



**ftn**informatika

# *Java Web Development*

Modul 2

Termin 3

# Sadržaj

## 1. *Spring*

1. Moduli radnog okvira
2. Inversion of Control (IoC)

## 2. *Spring Boot*

1. Uvod
2. Karakteristike
3. Arhitektura Spring MVC aplikacije sa procesom rada
4. Kreiranje projekta
5. Uvlačenje postojećeg
6. pom.xml

## 3. *Dependency Injection*

## 4. *Web sadržaj*

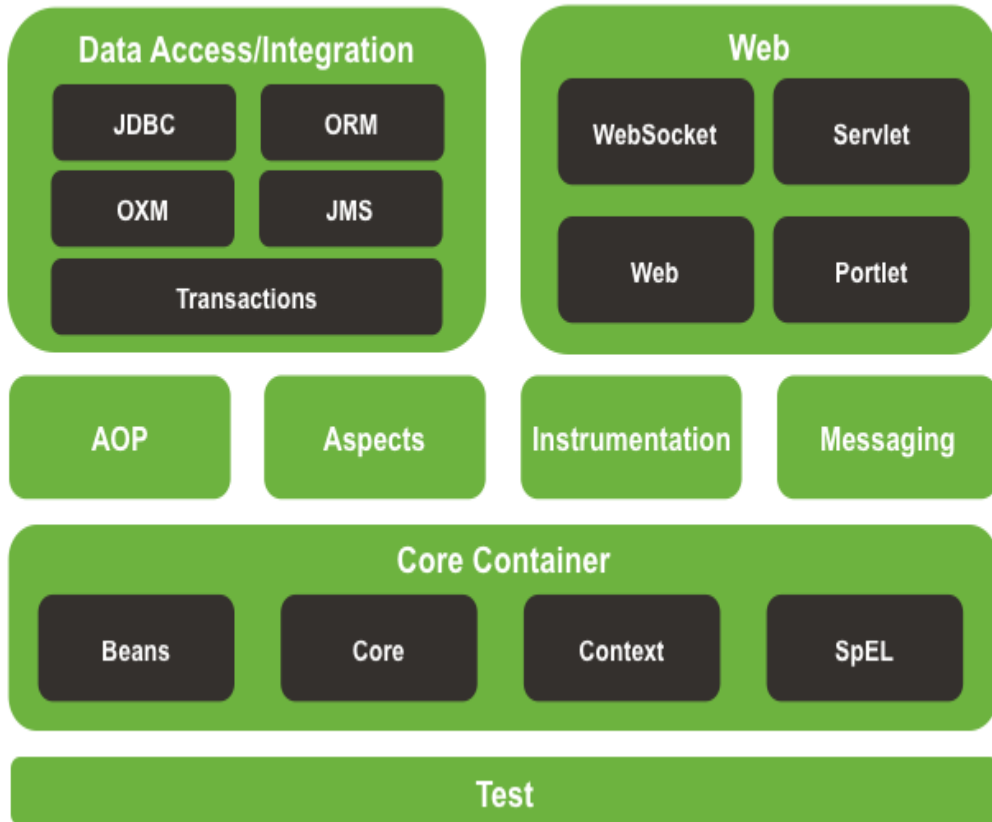
1. Statički
2. Dinamički

# Spring

## Moduli radnog okvira



### Spring Framework Runtime



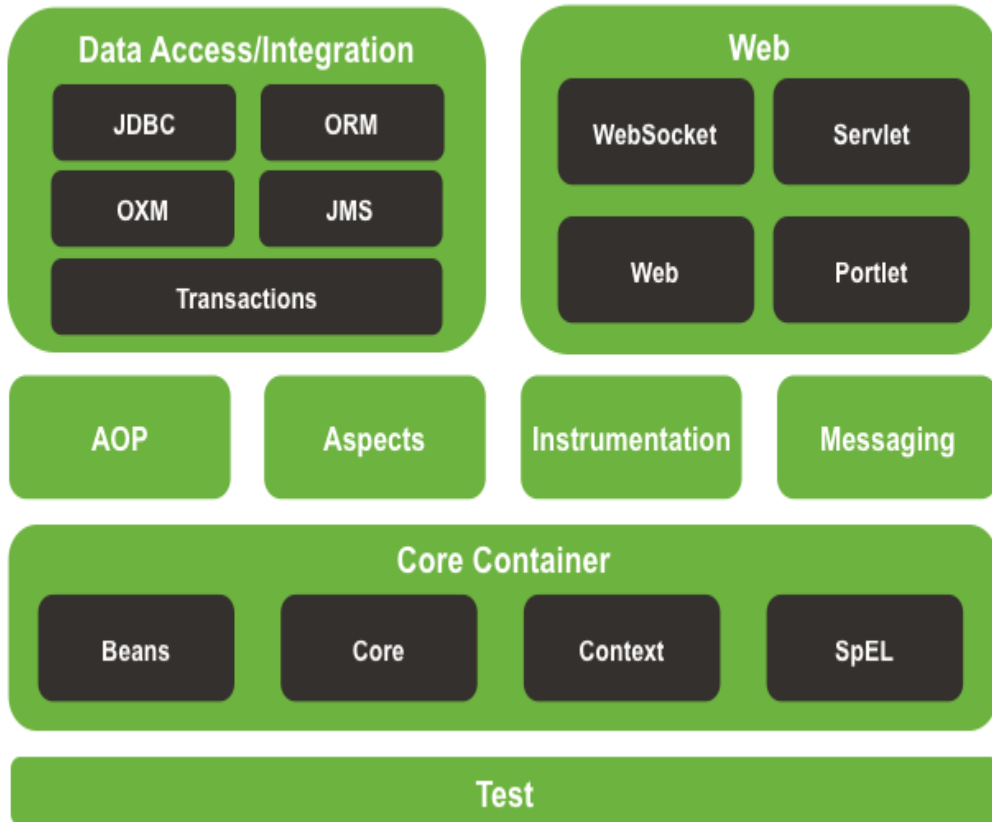
- Spring ima široko polje primene. Koristi se za izradu: veb aplikacija (klasična MVC arhitektura ili nova REST arhitektura), desktop aplikacija, mobilnih aplikacija, itd.
- Spring je radni okvir za izradu Java aplikacija
- Spring je podeljen na module, svaki modul zadužen za neku funkcionalnost.
- Spring Core modul implementira osnovne mehanizme rada Spring
- Razvoj Spring aplikacija se svodi na korišćenje Spring Core modula i onih modula koji su potrebni za konkretan tip aplikacije čija je funkcionalnost neophodno implementirati

# Spring

## Moduli radnog okvira



### Spring Framework Runtime



- Spring veb aplikacije zahtevaju uključenje modula Web.
- Ako je potreba perzistencija podataka uljučuje se Data Access/Integration modul.
- Više o Springu na <https://docs.spring.io/spring/docs/5.0.0.RC2/spring-framework-reference/overview.html>
