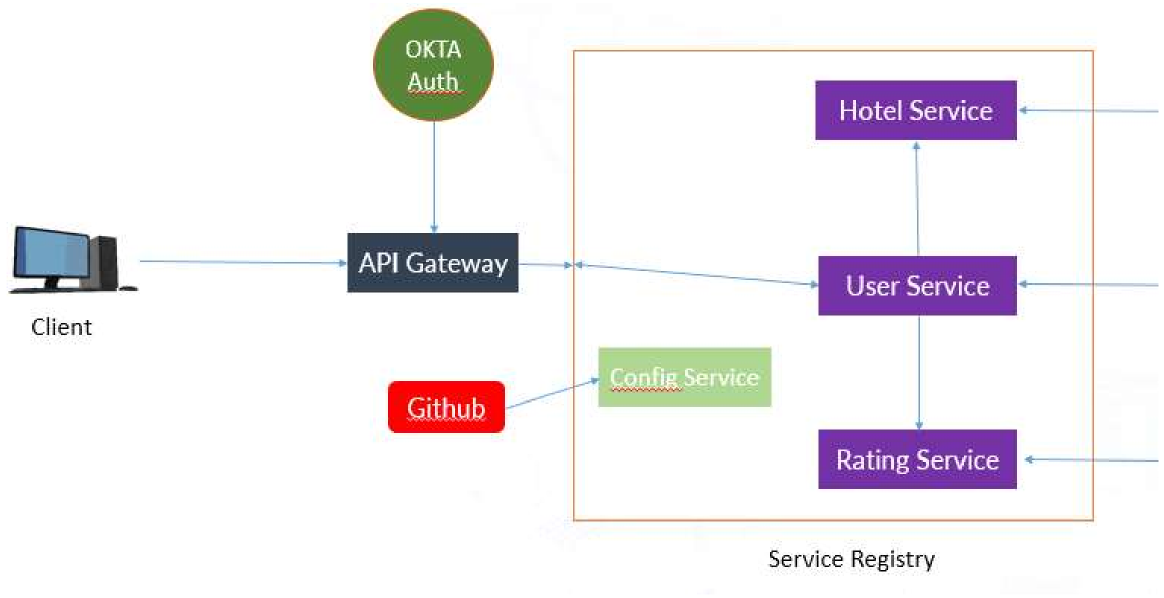
MICROSERVICE USING SPRING BOOT



Lets start building microservices

USER SERVICE MODULE

Entities:

User:

```
 package com.user.service.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Data
@AllArgsConstructor
```

```
@NoArgsConstructor
@Entity
@Table(name = "user_micro")
public class User {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long userId;
   private String userName;
   private String userEmail;
   private String about;

   @Transient
   private List<Rating> ratings=new ArrayList<>();
}
```

Rating:

```
package com.user.service.entities;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
public class Rating {

   private long ratingId;
   private long userId;
   private long hotelId;
   private int rating;
   private String feedback;
   private Hotel hotel;
}
```

Hotel:

```
package com.user.service.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Hotel {
   private long hotelId;
   private String hotelName;
   private String location;
   private String about;
}
```

Repository:

```
package com.user.service.repository;

import com.user.service.entities.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User,Long> {
}
```

Service:

```
package com.user.service.service;

import com.user.service.entities.User;

import java.util.List;


public interface UserService {
   User saveUser(User user);
   User getSingleUser(long userId);

   List<User> getAllUser();
}
```

ServiceImpl:

```
package com.user.service.service.impl;

import com.user.service.entities.Hotel;
import com.user.service.entities.Rating;
import com.user.service.entities.User;
import com.user.service.service.UserService;
import com.user.service.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class UserServiceImpl implements UserService {

  @Autowired
  private RestTemplate restTemplate;

  @Autowired
  private UserRepository userRepository;
  @Override
  public User saveUser(User user) {
    return userRepository.save(user);
  }

  @Override
  public User getSingleUser(long userId) {
    // get user from database by userid
    Optional<User> optional = userRepository.findById(userId);
    User user = optional.get();

    //get rating from rating service with the help of userId
    Rating[] forObject =
restTemplate.getForObject("http://localhost:8083/api/rating/user/" + user.getUserId(),
Rating[].class);
        List<Rating> ratings = Arrays.stream(forObject).collect(Collectors.toList());
        user.setRatings(ratings);

        // get the hotel with the help of ratingId
        ratings.stream().map(rating -> {
            Hotel hotel = restTemplate.getForObject("http://localhost:8082/api/hotel/"
+rating.getHotelId(), Hotel.class);
             rating.setHotel(hotel);
            return rating;
        }).collect(Collectors.toList());

        return user;
    }

    @Override
    public List<User> getAllUser() {

        List<User> all = userRepository.findAll();
       for(User user:all){
           ArrayList ratingOfUser =
restTemplate.getForObject("http://localhost:8083/api/rating/user/" + user.getUserId(),
ArrayList.class);
            user.setRatings(ratingOfUser);
       }
       return all;
    }
}
```

Controller:

```
package com.user.service.controller;

import com.user.service.entities.User;
import com.user.service.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
```

```java
@RestController
@RequestMapping("/api/user")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping("/save")
    public ResponseEntity<User> createUser(@RequestBody User user){
        User saveUser = userService.saveUser(user);
        return new ResponseEntity<>(saveUser, HttpStatus.CREATED);
    }


    @GetMapping("/{userId}")
    public ResponseEntity<User> getSingleUser(@PathVariable("userId") long userId){
        User singleUser = userService.getSingleUser(userId);
        return new ResponseEntity<>(singleUser,HttpStatus.OK);
    }

    @GetMapping("/getAll")
    public ResponseEntity<List<User>> getAllUser(){
        List<User> allUser = userService.getAllUser();
        return new ResponseEntity<>(allUser,HttpStatus.OK);
    }
}
```

Main Class:

```java
package com.user.service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
public class UserApplication {

   @Bean
   public RestTemplate restTemplate(){
      return new RestTemplate();
   }
```

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

}
```

application.yml:

```
server:
  port: 8081
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect
```

Pom.xml for dependency:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
  <optional>true</optional>
</dependency>


<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
```

```
</dependency>

<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <scope>runtime</scope>
</dependency>


<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

HOTEL MODULE

Entities:

```
package com.hotel.service.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Table(name = "hotels")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Hotel {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long hotelId;
   private String hotelName;
   private String location;
   private String about;
}
```

Repository:

```
package com.hotel.service.repository;

import com.hotel.service.entities.Hotel;
import org.springframework.data.jpa.repository.JpaRepository;

public interface HotelRepository extends JpaRepository<Hotel,Long> {
}
```

Service:

```
package com.hotel.service.service;

import com.hotel.service.entities.Hotel;

import java.util.List;

public interface HotelService {
   Hotel createHotel(Hotel hotel);
   Hotel getHotelById(long hotelId);
   List<Hotel> getAllHotel();
}
```

HotelServiceImpl:

```
package com.hotel.service.service.impl;

import com.hotel.service.entities.Hotel;
import com.hotel.service.repository.HotelRepository;
import com.hotel.service.service.HotelService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class HotelServiceImpl implements HotelService {
    @Autowired
    private HotelRepository hotelRepository;
    @Override
    public Hotel createHotel(Hotel hotel) {
        return hotelRepository.save(hotel);
    }

    @Override
    public Hotel getHotelById(long hotelId) {
        Optional<Hotel> optional = hotelRepository.findById(hotelId);
        Hotel hotel = optional.get();
        return hotel;
    }

    @Override
    public List<Hotel> getAllHotel() {
        return hotelRepository.findAll();
    }
}
```

Controller:

```
package com.hotel.service.controller;

import com.hotel.service.entities.Hotel;
import com.hotel.service.service.HotelService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/hotel")
public class HotelController {

    @Autowired
    private HotelService hotelService;

    @PostMapping("/save")
    public ResponseEntity<Hotel> createHotel(@RequestBody Hotel hotel){
        Hotel savedHotel = hotelService.createHotel(hotel);
        return new ResponseEntity<>(savedHotel, HttpStatus.CREATED);
    }

    @GetMapping("/{hotelId}")
    public ResponseEntity<Hotel> getSingleHotel(@PathVariable("hotelId")long hotelId){
        Hotel hotelById = hotelService.getHotelById(hotelId);
        return new ResponseEntity<>(hotelById,HttpStatus.OK);
    }

    @GetMapping("/getAll")
    public ResponseEntity<List<Hotel>> getAllHotel(){
        List<Hotel> allHotel = hotelService.getAllHotel();
        return new ResponseEntity<>(allHotel,HttpStatus.OK);
    }
}
```

Main Class:

```
package com.hotel.service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class HotelApplication {

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

}
```

application.yml:

```
server:
  port: 8082
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect
```

Pom.xml File For Dependency:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
  </dependency>
```

RATING MODULE

Entities:

```
package com.rating.service.entities;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Table(name = "rating")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Rating {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long ratingId;
   private long userId;
   private long hotelId;
   private  int rating;
   private String feedback;

}
```

Repository:

```
package com.rating.service.repository;

import com.rating.service.entities.Rating;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface RatingRepository extends JpaRepository<Rating,Long> {
   List<Rating> findByUserId(long userId);
   List<Rating> findByHotelId(long hotelId);
}
```

Service:

```
package com.rating.service.service;

import com.rating.service.entities.Rating;

import java.util.List;

public interface RatingService {

    // create Rating
    Rating createRating(Rating rating);

    // get All rating By RatingId
    List<Rating> getAllRating();

    // get AllRating By UserId
    List<Rating> getRatingByUserId(long userId);

    // get AllRating By hotelId
    List<Rating> getRatingByHotelId(long hotelId);
}
```

ServiceImpl:

```
package com.rating.service.service.impl;

import com.rating.service.entities.Rating;
import com.rating.service.repository.RatingRepository;
import com.rating.service.service.RatingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class RatingServiceImpl implements RatingService {

    @Autowired
    private RatingRepository repository;
    @Override
    public Rating createRating(Rating rating) {
        return repository.save(rating);
    }

    @Override
    public List<Rating> getAllRating() {
        return repository.findAll();
    }

    @Override
    public List<Rating> getRatingByUserId(long userId) {
        return repository.findByUserId(userId);
    }

    @Override
    public List<Rating> getRatingByHotelId(long hotelId) {
        return  repository.findByHotelId(hotelId);
    }
}
```

