

Sadržaj

1. Spring Boot uvod
2. Uvlačenje postojećeg Maven projekta
3. Spring Boot pom.xml
4. Spring Boot projekat
5. Controllers
6. Case study – CRUD bioskop veb aplikacija

Dodatno:

1. Anotacija @SpringBootApplication
2. JavaBean klase u Spring radnom okviru
3. Spring MVC workflow
4. Controllers
5. Handler method supported return types
6. Handler method supported argument types



Osnove web programiranja

Spring Boot

Termin 2 i Termin 3 i Termin 4

Spring Boot uvod

Uvod

- Na predmetu nećete učiti čist Spring radni okvir već ćete učiti jednu njegovu verziju koja se zove Spring Boot
- Spring Boot (<https://spring.io/projects/spring-boot>) projekat je nastao sa idejom da se olakša kreiranje *stand-alone* Spring aplikacije koja jednostavno može da se pokrene.
 - Ideja je da se Spring aplikacija kreira za nekoliko minuta
 - Nekada se u Springu konfiguracija komponenti i propratnih biblioteka zadavala u njihovim konfiguracionim fajlovima. Problem: Pri kreiranju novih Spring komponenti ili izmeni logike rada postojeće neophodno je izmeniti konfiguracioni fajl te komponente.
 - Spring Boot se koristi kako bi se pojednostavila konfiguracija Spring projekata, veoma malo konfigurisanja, sve se oslanja na predefinisane konfiguracije.
 - Umesto XML konfiguracionih fajlova (pre), koriste se anotacije u Java klasama i njihovim navođenjem sa menja predefinisana konfiguracija projekta

Spring Boot uvod

Uvod

- Spring Boot

- Dovoljno je samo uvući biblioteku u projekat (to će se raditi preko Maven alata), i ta biblioteka će automatski da radi. Podrazumevana konfiguracija za biblioteke se po potrebi može menjati.
- Svim veb aplikacijma neophodan je veb kontejner u kome će se izvršavati tj. veb server na kome će se izvršavati. To može biti Tomcat, Jetty... Spring Boot omogućava da se taj kontejner zapakuje sam sa aplikacijom, tj. da se on ugradi u aplikaciju i da se aplikacija izveze kao izvršivi jar fajl.
 - Na računaru na kome postoji samo Java pokrene se jar arhiva. Sistem radi ostalo, iz jar archive se pokrene Tomcat, zatim se u Tomcat veb server deplojuje veb aplikacija, pa se pokrene Tomcat.
- dostupno da se aplikacija izveze i kao war fajl i da se ručno war fajl postavi unutar neke instalacije veb kontejnera kao što je Tomcat

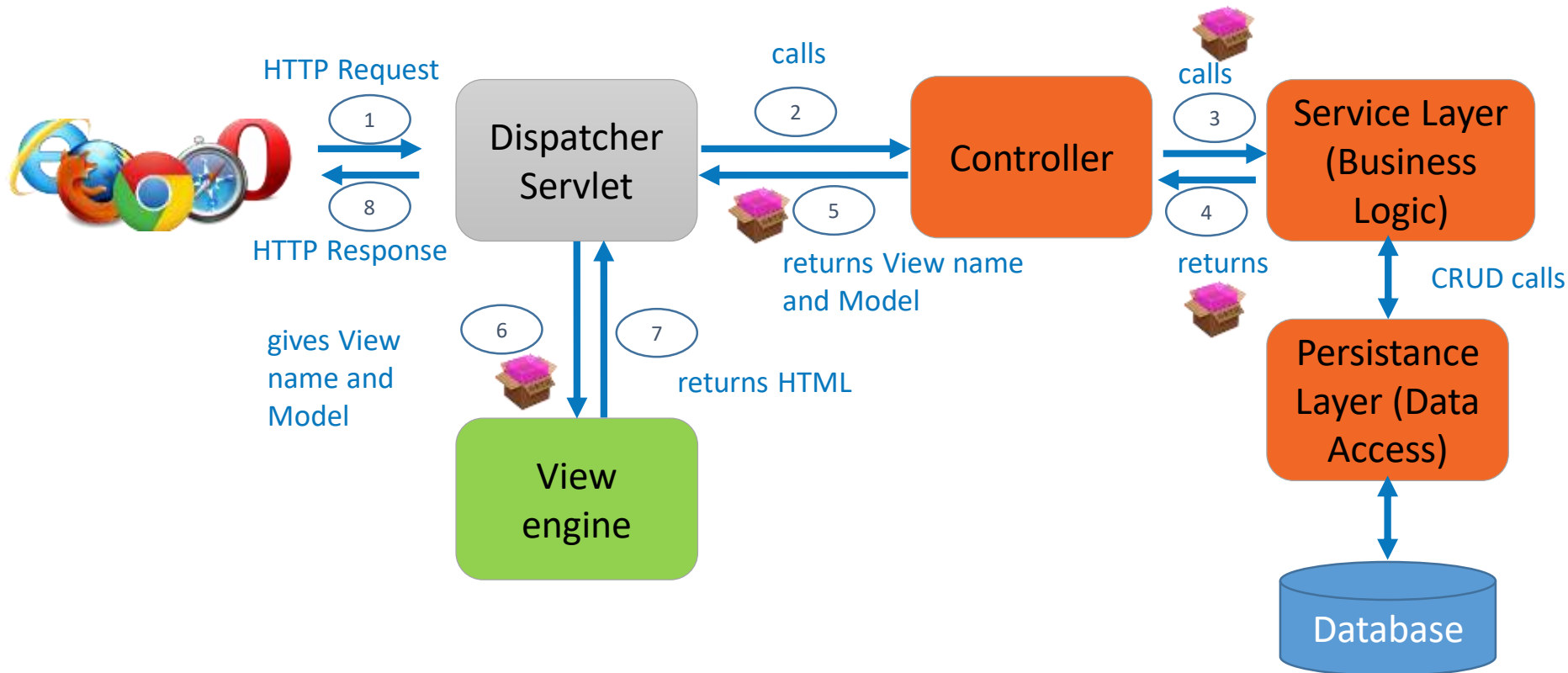
Spring Boot uvod

Karakteristike

- Spring Boot karakteristike
 - Jednostavnije se dobija konfigurisana Spring aplikacija
 - Jednostavnije pokretanje
 - Ugrađen veb server
 - Jednostavnije upravljanje paketima
 - Skup pripremljenih Maven artefakata
 - Konfiguriše Spring kontejner automatski gde god je moguće
- Ideja je da se programer fokusira inicijalno na razvoj aplikacije umesto na njen životni ciklus (konfiguraciju, postavljanje, upravljanje projektom, ...)
- Postoji klasa koja ima *main* metodu
 - Aplikacija se pokreće kao da je stand-alone aplikacija

Spring Boot uvod

Arhitektura Spring MVC aplikacije sa procesom rada– bez detalja



Spring Boot uvod

Arhitektura Spring MVC aplikacije sa procesom rada

- Uloga **DispatcherServlet** je da na osnovu HTTP zahteva i konfiguracije Spring aplikacije odluči tačno kom kontroleru će proslediti pristigli HTTP zahtev koji je zadužen za obradu tog zahteva. To odlučuje na osnovu URL.
 - DispatcherServlet ima ulogu Front Controller
- **Controller** su Java klase čije metode (**Handler Methods**) su zadužene za obradu različitih HTTP zahteva. Po potrebi metode kreiraju **model** i pozivaju metode servisnog sloja.
- **View engine** može biti npr. JSP, FreeMarker ili Thymeleaf template engine

Spring Boot uvod

Arhitektura Spring MVC aplikacije sa procesom rada

- **Servisni sloj** se koristi kao omotač (wrapper) DAO sloja i predstavlja opšteprihvaćen način za korišćenje DAO sloja. Radi sa **Modelom**.
 - U servisni sloj se stavljaju sve CRUD (create, retrieve, update, delete) metode koje perzistencioni sloj nudi
 - Može biti izostavljen tako što ćemo koristiti DAO sloj direktno, ali njegovo uvođenje može doneti mnoge pogodnosti:
 - **Zašto ne koristiti direktno perzistencioni sloj?** Zato što perzistencioni sloj nudi samo osnovne operacije. Ako želimo da proverimo polovnu logiku u tim podacima to nije moguće. Takve provere možemo vršiti u metodama u servisnom sloju.
 - **Preporuka je da se u Service sloju piše poslovna logika sistema**
- **Perzistencija podataka** u Spring aplikaciji se vrši putem **perzistentnog sloja**
 - Poznatiji pod nazivom DAO (Data Access Object) sloj
 - Ovaj sloj od nas „sakriva“ bazu podataka u obliku u kom je mi za sada poznajemo (skup tabela, odnosno relacija) i predstavlja je kao skup Java klasa.

Kreiranje novog Spring Boot projekta

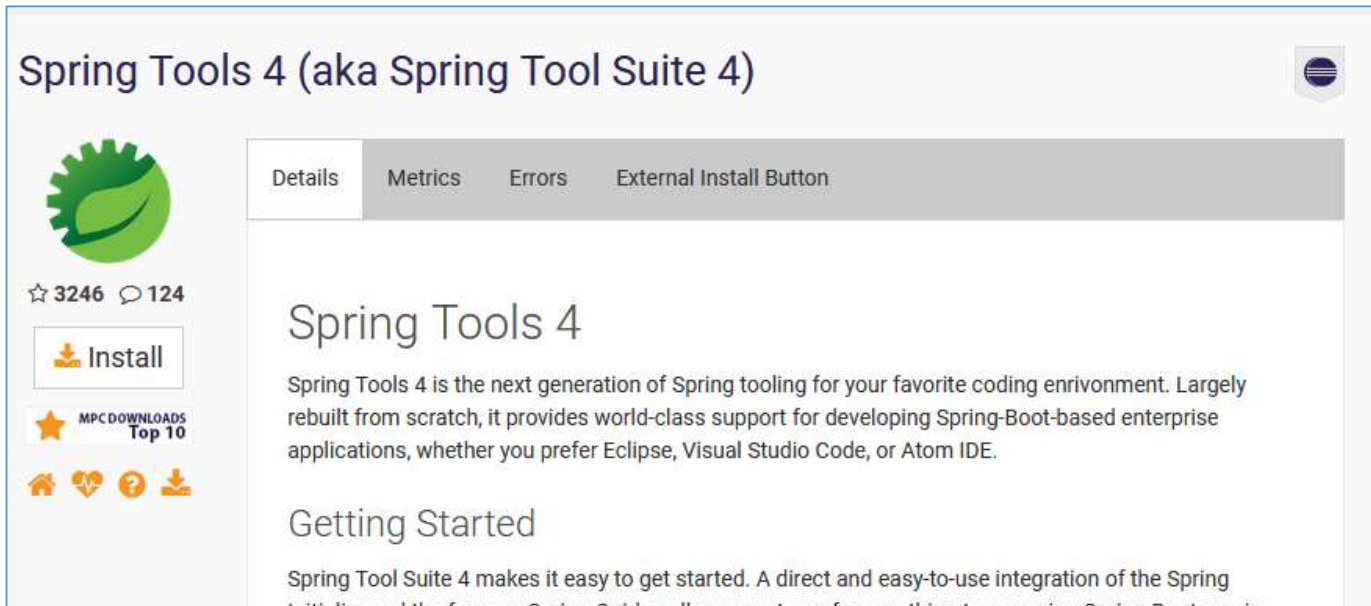
Kreiranje ručno kroz pom.xml

- Otići na Spring Boot sajt i pronaći konfiguraciju pom.xml fajla tako da ona podrži odgovarajući tip Spring Boot projekta (obično se može preuzeti kompletan pom.xml fajl iz dostupnih primera).
- <https://docs.spring.io/spring-boot/docs/1.1.4.RELEASE/reference/html/getting-started-first-application.html> (ili kucati u google “spring boot maven pom xml web” odabrati prvu stvar)
- <https://start.spring.io/>
- U pom.xml mora se navesti da je parent projekat SpringBoot projekat
- Koristimo wizard da odaberemo konfiguraciju
- Za web projekat trebalo bi dodati zavisnosti ka
 - Spring Web modulu
 - Veb serveru
 - Test bibliotekama
- Uvući prazan folder ImeProjekta, a u njemu foldere src i target i fajl pom.xml kao postojeći maven projekat

Kreiranje novog Spring Boot projekta

Kreiranje automatski kroz Eclipse plug-in

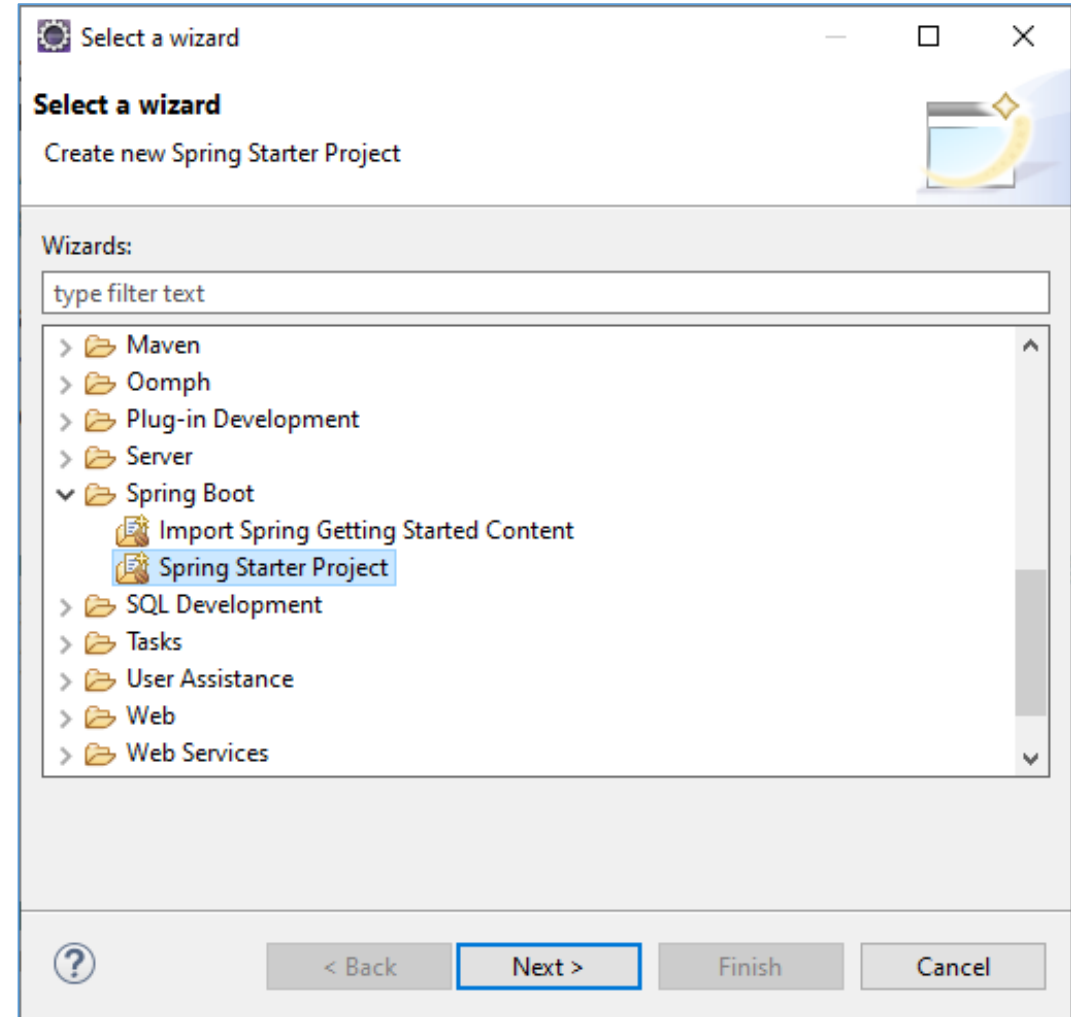
- Moguće je automatski definisati Spring Boot projekat ispočetka, gde će se kroz Eclipse plug-in definisati konfiguracija u pom.xml
- Koristiti **Spring Tools for Eclipse IDE**
<https://marketplace.eclipse.org/content/spring-tools-4-aka-spring-tool-suite-4>



