

spring-security-examples (0.0.1)

Maksim Kostromin

Version 0.0.1, 2018-07-23 00:53:03 UTC

Table of Contents

| | |
|--|----|
| 1. CSRF Protection with Single Page Apps using JS..... | 2 |
| 2. Keycloak and Spring Boot..... | 5 |
| 3. Spring 5 Security OAuth2 (Github / Facebook) | 6 |
| 3.1. spring-5-security-oauth2..... | 6 |
| 4. Others | 7 |
| 4.1. Web MVC: testing with mock user | 7 |
| 4.2. Web MVC: testing with web driver | 10 |
| 5. links..... | 18 |
| 6. Enjoy! :)..... | 19 |

Introduction

This documentation contains some help to [examples from spring-security-examples repository](#). It's contains some spring-security playground projects

Chapter 1. CSRF Protection with Single Page Apps using JS

user / password can't do post admin / admin can

security configuration

```
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

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

        auth
            .inMemoryAuthentication()
            .withUser("user")
                .password("password")
                .roles("USER")
            .and()
            .withUser("admin")
                .password("admin")
                .roles("ADMIN");
    }

    @Override
    public void configure(final WebSecurity web) throws Exception {

        web.ignoring()
            .antMatchers(
                "/favicon.ico",
                "/webjars/**",
                "/login.html",
                "/index.html",
                "/logout.html"
            );
    }

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

        http
            .authorizeRequests()
                .antMatchers(POST)
                .hasRole("ADMIN")
            .anyRequest()
                .authenticated()
            .and()
            .formLogin()
    }
}
```

```

        .defaultSuccessUrl("/", true)
        .permitAll()
        .and()
    .logout()
        .logoutUrl("/logout")
        .logoutSuccessUrl("/")
        .clearAuthentication(true)
        .deleteCookies("JSESSIONID")
        .invalidateHttpSession(false)
        .permitAll()
        .and()
    .headers()
        .frameOptions()
        .sameOrigin()
        .and()
    .csrf()
        .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
        .and()
    .sessionManagement()
        .sessionCreationPolicy(SessionCreationPolicy.NEVER)
;
}
}

```

manual logout endpoint

```

@GetMapping("/logout")
public String logoutGet(final HttpServletRequest request, final HttpServletResponse
response) {
    return logout(request, response);
}

@PostMapping("/logout")
public String logoutPost(final HttpServletRequest request, final HttpServletResponse
response) {
    return logout(request, response);
}

private String logout(final HttpServletRequest request, final HttpServletResponse
response) {

    Optional.ofNullable(SecurityContextHolder.getContext())
        .map(SecurityContext::getAuthentication)
        .ifPresent(authentication -> new SecurityContextLogoutHandler().logout
(request, response, authentication));

    return "redirect:/login";
}

```

set header from client cookie on javascript single page app

```
function getCookie(cookiePrefix) {
  var name = cookiePrefix + "=";
  var decodedCookie = decodeURIComponent(document.cookie);
  var ca = decodedCookie.split(';');
  for(var i = 0; i < ca.length; i++) {
    var c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}

var headers = {
  'X-XSRF-TOKEN': getCookie('XSRF-TOKEN'),
  'content-type': 'application/json',
};

var options = {
  method: 'post',
  headers: headers,
  credentials: 'include',
  body: { ololo: 'trololo ' }
};

fetch("/user", options)
  .then(data => data.json())
  .then(json => render(JSON.stringify(json)));
```

links:

1. [youtube talk](#)
2. [some demo](#)

Chapter 2. Keycloak and Spring Boot

TODO: in progress....

links:

1. [Keycloak project](#)
2. [Devoxx talk](#)

Chapter 3. Spring 5 Security OAuth2 (Github / Facebook)

3.1. spring-5-security-oauth2

1. spring-framework 5
2. spring-boot 2
3. oauth2
4. github
5. facebook
6. facebook + github together

build

```
bash ./gradlew
bash spring-mvc-facebook-github/build/libs/*.jar
bash spring-mvc-facebook/build/libs/*.jar
bash spring-mvc-github/build/libs/*.jar

http :8080
http :8080/login
```