Controller:

```
package com.rating.service.controller;

import com.rating.service.entities.Rating;
import com.rating.service.service.RatingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/rating")
public class RatingController {

    @Autowired
    private RatingService ratingService;

    @PostMapping("/save")
    public ResponseEntity<Rating> saveRating(@RequestBody Rating rating){
        Rating saveRating = ratingService.createRating(rating);
        return new ResponseEntity<>(saveRating, HttpStatus.CREATED);
    }

    @GetMapping("/getAll")
    public ResponseEntity<List<Rating>> getAllRating(){
        List<Rating> allRating = ratingService.getAllRating();
        return new ResponseEntity<>(allRating,HttpStatus.OK);
    }

    @GetMapping("/user/{userId}")
    public ResponseEntity<List<Rating>> getRatingByUserId(@PathVariable("userId")
long userId){
        List<Rating> ratingByUserId = ratingService.getRatingByUserId(userId);
        return new ResponseEntity<>(ratingByUserId,HttpStatus.OK);
    }

    @GetMapping("/hotel/{hotelId}")
    public ResponseEntity<List<Rating>> getRatingByHotelId(@PathVariable("hotelId")
long hotelId){
        List<Rating> ratingByHotelId = ratingService.getRatingByHotelId(hotelId);
        return new ResponseEntity<>(ratingByHotelId,HttpStatus.OK);
    }
}
```

Main Class:

```
package com.rating.service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RatingApplication {

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

application.yml:

```
server:
  port: 8083
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect
```

Pom.xml File For Dependency:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-devtools</artifactId>
   <scope>runtime</scope>
   <optional>true</optional>
</dependency>


<dependency>
   <groupId>com.mysql</groupId>
   <artifactId>mysql-connector-j</artifactId>
   <scope>runtime</scope>
```

```
</dependency>

<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

<u>USAGE OF FEIGN CLIENT</u>

- Add dependency:

```
<properties>
  <java.version>1.8</java.version>
  <spring-cloud.version>2021.0.7</spring-cloud.version>
</properties>

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

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- Annotated the main class with @EnableFeignClients |update main class

```
package com.user.service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableFeignClients
public class UserApplication {

@Bean
public RestTemplate restTemplate(){
return new RestTemplate();
}

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

- Create Interface and marked with @FeignClient

```
package com.user.service.externalService;
```

```
import com.user.service.entities.Hotel;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name = "unique-name", url = "http://localhost:8082")
public interface HotelService {

  @GetMapping("/api/hotel/{hotelId}")
  Hotel getSingleHotel(@PathVariable("hotelId") long hotelId);

}
```

- Update ServiceImpl:

```
package com.user.service.service.impl;

import com.user.service.entities.Hotel;
import com.user.service.entities.Rating;
import com.user.service.entities.User;
import com.user.service.externalService.HotelService;
import com.user.service.service.UserService;
import com.user.service.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private RestTemplate restTemplate;

@Autowired
private HotelService hotelService;

    @Autowired
    private UserRepository userRepository;
    @Override
    public User saveUser(User user) {
        return userRepository.save(user);
    }

    @Override
    public User getSingleUser(long userId) {
        // get user from database by userid
        Optional<User> optional = userRepository.findById(userId);
        User user = optional.get();
        //get rating from rating service with the help of userId
        Rating[] forObject =
restTemplate.getForObject("http://localhost:8083/api/rating/user/" +
user.getUserId(), Rating[].class);
        List<Rating> ratings = Arrays.stream(forObject).collect(Collectors.toList());
        user.setRatings(ratings);


        // get the hotel with the help of ratingId
        List<Rating> ratingList = ratings.stream().map(rating -> {
           // Hotel hotel =
restTemplate.getForObject("http://localhost:8082/api/hotel/" +
rating.getHotelId(), Hotel.class);
            Hotel singleHotel = hotelService.getSingleHotel(rating.getHotelId());
            rating.setHotel(singleHotel);
            return rating;
        }).collect(Collectors.toList());


        return user;
    }


    @Override
    public List<User> getAllUser() {


        List<User> all = userRepository.findAll();
       for(User user:all){
           ArrayList ratingOfUser =
restTemplate.getForObject("http://localhost:8083/api/rating/user/" +
user.getUserId(), ArrayList.class);
            user.setRatings(ratingOfUser);
        }
        return all;
    }
}
```

SERVICE REGISTRY/DISCOVERY SERVER

- Create a seprate Spring boot Project with dependencies like:

  Eureka Server:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

Cloud BootStrap:

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

Note:- whenever we have to use cloud dependeny then this dependency is mandatory

With version

```
<properties>
  <java.version>1.8</java.version>
  <spring-cloud.version>2021.0.7</spring-cloud.version>
```

```
</properties>



<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- Annotate main class with @EnableEurekaServer to configuration

```
package com.service.registry;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {

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

- Application.yml file for server Registry:

```yaml
server:
  port: 8761

eureka:
  instance:
    hostname: localhost
  client:
    register-with-eureka: false
    fetch-registry: false
```

<u>IMPLEMENTING SERVICE DISCOVERY CLIENT FOR EACH MICRO SERVICE</u>

<u>USER SERVICE</u>

- **Add these two dependency in pom.xml file :**

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

 <dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
 </dependency>
```

Note: for these dependency add properties in pom.xml and dependency management also.

```
 <properties>
   <java.version>1.8</java.version>
   <spring-cloud.version>2021.0.7</spring-cloud.version>
 </properties>
```

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- **Mark main class with @EnableEurekaClient and update main class:**

```
package com.user.service;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;
```

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

	@Bean
public RestTemplate restTemplate(){
return new RestTemplate();
}

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

}
```

- **Update application.yml file :**

```
server:
  port: 8081
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
```

```
 jpa:
  hibernate:
   ddl-auto: update
   show-sql: true
   properties:
    hibernate:
     dialect: org.hibernate.dialect.MySQL5Dialect

# for changing the name of application on server
 application:
  name: USER-SERVICE



# configuration for service discovery client
eureka:
 instance:
  prefer-ip-address: true
 client:
  fetch-registry: true
  register-with-eureka: true
  service-url:
   defaultZone: http://localhost:8761/eureka
```

HOTEL SERVICE

- **Add these two dependency in pom.xml file :**

```
 <dependency>
```

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

    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    </dependency>
```

Note: for these dependency add properties in pom.xml and dependency management also.

```xml
  <properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>2021.0.7</spring-cloud.version>
  </properties>
```

```xml
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- **Mark main class with @EnableEurekaClient and update main class:**

```
package com.hotel.service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class HotelApplication {

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

}
```

- **Update application.yml file :**

```
server:
  port: 8082
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect
```

```yaml
# for changing the name of application on server
  application:
    name: HOTEL-SERVICE

# configuration for service discovery client
eureka:
  instance:
    prefer-ip-address: true
  client:
    fetch-registry: true
    register-with-eureka: true
    service-url:
      defaultZone: http://localhost:8761/eureka
```

RATING SERVICE

- **Add these two dependency in pom.xml file :**

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

  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>
```

Note: for these dependency add properties in pom.xml and dependency management also.

```
 <properties>
   <java.version>1.8</java.version>
   <spring-cloud.version>2021.0.7</spring-cloud.version>
 </properties>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- **Mark main class with @EnableEurekaClient and update main class:**

```
package com.rating.service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class RatingApplication {
```

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


}
```

- **Update application.yml file :**

```
server:
  port: 8083
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect

  # for changing the name of application on server
  application:
    name: RATING-SERVICE
```

```
# configuration for service discovery client
eureka:
  instance:
    prefer-ip-address: true
  client:
    fetch-registry: true
    register-with-eureka: true
    service-url:
      defaultZone: http://localhost:8761/eureka
```

REPLACE HOST AND PORT NO WITH THEIR SERVICE NAME

USER SERVICE

- **Update user serviceImpl for replacing host and port no.**

```
package com.user.service.service.impl;
import com.user.service.entities.Hotel;
import com.user.service.entities.Rating;
import com.user.service.entities.User;
import com.user.service.externalService.HotelService;
import com.user.service.service.UserService;
import com.user.service.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

@Service
public class UserServiceImpl implements UserService {

   @Autowired
   private RestTemplate restTemplate;

@Autowired
private HotelService hotelService;

   @Autowired
   private UserRepository userRepository;
   @Override
   public User saveUser(User user) {
      return userRepository.save(user);
   }

   @Override
   public User getSingleUser(long userId) {
      // get user from database by userid
      Optional<User> optional = userRepository.findById(userId);
```

```
        User user = optional.get();


        //get rating from rating service with the help of userId

        Rating[] forObject = restTemplate.getForObject("http://RATING-
SERVICE/api/rating/user/" + user.getUserId(), Rating[].class);

        List<Rating> ratings = Arrays.stream(forObject).collect(Collectors.toList());

        user.setRatings(ratings);


        // get the hotel with the help of ratingId

        List<Rating> ratingList = ratings.stream().map(rating -> {

            Hotel singleHotel = restTemplate.getForObject("http://HOTEL-
SERVICE/api/hotel/" + rating.getHotelId(), Hotel.class);

            //Hotel singleHotel = hotelService.getSingleHotel(rating.getHotelId());

            rating.setHotel(singleHotel);

            return rating;

        }).collect(Collectors.toList());


        return user;

    }


    @Override

    public List<User> getAllUser() {


        List<User> all = userRepository.findAll();

       for(User user:all){

           ArrayList ratingOfUser = restTemplate.getForObject("http://RATING-
SERVICE/api/rating/user/" + user.getUserId(), ArrayList.class);

            user.setRatings(ratingOfUser);
        }
        return all;
    }
}
```

- Mark with @LoadBalanced annotation on RestTemplate for load balancing

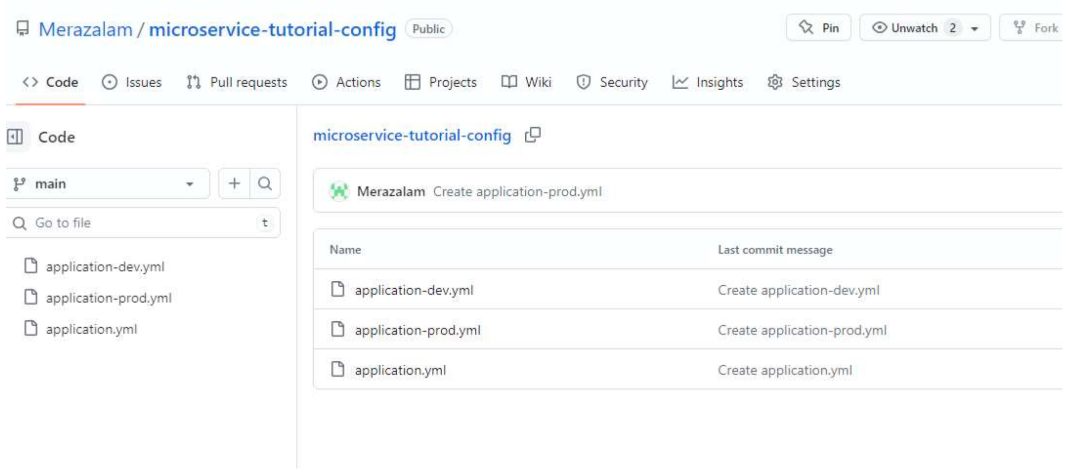
```
package com.user.service;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@SpringBootApplication
@EnableFeignClients
@EnableEurekaClient
public class UserApplication {

@Bean
@LoadBalanced
public RestTemplate restTemplate(){
return new RestTemplate();
}
public static void main(String[] args) {
SpringApplication.run(UserApplication.class, args);
                                                    }

}
```