Kreiranje novog Spring Boot projekta

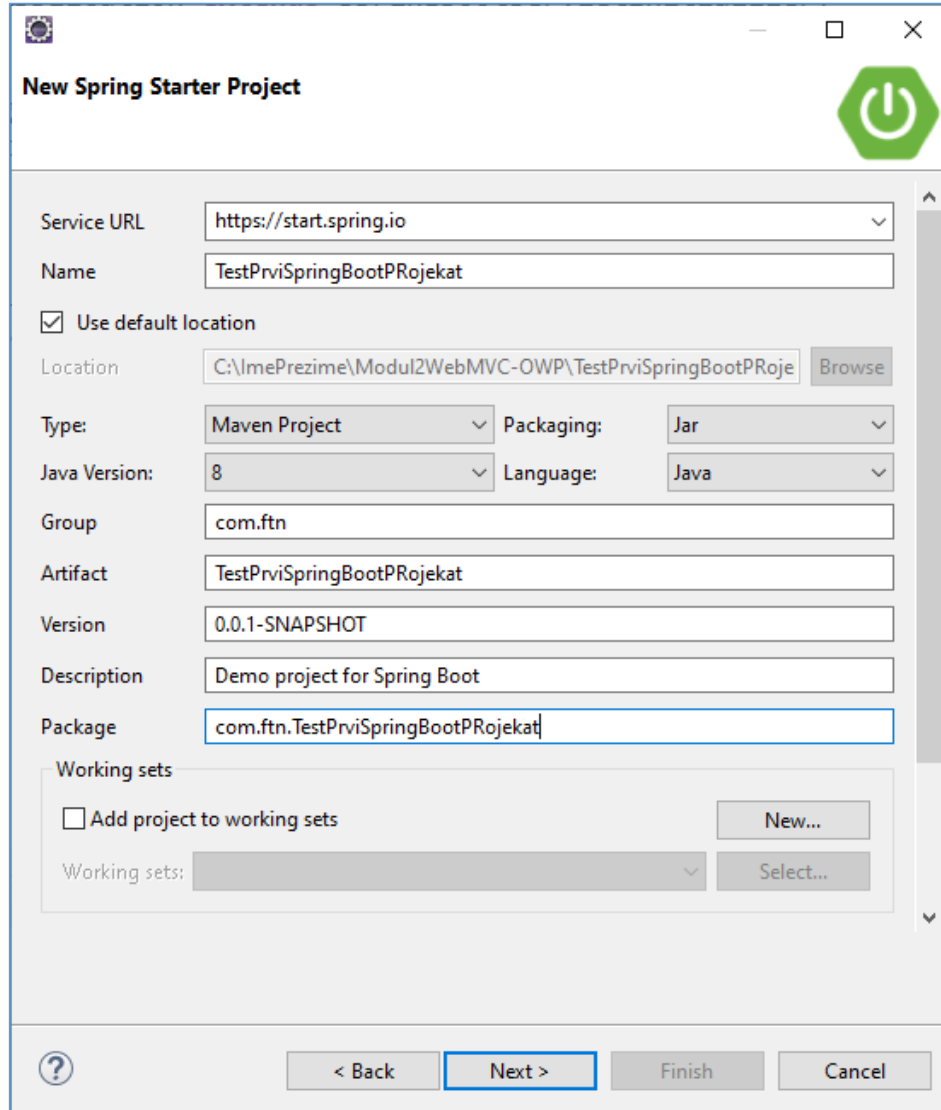
Kreiranje automatski kroz Eclipse plug-in

- Mogućnosti **Spring Tools**
- https://www.eclipse.org/community/eclipse_newsletter/2018/february/springboot.php
- <https://medium.com/danielpadua/java-spring-boot-eclipse-744454468670>



Kreiranje novog Spring Boot projekta

Kreiranje automatski kroz Eclipse plug-in



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

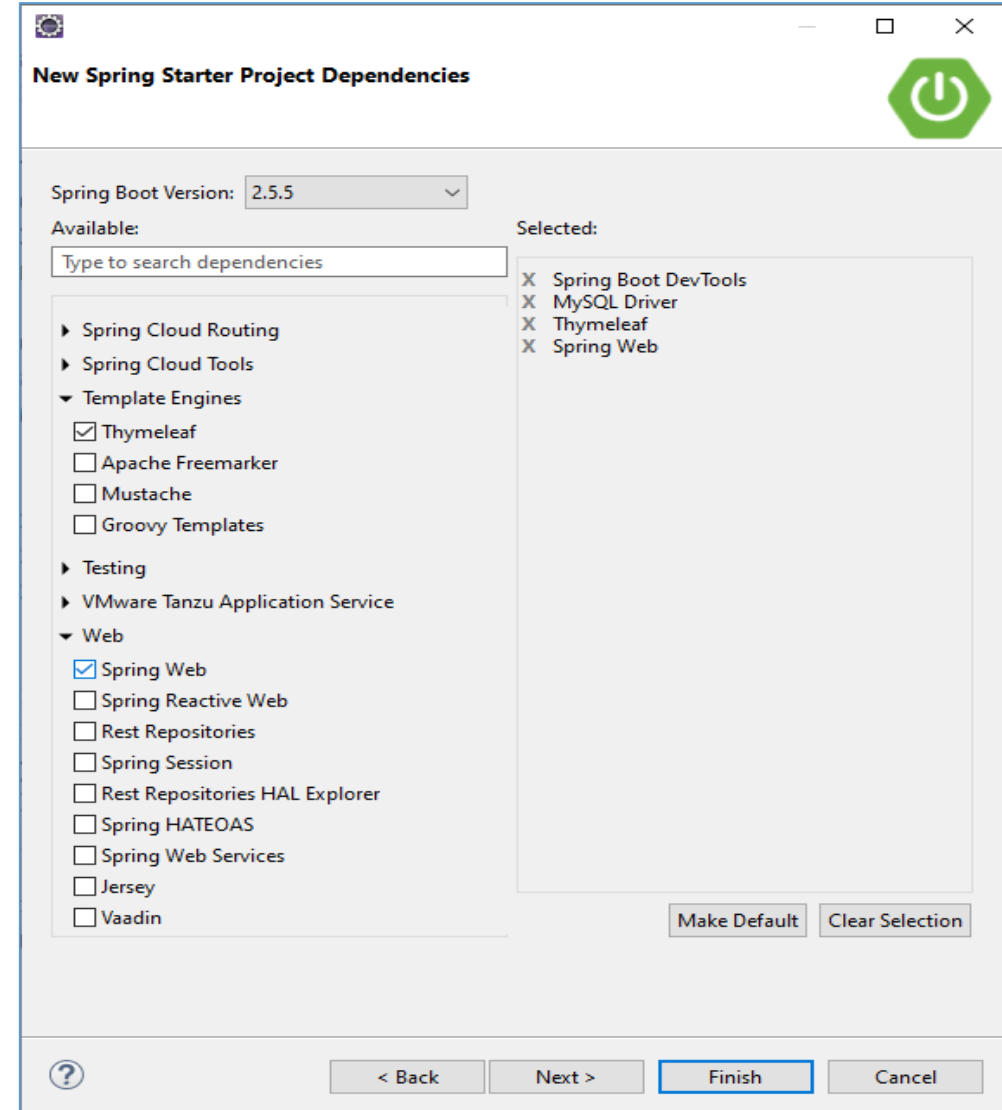
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



New Spring Starter Project Dependencies

Spring Boot Version:

Available:

Type to search dependencies

Selected:

- ☒ Spring Boot DevTools
- ☒ MySQL Driver
- ☒ Thymeleaf
- ☒ Spring Web

Spring Cloud Routing

Spring Cloud Tools

Template Engines

- ☒ Thymeleaf
- ☐ Apache Freemarker
- ☐ Mustache
- ☐ Groovy Templates

Testing

VMware Tanzu Application Service

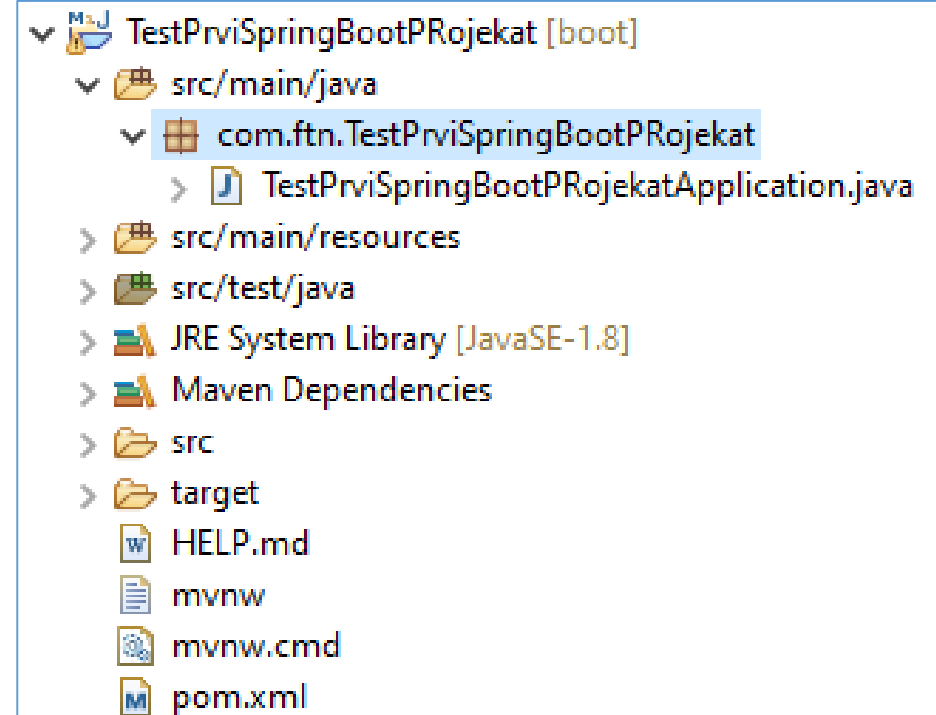
Web

- ☒ Spring Web
- ☐ Spring Reactive Web
- ☐ Rest Repositories
- ☐ Spring Session
- ☐ Rest Repositories HAL Explorer
- ☐ Spring HATEOAS
- ☐ Spring Web Services
- ☐ Jersey
- ☐ Vaadin

Kreiranje novog Spring Boot projekta

Kreiranje automatski kroz Eclipse plug-in

- Rezultat je iskonfigurisan SpringBoot projekat
- Ako nedostje nesto to opet treba dodati u pom.xml
- Mi ćemo kreirati SpringBoot projekat korišćenjem pom.xml fajla koji je kreiran ručno jer to ne zavisi od plug-ina u samom alatu.
- Ručno se može kreirati projekat bez obzira koji alat koristimo (Eclipse, NetBeans,...)



Uvlačenje postojećeg Maven projekta

Uvlačenje

- U okviru workspace ImePrezime/Predmet2Web raspakovati zip arhivu **ImportMavenVebProjekat.zip**. **Pogledajte sadržaj raspakovane arhive.**
- Pored standardnog importa projekta koji zahteva celokupan Eclipse projekat sa svim bibliotekama (loše jer veličina projekta može biti velika) dostupan je i **Maven importovanje projekta**
- Importovanje Maven projekta se razlikuje od standardnog importa projekta u Eclipse.
 - Nepohodno je samo posedovati foldere sa source kodom i resursima i pom.xml fajl.
- Iz *Java EE* perspective, klik na File->Import->Existing Maven Project, pa Next, odaberite folder ImePrezime/Predmet2Web u koji će se pretražiti za Maven projektima (može i više projekata od jednom da se importuje, projekti ne moraju da se nalaze u workspace Eclipse softvera). Klik na *Finish*.

Uvlačenje postojećeg Maven projekta

Uvlačenje

- Kada se pronađe pom.xml Maven će importovati porojekat, proći kroz pom.xml i prevući sve zavisnosti
- Za importovanje Maven projekta **ne trebaju nam dodatne stvari iz standardnog Eclipse projekta** (dodatni Eclipse fajlovi i folderi, kompajlirani fajlovi, preuzete bibliotekama, itd.).

Spring Boot pom.xml

Definisanje parent zavisnosti

- U `<project>` tagu novina je tag parent `<parent>`
- Tagom se definiše nasleđivanje artifakta
 - Podešavanja parent artifakta postaju i podešavanja nasleđenog artifakta i dependencies parent artifakta postaju dependencies nasleđenog artifakta.
- Sve Spring Boot aplikacije nasleđuju pomenuti parent artifact.

```
<parent>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-parent</artifactId>
```

```
    <version>2.2.5.RELEASE</version>
```

```
    <relativePath/>
```

```
</parent>
```


Spring Boot pom.xml

Definisanje projektne deliverable i verzije jave

- U fazi package kreira se projektna deliverablu kao jar arhiva sa `<packaging>jar</packaging>`
- U `<properties>` sekciji koristi se tag `<java.version>` koji je zamena za ranije definisane properti `maven.compiler.source` i `maven.compiler.target` tagove kojim se definiše verzija jave.

```
<packaging>jar</packaging>
```

```
<properties>
```

```
    <java.version>1.8</java.version>
```

```
</properties>
```

Spring Boot pom.xml

Obavezne zavisnosti

- Postoje 3 obavezne zavisnosti *spring-boot-starter-web*, *spring-boot-starter-tomcat* i *spring-boot-starter-test*.
- To su artifakti koji su napravljeni u Spring Boot sa ciljem da se prevuku svi moduli Spring i propratne konfiguracije koje su neophodne za izradu **Spring Boot veb projekta**.

Spring Boot pom.xml

Obavezne zavisnosti

- Artifakt *spring-boot-starter-web* je potreban za uključivanje Spring Web modula

```
<!-- uključivanje Spring Web modula -->
```

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>
```

Spring Boot pom.xml

Obavezne zavisnosti

- Artifakt *spring-boot-starter-tomcat* omogućuje da se Tomcat veb kontejner ugradi u izvršnu verziju aplikacije, ako postavimo da je pakovanje jar.
- Ako se pravi pakovanje war neophodno je navesti zavisnost za veb kontejner sa podešavanjem `<scope>provided</scope>`
- Može se koristiti i *Jetty* veb kontejner sa *spring-boot-starter-jetty*, ili *Undertow* sa *spring-boot-starter-undertow*

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-tomcat</artifactId>
```

```
    <scope>provided</scope>
```

```
</dependency>
```

Spring Boot pom.xml

Obavezne zavisnosti

- Artifakt *spring-boot-starter-test* omogućuje pribavljanje test okruženja, artifact će ujedno prevući i JUnit biblioteku

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-test</artifactId>
```

```
    <scope>test</scope>
```

```
</dependency>
```

Spring Boot pom.xml

Opcione zavisnosti

- Artifakt *spring-boot-starter-thymeleaf* je potreban za uključivanje *Thymeleaf* biblioteka. *Thymeleaf* je Java XML/XHTML/HTML5 templejt **View engine** koji se koristi za veb (servlet-based) i ne-veb oruženja

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
```

```
</dependency>
```

Spring Boot pom.xml

Opcione zavisnosti

- Artifakt *spring-boot-devtools* modul je poželjno koristiti tokom samog razvoja veb aplikacija jer omogućuje dodatne pogodnosti kao što su automatski restart, ne bi li razvoj aplikacije išao što lakše.
- Više o modulu na <https://www.baeldung.com/spring-boot-devtools>

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-devtools</artifactId>
```

```
    <scope>runtime</scope>
```

```
</dependency>
```

Spring Boot pom.xml

Opcione zavisnosti

- Artifakt *mysql-connector-java* omogućuje rad sa MySQL bazom podataka.
- Verzija se koristi jer je na računarima instalirana određena verzija MySQL baze

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>mysql-connector-java</artifactId>
```

```
    <version>5.1.6</version>
```

```
</dependency>
```


Spring Boot projekat

Pokretanje i konfiguracija

- Za pokretanje aplikacije neophodna je jedna Java klasa za koju će se zakačiti konfiguracija same aplikacije. Ta Java klasa će biti represent aplikacije.
- Klasa treba da nasledi *SpringBootServletInitializer* klasu i klasa treba da se anotira sa `@SpringBootApplication`
 - Kako se želi pokretati aplikacija preko jar arhive neophodna je i klasa sa main metodom, a ista klase će poslužiti u tu svrhu

Nastavak naredni termin

Spring Boot projekat

Pokretanje i konfiguracija

- Kreirana je klasa *TestMavenVebProjekatApplication* u rootu hijerahije paketa (obavezno ta lokacija) *com.ftn.ImportMavenVebProjekat*.
 - Klasa treba da nasledi `SpringBootServletInitializer`
 - Klasu je anotirana sa `@SpringBootApplication`
 - Klasa sadrži main metodu i kod za pokretanje

`@SpringBootApplication`

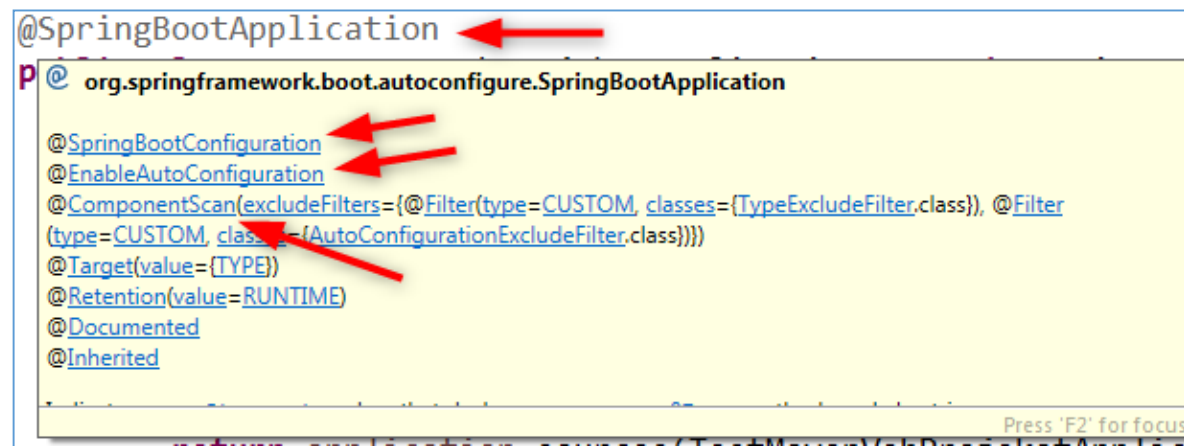
```
public class TestMavenVebProjekatApplication extends SpringBootServletInitializer
{

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

Spring Boot projekat

Anotacija @SpringBootApplication

- Povlači za sobom još tri anotacije @SpringBootConfiguration, @EnableAutoConfiguration i @ComponentScan
 - @SpringBootConfiguration – označava da se u *TestMavenWebApplication* klasi nalazi konfiguracija za Spring projekat
 - @EnableAutoConfiguration omogućava automatsku konfiguraciju *Spring Application Context*, pokušavajući da pogodi i konfiguriše binove, komponente i biblioteke koje aplikacija koristi
 - oslanja se na *pom.xml*
 - @ComponentScan govori Spring aplikaciji da prođe kroz kompletan projekat i za svaku Bean klasu da pročita njenu konfiguraciju iz Java koda.



