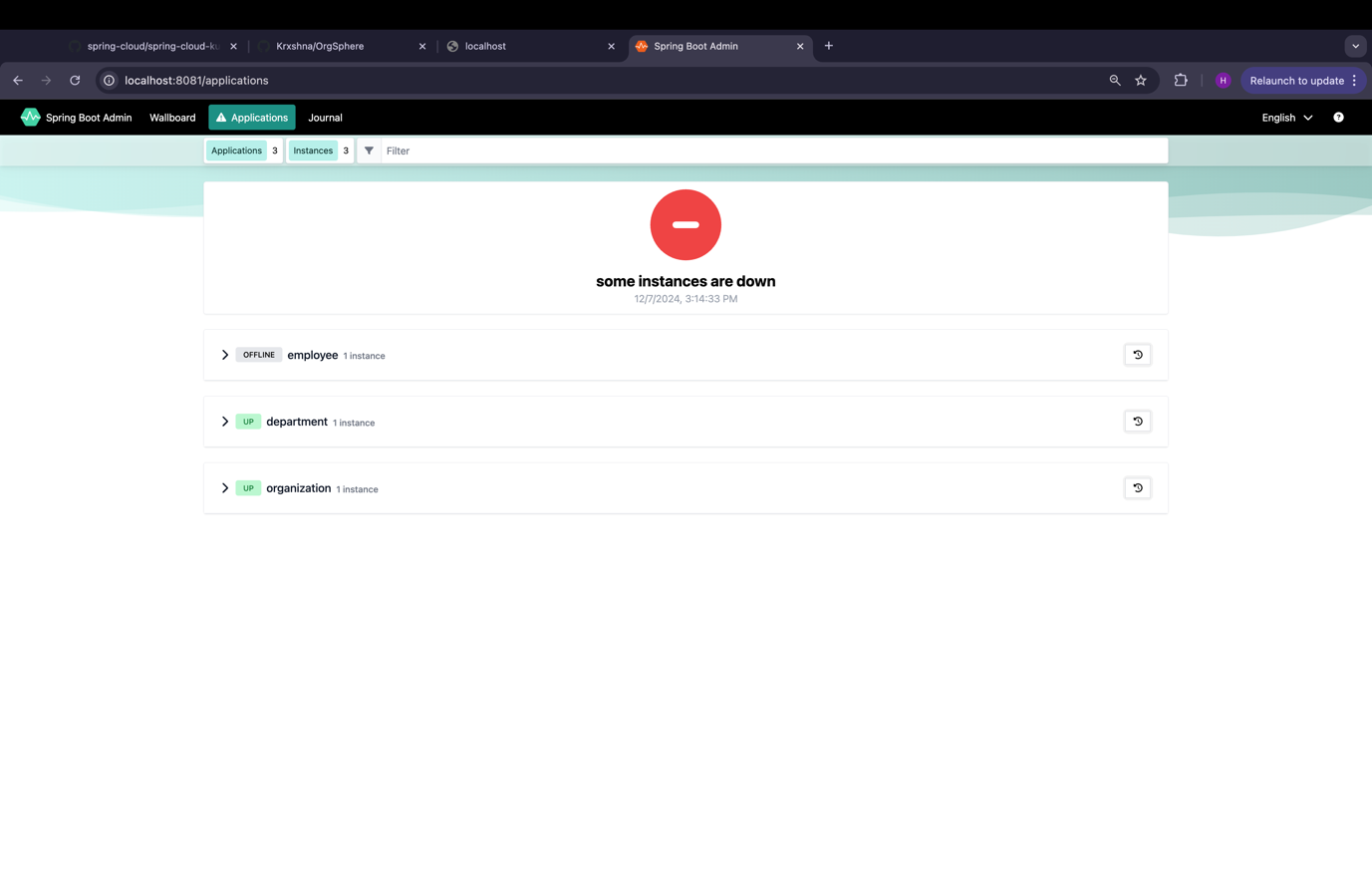
**Microservice Architecture Characteristics**

**1. Fault tolerance/circuit breaker:**

We use a circuit breaker in our application to enhance resilience by monitoring service interactions and preventing cascading failures. When a service experiences repeated failures, the circuit breaker temporarily stops requests to it, allowing time for recovery and ensuring the overall stability of the system.

Code snippet:

@GetMapping("/{id}/with-employees-and-delay")  
 public Department findByIdWithEmployeesAndDelay(@PathVariable("id") String id) {  
 *LOGGER*.info("Department findByIdWithEmployees: id={}", id);  
 Department department = repository.findById(id).orElseThrow();  
 CircuitBreaker circuitBreaker = circuitBreakerFactory.create("delayed-circuit");  
 List<Employee> employees = circuitBreaker.run(() ->  
 employeeClient.findByDepartmentWithDelay(department.getId()));  
 department.setEmployees(employees);  
 return department;  
 }



**2. Consistency between DB:**

In our architecture, the **Department**, **Organization**, and **Employee** microservices leverage a centralized MongoDB database, ensuring data consistency across these interconnected domains. This centralized approach eliminates duplication, enables seamless data sharing, and maintains uniformity in updates and queries. The **Admin** service, while independent, interacts with these services to ensure smooth operations and maintain consistency across the system.

