Backend Program: Course 3: Structure

Program — Build job-ready skills

- Enterprise Application Development by Using Spring Framework

Courses — Build competencies

- CRS1 : Spring Framework Foundation
- CRS 2 : Creating Restful Services Using SpringBoot
- CRS 3 : Creating and Managing Microservices
- CRS 4 : Establishing Communication Among Microservices in Synchronous and Asynchronous Way

Learning Sprints — Perform specific tasks

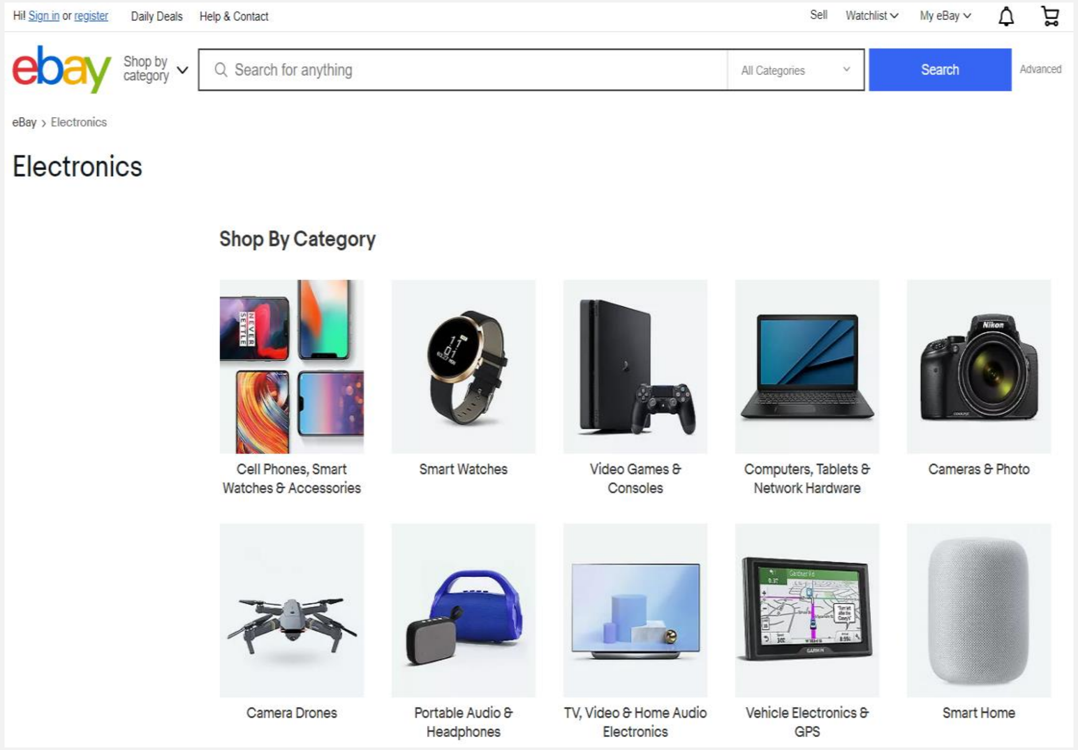
- Implement Client-Side Load Balancing by Using Netflix Ribbon
- Establish Synchronous Communication Among Microservices by Using Feign Client
- Configure a RabbitMQ Server for an Asynchronous Communication
- Establish Pub/Sub Communication Among Microservices Using RabbitMQ
- Consolidation of Microservices

