Haaga-Helia

Back End Programming

Spring Security

updated Minna Pellikka

7.2.2025

Haaga-Helia

Spring Security

- Spring Security is customizable authentication and access control framework for Spring based applications
- To get started add dependency to pom.xml

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

Haaga-Helia

Spring Security

- By default Spring Security enables following features (out of the box)
  - An AuthenticationManager bean with in-memory single user (username = user, password from the log)
  - Ignored (insecure) paths for common static resource locations like /css, /images…
  - HTTP Basic security for all other endpoints
  - Security events published to Spring ApplicationEventPublisher
  - Common low-level features (HSTS, XSS, CSRF, caching) provided by Spring Security are on by default

Haaga-Helia

Spring Security

- Adding dependency secures your application automatically
- Spring Boot create one test user and password can be seen in the console when application starts (see, SecurityDemo)

```
Using default security password: 837a95a3-3546-4896-9689-7711133e9ca6
```

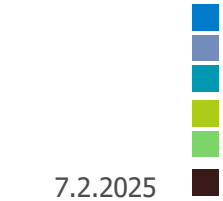
- Spring Security can be configured by creating Configuration class

```
@Configuration
@EnableMethodSecurity(securedEnabled = true) //you can use method level security
public class WebSecurityConfig {
```


Haaga-Helia

Spring Security

- WebSecurityConfig class contains a method `configure(HttpSecurity)` that defines which URL paths are secured and the path for login form



Haaga-Helia

Spring Security

```
@Configuration
public class WebSecurityConfig  {
@Bean
public SecurityFilterChain configure(HttpSecurity http) throws Exception {
        http
            .authorizeHttpRequests( authorize -> authorize
                .requestMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
             )
            .formLogin( formlogin -> formlogin
                .loginPage("/login")
                .defaultSuccessUrl("/studentlist", true)
                .permitAll()
            )
            .logout( logout -> logout
                .permitAll()
             );
            return http.build();
    }
}
```

/ and /home paths are configured to not require any authentication. All other paths must be authenticated

If you don't give loginPage your application will use the spring boot default login page.

Tells where to go after successful login

Spring Security

- Configuration examples
  - Requires user authentication in all URLs

```
public SecurityFilterChain configure(HttpSecurity http) throws Exception {
   http
      .authorizeHttpRequests( authorize -> authorize
            .anyRequest().authenticated()
       )
      .formLogin( formlogin -> formlogin
            .loginPage("/login")
            .defaultSuccessUrl("/studentlist", true)
            .permitAll()
       );
      return http.build();
}
```


Spring Security

- Configuration examples
  - Any user can access a request if the URL starts with "/resources/", equals "/signup", or equals "/about"
  - Any URL that starts with "/admin/" will be restricted to users who have the role 'ADMIN'.

```
import static org.springframework.security.web.util.matcher.AntPathRequestMatcher.antMatcher;
public SecurityFilterChain configure(HttpSecurity http) throws Exception {
  http.authorizeHttpRequests( authorize -> authorize
            .requestMatchers(antMatcher("/resources/**")).permitAll()
             .requestMatchers(antMatcher("/signup" )).permitAll()
            .requestMatchers(antMatcher("/about")).permitAll()
            .requestMatchers(antMatcher("/admin/**")).hasRole("ADMIN")
            .anyRequest().authenticated()
   )// ...
   return http.build() ; }
```


Spring Security

- Create in-memory users
  - This is only for testing and demo purposes (Security configuration class)
  - You can add multiple user using `Collection<UserDetails>`

```
@Bean
public UserDetailsService userDetailsService() {
  UserDetails user =     User.withDefaultPasswordEncoder()
        .username("user")
        .password("password")
        .roles("USER")
        .build();
        List<UserDetails> users = new ArrayList();
        users.add(user);
  return new InMemoryUserDetailsManager(users);
}
```

Haaga-Helia

Spring Security

- Login
  - Create method to controller and thymeleaf template for login
  - Thymeleaf: method = POST
  - Spring Security provides a filter that intercepts request to /login and authenticates the user
  - If the user fails to authenticate, the page is redirected to "/login?error" endpoint

```
<div th:if="${param.error}">
        Invalid username and password.
</div>
<form th:action="@{/login}" method="post">
        <div><label> User Name : <input type="text" name="username"/> </label></div>
        <div><label> Password: <input type="password" name="password"/> </label></div>
        <div><input type="submit" value="Sign In"/></div>
</form>
```

Haaga-Helia

Spring Security

- Logout
  - Thymeleaf: method = POST
  - After successfully logging out user will be redirected to "/login?logout‟ endpoint

```
<form th:action="@{/logout}" method="post">
    <input type="submit" value="Sign Out"/>
</form>
```

- You can show logged in user by using Spring Security

```
<h1>Hello <span sec:authentication="name"></span>!</h1>
```

Haaga-Helia

Spring Security

- CSRF
  - Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated (OWASP)
  - To protect against CSRF attacks we need to ensure there is something in the request that the ‘evil’ site is unable to provide.
  - CSRF protection is enabled as a default in Spring Security
  - With Thymeleaf the csrf token is automatically included for you

```
<input type="hidden" name="_csrf" value="d63f746f-c5f6-4cc6-99c0-9220ff784b23" /></form>
```