CONFIG SERVER FOR EXTRANALIZE CONFIGURATION ON SERVER

- Create a spring boot project with **dependencies** like,

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>
```

- Go on git hub and create Repository:



Merazalam Create application-prod.yml

Name	Last commit message
application-dev.yml	Create application-dev.yml
application-prod.yml	Create application-prod.yml
application.yml	Create application.yml

- **Marked with annotation like @EnableConfigServer in main class:**

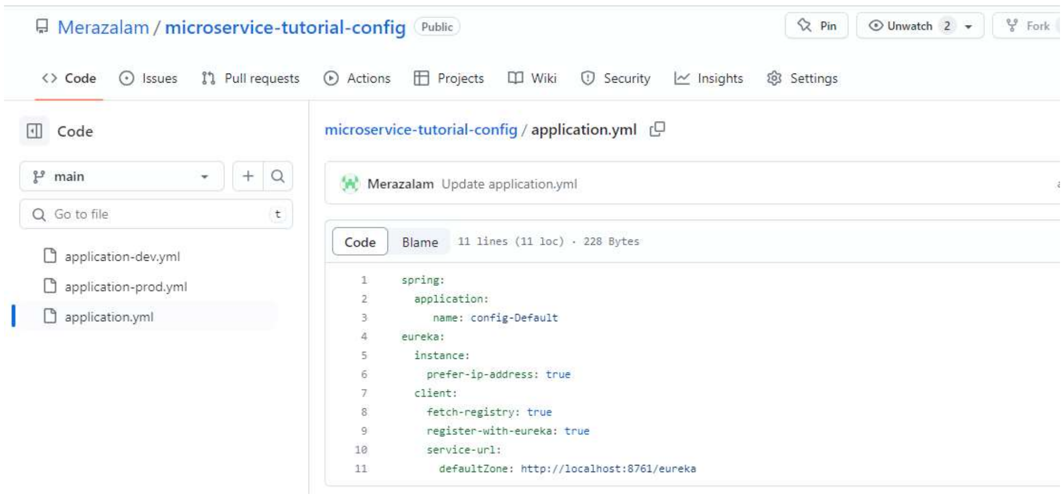
```
package com.config;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {

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

- **Give the configuration in application.yml file :**

```
server:
  port: 8084
```

```
spring:
  application:
    name: CONFIG-SERVER

  cloud:
    config:
      server:
        git:
          uri: https://github.com/Merazalam/microservice-tutorial-config
          clone-on-start: true
```

- **Go on git hub and give the common configuration for each micro service in each profiles**



```
spring:
  application:
    name: config-Default
eureka:
  instance:
    prefer-ip-address: true
  client:
    fetch-registry: true
    register-with-eureka: true
    service-url:
      defaultZone: http://localhost:8761/eureka
```

READING CONFIG FROM GITHUB

- **Add the dependency in pom.xml file of user:**

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

- **update application.yml file:**

```
   server:
port: 8081


spring:
 # for changing in different enviroment
 profiles:
   active: prod
 # reading configuration from git hub with the help of config server
 config:
   import: configserver:http://localhost:8084

 datasource:
   url: jdbc:mysql://localhost:3306/microsservices
   username: root
   password: test
   driver-class-name: com.mysql.cj.jdbc.Driver
   jpa:
   hibernate:
     ddl-auto: update
     show-sql: true
     properties:
       hibernate:
         dialect: org.hibernate.dialect.MySQL5Dialect

# for changing the name of application on server
#  application:
#    name: USER-SERVICE


# configuration for service discovery client
#eureka:
#  instance:
#    prefer-ip-address: true
#  client:
#    fetch-registry: true
#    register-with-eureka: true
#    service-url:
#      defaultZone: http://localhost:8761/eureka
```

Note: in hotel service and rating service do same step:

> add dependency of config client

> update application.yml file

API GETEWAY

> **add the dependency :**

```
 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-webflux</artifactId>
   </dependency>

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

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

   <dependency>
     <groupId>org.springframework.cloud</groupId>
     <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
   </dependency>

   <dependency>
     <groupId>org.projectlombok</groupId>
     <artifactId>lombok</artifactId>
     <optional>true</optional>
   </dependency>

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

> **marked main class with annotation like @EnableEurekaClient:**

```
package com.api;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class ApiGateWayApplication {

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

}
```

> **configure application.yml file:**

```
server:
  port: 8085

spring:
  application:
    name: API-GATEWAY
  config:
    import: configserver:http://localhost:8084


# configuration for service discovery client
#eureka:
#  instance:
#    prefer-ip-address: true
#  client:
#    fetch-registry: true
#    register-with-eureka: true
#    service-url:
#      defaultZone: http://localhost:8761/eureka



  cloud:
    gateway:
      routes:
        - id: USER-SERVICE
          uri: lb://USER-SERVICE
          predicates:
            - Path=/api/user/**

        - id: HOTEL-SERVICE
          uri: lb://HOTEL-SERVICE
          predicates:
            - Path=/api/hotel/**

        - id: RATING-SERVICE
          uri: lb://RATING-SERVICE
          predicates:
            - Path=/api/rating/**
```

API GATEWAY CONFIGURING MULTIPLE URL OF MICROSERVICE

> **create another controller** :

```
package com.hotel.service.controller;
```

```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Arrays;
import java.util.List;

@RestController
@RequestMapping("/api/staff")
public class StaffController {

    @GetMapping
    public ResponseEntity<List<String>> getStaff(){
        List<String> names = Arrays.asList("sahil", "punit", "sam" );
        return new ResponseEntity<>(names,HttpStatus.OK);
    }
}
```

> **update application.yml file of api gateway:**

```
server:
  port: 8085

spring:
  application:
    name: API-GATEWAY
  config:
    import: configserver:http://localhost:8084


# configuration for service discovery client
#eureka:
#  instance:
#    prefer-ip-address: true
#  client:
#    fetch-registry: true
#    register-with-eureka: true
#    service-url:
#      defaultZone: http://localhost:8761/eureka



  cloud:
    gateway:
```

```
routes:
  - id: USER-SERVICE
    uri: lb://USER-SERVICE
    predicates:
      - Path=/api/user/**

  - id: HOTEL-SERVICE
    uri: lb://HOTEL-SERVICE
    predicates:
      - Path=/api/hotel/**,/api/staff/**

  - id: RATING-SERVICE
    uri: lb://RATING-SERVICE
    predicates:
      - Path=/api/rating/**
```

HOW TO HANDLE IF MICROSERVICE IS FAULTY?

FAULT TOLERANCE

IMPELEMENTING CIRCUIT BREAKER USING RESILIENCE 4J LIBRARY

> add the dependency :

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>


<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-aop</artifactId>
</dependency>


<dependency>
        <groupId>io.github.resilience4j</groupId>
        <artifactId>resilience4j-spring-boot2</artifactId>
</dependency>
```

> **update the user Controller layer:**

```
package com.user.service.controller;

import com.user.service.entities.User;
import com.user.service.service.UserService;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/user")
public class UserController {

   Logger logger = LoggerFactory.getLogger(UserController.class);

   @Autowired
   private UserService userService;

   @PostMapping("/save")
   public ResponseEntity<User> createUser(@RequestBody User user){
      User saveUser = userService.saveUser(user);
      return new ResponseEntity<>(saveUser, HttpStatus.CREATED);
   }


   // applying circuit breaker annotation
   @GetMapping("/{userId}")
   @CircuitBreaker(name = "ratingHotelBreaker",fallbackMethod = "ratingHotelFallBack")
   public ResponseEntity<User> getSingleUser(@PathVariable("userId") long userId){
      User singleUser = userService.getSingleUser(userId);
      return new ResponseEntity<>(singleUser,HttpStatus.OK);
   }

   // creating fallback method for circuit breaker

   public ResponseEntity<User> ratingHotelFallBack(@PathVariable("userId") long
userId,Exception ex){
      logger.info("fall back is executed because service is down");
      User user = User.builder()
            .userEmail("dummy@gmail.com")
            .userName("dummy")
```

```
                .about("this user is created dummy")
                .userId(123)
                .build();
        return new ResponseEntity<>(user,HttpStatus.OK);
    }

    @GetMapping("/getAll")
    public ResponseEntity<List<User>> getAllUser(){
        List<User> allUser = userService.getAllUser();
        return new ResponseEntity<>(allUser,HttpStatus.OK);
    }
}
```

> **update the application.yml file:**

```
server:
  port: 8081


spring:

  # reading configuration from git hub with the help of config server
  config:
    import: configserver:http://localhost:8084

  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
    jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect

# for changing the name of application on server
  application:
    name: USER-SERVICE


# configuration for service discovery client
#eureka:
#  instance:
#    prefer-ip-address: true
#  client:
#    fetch-registry: true
#    register-with-eureka: true
#    service-url:
#      defaultZone: http://localhost:8761/eureka
# configuration for circuit breaker using resilience 4j
      # actuator
management:
  health:
    circuitbreaker:
      enabled: true
  endpoints:
    web:
      exposure:
        include: health

  endpoint:
    health:
      show-details: always


          # resilience 4j

resilience4j:
  circuitbreaker:
    instances:
      ratingHotelBreaker:
        registerHealthIndicator: true
        eventConsumerBufferSize: 10
        failureRateThreshold: 50
        minimumNumberOfCalls: 5
        automaticTransitionFromOpenToHalfOpenEnabled: true
        waitDurationInOpenState: 6s
        permittedNumberOfCallsInHalfOpenState: 3
        slidingWindowSize: 10
        slidingWindowType: COUNT_BASED
```