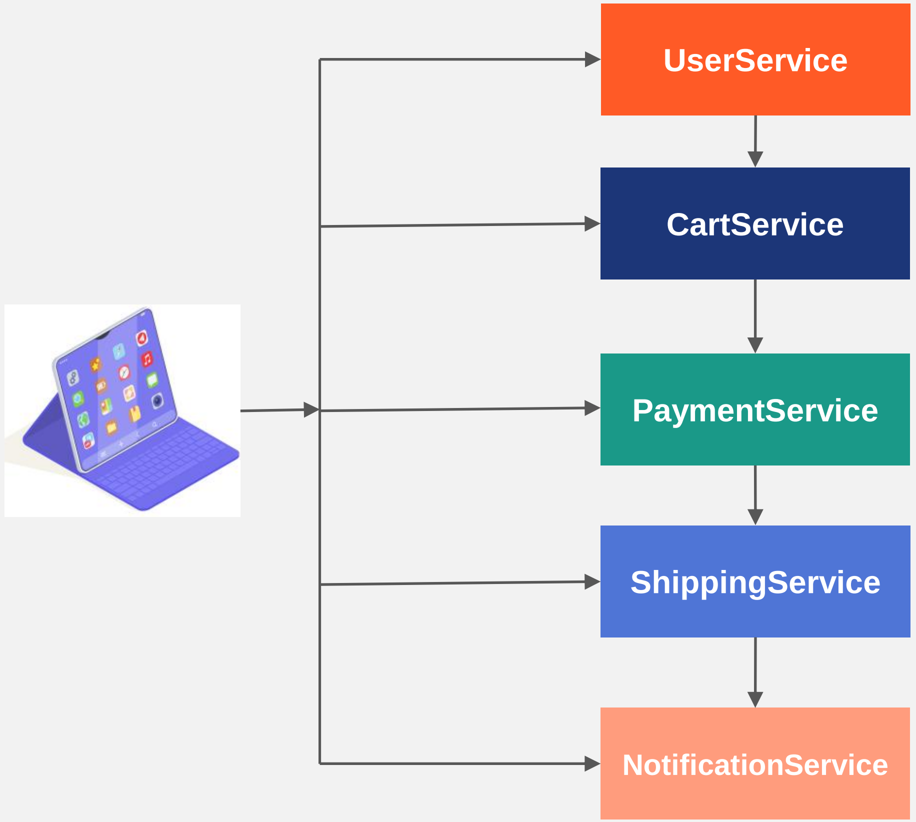
eBay

A large e-commerce application like eBay caters to a large numbers of customers around the globe.

Assume that in order to build the application there are multiple microservices involved.

- Can you share the process of buying a product on eBay ?





Application Workflow

- To make an online purchase, a user is required to register for an application and then add products to the cart.
- After users select the payment option, they are redirected to the payment gateway.
- After the payment is made the order is confirmed.
- An application has multiple services that cater to different requests from multiple clients.

Think and Tell

- Do you think all these operations occur in a sequence?
- Is it important for the services to interact with each other for the effective working of an application?
- How do they communicate?
- After a product is shipped to the user, is it mandatory for the `ShippingService` to send an acknowledgement to the `NotificationService` to notify the user that the product is shipped?

Establish Synchronous Communication Among Microservices by Using Feign Client

Learning Objectives

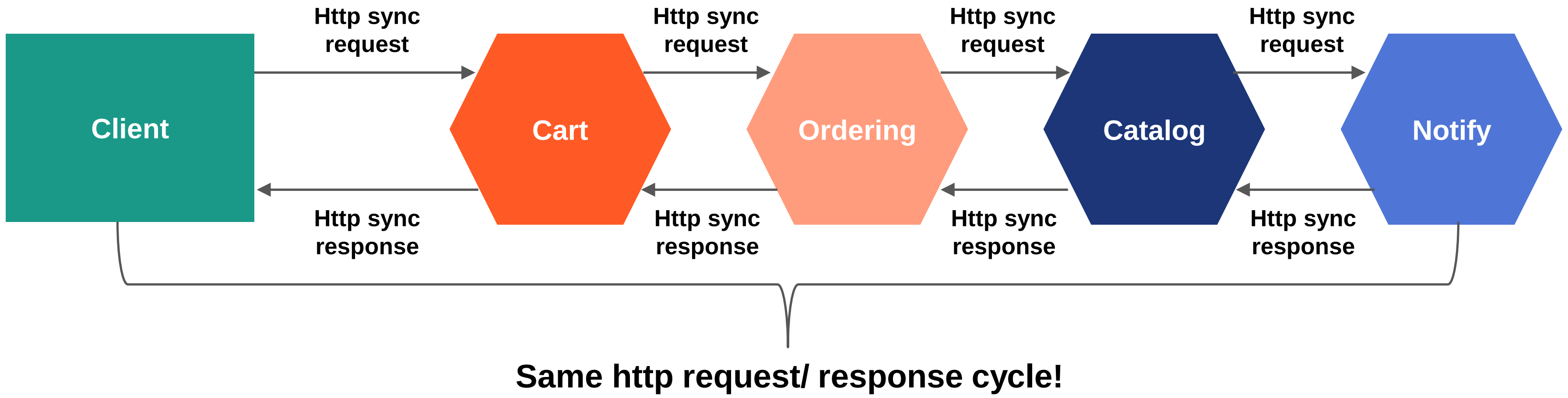
- Explain microservices communication
- Explore the types of communication
- Implement Feign client to establish synchronous communication between microservices

