



IT355 - WEB SISTEMI 2

Spring Security podokvir

Lekcija 10

PRIRUČNIK ZA STUDENTE

IT355 - WEB SISTEMI 2

Lekcija 10

SPRING SECURITY PODOKVIR

- ✓ Spring Security podokvir
- ✓ Poglavlje 1: Bezbednost URL pristupa
- ✓ Poglavlje 2: Prijavljivanje u Spring veb aplikaciji
- ✓ Poglavlje 3: Provera korisnika
- ✓ Poglavlje 4: Upravljanje kontrolom pristupa
- ✓ Poglavlje 5: Zaštita poziva metoda
- ✓ Poglavlje 6: Upravljanje bezbednošću u pogledima
- ✓ Poglavlje 7: OAuth2 autorizacija i server resursa
- ✓ Poglavlje 8: Pokazna vežba 10
- ✓ Poglavlje 9: Individualna vežba 10
- ✓ Poglavlje 10: Domaći zadatak 10
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Lekcija se bavi predstavljanjem Spring Security okvira.

U ovoj lekciji će biti pokazano kako je moguće učiniti aplikacije bezbednijim primenom *Spring Security* okvira koji je potprojekat Spring okvira. *Spring Security* je u početku nazvan **Acegi Security**, međutim, naziv mu je naknadno promenjen nakon kombinovanja sa *Spring Portfolio* projektima. Koristeći ovaj okvir moguće je osigurati bilo koju JAVA aplikaciju ali je njegov fokus uglavnom na web baziranim aplikacijama. Ove aplikacije, pogotovo one kojima je moguće pristupiti putem *Interneta*, veoma su ranjive na zlonamerne napade i zahtevaju veoma veliko angažovanje bezbednosnih tehnika prilikom njihovog kreiranja i održavanja.

Tokom izlaganja, o ovoj lekciji, biće govora o nekim veoma značajnim terminima iz domena bezbednosti aplikacija, podržanih Springom, a koje studenti moraju da savladaju:

- *Authentication;*
- *Principal;*
- *Authorization;*
- **Access control** .

Na kraju ove lekcije, student će razumeti osnovne bezbednosne koncepte i moći da ih primeni na web aplikacije na:

- URL nivou pristupa;
- nivou poziva metoda;
- nivou prikazivanja pogleda;
- nivou objekata domena.

▼ Poglavlje 1

Bezbednost URL pristupa

OSNOVNA BEZBEDNOSNA PODEŠAVANJA

Jednostavnim Java konfiguracijama, Spring osigurava URL pristup web aplikacijama.

Bojne veb aplikacije koriste izvesne URL linkove koji se smatraju kritičnim i važnim. Oni moraju da budu osigurani od neovlašćenog pristupa njima.

Okvir Spring Security omogućava obezbeđivanje URL-a u veb aplikacijama, veoma jednostavno, na deklarativan način primenom jednostavnih konfiguracija. Okvir vrši rukovanje bezbednošću primenom servlet filtera na HTTP zahteve. Za registrovanje filtera i pronalaženje konfiguracije, Spring Security obezbeđuje veoma jednostavnu podršku u formi nasleđivanja osnovne klase AbstractSecurityWebApplicationInitializer.

Dalje, okvir Spring Security dozvoljava podešavanje bezbednosti veb aplikacije kroz različite konfiguracione metode klase WebSecurityConfigurerAdapter. Ukoliko su zahtevi bezbednosti veb aplikacije jednostavni i standardni, moguće je koristiti podrazumevana bezbednosna podešavanja koja uključuju sledeće:

- servis povezivanja baziran na formi: na ovaj način je obezbeđena podrazumevana stranica koja sadrži formu za povezivanje (login form) korisnika na aplikaciju;
- osnovna HTTP provera: na ovaj način moguće je obraditi osnovnu proveru korisnika kroz proveru njihovih kredencijala iz zaglavlja HTTP zahteva. Ovde je, takođe, moguće izvršiti proveru zahteva kreiranih pomoću udaljenih protokola i veb servisa.
- servis za odjavljivanje (logout service): ovde je obezbeđen rukovalac povezan sa URL za odjavljivanje korisnika sa aplikacije;
- anonimno povezivanje: pridružuje prava i autoritete za dozvole za upravljanje anonimnim korisnicima kao regularnim;
- Servlet API integraciju: omogućava pristup bezbednosnim informacijama u veb aplikaciji preko standardnih Servlet API-ja, poput HttpServletRequest.isUserInRole() i HttpServletRequest.getUserPrincipal().
- CSFR (Cross - Site Request Forgery): implementacija zaštite od veb napada koji rezultuje preduzimanja neželjenih akcija krajnjeg korisnika. Zaštita podrazumeva kreiranje tokena i njegovo smeštanje u HttpSession objekat;
- bezbednosna zaglavlja (security headers): poput isključivanja keširanja za obezbeđene pakete, ovde je dostupna: XSS (Cross - Site Scripting) zaštita, bezbednost transporta podataka i X-Frame bezbednost.

Registrowanjem navedenih bezbednosnih servisa, moguće je specificirati URL šablone koji omogućavaju pristup aplikaciji konkretnim autoritetima (korisnicima). *Spring Security* izvodi obezbeđivanje aplikacije na osnovu kreiranih konfiguracija. Korisnik prvo mora da se uspešno registruje da bi dalje koristio aplikaciju, ne računajući funkcionalnosti koje su dostupne neautorizovanim korisnicima. *Spring Security* obezbeđuje nekoliko provajdera za proveru korisnika.

OSNOVNA PODEŠAVANJA - REGISTROVANJE FILTERA

Obezbeđivanje veb aplikacije predstavlja registrovanje filtera primenom Spring Security okvira

Prvi korak u obezbeđivanju veb aplikacije predstavlja registrovanje filtera primenom Spring Security okvira. Najlakši način za obavljanje navedenog zadatka predstavlja nasleđivanje klase *AbstractSecurityWebApplicationInitializer*.

```
package com.metropolitan.security;

import
org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

/**
 *
 * @author Vlada
 */
public class TodoSecurityInitializer extends
AbstractSecurityWebApplicationInitializer {

    public TodoSecurityInitializer() {
        super(TodoSecurityConfig.class);
    }
}
```

Klasa *AbstractSecurityWebApplicationInitializer* poseduje konstruktor koji preuzima jednu ili više konfiguracionih klasa. Zadatak ovih konfiguracionih klasa jeste pokretanje bezbednosnih mehanizama unutar Spring veb aplikacije.

Upravo će biti i urađeno kako je navedeno. Sledećim listingom je data jedna bezbednosna konfiguraciona Spring klasa.

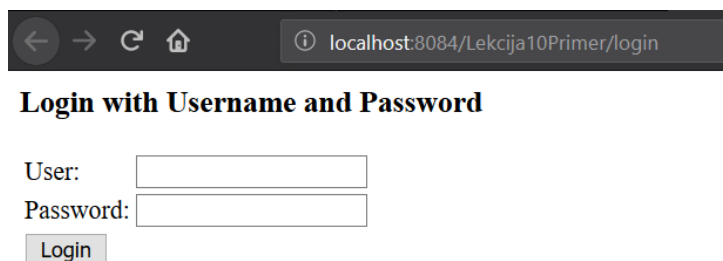
```
package com.metropolitan.security;

import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

/**
```

```
*
* @author Vlada
*/
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {}
```

Sada, kada bi bilo izvršeno prevođenje i angažovanje veb aplikacije i pokušao pristup linku: <http://localhost:8080/Lekcija10Primer/> učitala bi se podrazumevana *Spring Security* stranica za povezivanje.



Slika 1.1.1 Podrazumevana Spring Security stranica za povezivanje [izvor: autor]

OBEZBEĐIVANJE URL PRISTUPA

Osnovna HTTP provera i prijavljivanje bazirano na formi su omogućeni.

Sada će akcenat biti na jednoj konkretnoj metodi klase *WebSecurityConfigurerAdapter*. Metoda pod nazivom *configure()* će da uključi poziv *anyRequest().authenticated()*. Na ovaj način će biti predloženo *Spring Security* okviru, da svaki dolazeći zahtev mora da bude proveren od strane sistema. Takođe, osnovna *HTTP* provera i prijavljivanje bazirano na formi su omogućeni. Prijavljivanje bazirano na formi, takođe, uključuje i primenu podrazumevanog kreatora stranica koji će biti upotrebljen ukoliko programer eksplicitno ne specificira stranicu za prijavljivanje. Sledećim listingom je priložen opšti oblik navedene metode.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        .anyRequest().authenticated()
        .and()
        .formLogin().and()
        .httpBasic();
}
```

Sada je moguće napisati nekoliko bezbednosnih pravila. Pored jednostavne potrebe za povezivanjem, moguće je napisati nekoliko moćnih pravila pristupa za URL.

```
package com.metropolitan.security;
```

```
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;

/**
 *
 * @author Vlada
 */
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication().passwordEncoder(NoOpPasswordEncoder.getInstance())
        .withUser("vlada").password("user").authorities("USER")
        .and()
        .withUser("admin").password("admin").authorities("USER", "ADMIN");

    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/todos*").hasAuthority("USER")
            .antMatchers(HttpMethod.DELETE, "/todos*").hasAuthority("ADMIN")
            .and()
            .formLogin()
            .and()
            .csrf().disable();
    }
}
```

POJAŠNJENJA

Moguće podešavanje jednog ili više pravila autorizacije preko redefinisanja metode `configure()`.

Iz priloženog listinga je moguće primetiti da je moguće podešavanje jednog ili više pravila autorizacije preko redefinisanja metode `configure(HttpSecurity http)`.

Primenom metode `authorizeRequests()` započinje obezbeđivanje URL-a. Tada je moguće upotrebiti neki od mehanizama za pronalaženje podudaranja (*eng. matchers*), u konkretnom slučaju `antMatchers()`, za definisanje pravila podudaranja i tipa korisnika koji odgovaraju navedenom podudaranju. Ovde je bitno zapamtiti da je **neophodno uključiti džoker simbol "*" na kraj URL šablona**. Izostavljanjem ovog znaka biće nemoguće izvršiti proveru podudaranja URL koji sadrži parametre zahteva. **Na ovaj način hakeri će imati mogućnost veoma jednostavnog zaobilaženja sigurnosne provere dodavanjem proizvoljnog parametra zahteva**. U konkretnom slučaju, osigurani su svi pristupi URL-u `/todos` za korisnike koji imaju autoritet USER. Za poziv URL `/todos` sa zahtevom `DELETE`, neophodno je korisnik ima ulogu `ADMIN`.

Moguće je podesiti sve servise autentifikacije u redefinisanoj metodi `configure(AuthenticationManagerBuilder auth)`. *Spring Security* obezbeđuje nekoliko načina provere korisnika, uključujući proveru iz baze podataka ili *LDAP (Lightweight Directory Access Protocol)* repozitorijuma. Takođe, moguće je definisati detalje korisnika direktno, za jednostavne bezbednosne zahteve, poput korisničkog imena, lozinke i prava pristupa. Na kraju za rukovanje pravima pristupa dodat je koder lozinke kroz poziv: `passwordEncoder(NoOpPasswordEncoder.getInstance())`.

OBEZBEĐIVANJE URL PRISTUPA - DEMO

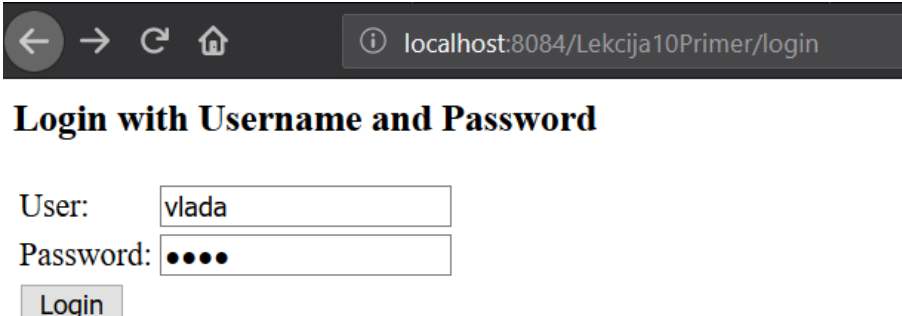
Pokretanjem aplikacije prate se implementirana bezbednosna podešavanja.

Sada je neophodno ponovo izvršiti prevođenje aplikacije i njeno ponovno angažovanje na aplikativnom serveru.

Pozivom linka:

```
localhost:8084/Lekcija10Primer/todos
```

automatski se vrši preusmeravanje na stranicu za prijavljivanje korisnika, kao na sledećoj slici.



← → ↻ 🏠 ⓘ localhost:8084/Lekcija10Primer/login

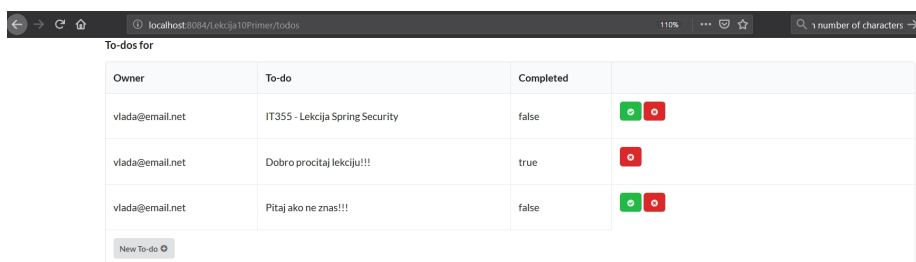
Login with Username and Password

User:

Password:

Slika 1.1.2 Prijavljivanje korisnika sa pravom pristupa [izvor: autor]

Ukoliko je prijavljivanje bilo uspešno, otvara se stranica koja je i bila cilj prilikom poziva prvog linka. Navedeno je prikazano sledećom stranicom.

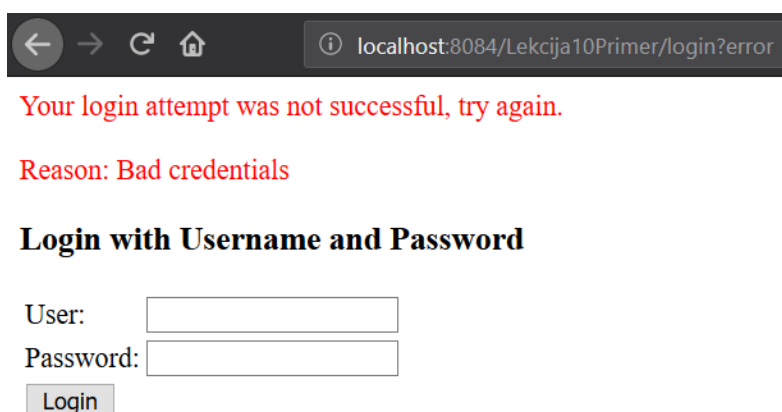


Owner	To-do	Completed	
vlada@email.net	IT355 - Lekcija Spring Security	false	● ●
vlada@email.net	Dobro procitaj lekciju!!!	true	●
vlada@email.net	Pitaj ako ne znas!!!	false	● ●

New To-do

Slika 1.1.3 Uspešno prijavljivanje na aplikaciju [izvor: autor]

U slučaju pokušaja pristupa aplikaciji od strane neautorizovanog korisnika ili greške prilikom unosa podataka za autorizovane korisnike, stranica za prijavljivanje će dati informaciju kao na sledećoj slici.



← → ↻ 🏠 ⓘ localhost:8084/Lekcija10Primer/login?error

Your login attempt was not successful, try again.

Reason: Bad credentials

Login with Username and Password

User:

Password:

Login

Slika 1.1.4 Neuspešno prijavljivanje na aplikaciju [izvor: autor]

VIDEO MATERIJAL

Spring Security 5 - Hello World example

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ 1.1 Spring Security CSFR zaštita

PRIMENA CSFR ZAŠTITE

Primena CSFR zaštite smanjuje rizik od CSFR napada.

Uopšteno gledano, jako dobra ideja je ostavljanje podrazumevanih podešavanja za CSFR (*Cross-site request forgery*) uključenim jer se na taj način smanjuje rizik od CSFR napada. Opcija je po automatizmu uključena primenom Spring Security okvira. Pozivom

`csrf().disable()`, kao što je to učinjeno u prethodnom primeru, ovu opciju je moguće ukloniti iz bezbednosnih podešavanja.

Kada je **CSFR** zaštita uključena, Spring Security okvir dodaje `CsrfFilter` filter na listu filtera koje koristi u zaštiti aplikacije. Tada navedeni filter koristi implementaciju interfejsa `CsrfTokenRepository` za generisanje i skladištenje tokena. Po osnovnim podešavanjima, ovu dužnost obavlja klasa `HttpSessionCsrfTokenRepository` koja koristi metode `HttpSession` interfejsa za skladištenje generisanih tokena. Takođe, ovaj zadatak može biti poveren klasi `CookieCsrfTokenRepository` koja čuva informacije tokena u kolačićima.

Klasa `HttpSessionCsrfTokenRepository` na sledeći način može biti eksplicitno podešena.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    ***
    HttpSessionCsrfTokenRepository repo = new HttpSessionCsrfTokenRepository();
    repo.setSessionAttributeName("csrf_token");
    repo.setParameterName("csrf_token");
    http.csrf().csrfTokenRepository(repo);
}
```

U konkretnom primeru, moguće je držati se osnovnih podešavanja i obrisati liniju `csrf().disable()` u drugoj metodi `configure()` klase `TodoSecurityConfig`.

