Securing a MVC Web Application

Description: Spring Boot is an open source, micro service-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its own codebase.

Project: Build a simple MVC web application with login security using Spring Security. This project is based on "Securing a Web Application" and just expands on the screenshots and step by step instructions.

Technology: This project uses the following technology:

- Integrated Development Environment (IDE):
  - Spring Tool Suite 4 (Version: 4.11.0.RELEASE)
- Java Development Kit (JDK):
  - Oracle's JDK 8 (1.8)
- Other tools:
  - Postman – a web and desktop application used for API testing.

Table of Contents

Glossary of Terminology

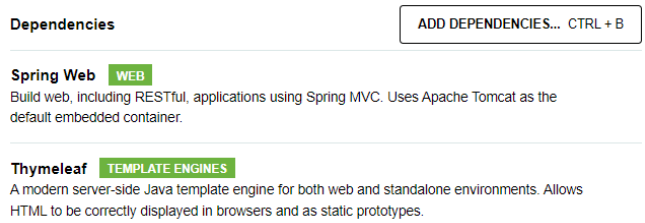
For a list of key terms and definitions used throughout this and various Spring Boot demo documents see the document titled “Appendix 01 Glossary”.

Generate Spring Boot Download

Follow the instructions in the document title “Appendix 02 Spring Initializr” to generate a spring boot download for this project.

When the document talks about adding dependencies add only these:

Spring Web
Thymeleaf



When the document talks about the items in the “**Project Metadata**” use the values shown below:

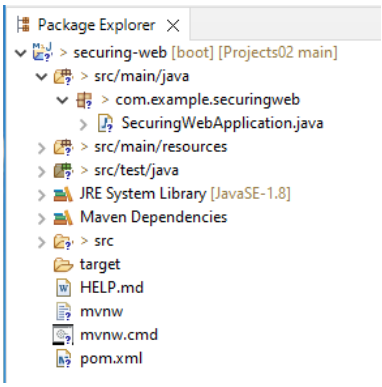
“**Group**” use “**com.example**”
“**Artifact**” use “**securing-web**”
“**Name**” use “**securing-web**”
“**Package name**” use “**com.example.securing-web**”

For other items in the “**Project Metadata**” use the defaults. Follow the instructions to extract the files from the zip file into the Sprint Tool Suite 4 workspace.

Import the Spring Boot Download

Import the project: “securing-web”

Follow the instructions in the document title “Appendix 03 Import Project” and import the “**securing-web**” project that was created with Spring Initializr. After importing the project, it should look like the following using the IDE “Package Explorer”.



Look at the pom.xml and find the added dependencies for this project.

```
19    <dependencies>
20        <dependency>
21            <groupId>org.springframework.boot</groupId>
22            <artifactId>spring-boot-starter-thymeleaf</artifactId>
23        </dependency>
24        <dependency>
25            <groupId>org.springframework.boot</groupId>
26            <artifactId>spring-boot-starter-web</artifactId>
27        </dependency>
28
29        <dependency>
30            <groupId>org.springframework.boot</groupId>
31            <artifactId>spring-boot-starter-test</artifactId>
32            <scope>test</scope>
33        </dependency>
34    </dependencies>
```

Project Discussion

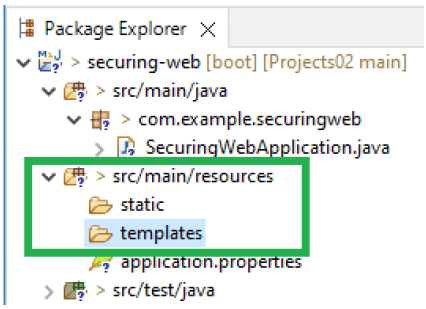
The final project will incorporate Spring Security for validation of the logon users. To start the project build an unsecure MVC application then add the Spring Security.

Create the Project from the Import

Using the Spring Tool Suite (STS) 4 IDE.

Create View Components

This project introduces creating view components within the STS 4 IDE. The view components are created in a "templates" resources folder.



The view components will incorporate Thymeleaf Spring Security. Thymeleaf support building view web components and assists with application security.