Microservices Communication

- The microservice architecture pattern is a distributed system running on different machines as a process or service.
- Each component of the system needs to interact with one another and coordinate their actions for the effective handling of client requests.
- Services often collaborate to handle the requests. Consequently, they must use an inter-process communication protocol.
- Deciding on how microservices communicate with one another is one of the most important and fundamental decisions to make when implementing a system based on the microservice architecture.
- There are two types of microservices communication:
  - Synchronous
  - Asynchronous

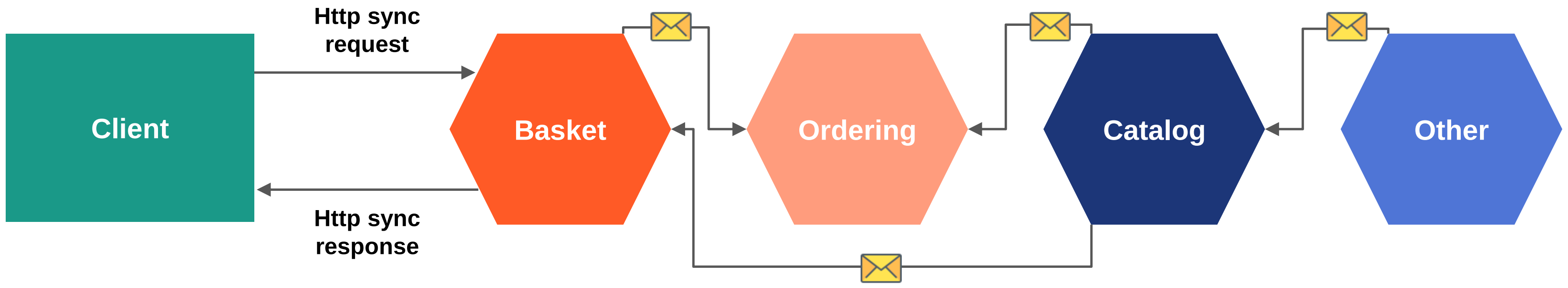
Synchronous Communication

- In synchronous communication, one microservice will communicate with another microservice through a rest endpoint over HTTP protocol.
- In this approach, the calling service will wait until the caller service responds.
- In synchronous communication a "chain" of requests is created between microservices while serving the client request.