```
@SpringBootApplication  
@org.springframework.boot.autoconfigure.SpringBootApplication  
  
@SpringBootConfiguration  
@EnableAutoConfiguration  
@ComponentScan(excludeFilters={@Filter(type=CUSTOM, classes={TypeExcludeFilter.class}), @Filter  
(type=CUSTOM, classes={AutoConfigurationExcludeFilter.class})})  
@Target(value={TYPE})  
@Retention(value=RUNTIME)  
@Documented  
@Inherited
```

Spring Boot projekat

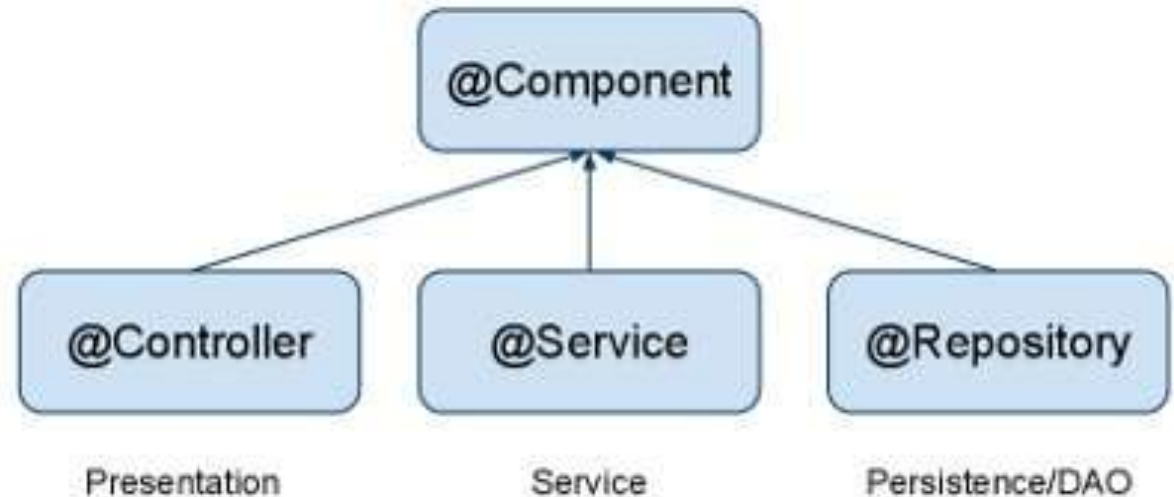
JavaBean klase u Spring radnom okviru

- Mnogi razvojni okviri zahtevaju da klase budu pisane uz poštovanje JavaBean standarda
- JavaBean je običan standard za pisanje klasa.
 - Standard nalaže da klasa mora zadovoljavati sledeće osobine da bi bila Bean:
 - Svi atributi klase moraju biti privatni (private) – koristiti getere i setere
 - Klasa mora imati javni (public) konstruktor bez parametara
 - Klasa implementira interfejs Serializable

Spring Boot projekat

JavaBean klase u Spring radnom okviru

- Ako želimo da klasa bude automatski pronađena i prepoznata od strane **Spring kontejnera** kao bean koja je komponenta Springa, treba je anotirati kao `@Component`. Umesto generičke anotacije `@Component` u praksi se koriste njene specijalizacije zavisno od uloge klase u aplikaciji.
- Hirerahija Spring komponenti
- auto-detect beans
- auto-configure beans
- implicit one-to-one mapping between the annotated class and the bean
- Svi bean su instancirani po Singleton paternu



Spring Boot projekat

Pokretanje projekta kao Java Aplikacije

- Metoda `SpringApplication.run` omogućuje pokretanje aplikacije kao obične Java aplikacije, desni klik pa *Run As-> Java Application*
- Klasa **SpringApplication** je deo Spring Boot
- Metoda `run` je statička metoda
- Prvim parametrom se navodi klasa koja predstavlja konfiguraciju projekta
 - Prethodno implicira da je moguće da se jedna Java klasa označi kao konfiguracija Spring projekta, a u drugoj Java klasi imati main metodu u kojoj se navodi pomenuti kod i poziva se prva Java klasa

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

Pokretanje kao Java Aplikacija

Spring Boot projekat

Pokretanje projekta kao Veb Aplikacije

- Neophodno je postaviti odgovarajuću vrednost **war** za **package tag** i **uključiti tomcat zavisnost sa podešavanjem `<scope>provided</scope>`**.
 - Promenjena je konfiguracija u pom.xml te treba promeniti i konfiguraciju samog projekta. Desni klik na projekat *Maven->Update Project*, pa dugme *Ok*.
- Takođe, neohodno je naznačiti da se Spring konfiguriše prilikom pokretanja aplikacije od strane veb kontejnera (konfiguracija se određuje u toku izvršavanja na osnovu samog veb kontejnera u kome će biti postavljena war arhiva).
 - U okviru klase PrviMavenVebProjekatApplication dodati metodu **configure**

```
@SpringBootApplication
```

```
public class TestMavenVebProjekatApplication extends SpringBootServletInitializer {
```

```
    @Override
```

```
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {  
        return application.sources(TestMavenVebProjekatApplication.class);  
    }
```

Pokretanje kao Veb Aplikacija

Spring Boot projekat

Pokretanje projekta kao Java Aplikacije

- Postaviti podešavanja u *pom.xml* tako da projektna deliverabla bude *jar* arhiva.
- Izvršiti ažuriranje konfiguracije projekta
- Cilj da se razvoj veb aplikacije olakša.

Controllers

Rad sa Kontrolerima

- U paketu *controller* se smeštaju svi kontroleri za Spring aplikacije.
- Sve klase koje će predstavljati kontrolere koji se koriste u aplikaciji anotiraju se sa `@Controller` ili sa `@RestController`.
 - Anotacija `@Controller` se koristi za defnisanje kontrolera kada je u pitanju klasična veb aplikacija po MVC arhitekturi.
 - Anotacija `@RestController` se koristi za defnisanje kontrolera kada je u pitanju klasična nova REST arhitektura veb aplikacije. Za klasu bi se definisao prefix *Api* da je u pitanju REST kontroler, nije neophodno ali je poželjno.

`@Controller`

```
public class ZdravoSveteController {...}
```

Controllers

Rad sa Kontrolerima

- Kontrolerima se dodaju dodatne anotacije za putanje na osnovu kojih se konfiguriše rad *DispatcherServlet*.
 - Anotacije se mogu postaviti za samu klasu ili metodu klase kontrolera.
- Kontroler će biti javno dostupan putem URL na osnovu definisane vrednosti u anotaciji `@RequestMapping`. `@RequestMapping` je anotacija koja se navodi za klasu i njom se definiše URL mapiranje za datu klasu.
 - `@RequestMapping` za klasu može da se izostavi i tada će kontroler biti dostupan preko ruta putanje tj. navođenjem `"/`.
 - Korišćenje: `@RequestMapping, @RequestMapping("/putanja"), @RequestMapping(value="/putanja")`

```
@Controller
```

```
@RequestMapping("/ZdravoSvete")
```

```
public class ZdravoSveteController {...}
```

Controllers

Rad sa Kontrolerima

- Otići na paket *com.ftn.TestMavenVebProjekat.controller*.
- U okviru controller paketa kreirati klasu *ZdravoSveteController*.
- Na kalasu *ZdravoSveteController* dodati anotacije `@Controller` i `@RequestMapping("/ZdravoSvete")`.

```
@Controller
```

```
@RequestMapping("/ZdravoSvete")
```

```
public class ZdravoSveteController {...}
```

Kontroler ZdravoSveteController

Controllers

Rad sa Kontrolerima

- U telu kontrolera definišu se javne metode koje trebaju da predstavljaju odgovor kontrolera za poziv odgovarajuće HTTP metode, te metode se nazivaju još i **Handler Methods**.
- Za svaku **Handler Method** treba da definiše ostatak URL koji se pridodaje na osnovni URL klase. Metoda je dostupna kao **adresa kontrolera + proširenje adrese navedeno u anotaciji**

Controllers

Rad sa Kontrolerima

- Metode se anotiraju sa: `@GetMapping`, `@PostMapping`, `@PutMapping`, `@DeleteMapping`, ili `@PatchMapping`, itd.
 - Korišćenje: `@GetMapping`, `@GetMapping("/putanja")`, `@GetMapping(value="/putanja")`.
 - Mapiranje višestrukih URL na istu metodu `@GetMapping(value={"/putanja1","/putanja2"})`
 - Preporuka je da naziv metoda započne sa skraćenicom za tip HTTP metode, pa zatim naziv implementirane funkcionalnosti, da bi se iz samog imena metode lakše razumelo šta metoda radi, npr. `getPrikaziSveFilmove`, `postDodajNoviFilm`,...
- Ako se koristi `@GetMapping` bez definisanja putanje to znači da je metoda dostupna preko URL definisanog za klasu.

```
@Controller
@RequestMapping("/ZdravoSvete")
public class ZdravoSveteController {

    @GetMapping
    public String getZdravo() {...}

}
```

Controllers

Rad sa Kontrolerima

- Povratni tip Handler Method metode za potrebe ovog predavanja definiše se kao **String** ili **void**. Više o tome u dodatnim materijalima.
- Kada se koristi povratni tip **String** tada se očekuje da se vrati ime prezentacije (view) koji se treba prikazati za pozvanu metodu.
 - Prezentacija može biti statička HTML stranica ili dinamički generisana uz mopoć nekog *Template Engine*.
 - Za vraćanje imena statičke stranice View Engine mora biti isključen.
- Svaka od metoda kontrolera koja vraća povratni tip String može biti dodatno anotirana sa `@ResponseBody`.
 - `@ResponseBody` naznačava se da će se u telu metode definisati telo HTTP odgovora (u našem slučaju povratni HTML), umesto da metoda vrati logičko ime prezentacije (view).

```
@Controller
```

```
@RequestMapping("/ZdravoSvete")
```

```
public class ZdravoSveteController {
```

```
    @GetMapping
```

```
    @ResponseBody
```

```
    public String getZdravo() {...}
```

Controllers

Rad sa Kontrolerima

- U klasi ZdravoSveteController postoji kreirana javna metoda *getZdravo* koja vraća **String** i koja je anotirana sa `@GetMapping` i `@ResponseBody`.
- U metodi *getZdravo* kreira se String reprezentacija HTML stanice koju treba da se vratiti kao povratnu vrednos te metode

Metode *getZdravo*

```
@Controller
@RequestMapping("/ZdravoSvete")
public class ZdravoSveteController {

    @GetMapping
    @ResponseBody
    public String getZdravo() {

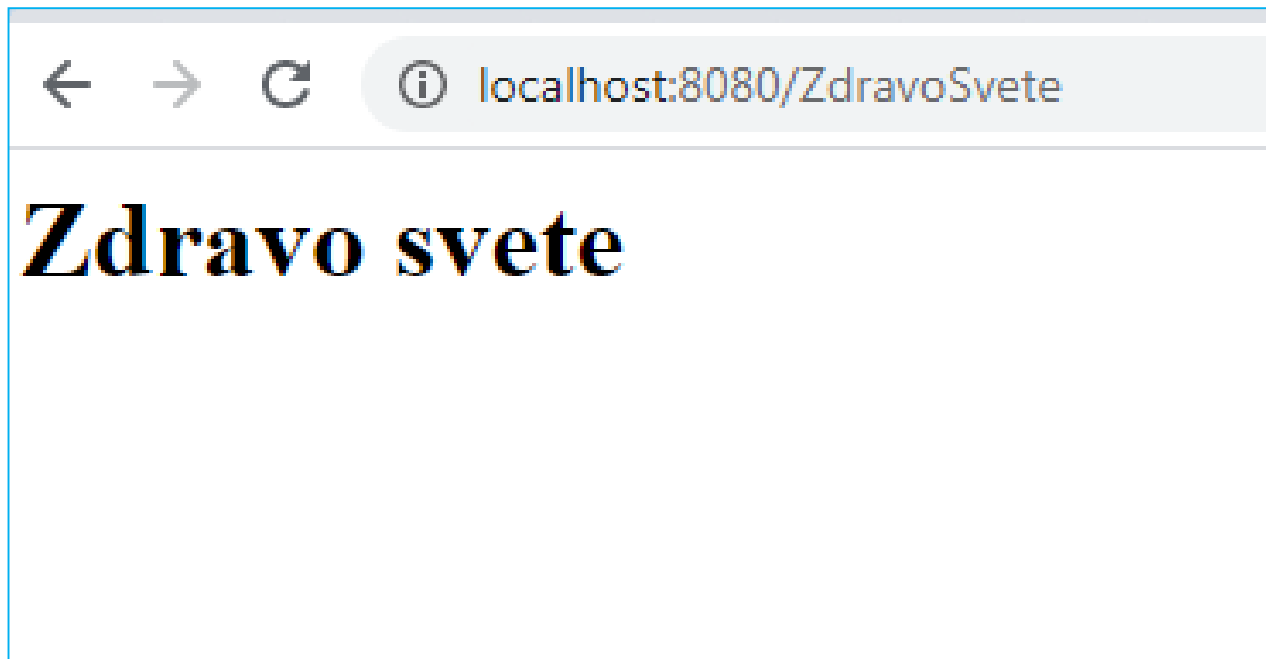
        String retHTML =
            "<html>\r\n" +
            "<head>\r\n" +
            "    <meta charset=\"UTF-8\">\r\n" +
            "    <title>Zdravo Svete</title>\r\n" +
            "</head>\r\n" +
            "<body>\r\n" +
            "    <h1>Zdravo svete</h1>\r\n" +
            "</body>\r\n" +
            "</html>";

        return retHTML;
    }
}
```

Controllers

Rad sa Kontrolerima

- Pokrenuti klasu TestMavenVebProjekatApplication kao običnu Java aplikaciju, desni klik pa *Run As-> Java Application*.
- U veb brauzeru unesite <http://localhost:8080/ZdravoSvete> dobićete prikaz



Controllers

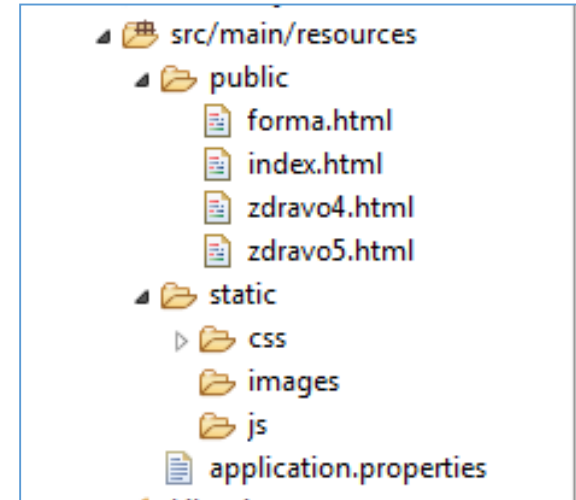
Uvlačenje projekta sa primerima

- U okviru workspace ImePrezime/Predmet2Web raspakovati zip arhivu **PrviMavenVebProjekat.zip**.
- **Uradite Maven importovanje projekta**
- Iz *Java EE* perspective, klik na File->Import->Existing Maven Project, pa Next, odaberite folder ImePrezime/Predmet2Web u koji će se pretražiti za Maven projektima (može i više projekata od jednom da se importuje, projekti ne moraju da se nalaze u workspace Eclipse softvera). Klik na *Finish*.
- Pokrenite projekat kao Java aplikaciju
- U veb brauzeru uneti <http://localhost:8080/PrviMavenVebProjekat/index.html> ili <http://localhost:8080/PrviMavenVebProjekat>
- Spring Boot će automatski prikazati index.html kao početnu stranicu ukoliko ona postoji kao javni resurs

Controllers

Uvlačenje projekta sa primerima

- Spring Boot je konfigurisan tako da prikazuje statičke sadržaje iz direktorijuma koji se nalaze u classpath
 - src/main/resources/static,
 - src/main/resources/public,
 - src/main/resources,
 - /META-INF/resources
- i iz korenskog direktorijuma za ServletContext
- Ne savetuje se korišćenje src/main/webapp foldera za smeštanje statičkog sadržaja ako je projektna deliverabla jar. Iako je direktorijum prihvaćeni standard, on se zapravo koristi samo kod pakovanja war, i ignorisan je od strane build alata ako se generiše jar.



Controllers

Dependency Injection (DI)

- Dependency Injection je softverski obrazac/princip koji se koristi kako bi se olakšalo instanciranje objekata.
- Osnovna ideja je da programer ne instancira samostalno objekte u kodu već da za instanciranje objekata prepusti radnom okviru u kome programira.
 - U kodu programer samo koristi promenljive a u njima će se “magično” naći objekti
- Dependency Injection je primer Inversion of Control principa

```
//GET: ZdravoSvete
@Controller
@RequestMapping(value="/ZdravoSvete")
public class ZdravoSveteController { injektuje

    //Specify the application URL ↙
    @Value("${server.servlet.contextPath}")
    private String contextPath;

    // GET: ZdravoSvete
    @GetMapping
    @ResponseBody
    public String getZdravo1() { koristi
        ↙
        String bURL = contextPath+"/";

        ...
    }
}
```

Controllers

Dependency Injection (DI)

- Postoje mnoge implementacije koje se mogu koristiti nezavisno od Spring-a.
- U okviru Spring Framework-a, DI se koristi kako bi se dobila referenca na neku komponentu
- Da bi Dependency Injection radio u Spring, potrebno je uključiti component-scan mehanizam. U Spring Boot, ovo podešavanje je deo anotacije `@SpringBootApplication` koja se već primenila za klasu koja predstavlja konfiguraciju Spring aplikacije.

Controllers

Dependency Injection (DI)

- **Spring kontejner** upravlja životnim ciklusom bean objekata
- Programer piše programski kod u kojem samo koristi objekte
- Spring kontejner je zadužen za kreiranje, inicijalizaciju, konfigurisanje i obezbeđivanje objekata dostupnim

```
@Controller
@RequestMapping(value="/PrihvatanjePodataka")
public class PrihvatanjePodatakaController {

    @Autowired ← injektuje
    private Environment env;

    // GET: PrihvatanjePodataka/preuzmi1
    @GetMapping(value="/preuzmi1")
    @ResponseBody
    public String preuzmi(HttpServletRequest request,
                          HttpServletResponse response) {
        //Specify the base URL for all relative URLs in a document
        String bURL = env.getProperty("server.servlet.contextPath") + "/";
        ...
    }
    ...
}
```

koristi

Controllers

Dependency Injection (DI)

- U našem slučaju se kao **Spring kontejner** koristi **Tomcat** i on će sam automatski inicijalizovati sve objekte anotirane sa `@Autowired`
- objekti anotirani sa `@Autowired` injektuju po kreiranju bean objekta za kontroler, pre poziva bilo koje od Handler metoda za kontroler.