TODO:

1. authorization callback URL: <http://localhost:8080/login/oauth2/code/github> (github is registration id from applicatin.yaml)
2. okta
3. google

links:

1. [Next Generation OAuth Support with Spring Security 5.0 - Joe Grandja](#)
2. [Github: jgrandja/springone2017-demo](#)
3. [Spring Boot and OAuth2](#)

generated by [daggerok-fatjar](#) yeoman generator

Chapter 4. Others

4.1. Web MVC: testing with mock user

```

@Log4j2
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //@formatter:off
        http
            .authorizeRequests()
            .antMatchers("/login", "/webjars", "/favicon.*")
            .permitAll()
            .anyRequest()
            .fullyAuthenticated()
            .and()
            .formLogin()
            .defaultSuccessUrl("/")
            .failureUrl("/login?error")
            .failureForwardUrl("/login?failure")
            .and()
            .logout()
            .clearAuthentication(true)
            .invalidateHttpSession(true)
            .deleteCookies("JSESSIONID", "PLAY_SESSION", "NXSESSIONID", "csrfToken",
"SESSION")
            .permitAll(true)
        ;
        //@formatter:on
    }

    @Override
    @Autowired
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        //@formatter:off
        HashMap.of("usr", "pwd")
            .forEach((username, password) -> Try.run(() -> auth
                .inMemoryAuthentication()
                .withUser(username)
                .password(passwordEncoder().encode(password))
                .roles("APP", "APP_USER", "APPLICATION_USER")));
        //@formatter:on
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }
}

```

index.html page

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Index Page</title>
  <link rel="shortcut icon" th:href="@{/favicon.ico}" type="image/x-icon">
</head>
<body>
<h1>Hola!</h1>
</body>
</html>
```

testing unauthorized access: application must redirect user to login page

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = RANDOM_PORT)
public class MockMvcSecurityTests {

    @Autowired
    WebApplicationContext wac;

    private MockMvc mvc;

    @Before
    public void setup() {
        this.mvc = MockMvcBuilders.webAppContextSetup(wac)
            .apply(springSecurity())
            .build();
    }

    @Test
    @SneakyThrows
    public void unauthorized_request_should_be_redirected_to_login_page() {
        mvc.perform(get("/"))
            .andExpect(status().isFound())
            .andExpect(header().string("location", containsString("/login")))
        ;
    }
}
```

testing login page: must be publicly accessible for non-authorized users

```
@Test
@sneakyThrows
public void login_page_is_publicly_accessible() {
    mvc.perform(get("/login"))
        .andExpect(status().isOk());
}
```

testing authorized request

```
@Test
@sneakyThrows
@WithMockUser
public void authorized_request_test() {
    mvc.perform(get("/"))
        .andExpect(status().isOk())
        .andExpect(content().contentType(parseMediaType("text/html; charset=UTF-8")))
        .andExpect(content().string(containsString("<title>Index Page</title>")));
}
```

4.2. Web MVC: testing with web driver

```

@Log4j2
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //@formatter:off
        http
            .authorizeRequests()
            .antMatchers("/login", "/webjars", "/favicon.*")
            .permitAll()
            .anyRequest()
            .fullyAuthenticated()
            .and()
            .formLogin()
            .defaultSuccessUrl("/")
            .failureUrl("/login?error")
            .failureForwardUrl("/login?failure")
            .and()
            .logout()
            .clearAuthentication(true)
            .invalidateHttpSession(true)
            .deleteCookies("JSESSIONID", "PLAY_SESSION", "NXSESSIONID", "csrfToken",
"SESSION")
            .permitAll(true)
        ;
        //@formatter:on
    }

    @Override
    @Autowired
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        //@formatter:off
        HashMap.of("usr", "pwd")
            .forEach((username, password) -> Try.run(() -> auth
                .inMemoryAuthentication()
                .withUser(username)
                .password(passwordEncoder().encode(password))
                .roles("APP", "APP_USER", "APPLICATION_USER")));
        //@formatter:on
    }

    @Bean
    PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }
}