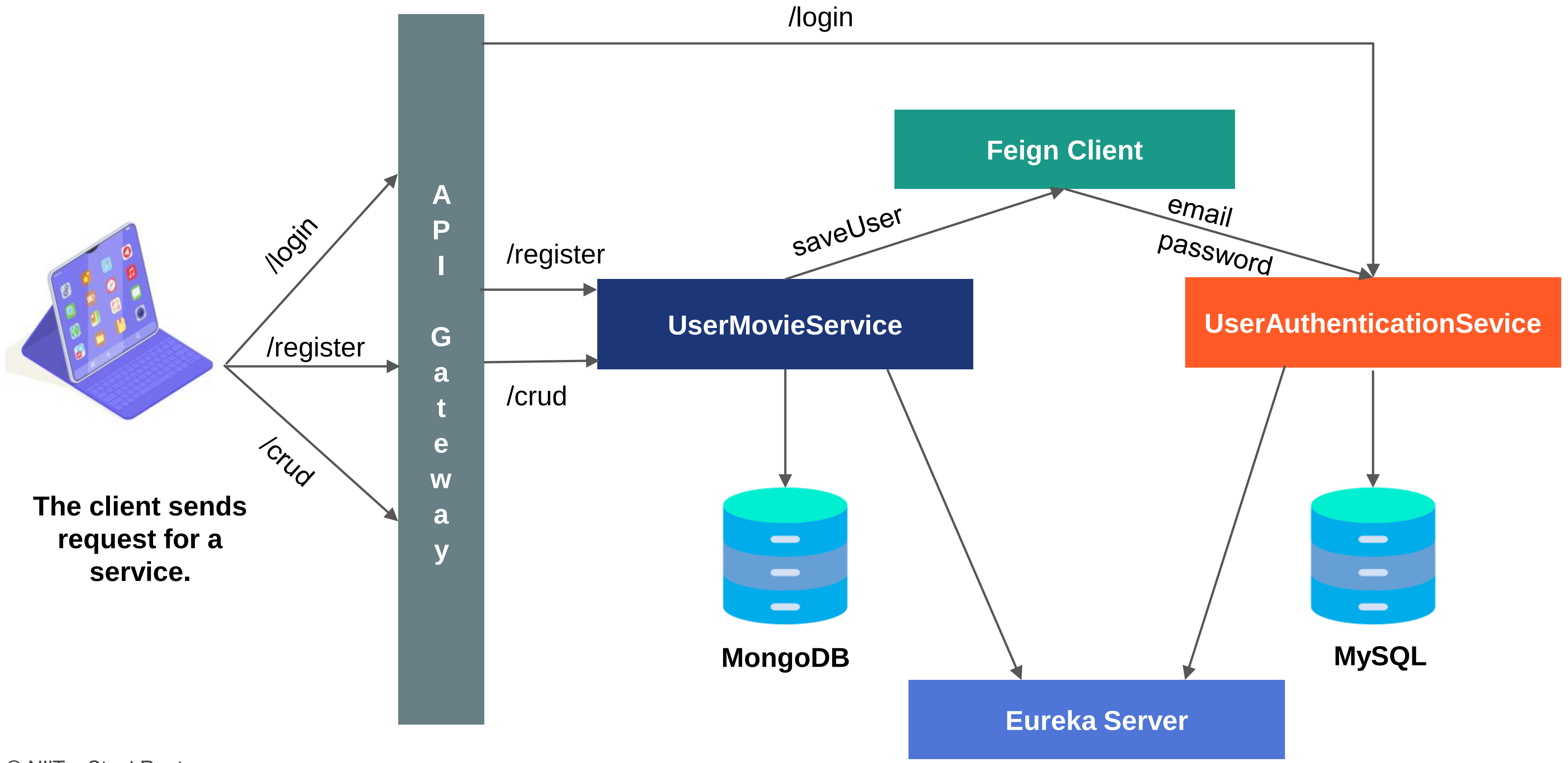
Asynchronous Communication

- In asynchronous communication, microservices use asynchronous messages or http polling to communicate with other microservices.
- The calling service will not wait for a response from the caller service.
- Asynchronous communication in microservices can be accomplished through message brokers like Apache Kafka, RabbitMQ etc.



Synchronous Communication Using Feign Client

Communication Between Microservices



The client sends request for a service.

Steps to Implement the Feign Client

- **Step 1:** Add dependency to the pom.xml of `UserMovieService`.

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

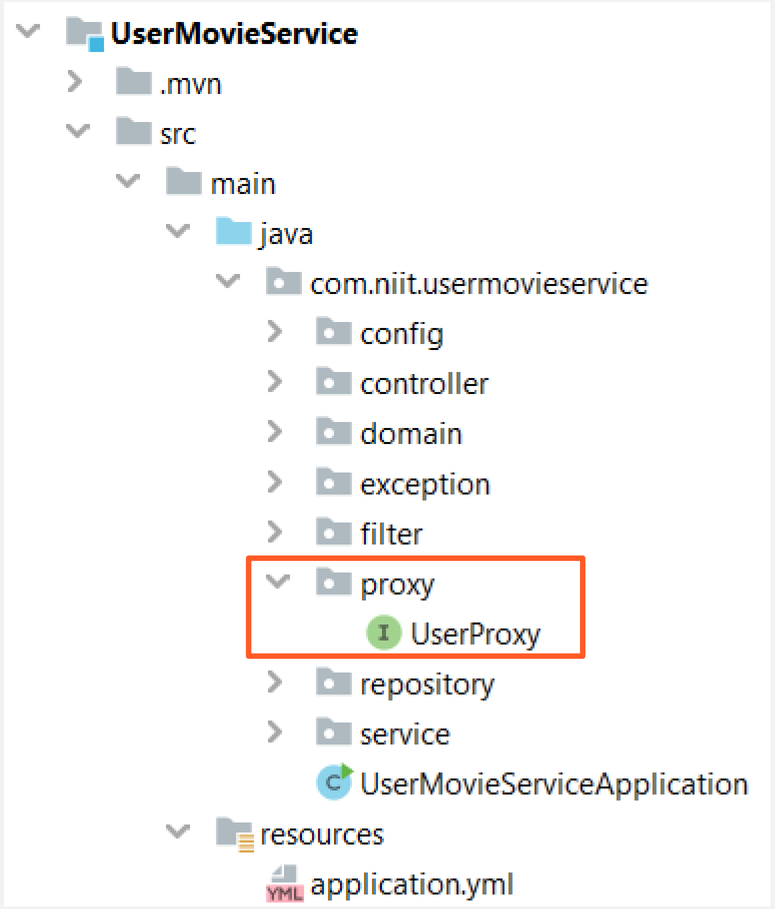
- **Step 2:** Enable Feign usage in the main application.

```
@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class UserMovieServiceApplication {

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

}
```

