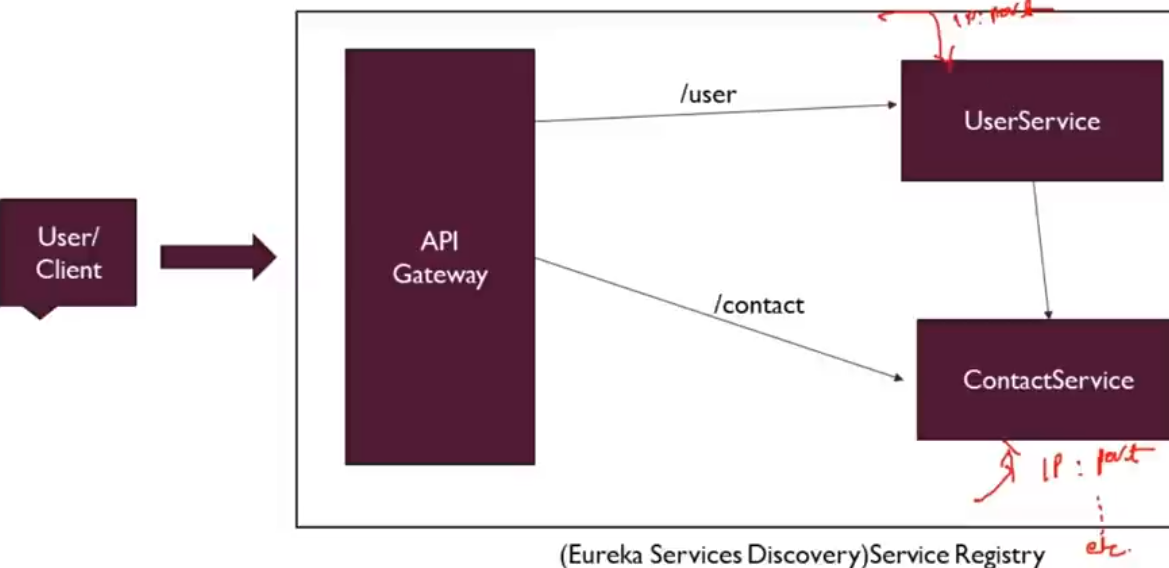






**PROJECT**

****

**REST Template**

It is used to call Microservices Synchronously,

It has a method getForObject which takes a url and a class(eg. List.class). This class indicates the return type that we are expecting from service, that we have called

Fault Tolerance in Spring boot -> If there is an fault, what is the impact of that fault. It is the property that enables a system to continue operating properly in the event of the failure of some of its components.

Resilience in Spring boot -> It means how many faults a system can tolerate.

Eureka Server (Spring Cloud Discovery)

How Eureka is different from RestTemplate.

**How to make microservice more Robust and Fault Tolerance?**

**Hytrix** is fault tolerance library developed by Netflix. It is designed to handle the challenges of latency and fault tolerance in Distributed Systems, particularly in microservices architecture. The primary Goal of hytrix is to improve the resilience of services by isolating and controlling the interactions between them. Hytrix provides features like Circuit Breaker, Fallback Mechanism and Request Timeouts.

**Resilience4J** is lightweight fault tolerant library inspired by hytrix. It provides features like circuit breaker, rate limiter, retries and bulkheads. Resilience4J has a more modular and functional approach and integrates well with Java 8 and higher.

**How Microservices can communicate with each other?**

There are 2 ways in which MS communicates:

1. Synchronous Communication

Rest Template, FeignClient, WebClient can be used for synchronous communication between 2 microservices. Ideally, we should minimize the number of synchronous calls between micro services because the networks are brittle and they introduced latency.

1. Asychronous Communication

In this type of communication, the client does not wait for response, instead it just sends the message to the message broker. Kafka, RabbitMQ can be used for asynchronous communication across microservices to achieve eventual consistency.