RETRY INPLEMENTATION IN USER MODULE

> **update the userController layer:**

```
package com.user.service.controller;

import com.user.service.entities.User;
import com.user.service.service.UserService;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import io.github.resilience4j.retry.annotation.Retry;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/user")
```

```
public class UserController {

    Logger logger = LoggerFactory.getLogger(UserController.class);

    @Autowired
    private UserService userService;

    @PostMapping("/save")
    public ResponseEntity<User> createUser(@RequestBody User user){
        User saveUser = userService.saveUser(user);
        return new ResponseEntity<>(saveUser, HttpStatus.CREATED);
    }


    // applying retry annotation
    int retryCount=1;
    @GetMapping("/{userId}")
    //@CircuitBreaker(name = "ratingHotelBreaker",fallbackMethod = "ratingHotelFallBack")
    @Retry(name = "ratingHotelService",fallbackMethod = "ratingHotelFallBack")
    public ResponseEntity<User> getSingleUser(@PathVariable("userId") long userId){
        logger.info("retry count: {}",retryCount);

                retryCount++;
        User singleUser = userService.getSingleUser(userId);
        return new ResponseEntity<>(singleUser,HttpStatus.OK);
    }

    // creating fallback method for circuit breaker

    public ResponseEntity<User> ratingHotelFallBack(@PathVariable("userId") long
userId,Exception ex){
        logger.info("fall back is executed because service is down");
        User user = User.builder()
                .userEmail("dummy@gmail.com")
                .userName("dummy")
                .about("this user is created dummy")
                .userId(123)
                .build();
        return new ResponseEntity<>(user,HttpStatus.OK);
    }

    @GetMapping("/getAll")
    public ResponseEntity<List<User>> getAllUser(){
        List<User> allUser = userService.getAllUser();
        return new ResponseEntity<>(allUser,HttpStatus.OK);
    }
}
```

> update application.yml of user :

```
server:
  port: 8081


spring:
  # reading configuration from git hub with the help of config server
  config:
    import: configserver:http://localhost:8084

  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
    jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect

# for changing the name of application on server
  application:
    name: USER-SERVICE


# configuration for service discovery client
#eureka:
#  instance:
#    prefer-ip-address: true
#  client:
#    fetch-registry: true
#    register-with-eureka: true
#    service-url:
#      defaultZone: http://localhost:8761/eureka


# configuration for circuit breaker using resilience 4j
             # actuator
management:
  health:
    circuitbreaker:
      enabled: true
  endpoints:
    web:
      exposure:
        include: health

  endpoint:
    health:
      show-details: always


                      # resilience 4j

resilience4j:
  circuitbreaker:
    instances:
      ratingHotelBreaker:
        registerHealthIndicator: true
        eventConsumerBufferSize: 10
        failureRateThreshold: 50
        minimumNumberOfCalls: 5
        automaticTransitionFromOpenToHalfOpenEnabled: true
        waitDurationInOpenState: 6s
        permittedNumberOfCallsInHalfOpenState: 3
        slidingWindowSize: 10
        slidingWindowType: COUNT_BASED

        # configuration for retry
  retry:
    instances:
      ratingHotelService:
        max-attempts: 3
        wait-duration: 2s
```

RATE LIMITER IMPLEMENTATION IN USER

> dependecies are required like spring actuator,resilience4j and aop

> update user controller :

```
package com.user.service.controller;

import com.user.service.entities.User;
import com.user.service.service.UserService;
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import io.github.resilience4j.ratelimiter.annotation.RateLimiter;
import io.github.resilience4j.retry.annotation.Retry;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/user")
public class UserController {

    Logger logger = LoggerFactory.getLogger(UserController.class);

    @Autowired
    private UserService userService;
    @PostMapping("/save")
    public ResponseEntity<User> createUser(@RequestBody User user){
        User saveUser = userService.saveUser(user);
        return new ResponseEntity<>(saveUser, HttpStatus.CREATED);
    }


    // applying rate limiter annotation

    @GetMapping("/{userId}")
    //@CircuitBreaker(name = "ratingHotelBreaker",fallbackMethod =
"ratingHotelFallBack")
    //@Retry(name = "ratingHotelService",fallbackMethod = "ratingHotelFallBack")
    @RateLimiter(name = "userRateLimiter",fallbackMethod =
"ratingHotelFallBack")
    public ResponseEntity<User> getSingleUser(@PathVariable("userId") long
userId){
        User singleUser = userService.getSingleUser(userId);
        return new ResponseEntity<>(singleUser,HttpStatus.OK);
    }

    // creating fallback method for circuit breaker

    public ResponseEntity<User> ratingHotelFallBack(@PathVariable("userId")
long userId,Exception ex){
        logger.info("fall back is executed because service is down");
        User user = User.builder()
                .userEmail("dummy@gmail.com")
                .userName("dummy")
                .about("this user is created dummy")
                .userId(123)
                .build();
        return new ResponseEntity<>(user,HttpStatus.OK);
    }

    @GetMapping("/getAll")
    public ResponseEntity<List<User>> getAllUser(){
        List<User> allUser = userService.getAllUser();
        return new ResponseEntity<>(allUser,HttpStatus.OK);
    }
}
```

> update application.yml file of user :

```
server:
  port: 8081
```

```
spring:

  # reading configuration from git hub with the help of config server
  config:
    import: configserver:http://localhost:8084

  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
    jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect

# for changing the name of application on server
  application:
    name: USER-SERVICE


# configuration for service discovery client
#eureka:
#  instance:
#    prefer-ip-address: true
#  client:
#    fetch-registry: true
#    register-with-eureka: true
#    service-url:
#      defaultZone: http://localhost:8761/eureka


# configuration for circuit breaker using resilience 4j
             # actuator
management:
  health:
    circuitbreaker:
      enabled: true
  endpoints:
    web:
      exposure:
```

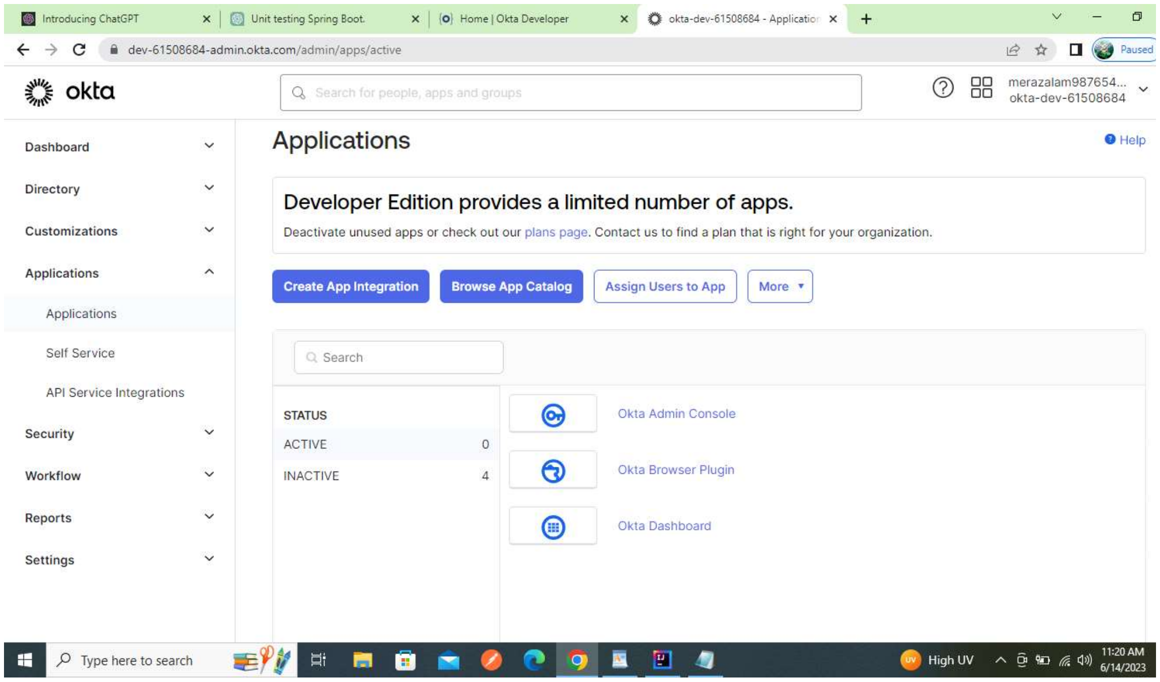
```
      include: health

  endpoint:
    health:
      show-details: always


                  # resilience 4j

resilience4j:
  circuitbreaker:
    instances:
      ratingHotelBreaker:
        registerHealthIndicator: true
        eventConsumerBufferSize: 10
        failureRateThreshold: 50
        minimumNumberOfCalls: 5
        automaticTransitionFromOpenToHalfOpenEnabled: true
        waitDurationInOpenState: 6s
        permittedNumberOfCallsInHalfOpenState: 3
        slidingWindowSize: 10
        slidingWindowType: COUNT_BASED

        # configuration for retry
  retry:
    instances:
      ratingHotelService:
        max-attempts: 3
        wait-duration: 2s


        # configuration for rate limiter
  ratelimiter:
    instances:
      userRateLimiter:
        limit-refresh-period: 4s
        limit-for-period: 2
        timeout-duration: 0s
```

SECURING MICROSERVICES WITH SPRING SECURITY AND OKTA AUTH

CREATING ACCOUNT ON OKTA

> signin with google or email

> create application for web :

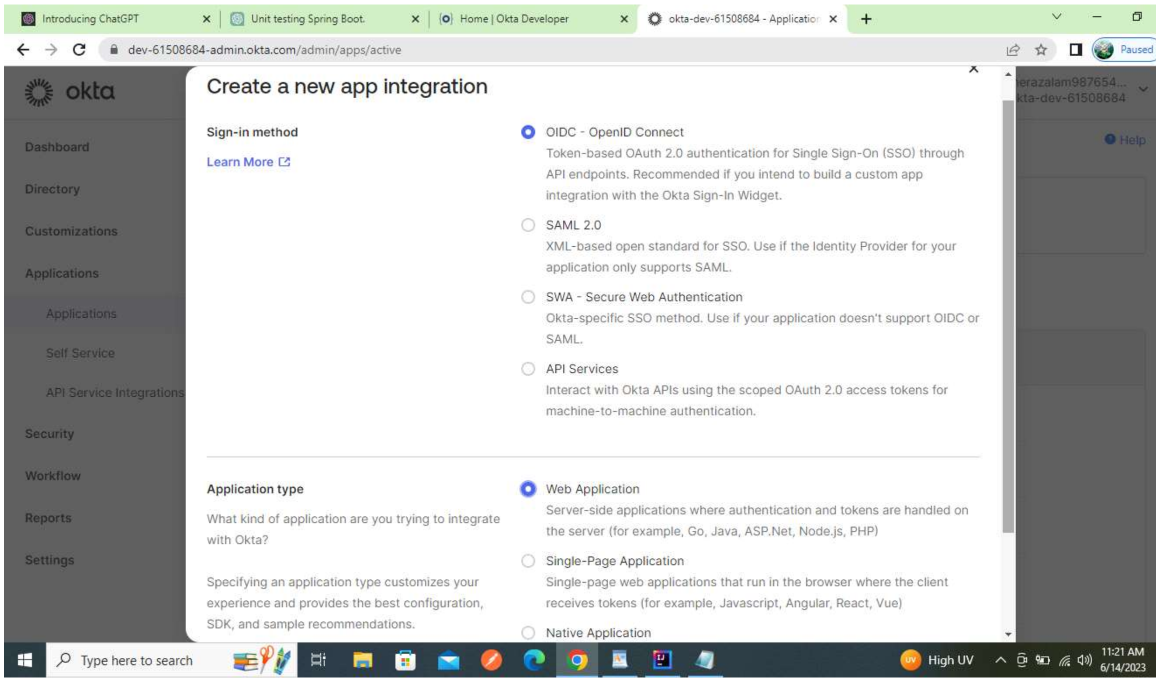


Applications

Developer Edition provides a limited number of apps.

