



ftninformatika

Java Web Development

Modul 2

Termin 8

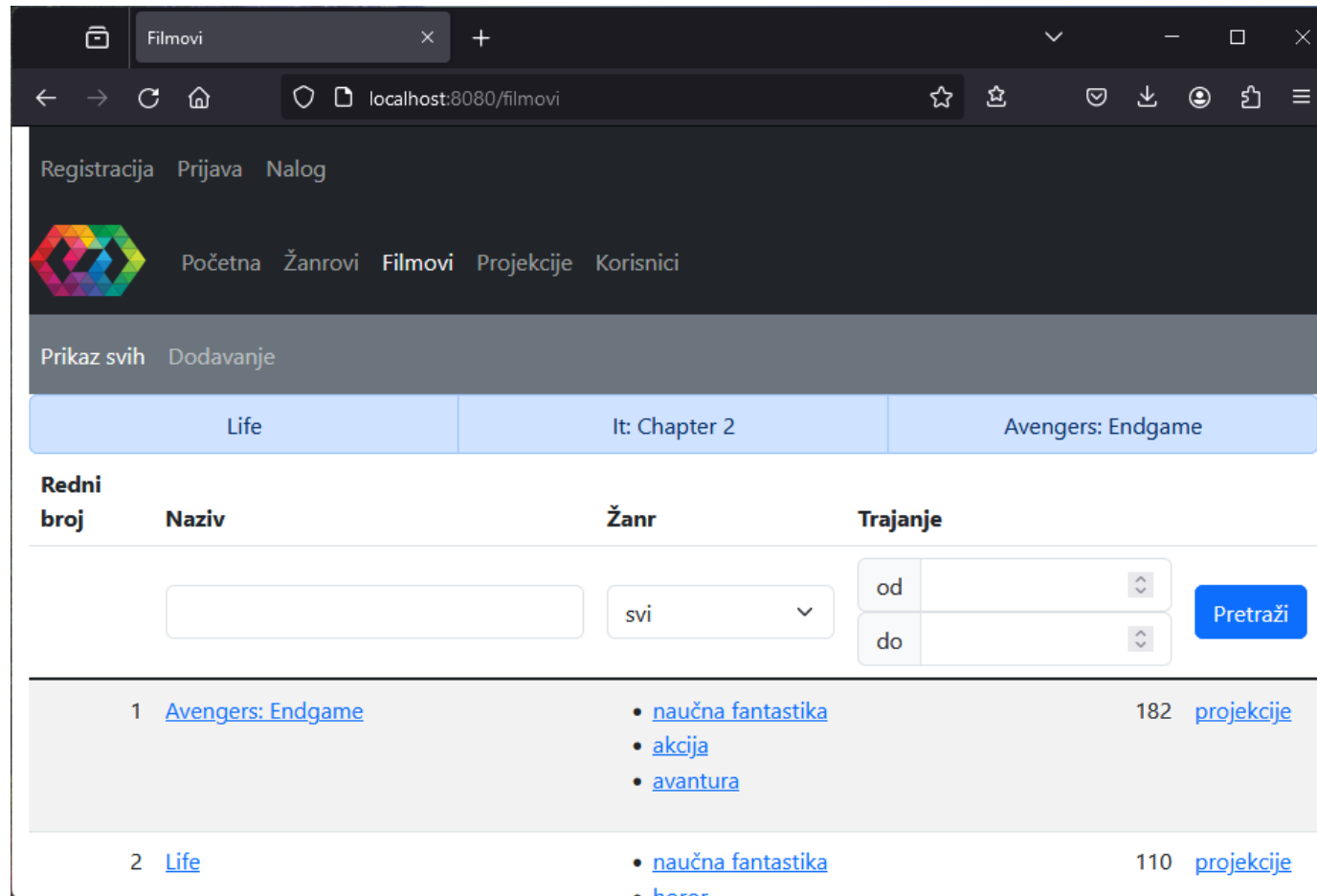
Sadržaj

1. Sesije
2. *Beans*
3. Sistem naloga
 - *HandlerInterceptor*
4. *Thymeleaf*
 - *th:if*
 - *th:unless*

Sesije

Problem

Kako prikazati istorijat posećenih filmova?



The screenshot shows a web application interface for movies. The browser address bar indicates the URL is `localhost:8080/filmovi`. The application has a dark theme and a navigation bar with links: Registracija, Prijava, Nalog, Početna, Žanrovi, **Filmovi**, Projekcije, and Korisnici. Below the navigation bar, there are tabs for 'Prikaz svih' and 'Dodavanje'. The main content area displays a list of movies. The first movie is 'Life', the second is 'It: Chapter 2', and the third is 'Avengers: Endgame'. Below the list, there is a search bar with a dropdown menu for 'Žanr' (Genre) set to 'svi' (all). The search results show two movies: 'Avengers: Endgame' and 'Life'. The 'Avengers: Endgame' entry shows a list of genres: naučna fantastika, akcija, and avantura, with a count of 182 projections. The 'Life' entry shows a list of genres: naučna fantastika and horor, with a count of 110 projections.

Redni broj	Naziv	Žanr	Trajanje
1	Avengers: Endgame	<ul style="list-style-type: none">naučna fantastikaakcijaavantura	182 projekcije
2	Life	<ul style="list-style-type: none">naučna fantastikahoror	110 projekcije

Sesije

Rešenje

```
@Component
public class FilmoviIstorijaBezSortiranja implements FilmoviIstorija {
    private static final int KAPACITET = 5;

    private final List<Long> poseceniIDs = new ArrayList<Long>();

    public void addPosecen(Long id) {
        if (poseceniIDs.contains(id)) {
            return;
        }
        poseceniIDs.add(id);
    }

    public void deletePosecen(Long id) {
        Iterator<Long> it = poseceniIDs.iterator();
        while (it.hasNext()) {
            long itID = it.next();
            if (itID == id) {
                it.remove();
                break;
            }
        }
    }

    public Collection<Long> getAllPoseceni() {
        if (poseceniIDs.size() > KAPACITET) {
            return poseceniIDs.subList(0, KAPACITET);
        }
        return Collections.unmodifiableCollection(poseceniIDs);
    }
}
```

Sesije

Rešenje

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {

    :

    @GetMapping("")
    public String getAll(ModelMap request,
        :
    ) {
        request.addAttribute("filmovi", filmService.get(
            naziv,
            zanrId,
            trajanjeOd, trajanjeDo));
        request.addAttribute("zanrovi", zanrService.getAll());
        request.addAttribute("poseceniFilmovi", filmService.getPoseceni());
        return "filmovi"; // forwarding na template
    }

    @GetMapping("/prikaz")
    public String get(ModelMap request,
        @RequestParam long id) {
        request.addAttribute("film", filmService.get(id));
        request.addAttribute("zanrovi", zanrService.getAll());
        return "filmovi-prikaz"; // forwarding na template
    }

    :

    @PostMapping("/brisanje")
    public String delete(@RequestParam long id) {
        filmService.delete(id);
        return "redirect:/filmovi";
    }

}
```

```
@Service
@Validated
public class DatabaseFilmService implements FilmService {
    private final FilmDAO filmDAO;
    private final ZanrDAO zanrDAO;
    private final ModelMapper mapper = new ModelMapper();

    private FilmoviIstorija filmoviIstorija;

    public DatabaseFilmService(FilmDAO filmDAO, ZanrDAO zanrDAO, FilmoviIstorija filmoviIstorija) {
        this.filmDAO = filmDAO;
        this.zanrDAO = zanrDAO;
        this.filmoviIstorija = filmoviIstorija;
    }

    :

    @Override
    public FilmDTOGet get(long id) {
        Film film = filmDAO.get(id);
        if (film == null) {
            throw new NoSuchElementException("Film nije pronaden!");
        }
        filmoviIstorija.addPosecen(film.getId());
        return createdDTO(film);
    }

    @Override
    public void delete(long id) {
        filmDAO.delete(id);
        filmoviIstorija.deletePosecen(id);
    }

    :

    @Override
    public Collection<FilmDTOGet> getPoseceni() {
        Collection<Long> poseceniFilmoviIDs = filmoviIstorija.getAllPoseceni();

        List<Film> poseceniFilmovi = new ArrayList<>();
        for (Long itFilmID: poseceniFilmoviIDs) {
            poseceniFilmovi.add(filmDAO.get(itFilmID));
        }
        return createdDTO(poseceniFilmovi);
    }

}
```

Sesije

Rešenje

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <link rel="stylesheet" type="text/css" href="/webjars/bootstrap/css/bootstrap.min.css"/>
  <title>Filmovi</title>
</head>
<body>
  <div class="container-fluid">
    :
    </nav>
  </div>
</div>
<div class="list-group list-group-horizontal">
  <a
    class="list-group-item list-group-item-action list-group-item-primary text-center"
    th:each="itFilm: ${poseceniFilmovi}"
    th:href="|/filmovi/prikaz?id=${itFilm.id}|"
    th:text="${itFilm.naziv}"
  >film 1</a>
</div>
<div class="row">
  <div class="col">
    <form>
      <table class="table table-striped">
        :
      </table>
    </form>
  </div>
</div>
</body>
</html>
```