**Micro Service Configuration**

Value annotation tricks

1. Giving default value @Value(“variable in properties file: default value”)
2. Managing Lists- If we have property value separated by commas in application.properties then we have an option to take it as a list
3. Managing Maps- just like lists we can manage maps as well.

@ConfigurationProperties -> It makes available all the configurations from application.properties available at the class.

Spring Boot Actuator:- It exposes a specific rest end point where we can get all the configuration properties that are exposed in our app, either by our self or by spring framework itself. For this to work we need to add actuator dependency. For example:- to expose all endpoints in actuator  
**management.endpoints.web.exposure.include=\***

**Spring Profiles**

Spring has feature called profiles which we can leverage, in order to have different configuration for different environment.

Naming convention for the profiles: - application-<profileName>.extention

By default, default profile are always active. The name of active profile is application.properties or application.yaml whichever we use.

To make any other profile as active provide this command

spring.profiles.active=profileName

**Explain MicroService Architecture**

It is a way in which we build our application by breaking them down into smaller pieces and deploying them seperately(possibly on seperate machines) and have them talk to each other.

Basically breaking monolithic into different pieces, which can be independently developed,deployed & maintained.

**What is monolithic Architecture**

In monolithic Architecture our application gets build and deployed as single unit.

It is like a big container in which all s/w components of a component are clubbed inside single package.

**Advantage Of Microservice.**

a. Small Change -> Small Impact. Unlike monolithic, small change does not bring large impact in terms of building and deploying those changes.

b. Technology diversity: We can have different microservices running in different technologies. for example we can have 1 microservice running in java, other in python and other in .net, which is fine as long as they are communicationg in json.

c. Increase Fault Tolerence: Single Microservice failure does not bring down the whole system.

d. Independent Deployment.

e. Provides Scalability.

**What is spring cloud.**

**Challenges faced in using microservices.**

a. Increase in Complexity & monitoring.

b. Co-ordinated deployment- in case microservices are dependent on each other, deployment must be co-ordinated.

c. Increase Efforts for configurations

**In which case microservice architecture best suited?**

a. When we need scalability.(Less hits less number of servers, more hits more number of servers)

b. When we need frequent deployments

**What is API Gateway?**

Api Gateway is a server which we act as single entry point into the system.It acts as abstraction layer to UI.

Advantages of having API gateway is that we can implement Authentication, Monitoring, Load balancing, caching, Logging,etc.

Api gateway is also responsible for request routing as well. i.e. all requests made by the client go through API gateway.

Disadvantage of API gateway.

* Since we need to hop api gateway, the speed will decrease.
* If API gateway is down, then our whole application will be down.

Most popular API gateway implementation are-

* Netflix API Gateway(Zuul) - Used by Most of the projects
* Amazon API Gateway
* Azure API Gateway

We can create our own API Gateway.

**What is Hystrix in Micro Services?**  
It is used to make our application fault tolerant.  
For Example: In case of cascading calls amongst MS where MS A calls MS B and MS B calls MS C and if MS C fails then MS B fails which lead to failure of MS A. To avoid this cascading failure we use the framework known as hystrix where if MS C fails then MS C return default response and it does not make it as fail. This default response is then passed to MS B then to MS A.

**What is Service Discovery?**  
If one MS wants to discover the URL & communicate to another service then for that we can use service discovery.   
There are 2 types of service discovery  
 a. Client Side Discovery  
 b. Server Side Discovery  
Consul(by HashiCorp) and Eureka(by netflix) are used for service discovery.

**What is Circuit Breaker.**

It is micro service design pattern which is used to handle fault tolerance and increase reliability.  
Suppose a Microservice ‘A’ is internally calling another Microservice ‘B’ and ‘B’ has some fault. Hence we will stop calling the faulty Microservice ‘B’. Instead, we call a dummy method that is called a ‘Fallback Method’. Therefore, calling a fallback method instead of an actual service (due to a fault) is called breaking the circuit. That’s why, we call this as a ‘Circuit Breaker’ Pattern.

We can achieve this functionality easily with the help of annotation @CircuitBreaker without writing a specific code.