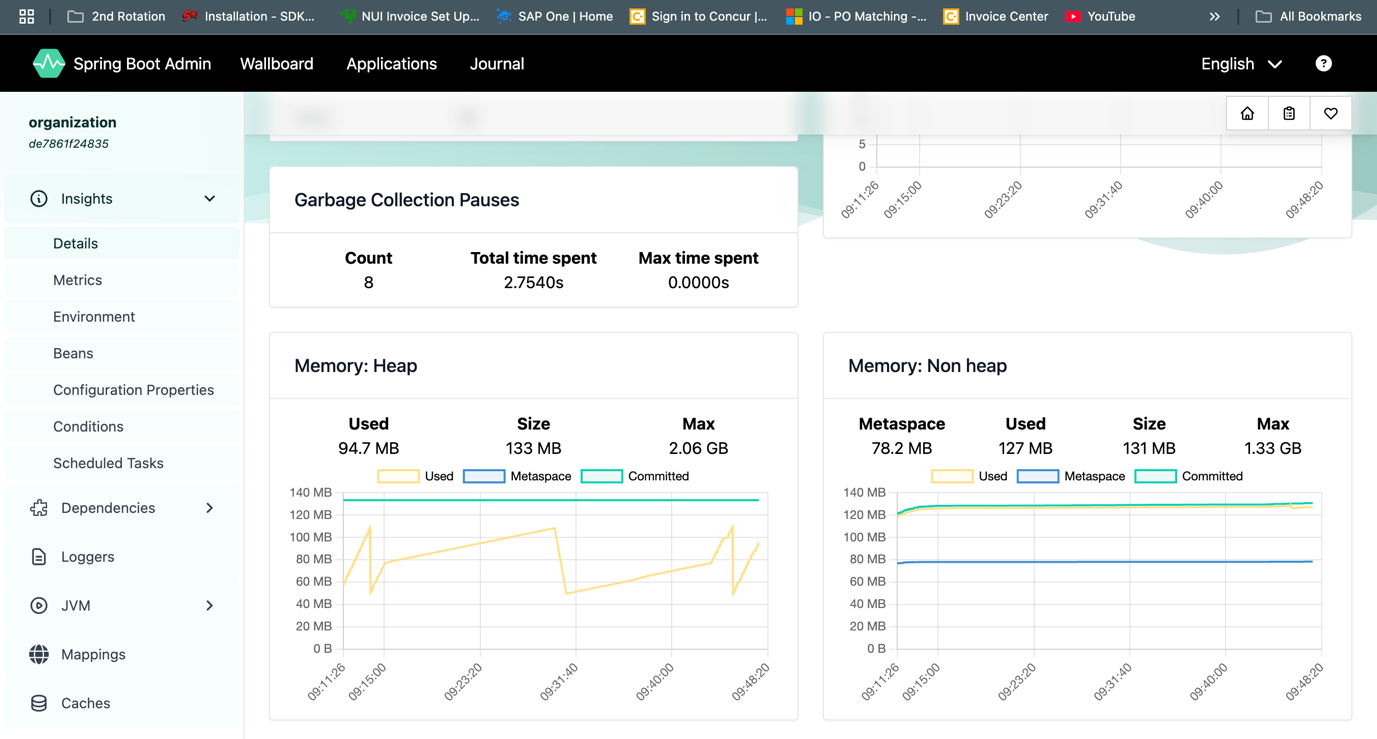
**3. Load Balancing using zuul/ribbon**

To implement **load balancing** for the four microservices (*department*, *organization*, *employee*, and *admin*), we are using **Zuul** and **Ribbon** as part of the Spring Cloud ecosystem. **Zuul**, acting as an API Gateway, routes incoming requests to the appropriate microservice based on predefined rules. **Ribbon**, integrated with Zuul, provides client-side load balancing by distributing traffic across multiple instances of a service. Together, they enhance scalability and fault tolerance by ensuring requests are evenly spread and avoiding overloading any single instance. Configuration is straightforward using Spring properties to define service instances and load-balancing policies.  
  
Code Snippet:

cloud:  
 loadbalancer:  
 ribbon:

enabled: false  
 kubernetes:  
 discovery:  
 all-namespaces: true

**4. Monitoring and Logging/Health Check**



**Repository Link:** [Click here](https://github.com/Krxshna/OrgSphere.git)

**Recording Link:** [Click here](https://drive.google.com/file/d/17sjhDk-6hweWzGnkiBJUPTu7rDN1pydZ/view?usp=sharing)

**Conclusion:**

The **OrgSphere Project** showcases the development and deployment of a microservices architecture using **Spring Boot**, **Spring Cloud**, and **Kubernetes**. It features services like **Employee Service**, **Organization Service**, and **Department Service**, demonstrating modularity, scalability, and real-world business use cases. From local setup to Docker containerization and Kubernetes deployment, the project highlights the complete lifecycle of cloud-native applications. It serves as a practical guide for building resilient, scalable, and production-ready microservices.