Ova metoda će sada imati sledeći oblik:




```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/todos*").hasAuthority("USER")
        .antMatchers(HttpMethod.DELETE, "/todos*").hasAuthority("ADMIN")
        .and()
        .formLogin();
}
```


DEMONSTARCIJA

Token je neophodno proslediti nazad serveru.

Kada je uključena CSFR zaštita, pokušaj da se obavi zadatak `complete` ili `delete`, na formi sa sledeće slike, biće neuspešan iz razloga odsustva CSFR tokena (slika 2).

To-dos for

Owner	To-do	Completed	
vlada@email.net	IT355 - Lekcija Spring Security	false	
vlada@email.net	Dobro procitaj lekciju!!!	true	
vlada@email.net	Pitaj ako ne znas!!!	false	

New To-do 

Slika 1.2.1 Izbor zadatka sa forme stranice todos.jsp [izvor: autor]






Slika 1.2.2 Neuspešno obavljanje zadataka sa forme [izvor: autor]


Da bi ovaj problem bio rešen, neophodno je obezbediti mehanizam pomoću kojeg se **CSFR** token vraća nazad serveru u zahtevima sa promenljivim sadržajem. Ovo je moguće obaviti na veoma jednostavan način dodavanjem skrivenog polja tipa **input** na odgovarajuće mesto u formi. **HttpSessionCsrfTokenRepository** otkriva token u sesiji pod atributom **_csrf** (po osnovnim podešavanjima, ukoliko nije drugačije eksplicitno podešeno). U formama za brisanje i kompletiranje zadataka, stranice todos.jsp, je moguće upotrebiti atribut **parameterName** za realizovanje navedenog **input** taga:

```
<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
```

Problem je rešen i aplikacija radi normalno. Klikom na zadatak **complete** (zeleno dugme - slika 1), token se vraća nazad serveru i dobija se izlaz kao na sledećoj slici.

To-dos for

Owner	To-do	Completed	
vlada@email.net	IT355 - Lekcija Spring Security	true	
vlada@email.net	Dobro procitaj lekciju!!!	true	
vlada@email.net	Pitaj ako ne znas!!!	false	

New To-do 

Slika 1.2.3 Uspešno obavljanje zadataka sa forme [izvor: autor]

Preuzmite urađen zadatak iz aktivnosti **Shared Resources** odmah iza ovog objekta učenja.

VIDEO MATERIJAL

Cross-Site Request Forgery video materijali

Java - Identify Cross-Site Request Forgery (CSRF) - video materijal

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Understanding CSRF, the video tutorial edition - video materijal

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Spring Security 50 CSRF Protection - video materijal

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 2

Prijavljivanje u Spring veb aplikaciji

DEFINISANJE PROBLEMA POVEZIVANJA

Spring Security podržava više načina za prijavljivanje korisnika na veb aplikaciju.

Prva stvar koja mora da se shvati, a tiče se domena bezbednosti veb aplikacija, jeste da bezbedne aplikacije zahtevaju prijavljivanje korisnika pre korišćenja njihovih funkcionalnosti. Ovo je posebno značajno za veb aplikacije koje se izvršavaju na *Internetu* jer su lako dostupne hakerima. Većina veb aplikacija mora da obezbedi način za unos korisničkih podataka za prijavljivanje na aplikaciju.

Spring Security podržava više načina za prijavljivanje korisnika na veb aplikaciju:

- korišćenje login forme;
- HTTP provera identiteta.

Neki delovi aplikacije mogu da dozvole pristup i anonimnim korisnicima, na primer stranica za dobrodošlicu. *Spring Security* podržava i *anonimni login servis* za dodeljivanje prava i obaveza korišćenja aplikacije anonimnim korisnicima.

Takođe, *Spring Security* podržava i *remember-me* (zapamti me) servis prijavljivanja za pamćenje identiteta korisnika tokom višestrukih sesija tako da nije potrebno ponovno prijavljivanje na aplikaciju nakon što je to prvi put urađeno, pod uslovom da se korisnik nije odjavio sa sistema pre toga.

Za kvalitetnije razumevanje različitih *login* mehanizama, neophodno je prvo isključiti automatsku HTTP konfiguraciju uklanjanjem atributa *auto-config*.

ISKLUČIVANJE OSNOVNIH BEZBEDNOSNIH PODEŠAVANJA

Ukoliko se isključe podrazumevana podešavanja neophodno je podesiti bezbednosne alate eksplicitno.

Za kvalitetnije razumevanje različitih mehanizama prijavljivanja izvodi se prvi korak - isključivanje podrazumevanih bezbednosnih konfiguracija.

Moguće je zadržati osnovna podešavanja, a samo neka od njih isključiti, kao na primer: **httpBasic().disable()**.

```
/**
 *
 * @author Vlada
 */
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    public TodoSecurityConfig() {
        super(true);
    }

}
```

Trebalo bi napomenuti da će servisi prijavljivanja na aplikaciju, o kojima će biti govora u nastavku, automatski biti registrovani ukoliko se omogući opcija *HTTP autoconfig*. Međutim, ukoliko se isključe podrazumevana podešavanja, ili je cilj redefinisane navedenih servisa, neophodno je podesiti odgovarajuće alate eksplicitno.

Pre omogućavanja mehanizama za proveru korisnika, neophodno je omogućiti osnovne *Spring Security* zahteve. Neophodni minimum zahteva predstavljaju rukovanje izuzecima i integracija sigurnosnog konteksta.

```
@Override
protected void configure(HttpSecurity http) {
    http.securityContext()
        .and()
        .exceptionHandling();
}
```

Bez pomenutih osnovnih podešavanja, Spring Security neće sačuvati korisnika nakon uspešnog prijavljivanja na aplikaciju. Takođe, neće obaviti ni odgovarajuće prevođenje izuzetaka za javljanje izuzetaka koji su u vezi sa bezbednosnim stavkama. U nastavku je moguće obezbediti i *Servlet API* integraciju da bi bilo moguće primenjivati metode na *HttpServletRequest* za obavljanje provera u JSP pogledima.

```
protected void configure(HttpSecurity http) {
    http.servletApi();
}
```

PRIMENA OSNOVNE HTTP PROVERE

Za osnovna HTTP podešavanja, prikazuje se dialog ili stranicu za prijavljivanje korisnika.

Podrška za osnovnu HTTP proveru (HTTP Basic authentication) može biti podešena preko metode `httpBasic()`. Kada su zahtevana osnovna HTTP podešavanja, veb pregledač će prikazati dialog ili specifičnu stranicu za prijavljivanje korisnika.

```
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            ...
            .httpBasic();
    }
}
```

Kada je **osnovna HTTP provera** uključena istovremeno kada i **provera korisnika bazirana na primeni forme**, poslednje navedena će biti upotrebljena. Iz navedenog razloga, ukoliko je želja da se korisnici prijavljuju na aplikaciju primenom **osnovne HTTP provere**, neophodno je onemogućiti **proveru korisnika bazirana na primeni forme**.

PRIJAVLJIVANJE PUTEM FORME

Problem se svodi na primenu veb stranice koja sadrži login formu za prijavljivanje.

Servis logovanja putem forme će kreirati web stranicu koja sadrži *login* formu na kojoj korisnici mogu da unesu vlastite podatke za prijavljivanje na aplikaciju. Ovaj mehanizam se podešava preko metode `formLogin()`.

```
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            ...
            .formLogin();
    }
}
```

Po osnovnim podešavanjima, okvir *Spring Security* će automatski kreirati stranicu za prijavljivanje i mapirati je pomoću URL-a `/login`. Dakle, moguće je dodati sledeći link, u na

primer poznatu stranicu [todos.jsp](#), koji bi omogućio navigaciju ka stranici za prijavljivanje korisnika:

```
<a href="<c:url value="/login" />">Login</a>
```

PRIJAVLJIVANJE PUTEM FORME - KREIRANJE VLASTITE STRANICE

Ukoliko se želi izbeći podrazumevana stranica za prijavljivanje, neophodno je obezbediti vlastitu.

Ukoliko se želi izbeći podrazumevana stranica za prijavljivanje, neophodno je obezbediti vlastitu. Na primer, moguće je kreirati sledeću [login.jsp](#) datoteku

```
<% -
    Document    : login
    Created on  : 02.11.2018., 10.55.27
    Author      : Vlada
- %>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
    <head>
        <title>Login</title>
        <link type="text/css" rel="stylesheet"
            href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.10/
semantic.min.css">
        <style type="text/css">
            body {
                background-color: #DADADA;
            }
            body > .grid {
                height: 100%;
            }
            .column {
                max-width: 450px;
            }
        </style>
    </head>

    <body>
        <div class="ui middle aligned center aligned grid">
            <div class="column">
                <h2 class="ui header">Log-in to your account</h2>
                <form method="POST" action="<c:url value="/login" />" class="ui
large form">
                    <input type="hidden" name="${_csrf.parameterName}"
```



```
value="${_csrf.token}"/>
    <div class="ui stacked segment">
        <div class="field">
            <div class="ui left icon input">
                <i class="user icon"></i>
                <input type="text" name="username"
placeholder="E-mail address">
            </div>
        </div>
        <div class="field">
            <div class="ui left icon input">
                <i class="lock icon"></i>
                <input type="password" name="password"
placeholder="Password">
            </div>
        </div>
        <div class="field">
            <div class="ui toggle checkbox">
                <input type="checkbox" name="remember-me">
                <label>Remember Me</label>
            </div>
        </div>
        <button class="ui fluid large submit green
button">Login</button>
    </div>
    <c:if test="${not empty param.error}">
        <div class="ui error message" style="display: block;">
            Authentication Failed<br/>
            Reason :
            ${sessionScope["SPRING_SECURITY_LAST_EXCEPTION"].message}
        </font>
        </div>
    </c:if>
</form>
</div>
</div>
</body>
</html>
```

Trebalo bi imati na umu da ovu datoteku ne bi trebalo čuvati unutar podrazumevanog WEB-INF foldera. Na taj način će biti onemogućen direktan pristup korisnika odgovarajućoj stranici.

Da bi okvir *Spring Security* mogao da prikaže ovako kreiranu stranicu, kada se pojavi zahtev za prijavljivanjem korisnika, neophodno je specificirati njen URL u konfiguracionoj metodi *loginPage()*, kao na sledeći način:

```
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
```

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        ...  
        .formLogin().loginPage("/login.jsp");  
}  
}
```

Ukoliko je cilj definisanja početne stranice koja se prikazuje nakon uspešnog prijavljivanja na sistem, moguće je u prethodnom listingu napraviti sledeću modifikaciju;

```
formLogin().loginPage("/login.jsp").defaultSuccessUrl("/todos");
```

PRIJAVLJIVANJE PUTEM FORME - PRIKAZIVANJE GREŠKE

Neophodno je podesiti vrednost URL-a za neuspešnu proveru korisnika i preusmeravanje.

Ukoliko se koristi podrazumevana *Spring Security* stranica za prijavljivanje korisnika na aplikaciju, kada je prijavljivanje neuspešno *Spring Security* okvir će kreirati stranicu za prijavljivanje koja će prikazivati i poruku o grešci.

U slučaju primene vlastite stranice za prijavljivanje korisnika, biće neophodno podesiti vrednost URL-a za neuspešnu proveru korisnika pomoću koje se vrši preusmeravanje na stranicu koja prikazuje grešku. Na primer, moguće je izvršiti preusmeravanje baš na stranicu za prijavljivanje u slučaju neuspešne provere korisnika, baš kao na sledeći način:

```
@Configuration  
@EnableWebSecurity  
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Override  
    protected void configure(HttpSecurity http) throws Exception {  
        http  
            ...  
            .formLogin().loginPage("/login.jsp")  
            .defaultSuccessUrl("/todos")  
            .failureUrl("login.jsp?error=true");  
    }  
}
```

U ovom slučaju, kreirana stranica za prijavljivanje proverava da li je prisutan parametar zahteva koji odgovara grešci. Ukoliko se desila greška biće neophodno prikazati poruku sa greškom pristupajući atributu sesije *SPRING_SECURITY_LAST_EXCEPTION* koji čuva poslednje javljeni izuzetak za aktuelnog korisnika. Za navedeno je zaslužan sledeći deo *login.jsp* stranice:

```
<form>
....
    <c:if test="${not empty param.error}">
        <div class="ui error message" style="display: block;">
            Authentication Failed<br/>
            Reason : ${sessionScope["SPRING_SECURITY_LAST_EXCEPTION"].message}
        </div>
    </c:if>
</form>
```

PRIMENA SERVISA ZA ODJAVLJIVANJE

Logout servis obezbeđuje upravljanje zahtevima za odjavljivanje sa aplikacije.

Logout servis obezbeđuje upravljanje zahtevima za odjavljivanje sa aplikacije. Moguće ga je podesiti primenom *logout()* konfiguracione metode na sledeći način:

```
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            ...
            .and()
            .logout();
    }
}
```

Po osnovnim podešavanjima, odjavljivanje je mapirano linkom `/logout` i reaguje samo na zahteve tipa *POST*. Za kompletno funkcionisanje navedenog servisa moguće je dodati malu formu na stranicu sa koje je moguće odjavljivanje sa sistema. Neka to ponovo bude *todos.jsp* u koju je dodata sledeća linija koda:

```
<form action="<c:url value="/logout"/>" method="post">
<button>Logout</button>
</form>
```

Ukoliko je cilj definisanje pristupa konkretnom URL-u, nakon odjavljivanja sa aplikacije, moguće primeniti konfiguracionu metodu *logoutSuccessUrl()* na sledeći način:

```
@Configuration
@EnableWebSecurity
```

```
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            ...
            .and()
            .logout().logoutSuccessUrl("/logout-success.jsp")
    }
}
```

PRIMENA SERVISA ZA ODJAVLJIVANJE - STRANICA ZA ODJAVLJIVANJE

Neophodno je kreirati stranicu koja će se prikazati nakon uspešnog odjavljivanja.

Neophodno je kreirati stranicu koja će se prikazati nakon uspešnog odjavljivanja korisnika sa aplikacije. Stranica je kreirana iz datoteke *logout-success.jsp*. Ova datoteka se takođe, baš kao i *login.jsp*, čuva izvan foldera WEB-INF.

Na posebnom delu ove stranice (sledeća slika i linija koda 33) je ugrađen kod koji ističe link koji omogućava ponovo navigaciju ka stranici *todos.jsp*. Naravno, ponovo se pokreću bezbednosni mehanizmi i neophodno je ponovno prijavljivanje korisnika na aplikaciju.

```
</head>

<body>
  <div class="ui middle aligned center aligned grid">
    <div class="column">
      <h2 class="ui header">You have been successfully logged out</h2>
      <div>Goto <a href="<@:url value="/todos"/>" class="link">todo list</a>.</div>
    </div>
  </div>
</body>
</html>
```

Slika 2.1 Link za preusmeravanje na početnu stranicu [izvor: autor]

```
<%--
  Document   : login-success
  Created on : 02.11.2018., 10.56.44
  Author      : Vlada
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
  <head>
```

```

<title>Login</title>
<link type="text/css" rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.10/
semantic.min.css">
<style type="text/css">
  body {
    background-color: #DADADA;
  }
  body > .grid {
    height: 100%;
  }
  .column {
    max-width: 450px;
  }
</style>
</head>

<body>
  <div class="ui middle aligned center aligned grid">
    <div class="column">
      <h2 class="ui header">You have been successfully logged out</h2>
      <div>Goto <a href="<c:url value="/todos"/>" class="link">todo
list</a>.</div>
    </div>
  </div>
</body>
</html>

```

SPREČAVANJE KEŠIRANJA STRANICA

Primenom metode `headers()`, veb pregledač će dobiti instrukciju da ne kešira stranice.

Nakon uspešnog odjavljivanja, iz aktuelnog primera, moguće je primetiti da klikom na dugme "**BACK**" veb pregledača ponovo je dostupna prethodna stranica. Navedeno je u direktnoj vezi sa činjenicom da je pregledač izvršio keširanje stranice. Omogućavanjem bezbednosnog zaglavlja, primenom konfiguracione metode `headers()`, veb pregledač će dobiti instrukciju da ne kešira stranice.

```

@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            ...
            .and()
            .headers();
    }
}

```

```
}  
  
}
```

PODRŠKA ZA ANONIMNO PRIJAVLJIVANJE NA APLIKACIJU

Servis za anonimno prijavljivanje podešava se konfiguracionom metodom `anonymous()`

Servis za anonimno prijavljivanje podešava se konfiguracionom *Java* metodom `anonymous()`. Ovde je moguće podesiti korisničko ime i uloge za anonimne korisnike, čiji su podrazumevani nazivi **`anonymousUser`** i **`ROLE_ANONYMOUS`**.

Sledećim listingom je demonstrirana implementacija gore opisanog servisa za anonimno prijavljivanje.