Moreover, there are generally three states of a Circuit Breaker Pattern :

1. **Closed**: In this state, The circuit breaker allows microservices to communicate as usual and monitor the number of failures occurring within the defined time period. If the failure count exceeds the specified threshold value, the circuit breaker will move to the Open state. If not, it will reset the failure count and timeout period.
2. **Open**: In this state, the communication between microservices is completely blocked. The circuit breaker will remain in the Open state until the timeout period ends. Then, it will move into the Half-Open state.
3. **Half Open**: In this state, the circuit breaker will allow a limited number of requests. If those requests are successful, the circuit breaker will switch the state to Closed and allow normal operations. If not, it will again block the requests for the defined timeout period.

Netflix Hytrix & Resilience4J are common libraries to implement circuit Breaker Pattern.

**What is Fault tolerance?**

Ability of micro services to perform their usual operation even in case of failure.

**What is Rate Limiter?**

It is used to control the rate at which requests are made to a particular service or API. Commonly used libraries for implementing Rate limiter are: Google Guava Rate Limiter, Netfilx Hyxtrix, Resilience4J, Spring Cloud Gateway

**What is retry mechanism?**

It is a mechanism, in which failed operation is executed multiple times before considering it permanently failed. Libraries used for implementing retry mechanism are: Retry4J, Resilience4J, Guava Retry.

**What is Load Balancing?**

Practice of distributing network traffic across multiple instance of services. Zuul, Spring Cloud Load Balancer, Google Cloud Load Balancing, Amazon ELB, etc

Load Balancer v/s API Gateway

|  |  |  |
| --- | --- | --- |
|  | Load Balancer | API Gateway |
| Purpose | Distributes network traffic over multiple instance of service. | Act as intermediator between client and collection of microservices. It provides authentication, authorization, rate limiting, caching, logging, monitoring, etc. |
|  |  |  |
|  |  |  |
|  |  |  |

Micro Services Design Pattern

1. Event Sourcing
2. SAGA
3. Database Per Micro Service
4. CQRS
5. API Gateway
6. Strangler
7. Circuit Breaker
8. BFF
9. Consumer-Driven Contract Tracing
10. Externalized Configuration

**Similarity & Difference between getForEntity & getForObject**

both getForEntity and getForObject are methods **used to make HTTP GET requests** to retrieve data from a remote service or API.

Where as postForEntity and postForObject are used to make HTTP POST requests to send the data to a remote service or API.

Difference

1. **Return Type**:

* getForEntity returns a ResponseEntity containing the full HTTP response (status, headers, body).
* getForObject returns the parsed response body directly, without the HTTP response metadata.

1. **Usage Scenario**:

* Use getForEntity when you need access to the complete HTTP response, including headers and status, in addition to the response body.
* Use getForObject when you are primarily interested in the response body and want a more concise way to retrieve it.

1. **Flexibility**:

* getForEntity provides greater flexibility and control over the entire response, making it suitable for cases where you need to examine the response metadata.

1. **Convenience**:

* getForObject is more convenient when you are primarily focused on deserializing the response body into a Java object.

Syntax:

RestTemplate restTemplate = new RestTemplate();  
String url = "https://api.example.com/data";  
ResponseEntity<String> responseEntity = restTemplate.**getForEntity(url, String.class)**;

String responseBody = restTemplate.**getForObject(url, String.class)**;

HttpEntity<MyDataObject> requestEntity = new HttpEntity<>(requestBody, headers);  
ResponseEntity<String> responseEntity = restTemplate.**postForEntity**(url, requestEntity, String.class);

// Create a request entity with the JSON object and headers

HttpEntity<MyDataObject> requestEntity = new HttpEntity<>(requestBody, headers);