```
@Controller
@RequestMapping(value="/PrihvatanjePodataka")
public class PrihvatanjePodatakaController {

    @Autowired ← injektuje
    private Environment env;

    // GET: PrihvatanjePodataka/preuzmi1
    @GetMapping(value="/preuzmi1")
    @ResponseBody
    public String preuzmi(HttpServletRequest request,
                          HttpServletResponse response) {
        //Specify the base URL for all relative URLs in a document
        String bURL = env.getProperty("server.servlet.contextPath") + "/";
        ...
    }
    ...
}
```

koristi

Controllers

Handler Method povratni tip i argumenti metode

- Povratni tip je String i metoda je anotirana sa `@ResponseBody`



ZdravoSveteController
getZdravo1()

```
//GET: ZdravoSvete
@Controller
@RequestMapping(value="/ZdravoSvete")
public class ZdravoSveteController {

    //Specify the application URL
    @Value("${server.servlet.contextPath}")
    private String contextPath;

    // GET: ZdravoSvete
    @GetMapping
    @ResponseBody
    public String getZdravo1() {

        //Specify the base URL for all relative URLs in a document
        String bURL = contextPath+"/";

        String retHTML =
            "<html>\r\n" +
            "<head>\r\n" +
            "    <meta charset=\"UTF-8\">\r\n" +
            "    <base href=\""+bURL+"\"> +
            "    <title>Zdravo Svete 1</title>\r\n" +
            "</head>\r\n" +
            "<body>\r\n" +
            "    <h1>Zdravo svete 1 - slovo ć</h1>\r\n" +
            "    <a href=\"index.html\">nazad</a>\r\n" +
            "</body>\r\n" +
            "</html>";

        return retHTML;
    }
}
```

Controllers

Handler Method povratni tip i argumenti metode

- Kada se koristi povratni tip **void** tada se očekuje da koristi **Servlet API** i parameter metode **HttpServletResponse response** i da se kroz njega unese HTML(kao kod Servleta što se radilo)
- Povratan tip je void i metoda koristi tipove argumenata **HttpServletRequest** i **HttpServletResponse**



```
// GET: ZdravoSvete/Zdravo2
// GET: ZdravoSvete/Zdravo2-DrugaPutanja
@GetMapping(value= {"/Zdravo2", "/Zdravo2-DrugaPutanja"})
public void getZdravo2(HttpServletRequest request, HttpServletResponse response) {
    try {
        //Specify the base URL for all relative URLs in a document
        String bURL = contextPath+"/";

        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out;
        out = response.getWriter();
        out.write(
            "<html>\r\n" +
            "<head>\r\n" +
            "    <meta charset=\"UTF-8\">\r\n" +
            "    <base href=\""+bURL+"\">\r\n" +
            "    <title>Zdravo Svete 2</title>\r\n" +
            "</head>\r\n" +
            "<body>\r\n" +
            "    <h1>Zdravo svete 2 - slovo ć</h1>\r\n" +
            "    <a href=\"/ZdravoSvete/Zdravo2-DrugaPutanja\">preko drugog URL</a><br/>\r\n" +
            "    <a href=\"/index.html\">nazad</a>\r\n" +
            "</body>\r\n" +
            "</html>");
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

ZdravoSveteController
getZdravo2()

Controllers

Handler Method povratni tip i argumenti metode

- Povratan tip je void i metoda koristi tipove argumenata argumenata Reader i Writer (od HttpServletRequest i HttpServletResponse)

Zdravo svete 3 - slovo ?

[nazad](#)

ZdravoSveteController
getZdravo3()

```
// GET: ZdravoSvete/Zdravo3
@GetMapping(value="/Zdravo3")
public void getZdravo3(Reader in, Writer out) {
    //Specify the base URL for all relative URLs in a document
    String bURL = contextPath+"/";

    try {
        out.write(
            "<html>\r\n" +
            "<head>\r\n" +
            "    <meta charset=\"UTF-8\">\r\n" +
            "    <base href=\""+bURL+"\"> +
            "    <title>Zdravo Svete 3</title>\r\n" +
            "</head>\r\n" +
            "<body>\r\n" +
            "    <h1>Zdravo svete 3 - slovo ć</h1>\r\n" +
            "    <a href=\"index.html\">nazad</a>\r\n" +
            "</body>\r\n" +
            "</html>");
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```



Controllers

Handler Method povratni tip i argumenti metode

- Povratan tip je void i metoda koristi redirekciju na statički resurs oslanjajući se na sendRedirect od HttpServletResponse
- Javni statički resurs zdravo4.html se nalazi u folderu src/main/resources/public

```
// GET: ZdravoSvete/Zdravo4
@GetMapping(value="/Zdravo4")
public void getZdravo4(HttpServletRequest request, HttpServletResponse response) {
    try {
        response.sendRedirect(contextPath+"/"+zdravo4.html);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```



ZdravoSveteController
getZdravo4()

Controllers

Handler Method povratni tip i argumenti metode

- Povratan tip je String i vraća naziv statičkog resurs "/zdravo5.html", da bi ovo bilo dozvoljeno View Engine mora biti isključen
- Javni statički resurs zdravo5.html se nalazi u folderu src/main/resources/public

```
//      When using String as a method return type the result may be name of view
template
//      da bi vraćao statičke HTML stranice isključujemo thymeleaf template
//      u fajlu application.properties
// GET: ZdravoSvete/Zdravo5
@RequestMapping(value="/Zdravo5")
public String getZdravo5() {
    //očekuje relativni url
    return (".."+"zdravo5.html");
    //očekuje apsolutni url, već uključeno ime aplikacije
//      return ("/"+"zdravo5.html");
}
```

ZdravoSveteController
getZdravo5()



Controllers

Prihvatanje podataka iz forme

- Prihvatanje podataka iz forme može se uraditi oslonivši se na Servlet API tj. koristeći HttpServletRequest request

```
@Controller
@RequestMapping(value="/PrihvatanjePodataka")
public class PrihvatanjePodatakaController {

    @Autowired
    private Environment env;

    // GET: PrihvatanjePodataka/preuzmi1
    @GetMapping(value="/preuzmi1")
    @ResponseBody
    public String preuzmi(HttpServletRequest request, HttpServletResponse response) {
        //Specify the base URL for all relative URLs in a document
        String bURL = env.getProperty("server.servlet.contextPath")+"/";

        StringBuilder retval = new StringBuilder();
        retval.append(
            "<html>\r\n" +
            "<head>\r\n" +
            "    <meta charset=\"UTF-8\">\r\n" +
            "    <base href=\""+bURL+"\"> +
            "    <title>Prihvatanje podataka</title>\r\n" +
            "</head>\r\n" +
            "<body>\r\n" +
            "    <h1>Prihvatanje podataka koristeći HttpServletRequest</h1>\r\n" +
            "    <p>Uneli ste ime <b>"+request.getParameter("ime")+"</b> i prezime"
            "    <a href=\"forma.html\">nazad</a>\r\n" +
            "</body>\r\n" +
            "</html>");
        return retval.toString();
    }
}
```

Prihvatanje podataka koristeći HttpServletRequest

Ime : Prezime :

Prihvatanje podataka koristeći HttpServletRequest

```
<br />
<br />
<form action="/PrihvatanjePodataka/preuzmi1" method="get"
    accept-charset="UTF-8">
    Ime :<input type="text" name="ime" /> Prezime :<input type="text"
        name="prezime" /> <br /> <input type="submit" value="Posalji" />
</form>
```

← → ↻ ⓘ localhost:8080/PrihvatanjePodataka/preuzmi1

Prihvatanje podataka korišćenjem HttpServletRequest

Uneli ste ime **Petar** i prezime **Petrović**

[nazad](#)

PrihvatanjePodatakaController preuzmi() i forma.html

Controllers

Prihvatanje podataka iz forme

- Prihvatanje podataka koristeći `@RequestParam` anotaciju argumenata Handler Metode
- `@RequestParam` se koristi za ekstrahovanje parametara upita, parametara iz formi i fajlova iz zahteva.
- Ime argumenta metode mora da se poklapa sa imenom ulaznog parametara forme
- Moguća je automatska konverzija parametara u primitivni tip ili objekat Wrapper klase

```
<form action="/PrihvatanjePodataka/preuzmi2" method="get"
      accept-charset="UTF-8">
  Ime :<input type="text" name="ime" /> Prezime :<input type="text"
      name="prezime" /> <br /> <input type="submit" value="Posalji" />
</form>
```

```
// GET: PrihvatanjePodataka/preuzmi2
@GetMapping(value="/preuzmi2")
@ResponseBody
public String preuzmi2(@RequestParam String ime,
                      @RequestParam String prezime) {
    //Specify the base URL for all relative URLs in a document
    String bURL = env.getProperty("server.servlet.contextPath")+"/";

    StringBuilder retVal = new StringBuilder();
    retVal.append(
        "<html>\r\n" +
        "<head>\r\n" +
        "  <meta charset=\"UTF-8\">\r\n" +
        "  <base href=\""+bURL+"\"> +
        "  <title>Prihvatanje podataka</title>\r\n" +
        "</head>\r\n" +
        "<body>\r\n" +
        "  <h1>Prihvatanje podataka koristeći @RequestParam ano
        "  <p>Uneli ste ime <b>"+ime+"</b> i prezime <b>"+prezim
        "  <a href=\"forma.html\">nazad</a>\r\n" +
        "</body>\r\n" +
        "</html>");
    return retVal.toString();
}
```

Controllers

Prihvatanje podataka iz forme

- Prihvatanje podataka koristeći `@RequestParam` anotaciju argumenata Handler Metode metode

```
// GET: PrihvatanjePodataka/preuzmi2
@GetMapping(value="/preuzmi2")
@ResponseBody
public String preuzmi2(@RequestParam String ime,
    @RequestParam String prezime) {
    //Specify the base URL for all relative URLs in a document
    String bURL = env.getProperty("server.servlet.contextPath")+"/";

    StringBuilder retval = new StringBuilder();
    retval.append(
        "<html>\r\n" +
        "<head>\r\n" +
        "    <meta charset=\"UTF-8\">\r\n" +
        "    <base href=\""+bURL+"\"> +
        "    <title>Prihvatanje podataka</title>\r\n" +
        "</head>\r\n" +
        "<body>\r\n" +
        "    <h1>Prihvatanje podataka koristeći @RequestParam anota
        "    <p>Uneli ste ime <b>"+ime+"</b> i prezime <b>"+prezime
        "    <a href=\""+bURL+"forma.html\">nazad</a>\r\n" +
        "</body>\r\n" +
        "</html>");
    return retval.toString();
}
```

Prihvatanje podataka koristeći `@RequestParam` anotaciju argumenata Handler Metode Metode

Ime : Prezime :

[nazad](#)

← → ↻ ⓘ localhost:8080/PrihvatanjePodataka/preuzmi2

Prihvatanje podataka koristeći `@R`

Uneli ste ime **Marko** i prezime **Marković**

[nazad](#)

PrihvatanjePodatakaController preuzmi2() i forma.html

Controllers

Prihvatanje podataka iz forme

- Ukoliko se argument metode anotira sa `@RequestParam` a ne postoji odgovarajući parametar upita ili parametar forme na koji bi se on mapirao iskočiće greška

PrihvatanjePodatakaController preuzmi2(), forma.html i greška

```
<form action="/PrihvatanjePodataka/preuzmi2" method="get"
accept-charset="UTF-8">
  Ime :<input type="text" name="ime" />
  <input type="submit" value="Posalji" />
</form>
```

```
// GET: PrihvatanjePodataka/preuzmi2
@GetMapping(value="/preuzmi2")
@ResponseBody
public String preuzmi2(@RequestParam String ime,
    @RequestParam String prezime) {
    //Specify the base URL for all relative URLs in a document
    String bURL = env.getProperty("server.servlet.contextPath")+"/";

    StringBuilder retval = new StringBuilder();
    retval.append(
        "<html>\r\n" +
        "<head>\r\n" +
        "    <meta charset=\"UTF-8\">\r\n" +
        "    <base href=\""+bURL+"\">\r\n" +
        "    <title>Prihvatanje podataka</title>\r\n" +
        "</head>\r\n" +
        "<body>\r\n" +
        "    <h1>Prihvatanje podataka koristeći @RequestParam anonični parametar</h1>\r\n" +
        "    <p>Uneli ste ime <b>"+ime+"</b> i prezime <b>"+prezime+"</b>\r\n" +
        "    <a href=\""+bURL+"forma.html\">nazad</a>\r\n" +
        "</body>\r\n" +
        "</html>");
    return retval.toString();
}
```

localhost:8080/PrihvatanjePodataka/preuzmi2

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon May 04 18:32:57 CEST 2020

There was an unexpected error (type=Bad Request, status=400).

Required String parameter 'prezime' is not present

org.springframework.web.bind.MissingServletRequestParameterException: Required String parameter 'prezime' is not present
at org.springframework.web.method.annotation.RequestParamMethodArgumentResolver.handleMissingValue(RequestParamMethodArgumentResolver.java:147)
at org.springframework.web.method.annotation.AbstractNamedValueMethodArgumentResolver.resolveArgument(AbstractNamedValueMethodArgumentResolver.java:114)
at org.springframework.web.method.support.HandlerMethodArgumentResolverComposite.resolveArgument(HandlerMethodArgumentResolverComposite.java:121)
at org.springframework.web.method.support.InvocableHandlerMethod.getMethodArgumentValues(InvocableHandlerMethod.java:168)
at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:134)

Controllers

Prihvatanje podataka iz forme

- `@RequestParam` anotacija može biti dodatno opisana atributima **name**, **value**, **required** i **defaultValue**
- Ukoliko se navede **name** onda se traži parametar forme po imenu, a njegova vrednost se smešta u varijablu koja je iza anotacije
 - Alternativa za **name** je **value** atribut `@RequestParam(value="ime")` ili skraćena anotacija `@RequestParam("ime")`
- `@RequestParam` može biti i opcioni ukoliko se navede da je **required=false**
 - U tom slučaju treba voditi računa da taj parametar može imati i **null** vrednost ukoliko nije prosleđen
- Varijanta opcione vrednosti `@RequestParam` za koji se može zadati predefinisana vrednost (ako se ne prosledi) moguće je sa atributom **defaultValue**

Controllers

Prihvatanje podataka iz forme

- Korišćenje atributa **name**

```
Prihvatanje podataka koristeći @RequestParam anotaciju sa dodatnim opisom atributa name
<br />
<br />
<form action="/PrihvatanjePodataka/preuzmi3" method="get"
    accept-charset="UTF-8">
    Ime :<input type="text" name="ime" />
    Prezime :<input type="text" name="prezime" /> <br />
    Pol:<input type="radio" name="pol" value="muški" checked/>
        M<input type="radio" name="pol" value="ženski"/>Ž<br />
    Plata:<input type="number" name="plata" min="1" value="1"/> <br />
    <input type="submit" value="Posalji" />
</form>
<hr />
```

```
// GET: PrihvatanjePodataka/preuzmi3
@GetMapping(value="/preuzmi3")
@ResponseBody
public String preuzmi3(@RequestParam(name="ime") String firstName,
    @RequestParam(name="prezime" ) String lastName,
    @RequestParam(name="pol", required = false ) String gender,
    @RequestParam(name="plata", defaultValue="0") Double salary) {

    if(gender==null)
        gender="neizjašnjen";

    //Specify the base URL for all relative URLs in a document
    String bURL = env.getProperty("server.servlet.contextPath")+"/";

    StringBuilder retval = new StringBuilder();
    retval.append(
        "<html>\r\n" +
        "<head>\r\n" +
        "    <meta charset=\"UTF-8\">\r\n" +
        "    <base href=\""+bURL+"\"> +
        "    <title>Prihvatanje podataka</title>\r\n" +
        "</head>\r\n" +
        "<body>\r\n" +
        "    <h1>Prihvatanje podataka koristeći @RequestParam anotaci
        "    <p>Uneli ste ime <b>"+firstName+"</b> i prezime <b>"+las
        "    <p>Uneli ste pol <b>"+gender+"</b></p>\r\n" +
        "    <p>Uneli ste platu <b>"+salary+"</b></p>\r\n" +
        "    <a href=\"forma.html\">nazad</a>\r\n" +
        "</body>\r\n" +
        "</html>");

    return retval.toString();
}
```

Controllers

Prihvatanje podataka iz forme

- Korišćenje atributa **name**

Prihvatanje podataka koristeći @RequestParam anotaciju sa dodatnim

Ime : Prezime :

Pol: ☒ M ☐ Ž

Plata:

Prihvatanje podataka

Uneli ste ime **Nikola** i prezime **Nikolić**

Uneli ste pol **muški**

Uneli ste platu **100.0**

[nazad](#)

PrihvatanjePodatakaController preuzmi3() i forma.html scenario 1

```
// GET: PrihvatanjePodataka/preuzmi3
@GetMapping(value="/preuzmi3")
@ResponseBody
public String preuzmi3(@RequestParam(name="ime") String firstName,
    @RequestParam(name="prezime" ) String lastName,
    @RequestParam(name="pol", required = false ) String gender,
    @RequestParam(name="plata", defaultValue="0") Double salary) {

    if(gender==null)
        gender="neizjašnjen";

    //Specify the base URL for all relative URLs in a document
    String bURL = env.getProperty("server.servlet.contextPath")+"/";

    StringBuilder retval = new StringBuilder();
    retval.append(
        "<html>\r\n" +
        "<head>\r\n" +
        "    <meta charset=\"UTF-8\">\r\n" +
        "    <base href=\""+bURL+"\"> +
        "    <title>Prihvatanje podataka</title>\r\n" +
        "</head>\r\n" +
        "<body>\r\n" +
        "    <h1>Prihvatanje podataka koristeći @RequestParam anotaci
        "    <p>Uneli ste ime <b>"+firstName+"</b> i prezime <b>"+las
        "    <p>Uneli ste pol <b>"+gender+"</b></p>\r\n" +
        "    <p>Uneli ste platu <b>"+salary+"</b></p>\r\n" +
        "    <a href=\"forma.html\">nazad</a>\r\n" +
        "</body>\r\n" +
        "</html>");

    return retval.toString();
}
```

Controllers

Prihvatanje podataka iz forme

- Korišćenje atributa **required** i **defaultValue**

Prihvatanje podataka koristeći `@RequestParam` anotaciju sa dodatnim opisom atributa `name`, `required`, `defaultValue`

Ime : Prezime :

Prihvatanje podataka koristeći `@RequestParam` anotaciju sa dodatnim opisom atributa

Uneli ste ime Steva i prezime Stević

Uneli ste pol neizjašnjen

Uneli ste platu 0.0

[nazad](#)

**PrihvatanjePodatakaController preuzmi3() i
forma.html scenario 2**

```
// GET: PrihvatanjePodataka/preuzmi3
@GetMapping(value="/preuzmi3")
@ResponseBody
public String preuzmi3(@RequestParam(name="ime") String firstName,
    @RequestParam(name="prezime" ) String lastName,
    @RequestParam(name="pol", required = false ) String gender,
    @RequestParam(name="plata", defaultValue="0") Double salary) {

    if(gender==null)
        gender="neizjašnjen";

    //Specify the base URL for all relative URLs in a document
    String bURL = env.getProperty("server.servlet.contextPath")+"/";

    StringBuilder retval = new StringBuilder();
    retval.append(
        "<html>\r\n" +
        "<head>\r\n" +
        "    <meta charset=\"UTF-8\">\r\n" +
        "    <base href=\""+bURL+"\"> +
        "    <title>Prihvatanje podataka</title>\r\n" +
        "</head>\r\n" +
        "<body>\r\n" +
        "    <h1>Prihvatanje podataka koristeći @RequestParam anotaci
        "    <p>Uneli ste ime <b>"+firstName+"</b> i prezime <b>"+las
        "    <p>Uneli ste pol <b>"+gender+"</b></p>\r\n" +
        "    <p>Uneli ste platu <b>"+salary+"</b></p>\r\n" +
        "    <a href=\"forma.html\">nazad</a>\r\n" +
        "</body>\r\n" +
        "</html>");

    return retval.toString();
}
```

Controllers

Prihvatanje podataka iz forme

- `@RequestParam` anotacija se može koristiti i za preuzimanje svih parametara forme. Mapiranje svih parametara forme moguće je korišćenjem pomenute notacije i argumenta metode tipa **`Map<String,String>`**.
 - Dobra strana je da je moguće odjednom preuzeti sve parametre, bez potrebe da se navode jedan po jedan.
 - Loša strana svi parametri su tipa `String` i nema automatske konverzije

Controllers

Prihvatanje podataka iz forme

- preuzimanje svih parametra, `@RequestParam` anotacija i argument metode tipa `Map<String,String>`

```
<form action="/PrihvatanjePodataka/preuzmi4" method="get"
  accept-charset="UTF-8">
  Ime :<input type="text" name="ime" />
  Prezime :<input type="text" name="prezime" /> <br />
  <input type="submit" value="Posalji" />
</form>
```

PrihvatanjePodatakaController preuzmi4() i forma.html

```
// GET: PrihvatanjePodataka/preuzmi4
@GetMapping(value="/preuzmi4")
@ResponseBody
public String preuzmi4(@RequestParam Map<String,String> allParameters) {

    String firstName=allParameters.get("ime");
    String lastName=allParameters.get("prezime");
    String gender=allParameters.get("pol")==null?"neizjašnjen": allParameters.get("pol");
    String salary=allParameters.get("plata")==null?"0": allParameters.get("plata");

    //Specify the base URL for all relative URLs in a document
    String bURL = env.getProperty("server.servlet.contextPath")+"/";

    StringBuilder retval = new StringBuilder();
    retval.append(
        "<html>\r\n" +
        "<head>\r\n" +
        "    <meta charset=\"UTF-8\">\r\n" +
        "    <base href=\"\"+bURL+"\">\" +
        "    <title>Prihvatanje podataka</title>\r\n" +
        "</head>\r\n" +
        "<body>\r\n" +
        "    <h1>Prihvatanje podataka koristeći @RequestParam anotaciju i ar
        "    <p>Uneli ste ime <b>"+firstName+"</b> i prezime <b>"+lastName+"</b></p>\r\n" +
        "    <p>Uneli ste pol <b>"+gender+"</b></p>\r\n" +
        "    <p>Uneli ste platu <b>"+salary+"</b></p>\r\n" +
        "    <a href=\"/forma.html\">nazad</a>\r\n" +
        "</body>\r\n" +
        "</html>");
    return retval.toString();
}
```

Controllers

Prihvatanje podataka iz forme

- `@ModelAttribute` anotacija se može koristiti za preuzimanje svih parametara forme pri čemu se oni automatski povezuju sa klasom iz sloja model koja je argument metode i atributima navedene klase.
 - Mapiranjem se kreira objekat klase.
 - Automatski se povezuju imena parametara forme sa nazivima atributima klase.
 - Ukoliko ne postoji parametar forme koji bi odgovarao nazivu atributa klase, atribut klase dobiće podrazumevanu vrenost za tip atributa
 - Ukoliko neki parametar forme ne može da se mapira na atribut klase mapiranje će se preskočiti

Controllers

Prihvatanje podataka iz forme

- preuzimanje svih parametra, `@ModelAttribute` anotacija i argument metode tipa klasa iz sloja Model.

```
public String preuzmi5(@ModelAttribute Korisnik korisnik, BindingResult result) {
    String error="";
    if (result.hasErrors()) {
        error = "greška prilikom preuzimanja podataka";
    }
    //Specify the base URL for all relative URLs in a document
    String bURL = env.getProperty("server.servlet.contextPath")+"/";
    StringBuilder retval = new StringBuilder();
    retval.append(
        "<html>\r\n" +
        "<head>\r\n" +
        "    <meta charset=\"UTF-8\">\r\n" +
        "    <base href=\""+bURL+"\"> +
        "    <title>Prihvatanje podataka</title>\r\n" +
        "</head>\r\n" +
        "<body>\r\n" +
        "    <h1>Prihvatanje podataka koristeći @ModelAttribute anotaciju sa proizvol
    if(result.hasErrors())
        retval.append("<p>"+error+"</b>");
    else
        retval.append(
            "<p>Uneli ste ime <b>"+korisnik.getIme()+"</b> i prezime <b>"+korisnik.getPrezime()
            "<p>Uneli ste pol <b>"+korisnik.getPol()+"</b></p>\r\n" +
            "<p>Uneli ste platu <b>"+korisnik.getPlata()+"</b></p>\r\n");

    retval.append(
        "    <a href=\"forma.html\">nazad</a>\r\n" +
        "</body>\r\n" +
        "</html>");
    return retval.toString();
}
```

```
<form action="/PrihvatanjePodataka/preuzmi5" method="get"
    accept-charset="UTF-8">
    Ime :<input type="text" name="ime" />
    Prezime :<input type="text" name="prezime" /> <br />
    Pol:<input type="radio" name="pol" value="muški" checked/>
        M<input type="radio" name="pol" value="ženski"/>Ž<br />
    Plata:<input type="number" name="plata" min="1" value="1"/> <br />
    Visak:<input type="text" name="visak" value="text"/> <br />
    <input type="submit" value="Posalji" />
</form>
```

```
public class Korisnik {

    String ime, prezime, pol;
    double plata;
```

PrihvatanjePodatakaController preuzmi5() i forma.html

Controllers

Inicijalizacija kontrolera

- Anotacijom `@PostConstruct` omogućeno je da se metoda izvrši po kreiranju objekta kontrolera pre nego što se on počne koristiti.
- Metoda **`init()`** anotirana sa `@PostConstruct` koristiće se za inicijalizaciju kontrolera
- U okviru metode moguće je npr. izvršiti popunjavanje objekata u `ServletContext`

```
@PostConstruct
public void init() {
    //Specify the base URL for all relative URLs in a document
    bURL = servletContext.getContextPath()+"/";

    Filmovi filmovi = new Filmovi();
    servletContext.setAttribute(FilmoviController.FILMOVI_KEY, filmovi);
}
```



Controllers

Pristup ServletContext objektu

- Primarni način pristupa ServletContext objektu je korišćenjem Dependency Injection obrazca

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController {

    @Autowired
    private ServletContext servletContext;
    private String bURL;
```



Controllers

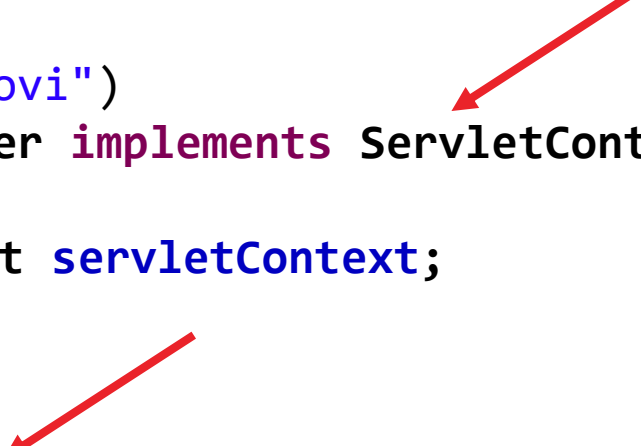
Pristup ServletContext objektu

- Alternativni način - Neophodno je da kontroler implementira interfejs `ServletContextAware` i redefiniše metodu `setServletContext`
- Spring će pozvati metodu `setServletContext` pokrenuti nakon kreiranja Bean objekta, a neposredno pre njegove inicijalizacije tj. pre poziva metode `init()`

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController implements ServletContextAware{

    private ServletContext servletContext;
    private String bURL;

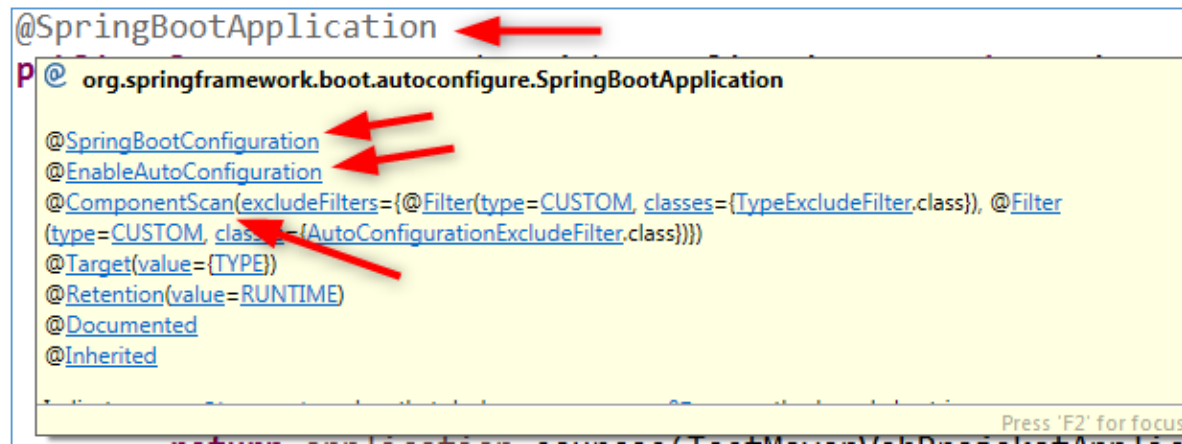
    @Override
    public void setServletContext(ServletContext servletContext) {
        this.servletContext = servletContext;
    }
}
```



Spring Boot projekat

Anotacija @SpringBootApplication – PODSEĆANJE

- Povlači za sobom još tri anotacije @SpringBootConfiguration, @EnableAutoConfiguration i @ComponentScan
 - @SpringBootConfiguration – označava da se u *TestMavenWebApplication* klasi nalazi konfiguracija za Spring projekat
 - @EnableAutoConfiguration omogućava automatsku konfiguraciju **Spring Application Context**, pokušavajući da pogodi i konfiguriše **binove, komponente i biblioteke** koje aplikacija koristi
 - oslanja se na *pom.xml*
 - @ComponentScan govori Spring aplikaciji da prođe kroz kompletan projekat i za svaku Bean klasu da pročita njenu konfiguraciju iz Java koda.



```
@SpringBootApplication  
@org.springframework.boot.autoconfigure.SpringBootApplication  
  
@SpringBootConfiguration  
@EnableAutoConfiguration  
@ComponentScan(excludeFilters={@Filter(type=CUSTOM, classes={TypeExcludeFilter.class}), @Filter  
(type=CUSTOM, classes={AutoConfigurationExcludeFilter.class})})  
@Target(value={TYPE})  
@Retention(value=RUNTIME)  
@Documented  
@Inherited
```

The screenshot shows the source code of the `@SpringBootApplication` annotation. Red arrows point from the text in the list above to the corresponding annotations in the code: `@SpringBootApplication`, `@SpringBootConfiguration`, `@EnableAutoConfiguration`, and `@ComponentScan`.

Controllers

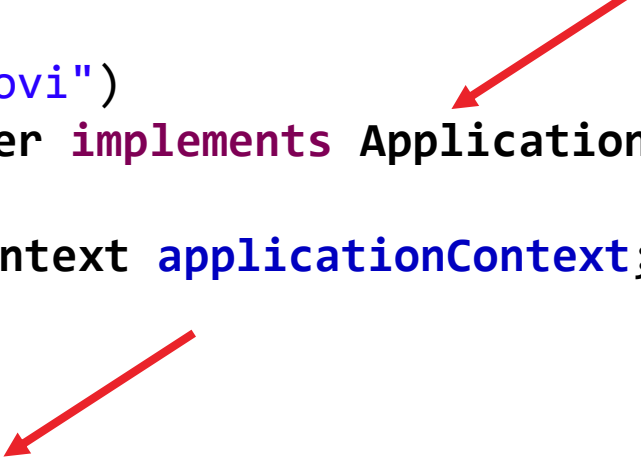
Pristup ApplicationContext objektu

- Primarni način - Neophodno je da kontroler implementira interfejs `ApplicationContextAware` i redefiniše metodu `setApplicationContext`
- Spring će pozvati metodu `setApplicationContext` pokrenuti nakon kreiranja Bean objekta, a neposredno pre njegove inicijalizacije tj. pre poziva metode `init()`

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController implements ApplicationContextAware {

    private ApplicationContext applicationContext;
    private String bURL;

    @Override
    public void setApplicationContext (ApplicationContext applicationContext) {
        this.applicationContext = applicationContext;
    }
}
```



Controllers

Pristup ApplicationContext objektu

- Alternativni način pristupa ApplicationContext objektu je korišćenjem Dependency Injection obrazca
- Ovaj način se ne preporučuje u literaturi jer u nekim uslovima i situacijama može doći do greške