```
@Configuration  
@EnableWebSecurity  
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Override  
    protected void configure(HttpSecurity http) throws Exception {  
        http  
            ...  
            .and()  
            .anonymous().principal("guest").authorities("ROLE_GUEST");  
    }  
}
```

PODRŠKA ZA PAMĆENJE KORISNIKA

Moguće je obezbediti pamćenje prijavljenog korisnika.

Ovu podršku moguće je podesiti konfiguracionom metodom `rememberMe()` u klasi Java konfiguracija. Po osnovnim podešavanjima, kodiraju se korisničko ime, lozinka i vreme isteka podrške pamćenja i privatni ključ kao token koji se čuva kao *kolačići* u korisnikovom web čitaču.

Primena statičkih **remeber-me** tokena može da izazove bezbednosne probleme jer postoji mogućnost da ih uhvate hakeri. **Spring Security** daje naprednu podršku za primenu poromenljivih tokena (**rolling tokens**), međuim ova podrška zahteva bazu podataka za čuvanje tokena. Za detalje u vezi sa naprednom primenom **remember-me** tokena

konsultujte **Spring Security** dokumentaciju: <https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/>.

```
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            ...
            .and()
            .rememberMe();
    }
}
```

KONFIGURACIONA KLASA

Metoda `configure()` objedinjuje pokazane bezbednosne alate na jednom mestu.

Kao rezime ovog dela lekcije, neophodno je sve prethodne bezbednosne mehanizme prijavljivanja korisnika putem forme objединiti na jednom mestu. Ovaj zadatak je poveren poznatoj metodi `configure(HttpSecurity http)`. Obezbeđena je podrška za:

- dodelu prava pristupa korisnicima;
- uključivanje prava prijave za anonimne korisnike;
- omogućavanje Servlet API i Security Context podrške;
- rukovanje izuzecima;
- pamćenje korisnika;
- mehanizme prijave preko kreirane forme;
- mehanizme za odjavljivanje sa aplikacije;
- sprečavanje keširanja stranica;

Kompletna kod konfiguracione klase, čiji je sastavni deo navedena metoda, priložen je sledećim listingom.

```
/**
 *
 * @author Vlada
 */
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {
```

```
public TodoSecurityConfig() {
    super(true);
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication().passwordEncoder(NoOpPasswordEncoder.getInstance())
        .withUser("vlada").password("user").authorities("USER")
        .and()
        .withUser("admin").password("admin").authorities("USER", "ADMIN");
}

@Override
protected void configure(HttpSecurity http) throws Exception {

    http.authorizeRequests()
        .antMatchers("/todos*").hasAuthority("USER")
        .antMatchers(HttpMethod.DELETE, "/todos*").hasAuthority("ADMIN")
        .and()
        .anonymous()
        .and()
        .servletApi()
        .and()
        .securityContext()
        .and()
        .exceptionHandling()
        .and()
        .rememberMe()
        .and()
        .formLogin()
            .loginPage("/login.jsp")
            .loginProcessingUrl("/login")
            .failureUrl("/login.jsp?error=true")
            .defaultSuccessUrl("/todos")
        .permitAll()
        .and()
        .logout().logoutSuccessUrl("/logout-success.jsp")
        .and()
        .headers()
        .and()
        .csrf();
}
}
```

DEMONSTRACIJA

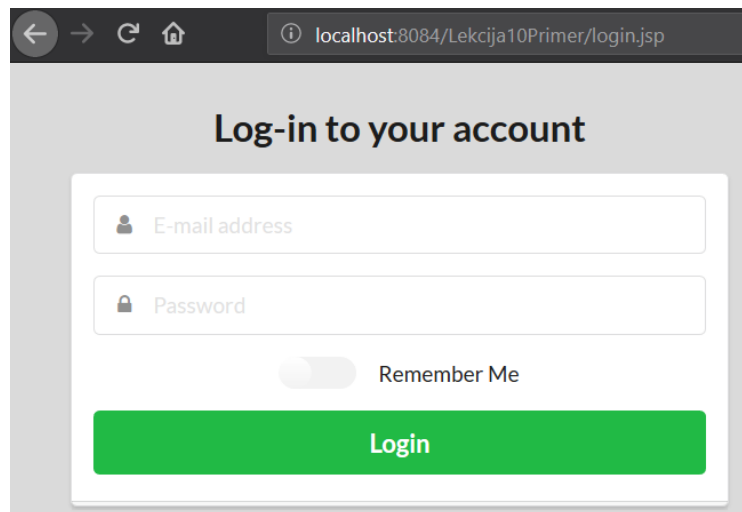
Demonstracija urađenog primera

Nakon urađenih modifikacija neophodno je ponovo izvršiti prevođenje programa i njegovo angažovanje na aplikativnom serveru *Apache Tomcat*.

Navođenjem linka

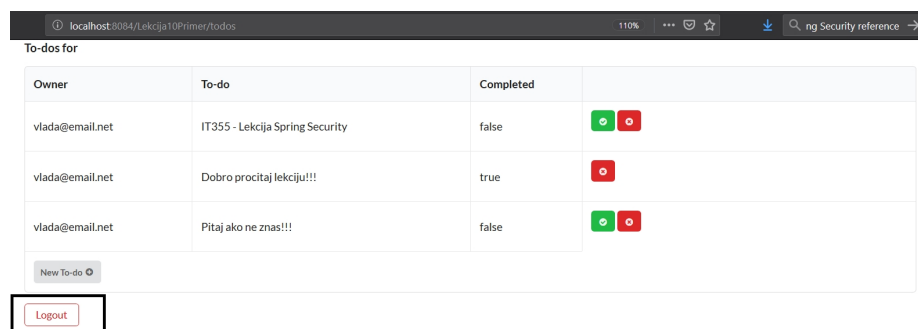
```
localhost:8084/Lekcija10Primer/todos
```

zahteva se otvaranje početne stranice. Zbog definisanih bezbednosnih mehanizama aplikacije, učitava se stranica za prijavljivanje korisnika koja je kreirana tokom prethodnog izlaganja. Izgled ove stranice je prikazan sledećom slikom:



Slika 2.2 Kreirana stranica za prijavljivanje korisnika. [izvor: autor]

Unošenjem podataka za dozvoljenog korisnika, vrši se uspešno prijavljivanje na aplikaciju i učitava se početna stranica. Ona je modifikovana da sada sadrži i dugme za odjavljivanje sa aplikacije. Izgled ove stranice je prikazan sledećom slikom:

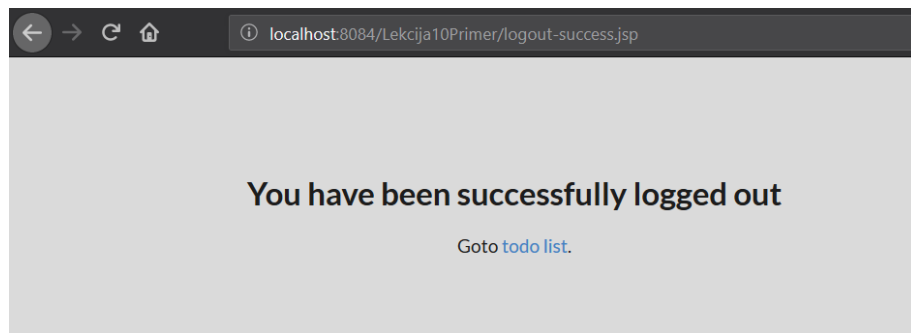


Slika 2.3 Početna stranica sa logout opcijom [izvor: autor]

DODATNA DEMONSTRACIJA I NAPOMENE

Demonstracija odjavljivanja i neuspešnog prijavljivanja

Klikom na dugme za odjavljivanje sa aplikacije otvara se stranica koja obaveštava korisnika da je uspešno odjavljen sa aplikacije. Navedena stranica je prikazana sledećom slikom.

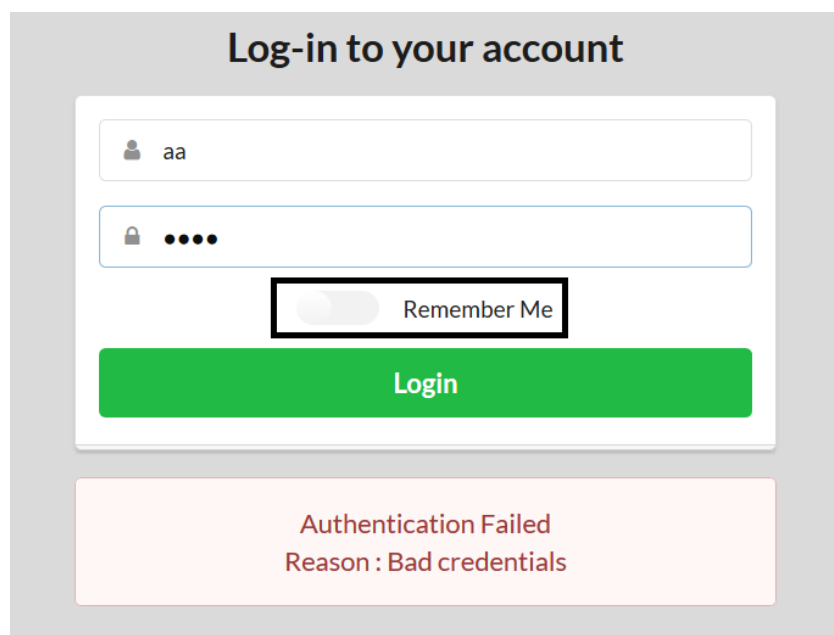


Slika 2.4 Uspešno odjavljivanje sa aplikacije

Moguće je primetiti da prikazana stranica sadrži link za povratak na početnu stranicu. Klikom na ovaj link ponovo se pokreću implementirani bezbednosni mehanizmi što znači da će pre početne stranice ponovo biti učitana stranica za prijavljivanje korisnika.

Sada je moguće testirati ovu stranicu za slučaj neuspešnog prijavljivanja korisnika na aplikaciju. U slučaju unosa korisnika koji nema prava pristupa aplikaciji, na stranici za prijavljivanje dobija se odgovarajuća poruka o grešci, baš kao na sledećoj slici.

Konačno, moguće je primetiti i implementiran check-box (u formi toggle dugmeta) za omogućavanje remember-me funkcionalnosti.



Slika 2.5 Neuspešno prijavljivanje i remember-me kontrola

VIDEO MATERIJAL

Rezime kroz video materijal - HTTP osnovna provera

Spring Boot Essentials 02 - How Spring Security works, HTTP Basic Authentication

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK ZA SAMOSTALNI RAD 1

Pokušajte samostalno

1. Iz OU1 preuzmite priloženi primer;
2. Implemetirajte samostalno sve bezbednosne modifikacije obrađene u ovom objektu učenja;
3. Ukoliko budete imali problema za izradom zadatka, preuzmite kompletno urađen primer nakon ovog OU, analizirajte vaše greške pa pokušajte ponovo da samostalno ispunite navedene zadatke.

▼ Poglavlje 3

Provera korisnika

NAČINI PROVERE KORISNIKA

Bezbednosni mehanizmi veb aplikacije zahtevaju proveru korisnika.

Kao što je pokazano, u prethodnom objektu učenja, kada korisnik pokuša da se poveže na aplikaciju da bi pristupio njenim bezbednim resursima, neophodno je proveriti informacije i prava pristupa u vezi sa korisnikom. Ovo je jedan od ključnih problema razvoja veb aplikacija i okvir Spring Security daje jednostavna i lako primenjiva rešenja navedenog problema.

Spring Security izvodi autentifikaciju korisnika pomoću jednog ili više provajdera autentifikacije, povezanih u formi lanca. Ako bilo koji od njih izvrši uspešnu proveru korisnika, korisnik će moći da se poveže na sistem. Ukoliko nijedan od provajdera nije uspeo da izvrši proveru ili je neki od njih poslao izveštaj da je korisnik isključen, zaključen ili su mu je pristup iz bilo kojeg drugog razloga onemogućen, korisnik neće moći da se poveže na aplikaciju.

U ovoj lekciji će biti prikazani različiti mehanizmi za proveru identiteta korisnika. Biće obrađeno:

- Provera korisnika čiji se login podaci čuvaju u memoriji aplikacije;
- Provera korisnika čiji se login podaci čuvaju u bazi podataka aplikacije;

PROVERA KORISNIKA ČIJI SE PODACI ČUVAJU U MEMORIJI APLIKACIJE

Ukoliko je mali broj korisnika moguće je definisati korisničke podatke u konfiguracionom fajlu.

Ukoliko aplikaciju koristi mali broj korisnika, svega nekoliko, moguće je definisati korisničke podatke u konfiguracionom Spring Security fajlu. Ovi podaci će biti učitani u memoriju aplikacije. Konfiguracioni fajl u ovom slučaju odgovara konfiguracionoj Java klasi, dok je definisanje korisnika, njihovih podataka i prava pristupa aplikaciji povereno poznatoj metodi configure() čiji je listing priložen.

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {

    auth.inMemoryAuthentication().passwordEncoder(NoOpPasswordEncoder.getInstance())
        .withUser("vlada").password("user").authorities("USER")
        .and()
```

```
.withUser("admin").password("admin").authorities("USER", "ADMIN")
.and()

.withUser("student").password("student").disabled(true).authorities("USER");
}
```

Iz priloženog listinga je moguće primetiti da je za definisanje korisničkih detalja upotrebljena metoda `inMemoryAuthentication()`. Korišćena je i metoda `passwordEncoder()` kojom je omogućeno kodiranje i čitanje lozinke korisnika. Primenom metode `withUsers()` definisani su korisnici i njihove uloge u aplikaciji. Za svakog od njih, specificirano je korisničko ime, lozinka, status (uključen ili isključen) i skup prava pristupa aplikaciji. Korisnik sa statusom `disabled(true)` je isključen i ne može da pristupi aplikaciji.

PROVERA KORISNIKA IZ BAZE PODATAKA

Spring Security poseduje ugrađene mehanizme za upite za proveru korisničkih login detalja.

Češći slučaj iz prakse je da se korisnički podaci, za prijavljivanje na aplikaciju, čuvaju u bazi podataka. *Spring Security* poseduje ugrađene mehanizma za pisanje upita za proveru korisničkih detalja iz baze podataka. Po osnovnim podešavanjima, upit sadrži sledeće elemente:

```
SELECT username, password, enabled
FROM users
WHERE username = ?
-----
SELECT username, authority
FROM authorities
WHERE username = ?
```

Da bi *Spring Security* mogao da izvršava ovakve upite i testira identitete korisnika, neophodno je kreirati i odgovarajuće tabele u bazi podataka:

```
CREATE TABLE USERS (
  USERNAME VARCHAR(10) NOT NULL,
  PASSWORD VARCHAR(32) NOT NULL,
  ENABLED SMALLINT,
  PRIMARY KEY (USERNAME)
);
-----
CREATE TABLE AUTHORITIES (
  USERNAME VARCHAR(10) NOT NULL,
  AUTHORITY VARCHAR(10) NOT NULL,
  FOREIGN KEY (USERNAME) REFERENCES USERS
);
```

Nakon unošenja podataka u tabele, moguće je izvršiti navedene upite, a to je prikazano sledećim slikama.

USERNAME	PASSWORD	ENABLED
vlada	user	1
admin	admin	1
student	student	0

Slika 3.1 Tabela USERS [izvor: autor]

USERNAME	AUTHORITY
vlada	USER
student	USER
admin	USER
admin	ADMIN

Slika 3.2 Tabela AUTHORITIES [izvor: autor]

DEKLARISANJE IZVORA PODATAKA

Neophodno je deklarirati izvor podataka za kreiranje konekcije na bazu podataka.

Da bi okvir *SpringSecurity* mogao da pristupi kreiranim tabelama, neophodno je deklarirati izvor podataka za obezbeđivanje konekcije sa bazom podataka. Budući da je ovde akcenat na najnovijim verzijama Spring okvira, insistira se na primeni Java konfiguracija. Ovde je od posebnog značaja primena konfiguracione metode *jdbcAuthentication()* kojoj će biti prosleđen izvor podataka (*data source*).