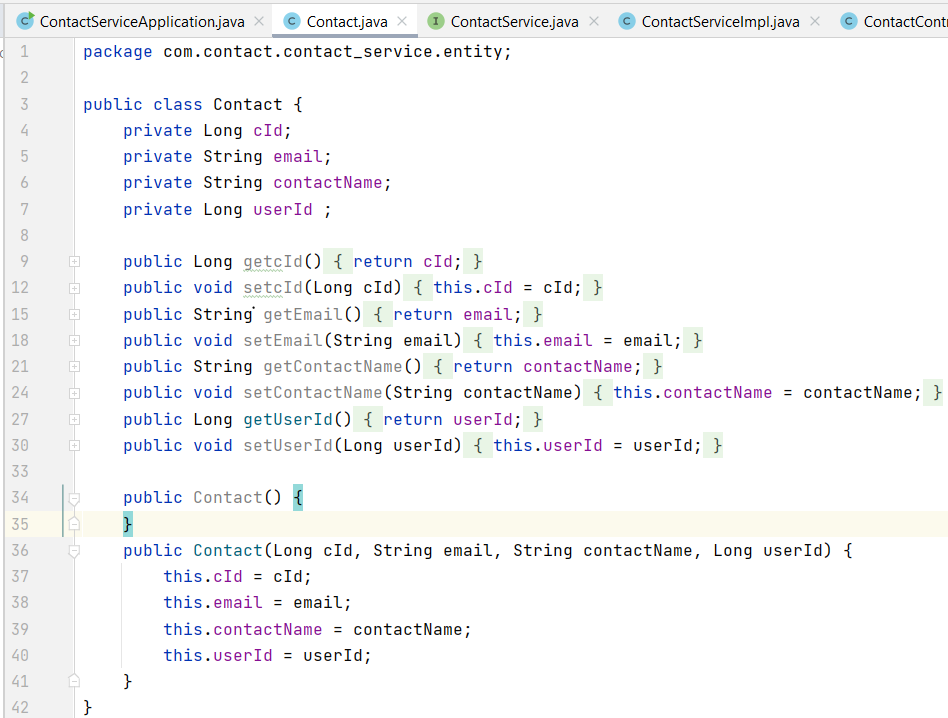
// Send the POST request and directly receive the parsed response body

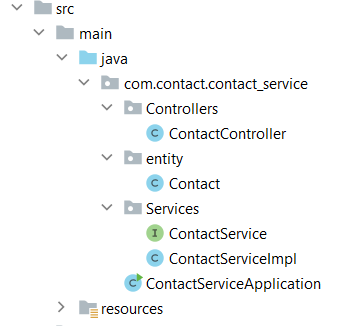
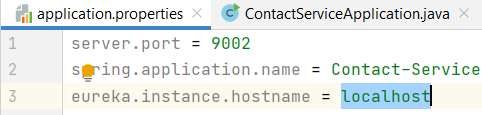
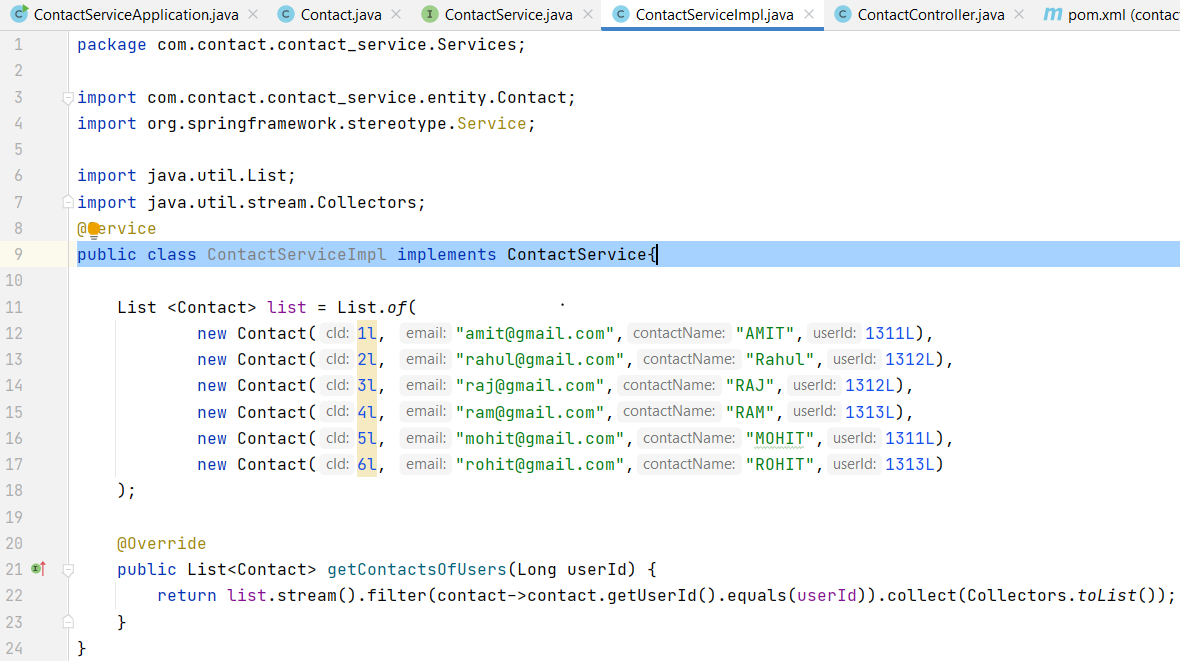
String responseBody = restTemplate.**postForObject**(url, requestEntity, String.class);