- Za kreiranje Spring aplikacija sve biblioteke za odabrane module se mogu uvezati ručno, ali to nije praktično

## Inversion of Control (IoC)

- u projektu se programiraju komponente koje će se ugraditi u već predefinisanu aplikaciju (aplikacija kao da već postoji samo je treba doraditi)
- **kontrola nije više na našoj strani, radni okvir poziva našu komponentu**
- radni okvir implementira učestale mehanizme, nizove koraka, koji se uvek izvršavaju u aplikaciji.
- kada se dođe do konkretnog koraka koji treba da se izvrši, tada je neophodno za radni okvir isprogramirati i u njemu ugraditi odgovarajuću komponentu, koja će izvršiti odgovarajuću funkcionalnost

# Spring

## Inversion of Control (IoC)

- Princip koji se koristi kod Spring radnog okvira, da programer kreira komponente koje Spring radni okvir poziva po potrebi zove se inverzija kontrole (**Inversion of Control**) ili može se čak i zvati **Hollywood Principle**
- U knjizi *Head First Design pattern* se **Hollywood Principle** princip objašnjava *With the Hollywood Principle, we allow low-level components to hook themselves into a system, but the high-level components determine when they are needed, and how. In other words, the high-level components give the low-level components a “don’t call us, we’ll call you” treatment.*