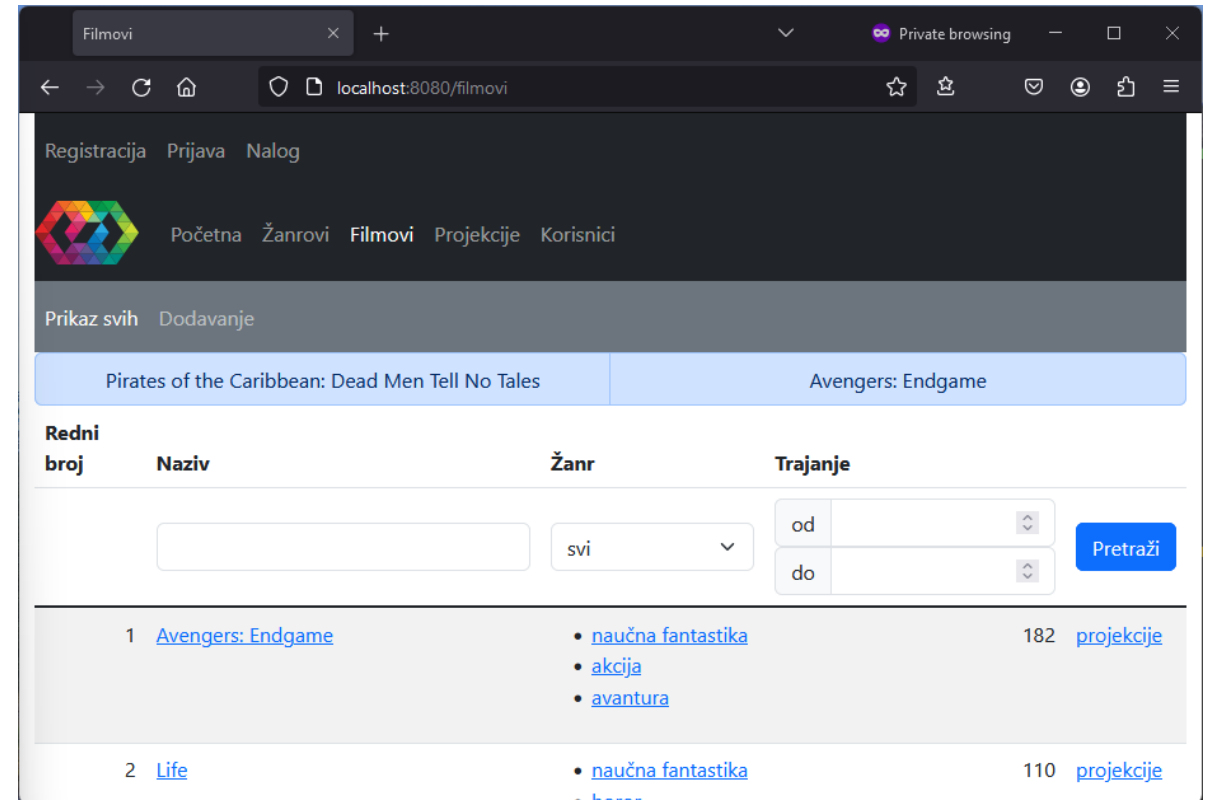
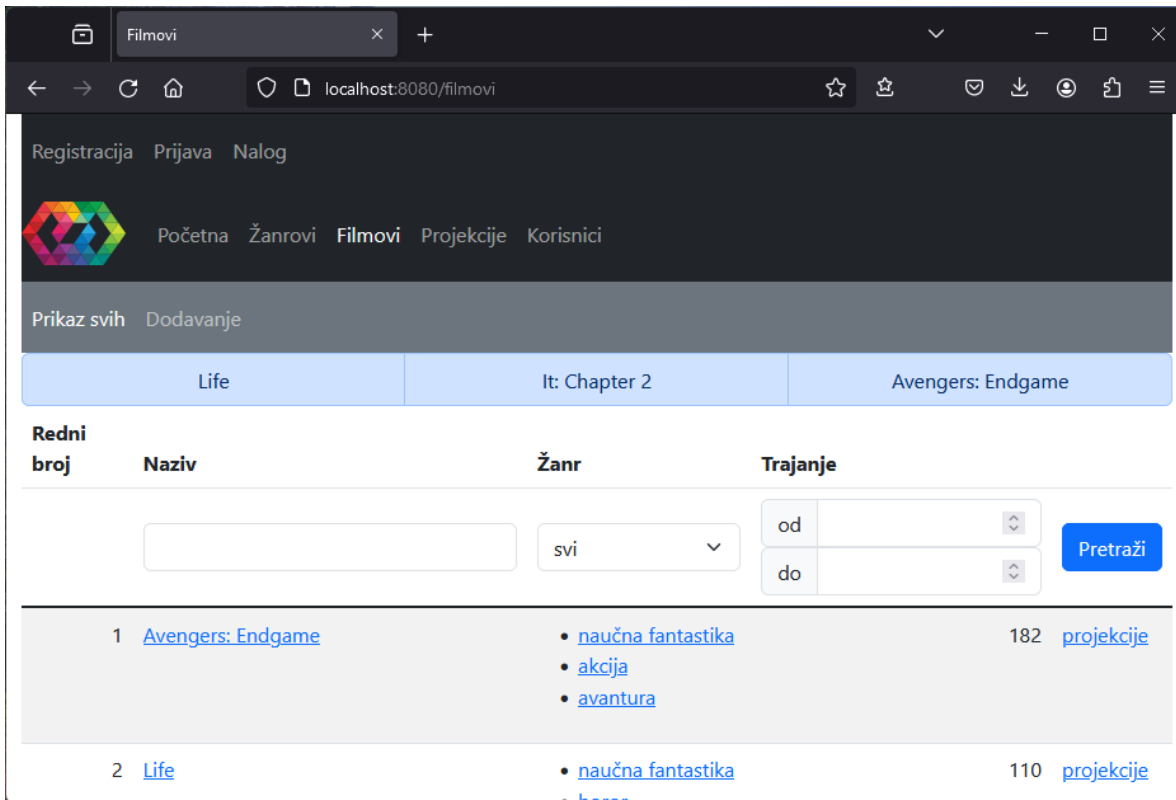
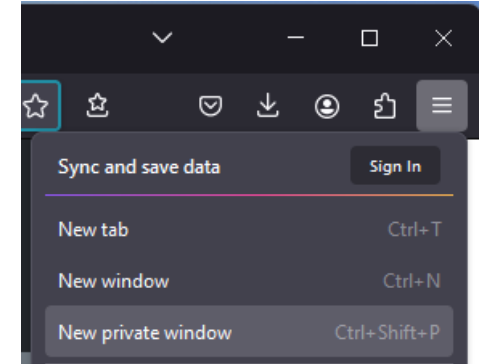
Primer

- *com.ftninformatika.jwd.modul2.termin8.bioskop*

Sesije

Problem

Kako za različite HTTP klijente prikazati istorijat posećenih filmova?



Sesije

Cookies

Zahtevi u HTTP protokolu su međusobno nezavisni (HTTP protokol je *stateless*).

Za određene funkcionalnosti, *web* aplikacija ima potrebu da razlikuje da li uzastopni zahtevi stižu od istog HTTP klijenta (*browser-a*) ili ne.

Uzastopni zahtevi od istog HTTP klijenta čine korisničku sesiju.

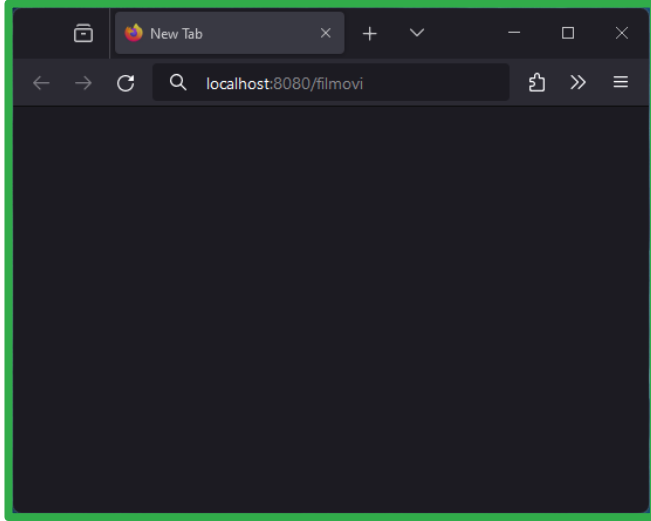
Korisnička sesija se identifikuje jedinstvenom *string* vrednošću koja se zove *cookie* (kolačić).

Cookies su mehanizam HTTP protokola koji omogućuje prepoznavanje HTTP klijenta koji se obraća web aplikaciji.

Ovaj mehanizam je implementiran na nivou HTTP klijenta (*browser-a*) i *server-a* (aplikacije), a ne protokola.

Sesije

Cookies

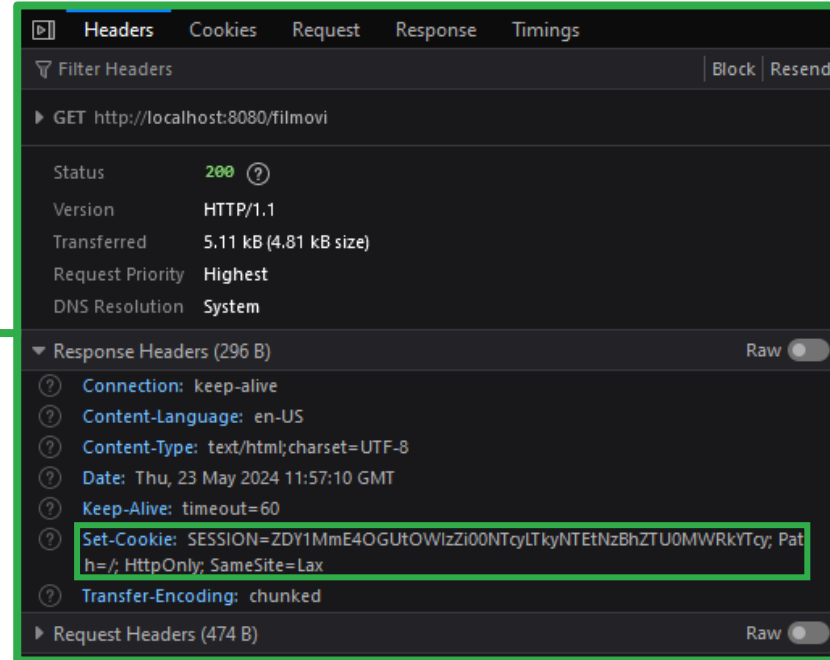
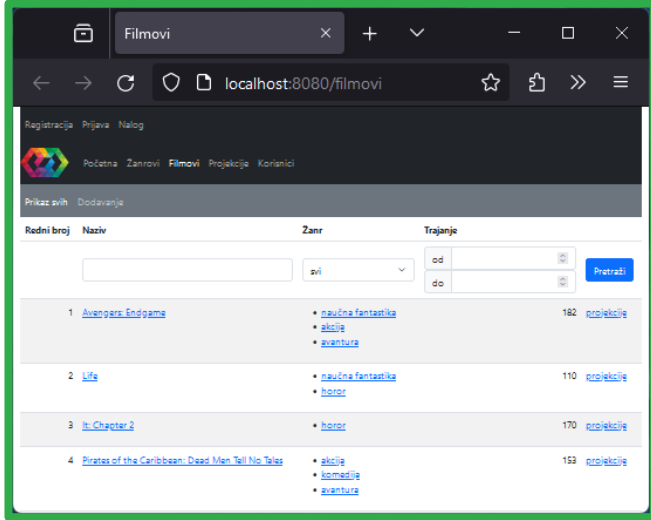


prvi zahtev

Spring

Sesije

Cookies



Spring

Session

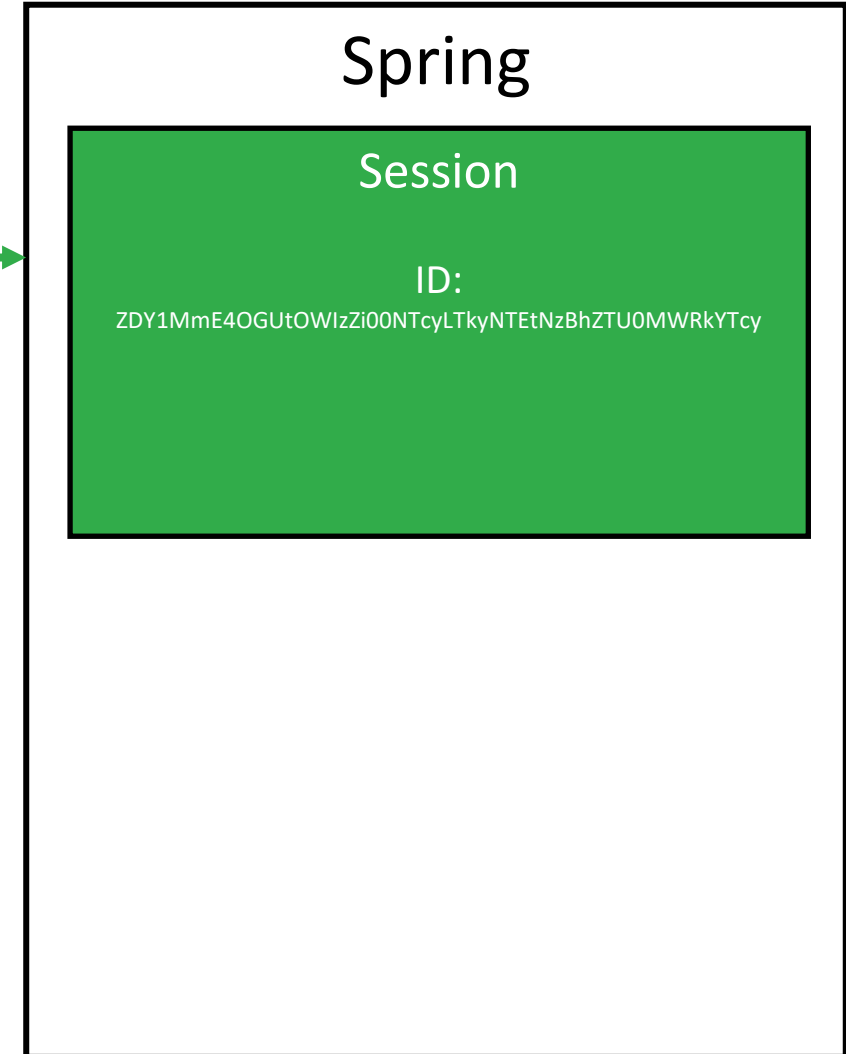
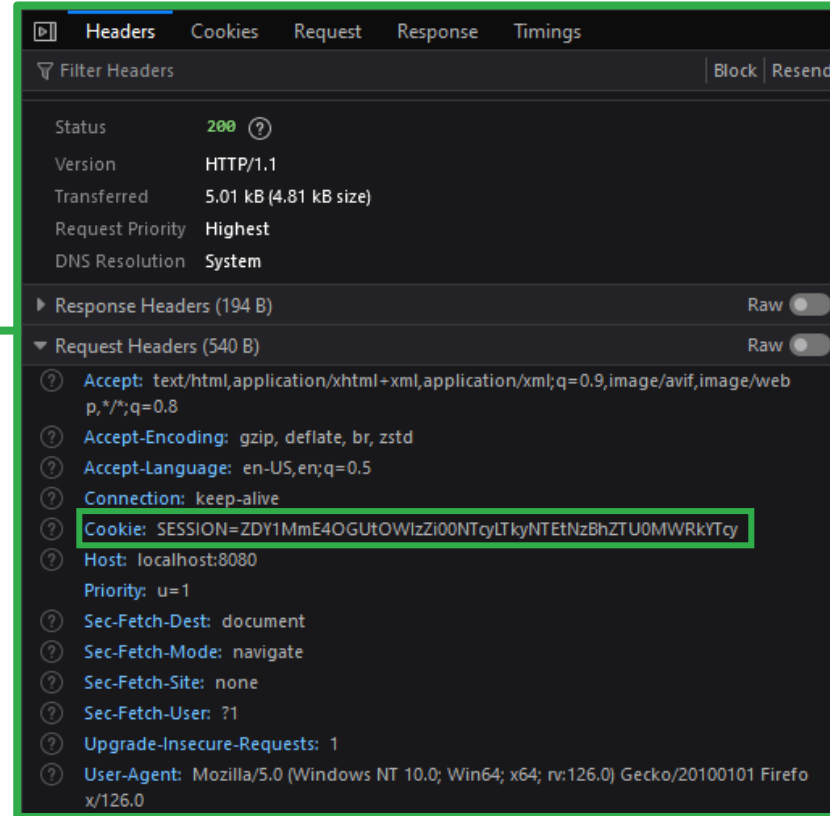
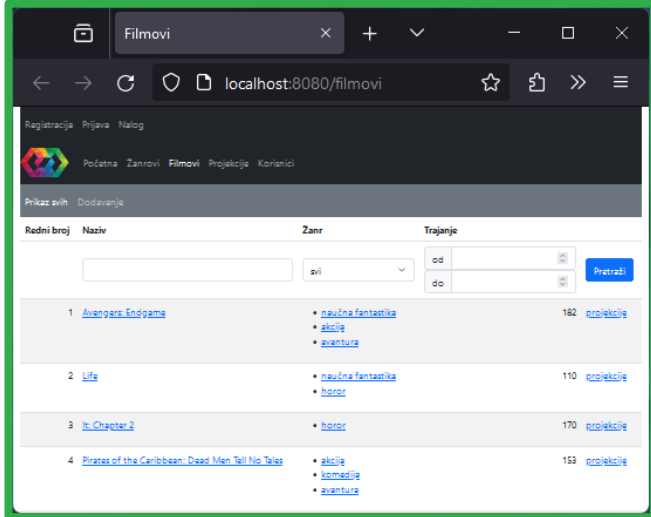
ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyNTEtNzBhZTU0MWRkYTcy