Sledećim listingom je priložen kod modifikovane konfiguracione klase za omogućavanje provere korisnika iz baze podataka.

```
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {
```

```
@Bean
public DataSource dataSource() {
    return new EmbeddedDatabaseBuilder()
        .setType(EmbeddedDatabaseType.H2)
        .setName("mem:board")
        .addScript("classpath:/schema.sql")
        .addScript("classpath:/data.sql")
        .build();
}

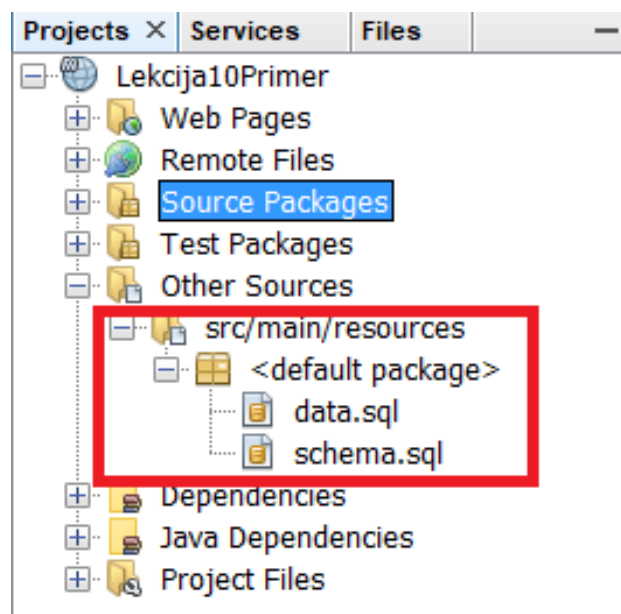
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.jdbcAuthentication().passwordEncoder(NoOpPasswordEncoder.getInstance()).dataSource(dataSource());
}
```

Metoda, obeležena anotacijom **@Bean**, upotrebljena za kreiranje izvora podataka, premeštena je iz konfiguracione klase **TodoWebConfig** u klasu **TodoSecurityConfig**.

DODATNA RAZMATRANJA U VEZI SA KORIŠĆENJEM BAZA PODATAKA

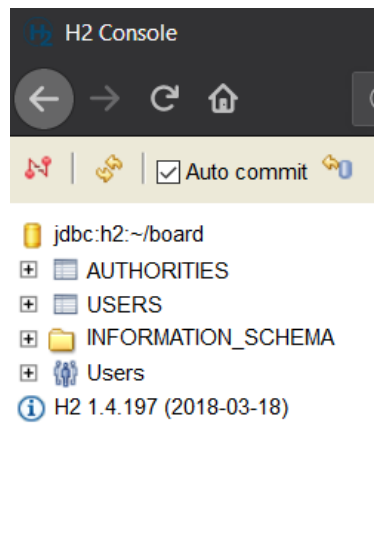
Kreiranje tabela korisnika aplikacije

Pokretanjem modifikovane aplikacije izvršavaju se dva SQL skripta, dodata u folder projekta: [src/main/resources](#). Prvi od njih kreira tabele u bazi podataka *board*, a drugi popunjava njene tabele. Sledećom slikom je prikazana lokacija navedenih SQL datoteka u projektu.



Slika 3.3 Položaj SQL datoteka u projektu [izvor: autor]

Učitavanjem kreiranog izvora podataka izvršavaju se i ovi skriptovi i u *H2* bazi podataka kreirane su tabele i dostupni su korisnici koji mogu da se prijave na aplikaciju. U veb konzoli *H2* baze podataka moguće je videti ove tabele.



Slika 3.4 Tabele u bazi podataka [izvor: autor]

Sledeći SQL listing popunjava navedene tabele baze podataka *board*.

```
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('vlada', 'user', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('admin', 'admin', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('student', 'student', 0);

INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('vlada', 'USER');
INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('student', 'USER');

INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('admin', 'USER');
INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('admin', 'ADMIN');
```

ENKRIPCIJA LOZINKE

Moguće je koristiti BCrypt - jednosmerni heš algoritam za šifriranje lozinki

U svim razmatranim slučajevima, lozinke korisnika su čuvane kao običan tekst. Ovakav pristup je osetljiv na hakerske napade. Da bi problem bio prevaziđen trebalo bi primeniti kodiranje lozinki pre njihovog čuvanja u bazi podataka. Okvir *Spring Security* daje podršku za nekoliko različitih algoritama za kodiranje lozinki. Na primer, moguće je koristiti *BCrypt* - jednosmerni heš algoritam za šifriranje lozinki.

Možda će biti potrebna pomoć za kreiranje **BCrypt** kodova za lozinke. Ovo je moguće uraditi pomoću online **BCrypt** kalkulatora: <https://www.dailycred.com/article/bcrypt-calculator>.

Sledećim listingom je prikazano čuvanje šifriranih lozinki u bazi podataka.

```
/**
 * Author: Vlada
 * Created: 03.11.2018.
 */

/*
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('vlada', 'user', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('admin', 'admin', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('student', 'student', 0);*/

INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('vlada', '$2a$04$j/
SRLDkHICK0JH.x0.GnBuRg6dsurq/GNHTieL.XxNU8ozBwnDOX.', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('admin',
'$2a$04$74kmEHHIgu8C0qYZtD38AeXc9azlmgabuPa0c9/ZS/u6CHlaRP3uK', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('student',
'$2a$04$h17KNRS1qW19Q9puIwpi/OV0xE/Eirvd.aYVDNz/ay.gzDhhU.3ou', 0);

INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('vlada', 'USER');
INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('student', 'USER');

INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('admin', 'USER');
INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('admin', 'ADMIN');
```

INSERT naredbe 11-14 menjaju naredbe 7-9 koje su sada stavljene pod komentar.

Sada je neophodno registrovati zrno tipa *BCryptPasswordEncoder* koje je neophodno za čitanje šifriranih lozinki i proveru njihovog podudaranja sa originalnom lozinkom unetom na formi za prijavljivanje korisnika.

```
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    ...

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .jdbcAuthentication()
```

```
        .passwordEncoder(passwordEncoder())  
        .dataSource(dataSource());  
    }  
}
```

Konačno, *BCryptPasswordEncoder* se predaje metodi *passwordEncoder()* u okviru metode *configure()*.

PROVERA KORISNIKA - DEMO

Provera implementiranog koda za proveru korisnika.

Sledećom slikom su prikazane validne lozinke i njihovi kodovi koji se čuvaju u bazi podataka.

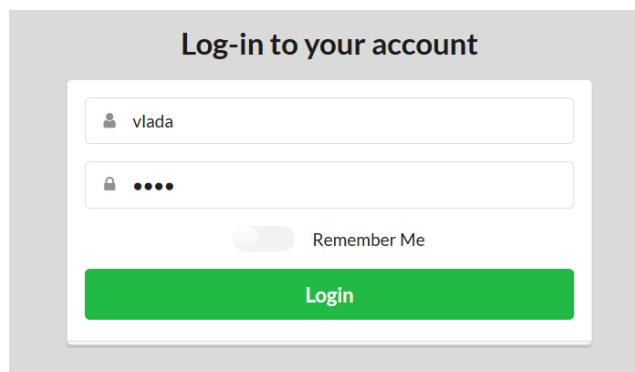
```
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('vlada', 'user', 1);  
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('admin', 'admin', 1);  
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('student', 'student', 0);  
  
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('vlada', '$2a$04$1/SB1DxHICk0JH_wO_GnBuRq6deurn/GNHTieL.XeNUSozBBnDOX', 1);  
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('admin', '$2a$04$74keFHHtgu8CqYz7n38AeXc2a1mcbuPa0c9/75/u6Ch1aRF3u6', 1);  
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('student', '$2a$04$hl7ENRSlqW19Q9puIwpl/OV0xE/Eirvd.aYVDNz/ay.gzDhhU.3ou', 0);
```

Slika 3.5 Lozinke i njihovi BCrypt kodovi [izvor: autor]

Sada je moguće ponovo prevesti aplikaciju, izvršiti njeno angažovanje na aplikativnom serveru i pratiti rezultate izvršavanja. Opet se navodi link, u veb pregledaču:

```
localhost:8084/Lekcija10Primer/todos
```






Zbog podešenih bezbednosnih mehanizama vrši se preusmeravanje na stranicu za prijavu korisnika.




Slika 3.6 Prijavljivanje korisnika [izvor: autor]

Unose se podaci za korisničko ime i lozinku. Ovde na scenu stupa *BCryptPasswordEncoder* koji proverava da li se uneta lozinka podudara sa šifrom koju je generisao *BCrypt* algoritam. Ukoliko su ovi podaci saglasni, korisnik ima pravo da se uloguje na aplikaciju.

To-dos for

Owner	To-do	Completed	
vlada@email.net	IT355 - Lekcija Spring Security	false	 
vlada@email.net	Dobro procitaj lekciju!!!	true	
vlada@email.net	Pitaj ako ne znas!!!	false	 

New To-do 

[Logout](#)

Slika 3.7 Korisnik je uspešno prijavljen na aplikaciju [izvor: autor]

VIDEO MATERIJAL

Rezime kroz video materijal - Spring Security JDBC Authentication with Password Encryption

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK ZA SAMOSTALNI RAD 2

Pokušajte samostalno da uradite

1. Preuzmite urađeni projekat iz prethodnog objekta učenja;
2. Ugradite programske modifikacije koje su navedene u OU *Provera korisnika*.

▼ Poglavlje 4

Upravljanje kontrolom pristupa

PROVERA KONTROLE PRISTUPA

Odluka da li je nekom korisniku dozvoljen pristup, ili ne, naziva se odluka kontrole pristupa.

Tokom procesa autentifikacije, aplikacija će dozvoliti pristup uspešno proverenom korisniku. Kada korisnik pokuša da pristupi resursima aplikacije, aplikacija bi trebalo da odluči da li je tom resursu dozvoljen pristup ili ne osnovu prava pristupa korisnika koji pokušava da im pristupi.

Odluka da li je nekom resursu dozvoljen pristup ili ne u aplikaciji naziva se odluka kontrole pristupa (access control decision). Ova odluka se kreira na osnovu statusa provere korisnika, prirode resursa i atributa pristupa. U ovu svrhu, *Spring Security* koristi menadžer kontrole pristupa koji implementira interfejs **AccessDecisionManager**. *Spring Security* nudi tri pogodna menadžera koja se baziraju na glasačkom pristupu.

PRINCIP GLASANJA

Spring Security nudi tri pogodna menadžera pristupa koja se baziraju na glasačkom pristupu.

Spring Security koristi menadžer kontrole pristupa koji implementira interfejs **AccessDecisionManager**. *Spring Security* nudi tri pogodna menadžera pristupa koja se baziraju na glasačkom pristupu.

Access Decision Manager	Specificira kada se dodeljuje pristup
AffirmativeBased	Najmanje jedan glasač je dao glas za odobrenje pristupa
ConsensusBased	Glasači donose odluku konsenzusom za odobrenje pristupa
UnanimousBased	Nijedan glasač nije klasao protiv pristupa - uzdržani ili za pristup

Slika 4.1 Menadžeri kontrole pristupa [izvor: autor]

Svaki od ovih menadžera zahteva grupu glasača koja mora da bude podešena za donošenje odluke o kontroli pristupa. Svaki od njih mora da implementira interfejs **AccessDecisionVoter**. Rezultati glasanja su predstavljaju konstantna polja **ACCESS_GRANTED**, **ACCESS_DENIED** i **ACCESS_ABSTAIN** (*uzdržan*) definisana u interfejsu *AccessDecisionVoter*.

Po osnovnim podešavanjima, ukoliko nijedan menadžer nije eksplicitno specificiran, *Spring Security* će automatski konfigurisati menadžer **AffirmativeBased**, sa sledeća dva glasača:

- **RoleVoter** - glasa o odluci kontrole pristupa na osnovu uloge korisnika. Obradiće pristupne attribute koji počinju prefiksom *Role_*. Glasće za pristup ukoliko korisnik poseduje identičnu ulogu koja je zahtevana za pristup, u suprotnom će glasati protiv. Ukoliko atribut ne sadrži prefiks *Role_* ovaj glasač će biti uzdržan od glasanja.
- **AuthenticatedVoter** - glasa o odluci kontrole pristupa na osnovu autentifikacije korisnika. Ovaj glasač će obraditi isključivo neki od sledećih atributa: **IS_AUTHENTICATED_FULLY**, **IS_AUTHENTICATED_REMEMBERED** i **IS_AUTHENTICATED_ANONYMOUSLY**. Glas se pristup resursima aplikacije ukoliko je nivo autentifikacije korisnika viši od posmatranog atributa. Od najvišeg ka najnižem nivou, nivoi autentifikacije su poređani na sledeći način: **fully authenticated**, **authentication remembered** i **anonymously authenticated**.

PODRAZUMEVANI MENADŽER ODLUKE

Spring će automatski podesiti menadžer kontrole pristupa ukoliko nijedan nije specificiran.

Po osnovnim podešavanjima, *Spring Security* okvir će automatski podesiti menadžera za donošenje odluke pristupa ukoliko nijedan nije specificiran. Podrazumevani menadžer odluke je ekvivalentan menadžeru definisanim sledećim *Java* konfiguracionim kodom:

```
@Bean
public AffirmativeBased accessDecisionManager() {
    List<AccessDecisionVoter> decisionVoters = Arrays.asList(new RoleVoter(), new
AuthenticatedVoter());
    return new AffirmativeBased(decisionVoters);
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .accessDecisionManager(accessDecisionManager())
        ...
}
```

Slika 4.2 Konfiguracija menadžera odluke [izvor: autor]

Podrazumevani menadžer odluke i njegovi glasači, trebalo bi da zadovolje većinu standardnih zahteva autorizacije. Ukoliko ne zadovoljavaju specifične korisnički definisane zahteve, moguće je na sličan način definisati korisnički prilagođenog glasača.

IMPLEMENTACIJA GLASAČA

*Kreirana klasa glasača mora da implementira interfejs **AccessDecisionVoter**.*

Za podršku prethodnom izlaganju moguće je kreirati glasača koji donosi odluku o pristupu na osnovu korisnikove IP adrese. U tu svrhu će biti kreirana klasa *IpAddressVoter* koja mora da implementira interfejs *AccessDecisionVoter*.

```
/**
 *
 * @author Vlada
 */
public class IpAddressVoter implements AccessDecisionVoter<Object> {

    private static final String IP_PREFIX = "IP_";
    private static final String IP_LOCAL_HOST = "IP_LOCAL_HOST";

    @Override
    public boolean supports(ConfigAttribute attribute) {
        return (attribute.getAttribute() != null) &&
attribute.getAttribute().startsWith(IP_PREFIX);
    }

    @Override
    public boolean supports(Class<?> clazz) {
        return true;
    }

    @Override
    public int vote(Authentication authentication, Object object,
Collection<ConfigAttribute> configList) {
        if (!(authentication.getDetails() instanceof WebAuthenticationDetails)) {
            return ACCESS_DENIED;
        }

        WebAuthenticationDetails details = (WebAuthenticationDetails)
authentication.getDetails();
        String address = details.getRemoteAddress();

        int result = ACCESS_ABSTAIN;

        for (ConfigAttribute config : configList) {
            result = ACCESS_DENIED;

            if (Objects.equals(IP_LOCAL_HOST, config.getAttribute())) {
                if (address.equals("127.0.0.1") ||
address.equals("0:0:0:0:0:0:0:1")) {
                    return ACCESS_GRANTED;
                }
            }
        }

        return result;
    }
}
```

Iz priloženog listinga moguće je primetiti da će glasač obrađivati samo attribute koji počinu prefiksom `IP_`. U ovom trenutku, podržan je samo pristupni atribut `IP_LOCAL_HOST`. Ukoliko korisnik predstavlja veb klijenta čija je IP adresa jednaka `127.0.0.1` (*Windows*) ili `0:0:0:0:0:0:1` (*Linux*) njemu može biti odobren pristup. U suprotnom, ovaj glasač glasa protiv pristupa korisnika veb aplikaciji. Ukoliko resurs ne poseduje pristupni atribut koji počinje sa `IP_`, glasač će biti uzdržan od glasanja.

DEFINISANJE KORISNIČKOG MENADŽERA ODLUKE

Moguće je uvesti vlastitog menadžera odluke za uključivanje definisanog glasača.

Kao nastavka prethodnom izlaganju moguće je uvesti vlastitog menadžera odluke koji će uključiti prethodno definisanog glasača. Navedenom odgovara sledeći konfiguracioni listing:

```
@Bean
public AffirmativeBased accessDecisionManager() {
    List<AccessDecisionVoter> decisionVoters = Arrays.asList(new RoleVoter(),
        new AuthenticatedVoter(), new IpAddressVoter());
    return new AffirmativeBased(decisionVoters);
}
```

Ako se želi da dozvoli pristup korisnicima računara na kojem se izvršava veb server (na primer server administratori) zadacima brisanja *to-do* stavki, bez prijavljivanja na aplikaciju, moguće je redefinisati poznatu `configuration()` metodu dodavanjem koda na sledeći način:

```
http.authorizeRequests()
    .accessDecisionManager()
    .antMatchers(HttpMethod.DELETE, "/todos*").access("ADMIN,IP_LOCAL_HOST");
```

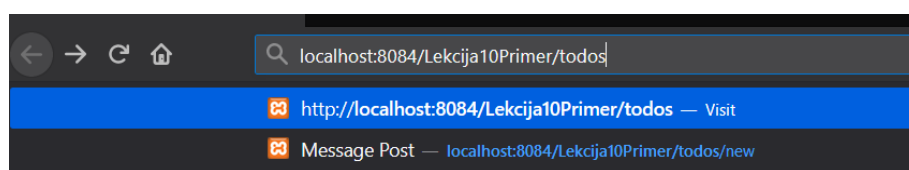
Sada, prilikom pozivanja URL direktno, *to-do* zadaci će biti uklonjeni. Za pristup zadacima kroz veb interfejs i dalje je neophodno prijavljivanje na sistem.

