OTH-Regensburg
Übungen zur Vorlesung
Softwareentwicklung

Übung Nr. 6
Spring Security

Aufgabe 1 – Erstellen eines neuen Projekts und der Abhängigkeiten

• Importieren Sie die folgenden Abhängigkeiten, um Spring Security und JPA zu verwenden:

```
<!-- https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa -->
                        <dependency>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-starter-data-jpa</artifactId>
                        </dependency>

                        <dependency>
                          <groupId>com.h2database</groupId>
                          <artifactId>h2</artifactId>
                          <scope>runtime</scope>
                        </dependency>

                        <!-- Spring security -->
                        <dependency>
                                <groupId>org.springframework.security</groupId>
                        <artifactId>spring-security-web</artifactId>
                        </dependency>

                        <dependency>
                        <groupId>org.springframework.security</groupId>
                        <artifactId>spring-security-config</artifactId>
```

Stellen Sie sicher, dass Sie auch die Abhängigkeiten für Thymeleaf, Thymeleaf-Dialekt, Spring Web, Bootstrap, Jquery usw. haben (kopieren Sie sie hier aus der POM-Datei in diesem Projekt: https://elearning.oth-regensburg.de/mod/resource/view.php?id=272942)

Fügen Sie der Datei application.properties diese Werte hinzu:

```
spring.thymeleaf.cache=false
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html

spring.messages.basename=messages
server.error.include-binding-errors=always

#DB
spring.datasource.url=jdbc:h2:mem:testdb;MODE=MySQL;NON_KEYWORDS=USER;
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
#spring.data.jpa.repositories.bootstrap-mode=default

spring.jpa.defer-datasource-initialization=true

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

spring.jpa.hibernate.ddl-auto=update
```

- Erstellen Sie einen HomeController im Paket-Controller, der ein @Getmapping „/home“ zur Ansicht „home.html“ hat.

- Führen Sie das Projekt aus. Versuchen Sie, auf „/home“ zuzugreifen, und prüfen Sie, ob Sie das von Spring Security bereitgestellte Login-View sehen. Sie können versuchen, sich mit „user“ und dem von Spring Security generierten Passwort im Terminal anzumelden.

Aufgabe 2- Erstellen des Benutzers und der Berechtigungen

- Definieren Sie unter dem Paketmodell eine Klasse „User“ mit mindestens den folgenden Attributen: ID, Login, Passwort, aktiv (Integer), E-Mail. Vergessen Sie nicht die Annotation @Entity.

- Erstellen Sie eine Klasse Authority (ID, Beschreibungsfelder). Diese Klasse stellt die Profile Ihres Systems dar (in meinem Projekt werde ich später die ADMIN- und STUDENT-Authorities über die Datei data.sql in die Datenbank einfügen)..

```
@Entity
@Table(name="authority")
public class Authority {

        @Id
        @GeneratedValue (strategy = GenerationType.IDENTITY)
        private Long id;
        private String description;

//getters and setters
}
```

Tipp: Sie können dies auch direkt in Ihrem Projekt mit Ihren Authorities und Benutzern als echtes Feature umsetzen, anstatt meinen Profilen hier zu folgen.

- Erstellen Sie das entsprechende Repository für den Benutzer.

```
@Repository
public interface UserRepository extends
JpaRepository<User, Long> {

        Optional<User> findUserByLogin(String login);
}
```

- Gehen Sie zurück zur Klasse „Benutzer“, um ihr eine Liste von Autoritäten hinzuzufügen. In diesem Fall verwende ich die JoinTable „userauthority“, um die Beziehung zu speichern.

```
@ManyToMany(fetch = FetchType.EAGER)
        @JoinTable(
                        name="userauthority",
                        joinColumns = @JoinColumn(name="iduser"),
                        inverseJoinColumns = @JoinColumn(name="idauthority")
                        )
        //my own type of authority, not from spring security
        private List<Authority> myauthorities = new ArrayList<Authority>();
```

Achtung: Das bedeutet, dass der Benutzer in diesem System gleichzeitig Administrator und Student sein kann (das System lässt das mit dieser Modellierung zu). Wenn das in Ihrem System nicht möglich ist, verwenden Sie vielleicht stattdessen eine @ManyToOne-Beziehung.