Set-Cookie atribut odgovora zadaje *cookie* koji će se koristiti u daljoj komunikaciji. Time HTTP klijent dobija svoj jedinstveni ID, kojim je dužan da identifikuje buduće zahteve.

Sesije

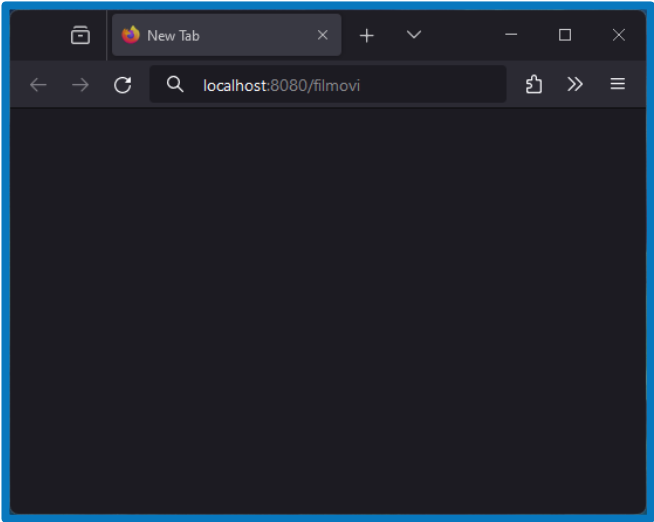
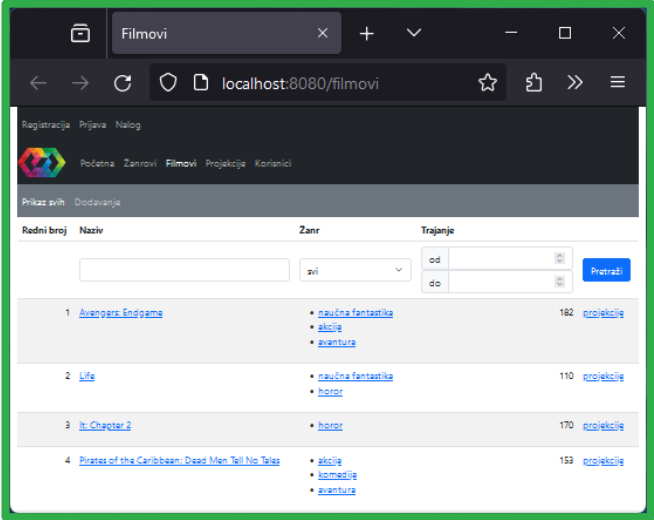
Cookies



Cookie atribut identifikuje svaki naredni zahtev sve dok aplikacija u odgovoru ponovo ne pošalje novi *Set-Cookie* atribut sa novom vrednošću (kada sesija istekne).

Sesije

Cookies



prvi zahtev

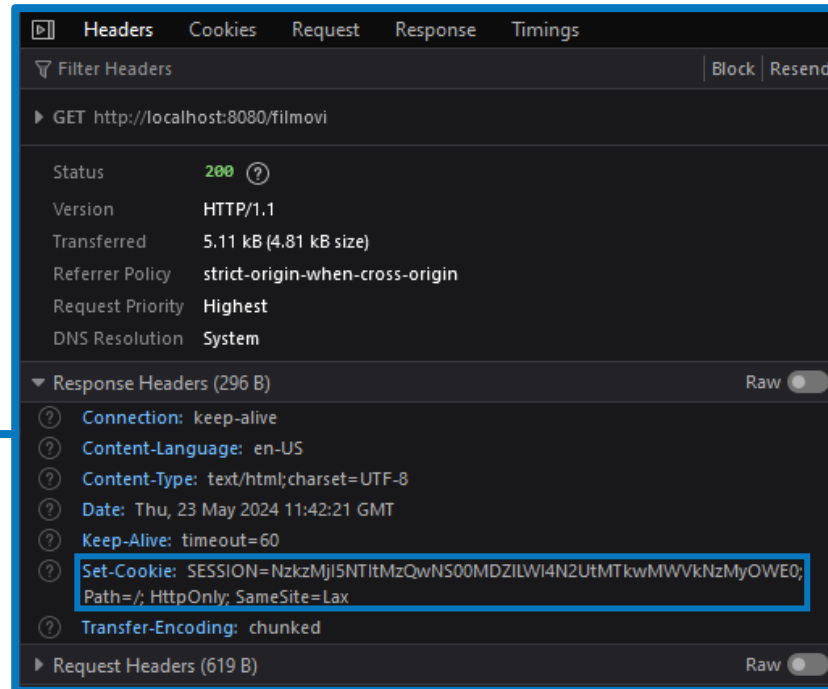
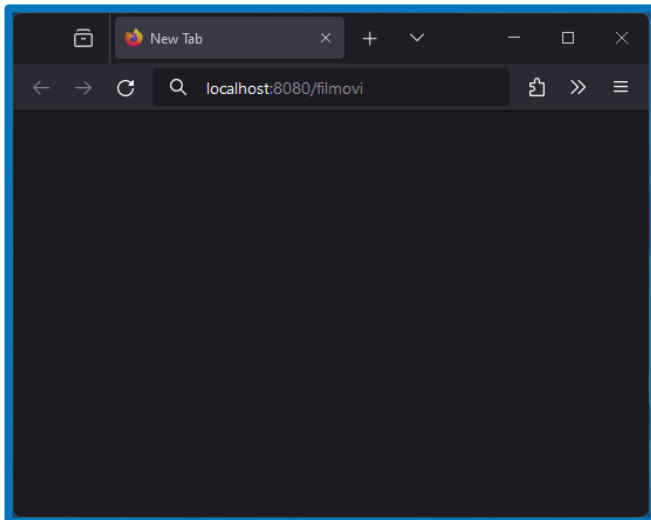
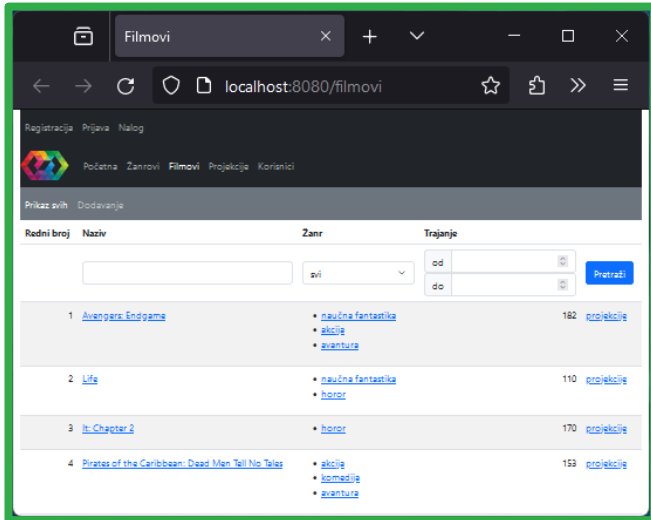
Spring