UPRAVLJANJE KONTROLOM PRISTUPA - DEMO

Demonstracija direktnog pristupa sa LOCAL_HOST

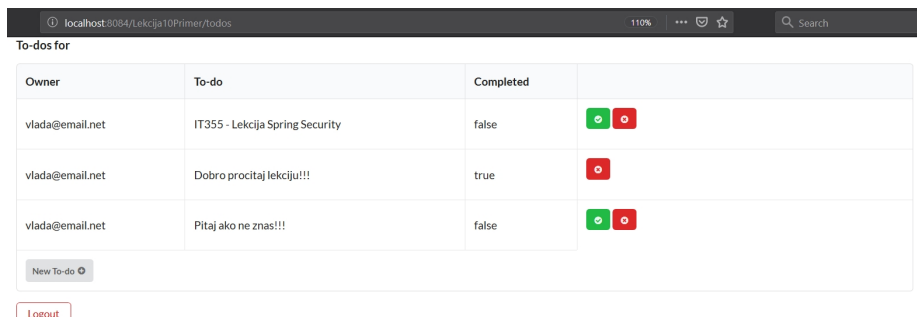
Kao što je istaknuto menadžerom odluke pristupa aplikaciji je dozvoljen direktan pristup korisnicima koji pristupaju aplikaciji sa *IP* adrese koja odgovara *localhost*-u.

Navedeno započinje pozivom stranice iz veb pregledača kao na sledećoj slici.



Slika 4.3 Pristup todos stranici sa localhost-a [izvor: autor]

Ovakvim korisnicima je dozvoljen direktan pristup i oni bez problema dolaze do tražene stranice, kao na sledećoj slici:



Slika 4.4 Direktan pristup todos.jsp stranici [izvor: autor]

ZADATAK ZA SAMOSTALNI RAD 3

Pokušajte samostalno da upravljate resursima aplikacije

1. Koristite primer *Board -Lekcija10Primer3* na kojem ste do sad radili;
2. Ugradite programske modifikacije koje su navedene u OU *Kontrola pristupa resursima aplikacije*.

Ukoliko budete imali problema da samostalno rešite postavljeni zadatak, preuzmite urađeni primer nakon ovog objekta učenja, dobro ga analizirajte, pa ponovo pokušajte samostalno da uradite zadatak.

▼ Poglavlje 5

Zaštita poziva metoda

PROBLEM POZIVA METODA

Ponekad je neophodno obaviti obezbeđivanje poziva metoda u servisnom nivou.

Ponekad je kao alternativu, ili dopunu, obezbeđivanju URL pristupa na veb nivou, neophodno obaviti obezbeđivanje poziva metoda u servisnom nivou. Na primer, u slučaju kada jedan kontroler mora da pozove više metoda iz servisnog nivoa, neophodno je primeniti posebno podešene bezbednosne kontrole ovih metoda.

Spring Security omogućava osiguravanje poziva metoda na deklarativan način. Neophodno je obeležiti metode deklarisanе u interfejsu zrna ili implementacionoj klasi nekom od anotacija: @Secured, @PreAuthorize/PostAuthorize ili @PreFilter/@PostFilter, a zatim obezbediti bezbednosne mehanizme za njih primenom anotacije @EnableGlobalMethodSecurity annotation.

OBEZBEĐIVANJE POZIVA METODE PRIMENOM ANOTACIJA

Poslednji pristup obezbeđivanju poziva metoda jeste primena anotacije @Secured.

Poslednji pristup obezbeđivanju poziva metoda jeste primena anotacije @Secured. Na primer, moguće je obeležiti ovom anotacijom metode u klasi TodoServiceImpl i specificirati pristupne attribute, kao njenu vrednost, čiji je tip String.

```
class TodoServiceImpl implements TodoService {  
  
    private final TodoRepository todoRepository;  
  
    TodoServiceImpl(TodoRepository todoRepository) {  
        this.todoRepository = todoRepository;  
    }  
  
    @Override  
    @Secured({"ROLE_USER", "ROLE_GUEST"})  
    public List<Todo> listTodos() {  
        return todoRepository.findAll();  
    }  
}
```

```

@Override
@Secured("ROLE_USER")
public void save(Todo todo) {
    this.todoRepository.save(todo);
}

@Override
@Secured("ROLE_USER")
public void complete(long id) {
    Todo todo = findById(id);
    todo.setCompleted(true);
    todoRepository.save(todo);
}

@Override
@Secured({"ROLE_ADMIN", "IP_LOCAL_HOST"})
public void remove(long id) {
    todoRepository.remove(id);
}

@Override
@Secured({"ROLE_USER", "ROLE_GUEST"})
public Todo findById(long id) {
    return todoRepository.findOne(id);
}
}

```

Konačno, neophodno je obezbediti mehanizam zaštite metoda. Ovo se postiže dodavanjem anotacije `@EnableGlobalMethodSecurity` na konfiguracionu klasu. Ukoliko je neophodno koristiti anotaciju `@Secured`, na prikazani način, neophodno je podesiti atribut `securedEnabled` na vrednost `true`, baš kao u sledećem listingu.

```

@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class TodoWebConfiguration { ... }

```

Veomaje važno dodati anotaciju `@EnableGlobalMethodSecurity` u konfiguraciju konteksta aplikacije koji sadrži zrna koja je neophodno zaštititi.

VIDEO MATERIJAL

Rezime kroz video materijal - Spring Security Tutorial: Authorization with Expressions - on Methods

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK ZA SAMOSTALNI RAD 4

Pokušajte samostalno

1. Preuzmite poslednju verziju aktuelnog primera;
2. Ugradite programske modifikacije koje su navedene u OU *Bezbednost poziva metoda*.

▼ Poglavlje 6

Upravljanje bezbednošću u pogledima

PROBLEM BEZBEDNOSTI POGLEDA

Rešavanje problema prikaza informacija o autentifikaciji korisnika u pogledima web aplikacije.

Ponekad je neophodno prikazati informacije u vezi sa autentifikacijom korisnika u pogledima veb aplikacije. Dalje, neophodno je odrediti sadržaj pogleda na osnovu ovlašćenja korisnika.

[Spring Security](#) obezbeđuje [JSP](#) biblioteku tagova za upravljanje bezbednošću u JSP pogledima. Biblioteka uključuje tagove koji mogu da prikažu informaciju o autentifikaciji korisnika i kreiraju sadržaj pogleda u zavisnosti od prava pristupa korisnika.

U primeru će biti prikazani odgovarajući podaci o korisniku u zaglavlju stranice za listanje poruka, na primer [todos.jsp](#). Prvo je neophodno importovati [Spring Security](#) definiciju biblioteke tagova.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
```

Tag `<sec:authentication>` prikazuje objekat autentifikacije tekućeg korisnika za prikazivanje njegovih osobina. Naziv osobine je moguće specificirati u atributu **property**. Na primer, moguće je dodati naziv korisnika kroz ovaj atribut na sledeći način:

```
<h4>Todos for <sec:authentication property="name" /></h4>
```

Kao dodatak mogućnosti direktnog dodavanja osobine autentifikacije korisnika, ovaj tag daje podršku čuvanju ove osobine u [JSP](#) promenljivoj. Naziv ove promenljive je specificiran kroz var atribut. Na primer, moguće je sačuvati osobinu prava pristupa koja sadrži dodeljena prava pristupa korisniku u [JSP](#) promenljivoj i izvršiti njihova dodavanja preko petlje `<c:forEach>`. Dalje, moguće je definisati oblast promenljive preko atributa [ascope](#).

USLOVNO PRIKAZIVANJE SADRŽAJA POGLEDA

Za sadržaj pogleda u zavisnosti od korisnikovih ovlašćenja obezbeđen je security:authorize tagom

Ukoliko se želi prikazati sadržaj pogleda u zavisnosti od korisnikovih ovlašćenja moguće je koristiti `<sec:authorize>` tag. Na primer, neophodno je odlučiti da li će biti prikazani autori poruka na osnovu njihovih ovlašćenja:

```
<td>
<sec:authorize ifAllGranted="ROLE_ADMIN,ROLE_USER">${todo.owner}</sec:authorize>
</td>
```

Ukoliko se želi da se priloženi sadržaj prikazuje samo onda kada su istovremeno korisniku odobrena izvesna ovlašćenja, biće korišćen atribut ***ifAllGranted***. U suprotnom, ukoliko sadržaj može biti prikazan sa bilo kojim ovlašćenjem, biće korišćen atribut ***ifAnyGranted***.

```
<td>
<sec:authorize ifAnyGranted="ROLE_ADMIN,ROLE_USER">${todo.owner}</sec:authorize>
</td>
```

Ukoliko se želi da se priloženi sadržaj prikazuje kada korisniku nisu odobrena izvesna ovlašćenja, biće korišćen atribut ***ifNotGranted***.

```
<td>
<sec:authorize ifNotGranted="ROLE_ADMIN,ROLE_USER">${todo.owner}</sec:authorize>
</td>
```

ZADATAK ZA SAMOSTALNI RAD 5

Pokušajte samostalno

1. Preuzmite najnoviju verziju aktuelnog projekta;
2. Ugradite programske modifikacije koje su navedene u OU *Upravljanje bezbednošću pogleda*.

▼ Poglavlje 7

OAuth2 autorizacija i server resursa

UVOD U OAUTH 2

OAuth 2 predstavlja metod autorizacije za obezbeđivanje pristupa resursima preko HTTP protokola.

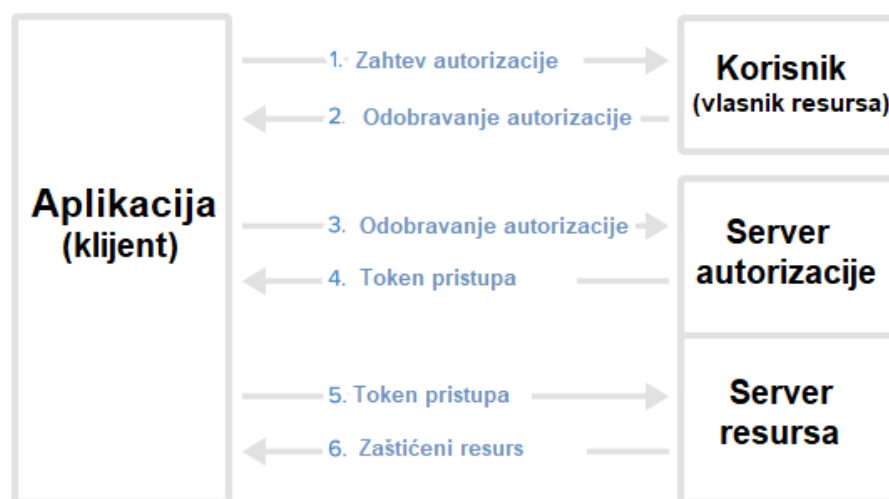
OAuth 2 predstavlja metod autorizacije za obezbeđivanje pristupa resursima preko HTTP protokola. Primarno, OAuth 2 omogućava aplikacijama "treće strane" dobijanje ograničenog pristupa HTTP servisima:

- bilo u ime vlasnika resursa, organizovanjem interakcije odobrenja između vlasnika resursa i HTTP usluge;
- ili omogućavanjem nezavisnoj aplikaciji da u svoje ime dobije pristup.

OAuth 2 definiše četiri različite uloge:

- vlasnik resursa (*resource owner*) - Korisnik koji koristi aplikaciju;
- klijent (*client*) - aplikacija (koju korisnik koristi) koja zahteva korisničke podatke sa servera resursa;
- server resursa (*resource server*) - skladište korisničkih podataka i HTTP servisa koje ima mogućnost vraćanja podataka proverenim klijentima;
- server autorizacije (*authorization server*) - odgovoran za proveru identiteta korisnika i dodelu tokena autorizacije. Token prihvata server resursa i proverava identitet korisnika.

Sledećom sliko je prikazana veza između navedenih OAuth 2 uloga:



Slika 7.1 OAuth 2 uloge [izvor: autor]

TOKENI PRISTUPA I OSVEŽAVANJA

Tokeni pristupa i osvežavanja su dva karakteristična tipa tokena za OAuth 2.

Token pristupa (*access token*) je string koji reprezentuje ovlašćenje dodeljeno klijentu. Tokeni predstavljaju specifične oblasti i trajanje pristupa, odobreno od vlasnika resursa i nametnutih od servera resursa i servera autorizacije.

Token osvežavanja (*refresh token*) klijentu izdaje, zajedno sa pristupnim tokenom, autorizacioni server i koristi se za dobijanje novog pristupnog tokena kada trenutni pristupni token postane nevažeći ili istekne. Takođe, ovaj token se koristi za dobijanje dodatnih tokena pristupa u identičnoj ili užoj oblasti (tokeni pristupa mogu da imaju kraći životni ciklus i manji broj dozvola nego što je odobrio server autorizacije).

Dužnost tokena pristupa je omogućavanje pristupa podacima pre njegovog isteka. Token osvežavanja ima za dužnost zahtevanje novih tokena pristupa kada postojeći postane nevažeći.

SERVER AUTORIZACIJE

Server autorizacije je odgovoran za proveru identiteta korisnika i dodelu tokena autorizacije.

Server autorizacije je odgovoran za proveru identiteta korisnika i dodelu tokena autorizacije. Token prihvata *server resursa* i proverava identitet korisnika.

Uvodimo prateći primer. Klasa koja implementira server autorizacije mora da bude obeležena anotacijom `@EnableAuthorizationServer` i da nasledi klasu `AuthorizationServerConfigurerAdapter`, kao u sledećem listingu:

```
package com.metropolitan.auth2demo.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;
import
org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;
import
org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;
import
```

```
org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationS
erverSecurityConfigurer;
/**
 *
 * @author Vlada
 */
@Configuration
@EnableAuthorizationServer
public class OAuth2AuthorizationServer extends AuthorizationServerConfigurerAdapter
{
    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    @Override
    public void configure(AuthorizationServerSecurityConfigurer security) throws
Exception {
        security
            .tokenKeyAccess("permitAll()")
            .checkTokenAccess("isAuthenticated()")
            .allowFormAuthenticationForClients();
    }

    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
        clients
            .inMemory()
            .withClient("clientapp")
            .secret(passwordEncoder.encode("123456"))
            .authorizedGrantTypes("password", "authorization_code", "refresh_token")
            .authorities("READ_ONLY_CLIENT")
            .scopes("read_profile_info")
            .resourceIds("oauth2-resource")
            .redirectUri("http://localhost:8080/login")
            .accessTokenValiditySeconds(5000)
            .refreshTokenValiditySeconds(50000);
    }
}
```

Spring Security OAuth koristi dve krajnje tačke (*endpoint*) za proveru tokena:

- [/oauth/check_token](#);
- [/oauth/token_key](#);

koje su, po osnovnim podešavanjima, pod zaštitom metoda: [denyAll\(\)](#), [tokenKeyAccess\(\)](#) i [checkTokenAccess\(\)](#).

[ClientDetailsServiceConfigurer](#) se koristi za definisanje ugrađenih (*in memory*) ili *JDBC* implementacija servisa detalja korisnika. U konkretnom slučaju je primenjena ugrađena implementacija sa sledećim važnim atributima:

- [clientId](#) – (obavezan) *id* klijenta.
- [secret](#) – (obavezan za pouzdane (*trusted*) klijente) tajna informacija klijenta, ukoliko postoji.

- [scope](#) - Oblast na koju je klijent ograničen. Ukoliko je oblast nedefinisana ili prazna (osnovna podešavanja), oblast ne ograničava klijenta.
- [authorizedGrantTypes](#) - Vrste dozvola za koje klijent poseduje ovlašćenje. Po osnovnim podešavanjima je prazno.
- [authorities](#) - Regularna Spring Security ovlašćenja dodeljena klijentu;
- [redirectUri](#) - mora da bude apsolutna putanja - preusmerava klijenta na novu krajnju tačku (ovde /login).

SERVER RESURSA

Server resursa predstavlja skladište korisničkih podataka i HTTP servisa.

Server resursa ([resource server](#)) predstavlja skladište korisničkih podataka i [HTTP](#) servisa koje ima mogućnost vraćanja podataka proverenim klijentima.

Klasa koja implementira [serverresursa](#) mora da bude obeležena anotacijom [@EnableResourceServer](#) i da nasledi klasu [ResourceServerConfigurerAdapter](#), kao u sledećem listingu:

```
package com.metropolitan.auth2demo.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.oauth2.config.annotation.web.configuration.EnableResour
ceServer;
import
org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServ
erConfigurerAdapter;
/**
 *
 * @author Vlada
 */
@Configuration
@EnableResourceServer
public class OAuth2ResourceServer extends ResourceServerConfigurerAdapter
{
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
            .antMatchers("/").permitAll()
            .antMatchers("/api/v1/**").authenticated();
    }
}
```

Navedena konfiguracija obezbeđuje zaštitu za sve krajnje tačke koje počinju sa [/api](#). Svim ostalim krajnjim tačkama se pristupa slobodno.

Server resursa, takođe, obezbeđuje mehanizam provere samih korisnika. U većini slučajeva to će biti prijava zasnovana na formama. Sledi konfiguraciona klasa kojom se definiše provera korisnika bazirana na formama i koja otključava linkove (URL) autorizacije metodom `permitAll()`.