# Spring Boot

## Uvod

- Na kursu 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

- 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



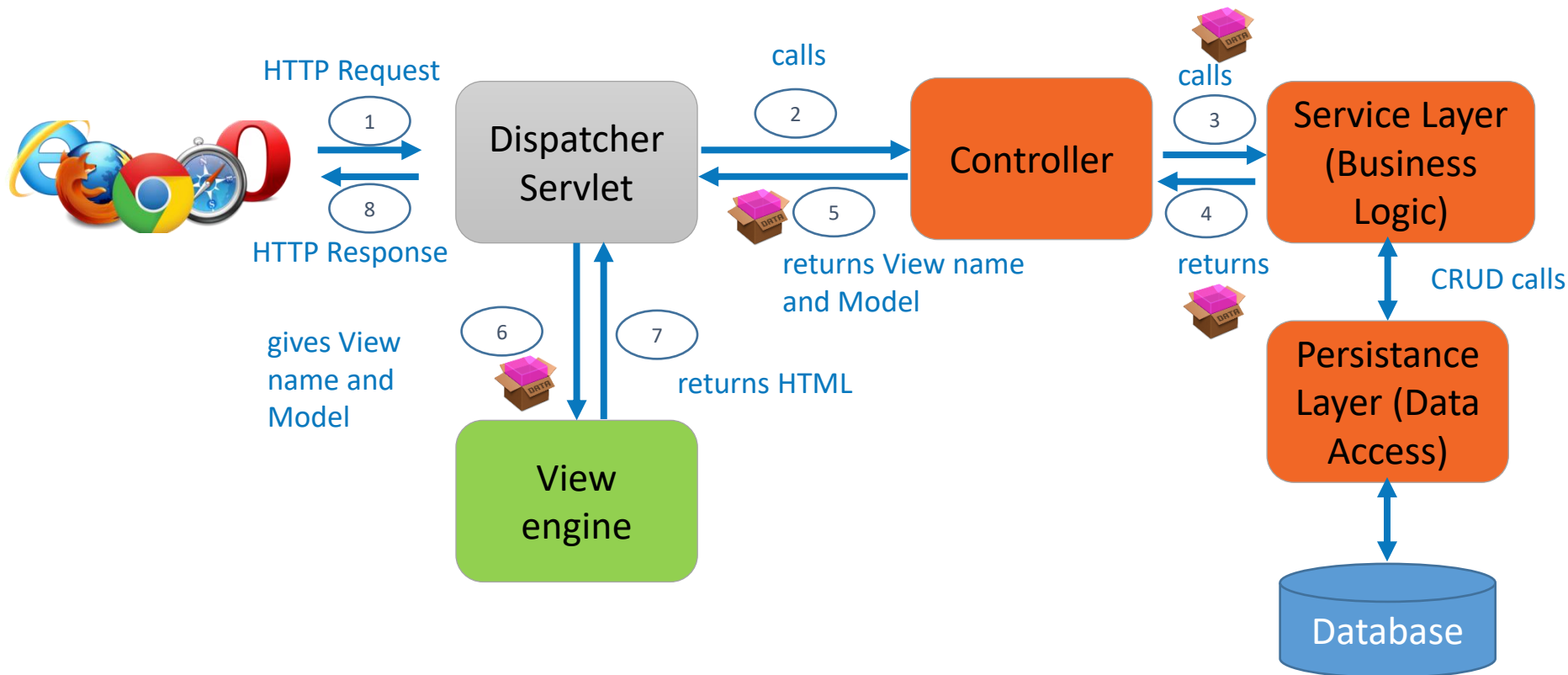
# Spring Boot

## 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

## Arhitektura Spring MVC aplikacije sa procesom rada – bez detalja



# Spring Boot

## 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

## 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/spring.io

- 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

# Uvlačenje postojećeg Maven projekta

## Uvlačenje

- U okviru workspace ImePrezime/Modul2WebMVC raspakovati zip arhivu **Modul2Termin3Bioskop.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/Modul2WebMVC 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>3.1.1</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>17</java.version>
</properties>
```

# 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

## 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 *bootstrap* omogućava korišćenje Bootstrap biblioteke.

```
<dependency>  
    <groupId>org.webjars</groupId>  
    <artifactId>bootstrap</artifactId>  
    <version>5.3.0</version>  
</dependency>
```

# Spring Boot pom.xml

## Opcione zavisnosti

- *WebJars* predstavljaju *client-side* zavisnosti zapakovane u JAR arhive. Artifakt webjars-locator koristimo automatsko razrešavanje verzija WebJar aseta.

```
<dependency>
```

```
    <groupId>org.webjars</groupId>
```

```
    <artifactId>webjars-locator</artifactId>
```

```
    <version>0.47</version>
```

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

```
</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

# Primer

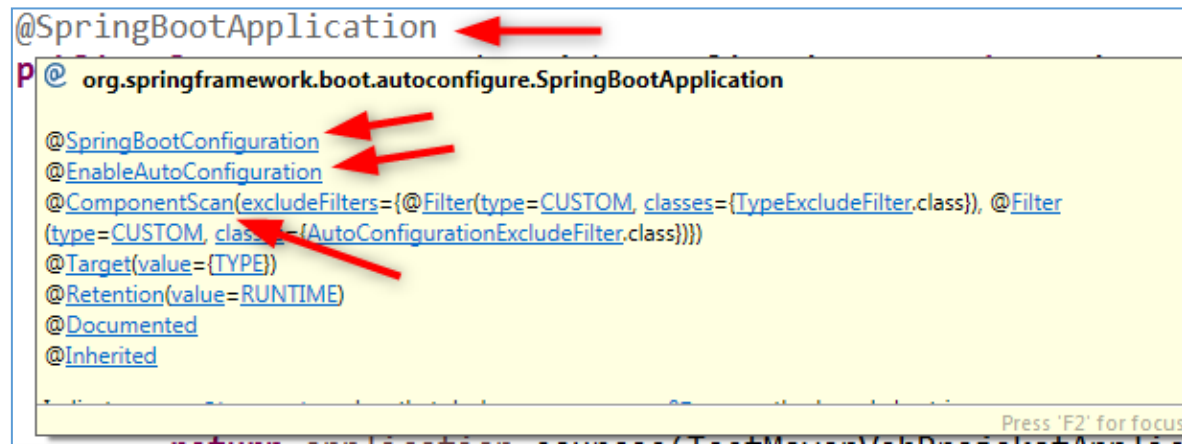
- *com.ftninformatika.jwd.modul2.termin3.bioskop.Application*



