Laborator 10: Introducere Spring security

1. Importați proiectul LAB10_START. Creați tabelul sales.customer. Creați o configurație pentru a rula aplicația pe serverul Tomacat:

```
DROP TABLE IF EXISTS sales.customer;
DROP TABLE IF EXISTS sales.users;
DROP TABLE IF EXISTS sales.roles;

CREATE TABLE sales.customer (
       id int8(10) NOT NULL AUTO_INCREMENT,
       first_name varchar(40) DEFAULT NULL,
       last_name varchar(40) DEFAULT NULL,
       email varchar(40) DEFAULT NULL, PRIMARY KEY (id) )
ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Endpoints:

```
Index Controller
(TO DO accesibil pentru toti utilizatorii)
http://localhost:8080/laborator10/
http://localhost:8080/laborator10/showForm

Hello Controller
(TO DO accesibil pentru utilizatori cu rol HELLO)
http://localhost:8080/laborator10/hello/hello/{name}
http://localhost:8080/laborator10/hello/showForm

Customer Controller
(TO DO accesibil pentru utilizatori cu rol CUSTOMER)
http://localhost:8080/laborator10/customer/list
http://localhost:8080/laborator10/customer/showForm
```

MAVEN DEPENDECIES

spring-security-web

```
<dependency>
	<groupId>org.springframework.security</groupId>
	<artifactId>spring-security-web</artifactId>
	<version>${springsecurity.version}</version>
</dependency>
```

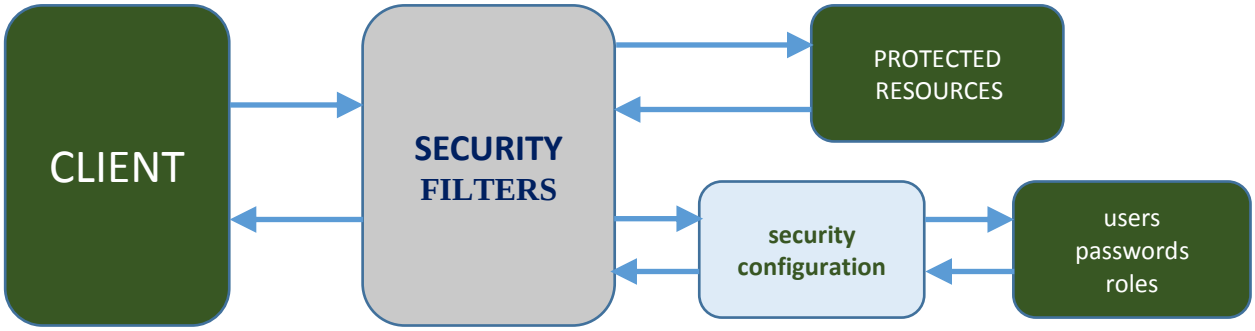
spring-security-config

```
<dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-config</artifactId>
        <version>${springsecurity.version}</version>
</dependency>
```

Versiunile pentru **Spring Security** nu sunt sincronizate cu versiunile pentru **Spring Framework** (core, aop, mvc etc.).
Verificați compatibilitatea cu spring-webmvc:
https://mvnrepository.com/artifact/org.springframework.security/spring-security-web/5.1.2.RELEASE

```
<springsecurity.version>5.1.2.RELEASE</springsecurity.version>
```

2. Adăugați în fișierul .pom dependențele pentru spring-security.



AbstractSecurityWebApplicationInitializer:
Inițializează și activează springSecurityFilterChain.

@EnableWebSecurity
WebSecurityConfigurerAdapter
Oferă metode pentru a configura infomrații despre utilizatori și drepturi, despre pagina de login și logout etc.

3. Creați în pachetul config clasa SecurityWebApplicationInitializer:

```
public class SecurityWebApplicationInitializer
        extends AbstractSecurityWebApplicationInitializer {
}
```

protected void configure(AuthenticationManagerBuilder auth) throws Exception
este metoda utilizată pentru configurarea utilizatorilor (in memory, database, ldap etc.)

4. Creați în pachetul config clasa SecurityConfig:

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

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

        @Override
        protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
auth.inMemoryAuthentication()
.withUser("userCustomer").password(passwordEncoder().encode("1234")).roles(
"CUSTOMER")
.and().withUser("userHello").password(passwordEncoder().encode("1234")).rol
es("HELLO");
}
```

protected void configure(HttpSecurity http) throws Exception
este metoda utilizată pentru configurarea formularelor de log-in și a path-urilor autorizate.
http.**authorizeRequests**() specifică requesturile care necesită autorizare

http.**formLogin()**
HttpSecurity.formLogin() întoarce un obiect de tipul **FormLoginConfigurer** utilizat pentru configurarea formularului de login.

.formLogin()
- .loginPage("/showLogInForm") indică requestul pentru pagina de autentificare
- .loginProcessingUrl("/authUser") specifică requestul pentru procesarea formularului.
- .failureUrl("/showErrorLogIn") indică requestul pentru pagina de eroare la autentificare

Formularul de login trebuie să includă un **form** care să cuprindă câmpurile **username** și **password** iar **action** trebuie să indice requestul specificat cu metoda loginProcessingUrl.

5. Suprascrieți metoda configure(HttpSecurity httpSecurity).

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http.authorizeRequests().anyRequest().authenticated()
                .and()
                .formLogin()
                    .loginPage("/showLogInForm")
                    .loginProcessingUrl("/authUser")
                    .permitAll();
    }
```

6. Creați controllerul pentru requestul /showLogInForm:

```
@GetMapping("/showLogInForm")
public String showLogInForm(){
    return "login";
}
```

Tratarea erorilor

Login page:

În caz de eroare se retrimite formularul de login împrună parametrul **error**.
Se poate verifica în view (jsp, tymeleaf) dacă a fost trimis parametrul error:

Themleaf: Wrong user or password
Jsp: <j:if test=”${param.error != null}”> Wrong user or password</j:if>

7. Includeți în view-ul login.html un mesaj care se va afișa în caz de erroare la login:

```
<p th:if="${param.error}" class="text-danger">Wrong user or
password</p>
```

Failure URL:

Se poate specifica utilizând metoda **failureUrl** clasei FormLoginConfigurer un url custom pentru afișarea unei pagini de eroare.

8. Afișați în pagina de login un mesaj de eroare folosind metoda failureUrl.

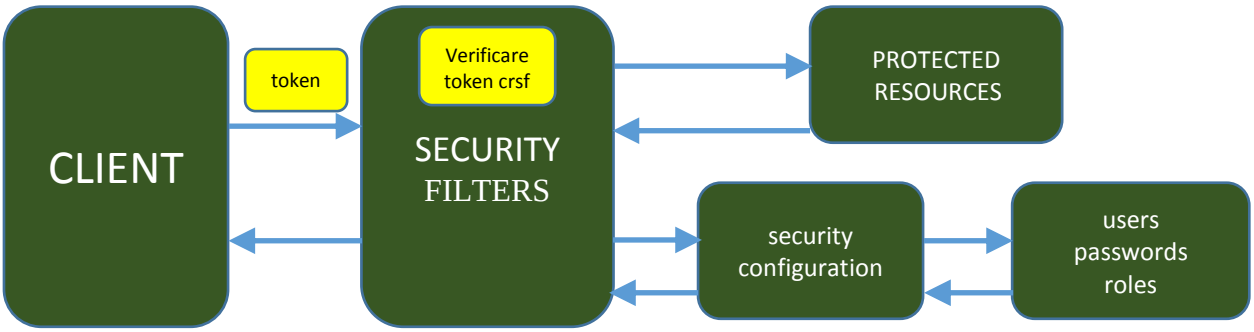
```
//protected void configure(HttpSecurity http)
.failureUrl("/showErrorLogIn")

//IndexXontroller
@GetMapping("/showErrorLogIn")
public String showErrorLogIn(Model model){
    model.addAttribute("errorMessage", "try again ... ");
    return "login";
}