```

index.html page

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Index Page</title>
  <link rel="shortcut icon" th:href="@{/favicon.ico}" type="image/x-icon">
</head>
<body>
  <h1>Hola!</h1>
</body>
</html>
```

4.2.1. HtmlUnit

testing login page: must be publicly accessible for non-authorized users

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = RANDOM_PORT)
public class HtmlUnitWebDriverSecurityTests {

    @Autowired
    Environment env;

    private HtmlUnitDriver driver;

    @Before
    public void setUp() throws Exception {
        this.driver = new LocalHostWebConnectionHtmlUnitDriver(env);
    }

    @Test
    @SneakyThrows
    public void login_page_is_publicly_accessible() {
        driver.get("/");
        assertThat(driver.getTitle()).contains("Login Page");
    }
}
```

testing login

```
@Test
@sneakyThrows
public void login_test() {

    driver.get("/");

    final WebElement form = driver.findElementByTagName("form");

    form.findElement(By.cssSelector("input[name=username]"))
        .sendKeys("usr");
    form.findElement(By.cssSelector("input[name=password]"))
        .sendKeys("pwd");
    form.submit();

    assertThat(driver.getTitle()).contains("Index Page");
}
}
```

4.2.2. Chrome using WebDriver

testing login page: must be publicly accessible for non-authorized users

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = RANDOM_PORT)
public class ChromeWebDriverSecurityTests {

    @LocalServerPort
    int port;

    private ChromeDriver driver;
    private String baseUrl;

    @Before
    public void setUp() throws Exception {
        final boolean headless = false;
        System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
        this.driver = new ChromeDriver(new ChromeOptions().setHeadless(headless));
        this.baseUrl = format("http://127.0.0.1:%d", port);
    }

    public void open(final String uri) {
        final boolean isValidUri = null != uri && uri.startsWith("/");
        final String path = isValidUri ? uri : "/" + uri;
        driver.get(baseUrl + path);
    }

    @Test
    @SneakyThrows
    public void login_page_is_publicly_accessible() {
        open("/");
        assertThat(driver.getTitle()).contains("Login Page");
    }
}
```


testing login

```
@Test
@sneakyThrows
public void login_test() {

    open("/");
    assertThat(driver.getTitle()).contains("Login Page");

    final WebElement form = driver.findElementByTagName("form");

    form.findElement(By.cssSelector("input[name=username]"))
        .sendKeys("usr");
    form.findElement(By.cssSelector("input[name=password]"))
        .sendKeys("pwd");
    form.submit();

    assertThat(driver.getTitle()).contains("Index Page");
}
```

4.2.3. Chrome using Selenide

testing login page: must be publicly accessible for non-authorized users

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = RANDOM_PORT)
public class ChromeWebDriverSecurityTests {

    @LocalServerPort
    int port;

    private ChromeDriver driver;
    private String baseUrl;

    @Before
    public void setUp() throws Exception {
        final boolean headless = false;
        System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
        this.driver = new ChromeDriver(new ChromeOptions().setHeadless(headless));
        this.baseUrl = format("http://127.0.0.1:%d", port);
    }

    public void open(final String uri) {
        final boolean isValidUri = null != uri && uri.startsWith("/");
        final String path = isValidUri ? uri : "/" + uri;
        driver.get(baseUrl + path);
    }

    @Test
    @SneakyThrows
    public void login_page_is_publicly_accessible() {
        open("/");
        assertThat(driver.getTitle()).contains("Login Page");
    }
}
```

```
@Test
@sneakyThrows
public void login_test() {

    open("/");
    assertThat(driver.getTitle()).contains("Login Page");

    final WebElement form = driver.findElementByTagName("form");

    form.findElement(By.cssSelector("input[name=username]"))
        .sendKeys("usr");
    form.findElement(By.cssSelector("input[name=password]"))
        .sendKeys("pwd");
    form.submit();

    assertThat(driver.getTitle()).contains("Index Page");
}
```

Chapter 5. links

1. [Asciidoctor attributes](#)

Chapter 6. Enjoy! :)