05 - Spring Security

Table of Contents

Agenda

- Authentication and Authorisation
- Introduction to Spring Security
- Role-based access control
  - Using the Security Configuration file
  - Using @PreAuth annotation

Authentication:

Process of verifying user credentials.
Who Are You?

Authorisation:

Process of determining what authenticated users are allowed to do.
What Permissions Do You Have?

Introduction to Spring Security:

Spring Security is a powerful and highly customisable authentication and access control framework. It provides comprehensive security services for Java EE-based enterprise software applications.

Hands-On

1. Go to Spring Initialiser - and download SpringSecurity Project with the following dependencies:

    1. Spring Web
    2. Lombok
    3. Spring Security
2. Open the Project and Run it
    1. Try to log in using the username as 'user' and password from the logs.
3. Go to application.properties and set Custom username and password

```
spring.security.user.name=Testing
spring.security.user.password=test123
```

4. Create `AccessController` in the controllers directory

```
@RestController
@RequestMapping("/")
public class AccessConrtoller {

    @GetMapping
    public String homePage(){
        return "WELCOME TO SPRING SECURITY HOME PAGE";
    }

    @GetMapping("subscriber")
    public String paidContent(){
        return "THIS CONTENT IS FOR OUR PAID USERS ONLY.";
    }

    @GetMapping("admin")
    public String adminDashboard(){
        return "WELCOME ADMIN! HERE ARE YOU SETTINGS.";
    }
}
```

5. Create the `SecurityConfig.java` file in the `config` directory

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

        // Configures security filters for HTTP requests
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http)
throws Exception{
        http
                        // Authorization configuration for different URL
patterns
                .authorizeHttpRequests(auth -> auth
```

```
                        .anyRequest().authenticated()// All other requests
require authentication
                )
                // Use HTTP Basic authentication with default settings
                .httpBasic(Customizer.withDefaults());
                // Build and return the configured HttpSecurity object
        return http.build();
    }

        // Configures a password encoder for encoding passwords
    @Bean
    public PasswordEncoder passwordEncoder(){
        return new BCryptPasswordEncoder();
    }

        // Configures in-memory users for authentication
    @Bean
    public UserDetailsManager users(){
    // Define user details for user1 (admin), user2 (subscriber), user3
(normal)
        UserDetails user1 = User.builder()
                .username("admin")
                .password(passwordEncoder().encode("admin")) // Encode
password using the configured password encoder
                .roles("ADMIN") // Assign roles to the user
                .build();

        UserDetails user2 = User.builder()
                .username("subscriber")
                .password(passwordEncoder().encode("subs"))
                .roles("SUBSCRIBER")
                .build();

        UserDetails user3 = User.builder()
                .username("normal")
                .password(passwordEncoder().encode("free"))
                .roles("NORMAL")
                .build();

                // Create an InMemoryUserDetailsManager with the defined
users
        return  new InMemoryUserDetailsManager(user1, user2, user3);
    }
}
```

- @Configuration - class-level annotation indicating that an object is a source of bean definitions.

- @EnableWebSecurity - `SecurityConfig` class is annotated with `@EnableWebSecurity` to enable Spring Security's web security support and provide the Spring MVC integration.
  Now the custom username and passwords will work

6. Role-based access using the `requesMatchers` in `SecurityConfig` file

```
.authorizeHttpRequests(auth -> auth
        .requestMatchers("/").permitAll() // Allow access to the root URL
for everyone

.requestMatchers("/subscriber/**").hasAnyRole("SUBSCRIBER","ADMIN") //
Allow access to /subscriber/** for users with roles SUBSCRIBER or ADMIN
        .requestMatchers("/admin/**").hasRole("ADMIN") // Allow access to
/admin/** for users with role ADMIN
        .anyRequest().authenticated()
)
```

Now specific users will get access to specific routes

7. Role-based access using `@PreAuth`:
   Create new route in `AccessContoller`

```
@RestController
@RequestMapping("/")
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class AccessConrtoller {

    /*--- existing code ---*/

    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("maintain")
    public String maintainPage(){
        return "MAINTENANCE HAPPENING HERE!";
    }
}
```

- @EnableGlobalMethodSecurity(prePostEnabled = true) - Enables method-level security annotations like @PreAuthorize.
- @PreAuthorize("hasRole('ADMIN')") - Ensures that only users with the ADMIN role can access the `maintainPage()` method

---