Aufgabe 3 – Create MyUserDetails and MyUserDetailsService (required by Spring Security)

- Erstellen Sie unter dem Paket „config“ eine Klasse MyUserDetails, die die Schnittstelle „UserDetails“ von Spring Security implementiert.

```
public class MyUserDetails implements UserDetails{


        private static final long serialVersionUID = 1L;
        private String userName;
        private String password;
        private boolean active;
        private List<GrantedAuthority> authorities;

        // do not forget to add the implemented methods
        …
}
```

• Die wichtigste Methode in der Klasse ist ihr Konstruktor, der ihre Collection „Autoritäten“ und ihre Eigenschaften „Benutzer“ und „Passwort“ ändert:

```
public MyUserDetails(User user) {

                // TODO Auto-generated constructor stub
                this.userName= user.getLogin();
                this.password= user.getPassword();
                System.out.println("password of the user is="+password);
                System.out.println("userName of the user is="+this.userName);
                this.active = (user.getActive()>0)? true : false;

                List<Authority> myauthorities = (List<Authority>) user.getMyauthorities();

                System.out.println("the user "+  user.getLogin() +" has "+ myauthorities.size() +" authorities");

                authorities = new ArrayList<>();

                for (int i=0; i< myauthorities.size(); i++){
                     authorities.add(new SimpleGrantedAuthority(myauthorities.get(i).getDescription().toUpperCase()));
                    System.out.println("the profile" + i +" of "+ user.getLogin() + " is "myauthorities.get(0).getDescription());
                }
}
```

- Stellen Sie sicher, dass Sie verstehen, was in der obigen Methode passiert.

- Sie müssen einige Methoden so einstellen, dass sie „true“ zurückgeben, wie die folgenden. Analysieren Sie, welche davon sinnvollerweise auf „true“ eingestellt werden sollten.

```
@Override
public boolean isAccountNonExpired() {
	// TODO Auto-generated method stub
	return true;
}

@Override
public boolean isAccountNonLocked() {
	// TODO Auto-generated method stub
	return true;
}

@Override
public boolean isCredentialsNonExpired() {
	// TODO Auto-generated method stub
	return true;
}

@Override
public boolean isEnabled() {
	// TODO Auto-generated method stub
	return this.active;
}
public boolean isActive() {
	return active;
}

public void setActive(boolean active) {
	this.active = active;
}
```

• Es ist Zeit, den „MyUserDetailsService“ unter dem Paket „service“ zu erstellen. Kommentieren Sie diese Komponente mit „@Service“.

```
@Service
public class MyUserDetailService implements UserDetailsService{


	@Autowired
	UserRepository userRepository;


	public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
		// TODO Auto-generated method stub
		Optional<User> oUser= userRepository.findUserByLogin(username);
		oUser.orElseThrow(()-> new UsernameNotFoundException("Not found "+username));
		System.out.println("User found at the UserDetailsService="+ oUser.get().getLogin());
		return new MyUserDetails(oUser.get());
	}

}
```

Diese Klasse greift auf die User-Table zu, sucht nach einem Benutzernamen und stellt (sofern vorhanden) sein in der Datenbank gespeichertes Passwort wieder her. Dieser Service gibt ein Objekt vom Typ MyUserDetails (aus dem vorherigen Schritt) zurück, damit der Spring-Container es authentifizieren kann.

Aufgabe 4 – Erstellen der SecurityConfiguration (wird von der Spring Security gefordert)

In dieser mit @Configuration kommentierten Klasse vereinen wir MyUserDetailService und den Passwort-Encoder erneut und definieren die URLs, die eine Authentifizierung und Autorisierung benötigen (in der SecurityFilterChain-Definition).

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {

		@Autowired
		MyUserDetailsService userDetailsService;

		 @Bean
		 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {


		 http.authorizeHttpRequests(rQ -> {
			rQ.requestMatchers("/h2-console/**", "/console/**", "/signup/", "/signin/").permitAll();
			rQ.requestMatchers("/images/**", "/js/**", "/webjars/**").permitAll();

			rQ.requestMatchers("/admin/**", "/settings/**").hasAuthority("ADMIN") ;

			rQ.requestMatchers("/api/search/", "/api/profile/", "/signout/").authenticated();
		       });
		  //deprecate code, try to migrate it!
		 http.formLogin()
		 //.loginPage("/login") , if you want to have a customized login page....
		 .and()
		 .logout()
		 //where the user goes after the logout
		 .logoutSuccessUrl("/login")
		 .invalidateHttpSession(true)
		 .permitAll();

				//  If you choose to disable the X-Frame-Options header (not recommended) by setting
.headers().frameOptions().disable(), then Spring Security will not add the X-Frame-Options header to the response.
				//  This means your application could be rendered in a frame, and also could be vulnerable to
Clickjacking attacks.
				//Since the frames in the H2 console UI (such as http://localhost:8080/h2-console/tables.do) are on
the same origin as the the H2 console (http://localhost:8080/h2-console), the browser will allow them to be displayed.