```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController {

    @Autowired
    private ApplicationContext applicationContext;
    private String bURL;
```

Controllers

Pristup ApplicationContext objektu

- WebApplicationContext je klasa naslednica ApplicationContext
- Moguće je pristupiti ServletContext i Enviroment objektima

@PostConstruct

public void init() {

//WebApplicationContext je klasa naslednica ApplicationContext

if (**applicationContext instanceof** WebApplicationContext) {

 servletContext = ((WebApplicationContext) applicationContext).getServletContext();

 //Specify the base URL for all relative URLs in a document

 bURL = servletContext.getContextPath()+"/";

}

 bURL = applicationContext.getEnvironment()

 .getProperty("server.servlet.contextPath")+"/";

...

}

Spring Boot projekat

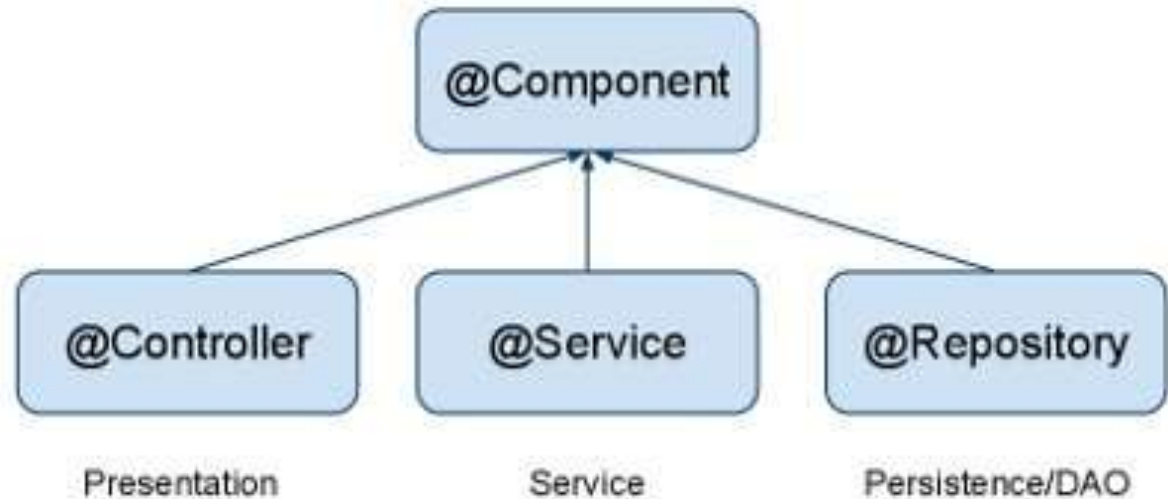
JavaBean klase u Spring radnom okviru – **PODSEĆANJE**

- Mnogi razvojni okviri zahtevaju da klase budu pisane uz poštovanje JavaBean standarda
- JavaBean je običan standard za pisanje klasa.
 - Standard nalaže da klasa mora zadovoljavati sledeće osobine da bi bila Bean:
 - Svi atributi klase moraju biti privatni (private) – koristiti getere i setere
 - Klasa mora imati javni (public) konstruktor bez parametara
 - Klasa implementira interfejs Serializable

Spring Boot projekat

JavaBean klase u Spring radnom okviru – **PODSEĆANJE**

- Ako želimo da klasa bude automatski pronađena i prepoznata od strane **Spring kontejnera** kao bean koja je komponenta Springa, treba je anotirati kao `@Component`. Umesto generičke anotacije `@Component` u praksi se koriste njene specijalizacije zavisno od uloge klase u aplikaciji.
- Hirerahija Spring komponenti
- **auto-detect** beans
- **auto-configure** beans
- **implicit one-to-one mapping** between the annotated class and the bean
- Svi bean su instancirani po Singleton paternu



Spring Boot projekat

JavaBean klase u Spring radnom okviru

- Ako želimo da klasu **eksplicitno** odredimo kao jedan bean objekat za koji sami **zadajemo njenu konfiguraciju** tada je potrebno koristiti anotaciju `@Bean`. Umesto da se oslonimo na Spring da to automatski uradi (anotacija `@Component`).
- Anotacija `@Bean` se koristi za anotiranje **metoda** koje će kao povratnu vrednost vratiti objekat određene klase koji postaje bean objekat.
- `@Bean` anotacijom moguće je zadati ime bean objekta na osnovu kojeg je on dostupan, metodu za inicijalizaciju i metodu za destrukciju bean objekta
- **explicitly** configure beans
- **explicit** one-to-one mapping between the annotated method and the bean

Spring Boot projekat

JavaBean klase u Spring radnom okviru

```
@Bean(name= {"memorijaAplikacije"}, initMethod="init", destroyMethod="destroy")
public MemorijaAplikacije getMemorijaAplikacije() {
    return new MemorijaAplikacije();
}
```

```
public class MemorijaAplikacije extends HashMap {
    @Override
    public String toString() {
        return "MemorijaAplikacije"+this.hashCode();
    }
    public void init() {
        //inicijalizacija
        System.out.println("init method called");
    }
    public void destroy() {
        //brisanje
        System.out.println("destroy method called");
    }
}
```

Spring Boot projekat

JavaBean klase u Spring radnom okviru

- Sve `@Bean` anotirane klase moraju da se nalaze unutar konfiguracije tj. neophodno je kreirati klasu i anotirati je sa `@Configuration`, čime se klasa preglašava kao dodatna konfiguracija za Spring.

```
@Configuration
```

```
public class SecondConfiguration {
```

```
    @Bean(name= {"memorijaAplikacije"},  
           initMethod="init", destroyMethod="destroy")
```

```
    public MemorijaAplikacije getMemorijaAplikacije() {  
        return new MemorijaAplikacije();  
    }
```

```
    public class MemorijaAplikacije extends HashMap {... }
```

```
}
```

Spring Boot projekat

Razlike između @Bean i @Component

1. @Component **auto detects** and **configures** the beans using classpath scanning whereas @Bean **explicitly declares** a single bean, rather than letting Spring do it automatically.
2. @Component **does not decouple** the declaration of the bean from the class definition, where as @Bean **decouples** the declaration of the bean from the class definition.
3. @Component is a **class level annotation** whereas @Bean is a **method level annotation** and name of the method serves as the bean name.
4. @Component **need not to be used with the @Configuration** annotation, whereas @Bean annotation has to be **used within the class which is annotated with @Configuration**.
5. We **cannot create a bean** of a class using @Component, if the class is outside Spring container. We **can create a bean** of a class using @Bean even if the class is present **outside the Spring container**. Functionality can be reused in other frameworks when class is outside Spring container.
6. @Component has **different specializations** like @Controller, @Repository and @Service whereas @Bean has **no specializations**.

Spring Boot projekat

Razlike između @Bean i @Component

Osobine	@Bean	@Component
mapiranje	Eksplicitno	Implicitno
Kontrola instanciranja	Moguće je upravljati logikom kreiranja instance bean	Nije moguće
razdvajanje	Razdvaja definiciju bean od deklaracije klase	Nije razdvojena definicija bean od deklaracije klase
nivo anotacije	Nivo metode	Nivo klase
@Configuration	Neophodno je da se definiše kao deo konfiguracije	Nije neophodno
Klasa izvan Spring kontejnera	Moguće je kreirati bean ako se klasa nalazi izvan Spring kontejnera. Ako se biznis logika nalazi u toj klasi tada se ona može iskoristiti u drugim radnim okvirima.	Nije moguće kreirati bean ako se klasa nalazi izvan Spring kontejnera. Ako se biznis logika nalazi u toj klasi tada se ona ne može iskoristiti u drugim radnim okvirima.
Specijalizacija	Bez specijalizacije	@Controller, @Repository i @Service

Spring Boot projekat

Pristup bean objektu

- Primarni način – bean objektu se pristupa korišćenjem Dependency Injection obrazca i anotacije `@Autowired`

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController {

    @Autowired
    private MemorijaAplikacije memorijaAplikacije;
```

Spring Boot projekat

Pristup bean objektu

- Alternativni način – bean objektu se pristupa preko ApplicationContext objekta i to:
 - pretragom klase za koju je kreiran bean ili
 - preko naziva bean (naziva metode na osnovu koje se kreira bean)

```
@Controller
```

```
@RequestMapping(value="/Filmovi")
```

```
public class FilmoviController {
```

```
    private MemorijaAplikacije memorijaAplikacije;
```

```
    /** inicijalizacija podataka za kontroler */
```

```
    @PostConstruct
```

```
    public void init() {
```

```
        memorijaAplikacije = applicationContext.getBean(MemorijaAplikacije.class);
```

```
        memorijaAplikacije = (MemorijaAplikacije)
```

```
        applicationContext.getBean("memorijaAplikacije");
```

```
        ...
```

Controllers

CRUD operacije

Nastavak naredni
termin

- Po pravilu jedan kontroler je **zadužen za manipulaciju jednim tipom entiteta**.
- U klasičnoj veb aplikaciji metode kontrolera koje odgovaraju na HTTP GET zahteve trebaju da obezbede vraćanje različitih HTML stranica koje obezbeđuju prikaz:
 - svih entiteta,
 - određenog entiteta,
 - forme za unos novog entiteta,
 - forme za izmenu postojećeg entiteta.
- Metode kontrolera koje odgovaraju na HTTP POST zahteve trebaju da omoguće preuzimanje i obradu parametara forme sa HTML stranica za unos novog entiteta, izmenu postojećeg entiteta, ili brisanje postojećeg entiteta, da izvrše poslovnu logiku sa modelovanim entitetima (sačuvaju/izmene/obrišu npr. entitet u bazu podatka) i ukoliko su prethodne operacije izvršene bez greške trebaju da upute Klijenta na HTML stranicu za prikaz svih entiteta.

Controllers

CRUD operacije i URL

- Očekivano je da se html stanica za prikaz svih entiteta dobija kroz bazični url za entitet, a da se sve ostale HTML stanice i metode za obradu parametar forme dobijaju u odnosu na dodatak putanje na bazični url za entitet
 - URL HTML stanice za **prikaz svih entiteta**, get zahtev: **/Filmovi**
 - URL HTML stanice za **prikaz određenog entiteta** , get zahtev: **/Filmovi/Details?id=1**
 - URL HTML stanice za **unos novog entiteta**, get zahtev: **/Filmovi/Create**
 - URL za **obradu podataka forme** za **unos** novog entiteta, post zahtev: **/Filmovi/Create**
 - URL HTML stanice za **prikaz izmene postojećeg entiteta**, get zahtev : **/Filmovi/Edit?id=1**
 - URL za **obradu podataka forme** za **izmenu** postojećeg entiteta , post zahtev: **/Filmovi/Edit**
 - URL za **obradu podataka forme** za **brisanje** postojećeg entiteta , post zahtev: **/Filmovi/Delete**

Case study – CRUD bioskop veb aplikacija

Rad sa filmovima

- Aplikacija ima za cilj da omogući rad online bioskopa
- **Trenutna implementacija omogućuje CRUD operacije sa entitetom film**
- Aplikacija Klijentu pruža HTML stranice za
 - pregled svih raspoloživih filmova
 - pregled određenog filma
 - prikaz forme za unos novog filma
- Aplikacija Klijentu omogućuje akcije za
 - kreiranje novog filma
 - izmenu podataka određenog filma
 - brisanje određenog filma

Case study – CRUD bioskop veb aplikacija

Prikaz svih filmova

• [filmovi](#)

• [projekcije](#)

Filmovi

r. br.	naziv	trajanje	projekcije
1	The Shining	146	projekcije Projekcije
2	One Flew Over the Cuckoo's Nest	133	projekcije Projekcije
3	The Little Shop of Horrors	72	projekcije Projekcije

• [dodavanje filma](#)

Case study – CRUD bioskop veb aplikacija

Prikaz određenog filma

• [filmovi](#)

• [projekcije](#)

Film

naziv:	The Shining
trajanje:	146
	Izmeni
	Obriši

165%

Case study – CRUD bioskop veb aplikacija

Prikaz forme za unos novog filma

• [filmovi](#)

• [projekcije](#)

Film

naziv:	<input type="text"/>
trajanje:	<input type="text" value="5"/>
<input type="button" value="Dodaj"/>	

165%

Case study – CRUD bioskop veb aplikacija

Model podataka Film

- Klasa Film reprezentuje entitet film u aplikaciji

```
public class Film {  
  
    private Long id;  
    private String naziv;  
    private int trajanje;  
  
    public Film() {  
        super();  
    }  
    ...  
}
```

Case study – CRUD bioskop veb aplikacija

Sadržaj datoteke filmovi.txt

1;The Shining;146

2;One Flew Over the Cuckoo's Nest;133

3;The Little Shop of Horrors;72

Case study – CRUD bioskop veb aplikacija

Model podataka Filmovi

- Klasa Filmovi reprezentuje kontejner za filmove
- Sadrži metode za čitanje filmova iz fajla i metode za manipulaciju sa kolekcijom filmova

```
public class Filmovi {  
  
    private Map<Long, Film> filmovi = new HashMap<>();  
    private long nextId = 1L;  
  
    /** Cita filmove iz datoteke i smesta ih u asocijativnu listu filmova. */  
    public Filmovi() {}  
  
    /** vraca film u odnosu na zadati id */  
    public Film findOne(Long id) {}  
  
    /** vraca sve filmove */  
    public List<Film> findAll() {}  
  
    /** cuva podatke o filmu filmove */  
    public Film save(Film film) {}  
  
    /** cuva podatke o svim filmovima */  
    public List<Film> save(List<Film> films) {}  
  
    /** brise film u odnosu na zadati id */  
    public Film delete(Long id) {}  
  
    /** brise filmove u odnosu na zadate ids */  
    public void delete(List<Long> ids) {}  
  
    /** vraca sve filmove ciji naziv zapocinje sa tekстом */  
    public List<Film> findByNaziv(String naziv) {}  
}
```


Case study – CRUD bioskop veb aplikacija

Model podataka Filmovi

```
public class Filmovi {
    private Map<Long, Film> filmovi = new HashMap<>();
    private long nextId = 1L;
    /** Cita filmove iz datoteke i smesta ih u asocijativnu listu filmova. */
    public Filmovi() {
        try {
            Path path = Paths.get(getClass().getClassLoader().getResource("filmovi.txt").toURI());
            System.out.println(path.toFile().getAbsolutePath());
            List<String> lines = Files.readAllLines(path, Charset.forName("UTF-8"));

            for (String line : lines) {
                line = line.trim();
                if (line.equals("") || line.indexOf('#') == 0)
                    continue;
                String[] tokens = line.split(";");
                Long id = Long.parseLong(tokens[0]);
                String naziv = tokens[1];
                int trajanje = Integer.parseInt(tokens[2]);

                filmovi.put(Long.parseLong(tokens[0]), new Film(id, naziv, trajanje));
                if(nextId<id)
                    nextId=id;
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Case study – CRUD bioskop veb aplikacija

FilmoviController

Predstavlja kontroler za rad sa entitetom Film

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController implements ServletContextAware{

    public static final String FILMOVI_KEY = "filmovi";
    // @Autowired
    private ServletContext servletContext;
    private String bURL;

    /** pristup ServletContext */
    public void setServletContext(ServletContext servletContext) {}

    /** inicijalizacija podataka za kontroler */
    public void init() {}

    /** pribavljjanje HTML stanice za prikaz svih entiteta, get zahtev */
    // GET: Filmovi
    public String index() {}
```

Case study – CRUD bioskop veb aplikacija

FilmoviController

Bioskop veb aplikacija

Predstavlja kontroler za rad sa entitetom Film

```
/** pribavljanje HTML stanice za prikaz određenog entiteta , get zahtev */  
// GET: Filmovi/Details?id=1  
public String details(@RequestParam Long id) {  
  
    /** pribavljanje HTML stanice za unos novog entiteta, get zahtev */  
    // GET: Filmovi/Create  
    public String create() {  
  
        /** obrada podataka forme za unos novog entiteta, post zahtev */  
        // POST: Filmovi/Create  
        public void create(@RequestParam String naziv, @RequestParam int trajanje, HttpServletResponse response)  
  
        /** obrada podataka forme za izmenu postojećeg entiteta, post zahtev */  
        // POST: Filmovi/Edit  
        public void edit(@ModelAttribute Film filmEdited , BindingResult result, HttpServletResponse response) t  
  
        /** obrada podataka forme za brisanje postojećeg entiteta, post zahtev */  
        // POST: Filmovi/Delete  
        public void delete(@RequestParam Long id, HttpServletResponse response) throws IOException {  

```

Case study – CRUD bioskop veb aplikacija

Kreiranje statistike filmova na nivou aplikacije

- U odnosu na korišćenje aplikacije od strane svih korisnika za sve filmove definisati statistiku za posećenost filmova.
 - Prethodno implicira da bi podaci trebali da se čuvaju u ServletContext ili ApplicationContext objektu.
- Statistički podaci se obično nalaze na dnu **svake** HTML stanice
 - Statistika se prikazuje tabelarno, za svaki posećeni film prikazuje se naziv filma i broj poseta za posmatrani film na nivou aplikacije
- Iskorišćene su klase modela FilmStatistika i FilmBrojac
 - FilmBrojac klasa čuva broj poseta za određeni film
 - FilmStatistika je kontejnerska klasa koja omogućava rad sa objektima klase FilmBrojac. Ona omogućava vođenje statistike za filmove, sadrži operacije uvećanja broja poseta za film, sortiranje posećenih filmova na osnovu broja poseta itd.
- Broj poseta za određeni film se uvećava za 1 kada Klijent zatraži URL HTML stanice za prikaz određenog filma.

Case study – CRUD bioskop veb aplikacija

Kreiranje statistike filmova na nivou aplikacije

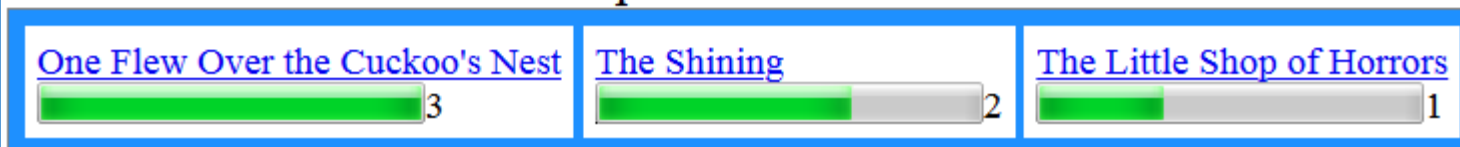
- [filmovi](#)
- [projekcije](#)

Filmovi

r. br.	naziv	trajanje	projekcije
1	The Shining	146	projekcije Projekcije
2	One Flew Over the Cuckoo's Nest	133	projekcije Projekcije
3	The Little Shop of Horrors	72	projekcije Projekcije

- [dodavanje filma](#)

Popularni filmovi



Case study – CRUD bioskop veb aplikacija

Kreiranje statistike filmova na nivou aplikacije

- Prikaz HTML kod za kreiranje ispisa u kontroleru FilmoviController

```
<table class="horizontalni-meni">
  <caption>Popularni filmovi</caption>
  <tbody><tr>
    <td>
      <ul>
        <li>
          <a href="http://localhost:8080/Filmovi/Details?id=2">One Flew Over the Cuckoo's Nest</a><br>
          <progress value="3" max="3"></progress>3
        </li>
        <li>
          <a href="http://localhost:8080/Filmovi/Details?id=1">The Shining</a><br>
          <progress value="2" max="3"></progress>2
        </li>
        <li>
          <a href="http://localhost:8080/Filmovi/Details?id=3">The Little Shop of Horrors</a><br>
          <progress value="1" max="3"></progress>1
        </li>
      </ul>
    </td>
  </tr>
</tbody></table>
```

Kreiranje statistike filmova na nivou aplikacije

Case study – CRUD bioskop veb aplikacija

Vežbanje posle predavanja

- Pokušajte da uradite KorisniciController

Case study – rad sa korisnicima

Vežba posle Termin 4 – stranica za registraciju korisnika

Neophodno je bilo kreirati HTML stranicu koja omogućava registraciju novih korisnika na sistem (npr. registracija.html) .