restTemplate.delete(url);

Source Code

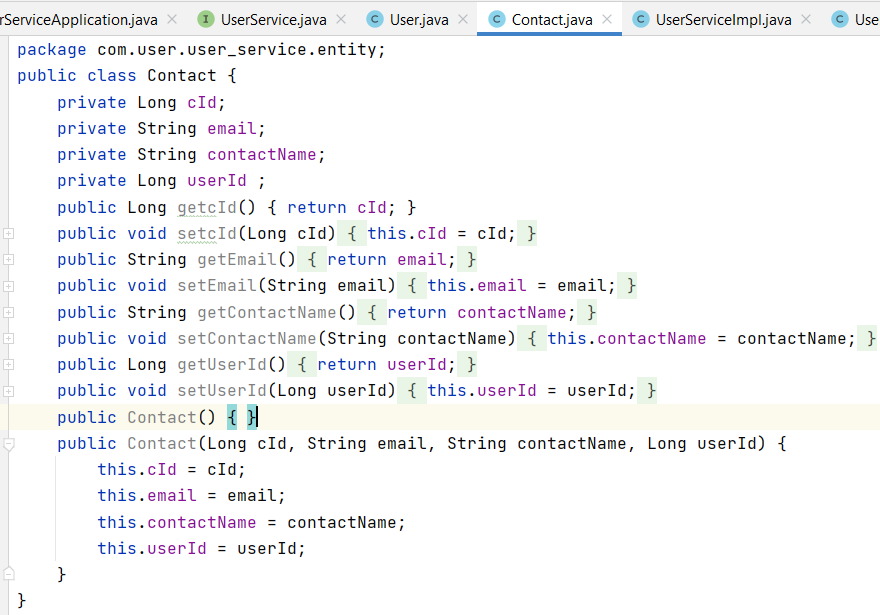