Session

ID:
ZDY1MmE4OGUtOWIzZi00NTcyLTkyNTEtNzBhZTU0MWRkYTcy

Sesije

Cookies



Spring

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyNTEtNzBhZTU0MWRkYTcy

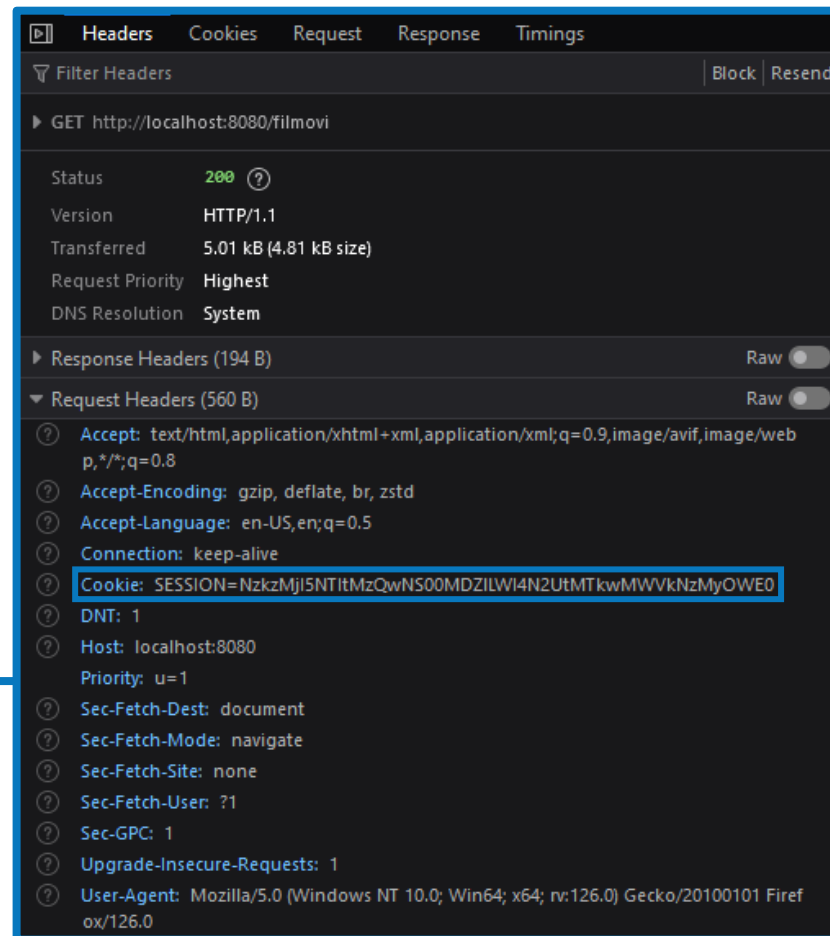
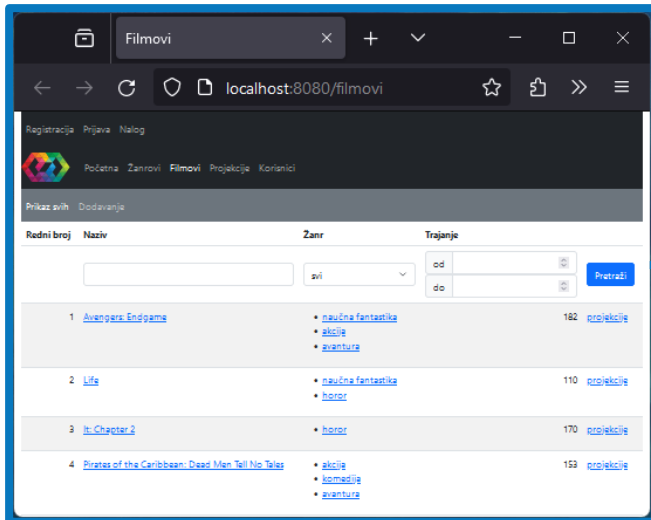
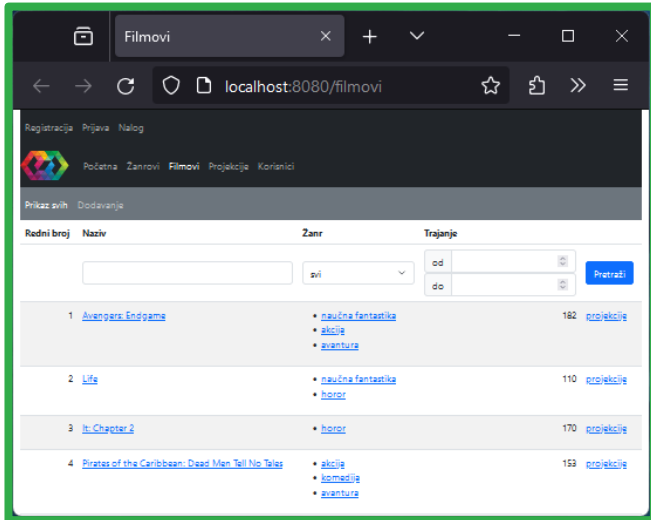
Session

ID:

NzgzMjl5NTItMzQwNS00MDZILWI4N2UtMTkwMWVKNzMyOWE0

Sesije

Cookies



Spring

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyNTEtNzBhZTU0MWRkYTcy

Session

ID:

NzgzMjl5NTItMzQwNS00MDZILWI4N2UtMTkwMWVKNzMyOWE0

Sesije

`@SessionScope` anotacija nalaže *Spring* aplikaciji da objekat komponente instancira za svakog HTTP klijenta, odnosno svaku sesiju pojedinačno.

Za svakog HTTP klijenta će na mestu upotrebe biti *inject*-ovan objekat komponente koji pripada njegovoj sesiji.

```
@Component
@SessionScope
public class FilmoviIstorijaBezSortiranja implements FilmoviIstorija {
    private static final int KAPACITET = 5;

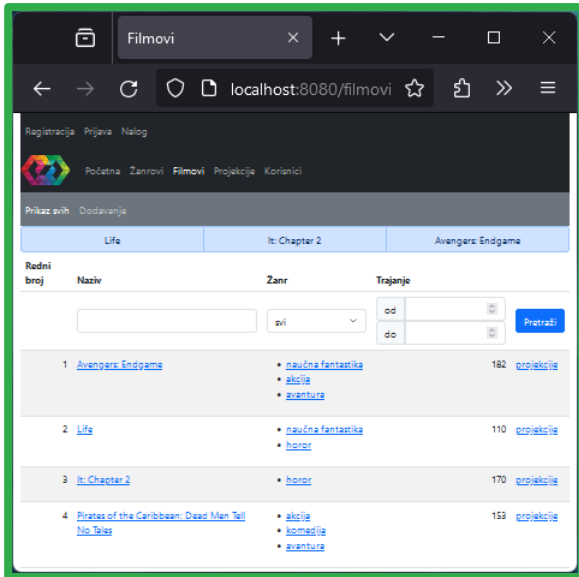
    private final List<Long> poseceniIDs = new ArrayList<Long>();

    public void addPosecen(Long id) {
        if (poseceniIDs.contains(id)) {
            return;
        }
        poseceniIDs.add(id);
    }

    public void deletePosecen(Long id) {
        Iterator<Long> it = poseceniIDs.iterator();
        while (it.hasNext()) {
            long itID = it.next();
            if (itID == id) {
                it.remove();
                break;
            }
        }
    }

    public Collection<Long> getAllPoseceni() {
        if (poseceniIDs.size() > KAPACITET) {
            return poseceniIDs.subList(0, KAPACITET);
        }
        return Collections.unmodifiableCollection(poseceniIDs);
    }
}
```


Sesije



```
@Service
@Validated
public class DatabaseFilmService implements FilmService {
    private final FilmDAO filmDAO;
    private final ZanrDAO zanrDAO;
    private final ModelMapper mapper = new ModelMapper();

    private FilmoviIstorija filmoviIstorija;

    public DatabaseFilmService(FilmDAO filmDAO, ZanrDAO zanrDAO, FilmoviIstorija filmoviIstorija) {
        this.filmDAO = filmDAO;
        this.zanrDAO = zanrDAO;
        this.filmoviIstorija = filmoviIstorija;
    }

    @Override
    public FilmDTOGet get(long id) {
        Film film = filmDAO.get(id);
        if (film == null) {
            throw new NoSuchElementException("Film nije pronaden!");
        }
        filmoviIstorija.addPosecen(film.getId());
        return createDTO(film);
    }

    @Override
    public void delete(long id) {
        filmDAO.delete(id);
        filmoviIstorija.deletePosecen(id);
    }

    @Override
    public Collection<FilmDTOGet> getPoseceni() {
        Collection<Long> poseceniFilmoviIDs = filmoviIstorija.getAllPoseceni();

        List<Film> poseceniFilmovi = new ArrayList<>();
        for (Long itFilmID: poseceniFilmoviIDs) {
            poseceniFilmovi.add(filmDAO.get(itFilmID));
        }
        return createDTO(poseceniFilmovi);
    }
}
```

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyN
TEtNzBhZTU0MWRkYTcy

FilmoviIstorija

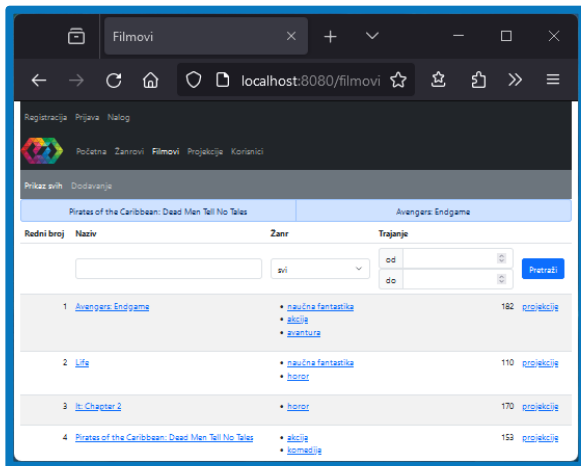
Session

ID:

NzgzMjI5NTItMzQwNS00MDZILWI4N
2UtMTkwMWVvKzMyOWE0

FilmoviIstorija

Sesije



```
@Service
@Validated
public class DatabaseFilmService implements FilmService {
    private final FilmDAO filmDAO;
    private final ZanrDAO zanrDAO;
    private final ModelMapper mapper = new ModelMapper();

    private FilmoviIstorija filmoviIstorija;

    public DatabaseFilmService(FilmDAO filmDAO, ZanrDAO zanrDAO, FilmoviIstorija filmoviIstorija) {
        this.filmDAO = filmDAO;
        this.zanrDAO = zanrDAO;
        this.filmoviIstorija = filmoviIstorija;
    }

    @Override
    public FilmDTOGet get(long id) {
        Film film = filmDAO.get(id);
        if (film == null) {
            throw new NoSuchElementException("Film nije pronaden!");
        }
        filmoviIstorija.addPosecen(film.getId());
        return createDTO(film);
    }

    @Override
    public void delete(long id) {
        filmDAO.delete(id);
        filmoviIstorija.deletePosecen(id);
    }

    @Override
    public Collection<FilmDTOGet> getPoseceni() {
        Collection<Long> poseceniFilmoviIDs = filmoviIstorija.getAllPoseceni();

        List<Film> poseceniFilmovi = new ArrayList<>();
        for (Long itFilmID: poseceniFilmoviIDs) {
            poseceniFilmovi.add(filmDAO.get(itFilmID));
        }
        return createDTO(poseceniFilmovi);
    }
}
```

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyN
TEtNzBhZTU0MMWRkYTcy

FilmoviIstorija

Session

ID:

NzkzMjl5NTItMzQwNS00MDZILWl4N
2UtMTkwMWVvKzMyOWE0

FilmoviIstorija

Sesije

Perzistencija

Sve sesije u aplikaciji postoje dok ne isteknu ili dok se aplikacija ne restartuje.

Stanja sesija se mogu trajno čuvati u bazi podataka.

Konfigurisano vreme isteka briše sesiju čak i iz baze podataka.

U prvom sledećem obraćanju HTTP klijenta nakon isteka sesije, aplikacija generiše novi *cookie*.

- *spring-session-jdbc* je biblioteka za automatsko čuvanje sesija u relacionoj bazi podataka i potrebno ju je uvesti *dependency* u *pom.xml*
- konfiguracioni parametri sesija se navode u *application.properties* datoteci.

```
<dependency>
  <groupId>org.springframework.session</groupId>
  <artifactId>spring-session-jdbc</artifactId>
  <scope>runtime</scope>
</dependency>
```

spring.session.store-type=jdbc	←	sesija se čuva u relacionoj bazi podataka (moguće su i druge)
spring.session.jdbc.initialize-schema=always	←	automatska inicijalizacija potrebnih tabela u šemi ako ne postoje
spring.session.timeout=900	←	vreme trajanja sesije nakon koga će biti obrisana i kreirana nova

Sesije

Perzistencija

Sve komponente koje se čuvaju u sesiji moraju implementirati interfejs *Serializable*. Ovo dozvoljava *Java* virtualnoj mašini da automatski upiše stanje objekata u bazu u binarnom obliku (*serialization*) i da ih pri čitanju ponovo kreira u memoriji (*deserialization*).

Svi izvedeni atributi komponenata, njihovi izvedeni atributi, itd., iz istog razloga moraju takođe implementirati interfejs *Serializable*.

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/Serializable.html>

```
@Component
@SessionScope
public class FilmoviIstorijaBezSortiranja implements Serializable, FilmoviIstorija {

    private static final long serialVersionUID = -2040018225168833511L;

    :

}
```

Sesije

Perzistencija

Sve komponente koje se čuvaju u sesiji moraju implementirati interfejs *Serializable*. Ovo dozvoljava *Java* virtualnoj mašini da automatski upiše stanje objekata u bazu u binarnom obliku (*serialization*) i da ih pri čitanju ponovo kreira u memoriji (*deserialization*).

Svi izvedeni atributi komponenata, njihovi izvedeni atributi, itd., iz istog razloga moraju takođe implementirati interfejs *Serializable*.