//login.html
<p th:if="${errorMessage}" class="text-danger"
th:text="${errorMessage}">Wrong user or password</p>
```

Cross-Site Request Forgery

Cross-Site Request Forgery (see surf): Executarea unei acțiuni nedorite într-o aplicație în care utilizatorul este logat.

Synchronizer Token Pattern
Presupune adăugarea unui token suplimentar în view-uri. La executarea unui request se vor verifica sessionId-ul și token-ul.
Spring Security generează automat un token **crsf** și îl include în fiecare **form.** La executarea fiecărui request se testează dacă tokenul pentru sesiunea curentă este valid.

Pentru mai multe detalii puteți consulta:
https://docs.spring.io/spring-security/site/docs/5.0.11.RELEASE/reference/htmlsingle/#csrf



USER Info

Presupune afișarea informațiilor despre utilizator se pot folosi spring-security-taglibs (JSP) sau Spring Security Integration Module (Tymeleaf)

JSP:

```
<dependency>
	<groupId>org.springframework.security</groupId>
	<artifactId>spring-security-taglibs</artifactId>
	<version>${springsecurity.version}</version>
</dependency>
```

Tymeleaf

```
<dependency>
	<groupId>org.thymeleaf.extras</groupId>
	<artifactId>thymeleaf-extras-springsecurity5</artifactId>
	<version>3.0.4.RELEASE</version>
</dependency>
```

9. Adăugați dependențele pentru Security Integration Module și configurați templateEngine:

```
// Lab10Config
@Bean(name = "additionalDialects")
    public SpringSecurityDialect additionalDialects(){
        return new SpringSecurityDialect();
}

@Bean(name = "templateEngine")
public SpringTemplateEngine templateEngine(ServletContext
servletContext){
          …
        springTemplateEngine.addDialect(additionalDialects());
        return springTemplateEngine;
}
```

10. Modificaț home.html astfel încât să se afișeze utilizatorul și rolurile acestuia.

```
Logged user: <span sec:authentication="name">user</span>
Roles: <span
sec:authentication="principal.authorities">[ROLE_USER]</span>
```

11. Adaugați userului userCustomer rolul HELLO.

Default logout:

Clasa HttpSecurity oferă suport pentru un URL pentru logout /logout default prin utilizarea metodei logout(). După logout automat este afișată pagina de login la care este trimis parametrul logout.

12. Adăugați în home.html un buton pentru logout.

```
<form th:action="@{/logout}" method="POST">
<input type="submit" value="Logout">
</form>
```

13. Adăugați în pagina de login un mesaj care se va afișa daca este prezent în request parametrul logout.

Restricted Aces

http.authorizeRequests() întoarce un obiect de tipul **ExpresionInterceptUrlRegistry** care permite configurarea requesturile care necesită autorizare

14. Restricționați accesul userilor care nu au rol CUSTOMER.

```
http.authorizeRequests()
       .antMatchers("/").hasRole("HELLO")
       .antMatchers("/customer/**").hasRole("CUSTOMER")
```

15. Adăugați o pagină custom care se va afișa în cazul în care accesul este restricționat pentru un anumit request. Testați.

```
http.authorizeRequests
                .antMatchers("/").hasRole("HELLO")
                .antMatchers("/customer/**").hasRole("CUSTOMER")
                .and()
                .formLogin()
                    .loginPage("/showLogInForm")
                    .failureUrl("/showErrorLogIn")
                    .loginProcessingUrl("/authUser")
                    .permitAll().
                and().exceptionHandling()
                    .accessDeniedPage("/accesDenied");
```

```
CREATE TABLE sales.users (
       username varchar(50),
       password char(60),
       enabled tinyint(1),
       PRIMARY KEY(username)
)

CREATE TABLE sales.authorities(
       username  varchar(50),
       authority varchar(50)
)
```

16. Adăugați în SecurityConfig o referință către bean-ul de tip DataSource:

```
@Autowired
private DataSource securityDataSource;
```

17. Modificați metoda configure(AuthenticationManagerBuilder auth) astfel încât să se utilieze autentificare jdbc.

```
auth.jdbcAuthentication().dataSource(securityDataSource);
```