# 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

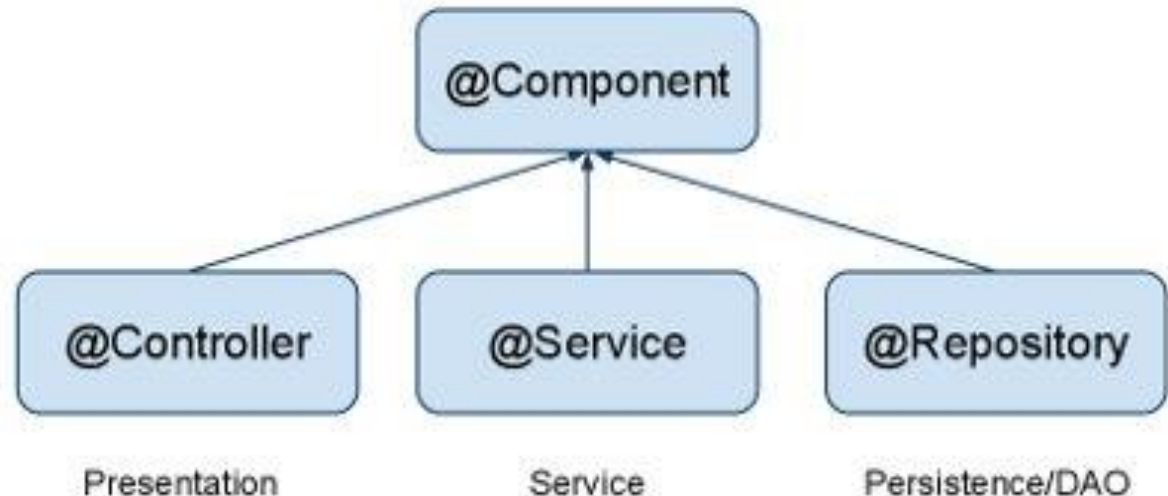
## JavaBean klase u Spring radnom okviru

- 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, treba je anotirati kao `@Component`. Umesto generičke anotacije `@Component` u praksi se koriste njene specijalizacije zavisno od uloge klase u aplikaciji.
- Hirerahija Spring komponenti



# 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(Application.class, args);  
}
```

# Kontroleri

## 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 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 FilmController {...}
```

# Kontroleri

## 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("/filmovi")
```

```
public class FilmoviController {...}
```

# Primer

- *com.ftninformatika.jwd.modul2.termin3.bioskop.web.controller.FilmController*

# Kontroleri

## 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**



# Kontroleri

## 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("/filmovi")
```

```
public class FilmController {
```

```
    @GetMapping("")
```

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

```
}
```

# Kontroleri

## 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("/filmovi")
```

```
public class FilmController {
```

```
    @GetMapping
```

```
    @ResponseBody
```

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

# Primer

- *com.ftninformatika.jwd.modul2.termin3.bioskop.web.controller*

# Dependency Injection (DI)

## Softverski obrazac/princip

- 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

# Dependency Injection (DI)

## DI u okviru Spring Framework-a

- 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.

# Dependency Injection (DI)

## DI u okviru Spring Framework-a

- **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**
- U našem slučaju se kao **Spring kontejner** koristi **Tomcat** i on će sam automatski inicijalizovati sve potrebne objekte koji imaju anotaciju `@Autowired`

**Od 4.3 verzije Spring-a, klase koje posjeduju samo jedan konstruktor mogu da izbace `@Autowired` anotaciju**

# Dependency Injection (DI)

## DI u okviru Spring Framework-a

- Anotaciju @Autowired možemo da staviti na nivou property-ja.

```
@Controller
@RequestMapping("/projekcije")
public class ProjekcijaController {

    @Autowired
    private Bioskop bioskop;

    public ProjekcijaController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }
}
```

# Dependency Injection (DI)

## DI u okviru Spring Framework-a

- Anotaciju @Autowired možemo da staviti na nivou konstruktora.

```
@Controller
@RequestMapping("/korisnici")
public class KorisnikController {

    private Bioskop bioskop;

    @Autowired
    public KorisnikController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }
}
```



# Dependency Injection (DI)

## DI u okviru Spring Framework-a

- Anotaciju @Autowired možemo da staviti na nivou set metode.