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/io/Serializable.html>

```
@Component
@SessionScope
@SuppressWarnings("serial")
public class FilmoviIstorijaBezSortiranja implements Serializable, FilmoviIstorija {

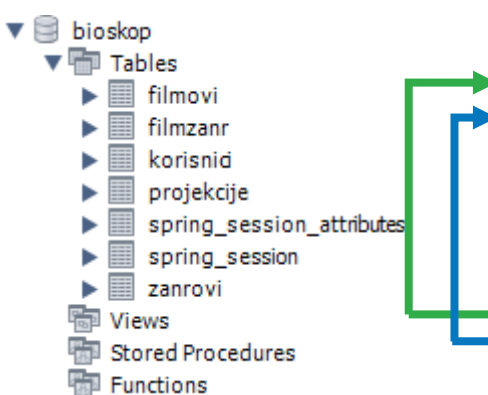
    :

}
```

Sesije

Perzistencija

- *spring_session tabela* čuva podatke o sesijama
- *spring_session_attributes* tabela čuva podatke o atributima (*bean*-ovima) u sesijama.



spring_session

PRIMARY_ID	SESSION_ID	CREATION_TIME	LAST_ACCESS_TIME	MAX_INACTIVE_INTERVAL	EXPIRY_TIME	PRINCIPAL_NAME
6b12770a-f933-4b35-a59e-dbb0357264e0	9c6f0485-283b-4c79-be36-2a21806e2c79	1716552545810	1716552546048	900	1716553446048	NULL
eedaaff5-645c-4f3a-9de2-2acf7ab1cb6e	e0b08b86-8d95-4164-8368-fbb8b5d832ea	1716552564278	1716552564320	900	1716553464320	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

spring_session_attributes

SESSION_PRIMARY_ID	ATTRIBUTE_NAME	ATTRIBUTE_BYTES
6b12770a-f933-4b35-a59e-dbb0357264e0	scopedTarget.filmoviIstorija	BLOB
eedaaff5-645c-4f3a-9de2-2acf7ab1cb6e	scopedTarget.filmoviIstorija	BLOB
NULL	NULL	NULL

Beans

Beans su osnovna gradivna jedinica *Spring* aplikacije (ono što je podložno DI mehanizmu).

Komponente spadaju u *Beans*.

Postoje 2 načina konfiguracije komponenata:

- *@Component* anotacijom nad klasom komponente
- *@Bean* anotacijom nad metodom konfiguracione klase koja je kreira

Konfiguracione klase su klase čije su metode zadužene za konfigurisanje *bean*-ova. One se obeležavaju anotacijom *@Configuration*.

Konfiguracione klase se tipično koriste onda kada komponente u aplikaciji nisu *1st-party* porekla, odnosno kada ne postoji *source* kod klase komponente koja bi bila obeležena anotacijom *@Component*.

```
@Component
@SessionScope
public class FilmoviIstorijaBezSortiranja
    implements Serializable, FilmoviIstorija {
    :
}
```

2 načina
konfiguracije
komponente
(ne koristiti
oba istovremeno)

```
@Configuration
public class SessionConfig {
    @Bean
    @SessionScope
    public FilmoviIstorija filmoviIstorija() {
        return new FilmoviIstorijaBezSortiranja();
    }
    :
}
```

Beans

Scopes

Beans definišu različite opsege važenja (*scopes*):

- *singleton* (podrazumevani *scope*, ne obeležava se posebnom anotacijom): jedna instanca objekta *bean* klase će biti kreirana na nivou cele aplikacije i inject-ovana na mestu upotrebe
- *session* (obeležava se anotacijom *@SessionScope*): jedna instanca objekta *bean* klase će biti kreirana za svakog HTTP klijenta posebno (za svaku sesiju) i *inject*-ovana na mestu upotrebe
- *request* (obeležava se anotacijom *@RequestScope*): jedna instanca objekta *bean* klase će biti kreirana za svaki HTTP zahtev posebno i *inject*-ovana na mestu upotrebe (naročito pogodno ako se zahtev *forward*-uje kroz više resursa)
- i drugi ...

<https://docs.spring.io/spring-framework/reference/core/beans/factory-scopes.html>

Primer

- *com.ftninformatika.jwd.modul2.termin8.bioskop*

Zadatak 1

Po ugledu na primer *com.ftninformatika.jwd.modul2.termin8.bioskop* implementirati spisak posećenih artikala u zadatku *com.ftninformatika.jwd.modul2.termin8.dostava*:

1. Navesti *spring-session-jdbc dependency* u datoteci *pom.xml*
2. Navesti konfiguracione parametre za sesija u datoteci *application.properties*
3. Definisati interfejs komponente *Artiklilstorija* u paketu *model.session*
4. Implementirati interfejs *Artiklilstorija* u komponenti *ArtiklilstorijaBezSortiranja* u paketu *model.session.impl* i navesti opseg važenja *@SessionScope*
5. Dopisati metodu *Collection<ArtikalDTOGet> getPoseceni()* u iterfejsu *ArtikalService*
6. *Inject*-ovati interfejs komponente *Artiklilstorija* u servis *DatabaseArtikalService*
7. Dopuniti metode *get(long id)* i *delete(long id)* servisa da ažuriraju stanje komponente *Artiklilstorija*
8. Implementirati metodu *getPoseceni()* u servisu *DatabaseArtikalService* tako da vraća stanje komponente *Artiklilstorija*
9. Pozvati metodu *getPoseceni()* servisa u *get(...)* metodi *ArtikliController controller*-a
10. Dopuniti template *artikli.html* tako da se u njemu vrši prikaz posećenih artikala i testirati funkcionalnost u 2 različite sesije

Zadatak 2

U zadatku *com.ftninformatika.jwd.modul2.termin8.dostava* napraviti novu implementaciju interfejsa *Artiklilstorija* koja vrši i sortiranje posećenih artikala po opadajućem redosledu broja pristupa:

1. Implementirati interfejs *Artiklilstorija* u komponenti *ArtiklilstorijaSortiranoPoBrojuPristupa* u paketu *model.session.impl* i navesti opseg važenja *@SessionScope*
2. Aktivirati komponentu anotacijom *@Primary*
3. U metodama komponente implementirati potrebnu logiku

Sistem naloga

Problem

Kako pristup nekim resursima aplikacije (npr. spisku svih korisnika) dozvoliti samo određenim korisnicima (administratorima)?

Kako neke funkcionalnosti aplikacije (npr. dodavanje, izmena, brisanje) dozvoliti samo određenim korisnicima (administratorima)?

Kako novim korisnicima omogućiti da se registruju na sistem bez učešća administratora?

Sistem naloga

Rešenje

1. Napraviti dodatne funkcionalnosti:

- Registracija (dodavanje korisnika bez učešća administratora)
- Pikaz i izmena sopstvenih podataka korisnika bez učešća administratora
- Prijava (utvrđivanje autentičnosti korisnika i čuvanje traga o tome u sesiji)
- Odjava (brisanje traga o autentičnosti korisnika iz sesije)

2. Pri svakom zahtevu koji bi trebalo da bude zaštićen od neovlašćenog pristupa proveriti da li u sesiji postoji trag o autentičnosti korisnika (autentikacija), a zatim proveriti da li taj korisnik ima prava da obavlja operaciju koju definiše taj zahtev (autorizacija):

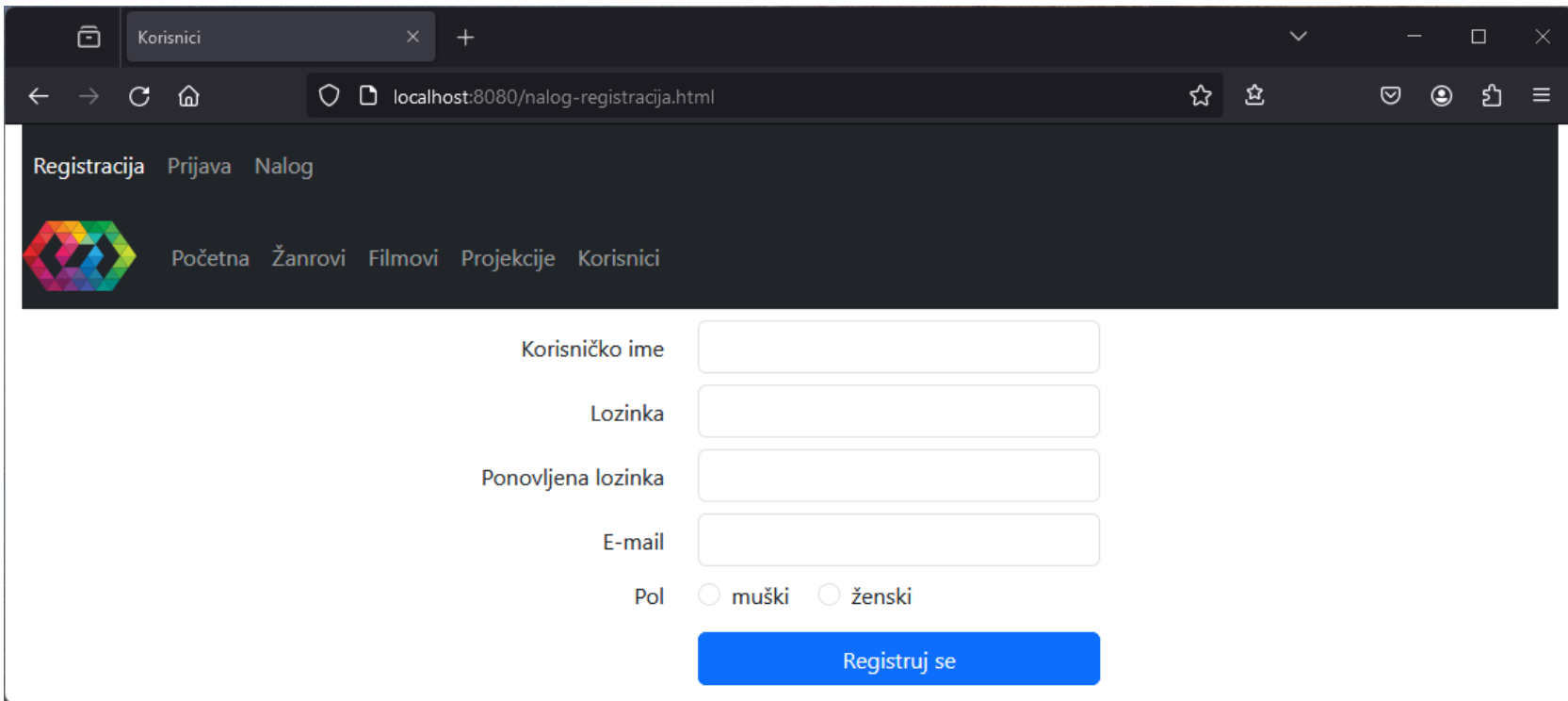
- da li je autentični korisnik administrator?
- u slučaju prikaza i izmene svojih podataka, da li je autentični korisnik baš onaj čiji se podaci prikazuju i menjaju?

Sistem naloga

Registracija

Registracija je operacija dodavanja korisnika bez učešća administratora.

Korisnik mora navesti korisničko ime koje nije prethodno postojalo u sistemu.



