# Spring Boot Security

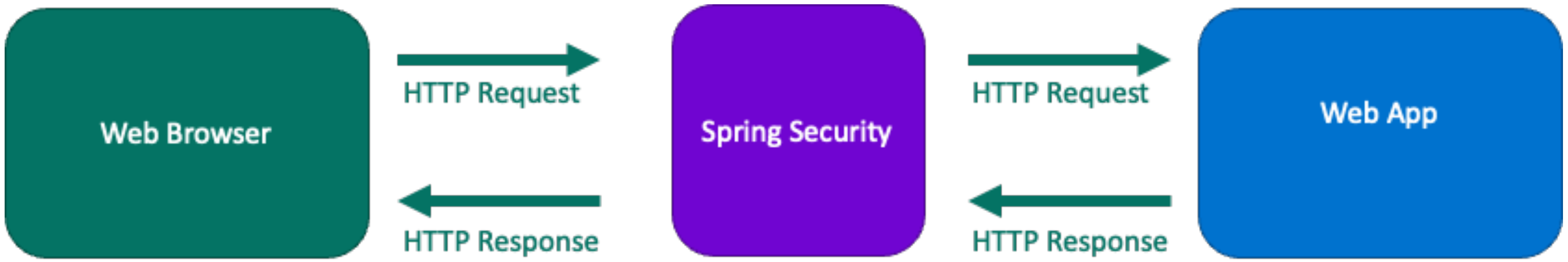
## Spring Security

- Spring Security is a powerful and customizable module in Spring framework
- It provides a wide range of security features, including authentication, authorization and support for various security protocols and standards
- Spring Security can be easily integrated with Spring Boot

## Spring Security - benefits

- Spring Security can be easily integrated with Spring Boot
- It is highly customizable and can be configured to meet security requirements
- Has pre-built security features such as login pages, password encryption and session management
- It protects against common security threats like cross site scripting (XSS) and cross-site request forgery (SCRF)

# Spring Security



## Steps to implement Spring Security

- Add the Spring Security dependency to pom.xml (starter dependency)
- Everything in the web application will be protected
- Then configure Spring Security

## Demo 1 - Start using Spring Security

- Enable Spring Security
- The Configuration

## Tutorial - Step 1

- Use the SpringSecurityStarter application
- Try to run the application and go to localhost:8080 and you should see that there are two pages, start and secret, with corresponding Controller methods
- Spring Security is not enabled and is not protecting the application
- Now go to the pom.xml file and uncomment the starter dependency for spring-boot-starter-security, update Maven and restart the application
- Then try to go to localhost:8080 and you should see that all of the application is now protected by Spring Security

## Tutorial - Step 2

- You have not configured any users with username and password, so how could you log in to the application?
- Spring Boot provides  a generated password in the console, log in with the default username user and the password you find in the console
- Now you should be able to access the start page and the secret page
- But we can’t continue using generated passwords, can we?

## Tutorial - Step 3

- Uncomment all the code in the class SecurityConfig
- Look at the auth0FilterChain method. The URL to the start page is permitted for everyone and if trying to access any other URL will result in a default login form being used
- Look at the userDetailsService method, here two users are defined with username, password and different roles
- Now restart the application and try the secret page, login with the username and password of one of the users in the userDetailsService method

## Tutorial - Step 4

- Restart the application (so that you are logged out)
- Now try to access the URL localhost:8080/home
- This is not possible, but by changing the configuration we can make this URL permitted for everyone by adding the string "/home" as a second parameter in the requestMatchers method, comma separated after the string "/" like this: requestMatchers("/", "/home").permitAll()
- Adding a URL here will mean it is permitted without logging in, so now both localhost:8080/ and localhost:8080/home are permitted for everyone

# Tutorial - Step 5

- Restart the application (so that you are logged out) and try to access the URL localhost:8080/test
- After logging in you would get a 404 Not Found error because the URL /test does not exist in the application
- To avoid this you could specify a default success url that will always be displayed after loggin in. Change the configuration like this, restart the application and try again!

```
http.formLogin().defaultSuccessUrl( defaultSuccessUrl: "/secret",  alwaysUse: true);
```

## Tutorial - Step 6

- Cumbersome to have to restart the application to log out?

- Add this form to the secret page:

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

- Then restart the application, log in and you should be able to log out!