```
@Controller
@RequestMapping("/zanrovi")
public class ZanrController {

    private Bioskop bioskop;

    public ZanrController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }

    @Autowired
    public void setBioskop(Bioskop bioskop) {
        this.bioskop = bioskop;
    }
}
```

# Dependency Injection

## Implementacija

```
@Controller
public class ZavrController {
    private Bioskop bioskop;

    public ZavrController(Bioskop bioskop) { ← instantiate
        this.bioskop = bioskop;
    }

    :
}
```

```
@Controller
public class FilmController {
    private Bioskop bioskop;

    public FilmController(Bioskop bioskop) { ← instantiate
        this.bioskop = bioskop;
    }

    :
}
```

```
@Controller
public class ProjekcijaController {
    private Bioskop bioskop;

    public ProjekcijaController(Bioskop bioskop) { ← instantiate
        this.bioskop = bioskop;
    }

    :
}
```

⋮

```
@Component
public class Bioskop {
    private final Map<Long, Zavr> zavr = new LinkedHashMap<>();
    private final Map<Long, Film> filmovi = new LinkedHashMap<>();
    private final Map<Long, Projekcija> projekcije = new LinkedHashMap<>();
    private final Map<String, Korisnik> korisnici = new LinkedHashMap<>();

    public Bioskop() {

        :
    }

    :
}
```

# Dependency Injection

## Implementacija

```
@Controller
public class ZavrController {
    private Bioskop bioskop; depends on!

    public ZavrController(Bioskop bioskop) { ← instantiate
        this.bioskop = bioskop;
    }
    :
}
```

```
@Controller
public class FilmController {
    private Bioskop bioskop; depends on!

    public FilmController(Bioskop bioskop) { ← instantiate
        this.bioskop = bioskop;
    }
    :
}
```

```
@Controller
public class ProjekcijaController {
    private Bioskop bioskop; depends on!

    public ProjekcijaController(Bioskop bioskop) { ← instantiate
        this.bioskop = bioskop;
    }
    :
}
```

⋮

```
@Component
public class Bioskop {
    private final Map<Long, Zavr> zavr = new LinkedHashMap<>();
    private final Map<Long, Film> filmovi = new LinkedHashMap<>();
    private final Map<Long, Projekcija> projekcije = new LinkedHashMap<>();
    private final Map<String, Korisnik> korisnici = new LinkedHashMap<>();

    public Bioskop() {
        :
    }
    :
}
```

# Dependency Injection

## Implementacija

```
@Controller
public class ZavrController {
    private Bioskop bioskop; depends on!

    public ZavrController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }
    :
}
```

```
@Controller
public class FilmController {
    private Bioskop bioskop; depends on!

    public FilmController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }
    :
}
```

```
@Controller
public class ProjekcijaController {
    private Bioskop bioskop; depends on!

    public ProjekcijaController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }
    :
}
```

:

```
@Component
public class Bioskop {
    private final Map<Long, Zavr> zavrvi = new LinkedHashMap<>();
    private final Map<Long, Film> filmovi = new LinkedHashMap<>();
    private final Map<Long, Projekcija> projekcije = new LinkedHashMap<>();
    private final Map<String, Korisnik> korisnici = new LinkedHashMap<>();

    public Bioskop() { ← instantiate
    }
    :
}
```

# Dependency Injection

## Implementacija

```
@Controller
public class ZavrController {
    private Bioskop bioskop;

    public ZavrController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }

    :
}
```

```
@Controller
public class FilmController {
    private Bioskop bioskop;

    public FilmController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }

    :
}
```

```
@Controller
public class ProjekcijaController {
    private Bioskop bioskop;

    public ProjekcijaController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }

    :
}
```

⋮

inject

```
@Component
public class Bioskop {
    private final Map<Long, Zavr> zavrvi = new LinkedHashMap<>();
    private final Map<Long, Film> filmovi = new LinkedHashMap<>();
    private final Map<Long, Projekcija> projekcije = new LinkedHashMap<>();
    private final Map<String, Korisnik> korisnici = new LinkedHashMap<>();