Haaga-Helia

Spring Security

- Spring Security Thymeleaf dialects can be used to show different content to different roles
- Add dependency

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
```

- Add definition for sec: attributes to your Thymeleaf template

```
<html xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity6">
```

- Example: Only ADMIN users can delete students (StudentList example)

```
<td sec:authorize="hasAuthority ('ADMIN')"><a
th:href="@{/delete/{id}(id=${student.id})}">Delete</a></td>
```

Note! In the case of in-memory users use hasRole instead of hasAuthority.

Haaga-Helia

Method level security

- Add following annotation to Web Security config class

```
@EnableMethodSecurity(securedEnabled = true)
```

- Add `@PreAuthorize` annotation to the method you want to secure

```
@RequestMapping(value = "/delete/{id}", method = RequestMethod.GET)
@PreAuthorize("hasRole('ADMIN')")
public String deleteStudent(@PathVariable("id") Long studentId, Model
      model) {
  repository.deleteById(studentId);
  return "redirect:../studentlist";
}
```

Haaga-Helia

Spring Security: User entity

- How to use Users from database in authentication?

1.) Create User Entity

```
@Entity(name= "users")
public class User {  // or AppUser
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false, updatable = false)
    private Long id;

    // Username with unique constraint
    @Column(name = "username", nullable = false, unique = true)
    private String username;

    @Column(name = "password", nullable = false)
    private String passwordHash;

    @Column(name = "role", nullable = false)
    private String role;
    // Getters and setters...
```

Haaga-Helia

Spring Security: User entity

2.) Create User crud repository

```
public interface UserRepository extends CrudRepository<User, Long> {
    User findByUsername(String username);
}
```

Haaga-Helia

Spring Security: User entity

3.) Implement `UserDetailService` interface. Spring Security is using it to authenticate and authorize user

```
@Service
public class UserDetailServiceImpl implements UserDetailsService  {
   private final UserRepository repository;

   public UserDetailServiceImpl(UserRepository userRepository) {
      this.repository = userRepository;
   }
   @Override
   public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
   User curruser = repository.findByUsername(username);
   UserDetails user = new org.springframework.security.core.userdetails.User(username,
         curruser.getPasswordHash(),
         AuthorityUtils.createAuthorityList(curruser.getRole()));
   return user;
  }
}
```

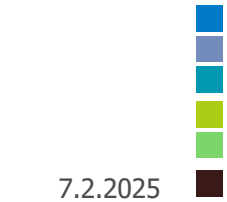

Haaga-Helia

Spring Security: User entity

4.) Change Spring Security configuration to use your UserDetailService implementation.

Use `BCryptPasswordEncoder` to encrypt passwords using Bcrypt hash algorithm (Default number of rounds is 10). You can also use constructor to give strength between 4-31.

```
...public class WebSecurityConfig {
private UserDetailsService userDetailsService; // type of attribute -> interface
private UserDetailsService userDetailsService; public WebSecurityConfig(UserDetailsService
userDetailsService) {
        this.userDetailsService = userDetailsService;
}
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(new
        BCryptPasswordEncoder());
}
```

Haaga-Helia

Spring Security: User entity

5.) Create some demo users to your database in CommandLineRunner.
Hint: You can use Bcrypt calculators to create hashed passwords.

```
// Create users: admin/admin user/user
User user1 = new User("user",
"$2a$06$3jYRJrg0ghaaypjZ/.g4SethoeA51ph3UD4kZi9oPkeMTpjKU5uo6", "USER");
User user2 = new User("admin",
"$2a$10$0MMwY.IQqpsVc1jC8u7IJ.2rT8b0Cd3b3sfIBGV2zfgnPGtT4r0.C", "ADMIN");
urepository.save(user1);
urepository.save(user2);
```

Haaga-Helia

HttpSession

- Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user https://jakarta.ee/specifications/platform/9/apidocs/index.html?jakarta/servlet/http/HttpSession.html
- Session information is scoped only to the current web application

Haaga-Helia

HttpSession

- How to use HttpSession
  - Inject HttpSession (@Autowired)
  - Use `addAttribute` and `getAttribute` methods to store and retrieve data

```
@Autowired
private HttpSession session;
...
session.setAttribute("myObject", new SomeObject());
 ...
SomeObject s = session.getAttribute("myObject");
```

Haaga-Helia

HttpSession

- Session can be invalidated (unbind all objects bound to session) when user logs out from the application
- Modify Spring security configuration class

```
@Bean
public SecurityFilterChain configure(HttpSecurity http) throws Exception {
  http
     .authorizeHttpRequests(authorize -> authorize
         .anyRequest().authenticated())
      .formLogin(formlogin -> formlogin
          .loginPage("/login").defaultSuccessUrl("/studentlist", true).permitAll())
      .logout(formlogin -> formlogin
          .permitAll().invalidateHttpSession(true)) // Invalidate session
   // ...
   return http.build();
}
```