Korisnici

localhost:8080/nalog-registracija.html

Registracija Prijava Nalog

Početna Žanrovi Filmovi Projekcije Korisnici

Korisničko ime

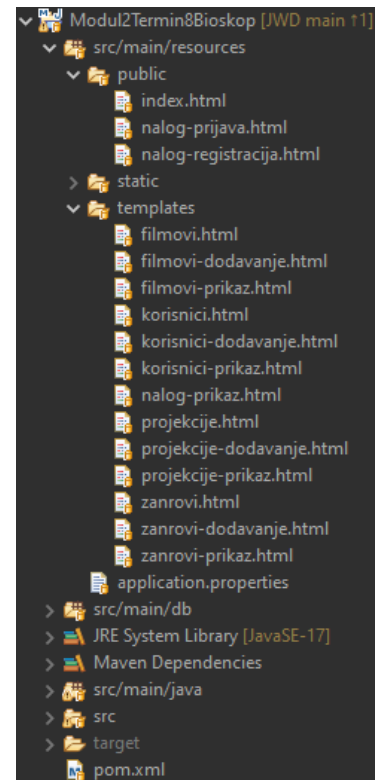
Lozinka

Ponovljena lozinka

E-mail

Pol ☐ muški ☐ ženski

Registruj se

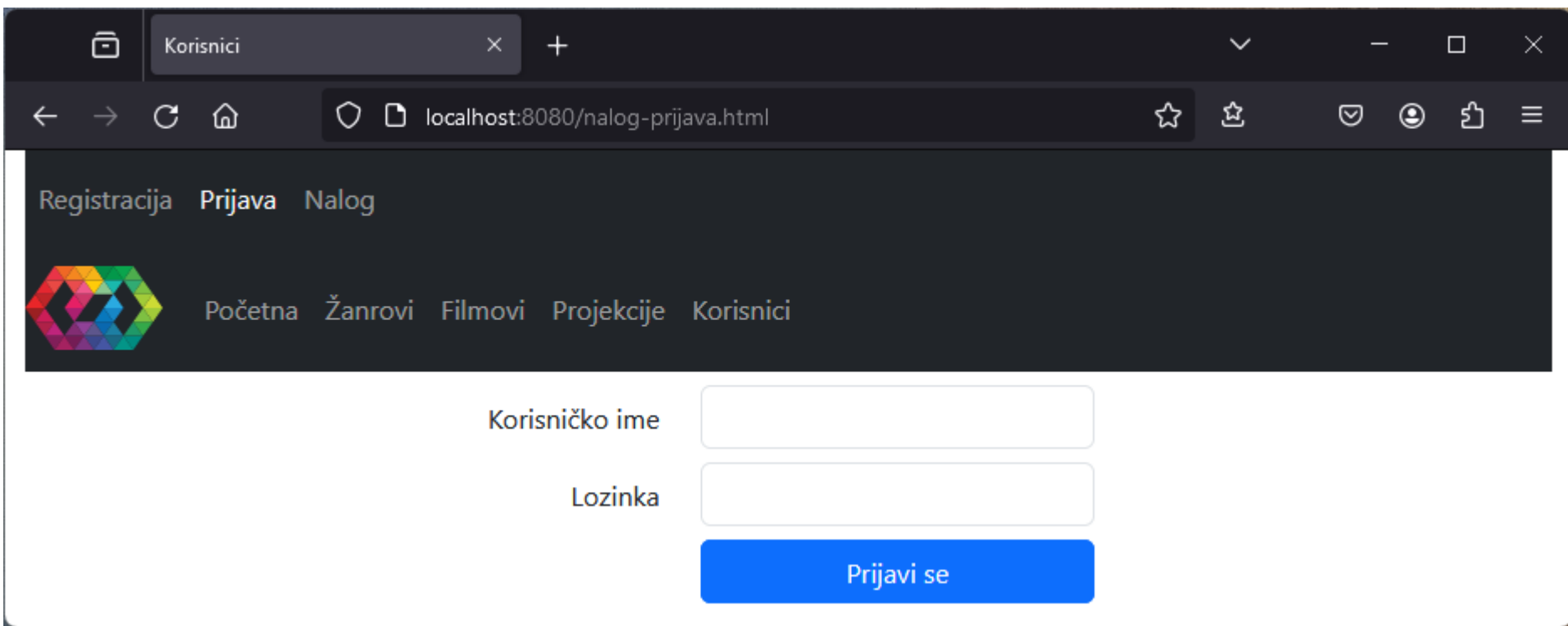


Sistem naloga

Prijava

Prijava je operacija utvrđivanja autentičnosti korisnika i čuvanje traga o tome u HTTP sesiji klijenta.

Korisnik mora da poznaje svoje korisničko ime i lozinku da bi njegov identitet bio potvrđen.



Korisnici

localhost:8080/nalog-prijava.html

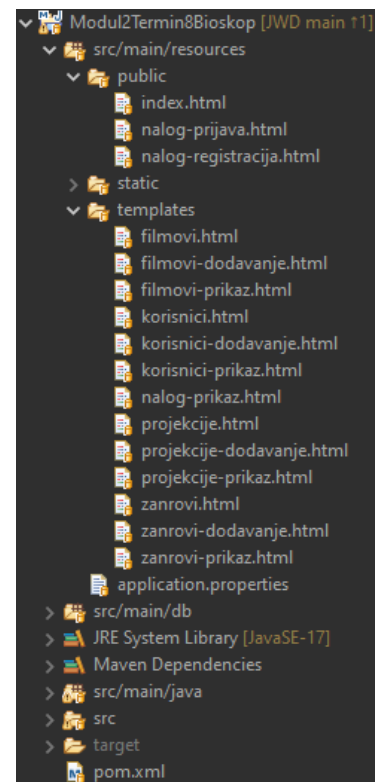
Registracija Prijava Nalog

Početna Žanrovi Filmovi Projekcije Korisnici

Korisničko ime

Lozinka

Prijavi se



Sistem naloga

Prikaz i izmena sopstvenih podataka korisnika

Korisnik može da menja sve podatke koje je zadao prilikom registracije, osim korisničkog imena.

Korisnici

localhost:8080/nalog/prikaz

Registracija Prijava Nalog

Početna Žanrovi Filmovi Projekcije Korisnici

Korisničko ime a

Lozinka

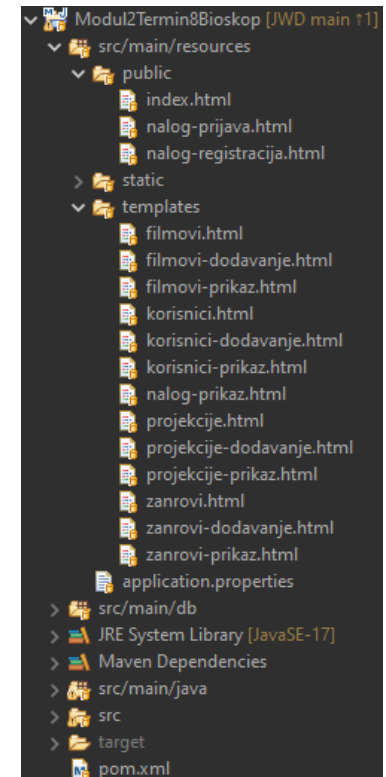
Ponovljena lozinka

E-mail a@a.com

Pol ☒ muški ☐ ženski

Izmeni

Odjavi se



Sistem naloga

Odjava

Odjava je operacija brisanja traga o autentičnosti u sesiji HTTP klijenta.

Korisnici

localhost:8080/nalog/prikaz

Registracija Prijava Nalog

Početna Žanrovi Filmovi Projekcije Korisnici

Korisničko ime a

Lozinka

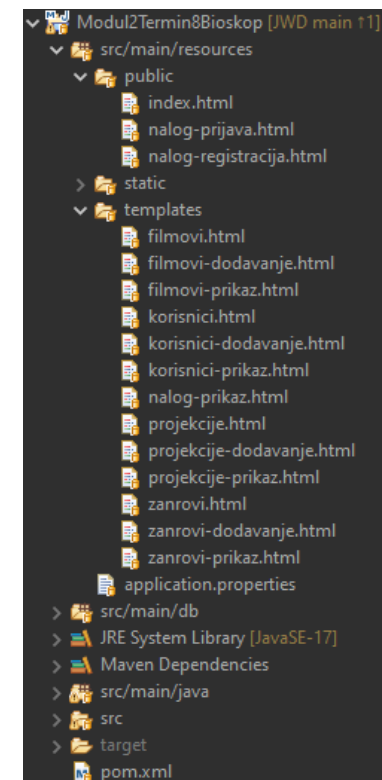
Ponovljena lozinka

E-mail a@a.com

Pol ☒ muški ☐ ženski

Izmeni

Odjavi se



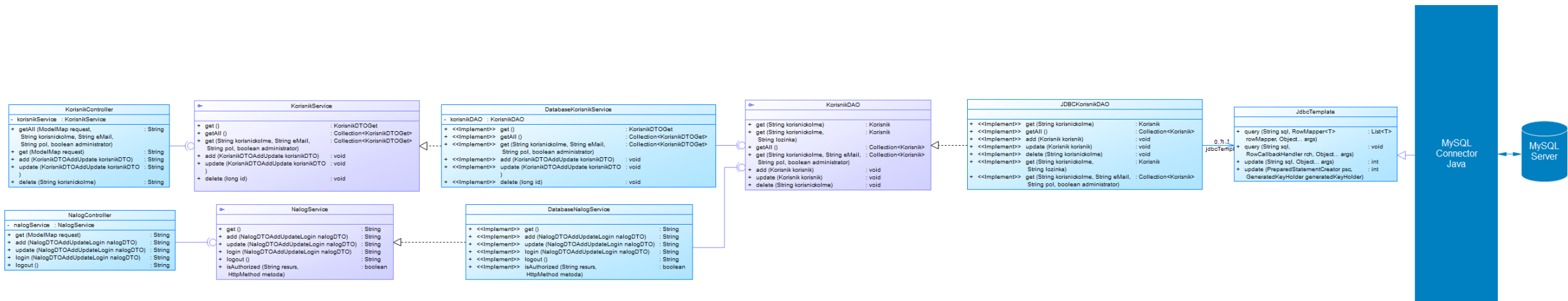
Sistem naloga

Arhitektura

Iako se u bazi oslanjaju na iste podatke, treba razdvojiti funkcionalnosti:

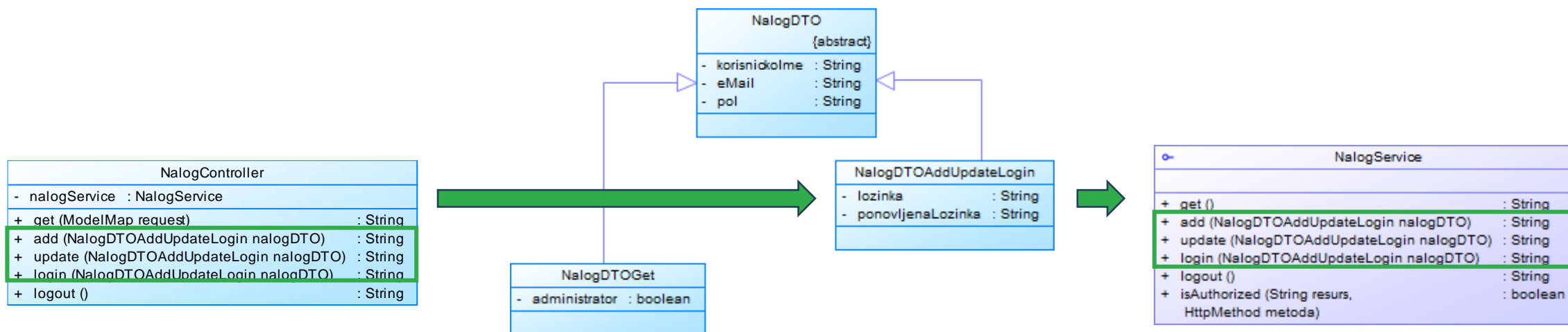
- koje administratorima omogućuju prikaz, dodavanje, izmenu i brisanje korisnika (*KorisnikController*, *KorisnikService*)
- koje korisnicima omogućuju registraciju, izmenu svog naloga, prijavu, odjavu, autentikaciju i autorizaciju (*NalogController*, *NalogService*)

Iako su na prvi pogled slične, logike u nekim od ovih operacija se suštinski razlikuju i može doći do njihovog komplikovanja i čak i bezbedonosnih propusta ako bi bile obuhvaćene istim *controller*-om, odnosno servisom.