				http
				.headers(headers -> headers
				   .frameOptions(frameOptions -> frameOptions
				      .sameOrigin())
				);

				 http.csrf(AbstractHttpConfigurer::disable);

			return http.build();
		 }

		@Bean
	public AuthenticationManager authenticationManager(
				AuthenticationConfiguration authenticationConfiguration) throws Exception {
	   return authenticationConfiguration.getAuthenticationManager();
	}

		@Bean
	public PasswordEncoder getPasswordEncoder() {
				return new BCryptPasswordEncoder();
				//return NoOpPasswordEncoder.getInstance();
    }
}
```

Erstellen Sie folgende Regeln für die jeweiligen URLs:

URL	Authorities
/home	No need of authentication
/admin	ADMIN
/student	ADMIN OR STUDENT

- Bearbeiten Sie die SecurityFilterChain in der SecurityConfig-Klasse über der URL-Konfiguration „/admin“.

```
.requestMatchers("/home").permitAll()
.requestMatchers ("/student/**").hasAnyAuthority("ADMIN","STUDENT")
```

Mehr zu Spring Security Expressions:
https://www.baeldung.com/spring-security-expressions

Verknüpfen Sie die obigen URLs im HomeController mit den Seiten home.html, admin.html und student.html.

```
@Controller
public class HomeController {

@Autowired
UserRepository userRepository;

@RequestMapping(value={"/", "/home"})
        public String home(HttpServletRequest request, Principal principal) {

                return “home”;
        }
@RequestMapping(“/admin“)
        public String showAdminHome(HttpServletRequest request, Principal principal){

                        return "admin";
        }

@RequestMapping(“/student“)
        public String showStudentHome(HttpServletRequest request, Principal principal){

                        return "student";
        }
```

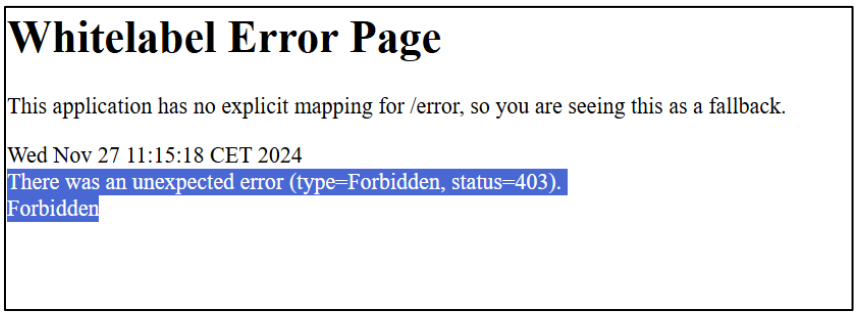
- Jetzt ist es an der Zeit, die Datenbank mit einigen Benutzern und Berechtigungen zu füllen. Erstellen Sie die folgende Datei data.sql:

```
INSERT INTO USER (email, password, login, active) VALUES ( 'thomas@gmail.com',
$2a$12$8K4uC9YPl659Qnz6NUqy9e35xsoQ/OIsaVhIWRJP913VpsulQGZNy', 'thomas',
1);
INSERT INTO USER (email, password, login, active) VALUES ( 'anja@gmail.com',
'$2a$12$8K4uC9YPl659Qnz6NUqy9e35xsoQ/OIsaVhIWRJP913VpsulQGZNy', 'anja', 1);

INSERT INTO AUTHORITY (description) VALUES ( 'ADMIN');
INSERT INTO AUTHORITY (description) VALUES ( 'STUDENT');

INSERT INTO USERAUTHORITY (IDUSER, IDAUTHORITY) VALUES ( 1, 1);
INSERT INTO USERAUTHORITY (IDUSER, IDAUTHORITY) VALUES ( 2, 2);
```

Verwenden Sie diese URL, um Ihre ersten Passwörter zu verschlüsseln:
https://bcrypt-generator.com/

In diesem Fall ist Thomas ein Admin, Anja eine Studentin.

- Versuchen Sie, auf /student als Administrator
- Verwenden Sie /logout, um die Sitzung zu beenden.
- und /admin als Student zuzugreifen. In diesem Fall bekommt man:



Nun sind die Seiten durch Authentifizierung geschützt und mit den entsprechenden Autorisierungsrechten ausgestattet.