    public Bioskop() {
        :
    }

    :
}
```

# HTTP

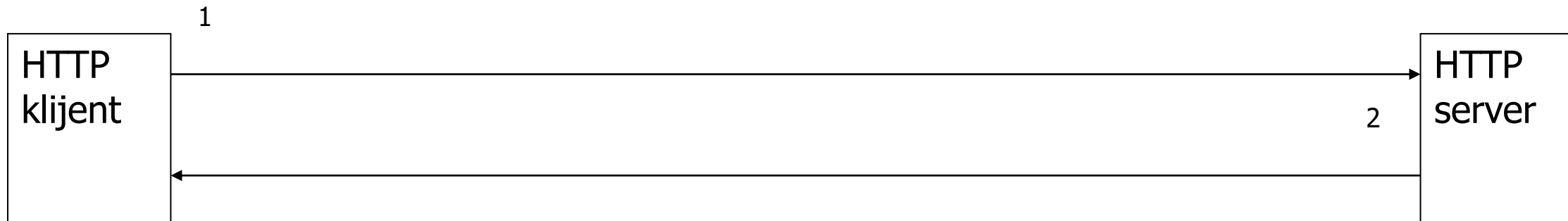
## Vrste WWW sadržaja

- statički (unapred uskladišteni)
- dinamički (generisani po zahtevu)

# HTTP

## Isporuka statičkog sadržaj

statički sadržaji se nalaze u okviru datoteka WWW servera

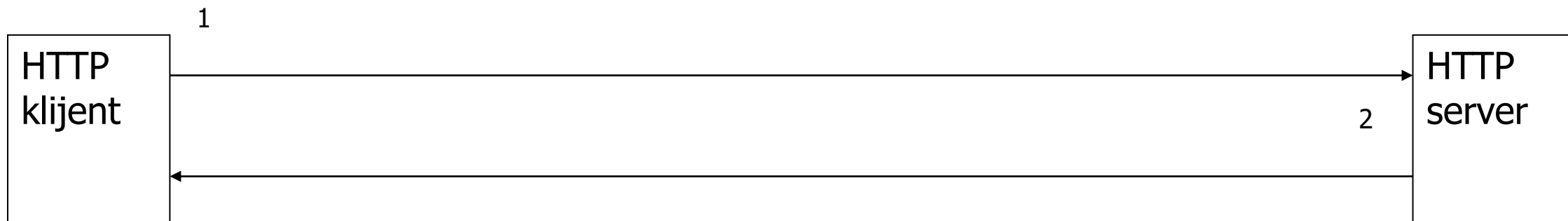


1. klijent zahteva datoteku
2. server je učitava sa svog fajl-sistema i šalje je klijentu

# HTTP

## Isporuka dinamičkog sadržaj

traženi sadržaj se generiše po zahtevu i šalje klijentu

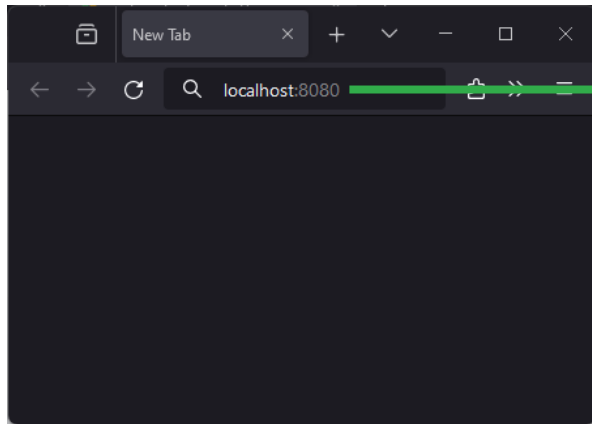


1. klijent zahteva "datoteku"
2. server je generiše i šalje klijentu. Datoteka se ne snima u fajl sistemu servera

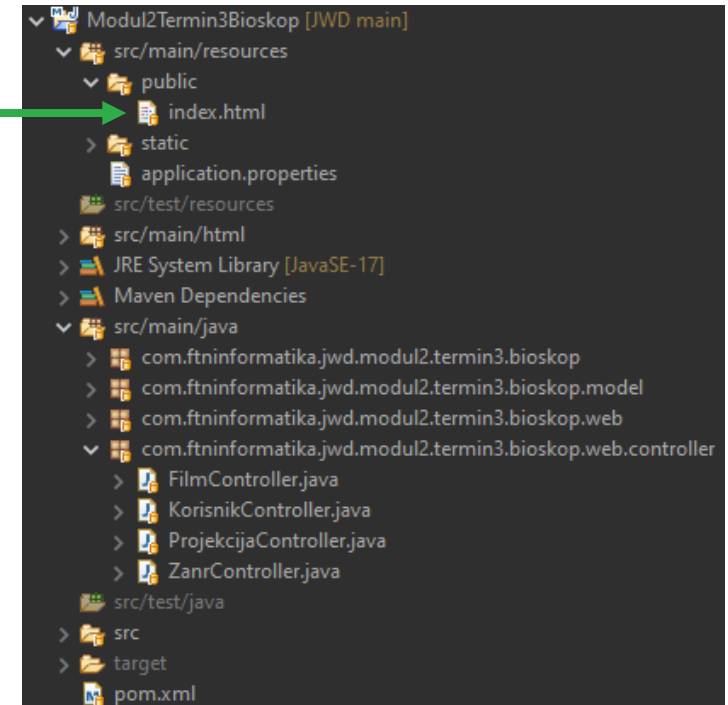
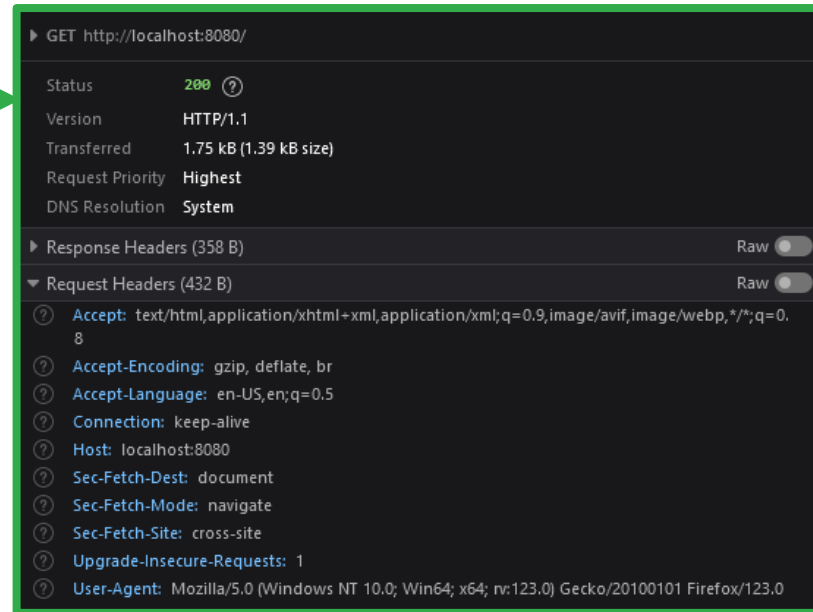


# HTTP

## GET (statički sadržaj)



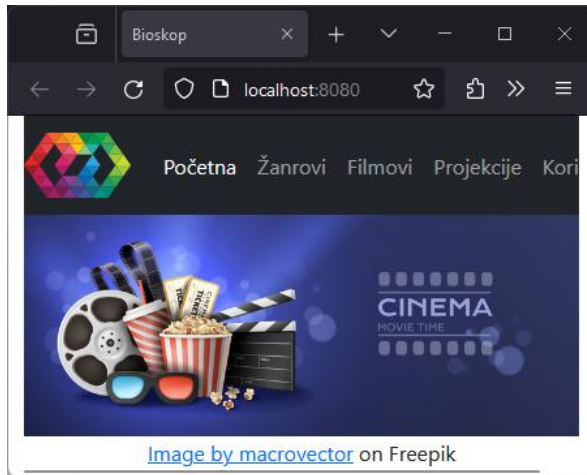
request



# HTTP

## GET (statički sadržaj)

response



GET http://localhost:8080/

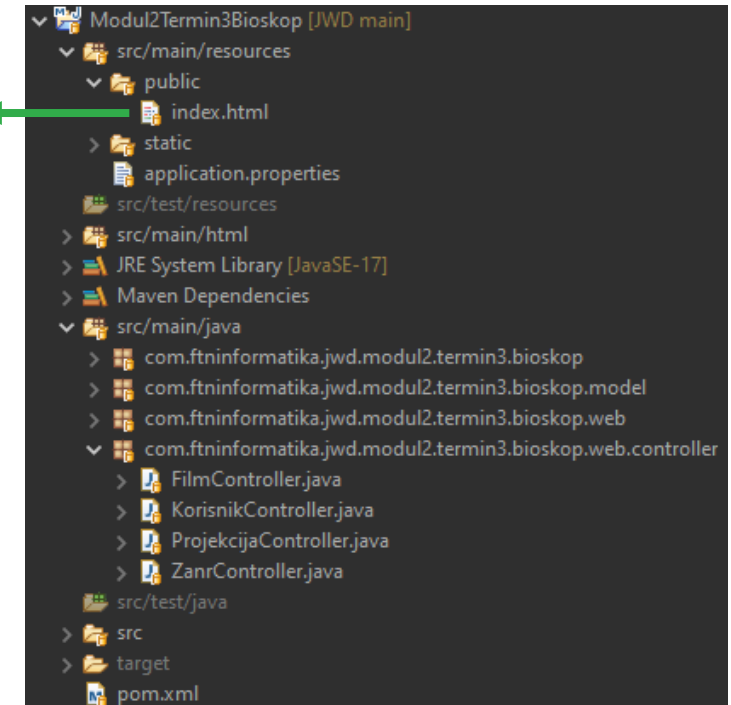
Status: 200  
Version: HTTP/1.1  
Transferred: 1.75 kB (1.39 kB size)  
Request Priority: Highest  
DNS Resolution: System

Response Headers (358 B)

- Accept-Ranges: bytes
- Cache-Control: no-store
- Connection: keep-alive