```
package com.metropolitan.auth2demo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.annotation.Order;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

/**
 *
 * @author Vlada
 */
@Configuration
@Order(1)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.requestMatchers()
            .antMatchers("/login", "/oauth/authorize")
            .and()
            .authorizeRequests()
            .anyRequest().authenticated()
            .and()
            .formLogin().permitAll();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
            .withUser("vladimir")
            .password(passwordEncoder().encode("123456"))
            .roles("USER");
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

OAUTH2 ZAŠTIĆENI REST RESURSI

Za potrebe demonstracije biće kreiran jedan API.

Za potrebe demonstracije biće kreiran jedan API koji će vratiti naziv (*name*) i *email* prijavljenog korisnika. Sledi listing klase *RestResource.java*:

```
package com.metropolitan.auth2demo.controller;

import com.metropolitan.auth2demo.model.UserProfile;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.User;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
/**
 *
 * @author Vlada
 */
@Controller
public class RestResource
{
    @RequestMapping("/api/users/me")
    public ResponseEntity<UserProfile> profile()
    {
        //Kreira podatke za demonstraciju
        User user = (User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String email = user.getUsername() + "@primer.com";

        UserProfile profile = new UserProfile();
        profile.setName(user.getUsername());
        profile.setEmail(email);

        return ResponseEntity.ok(profile);
    }
}
```

Pored priloženog listinga za REST kontroler neophodno je priložiti i listing kojim je definisana modelska klasa *UserProfile.java*:

```
package com.metropolitan.auth2demo.model;

/**
 *
 * @author Vlada
 */
public class UserProfile
{
    private String name;
    private String email;
```

```
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getEmail() {
    return email;
}

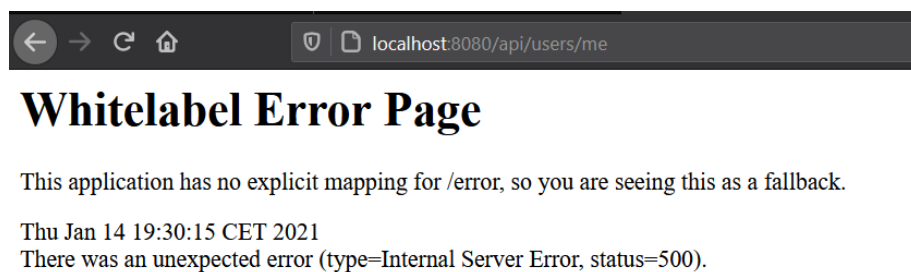
public void setEmail(String email) {
    this.email = email;
}

@Override
public String toString() {
    return "UserProfile [name=" + name + ", email=" + email + "]\n";
}
}
```

DEMONSTRACIJA

*Aplikacije "treće strane" ne mogu da pristupe API - njima je potreban **oauth2 token**.*

Ako se pogleda priloženi kontroler, postoji API link <http://localhost:8080/api/users/me> kojim je moguće obaviti pristup direktnim navođenjem korisničkog imena i lozinke putem login forme. Međutim, aplikacije "treće strane" ne mogu da pristupe API - ju kao što se to radi u na standardan način u veb pregledačima. Njima je potreban **oauth2** token.

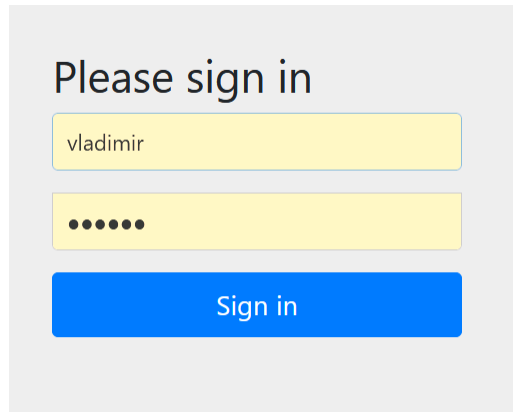


Slika 7.2 OPristup bez dodeljenog tokena [izvor: autor]

Dakle, neophodno je obaviti prvi korak za dobijanje odobrenja od vlasnika resursa iz URL-a:

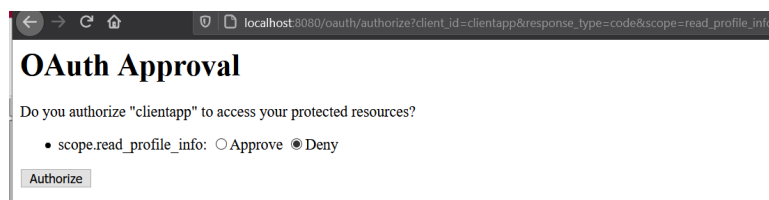
```
http://localhost:8080/oauth/  
authorize?client_id=clientapp&response_type=code&scope=read_profile_info
```

Aktiviranjem ovog linka vrši se navigacija ka [login](#) stranici. Iz priloženog listinga klase [SecurityConfig](#) moguće je primetiti da su za potrebe primera definisani korisničko ime i lozinka vrednostima "Vladimir" i "123456", respektivno.



Slika 7.3 Login stranica [izvor: autor]

Nakon uspešnog prijavljivanja korisnika, vrši se preusmeravanje na stranicu na kojoj je moguće dodeliti prava pristupa aplikacijama "treće strane".



Slika 7.4 Odobravanje pristupa aplikacijama "treće strane" [izvor: autor]

Nakon odobrenog pristupa vrši se preusmeravanje ka linku oblika: `http://localhost:8080/login?code=jKRL71` gde vrednost "[jKRL71](#)" predstavlja autorizacioni kod aplikacije "treće strane".

DOBIJANJE TOKENA PRISTUPA OD SERVERA AUTORIZACIJE

Aplikacija može da upotrebi pravo pristupa za dobijanje tokena pristupa.

Sada aplikacija može da upotrebi pravo pristupa za dobijanje tokena pristupa. Neophodno je obaviti sledeći zahtev primenom nekog [REST](#) klijenta (npr. [Postman](#), [RESTClient](#) za [Firefox](#) i sl.):

```
http://localhost:8080/oauth/token
```

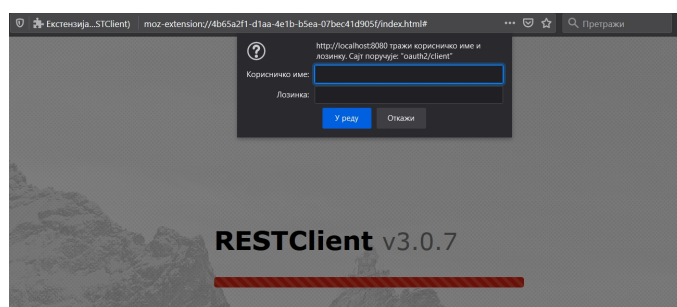
Headers:

Content-Type: application/x-www-form-urlencoded
authorization: Basic Y2xpZW50YXBwOjEyMzQ1Ng==

Form data - application/x-www-form-urlencoded:

grant_type=authorization_code
code=jKRL71
redirect_uri=http://localhost:8080/login

Odgovarajući **POST** zahtev će zatražiti kredencijale klijent aplikacije na sledeći način (korišćen je **RESTClient** za **Firefox**):

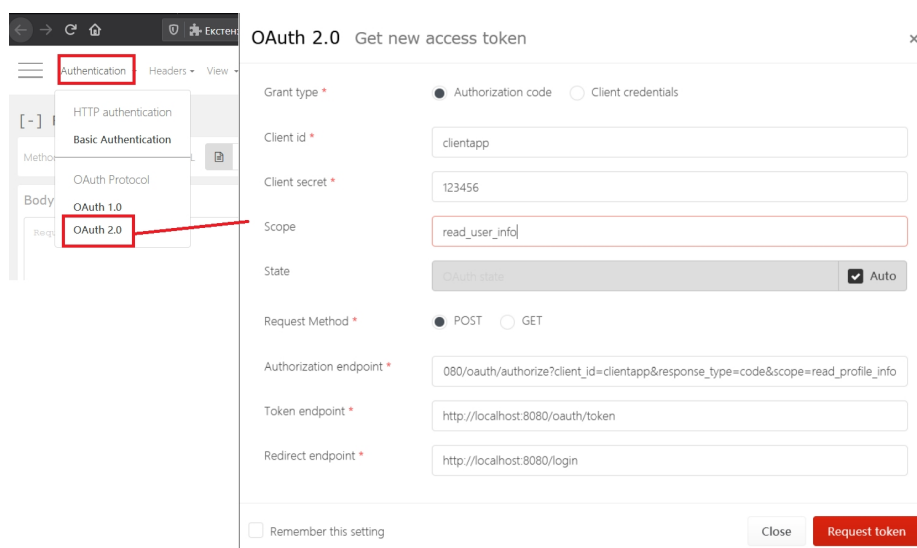


Slika 7.5 Kredencijali za klijent aplikaciju [izvor: autor]

Navedeno je moguće uraditi i pomoću odgovarajućeg cURL upita:

```
curl -X POST --user clientapp:123456 http://localhost:8080/oauth/token
-H "content-type: application/x-www-form-urlencoded"
-d
"code=jKRL71&grant_type=authorization_code&redirect_uri=http%3A%2F%2Flocalhost%3A8080%2Flogin&scope=read_user_info"
```

Moguće je i koristiti **OAuth 2** čarobnjak **REST** klijent alata **RESTClient** za **Firefox**:



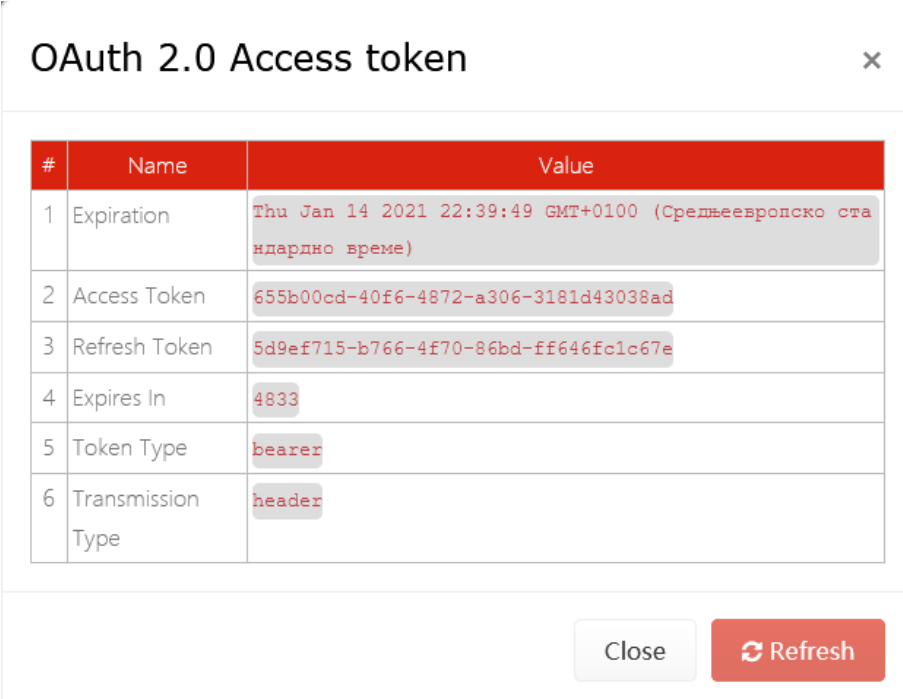
Slika 7.6 OAuth 2 čarobnjak REST klijent alata RESTClient za Firefox [izvor: autor]

PRIKAZ DOBIJENIH TOKENA

Klik na dugme Request Token i biće upućen zahtev za tokenom.

Ako se osvrnemo na prethodnu sliku, sve što je potrebno jeste klik na dugme *Request Token* i biće upućen zahtev za tokenom.

Sledećom slikom su prikazani kreirani tokeni pristupa i osvežavanja, zajedno sa vremenom validnosti pristupnog tokena.

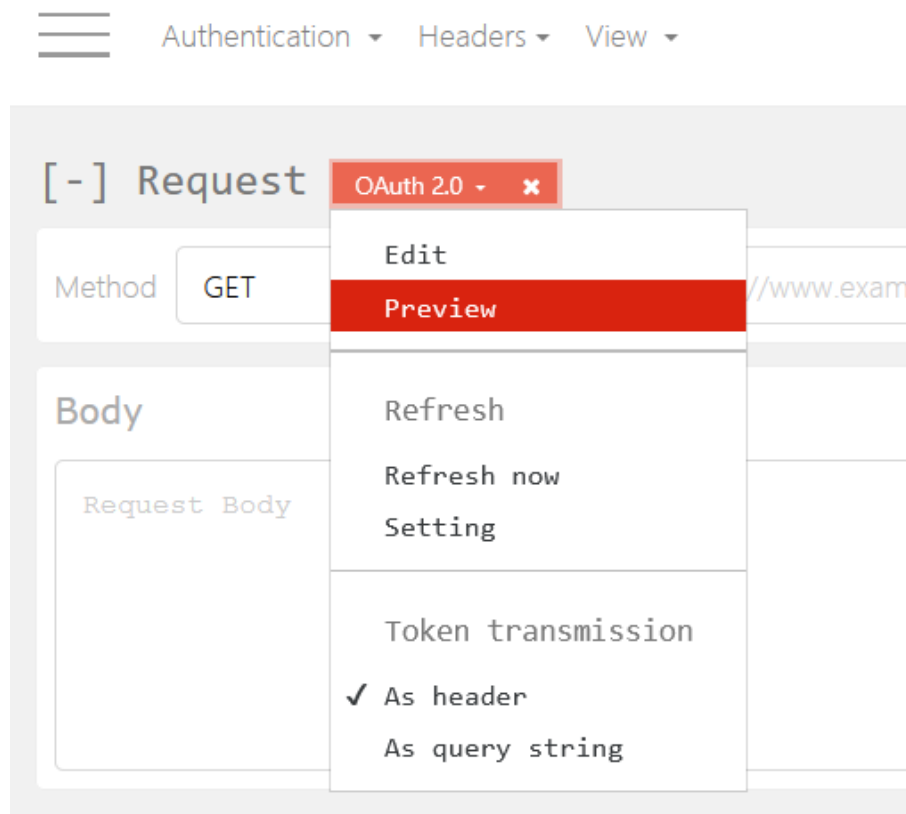


The screenshot shows a window titled "OAuth 2.0 Access token" with a close button (X) in the top right corner. Inside the window is a table with 3 columns: "#", "Name", and "Value". The table contains 6 rows of data. Below the table are two buttons: "Close" and "Refresh".

#	Name	Value
1	Expiration	Thu Jan 14 2021 22:39:49 GMT+0100 (Средњеевропско стандардно време)
2	Access Token	655b00cd-40f6-4872-a306-3181d43038ad
3	Refresh Token	5d9ef715-b766-4f70-86bd-ff646fc1c67e
4	Expires In	4833
5	Token Type	bearer
6	Transmission Type	header

Slika 7.7 Prikaz dobijenih tokena [izvor: autor]

Da bi mogli da dobijemo ovaj izveštaj, neophodno je za kreirani upit (slika 6) izabrati opciju *Preview*, baš kao na sledećoj slici:



Slika 7.8 Zahtev za pregledom kreiranih tokena [izvor: autor]

PRISTUP KORISNIČKIM PODACIMA SA SERVERA RESURSA

Kada smo dobili pristupni token, možemo otići na server resursa da preuzmemo zaštićene podatke.

Kada smo dobili pristupni token, možemo otići na [server resursa](#) da preuzmemo zaštićene korisničke podatke.