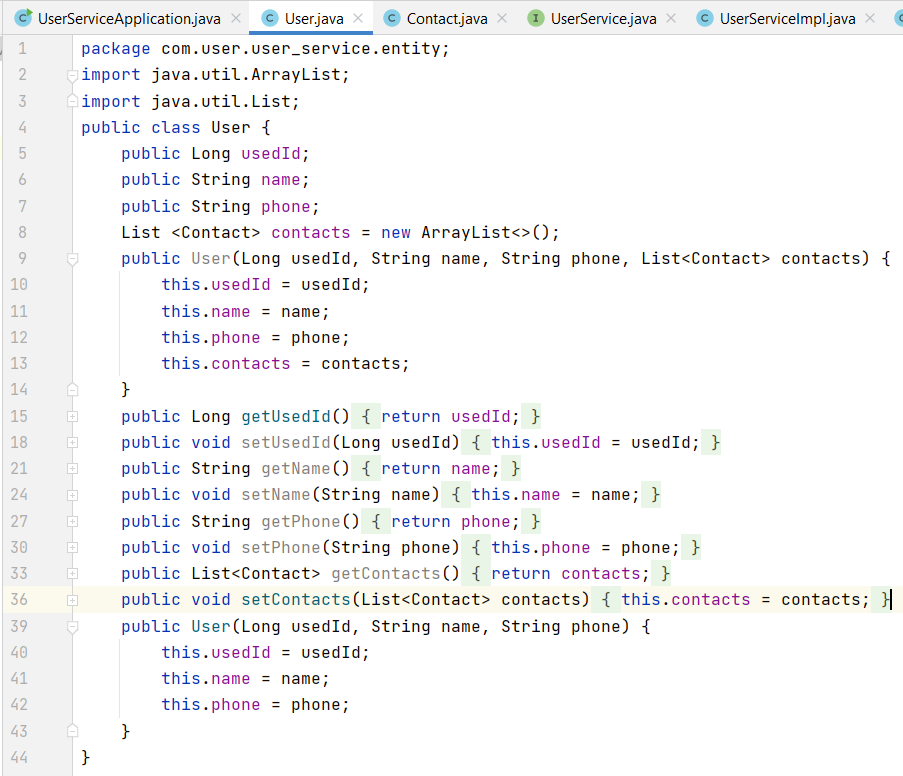
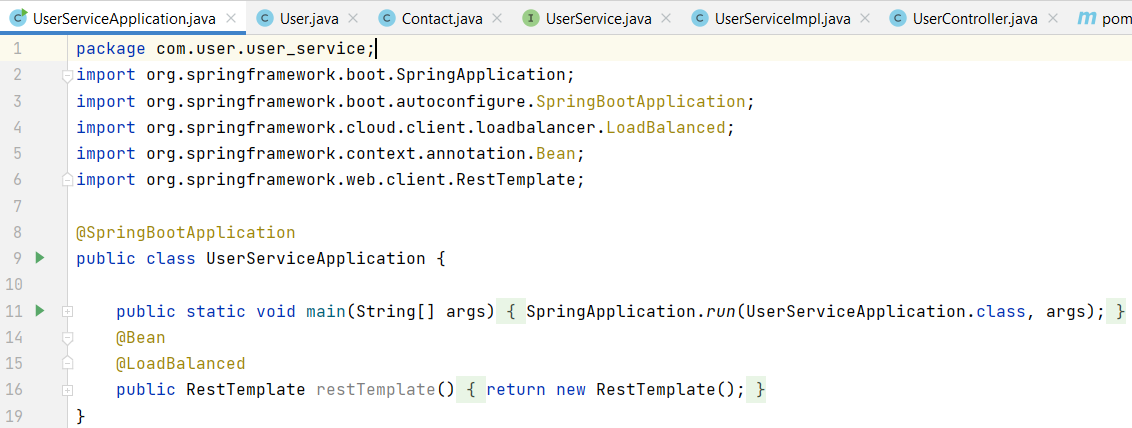
**Contact Service**