- Content-Language: en-US
- Content-Length: 1389
- Content-Type: text/html; charset=UTF-8
- Date: Mon, 26 Feb 2024 20:19:55 GMT
- Keep-Alive: timeout=60
- Last-Modified: Mon, 26 Feb 2024 14:12:26 GMT
- Vary: Origin
- Vary: Access-Control-Request-Method
- Vary: Access-Control-Request-Headers

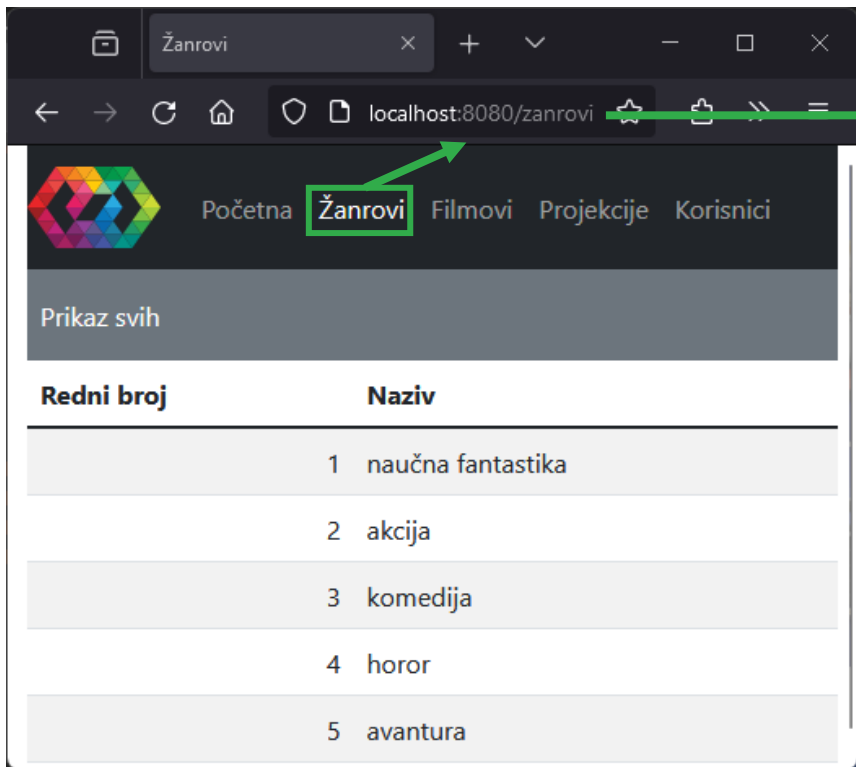
HTML

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <link rel="stylesheet" type="text/css" href="/webjars/bootstrap/css/bootstrap.min.css"/>
6   <title>Bioskop</title>
7 </head>
8 <body>
9   <div class="container-fluid">
10    <div class="row">
11      <div class="col">
12        <nav class="navbar navbar-expand navbar-dark bg-dark">
13          <a class="navbar-brand" href="https://enastava.ftninformatika.com">
14            
15          </a>
16          <div class="navbar-nav">
17            <a class="nav-item nav-link active" href="index.html">Početna</a>
18            <a class="nav-item nav-link" href="/zanrovi">Žanrovi</a>
19            <a class="nav-item nav-link" href="/filmovi">Filmovi</a>
20            <a class="nav-item nav-link" href="/projekcije">Projekcije</a>
21            <a class="nav-item nav-link" href="/korisnici">Korisnici</a>
22          </div>
23        </div>
24      </div>
25      <div class="row">
26        <div class="col text-center">
27          
28          <div><a href="https://www.freepik.com/free-vector/realistic-horizontal-cinema-movie-time-poster-with-3">
29          </div>
30        </div>
31      </div>
32    </div>
33  </body>
34 </html>
```



# HTTP

## GET (dinamički sadržaj)



request

/zanrovi

response

```
@Controller
@RequestMapping("/zanrovi")
public class ZandrController {
    private Bioskop bioskop;

    public ZandrController(Bioskop bioskop) {
        this.bioskop = bioskop;
    }

    @GetMapping("")
    @ResponseBody
    public String getAll() {
        List<Zanr> zanrovi = new ArrayList<>(bioskop.getZanrovi().values());

        StringBuilder response = new StringBuilder();
        response.append(
            "<!DOCTYPE html>\r\n"
            + "<html>\r\n"
            + "<head>\r\n"
            + " <meta charset=\"UTF-8\">\r\n"
            :
            + " </div>\r\n"
            + " </div>\r\n"
            + "</body>\r\n"
            + "</html>"
        );
        return response.toString();
    }
}
```

# Primer

- *com.ftninformatika.jwd.modul2.termin3.bioskop*