- Korisnik koji se registruje na sistem treba da unese ime, prezime, pol, platu, korisnicko ime i korisnicku šifru (pogledati klasu Korisnik).
 - pri registraciji korisnika neophodno je 2 puta da se unese korisnička šifra
 - sva polja su obavezna osim plate
 - plata je numerička ne negativna vrednost
 - Iskoristi HTML 5 za unos numeričkih vrednosti
- HTML stranica registracija.html kreirana je tako se se zadrži postojeći izgled stilova aplikacije (po ugledu HTML stanicu za unos novog filma)

Case study – rad sa korisnicima

Vežba posle Termin 4 – stranica za registraciju korisnika

Izgled HTML stranice registracija.html koja omogućava registraciju novih korisnika na sistem.

- [filmovi](#)
- [projekcije](#)
- [korisnici](#)

Registracija korisnika

ime:	<input type="text" value="Petar"/>
prezime:	<input type="text" value="Petrović"/>
pol:	<input checked="" type="radio"/> M <input type="radio"/> Ž
plata:	<input type="text" value="54000.00"/>
korisničko ime:	<input type="text" value="pera"/>
korisnicka šifra:	<input type="password" value="••••"/>
ponovljena šifra:	<input type="password" value="••••"/>
<input type="button" value="Dodaj"/>	

```
<ul>
  <li><a href="Filmovi">filmovi</a></li>
  <li><a href="Projekcije">projekcije</a></li>
  <li><a href="Korisnici">korisnici</a></li>
</ul>
<form method="post" action="Korisnici/Create">
  <table>
    <caption>Registracija korisnika</caption>
    <tr><th>ime:</th><td><input type="text" value="" name="ime" required/></td></tr>
    <tr><th>prezime:</th><td><input type="text" value="" name="prezime" required/></td></tr>
    <tr><th>pol:</th><td><input type="radio" name="pol" value="muški" checked/>M
      <input type="radio" name="pol" value="ženski"/>Ž</td></tr>
    <tr><th>plata:</th><td><input type="number" min="0" step="any" name="plata"/></td></tr>
    <tr><th>korisničko ime:</th><td><input type="text" value=""
      name="korisnickoIme" required/></td></tr>
    <tr><th>korisnicka šifra:</th><td><input type="password" value=""
      name="korisnickaSifra" required/></td></tr>
    <tr><th>ponovljena šifra:</th><td><input type="password" value=""
      name="ponovljenaSifra" required/></td></tr>
    <tr><th></th><td><input type="submit" value="Dodaj" /></td>
  </table>
</form>
```

registracija.html

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

Neophodno je bilo kreirati kontroler KorisnikController koji omogućava CRUD operacije sa korisnicima (kreiran je po ugledu na klasu FilmController)

- Mapu svih korisnika (kao i filmovi što su urađeni) čuva se u memoriji na nivou aplikacije (dodavanjem objekta u ServletContext ili dodavanjem objekata za kreirani bean u ApplicationContext)
- Aplikacija Klijentu pruža HTML stranice za
 - pregled svih korisnika (ime, prezime i korisničko ime prikazati za svakog korisnika)
 - pregled određenog korisnika
 - prikaz forme za registraciju novog korisnika
- Sve HTML stranice su kreirane tako se se zadrži postojeći izgled stilova aplikacije (po ugledu na generisane HTML stranice kontrolera FilmController za rad sa entitetom Film)

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Mapu svih korisnika čuvati u memoriji na nivou aplikacije dodavanjem objekta u ServletContext – verzija 1

```
@Controller
@RequestMapping(value="/Korisnici")
public class KorisnikController {
    public static final String KORISNICI_KEY = "korisnici";

    @Autowired
    private ServletContext servletContext;
    private String bURL;

    /** inicijalizacija podataka za kontroler */
    @PostConstruct
    public void init() {
        //Specify the base URL for all relative URLs in a document
        bURL = servletContext.getContextPath()+"/";
        HashMap<String, Korisnik> korisnici = new HashMap<String, Korisnik>();
        korisnici.put("pera", new Korisnik("Petar", "Petarović", "muški", "pera", "pera", 60000));
        korisnici.put("steva", new Korisnik("Steva", "Stević", "muški", "steva", "steva", 50000));
        korisnici.put("jova", new Korisnik("Jova", "Jović", "muški", "jova", "jova", 45000));
        servletContext.setAttribute(KorisnikController.KORISNICI_KEY, korisnici);
    }
}
```

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Mapu svih korisnika čuvati u memoriji na nivou aplikacije. Mapa se dodaje kao objekat za kreirani bean u ApplicationContext – verzija 2

```
@Controller
@RequestMapping(value="/Korisnici")
public class KorisnikController {
    public static final String KORISNICI_KEY = "korisnici";

    @Autowired
    private ServletContext servletContext;
    @Autowired
    private MemorijaAplikacije memorijaAplikacije;
    private String bURL;

    /** inicijalizacija podataka za kontroler */
    @PostConstruct
    public void init() {
        //Specify the base URL for all relative URLs in a document
        bURL = servletContext.getContextPath()+"/";
        HashMap<String, Korisnik> korisnici = new HashMap<String, Korisnik>();
        korisnici.put("pera", new Korisnik("Petar", "Petarović", "muški", "pera", "pera", 60000));
        korisnici.put("steva", new Korisnik("Steva", "Stević", "muški", "steva", "steva", 50000));
        korisnici.put("jova", new Korisnik("Jova", "Jović", "muški", "jova", "jova", 45000));
        memorijaAplikacije.put(KorisnikController.KORISNICI_KEY, korisnici);
    }
}
```

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Aplikacija Klijentu pruža HTML stranice za
 - **pregled svih korisnika** (ime, prezime i korisničko ime prikazati za svakog korisnika)
 - pregled određenog korisnika
 - prikaz forme za registraciju novog korisnika



← → ↻ ⓘ localhost:8080/BioskopVebAplikacija/Korisnici

- [filmovi](#)
- [projekcije](#)
- [korisnici](#)

Korisnici

r. br.	ime	prezime	korisničko ime
1	Petar	Petarović	pera
2	Jova	Jović	jova
3	Steva	Stević	steva

- [registracija korisnika](#)

KorisnikController get index()

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Aplikacija Klijentu pruža HTML stranice za
 - pregled svih korisnika (ime, prezime i korisničko ime prikazati za svakog korisnika)
 - **pregled određenog korisnika**
 - prikaz forme za registraciju novog korisnika

• [filmovi](#)
• [projekcije](#)
• [korisnici](#)

Korisnik

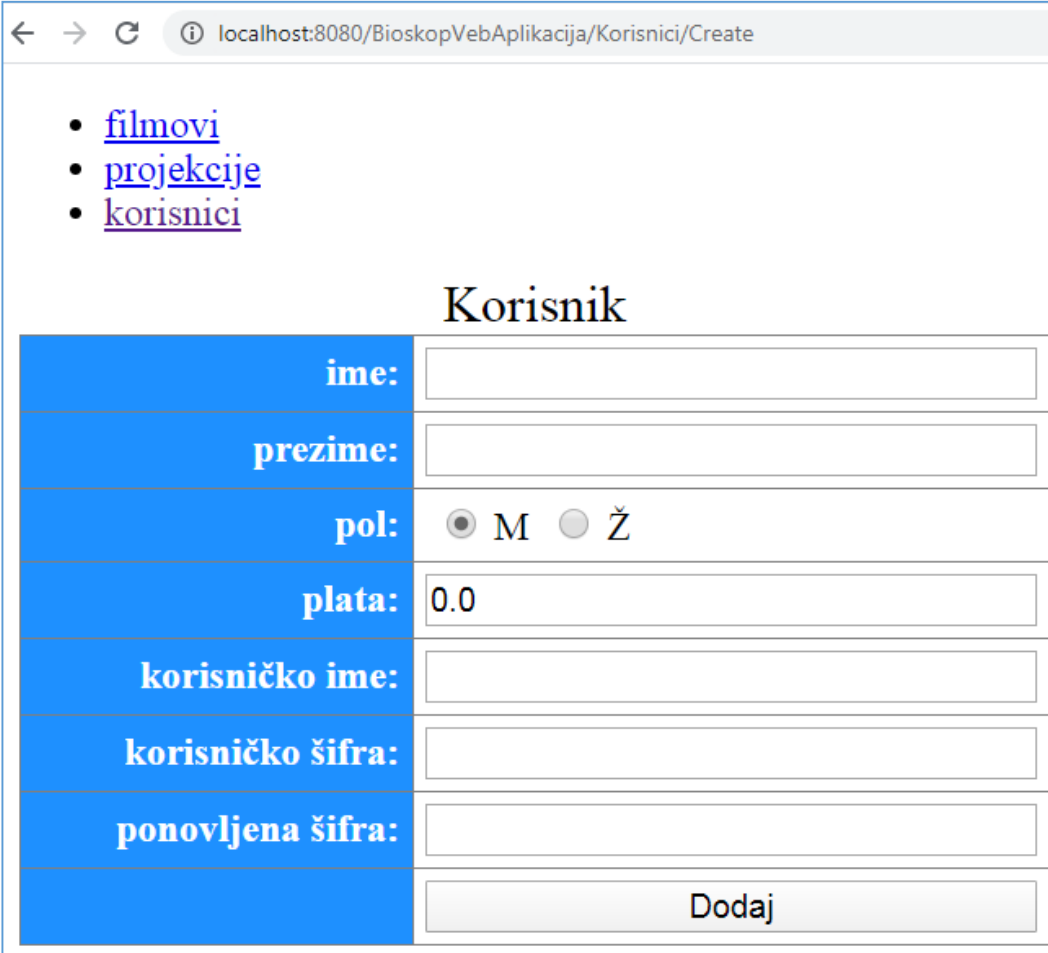
ime:	<input type="text" value="Petar"/>
prezime:	<input type="text" value="Petarović"/>
pol:	<input checked="" type="radio"/> M <input type="radio"/> Ž
plata:	<input type="text" value="60000.0"/>
korisničko ime:	<input type="text" value="pera"/>
korisničko šifra:	<input type="password" value="...."/>
ponovljena šifra:	<input type="password" value="...."/>
	<input type="button" value="Izmeni"/>
	<input type="button" value="Obriši"/>

KorisnikController get details() i vratiHTMLZaKorisnika()

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Aplikacija Klijentu pruža HTML stranice za
 - pregled svih korisnika (ime, prezime i korisničko ime prikazati za svakog korisnika)
 - pregled određenog korisnika
 - **prikaz forme za registraciju novog korisnika**



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/BioskopVebAplikacija/Korisnici/Create'. The page content includes a list of links: [filmovi](#), [projekcije](#), and [korisnici](#). Below the links is a form titled 'Korisnik' with the following fields and controls:

	Korisnik
ime:	<input type="text"/>
prezime:	<input type="text"/>
pol:	<input checked="" type="radio"/> M <input type="radio"/> Ž
plata:	<input type="text" value="0.0"/>
korisničko ime:	<input type="text"/>
korisničko šifra:	<input type="text"/>
ponovljena šifra:	<input type="text"/>
	<input type="button" value="Dodaj"/>

KorisnikController get create()

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

Neophodno je bilo kreirati kontroler KorisnikController koji omogućava CRUD operacije sa korisnicima (kreiran je po ugledu na klasu FilmController)

- Aplikacija Klijentu omogućuje akcije za
 - registraciju novog korisnika
 - izmenu podataka određenog korisnika
 - brisanje određenog korisnika
- Sve akcije kontrolera izvršiće odgovarajuću programsku logiku i redirektovati Klijenta na odgovarajuću HTML stranicu (po ugledu na akcije kontrolera FilmController za rad sa entitetom Film).

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Aplikacija Klijentu omogućuje akcije za
 - registraciju novog korisnika
 - izmenu podataka određenog korisnika
 - brisanje određenog korisnika

```
@PostMapping(value="/Create")
public void create(@ModelAttribute Korisnik korisnik, @RequestParam(defaultValue="") String ponovljenaSifra,
    BindingResult result, HttpServletResponse response) throws IOException {
    //preuzimanje vrednosti iz konteksta
    HashMap<String, Korisnik> korisnici = (HashMap<String, Korisnik>)
        servletContext.getAttribute(KorisnikController.KORISNICI_KEY);
    String greska = "";
    if (result.hasErrors()) {
        greska = "greška prilikom preuzimanja podataka<br/>";
    }
    if(korisnik.getKorisnickoIme().trim().equals(""))
        greska+="korisničko ime nije uneseno<br/>";
    if(korisnici.get(korisnik.getKorisnickoIme())!=null)
        greska+="korisničko ime nije jedinstveno<br/>";
    if(korisnik.getKorisnickaSifra().trim().equals(""))
        greska+="korisnička šifra nije unesena<br/>";
    if(!korisnik.getKorisnickaSifra().equals(ponovljenaSifra))
        greska+="korisničke šifre se ne podudaraju<br/>";
    if(korisnik.getPlata()<0)
        greska+="korisnička plata je ne negativna vrednost<br/>";
    if(!greska.equals("")) {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out;
        out = response.getWriter();
        out.write(vratiHTMLZaKorisnika("Create", greska, korisnik, ponovljenaSifra, "Korisnik", "Korisnik", "Korisnici/Create"));
        return;
    }

    korisnici.put(korisnik.getKorisnickoIme(), korisnik);
    response.sendRedirect(bURL+"Korisnici");
}
```

KorisnikController post create()

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Aplikacija Klijentu omogućuje akcije za
 - registraciju novog korisnika
 - **izmenu podataka određenog korisnika**
 - brisanje određenog korisnika

```
@SuppressWarnings("unchecked")
// POST: Korisnici/Edit
@PostMapping(value="/Edit")
public void edit(@ModelAttribute Korisnik korisnikEdited, @RequestParam String ponovljenaSifra,
    BindingResult result, HttpServletResponse response) throws IOException {
    //preuzimanje vrednosti iz konteksta
    HashMap<String, Korisnik> korisnici = (HashMap<String, Korisnik>)
        servletContext.getAttribute(KorisnikController.KORISNICI_KEY);
    String greska = "";
    if (result.hasErrors() || korisnikEdited==null) {
        greska = "greška prilikom preuzimanja podataka<br/>";
    }
    if(korisnici.get(korisnikEdited.getKorisnickoIme())==null)
        greska+="korisnik ne postoji u evidenciji<br/>";
    if(!korisnikEdited.getKorisnickaSifra().equals(ponovljenaSifra))
        greska+="korisničke šifre se ne podudaraju<br/>";
    if(korisnikEdited.getPlata()<0)
        greska+="korisnička plata je ne negativna vrednost<br/>";
    if(!greska.equals("")) {
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out;
        out = response.getWriter();
        out.write(vratiHTMLZaKorisnika("Details", greska, korisnikEdited, ponovljenaSifra, "Korisnik", "Korisnik", "Korisnici/Edit"));
        return;
    }
}
```

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Aplikacija Klijentu omogućuje akcije za
 - registraciju novog korisnika
 - **izmenu podataka određenog korisnika**
 - brisanje određenog korisnika

KorisnikController post edit()

```
//zasto se korisnik preuzima iz repozitorijuma, a mu se menjaju podaci
//zasto se preuzeti korsnik iz forme direktno ne sačuva u repoyitorijuma
Korisnik korisnik = korisnici.get(korisnikEdited.getKorisnickoIme());
if(korisnik!=null && korisnik.getKorisnickaSifra().equals(ponovljenaSifra)) {
    if(korisnikEdited.getIme()!=null && !korisnikEdited.getIme().trim().equals(""))
        korisnik.setIme(korisnikEdited.getIme());
    if(korisnikEdited.getPrezime()!=null && !korisnikEdited.getPrezime().trim().equals(""))
        korisnik.setPrezime(korisnikEdited.getPrezime());
    if(korisnikEdited.getPol()!=null && !korisnikEdited.getPol().trim().equals(""))
        korisnik.setPol(korisnikEdited.getPol());
    if(korisnikEdited.getPlata() > 0)
        korisnik.setPlata(korisnikEdited.getPlata());
    if(korisnikEdited.getKorisnickaSifra()!=null && !korisnikEdited.getKorisnickaSifra().trim().equals(""))
        korisnik.setKorisnickaSifra(korisnikEdited.getKorisnickaSifra());
}
response.sendRedirect(bURL+"Korisnici");
}
```

Case study – rad sa korisnicima

Vežba posle Termin 4 – KorisnikController

- Aplikacija Klijentu omogućuje akcije za
 - registraciju novog korisnika
 - izmenu podataka određenog korisnika
 - **brisanje određenog korisnika**

KorisnikController post delete()

```
// POST: Korisnici/Delete
@PostMapping(value="/Delete")
public void delete(@RequestParam String korisnickoIme, HttpServletResponse response) throws IOException {
    //preuzimanje vrednosti iz konteksta
    HashMap<String, Korisnik> korisnici = (HashMap<String, Korisnik>)servletContext.getAttribute(KorisnikController.KORISNICI_KEY);
    Korisnik deleted = korisnici.remove(korisnickoIme);
    response.sendRedirect(bURL+"Korisnici");
}
```

Dodatno

Anotacija @SpringBootApplication

Anotacija @SpringBootApplication

- Povlači za sobom još tri anotacije (vidi sliku) @SpringBootConfiguration, @EnableAutoConfiguration i @ComponentScan

```
@SpringBootConfiguration  
@EnableAutoConfiguration  
@ComponentScan(excludeFilters={@Filter(type=CUSTOM, classes={TypeExcludeFilter.class}), @Filter(type=CUSTOM, classes={AutoConfigurationExcludeFilter.class})})  
@Target(value={TYPE})  
@Retention(value=RUNTIME)  
@Documented  
@Inherited
```

Indicates a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning. This is a convenience annotation that is equivalent to declaring @Configuration, @EnableAutoConfiguration and @ComponentScan.