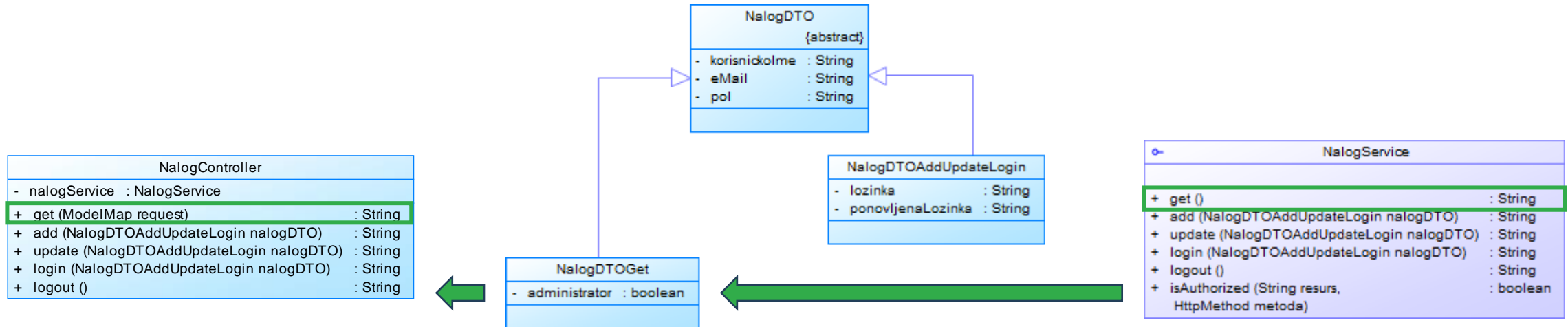
Sistem naloga

DTO



Sistem naloga

DTO



Sistem naloga

Rešenje

```
@Component
@SessionScope
@SuppressWarnings("serial")
public class Prijava implements Serializable {
    private String korisnickoIme;

    public String getKorisnickoIme() {
        return korisnickoIme;
    }

    public void setKorisnickoIme(String korisnickoIme) {
        this.korisnickoIme = korisnickoIme;
    }
}
```

Sistem naloga

Rešenje

```
@Service
@Validated
public class DatabaseNalogService implements NalogService {
    private final KorisnikDAO korisnikDAO;
    private final ModelMapper mapper = new ModelMapper();

    private Prijava prijava; // session

    public DatabaseNalogService(KorisnikDAO korisnikDAO, Prijava prijava) {
        this.korisnikDAO = korisnikDAO;
        this.prijava = prijava;
    }

    :

    @Override
    public void login(NalogDTOAddUpdateLogin nalogDTO) {
        Korisnik korisnik = korisnikDAO.get(
            nalogDTO.getKorisnickoIme(),
            nalogDTO.getLozinka());
        if (korisnik == null) {
            throw new IllegalArgumentException("Neuspešna prijava!");
        }
        prijava.setKorisnickoIme(korisnik.getKorisnickoIme()); // session
    }

    @Override
    public void logout() {
        prijava.setKorisnickoIme(null); // session
    }

    :
}
```

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyN
TEtNzBhZTU0MWRkYTcy

Filmovilstorija

Prijava:

- korisnickolme: null

Session

ID:

NzkzMjl5NTItMzQwNS00MDZILWI4N
2UtMTkwMWVvKnZMyOWE0

Filmovilstorija

Prijava:

- korisnickolme: null

Sistem naloga

Rešenje

Korisnici

Registracija Prijava Nalog

Početna Žanrovi Filmovi Projekcije Korisnici

Korisničko ime: a

Lozinka: •

Prijava

```
@Service
@Validated
public class DatabaseNalogService implements NalogService {
    private final KorisnikDAO korisnikDAO;
    private final IMapper mapper = new IMapper();

    private Prijava prijava; // session

    public DatabaseNalogService(KorisnikDAO korisnikDAO, Prijava prijava) {
        this.korisnikDAO = korisnikDAO;
        this.prijava = prijava;
    }

    @Override
    public void login(NalogDTOAddUpdateLogin nalogDTO) {
        Korisnik korisnik = korisnikDAO.get(
            nalogDTO.getKorisnickoIme(),
            nalogDTO.getLozinka());
        if (korisnik == null) {
            throw new IllegalArgumentException("Neuspešna prijava!");
        }
        prijava.setKorisnickoIme(korisnik.getKorisnickoIme()); // session
    }

    @Override
    public void logout() {
        prijava.setKorisnickoIme(null); // session
    }
}
```

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyN
TEtNzBhZTU0MWRkYTcy

Filmovistorija

Prijava:

- korisnickolme: a

Session

ID:

NzkzMjI5NTItMzQwNS00MDZILWI4N
2UtMTkwMWVvKnZMyOWE0

Filmovistorija

Prijava:

- korisnickolme: null

Sistem naloga

Rešenje

Korisnici

Registracija Prijava Nalog

Početna Žanrovi Filmovi Projekcije Korisnici

Korisničko ime: b

Lozinka: •

Prijava

```
@Service
@Validated
public class DatabaseNalogService implements NalogService {
    private final KorisnikDAO korisnikDAO;
    private final ModelMapper mapper = new ModelMapper();

    private Prijava prijava; // session

    public DatabaseNalogService(KorisnikDAO korisnikDAO, Prijava prijava) {
        this.korisnikDAO = korisnikDAO;
        this.prijava = prijava;
    }

    :

    @Override
    public void login(NalogDTOAddUpdateLogin nalogDTO) {
        Korisnik korisnik = korisnikDAO.get(
            nalogDTO.getKorisnickoIme(),
            nalogDTO.getLozinka());
        if (korisnik == null) {
            throw new IllegalArgumentException("Neuspešna prijava!");
        }
        prijava.setKorisnickoIme(korisnik.getKorisnickoIme()); // session
    }

    @Override
    public void logout() {
        prijava.setKorisnickoIme(null); // session
    }

    :
}
```

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyN
TEtNzBhZTU0MWRkYTcy

Filmovilstorija

Prijava:

- korisnickolme: a

Session

ID:

NzkzMjl5NTItMzQwNS00MDZILWI4N
2UtMTkwMWVvKnZMyOWEO

Filmovilstorija

Prijava:

- korisnickolme: b

Sistem naloga

Rešenje

Registracija Prijava Nalog

Početna Žanrovi Filmovi Projekcije Korisnici

Korisničko ime a

Lozinka

Ponovljena lozinka

E-mail a@a.com

Pol ☒ muški ☐ ženski

Izmeni

Odlog se

```
@Service
@Validated
public class DatabaseNalogService implements NalogService {
    private final KorisnikDAO korisnikDAO;
    private final ModelMapper mapper = new ModelMapper();

    private Prijava prijava; // session

    public DatabaseNalogService(KorisnikDAO korisnikDAO, Prijava prijava) {
        this.korisnikDAO = korisnikDAO;
        this.prijava = prijava;
    }

    :

    @Override
    public void login(NalogDTOAddUpdateLogin nalogDTO) {
        Korisnik korisnik = korisnikDAO.get(
            nalogDTO.getKorisnickoIme(),
            nalogDTO.getLozinka());
        if (korisnik == null) {
            throw new IllegalArgumentException("Neuspešna prijava!");
        }
        prijava.setKorisnickoIme(korisnik.getKorisnickoIme());
    }

    @Override
    public void logout() {
        prijava.setKorisnickoIme(null); // session
    }

    :
}
```

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyN
TEtNzBhZTU0MWRkYTcy

Filmovilstorija

Prijava:

- korisnickolme: null

Session

ID:

NzkzMjI5NTItMzQwNS00MDZILWI4N
2UtMTkwMWVvKnZMyOWEO

Filmovilstorija

Prijava:

- korisnickolme: b

Sistem naloga

Rešenje

Korisničko ime b

Lozinka

Ponovljena lozinka

E-mail b@b.com

Pol ☐ muški ☒ ženski

Izmeni

Očisti se

```
@Service
@Validated
public class DatabaseNalogService implements NalogService {
    private final KorisnikDAO korisnikDAO;
    private final ModelMapper mapper = new ModelMapper();

    private Prijava prijava; // session

    public DatabaseNalogService(KorisnikDAO korisnikDAO, Prijava prijava) {
        this.korisnikDAO = korisnikDAO;
        this.prijava = prijava;
    }

    :

    @Override
    public void login(NalogDTOAddUpdateLogin nalogDTO) {
        Korisnik korisnik = korisnikDAO.get(
            nalogDTO.getKorisnickoIme(),
            nalogDTO.getLozinka());
        if (korisnik == null) {
            throw new IllegalArgumentException("Neuspešna prijava!");
        }
        prijava.setKorisnickoIme(korisnik.getKorisnickoIme()); // session
    }

    @Override
    public void logout() {
        prijava.setKorisnickoIme(null); // session
    }

    :
}
```

Session

ID:

ZDY1MmE4OGUtOWIzZi00NTcyLTkyN
TEtNzBhZTU0MWRkYTcy

Filmovilstorija

Prijava:

- korisnickolme: null

Session

ID:

NzkzMjI5NTItMzQwNS00MDZILWI4N
2UtMTkwMWVvKzMyOWE0

Filmovilstorija

Prijava:

- korisnickolme: null

Sistem naloga

HandlerInterceptor

HandlerInterceptor (presretač obrađivača zahteva) je interfejs čija se metoda *preHandle(...)* poziva pri HTTP zahtevima ka resursima koje je zadužen da presreće, pre nego što zahtev biva prosleđen njima.

WebMvcConfigurer je interfejs čija metoda *addInterceptors(...)* omogućuje konfiguraciju *interceptor*-a.

InterceptorRegistry je klasa čija metoda *addInterceptor(...)* omogućava registraciju *interceptor*-a, tj. odabir URL nastavaka ka resursima do kojih će zahtevi biti presretani.

```
@Configuration
public class InterceptorConfig implements WebMvcConfigurer {
    private final AutorizacijaInterceptor autorizacijaInterceptor;

    public InterceptorConfig(AutorizacijaInterceptor autorizacijaInterceptor) {
        this.autorizacijaInterceptor = autorizacijaInterceptor;
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(authorizacijaInterceptor).addPathPatterns(
            "/korisnici/**",
            "/zanrovi/**",
            "/filmovi/**",
            "/projekcije/**");
    }
}
```

inject

```
@Component
public class AutorizacijaInterceptor implements HandlerInterceptor {
    private final NalogService nalogService;

    public AutorizacijaInterceptor(NalogService nalogService) {
        this.nalogService = nalogService;
    }

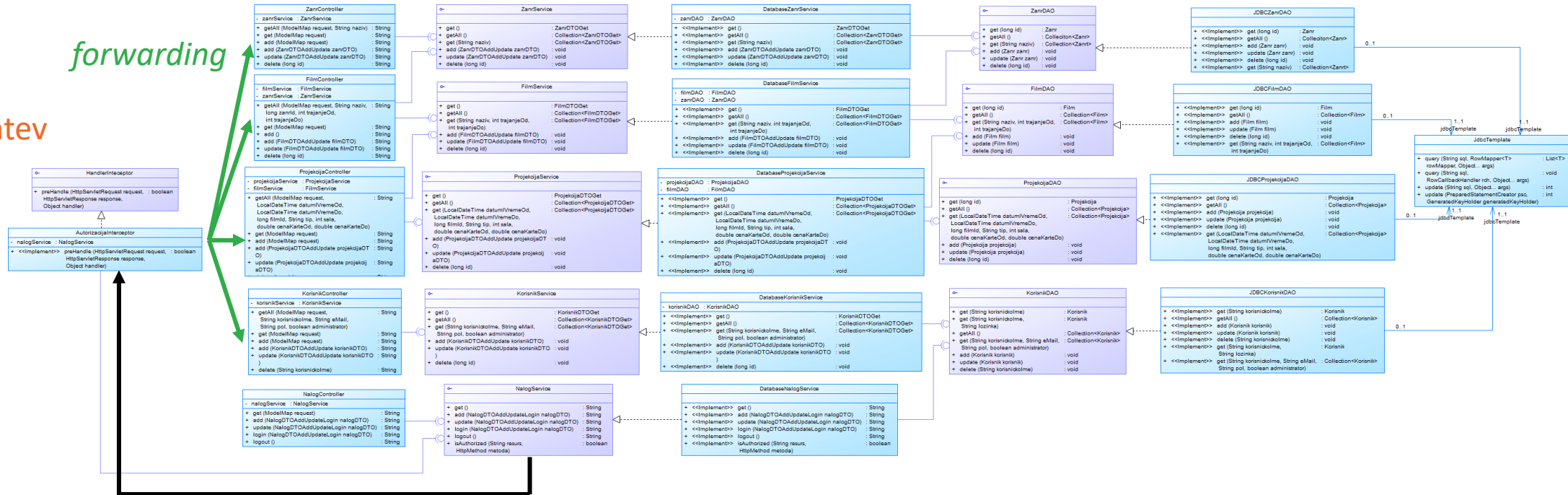
    @Override
    public boolean preHandle(
        HttpServletRequest request,
        HttpServletResponse response,
        Object handler) throws Exception {
        :
    }
}
```

