Laborator 16: RESTful Webservices Zuul API Gateway

Zuul API Gateway

Interceptează cererile și le deleagă microsericiilor. Poate fi folosit pentru:
-- autentificare/securitate centralizată
-- fault tolerance/debugging
-- logging etc.

1. Folosind start.spring.io creați un proiect cu Artifact Id: zuul-api-gateway. Adăugați dependențele: Zuul și Eureka Discovery.

2. Adăugați în application.properties:

```
spring.application.name=zuul-api-gateway
server.port=8762
eureka.client.service-url.default-zone = http://localhost:8761/eureka
```

3. Adnotați clasa ZuulApiGatewayApplication pentru a activa proxy-ul Zuul default:

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

Zuul Filter

Metode:

-- filterOrder	setează prioritatea în cazul în care există mai multe filtre.
-- shouldFilter	stabilește dacă acțiunea filtrului se va exeuta.
-- filterType	stabilește momentul în care se va executa filtrul. (pre/post/error)
-- run	stabilește acțiunile executate de filtru.

4. Implementați clasa abstractă ZuulFilter:

```
@Component
public class LoggingFilter extends ZuulFilter {
    @Override
    public String filterType() {
        return "pre";
    }

    @Override
    public int filterOrder() {
        return 1;
    }

    @Override
    public boolean shouldFilter() {
        return true;
    }

    @Override
    public Object run() throws ZuulException {
        return null;
    }
}
```

5. În metoda run, creați o variabilă care va reține cererea http care va fi interceptată de către filtru și creați un log a requestu-ului:

```
RequestContext.getCurrentContext().getRequest();@Override
    public Object run() throws ZuulException {
        HttpServletRequest req =
                RequestContext.getCurrentContext().getRequest();
        log.info("Request {}  uri {} ",req, req.getRequestURI());
        return null;
    }
```

6. Porniți în ordine serverul eureka, aplicația subscription și proxy-ul zuul. Testați requestul:

```
http://localhost:8762/subscription/subscription/coach/James/sport/tennis

http://localhost:8762/{applicationname}/subscription/coach/James/sport/tennis
```

7. Porniți pricecalculator și testați:

```
http://localhost:8762/pricecalculator/subscriptionfeign/coach/James/sport/tennis/months/3
```

8. Modificați în aplicația pricecalculator interfața SubscriptionServiceProxy.

```
//@FeignClient(name = "subscription"/*, url = "localhost:8082"*/)
//@RibbonClient(name = "subscription")
@FeignClient(name = "zuul-api-gateway")
@RibbonClient(name = "subscription")
public interface SubscriptionServiceProxy {
//@GetMapping("/subscription/coach/{coach}/sport/{sport}")

@GetMapping("/subscription/subscription/coach/{coach}/sport/{sport}")
SubscriptionPrice findByCoachAndSport(@PathVariable String coach,
                                      @PathVariable String sport);
}
```

9. Testați:

```
http://localhost:8762/pricecalculator/subscriptionfeign/coach/James/sport/tennis/months/3
```

Autentificare Oauth https://oauth.net/2/

Protocol care stabilește modul în care sunt gestionate (protejate) resuresele transmise către clienți de unul sau mai multe servicii.

Roluri:
-- Client
-- Resource Owner
-- Resource Server serviciul care servește clientului resurse
-- Authorization Server intermediar între client și Resource Owner

JWT JSON Web Token

Avantaje:
-- stateless authorization
-- mai multe servere pot utiliza același authorization server.
-- format (JSON) ușor de utilizat

Structura JWT:
-- **header** conține tipul tokenului și tipul algoritmului folosit pentru criptare

{type: "JWT", hash: "HS256"}

-- **payload** conține date despre utilizator

{username: "hello", email: "hello@apbdoo.com"}

-- **signature** obținută folosind o cheie secretă

Workflow JWT JSON Web Token

-- clientul trimite o cerere de autorizare la authentication server.
-- clientul primește un token.
-- fiecare cerere va fi însoțită de token. Serverul va valida tokenul.