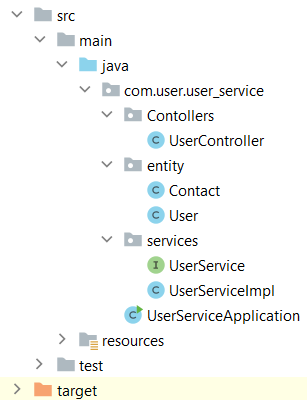
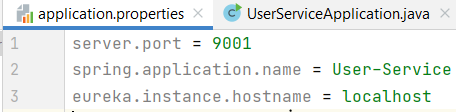
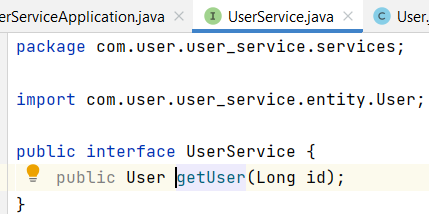
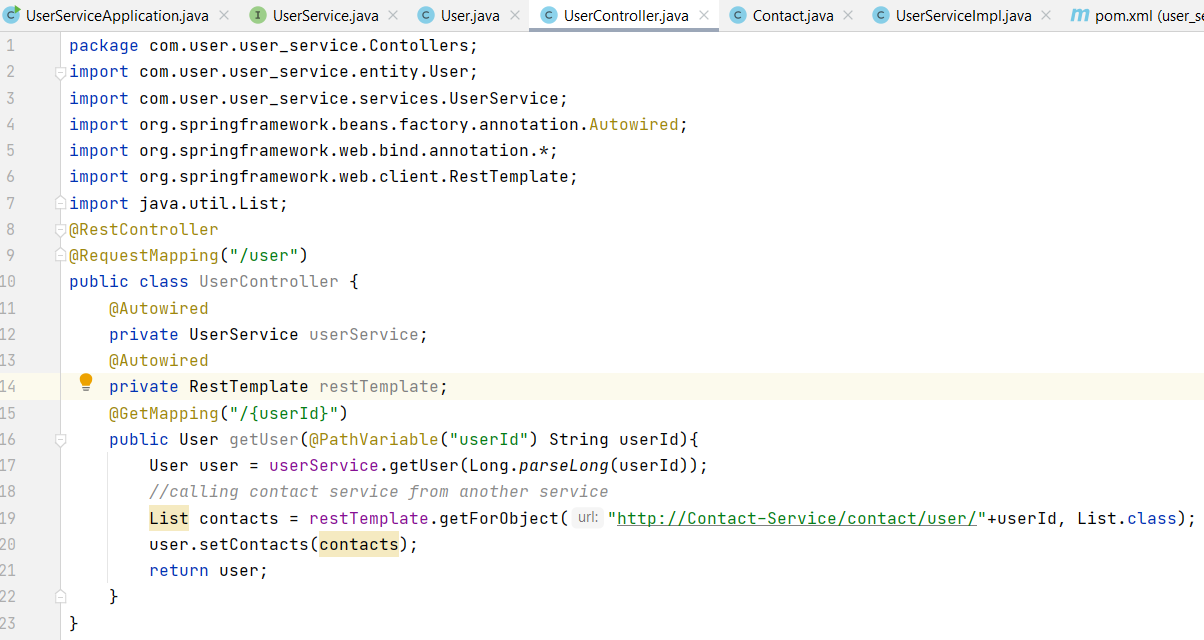






**User Service**

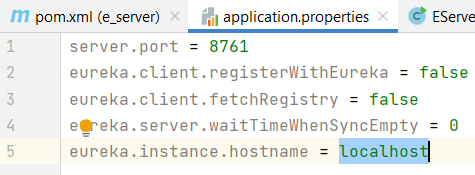
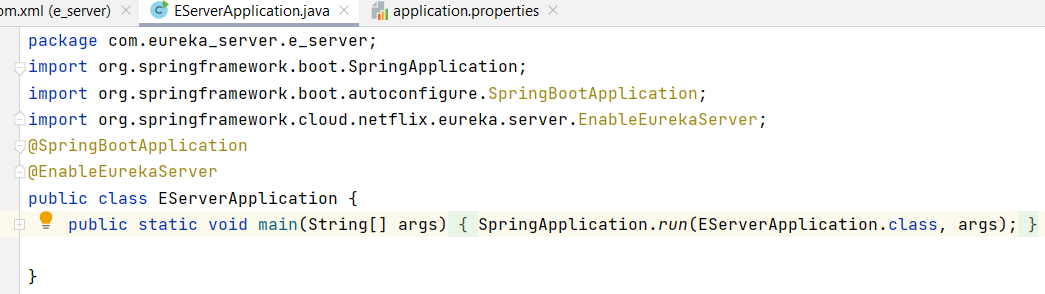






**Eureka Server**





**API Gateway**