Sistem na loga

HandlerInterceptor

forwarding

zahtev



autentikacija, autorizacija: *true*

Sistem na loga

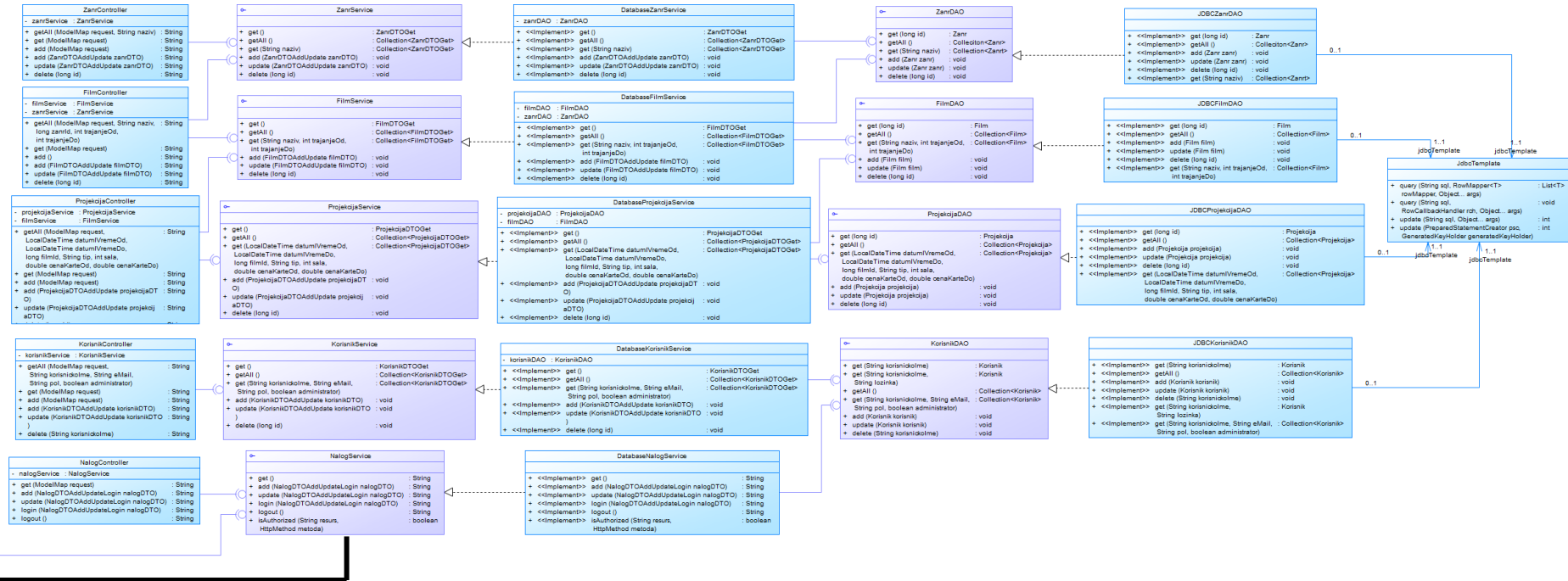
HandlerInterceptor

zahtev

redirekcija



autentikacija, autorizacija: *false*



Sistem naloga

HandlerInterceptor

```
@Component
public class AutorizacijaInterceptor implements HandlerInterceptor {
    private final NalogService nalogService;

    public AutorizacijaInterceptor(NalogService nalogService) {
        this.nalogService = nalogService;
    }

    @Override
    public boolean preHandle(
        HttpServletRequest request,
        HttpServletResponse response,
        Object handler) throws Exception {
        String resurs = request.getRequestURI();
        HttpMethod metoda = HttpMethod.valueOf(request.getMethod());
        boolean ovlasčen = nalogService.isAuthorized(resurs, metoda);

        if (!ovlasčen) {
            response.sendRedirect("/nalog-prijava.html");
            return false;
        }
        return true;
    }
}
```

inject

```
@Service
@Validated
public class DatabaseNalogService implements NalogService {
    private final KorisnikDAO korisnikDAO;
    private final ModelMapper mapper = new ModelMapper();

    private Prijava prijava; // session

    public DatabaseNalogService(KorisnikDAO korisnikDAO, Prijava prijava) {
        this.korisnikDAO = korisnikDAO;
        this.prijava = prijava;
    }

    @Override
    public boolean isAuthorized(String resurs, HttpMethod metoda) {
        // uvek dozvoljeno
        if (resurs.contains("/zanrovi") && metoda.equals(HttpMethod.GET))
            return true;
        if (resurs.contains("/filmovi") && metoda.equals(HttpMethod.GET))
            return true;
        if (resurs.contains("/projekcije") && metoda.equals(HttpMethod.GET))
            return true;

        // zahteva ovlašćenje
        String korisnickoIme = prijava.getKorisnickoIme(); // session
        if (korisnickoIme == null) {
            return false;
        }
        Korisnik korisnik = korisnikDAO.get(korisnickoIme);
        if (korisnik == null) {
            return false;
        }
        return korisnik.isAdministrator();
    }
}
```

Primer

- *com.ftninformatika.jwd.modul2.termin8.bioskop*

Thymeleaf

th:if, th:unless

Kako iz interfejsa sakriti komande koje korisnik ne treba da vidi?

```
@Controller
@RequestMapping("/filmovi")
public class FilmController {
    private final FilmService filmService;
    private final ZanrService zanrService;
    private final NalogService nalogService;

    public FilmController(
        FilmService filmService,
        ZanrService zanrService,
        NalogService nalogService) {
        this.filmService = filmService;
        this.zanrService = zanrService;
        this.nalogService = nalogService;
    }

    @GetMapping("")
    public String getAll(ModelMap request,
        @RequestParam(required = false, defaultValue = "") String naziv,
        @RequestParam(required = false, defaultValue = "0") long zanrId,
        @RequestParam(required = false, defaultValue = "0") int trajanjeOd,
        @RequestParam(required = false, defaultValue = "0") int trajanjeDo) {
        request.addAttribute("filmovi", filmService.get(
            naziv,
            zanrId,
            trajanjeOd, trajanjeDo));
        request.addAttribute("zanrovi", zanrService.getAll());
        request.addAttribute("poseceniFilmovi", filmService.getPoseceni());
        request.addAttribute("isAdministrator", nalogService.isAdministrator());
        return "filmovi";
    }
}
```

inject

```
@Service
@Validated
public class DatabaseNalogService implements NalogService {
    private final KorisnikDAO korisnikDAO;
    private final ModelMapper mapper = new ModelMapper();

    private Prijava prijava; // session

    public DatabaseNalogService(
        KorisnikDAO korisnikDAO,
        Prijava prijava) {
        this.korisnikDAO = korisnikDAO;
        this.prijava = prijava;
    }

    @Override
    public boolean isAdministrator() {
        String korisnickoIme = prijava.getKorisnickoIme(); // session
        if (korisnickoIme == null) {
            return false;
        }
        Korisnik korisnik = korisnikDAO.get(korisnickoIme);
        if (korisnik == null) {
            return false;
        }
        return korisnik.isAdministrator();
    }
}
```

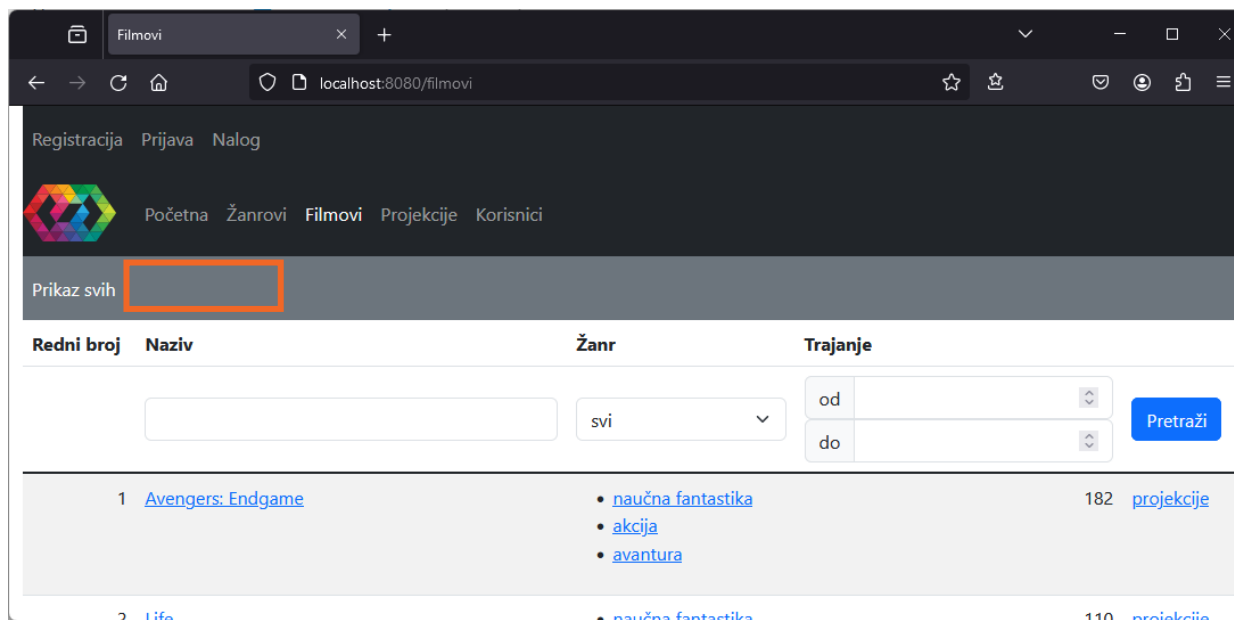

Thymeleaf

th:if, th:unless

th:if direktiva prikazuje element (sa svim podelementima) ukoliko je vrednost izraza *true*

th:unless direktiva sakriva element (sa svim podelementima) ukoliko je vrednost izraza *true*

```
<nav class="navbar navbar-expand navbar-dark bg-secondary">
  <div class="navbar-nav">
    <a class="nav-item nav-link active" href="/filmovi">Prikaz svih</a>
    <a th:if="${isAdministrator}" class="nav-item nav-link" href="/filmovi/dodavanje">Dodavanje</a>
  </div>
</nav>
```



Zadatak 3

Po ugledu na primer *com.ftninformatika.jwd.modul2.termin8.bioskop* implementirati sistem naloga u zadatku *com.ftninformatika.jwd.modul2.termin8.dostava*:

1. Definisati DTO klase za naloge u paketu *dto*:
 - i. apstraktnu DTO klasu koja sadrži sve deljene attribute
 - ii. konkretnu DTO klasu za povratne vrednosti *get* metoda servisa
 - iii. Konkretnu DTO klasu za parametre *add* i *update* i *login* metoda servisa
2. Definisati *NalogService* interfejs u paketu *service*
3. Započeti implementaciju servisa *DatabaseNalogService* u paketu *service.impl*
4. Započeti implementaciju *controller*-a *NalogController*
5. *Inject*-ovati *KorisnikDAO* u *DatabaseNalogService*
6. *Inject*-ovati *NalogService* u *NalogController*
7. Kopirati stranice *nalog-registracija.html*, *nalog-prijava.html* u direktorijum *resources/public*
8. Kopirati template *nalog-prikaz.html* u direktorijum *resources/templates*
9. Kopirati meni za upravljanje nalogom na sve stranice
10. Implementirati registraciju, prikaz, izmenu naloga, prijavu i odjavu
11. Definisati i registrovati *interceptor* za autorizaciju u paketu *web.interceptors*
12. *Inject*-ovati *NalogService* u *interceptor* za autorizaciju
13. Implementirati *isAuthorized(...)* metodu *DatabaseNalogService* servisa i *preHandle(...)* metodu *interceptor*-a za autorizaciju i testirati neovlašćen pristup zaštićenim resursima

Zadatak 4

U zadatku *com.ftninformatika.jwd.modul2.termin8.dostava* dopuniti metodu *isAuthorized(...)* *DatabaseNalogService* servisa da zabrani neovlašćeni pristup i stranicama za dodavanje entiteta.