10. Adăugați dependențele pentru spring security în proiectul zuul-api-gateway:

```
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-test</artifactId>
   <scope>test</scope>
</dependency>

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

11. Tokenul jwt va fi salvat într-o baza de date mongo. Adăugați dependeța pentru jwt și mongo:

```
<dependency>
       <groupId>io.jsonwebtoken</groupId>
       <artifactId>jjwt</artifactId>
       <version>0.9.1</version>
</dependency>

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

12. Creați pachetul auth cu clasele User și Role.

```
@Data
public class Role {
    private Integer id;
    private String  role;
}
```

```
@Data
@Document
public class User {

    @Id
    private String id;
    @Email(message = "enter valid email")
    @NotEmpty(message = "provide email")
    private String email;

    @NotEmpty(message = "provide password")
    private String password;

    @NotEmpty(message = "provide name")
    private String name;

    private Integer active=1;
    private boolean isLocked=false;
    private boolean isExpired=false;
    private boolean isEnabled=true;

    private Set<Role> role;
}
```

13. Instalați și porniți serverul mongo și clientul mongo.

```
>>mongod
>>mongo
```

14. Configurați conexiunea la baza de date mongo în fișierul application.properties.

```
spring.data.mongodb.authentication-database=admin
spring.data.mongodb.username=root
spring.data.mongodb.password=root
spring.data.mongodb.database=test
spring.data.mongodb.port=27017
spring.data.mongodb.host=localhost
```

15. Creați clasa MongoUserDetails.

```
public class MongoUserDetails implements UserDetails {
    private String username;
    private String password;
    private Integer active;
    private boolean isLocked;
    private boolean isExpired;
    private boolean isEnabled;
    private List<GrantedAuthority> grantedAuthorities;


    public MongoUserDetails(String username, String
password,Integer active, boolean isLocked, boolean isExpired,
boolean isEnabled, String [] authorities) {
        username = username;
        password = password;
        active = active;
        isLocked = isLocked;
        isExpired = isExpired;
        isEnabled = isEnabled;
        grantedAuthorities =
AuthorityUtils.createAuthorityList(authorities);
    }

    @Override
    public Collection<? extends GrantedAuthority>
getAuthorities() {
        return grantedAuthorities;
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return username;
    }

    @Override
    public boolean isAccountNonExpired() {
        return !isExpired;
    }

    @Override
    public boolean isAccountNonLocked() {
        return !isLocked;
    }
```

```
    @Override
    public boolean isCredentialsNonExpired() {
        return !isExpired;
    }

    @Override
    public boolean isEnabled() {
        return isEnabled;
    }
}
```

16. Folosind clientul mongo, creați utilizatorul root:

```
use admin

db.createUser(
{
    user: "root",
    pwd: "root",
    roles: [ "root" ]
})
```

17. Creați în pachetul repositories, un repository mongo.

```
@Repository
public interface UserRepository extends
MongoRepository<User,String> {
     @Query(value="{'email' : ?0}")
     User findByEmail(String email);
}
```

18. Creați în pachetul services, servicul care va returna detaliile utilizatorului în funcție de email:

```
@Service
public class UserService implements UserDetailsService {
    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {
        User user = userRepository.findByEmail(email);
        if (user == null || user.getRole() == null ||
user.getRole().isEmpty()) {
```

```
            throw new UsernameNotFoundException("Invalid username
or password.");
        }
        String[] authorities = new String[user.getRole().size()];
        int count = 0;
        for (Role role : user.getRole()) {
            authorities[count] = "ROLE_" + role.getRole();
            count++;
        }
        MongoUserDetails userDetails =
                new MongoUserDetails(user.getEmail(),
                                     user.getPassword(),
                                     user.getActive(),
                user.isLocked(), user.isExpired(),
user.isEnabled(), authorities);
        return userDetails;
    }
}
```

19. Creați în mongo colecția User:

```
db.createCollection("user")
```

20. Adăugați utilizatorul test:

```
db.user.insert({"id":"usertest@yahoo.com","email":"usertest@yahoo.com","pas
sword":"test","name":"test", "role":[{"role":"admin"}]})
```

21. Adăugați clasa. Executați testul:

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserServiceTest {
    public static final String EMAIL = "usertest@yahoo.com";

    @Autowired
    UserService userService;

    @Test
    public void testUserService(){
        UserDetails userDetails = userService.loadUserByUsername(EMAIL);
        assertEquals(EMAIL, userDetails.getUsername());
    }
}
```