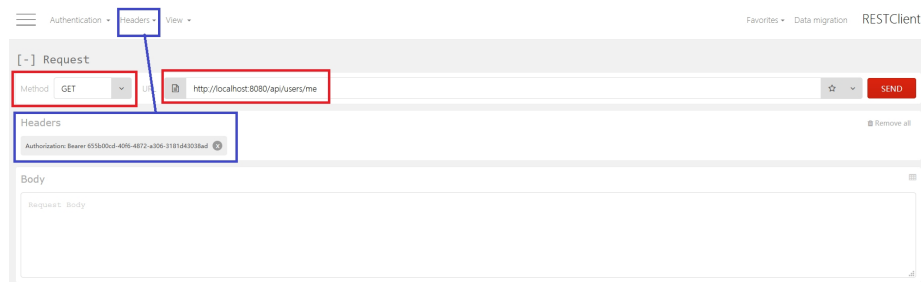
Dovoljno je izvršiti sledeći zahtev koji pod ključem [Authorization](#) uzima dobijeni pristupni token:

```
curl -X GET
-H 'Authorization: Bearer 655b00cd-40f6-4872-a306-3181d43038ad'
-i 'http://localhost:8080/api/users/me '
```

Sve je ovo moguće i putem čarobnjaka gde se kao zaglavlje definiše:

```
Authorization: Bearer 655b00cd-40f6-4872-a306-3181d43038ad
```

a kao metoda i URL, [GET](#) i <http://localhost:8080/api/users/me>, respektivno. Navedeno je pokazano sledećom slikom:



Slika 7.9 Pristup korisničkim podacima sa servera resursa - upit [izvor: autor]

Konačno, nakon obavljenog upita, server resursa odgovara zaštićenim korisničkim podacima, a to možemo da vidimo na sledećoj slici:



Slika 7.10 Odgovor servera resursa [izvor: autor]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

MAVEN ZAVISNOSTI

Za kraj diskusije, navode se zavisnosti definisane datotekom pom.xml.

Prateći primer je razvijan kao [Maven Spring Boot](#) projekat. Za kraj diskusije, navode se zavisnosti definisane datotekom pom.xml sa svim primenjenim zavisnostima:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/
xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.4.1</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.metropolitan</groupId>
  <artifactId>auth2demo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
```

```
<name>auth2demo</name>
<description>Demo project for Spring Boot</description>

<properties>
  <java.version>1.8</java.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.security.oauth.boot</groupId>
    <artifactId>spring-security-oauth2-autoconfigure</artifactId>
    <version>2.1.8.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

</project>
```

ZADATAK ZA SAMOSTALNI RAD 6

Pokušajte samostalno

Nakon ovog objekta učenja, preuzmite urađeni primer i još jednom detaljno se upustite u analizu OAUTH 2.

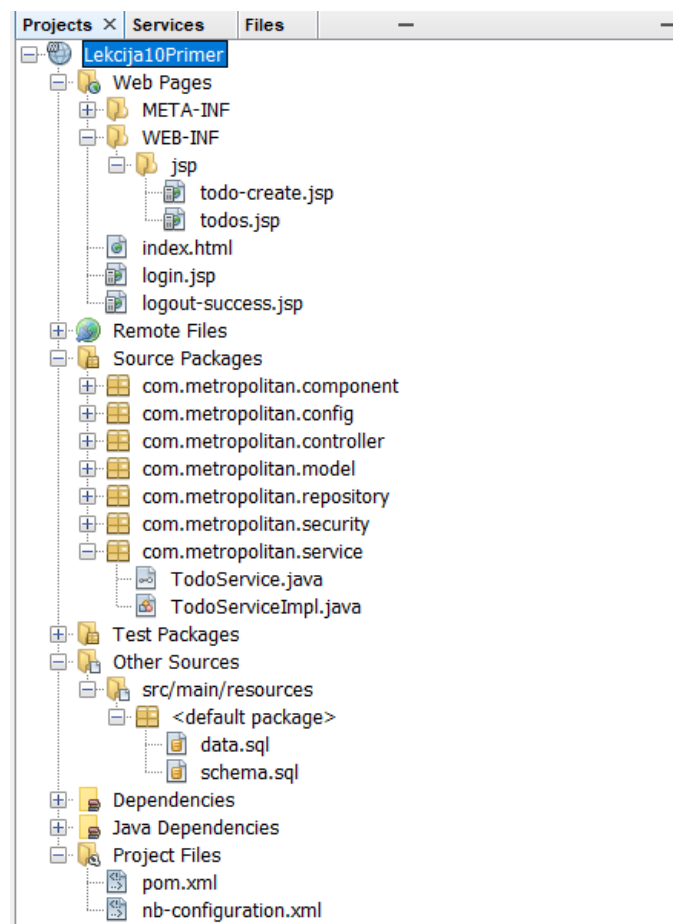
▼ Poglavlje 8

Pokazna vežba 10

STRUKTURA PROJEKTA (PREZENTACIJA 45 MINUTA)

Definisanje strukture projekta

Neophodno je objediniti segmente aplikacije, obrađivane u predavanjima, u funkcionalnu celinu. Sledećom slikom je prikazana struktura aplikacije koja se kreira.



Slika 8.1 Struktura aplikacije [izvor: autor]

POM.XML

Dodaju se zavisnosti, među njima i bezbednosni mehanizmi.

Dodaju se zavisnosti, među njima i bezbednosni mehanizmi. Sledećim listingom su prikazane sve zavisnosti, dodaci i platforme, koje će biti korišćene, objedinjene u *Maven* fajlu *pom.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.metropolitan</groupId>
  <artifactId>Lekcija10Primer</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>Lekcija10Primer</name>

  <properties>
    <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>taglibs</groupId>
      <artifactId>standard</artifactId>
      <version>1.1.2</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>5.1.0.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.1.0.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>5.1.0.RELEASE</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-web</artifactId>
      <version>5.1.0.RELEASE</version>
    </dependency>
  </dependencies>
</project>
```

```

</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>5.0.6.RELEASE</version>
  <type>jar</type>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>5.0.6.RELEASE</version>
  <type>jar</type>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>5.1.0.RELEASE</version>
  <type>jar</type>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>5.1.0.RELEASE</version>
  <type>jar</type>
</dependency>
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>2.0.1.Final</version>
  <type>jar</type>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-core</artifactId>
  <version>5.0.6.RELEASE</version>
  <type>jar</type>
</dependency>
<dependency>
  <groupId>javax</groupId>
  <artifactId>javaee-web-api</artifactId>
  <version>7.0</version>
  <scope>provided</scope>
</dependency>

<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <version>1.4.197</version>
</dependency>

</dependencies>

<build>

```

```

<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.1</version>
    <configuration>
      <source>1.8</source>
      <target>1.8</target>
      <compilerArguments>
        <endorseddirs>${endorsed.dir}</endorseddirs>
      </compilerArguments>
    </configuration>
  </plugin>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>2.3</version>
    <configuration>
      <failOnMissingWebXml>>false</failOnMissingWebXml>
    </configuration>
  </plugin>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifactId>
    <version>2.6</version>
    <executions>
      <execution>
        <phase>validate</phase>
        <goals>
          <goal>copy</goal>
        </goals>
        <configuration>
          <outputDirectory>${endorsed.dir}</outputDirectory>
          <silent>>true</silent>
          <artifactItems>
            <artifactItem>
              <groupId>javax</groupId>
              <artifactId>javaee-endorsed-api</artifactId>
              <version>7.0</version>
              <type>jar</type>
            </artifactItem>
          </artifactItems>
        </configuration>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>

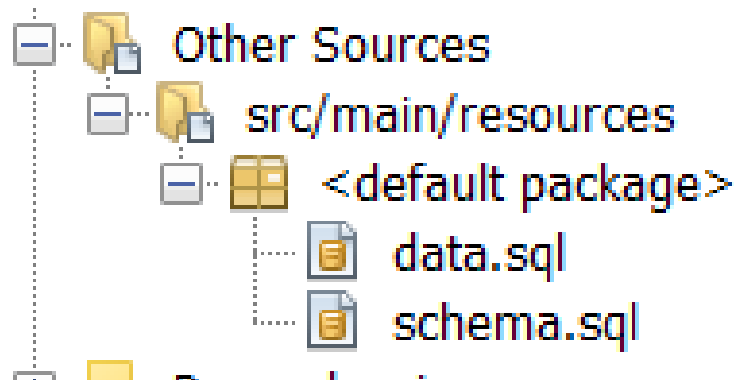
</project>

```

BAZA PODATAKA

Koristi se H2 baza podataka

U primeru se koristi *H2* baza podataka. Na lokaciji sa slike nalaze se dva SQL fajla. Pokretanjem aplikacije prvi od njih kreira tabele, a drugi popunjava kreirane tabele baze podataka.



Slika 8.2 SQL skriptovi za kreiranje baze podataka [izvor: autor]

Sledi SQL listing za kreiranje tabeleabaze podataka.

```
CREATE TABLE todo (
  id bigint AUTO_INCREMENT PRIMARY KEY ,
  owner VARCHAR(255) NOT NULL,
  description VARCHAR(255) NOT NULL,
  completed BOOLEAN NOT NULL DEFAULT false
);

CREATE TABLE USERS (
  USERNAME    VARCHAR(50)    NOT NULL,
  PASSWORD    VARCHAR(60)    NOT NULL,
  ENABLED     SMALLINT,
  PRIMARY KEY (USERNAME)
);

CREATE TABLE AUTHORITIES (
  USERNAME    VARCHAR(50)    NOT NULL,
  AUTHORITY    VARCHAR(50)    NOT NULL,
  FOREIGN KEY (USERNAME) REFERENCES USERS
);
```

Slede podaci za tabele:

```
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('vlada', 'user', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('admin', 'admin', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('student', 'student', 0);*/

INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('vlada', '$2a$04$j/
```

```
SRLDKHICK0JH.x0.GnBuRg6dsurq/GNHTieL.XxNU8ozBwnDOX.', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('admin',
'$2a$04$74kmEHHIgu8C0qYZtD38AeXc9azlmgabuPa0c9/ZS/u6CHlaRP3uK', 1);
INSERT INTO USERS (USERNAME, PASSWORD, ENABLED) VALUES ('student',
'$2a$04$h17KNRS1qW19Q9puIwpi/OV0xE/Eirvd.aYVDNz/ay.gzDhhU.3ou', 0);

INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('vlada', 'USER');
INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('student', 'USER');

INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('admin', 'USER');
INSERT INTO AUTHORITIES (USERNAME, AUTHORITY) VALUES('admin', 'ADMIN');
```

KONFIGURACIJE - WEB

Kontekst aplikacije i veb inicijalizator.

Sledi listing veb inicijalizatora.

```
package com.metropolitan.config;

import javax.servlet.Filter;

import org.springframework.web.filter.HiddenHttpMethodFilter;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletIni
tializer;

/**
 *
 * @author Vlada
 */
public class TodoWebInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class<?>[0];
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class<?>[]{TodoWebConfig.class};
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/*"};
    }
}
```



```
@Override
protected Filter[] getServletFilters() {
    return new Filter[]{
        new HiddenHttpMethodFilter()
    };
}
}
```

Glavna konfiguraciona klasa.

```
package com.metropolitan.config;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseBuilder;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseType;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

/**
 *
 * @author Vlada
 */
@Configuration
@EnableWebMvc
@ComponentScan(value = "com.metropolitan", excludeFilters = {
    @ComponentScan.Filter(Configuration.class)})
public class TodoWebConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/").setViewName("redirect:/todos");
    }

    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver viewResolver = new
InternalResourceViewResolver();
        viewResolver.setPrefix("/WEB-INF/jsp/");
        viewResolver.setSuffix(".jsp");
        return viewResolver;
    }

    /* @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder()
```

```

        .setType(EmbeddedDatabaseType.H2)
        .setName("board")
        .addScript("classpath:/schema.sql")
        .build();
    }*/
    /* @Bean
    public DataSource dataSource() {
        DriverManagerDataSource driver = new DriverManagerDataSource();
        driver.setDriverClassName("org.postgresql.Driver");
        driver.setUrl("jdbc:postgresql://localhost/lekcija10");
        driver.setUsername("postgres");
        driver.setPassword("vlada");

        return driver;
    }*/
}

```

KONFIGURACIJE - BEZBEDNOST

Inicijalizator, konfiguraciona klasa i glasač

Sledi listing inicijalizatora bezbednosnih podešavanja.

```

package com.metropolitan.security;

import
org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

/**
 *
 * @author Vlada
 */
public class TodoSecurityInitializer extends
AbstractSecurityWebApplicationInitializer {

    public TodoSecurityInitializer() {
        super(TodoSecurityConfig.class);
    }
}

```

Glavna konfiguraciona bezbednosna klasa.

```

package com.metropolitan.security;

import javax.sql.DataSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseBuilder;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseType;

```

```
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

/**
 *
 * @author Vlada
 */
@Configuration
@EnableWebSecurity
public class TodoSecurityConfig extends WebSecurityConfigurerAdapter {

    @Bean
    public DataSource dataSource() {
        return new EmbeddedDatabaseBuilder()
            .setType(EmbeddedDatabaseType.H2)
            .setName("mem:board")
            .addScript("classpath:/schema.sql")
            .addScript("classpath:/data.sql")
            .build();
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /* @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
auth.jdbcAuthentication().passwordEncoder(NoOpPasswordEncoder.getInstance()).dataSource(dataSource());
    }*/
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .jdbcAuthentication()
            .passwordEncoder(passwordEncoder())
            .dataSource(dataSource());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests()
```

```

        .antMatchers("/todos*").hasAuthority("USER")
        .antMatchers(HttpMethod.DELETE, "/todos*").hasAuthority("ADMIN")
        .and()
        .httpBasic().disable()
        .formLogin()
        .loginPage("/login.jsp")
        .loginProcessingUrl("/login")
        .failureUrl("/login.jsp?error=true")
        .permitAll()
        .and()
        .logout().logoutSuccessUrl("/logout-success.jsp");
    }
}

```

```

package com.metropolitan.security;
import java.util.Collection;
import java.util.Objects;

import org.springframework.security.access.AccessDecisionVoter;
import org.springframework.security.access.ConfigAttribute;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.WebAuthenticationDetails;

/**
 *
 * @author Vlada
 */
public class IpAddressVoter implements AccessDecisionVoter<Object> {

    private static final String IP_PREFIX = "IP_";
    private static final String IP_LOCAL_HOST = "IP_LOCAL_HOST";

    @Override
    public boolean supports(ConfigAttribute attribute) {
        return (attribute.getAttribute() != null) &&
attribute.getAttribute().startsWith(IP_PREFIX);
    }

    @Override
    public boolean supports(Class<?> clazz) {
        return true;
    }

    @Override
    public int vote(Authentication authentication, Object object,
Collection<ConfigAttribute> configList) {
        if (!(authentication.getDetails() instanceof WebAuthenticationDetails)) {
            return ACCESS_DENIED;
        }

        WebAuthenticationDetails details = (WebAuthenticationDetails)
authentication.getDetails();

```

```

        String address = details.getRemoteAddress();

        int result = ACCESS_ABSTAIN;

        for (ConfigAttribute config : configList) {
            result = ACCESS_DENIED;

            if (Objects.equals(IP_LOCAL_HOST, config.getAttribute())) {
                if (address.equals("127.0.0.1") ||
address.equals("0:0:0:0:0:0:0:1")) {
                    return ACCESS_GRANTED;
                }
            }
        }

        return result;
    }
}

```

U predavanjima je obrađen i glasač.

```

package com.metropolitan.security;
import java.util.Collection;
import java.util.Objects;

import org.springframework.security.access.AccessDecisionVoter;
import org.springframework.security.access.ConfigAttribute;
import org.springframework.security.core.Authentication;
import org.springframework.security.web.authentication.WebAuthenticationDetails;

/**
 *
 * @author Vlada
 */
public class IpAddressVoter implements AccessDecisionVoter<Object> {

    private static final String IP_PREFIX = "IP_";
    private static final String IP_LOCAL_HOST = "IP_LOCAL_HOST";

    @Override
    public boolean supports(ConfigAttribute attribute) {
        return (attribute.getAttribute() != null) &&
attribute.getAttribute().startsWith(IP_PREFIX);
    }

    @Override
    public boolean supports(Class<?> clazz) {
        return true;
    }

    @Override
    public int vote(Authentication authentication, Object object,
Collection<ConfigAttribute> configList) {

```

```

        if (!(authentication.getDetails() instanceof WebAuthenticationDetails)) {
            return ACCESS_DENIED;
        }

        WebAuthenticationDetails details = (WebAuthenticationDetails)
authentication.getDetails();
        String address = details.getRemoteAddress();

        int result = ACCESS_ABSTAIN;

        for (ConfigAttribute config : configList) {
            result = ACCESS_DENIED;

            if (Objects.equals(IP_LOCAL_HOST, config.getAttribute())) {
                if (address.equals("127.0.0.1") ||
address.equals("0:0:0:0:0:0:0:1")) {
                    return ACCESS_GRANTED;
                }
            }
        }

        return result;
    }
}

```

MODEL

Kreiranje modelskog nivoa

Početak Java razvoja, nakon podešavanja, započinje razvojem modelskog nivoa. Aplikacija sadrži jednu modelsku klasu prikazanu sledećim listingom.

```

package com.metropolitan.model;

import javax.validation.constraints.NotEmpty;
/**
 *
 * @author Vlada
 */

public class Todo {

    private Long id;

    private String owner;

    @NotEmpty
    private String description;
}

```

```
private boolean completed = false;

public Todo() {}

public Todo(String owner, String description) {
    this.owner=owner;
    this.description=description;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getOwner() {
    return owner;
}

public void setOwner(String owner) {
    this.owner = owner;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public boolean isCompleted() {
    return completed;
}

public void setCompleted(boolean completed) {
    this.completed = completed;
}

@Override
public String toString() {

    return String.format("Todo [id=%d, description=%s, owner=%s,
completed=%b]\n",this.id, this.description, this.owner, this.completed);
}
}
```

KONTROLER

Nivo upravljanja zahtevima.