Create View Component: home.html

> "The web application includes two simple views: a home page and a "Hello, World" page. The home page is defined in the following Thymeleaf template (from `src/main/resources/templates/home.html`)" -- "Securing a Web Application"

Create a file under src/main/resources/templates named: "home.html"
Right click on "templates" folder ➔ "New" ➔ "File"
Add the following code to the file:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Spring Security Example</title>
    </head>
    <body>
        <h1>Welcome!</h1>

        <p>Click <a th:href="@{/hello}">here</a> to see a greeting.</p>
    </body>
</html>
```

Create View Component: home.html

> "This simple view includes a link to the `/hello` page, which is defined in the following Thymeleaf template (from `src/main/resources/templates/hello.html`)" -- "Securing a Web Application"

Create a file under src/main/resources/templates named: "hello.html"
Add the following code to the file:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
      xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1>Hello world!</h1>
    </body>
</html>
```

Create MVC Component

> "The web application is based on Spring MVC. As a result, you need to configure Spring MVC and set up view controllers to expose these templates. The following listing (from `src/main/java/com/example/securingweb/MvcConfig.java`) shows a class that configures Spring MVC in the application" -- "Securing a Web Application"

Create MVC Component: MvcConfig.java

Create a Java class under src/main/java/com/example/securingweb named: "MvcConfig"

Replace the generated class shell with the following code:

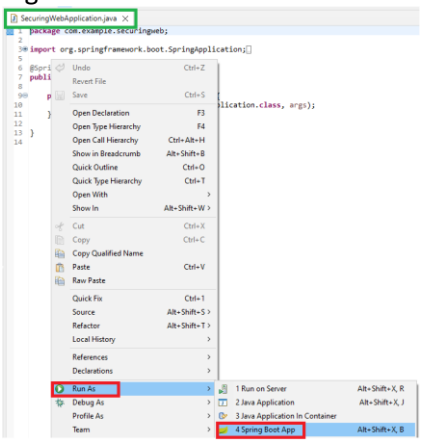
```
package com.example.securingweb;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class MvcConfig implements WebMvcConfigurer {

        public void addViewControllers(ViewControllerRegistry registry) {
                registry.addViewController("/home").setViewName("home");
                registry.addViewController("/").setViewName("home");
                registry.addViewController("/hello").setViewName("hello");
                registry.addViewController("/login").setViewName("login");
        }

}
```

> "The `addViewControllers()` method (which overrides the method of the same name in `WebMvcConfigurer`) adds four view controllers. Two of the view controllers reference the view whose name is `home` (defined in `home.html`), and another references the view

named `hello` (defined in `hello.html`). The fourth view controller references another view named `login`. You will create that view in the next section." -- "Securing a Web Application"

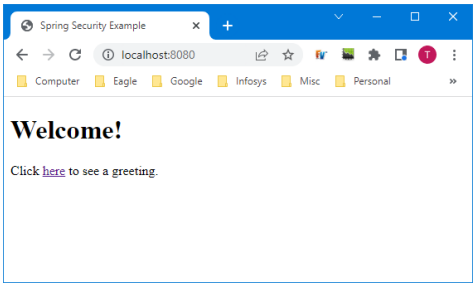
Test the Unsecured Application

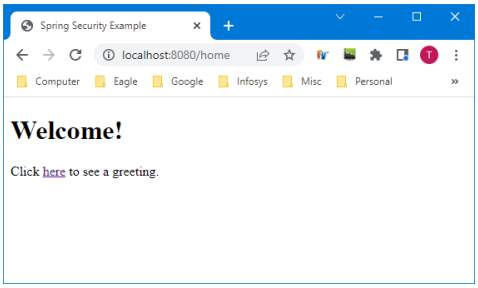
At this point the project complete a simple web application without security.
Start the application in the IDE as a Spring Boot Application.
Right click inside the main class ➔ "Run As" ➔ "Spring Boot App".