Since:

1.2.0

Author:

Anotacija @SpringBootApplication

Anotacija @EnableAutoConfiguration

- Anotacija *@EnableAutoConfiguration* omogućava automatsku konfiguraciju *Spring Application Context*, pokušavajući da pogodi i konfiguriše binove, komponente i biblioteke koje aplikacija koristi
 - oslanja se na pom.xml. Radi se za sve biblioteke koje su tamo navedene i njihove zavisnosti. U pom.xml ako navedemo da koristimo Tomcat veb kontejner to znači da će se tomcat-embedded.jar će se naći u u *classpath* aplikacije
 - automatska konfiguracija se oslanja na *classpath* aplikacije. Kako se u classpath nalazi *tomcat-embedded.jar* to znači da treba da se uradi automatska konfiguracija te biblioteke

Anotacija @SpringBootApplication

Anotacija @ComponentScan

- Anotacija *@ComponentScan* označava Spring aplikaciji da prođe kroz kompletan projekat i za svaku Bean klasu da pročita njenu konfiguraciju iz Java koda.
 - Anotacijom nad klasom se klasa proglašava Spring Bean klasom (*@Component*, *@Controller*, *@Service*, *@Repository*).
- Skeniranje se izvršava u dubinu za sve pakete od paketa klase za koju se koristi anotacija *@ComponentScan*.
 - U našem slučaju skeniranje se vrši od paketa *com.ftn.PrviMavenVebProjekat* za sve Java klase koje se nalaze u tom paketu i podpaketima tog paketa .

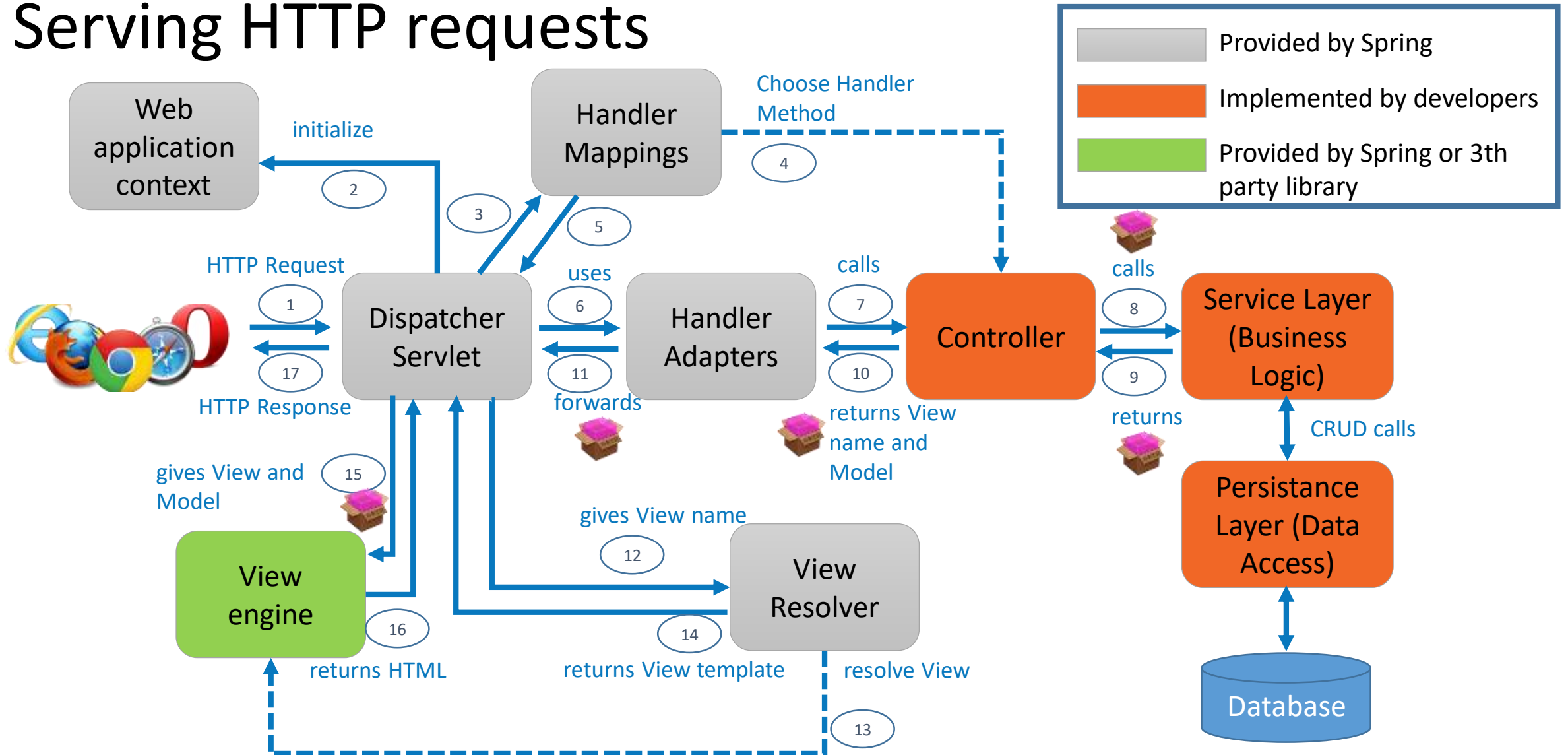
JavaBean klase u Spring radnom okviru

Anotacija @Component

- Mnogi razvojni okviri zahtevaju da klase budu pisane uz poštovanje JavaBean standarda.
- Gde se označava da je neka klasa Bean i da će objektima te klase biti upravljano od strane kontejnera?
 - Anotacijom nad klasom se klasa proglašava Spring Bean klasom (@Component, @Controller, @Service, @Repository)

Spring MVC workflow

Serving HTTP requests



Spring MVC workflow

DispatcherServlet

- The *DispatcherServlet* is one of the important components of the Spring MVC web framework and acts as a *Front Controller*.
- It is inherited from `javax.servlet.http.HttpServlet`, it is typically configured in the `web.xml` file.
- Front Controller is a single entry point for a client request and its' job is to receive all incoming request and delegates to other components for actual processing like Spring MVC controllers which are annotated using `@Controller` and `@RequestMapping` annotations and ViewResolvers like the `InternalResourceViewResolver` class

Spring MVC workflow

DispatcherServlet

- DispatcherServlet is used following things in Spring MVC:
 - Receives all request as Front Controller and provides a single entry point to the application
 - Mapping requests to correct [Spring MVC](#) controller
 - Consulting ViewResolvers to find correct View
 - Forwarding request to chosen View for rendering
 - Returning the response to the client
 - Creates web-context to initialize the web-specific beans like [controllers](#), view resolvers and handler mapping

Spring MVC workflow

Spring MVC workflow to serve HTTP requests

DispatcherServlet is a Front Controller and provides a single entry point for application

1. **DispatcherServlet** receives HTTP requests from the **Browser**.
2. **DispatcherServlet** initializes **WebApplicationContext**, this is doing only one time. During initialization of WebApplicationContext all the Spring Beans are created and stored and their xml configurations are read

DispatcherServlet uses handler mappings and handler adapters to map a request to the Spring MVC Controllers.

3. **DispatcherServlet** asks **Handler Mappings** to see which method of which class is responsible for processing this request.
4. **A Handler Mappings** is like a database of Handler Method where a single Handler Method is a method in a Java class that maps to specific URL and handle incoming HTTP request. **Handler Mappings** uses **@Controller** and **@RequestMapping** annotation in Java classes for that purpose to choose an appropriate **Handler Method**.

Spring MVC workflow

Spring MVC workflow to serve HTTP requests

5. **Handler mappings** send back the Handler Method's details to the **DispatcherServlet**.
6. **DispatcherServlet** knows which method is responsible for handling this request. It uses **Handler Adapters** to make a call to that handler method of Controller. The HTTP request data will also be passed to the handler method as parameters.
7. **Handler Adapters** calls the **Handler Method of Controller**.
8. **Handler Method of Controller** processes the input, if needed it creates Model and calls **Service** methods.
9. **Service** provides CRUD operations for Model and if needed it returns Model to **Handler Method of Controller**.

Spring MVC workflow

Spring MVC workflow to serve HTTP requests

- 10. Handler method** returns to **Handler Adapters** the logical view name (View) for the HTML template and the data (Model) needed to render View. Controllers' handler methods can be configured not to return logical view name by declaring return type as void, and by doing that handler method can be used to directly return the content of HTTP response.
- 11. Handler Adapters** forwards the logical view name and data (Model) to **DispatcherServlet**.
- 12. DispatcherServlet** gives the view name to the **View Resolver** to get the actual view content

Spring MVC workflow

Spring MVC workflow to serve HTTP requests

DispatcherServlet uses InternalResourceViewResolver which uses prefix and suffix to convert a logical view name into a physical resource like a JSP page or a FreeMarker or Thymeleaf template e.g. "home" to /WEB-INF/home.jsp.

13. View resolver resolves the View by its view name.

14. View resolver returns the actual HTML template content back.

15. Dispatcher gives the Model and View to the **View Engine** (JSP or Thymeleaf template engine, for example).

16. View engine merges the view template with the data and produces plain old HTML and sends it back to DispatcherServlet

17. DispatcherServlet sends the generated HTML back to the browser

Handler method supported return types

Tipovi povratnih vrednosti

- A ModelAndView object, with the model implicitly enriched with command objects and the results of @ModelAttribute annotated reference data accessor methods.
- A Model object, with the view name implicitly determined through a RequestToViewNameTranslator and the model implicitly enriched with command objects and the results of @ModelAttribute annotated reference data accessor methods.
- A Map object for exposing a model, with the view name implicitly determined through a RequestToViewNameTranslator and the model implicitly enriched with command objects and the results of @ModelAttribute annotated reference data accessor methods.
- A View object, with the model implicitly determined through command objects and @ModelAttribute annotated reference data accessor methods. The handler method may also programmatically enrich the model by declaring a Model argument (see above).

Handler method supported return types

Tipovi povratnih vrednosti

- A String value that is interpreted as the logical view name, with the model implicitly determined through command objects and `@ModelAttribute` annotated reference data accessor methods. The handler method may also programmatically enrich the model by declaring a Model argument (see above).
- void if the method handles the response itself (by writing the response content directly, declaring an argument of type `ServletResponse` / `HttpServletResponse` for that purpose) or if the view name is supposed to be implicitly determined through a `RequestToViewNameTranslator` (not declaring a response argument in the handler method signature).
- If the method is annotated with `@ResponseBody`, the return type is written to the response HTTP body. The return value will be converted to the declared method argument type using `HttpMessageConverter`'s. See [the section called “Mapping the response body with the @ResponseBody annotation”](#).
- An `HttpEntity<?>` or `ResponseEntity<?>` object to provide access to the Servlet response HTTP headers and contents. The entity body will be converted to the response stream using `HttpMessageConverter`'s. See [the section called “Using HttpEntity”](#).

Handler method supported return types

Tipovi povratnih vrednosti

- An HttpHeaders object to return a response with no body.
- A Callable<?> can be returned when the application wants to produce the return value asynchronously in a thread managed by Spring MVC.
- A DeferredResult<?> can be returned when the application wants to produce the return value from a thread of its own choosing.
- Any other return type is considered to be a single model attribute to be exposed to the view, using the attribute name specified through @ModelAttribute at the method level (or the default attribute name based on the return type class name). The model is implicitly enriched with command objects and the results of @ModelAttribute annotated reference data accessor methods.
- Više informacija na <https://docs.spring.io/spring/docs/4.0.3.RELEASE/spring-framework-reference/htmlsingle/#mvc-ann-return-types>
- Kada se bude radio prezentacioni deo sa Thymeleaf View Engine koristiće se neke od pomenutih vrednosti

Handler method supported return types

Tipovi povratnih vrednosti

- Primer kada bi se vraćao ModelAndView objekat, obavezno korišćenje View Engine

@Controller

```
public class MojKontroler {  
    @RequestMapping(value = "/getDateAndTime")  
    public ModelAndView getDateAndTime() {  
        LocalDateTime now = LocalDateTime.now();  
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss");  
        String date_time = dtf.format(now);  
  
        //the logical view name  
        ModelAndView mav = new ModelAndView("showMessage");  
  
        //add object as attribute now of model  
        mav.addObject("now", now);  
        mav.addObject("date_time", date_time);  
  
        return mav;  
    }  
}
```

Handler method supported return types

Tipovi povratnih vrednosti

- ModelAndView objekat služi kao pogodan “noseći” objekat za podatke koje View treba da prikaže, i informaciju o samom View-u koji treba iskoristiti za prikaz.

Handler method supported argument types

Tipovi argumenata funkcija

- Request or response objects (Servlet API). Choose any specific request or response type, for example `ServletRequest` or `HttpServletRequest`.
- Session object (Servlet API): of type `HttpSession`. An argument of this type enforces the presence of a corresponding session. As a consequence, such an argument is never null.
- `org.springframework.web.context.request.WebRequest` or `org.springframework.web.context.request.NativeWebRequest`. Allows for generic request parameter access as well as request/session attribute access, without ties to the native Servlet/Portlet API.
- `java.util.Locale` for the current request locale, determined by the most specific locale resolver available, in effect, the configured `LocaleResolver` in a Servlet environment.

Handler method supported argument types

Tipovi argumenata funkcija

- `java.io.InputStream` / `java.io.Reader` for access to the request's content. This value is the raw `InputStream/Reader` as exposed by the Servlet API.
- `java.io.OutputStream` / `java.io.Writer` for generating the response's content. This value is the raw `OutputStream/Writer` as exposed by the Servlet API.
- `org.springframework.http.HttpMethod` for the HTTP request method.
- `java.security.Principal` containing the currently authenticated user.
- `@PathVariable` annotated parameters for access to URI template variables. See [the section called "URI Template Patterns"](#).
- `@MatrixVariable` annotated parameters for access to name-value pairs located in URI path segments. See [the section called "Matrix Variables"](#).
- `@RequestParam` annotated parameters for access to specific Servlet request parameters. Parameter values are converted to the declared method argument type. See [the section called "Binding request parameters to method parameters with @RequestParam"](#).

Handler method supported argument types

Tipovi argumenata funkcija

- `@RequestHeader` annotated parameters for access to specific Servlet request HTTP headers. Parameter values are converted to the declared method argument type.
- `@RequestBody` annotated parameters for access to the HTTP request body. Parameter values are converted to the declared method argument type using ``HttpMessageConverter`s`. See [the section called “Mapping the request body with the `@RequestBody` annotation”](#).
- `@RequestPart` annotated parameters for access to the content of a "multipart/form-data" request part. See [Section 16.11.5, “Handling a file upload request from programmatic clients”](#) and [Section 16.11, “Spring’s multipart \(file upload\) support”](#).
- `HttpEntity<?>` parameters for access to the Servlet request HTTP headers and contents. The request stream will be converted to the entity body using ``HttpMessageConverter`s`. See [the section called “Using `HttpEntity`”](#).

Handler method supported argument types

Tipovi argumenata funkcija

- `java.util.Map` / `org.springframework.ui.Model` / `org.springframework.ui.ModelMap` for enriching the implicit model that is exposed to the web view.
- `org.springframework.web.servlet.mvc.support.RedirectAttributes` to specify the exact set of attributes to use in case of a redirect and also to add flash attributes (attributes stored temporarily on the server-side to make them available to the request after the redirect). `RedirectAttributes` is used instead of the implicit model if the method returns a "redirect:" prefixed view name or `RedirectView`.
- Command or form objects to bind request parameters to bean properties (via setters) or directly to fields, with customizable type conversion, depending on `@InitBinder` methods and/or the `HandlerAdapter` configuration. See the `webBindingInitializer` property on `RequestMappingHandlerAdapter`. Such command objects along with their validation results will be exposed as model attributes by default, using the command class class name - e.g. model attribute "orderAddress" for a command object of type "some.package.OrderAddress". The `ModelAttribute` annotation can be used on a method argument to customize the model attribute name used.

Handler method supported argument types

Tipovi argumenata funkcija

- `org.springframework.validation.Errors` / `org.springframework.validation.BindingResult` validation results for a preceding command or form object (the immediately preceding method argument).
- `org.springframework.web.bind.support.SessionStatus` status handle for marking form processing as complete, which triggers the cleanup of session attributes that have been indicated by the `@SessionAttributes` annotation at the handler type level.
- `org.springframework.web.util.UriComponentsBuilder` a builder for preparing a URL relative to the current request's host, port, scheme, context path, and the literal part of the servlet mapping.

Više o tome na <https://docs.spring.io/spring/docs/4.0.3.RELEASE/spring-framework-reference/htmlsingle/#mvc-ann-arguments>