

**UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET**

Mikroservis Diskusije

ONLINE PLATFORMA ZA UČENJE C++-a

Dokumentacija za zadatak 2.

Predmet: Praktikum - Napredne web tehnologije

Predmetni nastavnik:
Doc. dr Vensada Okanović, dipl. ing. el.

Student: Lejla Nukić

Broj indeksa:16738/1125

Sarajevo, mart 2017.

Pojedinačna dokumentacija za Zadatak 2.

1. U sklopu ovog zadatka implementirane su dodatne metode REST servisa koje nisu generisane unutar repository-a iz prošlosedmičnog zadatka.

Konkretno, ovdje će biti dat kratak opis implementacije i primjene istih koje se nalaze u sklopu mikroservisa ModulDiskusije.

Kao rezultat implementacije ovih metoda, dobit ćemo odgovore na HTTP zahtjeve u JSON reprezentaciji.

U okviru prošlog zadatka smo kreirali odgovarajuće klase, odnosno modele čije ćemo objekte vraćati u JSON formatu kao odgovor na odgovarajući HTTP zahtjev.

Nakon toga, potrebno je kreirati odgovarajuće resurse, odnosno controllere koji će opsluživati i rukovati HTTP zahtjevima. Controller-i su smješteni unutar paketa `com.example.controllers`.

Na sljedećoj slici je prikazan trenutni kod napisan u controlleru `RegisteredUserController`:



```
Dashboard RegisteredUserController.java
package com.example.controllers;

import java.util.concurrent.atomic.AtomicLong;

@RestController
public class RegisteredUserController {

    private final AtomicLong br = new AtomicLong();

    @RequestMapping("/korisnici")
    public RegisteredUser korisnici(@RequestParam(value="username") String username){

        RegisteredUser rg = new RegisteredUser();
        rg.setUsername(username);
        rg.setId(br.incrementAndGet());
        return rg;
    }
}
```

Ove resurse je potrebno anotirati sa anotacijom `@RestController` kako bismo označili resurs kao controller koji će handle-ati odgovarajuće http zahtjeve. Pomoću anotacije `@RequestMapping` omogućeno je da se handle-aju zahtjevi na stazu `/korisnici`, da se ti zahtjevi mapiraju na metodu `korisnici()`, a da odgovor na te zahtjeve bude instanca klase `RegisteredUser`. Pomoću anotacije `@RequestParam` omogućeno je da se vrijednost parametra **username** koju pošaljemo u sklopu http zahtjeva smjesti u parametar `username` metode `korisnici()` i da se kasnije ta vrijednost koristi za

postavljanje vrijednosti atributa nove instance klase `RegisteredUser` i sl.

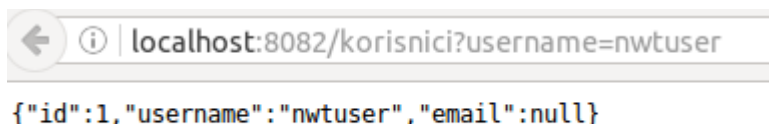
Tijelo metode je jednostavno i trenutno služi samo za kreiranje i vraćanje nove instance klase `RegisteredUser` koja ima `id` postavljen u skladu sa vrijednošću varijable `counter` i `username` u skladu sa vrijednosti parametra `username` iz http zahtjeva.

Treba napomenuti da nam je RESTful web service controller omogućio da popunjavamo i vraćamo objekat klase `RegisteredUser`. Također, ovaj objekat će biti zapisan kao JSON objekat u sklopu slanja odgovora na http zahtjev. Ovu konverziju će automatski obaviti Spring-ov konverter poruka [MappingJackson2HttpMessageConverter](#).

Za testiranje controllera dovoljno je pokrenuti aplikaciju i ukucati u browser sljedeći sadržaj:

<http://localhost:8082/korisnici?username=nwtuser>,

i kao rezultat ćemo dobiti sljedeći odgovor:



```
localhost:8082/korisnici?username=nwtuser  
{"id":1,"username":"nwtuser","email":null}
```

Vidimo da se radi o objektu u JSON formatu sa odgovarajućim `username`-om.

Na sljedećoj slici je prikazan još jedan controller, odnosno `TagController`.



```
Dashboard RegisteredUserController.java TagController.java  
package com.example.controllers;  
  
import java.util.ArrayList;  
  
@RestController  
public class TagController {  
  
    private final AtomicLong br = new AtomicLong();  
  
    @Autowired  
    private TagRepository tr;  
  
    @RequestMapping("/tagovi")  
    public Tag tagovi(@RequestParam(value="title") String title){  
        return new Tag(br.incrementAndGet(),title);  
    }  
  
    @RequestMapping("/tagovideset")  
    public List<Tag> prvihDeset(@RequestParam(value="name") String name){  
        return tr.findFirst10ByName(name);  
    }  
  
    @RequestMapping("/nema")  
    public Boolean nema(@RequestParam("like") String like){  
        List<Tag> lt=tr.findByNameLike("%"+like+"%");  
  
        return lt.isEmpty();  
    }  
}
```

Za razliku od prethodnog controllera vidimo da ovaj controller ima u sebi privatni atribut **tr** tipa TagRepository, koji je anotiran @Autowired anotacijom, obzirom da je TagRepository zapravo interfejs koji sadrži odgovarajuće metode koje ćemo pozivati u sklopu ovog controllera.