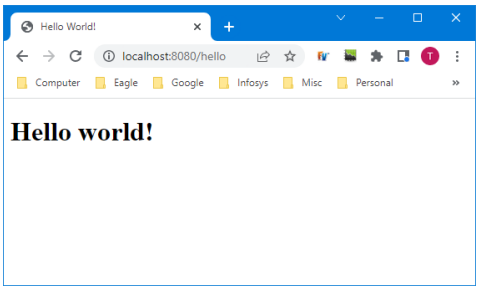


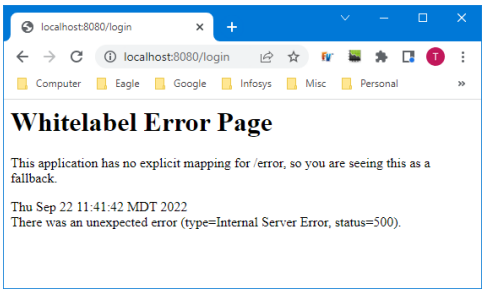
Test the application by using each of the URL endpoints defined in the MvcConfig class. At this time the login endpoint is not defined in the program and will display an error.

```
http://localhost:8080/
http://localhost:8080/home
http://localhost:8080/hello
http://localhost:8080/login
```









Make the Unsecured Application a Secured Application

Set up Spring Security

Suppose that you want to prevent unauthorized users from viewing the greeting page at `/hello`. As it is now, if visitors click the link on the home page, they see the greeting with no barriers to stop them. You need to add a barrier that forces the visitor to sign in before they can see that page.

You do that by configuring Spring Security in the application. If Spring Security is on the classpath, Spring Boot automatically secures all HTTP endpoints with "basic" authentication. However, you can further customize the security settings. The first thing you need to do is add Spring Security to the classpath.
-- "Securing a Web Application"

Setting up Spring Security: Maven Project

Update the pom.xml to add enhanced Spring Security. Add the following to the file:

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

```
20    <dependencies>
21        <dependency>
22            <groupId>org.springframework.boot</groupId>
23            <artifactId>spring-boot-starter-thymeleaf</artifactId>
24        </dependency>
25        <dependency>
26            <groupId>org.springframework.boot</groupId>
27            <artifactId>spring-boot-starter-web</artifactId>
28        </dependency>
29        <dependency>
30            <groupId>org.springframework.boot</groupId>
31            <artifactId>spring-boot-starter-security</artifactId>
32        </dependency>
33        <dependency>
34            <groupId>org.springframework.security</groupId>
35            <artifactId>spring-security-test</artifactId>
36            <scope>test</scope>
37        </dependency>
38
39        <dependency>
40            <groupId>org.springframework.boot</groupId>
41            <artifactId>spring-boot-starter-test</artifactId>
42            <scope>test</scope>
43        </dependency>
44    </dependencies>
```

Create Security Configuration Class: WebSecurityConfig

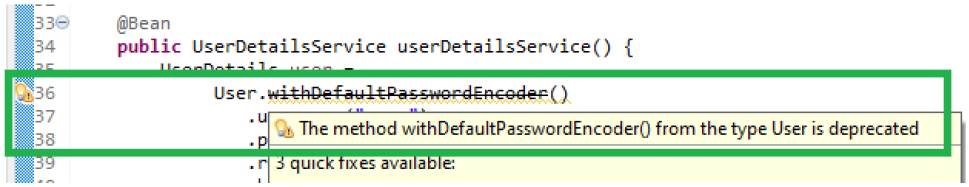
Create a Java class under src/main/java/com/example/securingweb named: "WebSecurityConfig". This configuration will restrict access to the greeting page.

Replace the generated class shell with the following code:

```
package com.example.securingweb;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
```

```java
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

	@Bean
	public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
		http
			.authorizeHttpRequests((requests) -> requests
				.antMatchers("/", "/home").permitAll()
				.anyRequest().authenticated()
			)
			.formLogin((form) -> form
				.loginPage("/login")
				.permitAll()
			)
			.logout((logout) -> logout.permitAll());

		return http.build();
	}

	@Bean
	public UserDetailsService userDetailsService() {
		UserDetails user =
			User.withDefaultPasswordEncoder()
				.username("user")
				.password("password")
				.roles("USER")
				.build();

		return new InMemoryUserDetailsManager(user);
	}
}
```

You will notice there is a warning concerning using withDefaultPasswordEncoder() method. The method is deprecated. However it can still be used in a demo program. It should not be used in a production environment.



Spring IO Documentation:

Deprecated.
Using this method is not considered safe for production, but is acceptable for demos and getting started. For production purposes, ensure the password is encoded externally. See the method Javadoc for additional details. There are no plans to remove this support. It is deprecated to indicate that this is considered insecure for production purposes.