Deactivate unused apps or check out our plans page. Contact us to find a plan that is right for your organization.

STATUS	
ACTIVE	0
INACTIVE	4


Create a new app integration

Sign-in method

Learn More

[✓ OIDC - OpenID Connect] Token-based OAuth 2.0 authentication for Single Sign-On (SSO) through API endpoints. Recommended if you intend to build a custom app integration with the Okta Sign-In Widget.

SAML 2.0 XML-based open standard for SSO. Use if the Identity Provider for your application only supports SAML.

SWA - Secure Web Authentication Okta-specific SSO method. Use if your application doesn't support OIDC or SAML.

API Services Interact with Okta APIs using the scoped OAuth 2.0 access tokens for machine-to-machine authentication.

Application type

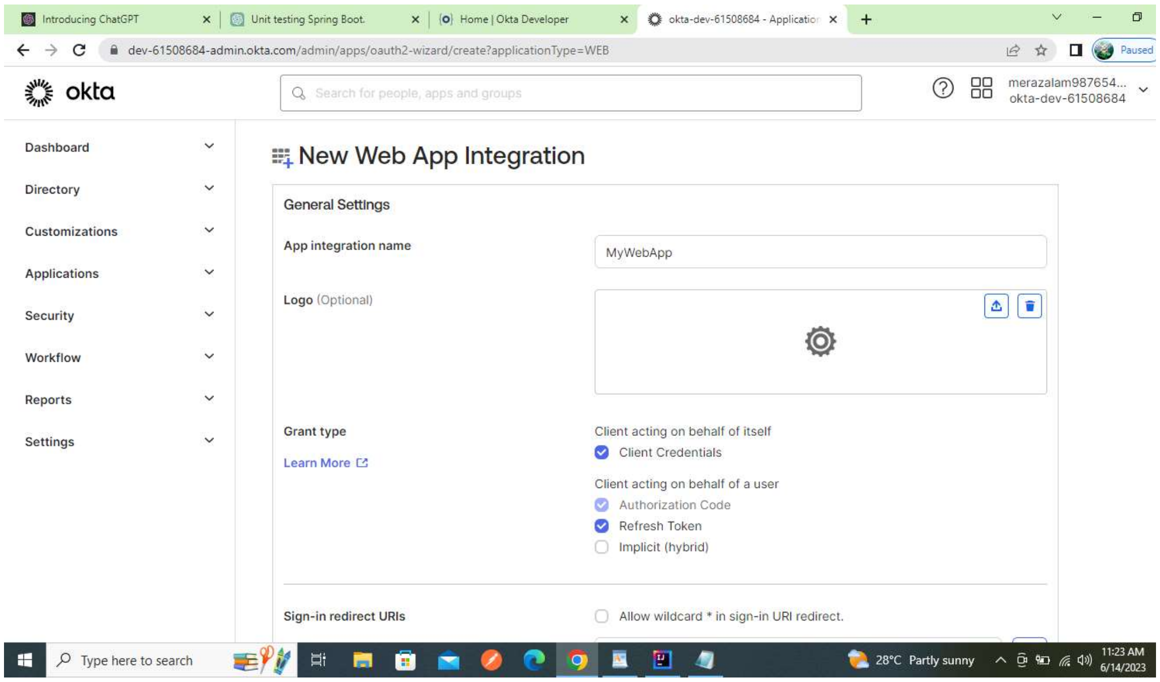
What kind of application are you trying to integrate with Okta?

Specifying an application type customizes your experience and provides the best configuration, SDK, and sample recommendations.

[✓ Web Application] Server-side applications where authentication and tokens are handled on the server (for example, Go, Java, ASP.Net, Node.js, PHP)

Single-Page Application Single-page web applications that run in the browser where the client receives tokens (for example, Javascript, Angular, React, Vue)

Native Application


New Web App Integration

General Settings

App integration name: MyWebApp

Logo (Optional)

Grant type

Learn More

Client acting on behalf of itself

[✓ Client Credentials]

Client acting on behalf of a user

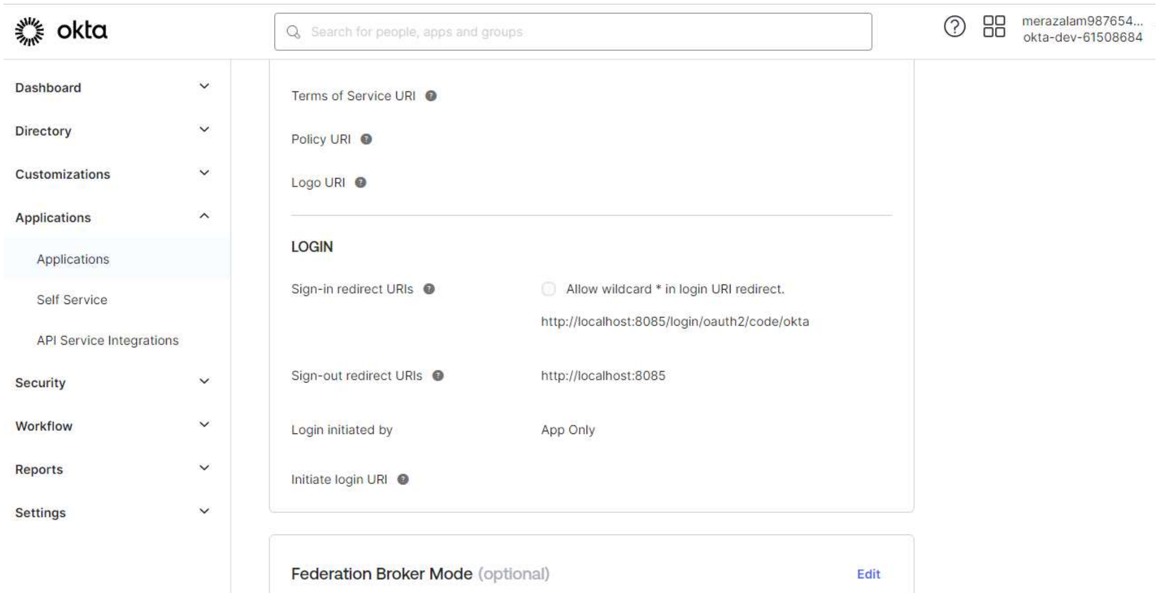
[✓ Authorization Code]

[✓ Refresh Token]

Implicit (hybrid)

Sign-in redirect URIs

Allow wildcard * in sign-in URI redirect.


Terms of Service URI

Policy URI

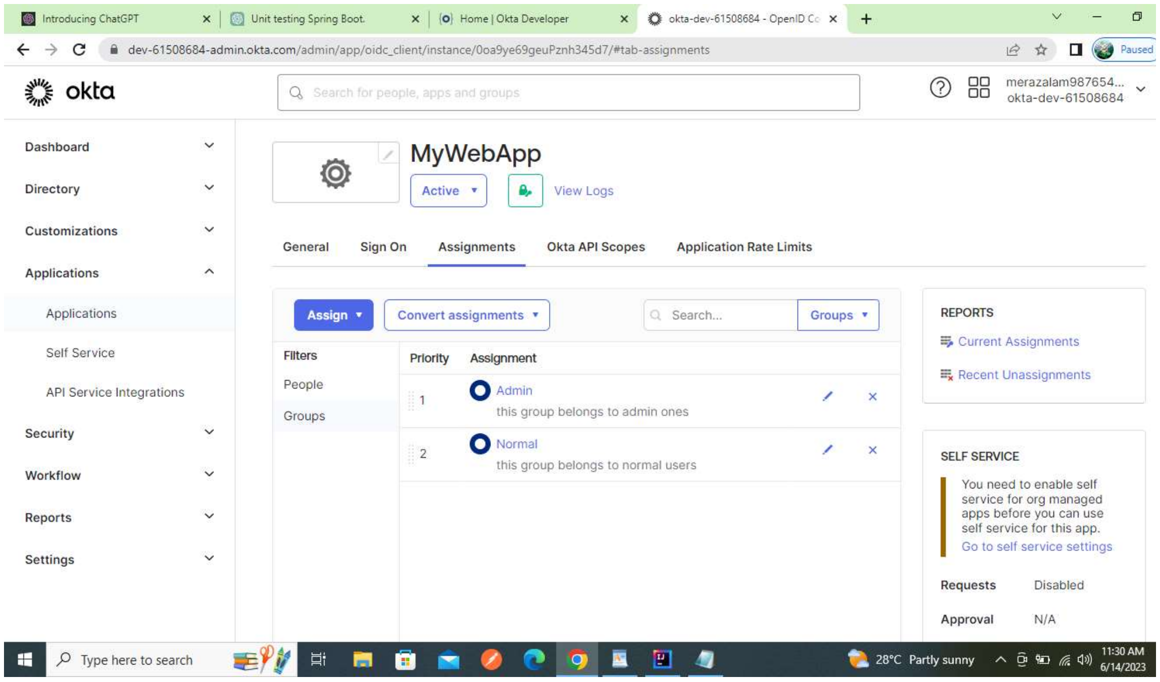
Logo URI

LOGIN

Sign-in redirect URIs	Allow wildcard * in login URI redirect. http://localhost:8085/login/oauth2/code/okta
Sign-out redirect URIs	http://localhost:8085
Login initiated by	App Only
Initiate login URI	