Neophodno je kreirati i kontroler, koji nije bio razmatran u predavanjima, a neophodan je za mapiranje i rukovanje zahtevima.

```
package com.metropolitan.controller;
import com.metropolitan.model.Todo;
import com.metropolitan.service.TodoService;
import java.util.List;

import javax.validation.Valid;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestMapping;

/**
 *
 * @author Vlada
 */
@Controller
@RequestMapping("/todos")
public class TodoController {

    private final TodoService todoService;

    public TodoController(TodoService todoService) {
        this.todoService = todoService;
    }

    @GetMapping
    public String list(Model model) {
        List<Todo> todos = todoService.listTodos();
        model.addAttribute("todos", todos);
        return "todos";
    }

    @GetMapping("/new")
    public String create(Model model) {
        model.addAttribute("todo", new Todo());
    }
}
```



```
        return "todo-create";
    }

    @PostMapping
    public String newMessage(@ModelAttribute @Valid Todo todo, BindingResult
errors) {

        if (errors.hasErrors()) {
            return "todo-create";
        }
        String owner = "vlada@email.net";
        todo.setOwner(owner);
        todoService.save(todo);
        return "redirect:/todos";
    }

    @PutMapping("/{todoId}/completed")
    public String complete(@PathVariable("todoId") long todoId) {
        this.todoService.complete(todoId);
        return "redirect:/todos";
    }

    @DeleteMapping("/{todoId}")
    public String delete(@PathVariable("todoId") long todoId) {
        this.todoService.remove(todoId);
        return "redirect:/todos";
    }

    @InitBinder
    public void initBinder(WebDataBinder binder) {
        // We don't want to bind the id and owner fields as we control them in this
controller and service instead.
        binder.setDisallowedFields("id", "owner");
    }
}
```

SERVISNI NIVO

Kreiranje interfejsa i implementacione klase servisnog nivoa.

Sledi interfejs servisnog nivoa.

```
package com.metropolitan.service;

import com.metropolitan.model.Todo;
import java.util.List;

/**
 *
```

```
* @author Vlada
*/
public interface TodoService {

    List<Todo> listTodos();

    void save(Todo todo);

    void complete(long id);

    void remove(long id);

    Todo findById(long id);
}
```

Sledi implementaciona klasa kreiranog interfejsa.

```
package com.metropolitan.service;
import com.metropolitan.model.Todo;
import com.metropolitan.repository.TodoRepository;
import java.util.List;

import javax.transaction.Transactional;

import org.springframework.stereotype.Service;

/**
 *
 * @author Vlada
 */
@Service
@Transactional
class TodoServiceImpl implements TodoService {

    private final TodoRepository todoRepository;

    TodoServiceImpl(TodoRepository todoRepository) {
        this.todoRepository = todoRepository;
    }

    @Override
    public List<Todo> listTodos() {
        return todoRepository.findAll();
    }

    @Override
    public void save(Todo todo) {
        this.todoRepository.save(todo);
    }

    @Override
    public void complete(long id) {
        Todo todo = findById(id);
    }
}
```

```
        todo.setCompleted(true);
        todoRepository.save(todo);
    }

    @Override
    public void remove(long id) {
        todoRepository.remove(id);
    }

    @Override
    public Todo findById(long id) {
        return todoRepository.findOne(id);
    }
}
```

REPOZITORIJUM

Kreiranje interfejsa i implementacione klase nivoa repozitorijuma.

Sledi interfejs nivoa repozitorijuma.

```
package com.metropolitan.repository;
import com.metropolitan.model.TODO;
import java.util.List;
/**
 *
 * @author Vlada
 */

public interface TodoRepository {

    List<TODO> findAll();

    TODO findOne(long id);

    void remove(long id);

    TODO save(TODO todo);

    List<TODO> findByOwner(String author);

}
```

Sledi implementaciona klasa kreiranog interfejsa.

```
package com.metropolitan.repository;
import com.metropolitan.model.TODO;
import java.sql.PreparedStatement;
import java.util.List;
```

```
import javax.sql.DataSource;

import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.stereotype.Repository;

/**
 *
 * @author Vlada
 */
@Repository
class JdbcTodoRepository implements TodoRepository {

    private final JdbcTemplate jdbc;

    JdbcTodoRepository(DataSource dataSource) {
        this.jdbc=new JdbcTemplate(dataSource);
    }

    @Override
    public List<Todo> findAll() {
        return this.jdbc.query("select * from todo order by id",
        BeanPropertyRowMapper.newInstance(Todo.class));
    }

    @Override
    public List<Todo> findByOwner(String owner) {
        return this.jdbc.query("select * from todo where owner=? order by id",
        BeanPropertyRowMapper.newInstance(Todo.class), owner);
    }

    @Override
    public Todo findOne(long id) {
        return this.jdbc.queryForObject("select * from todo where id=?",
        BeanPropertyRowMapper.newInstance(Todo.class), id);
    }

    @Override
    public Todo save(Todo todo) {

        if (todo.getId() == null) {
            final String sql = "insert into todo (owner, description, completed)
values (?, ?, ?)";
            final GeneratedKeyHolder keyHolder = new GeneratedKeyHolder();

            this.jdbc.update(con -> {
                PreparedStatement ps = con.prepareStatement(sql, new String[] {
                "id" });

                ps.setString(1, todo.getOwner());
                ps.setString(2, todo.getDescription());
                ps.setBoolean(3, todo.isCompleted());
```

```

        return ps;
    }, keyHolder);

    todo.setId(keyHolder.getKey().longValue());
} else {
    final String sql = "update todo set owner=?, description=?, completed=?
where id=?";
    this.jdbc.update(sql, todo.getOwner(), todo.getDescription(),
todo.isCompleted(), todo.getId());
}
return todo;
}

@Override
public void remove(long id) {
    this.jdbc.update("delete from todo where id=?", id);
}
}

```

TABLA SA PORUKAMA

Tabla sa porukama je rešena kao komponenta.

Dodavanje poruka sa zadacima na tablu sa porukama. Klasa je obeležena sa *@Component*.

```

package com.metropolitan.component;
import com.metropolitan.model.TODO;
import com.metropolitan.service.TODOService;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

/**
 *
 * @author Vlada
 */
@Component
class TODOInitializer {

    private final TODOService messageBoardService;

    TODOInitializer(TODOService messageBoardService) {
        this.messageBoardService = messageBoardService;
    }

    @PostConstruct
    public void setup() {

        TODO todo = new TODO();
        todo.setOwner("vlada@email.net");
    }
}

```

```

        todo.setDescription("IT355 - Lekcija Spring Security");

        messageBoardService.save(todo);

        todo = new Todo();
        todo.setOwner("vlada@email.net");
        todo.setDescription("Dobro procitaj lekciju!!!");
        todo.setCompleted(true);
        messageBoardService.save(todo);

        todo = new Todo();
        todo.setOwner("vlada@email.net");
        todo.setDescription("Pitaj ako ne znas!!!");

        messageBoardService.save(todo);

    }
}

```

JSP - PRIJAVA I ODJAVA KORISNIKA

Stranice za prijavu i odjavu korisnika.

Sledi listing stranice za prijavu korisnika.

```

<%- -
    Document    : login
    Created on   : 02.11.2018., 10.55.27
    Author      : Vlada
- -%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
    <head>
        <title>Login</title>
        <link type="text/css" rel="stylesheet"
            href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.10/
semantic.min.css">
        <style type="text/css">
            body {
                background-color: #DADADA;
            }
            body > .grid {
                height: 100%;
            }
            .column {
                max-width: 450px;
            }
        </style>
    </head>
    <body>
        <div class="ui grid">
            <div class="column">
                <div>
                    <h1>Login</h1>
                </div>
            </div>
        </div>
    </body>
</html>

```

```

    }
  </style>
</head>

<body>
  <div class="ui middle aligned center aligned grid">
    <div class="column">
      <h2 class="ui header">Log-in to your account</h2>
      <form method="POST" action="<c:url value="/login" />" class="ui
large form">
        <input type="hidden" name="${_csrf.parameterName}"
value="${_csrf.token}"/>
        <div class="ui stacked segment">
          <div class="field">
            <div class="ui left icon input">
              <i class="user icon"></i>
              <input type="text" name="username"
placeholder="E-mail address">
            </div>
          </div>
          <div class="field">
            <div class="ui left icon input">
              <i class="lock icon"></i>
              <input type="password" name="password"
placeholder="Password">
            </div>
          </div>
          <div class="field">
            <div class="ui toggle checkbox">
              <input type="checkbox" name="remember-me">
              <label>Remember Me</label>
            </div>
          </div>
          <button class="ui fluid large submit green
button">Login</button>
        </div>
        <c:if test="${not empty param.error}">
          <div class="ui error message" style="display: block;">
            Authentication Failed<br/>
            Reason :
            ${sessionScope["SPRING_SECURITY_LAST_EXCEPTION"].message}
          </font>
        </div>
      </c:if>
    </form>
  </div>
</div>
</body>
</html>

```

Sledi listing stranice za odjavu korisnika.

```
<%- -
    Document    : login-success
    Created on   : 02.11.2018., 10.56.44
    Author       : Vlada
- -%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
    <head>
        <title>Login</title>
        <link type="text/css" rel="stylesheet"
            href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.10/
semantic.min.css">
        <style type="text/css">
            body {
                background-color: #DADADA;
            }
            body > .grid {
                height: 100%;
            }
            .column {
                max-width: 450px;
            }
        </style>
    </head>

    <body>
        <div class="ui middle aligned center aligned grid">
            <div class="column">
                <h2 class="ui header">You have been successfully logged out</h2>
                <div>Goto <a href="<c:url value="/todos"/>" class="link">todo
list</a>.</div>
            </div>
        </div>
    </body>
</html>
```

JSP - TODOS.JSP I TODO-CREATE.JSP

Kraj definicije aplikacije kroz kreiranje pogleda.

Građenje aplikacije se završava sa dva konkretna pogleda. Sledi glavni pogled *todos.jsp*.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
```



```

<head>
  <title>To-do List</title>
  <link type="text/css" rel="stylesheet"
    href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.10/
semantic.min.css">
</head>

<body>
  <div class="ui container">
    <h4>To-dos for ${principal.name}</h4>
    <table class="ui celled table">
      <thead>
        <tr>
          <th class="three wide">Owner</th>
          <th class="five wide">To-do</th>
          <th class="two wide">Completed</th>
          <th class="six wide">&nbsp;</th>
        </tr>
      </thead>
      <tbody>
        <c:forEach items="${todos}" var="todo">
          <tr>
            <td>${todo.owner}</td>
            <td>${todo.description}</td>
            <td>${todo.completed}</td>
            <td>
              <c:if test="${!todo.completed}">
                <c:url value="/todos/${todo.id}/completed"
var="completedUrl"/>
                <form action="${completedUrl}" method="post"
style="float: left;">
                  <input type="hidden"
name="${_csrf.parameterName}" value="${_csrf.token}"/>
                  <input type="hidden" name="_method"
value="PUT"/>
                  <button class="ui mini green icon
button"><i class="check circle icon"></i></button>

                  </form>
                </c:if>
                <c:url value="/todos/${todo.id}" var="deleteUrl"/>
                <form action="${deleteUrl}" method="post"
style="float: left;">
                  <input type="hidden"
name="${_csrf.parameterName}" value="${_csrf.token}"/>
                  <input type="hidden" name="_method"
value="DELETE"/>
                  <button class="ui mini red icon button"><i
class="remove circle icon"></i></button>

                  </form>
                </td>
              </tr>
            </tbody>
          </table>
        </div>
      </body>
    </html>
  </div>

```

```

        </c:forEach>
        <tr>
            <td colspan="3">
                <a class="ui mini icon button" href="<c:url
value="/todos/new"/>">New To-do <i
                        class="add circle icon"></i></a>
            </td>
        </tr>
    </tbody>
</table>
<form action="<c:url value="/logout"/>" method="post">
    <input type="hidden" name="{_csrf.parameterName}"
value="{_csrf.token}"/>
    <button class="ui small red basic compact
button">Logout</button>
</form>

</div>
</body>
</html>

```

Sledi pogled *todo-create.jsp*.

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<html>
    <head>
        <title>Message Post</title>
        <link type="text/css" rel="stylesheet"
            href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.10/
semantic.min.css">
    </head>

    <body>
        <div class="ui container">
            <h4>New To-do</h4>
            <c:url value="/todos" var="uri"/>
            <form:form method="POST" modelAttribute="todo" action="{uri}"
class="ui form">
                <fieldset>
                    <legend>To-do</legend>
                    <div class="field">
                        <label>To-do</label>
                        <form:input path="description"/>
                    </div>
                    <div class="field">
                        <label>Completed</label>
                        <form:checkbox path="completed" />
                    </div>
                    <button class="ui mini primary button">Save <i class="send
icon"></i></button>

```

```
        </fieldset>
      </form:form>
    </div>
  </body>
</html>
```

▼ Poglavlje 9

Individualna vežba 10

INDIVIDUALNA VEŽBA (90 MINUTA)

Samostalna primena Spring Security okvira

Napraviti Spring veb aplikaciju koja ima sledeće entitete:

- Student,
- Profesor,
- Ispit i
- Predmet.

Svaki student i profesor imaju listu predmeta i ispita, i ovi podaci se nalaze u bazi podataka. Aplikacija treba da bude *CRUD* aplikacija sa *MySQL* ili *Postgresql* bazom podataka.

Studenti i profesori imaju role, profesor korisničku i admin rolu, dok student ima samo korisničku rolu.

Dodati *Spring Security* podešavanja tako da korisnici mogu da vide samo svoje predmete, dok profesori mogu da vide listu svih studenata, predmeta, svoje predmete i listu svih ispita.

Kreirati sve potrebne kontrolere i jsp fajlove.

▼ Poglavlje 10

Domaći zadatak 10

DOMAĆI ZADATAK BROJ 10 (120 MIN)

Cilj ove sekcije je da prikaže studentu šta treba da uradi za domaći zadatak 10

Na Spring projekat iz prethodnog domaćeg zadatka dodati:

1. Prijavljivanje na aplikaciju i odjavljivanje sa aplikacije,
2. Autentifikaciju korisnika iz baze podataka koristeći *Spring Security po* uzoru na primer obrađen na času predavanja.
3. Sva *Spring Security* podešavanja i korekcije na projektu je neophodno detaljno dokumentovati.
4. Spring projekat treba izraditi koristeći materijal sa predavanja i vežbi i potrebno je poslati mejl asistentu sa zipovanim Spring projektom.
5. **Student može da dobije dodatne poene za zalaganje implementacijom OAuth 2 mehanizma zaštite.**

Nakon obaveznog zadatka, studenti dobijaju na email različite zadatke od predmetnog asistenta.

▼ Poglavlje 11

Zaključak

ZAKLJUČAK

Oblast bavljenja Lekcije 10 je okvir Spring Security.

U ovoj lekciji će je pokazano kako je moguće učiniti aplikacije bezbednijim primenom *Spring Security* okvira koji je potprojekat Spring okvira. Koristeći ovaj okvir moguće je osigurati bilo koju JAVA aplikaciju ali je njegov fokus uglavnom na veb baziranim aplikacijama. Ove aplikacije, pogotovo one kojima je moguće pristupiti putem *Interneta*, veoma su ranjive na zlonamerne napade i zahtevaju veoma veliko angažovanje bezbednosnih tehnika prilikom njihovog kreiranja i održavanja.

Tokom izlaganja, u ovoj lekciji, diskutovano je o nekim veoma značajnim terminima iz domena bezbednosti aplikacija, podržanih Springom, a koje studenti moraju da savladaju:

- *Authentication;*
- *Principal;*
- *Authorization;*
- **Access control .**

Savladavanjem ove lekcije student je sada osposobljen da koristi osnovne bezbednosne koncepte u *Spring (Boot)* veb aplikacijama na:

- URL nivou pristupa;
- nivou poziva metoda;
- nivou prikazivanja pogleda;
- nivou objekata domena.

LITERATURA

Za pripremu Lekcije 10 upotrebljena je najnovija literatura.

Za pripremu lekcije korišćena je najnovija pisana i elektronska literatura:

1. Marten Dainum, Josh Long, and Daniel Rubio, Spring 5 Recipes Fourth Edition, Apress
2. Spring Framework Reference Documentation - <https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/>
3. Craig Walls, Spring in Action, Manning
4. Craig Walls, Spring Boot in Action, Manning

Dopunska literatura:

1. <http://www.javacodegeeks.com/tutorials/java-tutorials/enterprise-java-tutorials/spring-tutorials/>
2. <http://www.tutorialspoint.com/spring/>
3. <http://www.javatpoint.com/spring-tutorial>