The `WebSecurityConfig` class is annotated with `@EnableWebSecurity` to enable Spring Security's web security support and provide the Spring MVC integration. It also exposes two beans to set some specifics for the web security configuration:

The `SecurityFilterChain` bean defines which URL paths should be secured and which should not. Specifically, the `/` and `/home` paths are configured to not require any authentication. All other paths must be authenticated.

When a user successfully logs in, they are redirected to the previously requested page that required authentication. There is a custom `/login` page (which is specified by `loginPage()`), and everyone is allowed to view it.

The `UserDetailsService` bean sets up an in-memory user store with a single user. That user is given a user name of `user`, a password of `password`, and a role of `USER`.

-- "Securing a Web Application"

Create View Component: login.html

We saw an error when we tested the login endpoint. This is due to not having a login landing page. Here we create the login.html page.

Create a file under src/main/resources/templates named: "login.html"
Add the following code to the file:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
      xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Spring Security Example </title>
    </head>
    <body>
        <div th:if="${param.error}">
            Invalid username and password.
        </div>
        <div th:if="${param.logout}">
            You have been logged out.
        </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>
    </body>
</html>
```

Update View Component: home.html

Update the page to display user information and status.

Create a file under src/main/resources/templates named: "hello.html"
Replace the one line between the body tags with the following:

```
        <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
        <form th:action="@{/logout}" method="post">
            <input type="submit" value="Sign Out"/>
        </form>
```

```
hello.html
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
3        xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
4      <head>
5          <title>Hello World!</title>
6      </head>
7      <body>
8          <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
9          <form th:action="@{/logout}" method="post">
10             <input type="submit" value="Sign Out"/>
11         </form>
12     </body>
13 </html>
```

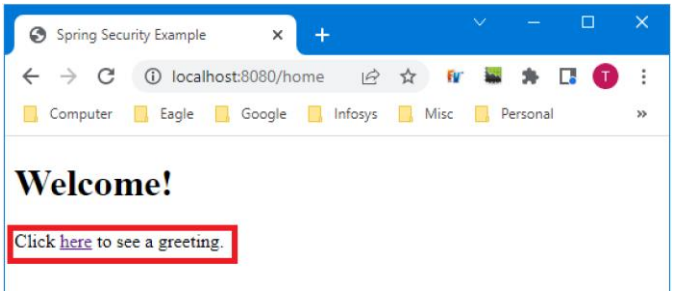
Test the Secure Application

At this point the project complete a simple web application without security.
Start the application in the IDE as a Spring Boot Application.
Right click inside the main class ➔ "Run As" ➔ "Spring Boot App".

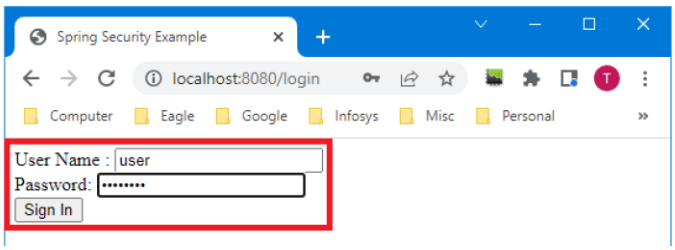
Use the home endpoint in a browser:

```
http://localhost:8080/home
```

Use the click "here" link.

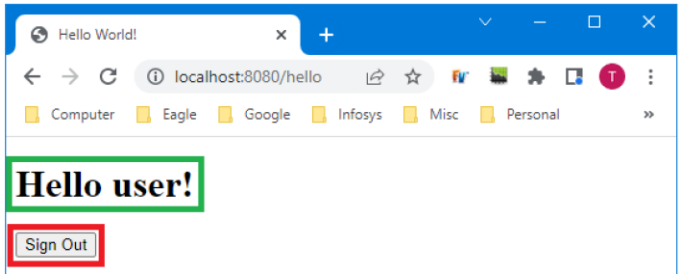


In the unsecure version of the application you would have been sent to the hello page. Now you are directed to the login page:



Enter "User Name:" "user"
Enter "Password:" "password"
"Sign In"

Once authenticated you are sent to the hello page where it now display the user name and a sign out feature.



Again the authentication method in this program is fine for demonstration purpose but should never be used in a production environment.

Project Conclusion

This concludes the simple MVC web application with login security using Spring Security. The login security method is deprecated where is can be used in a demo program and should never be used in a production environment.