Federation Broker Mode (optional) Edit

> create groups for assining with the project:


dev-61508684-admin.okta.com/admin/app/oidc_client/instance/0oa9ye69geuPznh345d7/#tab-assignments

MyWebApp

Active · View Logs

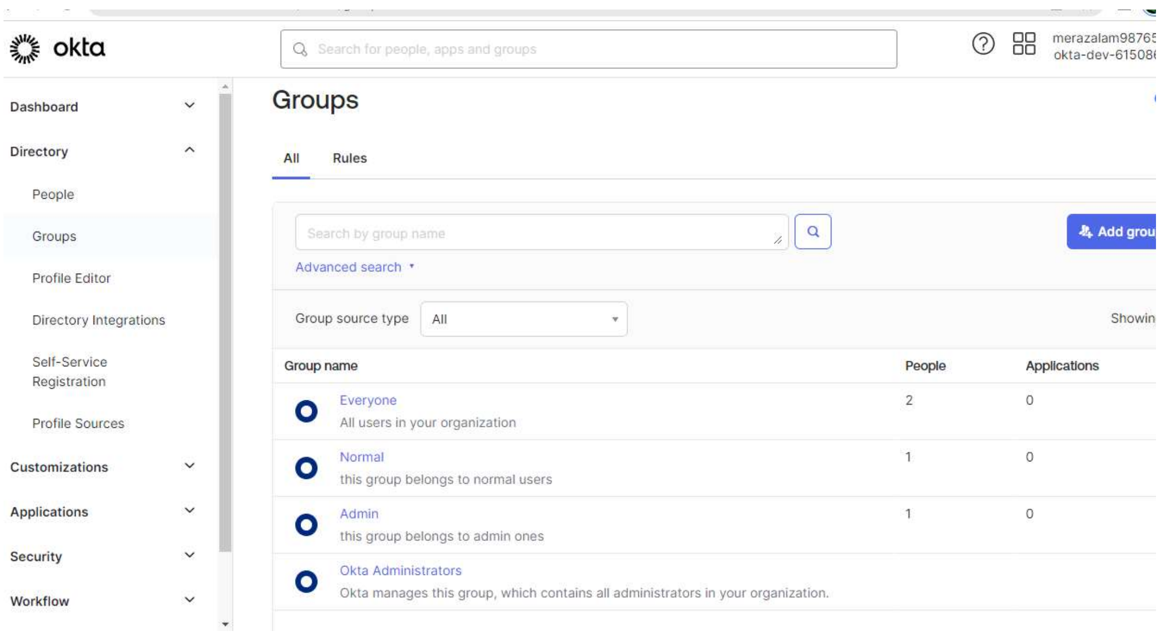
General · Sign On · Assignments · Okta API Scopes · Application Rate Limits

Priority	Assignment
1	Admin this group belongs to admin ones
2	Normal this group belongs to normal users

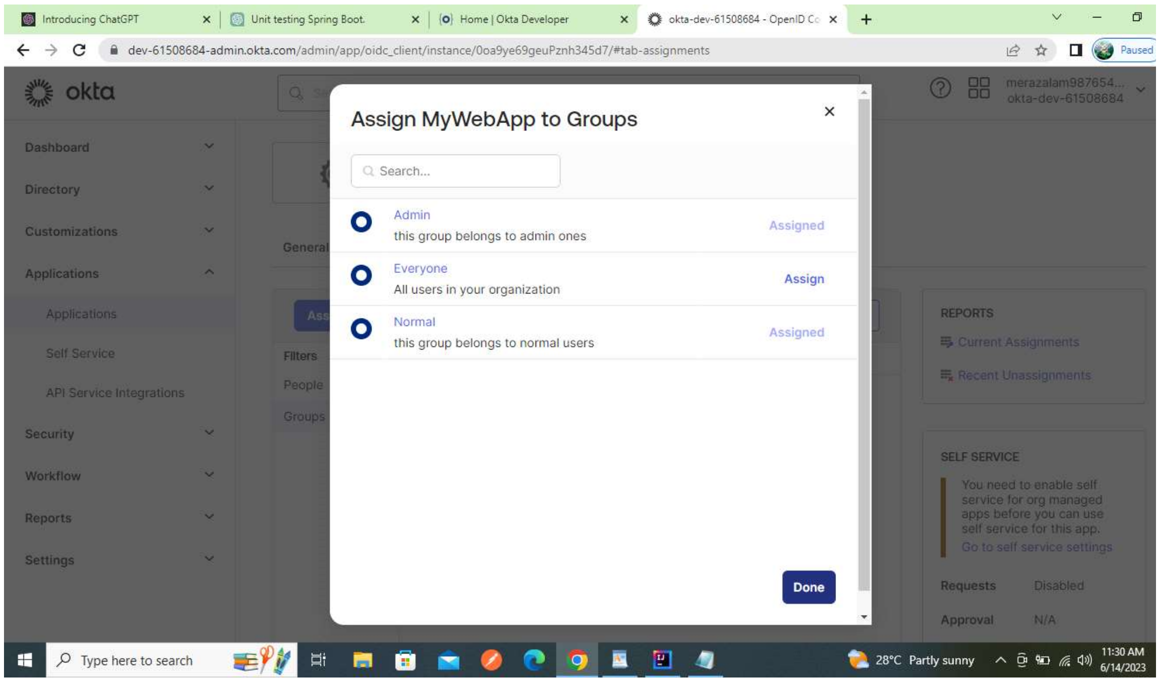
SELF SERVICE

You need to enable self service for org managed apps before you can use self service for this app.
Go to self service settings

Requests	Disabled
Approval	N/A



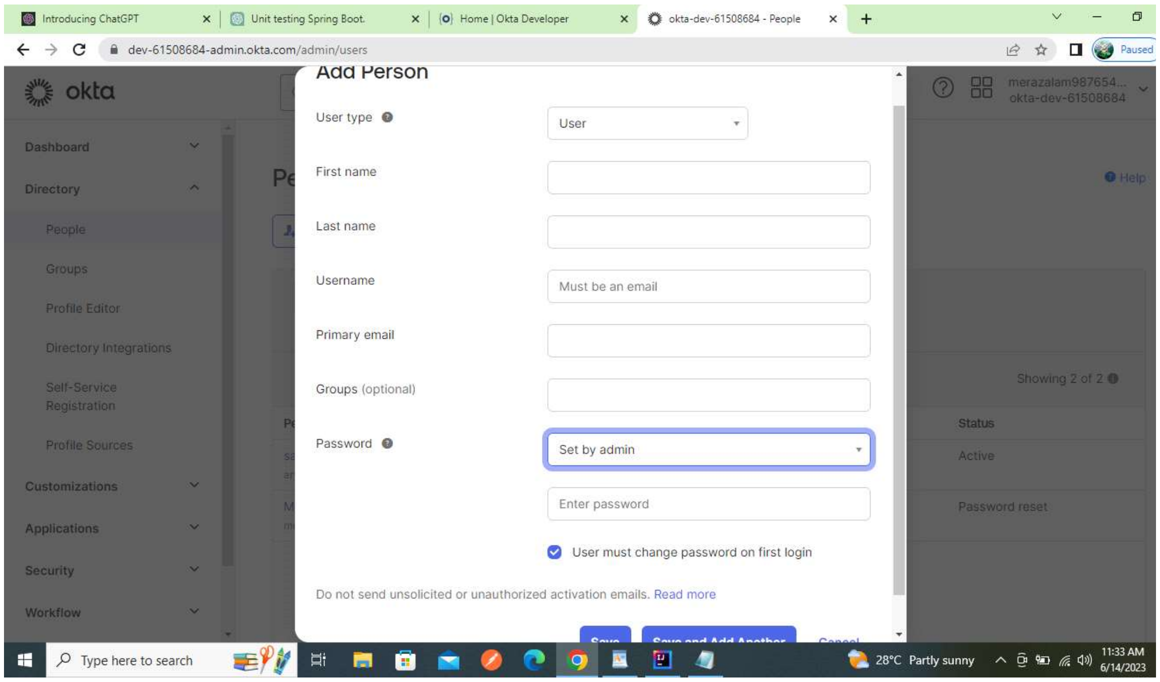
Group name	People	Applications
Everyone All users in your organization	2	0
Normal this group belongs to normal users	1	0
Admin this group belongs to admin ones	1	0
Okta Administrators Okta manages this group, which contains all administrators in your organization.		



Assign MyWebApp to Groups

Admin this group belongs to admin ones	Assigned
Everyone All users in your organization	Assign
Normal this group belongs to normal users	Assigned

> create the new person :



dev-61508684-admin.okta.com/admin/users

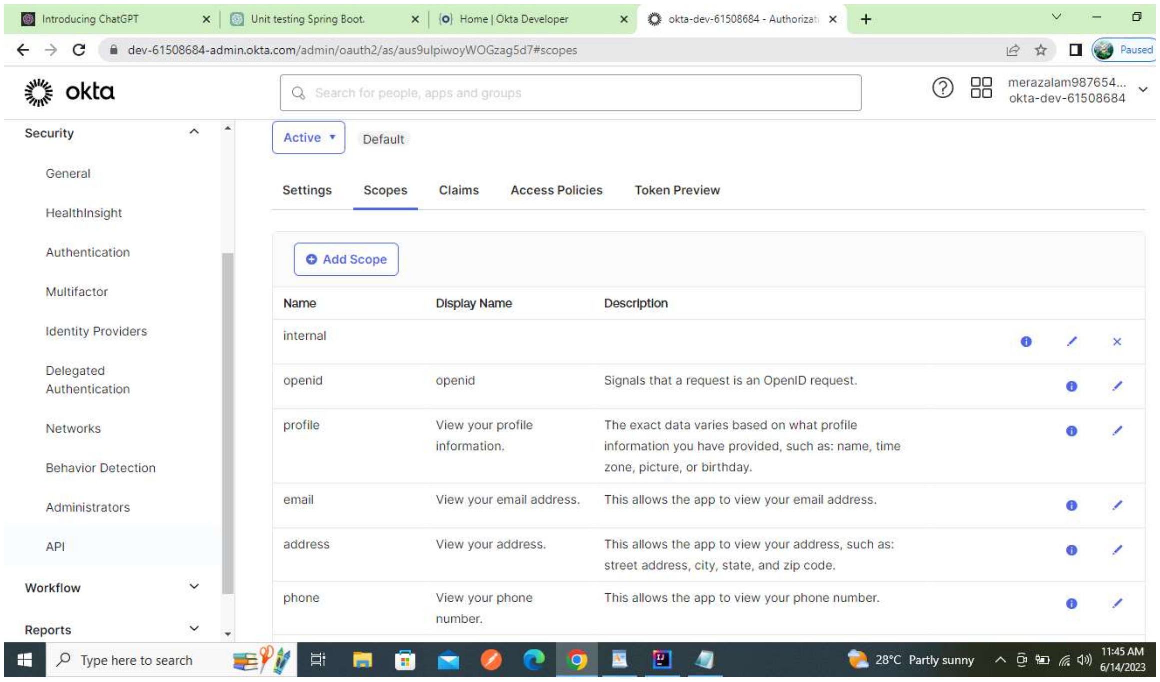
Add Person

User type	User
First name	
Last name	
Username	Must be an email
Primary email	
Groups (optional)	
Password	Set by admin
	Enter password