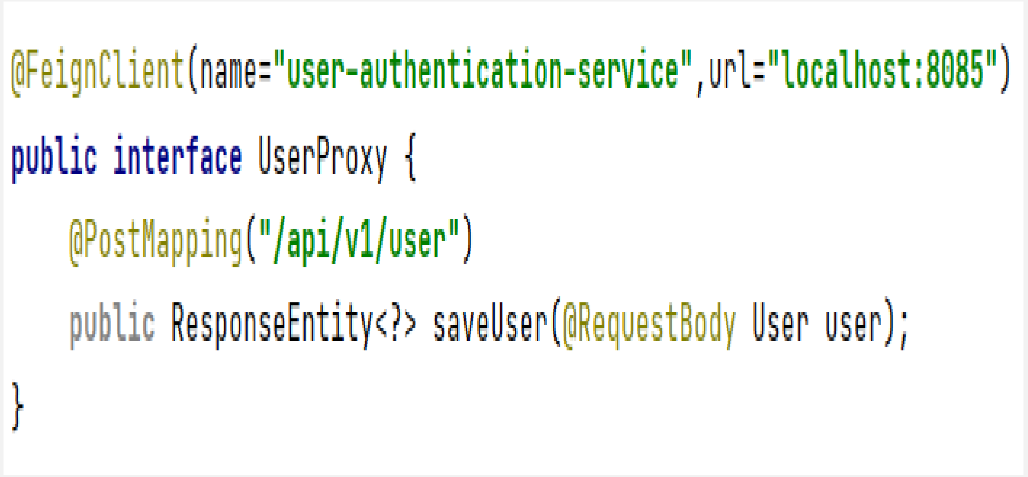
Proxy

Step 3:

Create a `UserProxy` interface in the `UserMovieService` that will be used to communicate with the `UserAuthenticationService`



```
UserMovieService
  .mvn
  src
    main
      java
        com.niit.usermovieservice
          config
          controller
          domain
          exception
          filter
          proxy
            UserProxy
          repository
          service
          UserMovieServiceApplication
      resources
        application.yml
```



```
@FeignClient(name="user-authentication-service",url="localhost:8085")
public interface UserProxy {
    @PostMapping("/api/v1/user")
    public ResponseEntity<?> saveUser(@RequestBody User user);
}
```

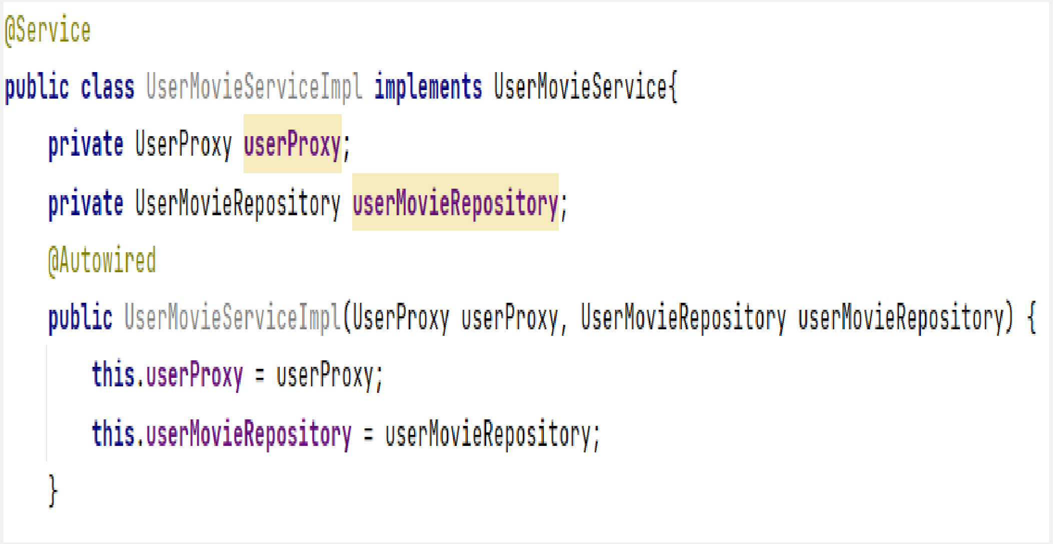
Proxy

- The Proxy interface is used to make outbound API calls to the other service, in this case, it is the `UserAuthenticationService`.
- Annotate the `UserProxy` with `@FeignClient` annotation.
- The annotation takes two parameters
  - `name` – The service for which the call is made.
  - `url` – This is the path to the service.