Na sljedećoj slici je prikazan kod TagRepository-a:

```
package com.example.repositories;

import java.util.ArrayList;

@RepositoryRestResource(path="tags", collectionResourceRel="tags")
public interface TagRepository extends CrudRepository<Tag, Long> {

    public List<Tag> findFirst10ByName(String name);

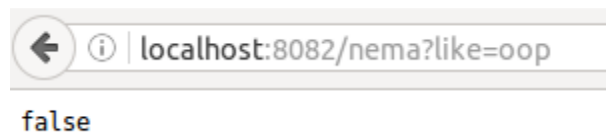
    public List<Tag> findByNameLike(String name);
}
```

U okviru repository-ja dodane su metode sa nazivima u skladu sa konvencijom imenovanja koja omogućava generisanje odgovarajućih query-ja. (Izvor: <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.query-creation>)

Prva metoda vraća prvih 10 tag-ova sa imenom koje je proslijeđeno kao parametar metodi, a druga metoda vraća one tagove čija imena sadržavaju kao podstring vrijednost proslijeđenu kao parametar metode.

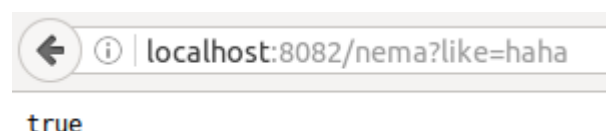
U okviru metode nema() TagControllera, pozivamo metodu findByNameLike() nad **tr**-om kako bismo provjerili da li postoje tagovi koji u imenu sadrže kao podstring proslijeđen kao vrijednost parametra like u sklopu http zahtjeva.

Na sljedećim slikama prikazan je rezultat dva poslana http get zahtjeva:



Ovdje vidimo da smo kao rezultat dobili false, odnosno lista vraćenih tagova nije prazna što je indikator da postoji tag koji u imenu sadrži kao podstring oop.

Međutim, na sljedećoj slici vidimo da je rezultat true u slučaju da je poslana vrijednost parametra like haha, što znači da je lista vraćenih tagova prazna, odnosno ne postoji nijedan tag sa imenom koje kao podstring sadrži haha.



2. Za potrebe centralizovane konfiguracije kreiran je novi projekat, te je dodan sljedeći kod.

```
package com.example;

import org.springframework.boot.SpringApplication;

@EnableConfigServer
@SpringBootApplication
public class ConfigurationServerModuleApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigurationServerModuleApplication.class, args);
        System.out.println("${HOME}");
    }
}
```

Može se uočiti nova anotacija `@EnableConfigServer` koja će obezbijediti da se konfigurira server sa kojim će ostale aplikacije komunicirati.

U okviru `application.properties`-a ovog projekta dodan je sljedeći kod:

```
server.port=8888

#spring.cloud.config.server.git.uri=${HOME}/Desktop/gitrepo
spring.cloud.config.server.git.uri=https://github.com/AmarPanjeta/NWTCentralizedConf.git
security.user.name=root
security.user.password=s3cr3t
```

kako bi se centralizovana konfiguracija koja se nalazi na ovom github repozitoriju, sa servera isporučivala odgovarajućim klijentima.

Nakon toga je potrebno na klijentu kreirati novi fajl `bootstrap.properties` (u `src/main/resources`) i dodati sljedeći kod unutar njega:

```
spring.application.name=discussions-client
spring.cloud.config.uri=http://localhost:8888
management.security.enabled=false
spring.cloud.config.username=root
spring.cloud.config.password=s3cr3t
```

a `application.properties` na klijentu možemo sada ostaviti praznim.

Na novokreiranom repozitorijuu za konfiguraciju, dovoljno je bilo kreirati odgovarajuće fajlove koji čuvaju konfiguraciju za pojedine klijente, pa je tako za mikroservise ModulDiskusije kreiran fajl `discussions-client.properties` sa sljedećim sadržajem:

```
server.port=8082
```

```
spring.jpa.database=POSTGRESQL  
spring.datasource.platform=postgres  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=update  
spring.database.driverClassName=org.postgresql.Driver  
spring.datasource.url=jdbc:postgresql://localhost:5432/nwtdiscussionsmodule  
spring.datasource.username=postgres  
spring.datasource.password=dbpass
```

Važno je uočiti da se za ovaj mikroservis na serveru rezervisao port 8082, te će preko ovog porta ostali mikroservisi moći komunicirati sa mikroservisom ModulDiskusije.