☑ User must change password on first login

Do not send unsolicited or unauthorized activation emails. Read more

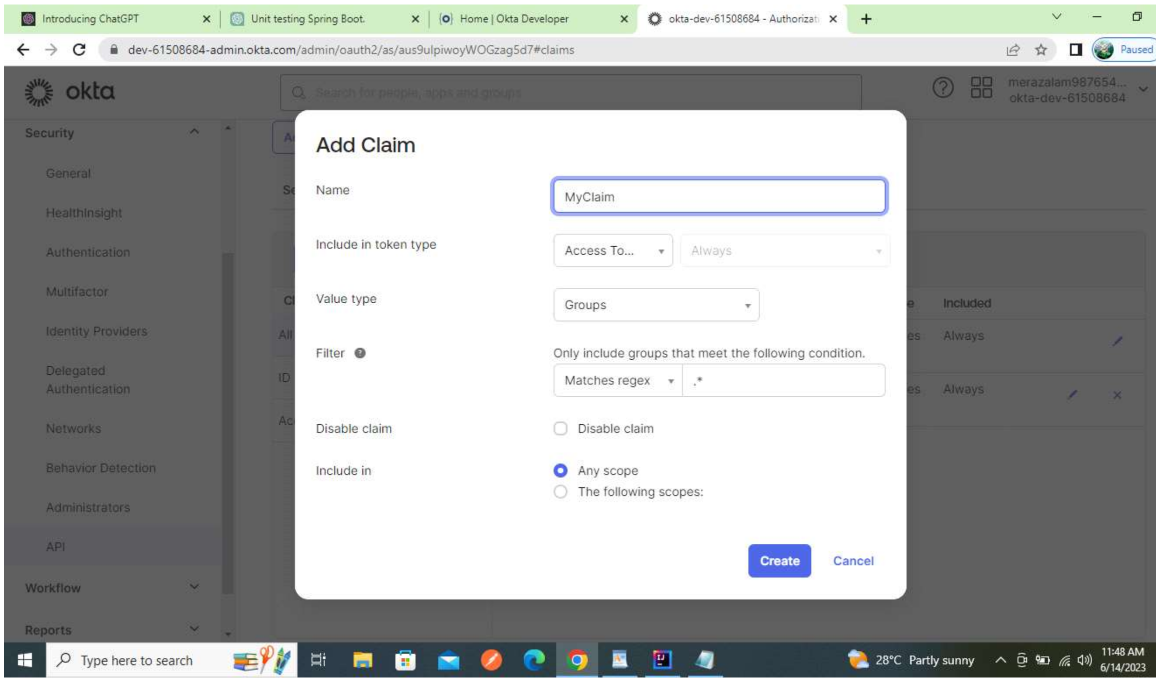
> add the scope :



dev-61508684-admin.okta.com/admin/oauth2/as/aus9ulpiwoyWOGzag5d7#scopes

Name	Display Name	Description
internal		
openid	openid	Signals that a request is an OpenID request.
profile	View your profile information.	The exact data varies based on what profile information you have provided, such as: name, time zone, picture, or birthday.
email	View your email address.	This allows the app to view your email address.
address	View your address.	This allows the app to view your address, such as: street address, city, state, and zip code.
phone	View your phone number.	This allows the app to view your phone number.

> add the claim :


Add Claim

Name: MyClaim

Include in token type: Access To... | Always

Value type: Groups

Filter: Only include groups that meet the following condition. Matches regex | .*

Disable claim: Disable claim

Include in: [✓ Any scope] The following scopes:

Create Cancel

IMPLEMENTING SPRING SECURITY AT API GATEWAY USING OKTA

> add okta dependecy with spring security dependency:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>


<dependency>
        <groupId>com.okta.spring</groupId>
        <artifactId>okta-spring-boot-starter</artifactId>
        <version>2.1.6</version>
</dependency>
```

Note- make sure that we should have dependecies like: web flux and api gateway in gateway then we can use okta and spring security with api gateway

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
</dependency>

<dependency>
```

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

configure spring security in api gateway:

> **create SecurityConfig class in config package** :-

```
  package com.api.Config;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.reactive.EnableWebFluxSecurity;
import org.springframework.security.config.web.server.ServerHttpSecurity;
import org.springframework.security.web.server.SecurityWebFilterChain;

@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {

   @Bean
   public SecurityWebFilterChain securityWebFilterChain(ServerHttpSecurity httpSecurity){
      httpSecurity
            .authorizeExchange()
            .anyExchange()
            .authenticated()
            .and()
            .oauth2Client()
            .and()
            .oauth2ResourceServer()
            .jwt();

      return httpSecurity.build();
   }

}
```

> **create a class AuthController in controller package** :-

```
package com.api.controller;
import com.api.model.AuthResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClient;
import org.springframework.security.oauth2.client.annotation.RegisteredOAuth2AuthorizedClient;
import org.springframework.security.oauth2.core.oidc.user.OidcUser;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/auth")
public class AuthController {

    private Logger logger = LoggerFactory.getLogger(AuthController.class);


    @GetMapping("/login")
    public ResponseEntity<AuthResponse> login(
            @RegisteredOAuth2AuthorizedClient("okta") OAuth2AuthorizedClient client,
            @AuthenticationPrincipal OidcUser user,
            Model  model
            ){
        logger.info("user email id:{}",user.getEmail());

        //creating authResponse object
        AuthResponse authResponse = new AuthResponse();

        //setting email to authResponse
        authResponse.setUserId(user.getEmail());

        //setting take to authResponse
        authResponse.setAccessToken(client.getAccessToken().getTokenValue());

        authResponse.setRefreshToken(client.getRefreshToken().getTokenValue());

        authResponse.setExpireAt(client.getAccessToken().getExpiresAt().getEpochSecond());

        // creating collection
        List<String> authorities = user.getAuthorities().stream().map(grantedAuthority -> {
            return grantedAuthority.getAuthority();
        }).collect(Collectors.toList());

        // setting for authorities of collection
        authResponse.setAuthorities(authorities);

        return new ResponseEntity<>(authResponse, HttpStatus.OK);
    }
}
```

> create AuthResponse class in model package:

```
package com.api.model;
import lombok.*;
import java.util.Collection;

@Data
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class AuthResponse {

    private String userId;
    private String accessToken;
    private String refreshToken;
    private long expireAt;
    private Collection<String> authorities;
}
```

> update application.yml file for configuration of Api Gateway:

```
server:
  port: 8085

spring:
  application:
    name: API-GATEWAY
  config:
    import: configserver:http://localhost:8084


# configuration for service discovery client
#eureka:
#  instance:
#    prefer-ip-address: true
#  client:
#    fetch-registry: true
#    register-with-eureka: true
#    service-url:
#      defaultZone: http://localhost:8761/eureka
```

```
  cloud:
    gateway:
      routes:
        - id: USER-SERVICE
          uri: lb://USER-SERVICE
          predicates:
            - Path=/api/user/**

        - id: HOTEL-SERVICE
          uri: lb://HOTEL-SERVICE
          predicates:
            - Path=/api/hotel/**,/api/staff/**

        - id: RATING-SERVICE
          uri: lb://RATING-SERVICE
          predicates:
            - Path=/api/rating/**

# okta configuration for securing api gateway
okta:
  oauth2:
    issuer: https://dev-61508684.okta.com/oauth2/default
    audience: api://default
    client-id: 0oa9ye69geuPznh345d7
    client-secret: wlWzVa97oIylfEGBEcR9DiG-XFhwrQNxJu6gCyaF
    scopes: openid, email, offline_access
```

implementing security in such a way so that it calls to another service with token in header.

> add the dependencies in user like :

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
        <groupId>com.okta.spring</groupId>
        <artifactId>okta-spring-boot-starter</artifactId>
        <version>2.1.6</version>
```

```
</dependency>
```

> configuring the application.yml of user:

```
server:
  port: 8081


spring:

  # reading configuration from git hub with the help of config server
  config:
    import: configserver:http://localhost:8084

  datasource:
    url: jdbc:mysql://localhost:3306/microsservices
    username: root
    password: test
    driver-class-name: com.mysql.cj.jdbc.Driver
    jpa:
    hibernate:
      ddl-auto: update
      show-sql: true
      properties:
        hibernate:
          dialect: org.hibernate.dialect.MySQL5Dialect

# for changing the name of application on server
  application:
    name: USER-SERVICE


# configuration for service discovery client
#eureka:
#  instance:
#    prefer-ip-address: true
#  client:
#    fetch-registry: true
#    register-with-eureka: true
#    service-url:
#      defaultZone: http://localhost:8761/eureka


# configuration for circuit breaker using resilience 4j
             # actuator
management:
  health:
    circuitbreaker:
      enabled: true
  endpoints:
    web:
      exposure:
        include: health

  endpoint:
    health:
      show-details: always


                # resilience 4j

resilience4j:
  circuitbreaker:
    instances:
      ratingHotelBreaker:
        registerHealthIndicator: true
        eventConsumerBufferSize: 10
        failureRateThreshold: 50
        minimumNumberOfCalls: 5
        automaticTransitionFromOpenToHalfOpenEnabled: true
        waitDurationInOpenState: 6s
        permittedNumberOfCallsInHalfOpenState: 3
        slidingWindowSize: 10
        slidingWindowType: COUNT_BASED

        # configuration for retry
  retry:
    instances:
      ratingHotelService:
        max-attempts: 3
        wait-duration: 2s


        # configuration for rate limiter
  ratelimiter:
    instances:
      userRateLimiter:
        limit-refresh-period: 4s
        limit-for-period: 2
        timeout-duration: 2s


# okta configuration
okta:
  oauth2:
    issuer: https://dev-61508684.okta.com/oauth2/default
    audience: api://default