## Tutorial - Step 7

- Maybe you would rather get back to the start page when loggin out?
- Add this line in the auth0FilterChain method to specify where to go after a successful logout:

```
http.logout().logoutSuccessUrl("/");
```

- Then try this out and see where you end up after having logged out!

## Tutorial - Step 8

- When logged in on the secret page, why not display the username of the user that is currently logged in?
- Uncomment the dependency for thymeleaf-extras-springsecurity6 in pom.xml to make it possible to use sec: attributes in thymeleaf (sec is for security thymeleaf attributes)
- Then add this line in the secret page and you should see the logged in username on the secret page:

```
<p>Logged in as:  <span sec:authentication="name"></span></p>
```

# Tutorial - Step 9

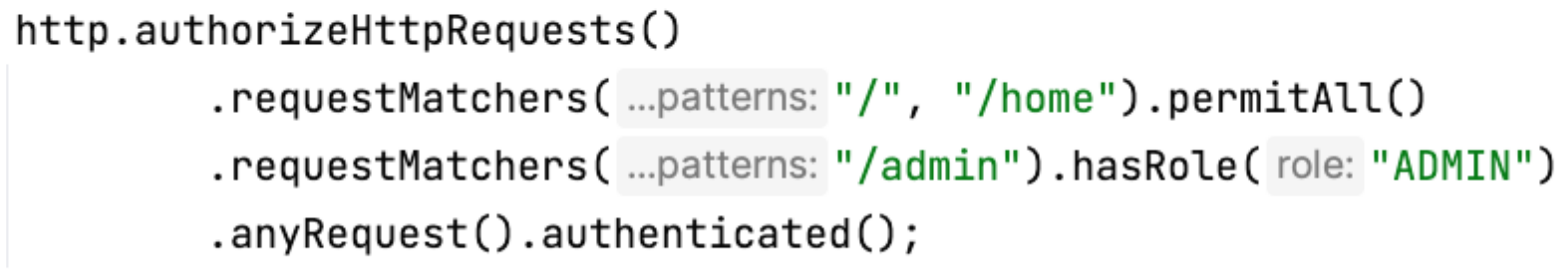
- Both users from the SecurityConfig class can access the same things when logged in, but maybe the admin should see more information on the secret page than the user?

- Add this html code in the secret page and it will only be visible when it is the admin that is logged in:

```
<div sec:authorize="hasRole('ADMIN')">
    <h2>Very very secret - only for admins!</h2>
</div>
```

## Tutorial - Step 10

- Maybe only the admin should have access to a special admin page?
- Create a new template called admin and a new Controller method with a GetMapping to /admin that returns the name of the template
- Then change the configuration like this and you should only be able to access the admin page with /admin if you are logged in as admin:



```
http.authorizeHttpRequests()
        .requestMatchers( ...patterns: "/", "/home").permitAll()
        .requestMatchers( ...patterns: "/admin").hasRole( role: "ADMIN")
        .anyRequest().authenticated();
```

# Tutorial - Step 11

- Is it inconvenient to have to write the URL to /admin in the address bar?

- Add a link to the /admin page in the secret page, but only visible to the admin by adding it inside the div with admin access:

```
<div sec:authorize="hasRole('ADMIN')">
    <h2>Very very secret - only for admins!</h2>
    <a href="/admin">Admin page</a>
</div>
```

# Tutorial - Step 12

- Create a custom login page instead of the one generated by Spring Security by creating a GetMapping method in the Controller with a mapping to /login and that returns the template name login, and change the configuration in SecurityConfig of the http.formLogin to this:

```
http.formLogin().loginPage("/login").defaultSuccessUrl("/secret", false).permitAll();
```

- Create a new template with a login form like this:

```
<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>
```

## Tutorial - Step 13

- Now you should be able to use your new login page, but did you notice that if you enter incorrect username or password, that you don't get any error message about this?

- Add this div to the login page:

```
<div th:if="${param.error}">
    Invalid username and password.
</div>
```

- Now you will get an error message when trying to log in with incorrect information. How it works? Do you see the request param ?error in the address bar in the web browser? This is what the th:if is checking

## End of the Tutorial

- Try to use your new skills to secure another application!
- Learn more about Spring Security here:
- https://spring.io/projects/spring-security