Microservice Implementation

Learning outcome

you will implement spring-boot microservice to achieve the following

- Service
- Separate Database
- Service Registry
- Client Side Service Discovery
- API Gateway

Scenario

Suppose, you want to build an online shop in microservice architecture. You have decided to build 4 services to fulfil you business need. Those are -

- customer microservice
- inventory microservice
- employee microservice
- order microservice

your task is to build those services by following *API-Gateway*, *Service Registry*, *Database per service* microservice pattern.

To test whether your services work properly, use postman to create collection of api requests.

Solution

To achieve the above scenario you have to build 6 services. Below some outline is given to implement the application.

Employee Service

Create a spring-boot application using spring-boot initializer.

Name: employee-service

Dependencies

- Spring web
- Spring mongodb
- lombok
- Eureka Discovery Client

In src.main.java.package create the following packages

- controller
- entity
- repository
- service

In entity package, create an employee class namely Employee with the following attributes' id, name, designation, salary. Annotate the class properly.

```
@Document(collection = "employees")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Employee {

    @Id
    private String id;
    @Field
    private String name;
    @Field
    private String designation;
    @Field
    private double salary;
}
```

In controller package, create a controller namely EmployeeController. Here, Rest methods will be implemented for employee. You can add the following code.

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

    @PostMapping("/")
    public Employee saveEmployee(@RequestBody Employee employee){
        return employeeService.saveEmployee(employee);
    }

    @GetMapping("/{id}")
    public Employee findEmployeeById(@PathVariable("id") String employeeId){
        return employeeService.findEmployeeById(employeeId);
    }
}
```

In service package, create EmployeeServicd class and add the following code

```
@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;

    public Employee saveEmployee(Employee employee) {
        return employeeRepository.save(employee);
    }

    public Employee findEmployeeById(String userId) {
        return employeeRepository.findEmployeeById(userId);
    }
}
```

In Repository package, create EmployeeRepository and add the following code.

```
@Repository
public interface EmployeeRepository extends MongoRepository<Employee, String> {
    Employee findEmployeeById(String userId);
}
```

In application.properties file `spring.data.mongodb.uri=mongodb+srv://jubair:password@cluster0.ag2rk.mongodb.net/databaseName?retryWrites=true&w=majority`

create an application.yml file in the resources folder and add the following code

```
server:
    port: 9002

spring:
    application:
        name: EMPLOYEE-SERVICE


eureka:
    client:
        register-with-eureka: true
        fetch-registry: true
    service-url:
        defaultZone: http://localhost:8761/eureka/
    instance:
        hostname: localhost
```

In EmployeeServiceApplication, add the following code.

```
@SpringBootApplication
@EnableEurekaClient
public class EmployeeServiceApplication {

    public static void main(String[] args) {
        SpringApplication.run(EmployeeServiceApplication.class, args);
    }
}
```

You have to create customer, product, order services like the employee service.

Product Service

product entity consists of id, name, description, price

Customer Service

customer entity consists of id, name, address, age

API-Gateway service

Dependency

- Eureka Discovery client
- Gateway

In api-gateway, you have to add application.yml file. Add other routes for customer, product and order.

```
server:
  port: 9003

spring:
  application:
    name: API-GATEWAY
  cloud:
    gateway:
      routes:
        - id: EPLOYEE-SERVICE
          uri: lb://EPLOYEE-SERVICE
          predicates:
            - Path=/employees/**

eureka:
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://localhost:8761/eureka/
    instance:
      hostname: localhost
```

Enable Eureka client for api-gateway also

```
@SpringBootApplication
@EnableEurekaClient
public class ApiGatewayApplication {
    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }
}
```

Service Registry

Dependency

- Eureka server

In service-registry, you have to add application.yml file.

```
server:
  port: 8761

spring:
  application:
    name: SERVICE-REGISTRY

eureka
  server:
    register-with-eureka: false
    fetch-registry: false
```

Enable Eureka server for servie-registry

```
@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {
    public static void main(String[] args) {
        SpringApplication.run(ServiceRegistryApplication.class, args);
    }
}
```

Bonus

For Bonus marks do the following.

- create an order service which can communicate customer, product services.

Order Service

order entity consists of id, date, customerId, productId, employeeId. It has same dependencies like employee service.

To do that you have to implement RestTemplate. In OrderServiceApplication class you have to add some additional code. Like the following

```
@SpringBootApplication
@EnableEurekaClient
public class OrderServiceApplication {
    // other code
    @Bean
    @LoadBalanced
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }
}
```

Create a package valueObject. There you have to create customer, product classes which are like customer, product entities to receive customer, product data along with order data. Add another class ResponseValueObject like the following to return a CustomerProductOrder.

```
public class ResponseValueObject{
    private Customer customer
    private Product product;
    private Employee employee;
    private Order;
}
```

Add the following code in OrderService along with other code.

```
@Service
public class UserService {

    /* other code
    ----
    ----
    */

    @Autowired
    private RestTemplate restTemplate;

    public ResponseValueObject getUserWithDepartment(Long userId) {
        ResponseValueObject ResponseValueObject = new ResponseValueObject();

        Order order = orderRepository.findOrderById(orderId);
        Product product = restTemplate.getForObject("http://PRODUCT-SERVICE/products/" + order.getProductId(), Product.class);
        Customer customer = restTemplate.getForObject("http://CUSTOMER-SERVICE/customers/" + order.getCustomerId(), Customer.class);
        Employee employee = restTemplate.getForObject("http://Employee-SERVICE/employees/" + order.getEmployeeId(), Employee.class);

        // The bellow line is also correct. static port is replaced by service
        // Customer customer = restTemplate.getForObject("http://localhost:9001/customers/" + order.getCustomerId(), Customer.class)

        ResponseValueObject.setCustomer(customer);
        ResponseValueObject.setProduct(product);
        ResponseValueObject.setProduct(employee);
        ResponseValueObject.setOrder(order);

        return ResponseValueObject;
    }
}
```

THANK YOU!