# securing user as a client
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://dev-61508684.okta.com/oauth2/default
      client:
        registration:
          my-internal-client:
            provider: okta
            authorization-grant-type: client-credentials
            scope: internal
            client-id: 0oa9ye69geuPznh345d7
            client-secret: wlWzVa97oIyIfEGBEcR9DiG-XFhwrQNxJu6gCyaF

      provider:
        okta:
          issuer-id: https://dev-61508684.okta.com/oauth2/default
```

> creating WebSecurityConfig class in config package in user:

```
package com.user.service.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurit
y;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig {


    @Bean
    public SecurityFilterChain filterChain(HttpSecurity security) throws Exception {
        security
                .authorizeHttpRequests()
                .anyRequest()
                .authenticated()
```

```
            .and()
            .oauth2ResourceServer()
            .jwt();
        return security.build();
    }
}
```

CREATING FEIGN CLIENT INTERCEPTOR IN USER

> create a FeignClientInterceptor in interceptor package:

```
package com.user.service.interceptor;

import feign.RequestInterceptor;
import feign.RequestTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.oauth2.client.OAuth2AuthorizeRequest;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientManager;
import org.springframework.stereotype.Component;

@Configuration
@Component
public class FeingClientInterceptor implements RequestInterceptor {

    @Autowired
    private OAuth2AuthorizedClientManager manager;

    @Override
    public void apply(RequestTemplate requestTemplate) {
        String token = manager.authorize(OAuth2AuthorizeRequest
                      .withClientRegistrationId("my-internal-                 client")
                       .principal("internal").build())
                       .getAccessToken().getTokenValue();
        requestTemplate.header("Authorization","Bearer"+token);
    }
}
```

> create OAuth2AuthorizedClientManager Bean in Main class:

```
@Bean
public OAuth2AuthorizedClientManager manager(
                  ClientRegistrationRepository clientRegistrationRepository,
                  OAuth2AuthorizedClientRepository auth2AuthorizedClientRepository
){
        OAuth2AuthorizedClientProvider provider=
OAuth2AuthorizedClientProviderBuilder.builder().clientCredentials().build();

        DefaultOAuth2AuthorizedClientManager defaultOAuth2AuthorizedClientManager = new
DefaultOAuth2AuthorizedClientManager(clientRegistrationRepository,auth2AuthorizedClientRepo
sitory);
        // setting the authorized client provider
        defaultOAuth2AuthorizedClientManager.setAuthorizedClientProvider(provider);
        return defaultOAuth2AuthorizedClientManager;
}
```

CREATING RESTTEMPLATE INTERCEPTOR

> create a class named RestTemplateInterCeptor extends ClientHttpRequestInterceptor:

```
package com.user.service.interceptor;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpRequest;
import org.springframework.http.client.ClientHttpRequestExecution;
import org.springframework.http.client.ClientHttpRequestInterceptor;
import org.springframework.http.client.ClientHttpResponse;
import org.springframework.security.oauth2.client.OAuth2AuthorizeRequest;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientManager;

import java.io.IOException;

public class RestTemplateInterceptor implements ClientHttpRequestInterceptor {


    private OAuth2AuthorizedClientManager manager;

    public RestTemplateInterceptor(OAuth2AuthorizedClientManager manager) {
        this.manager = manager;
    }

    @Override
    public ClientHttpResponse intercept(HttpRequest request, byte[] body,
ClientHttpRequestExecution execution) throws IOException {
        // getting the token
        String token = manager.authorize(OAuth2AuthorizeRequest
                .withClientRegistrationId("my-internal-client")
                .principal("internal").build())
            .getAccessToken().getTokenValue();
        request.getHeaders().add("Authorization","Bearer"+token);
        return execution.execute(request, body);
    }
}
```

> update the restTemplate Bean in main class:

```
package com.user.service;

import com.user.service.interceptor.RestTemplateInterceptor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.http.client.ClientHttpRequestInterceptor;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientManager;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProvider;
import org.springframework.security.oauth2.client.OAuth2AuthorizedClientProviderBuilder;
import org.springframework.security.oauth2.client.registration.ClientRegistrationRepository;
import org.springframework.security.oauth2.client.web.DefaultOAuth2AuthorizedClientManager;
import org.springframework.security.oauth2.client.web.OAuth2AuthorizedClientRepository;
import org.springframework.web.client.RestTemplate;

import java.util.ArrayList;
import java.util.List;

@SpringBootApplication
@EnableFeignClients
@EnableEurekaClient
public class UserApplication {
        @Autowired
        private ClientRegistrationRepository clientRegistrationRepository;
        @Autowired
        private OAuth2AuthorizedClientRepository auth2AuthorizedClientRepository;

        @Bean
        @LoadBalanced
        public RestTemplate restTemplate(){
                RestTemplate restTemplate = new RestTemplate();
                List<ClientHttpRequestInterceptor> interceptors= new ArrayList<>();
                interceptors.add(new
RestTemplateInterceptor(manager(clientRegistrationRepository,auth2AuthorizedClientRepository)
));
                return restTemplate;
        }
```

```
	// declare the bean of OAuth2AuthorizedClient manager
	@Bean
	public OAuth2AuthorizedClientManager manager(
			ClientRegistrationRepository clientRegistrationRepository,
			OAuth2AuthorizedClientRepository auth2AuthorizedClientRepository
	){
		OAuth2AuthorizedClientProvider provider=
OAuth2AuthorizedClientProviderBuilder.builder().clientCredentials().build();

		DefaultOAuth2AuthorizedClientManager defaultOAuth2AuthorizedClientManager
= new
DefaultOAuth2AuthorizedClientManager(clientRegistrationRepository,auth2AuthorizedClientRepo
sitory);
		// setting the authorized client provider
		defaultOAuth2AuthorizedClientManager.setAuthorizedClientProvider(provider);
		return defaultOAuth2AuthorizedClientManager;
	}

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

}
```

IMPLEMENTING SPRING SECURITY WITH OKTA AT RATING SERVICE

> add the dependency in rating service:

```
<dependency>
	<groupId>org.springframework.boot</groupId>
	<artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
	<groupId>com.okta.spring</groupId>
	<artifactId>okta-spring-boot-starter</artifactId>
	<version>2.1.6</version>
</dependency>
```

> update configuration of rating in application.yml file:

```
# okta configuration
okta:
  oauth2:
    issuer: https://dev-61508684.okta.com/oauth2/default
    audience: api://default
```

> create SecurityConfig in config package in rating service:

```
package com.rating.service.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurit
y;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity security) throws Exception {
        security
                .authorizeHttpRequests()
                .anyRequest()
                .authenticated()
                .and()
                .oauth2ResourceServer()
                .jwt();
        return security.build();
    }
}
```

> update Rating Controller for giving the authority using@PreAuthorize:

```
package com.rating.service.controller;

import com.rating.service.entities.Rating;
import com.rating.service.service.RatingService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.util.List;
```

```
@RestController
@RequestMapping("/api/rating")
public class RatingController {

   @Autowired
   private RatingService ratingService;
   @PreAuthorize("hasAuthority('Admin')")
   @PostMapping("/save")
   public ResponseEntity<Rating> saveRating(@RequestBody Rating rating){
      Rating saveRating = ratingService.createRating(rating);
      return new ResponseEntity<>(saveRating, HttpStatus.CREATED);
   }

   @GetMapping("/getAll")
   public ResponseEntity<List<Rating>> getAllRating(){
      List<Rating> allRating = ratingService.getAllRating();
      return new ResponseEntity<>(allRating,HttpStatus.OK);
   }
   @PreAuthorize("hasAuthority('SCOPE_internal') || hasAuthority('Admin')")
   @GetMapping("/user/{userId}")
   public ResponseEntity<List<Rating>> getRatingByUserId(@PathVariable("userId") long userId)
{
      List<Rating> ratingByUserId = ratingService.getRatingByUserId(userId);
      return new ResponseEntity<>(ratingByUserId,HttpStatus.OK);
   }

   @GetMapping("/hotel/{hotelId}")
   public ResponseEntity<List<Rating>> getRatingByHotelId(@PathVariable("hotelId") long
hotelId){
      List<Rating> ratingByHotelId = ratingService.getRatingByHotelId(hotelId);
      return new ResponseEntity<>(ratingByHotelId,HttpStatus.OK);
   }
}
```

<u>IMPLEMENTING SPRING SECURITY WITH OKTA AT HOTEL SERVICE</u>

> add the dependency in hotel service:

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
        <groupId>com.okta.spring</groupId>
```

```
            <artifactId>okta-spring-boot-starter</artifactId>
            <version>2.1.6</version>
</dependency>
```

> update configuration of hotel in application.yml file:

```
# okta configuration
okta:
  oauth2:
    issuer: https://dev-61508684.okta.com/oauth2/default
    audience: api://default
```

> create a class SecurityConfig in config package in hotel:

```
package com.hotel.service.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurit
y;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity security) throws Exception {
      security
            .authorizeHttpRequests()
            .anyRequest()
            .authenticated()
            .and()
            .oauth2ResourceServer()
            .jwt();
      return security.build();
   }
}
```

> update Hotel Controller for giving the authority using@PreAuthorize:

```
package com.hotel.service.controller;
```

```
import com.hotel.service.entities.Hotel;
import com.hotel.service.service.HotelService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/hotel")
public class HotelController {

    @Autowired
    private HotelService hotelService;

    @PreAuthorize("hasAuthority('Admin')")
    @PostMapping("/save")
    public ResponseEntity<Hotel> createHotel(@RequestBody Hotel hotel){
        Hotel savedHotel = hotelService.createHotel(hotel);
        return new ResponseEntity<>(savedHotel, HttpStatus.CREATED);
    }

    @PreAuthorize("hasAuthority('SCOPE_internal')")
    @GetMapping("/{hotelId}")
    public ResponseEntity<Hotel> getSingleHotel(@PathVariable("hotelId")long hotelId){
        Hotel hotelById = hotelService.getHotelById(hotelId);
        return new ResponseEntity<>(hotelById,HttpStatus.OK);
    }

    @PreAuthorize("hasAuthority('SCOPE_internal') || hasAuthority('Admin')")
    @GetMapping("/getAll")
    public ResponseEntity<List<Hotel>> getAllHotel(){
        List<Hotel> allHotel = hotelService.getAllHotel();
        return new ResponseEntity<>(allHotel,HttpStatus.OK);
    }
}
```

Note:- same way we can do in UserController also

TESTING MICROSERVICE APP AFTER SECURING

> Run all the services :

>