Service Layer

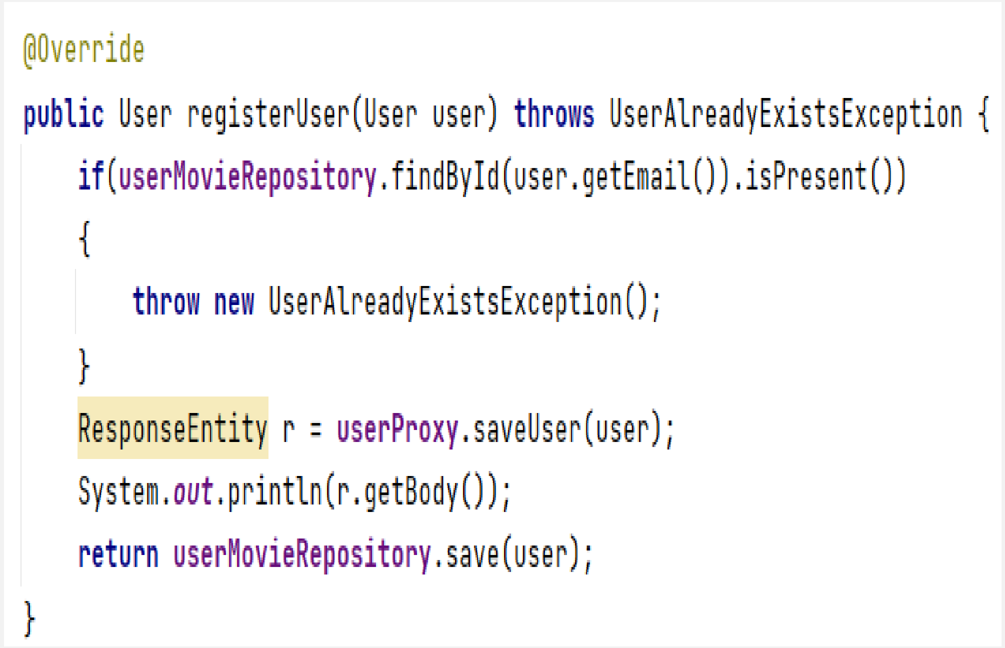
Step 4:

Autowire the proxy in the service layer.



```
@Service
public class UserMovieServiceImpl implements UserMovieService{
    private UserProxy userProxy;
    private UserMovieRepository userMovieRepository;
    @Autowired
    public UserMovieServiceImpl(UserProxy userProxy, UserMovieRepository userMovieRepository) {
        this.userProxy = userProxy;
        this.userMovieRepository = userMovieRepository;
    }
```

Service Layer



```
@Override
public User registerUser(User user) throws UserAlreadyExistsException {
    if(userMovieRepository.findById(user.getEmail()).isPresent())
    {
        throw new UserAlreadyExistsException();
    }
    ResponseEntity r = userProxy.saveUser(user);
    System.out.println(r.getBody());
    return userMovieRepository.save(user);
}
```

Step 5:

- The user data will be saved in the `UserAuthenticationService` when a new user registers in the `UserMovieService`.
- Use the proxy to send the data to the `UserAuthenticationService`.

Step 6:

- Run the application and test in Postman.

Streaming Application

Consider a streaming application that enables users to watch movies on any smart device. The application provides multiple features to all its registered users. A user needs to register with the application in order to access some of its features. Let us create multiple microservices for the streaming application.

1. A user must first register with the application.
2. Use credentials such as Id and password to login.
3. Access the features provided by the streaming application, like adding favourites, compiling a watch later list, etc.

Now, let us create a parent project called **MovieApplication.** This will contain the **UserAuthenticationService** and the **UserMovieService** as microservices. Implement Feign Client to send the user data to the **UserAuthenticationService,** once a new user registers. Dockerize the application.

DEMO

Quick Check

In Feign Client implementation, we declare and annotate a proxy interface while the actual implementation is provided at ____________________.

1. compile time
2. runtime

Quick Check: Solution

In Feign Client implementation, we declare and annotate a proxy interface while the actual implementation is provided at ___________________.

1. compile time

2. runtime

Key Takeaways

- Microservices communication
- Synchronous communication
- Asynchronous communication
- Implement synchronous communication using a Feign client

Thank you!