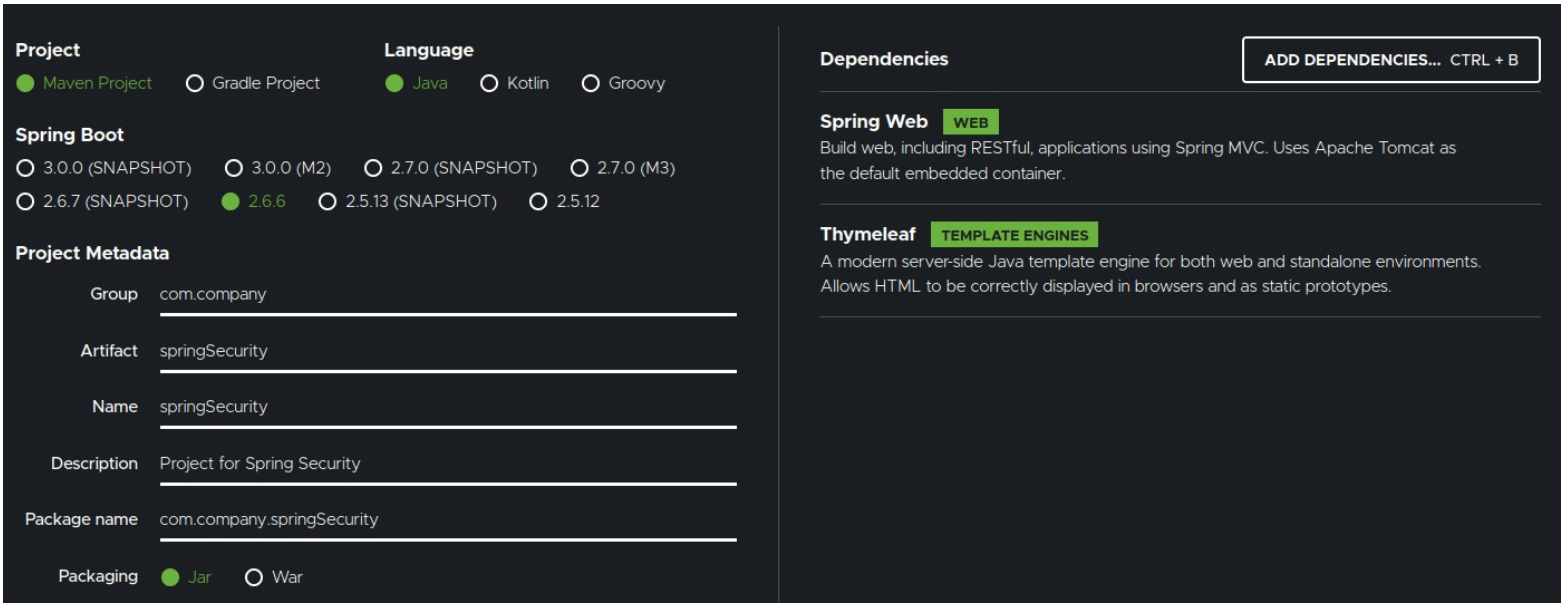
Внедрение в проект Spring security

Spring Security это Java/Java EE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для промышленных приложений, созданных с помощью Spring Framework.

1. Для начала создадим проект.

Проект будем создавать используя Spring Initializr https://start.spring.io/



Добавляем зависимости:

Spring Web для подключения модуля для создания web сайтов.

Spring Thymeleaf Это механизм шаблонов Java для обработки и создания HTML, XML, JavaScript, CSS и текста.

2. Откроим проект в Idea, создадим файлы html

В папке «resources/templates» создадим файл home.html и внесём в него следующие данные:

```
<!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>
```

Эта ссылка `th:href="@{/hello}"` указывает на страницу /hello в той же директории.
Создадим её и внесём в неё следующий код:

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

В последующем мы сможем получить имя пользователя, после его входа через следующий метод:

```
 <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
```

Далее создадим в той же папке файл login.html с содержанием:

```
<!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>
```

Этот шаблон Thymeleaf представляет собой форму, которая фиксирует имя пользователя и пароль и отправляет их на /login. В соответствии с настройками Spring Security предоставляет фильтр, который перехватывает этот запрос и аутентифицирует пользователя. Если пользователь не проходит аутентификацию, страница перенаправляется на /login?error, и на вашей странице появится соответствующее сообщение об ошибке. После успешного выхода ваша заявка будет отправлена на /login?logout, и на вашей странице отобразится соответствующее сообщение об успешном завершении.

3. Создадим класс SpringMVC в основной директории.

Веб-приложение основано на Spring MVC. В результате нам необходимо настроить Spring MVC и настроить контроллеры представления для предоставления этих шаблонов. Следующий код показывает класс, который настраивает Spring MVC в приложении:

```
package com.company.springSecurity;

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 {

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

}
```

addViewControllers()метод (который переопределяет одноименный метод в WebMvcConfigurer) добавляет четыре контроллера представления. Два контроллера представления ссылаются на представление, имя которого home(определено в home.html), а другой ссылается на представление с именем hello(определено в hello.html). Четвертый контроллер представления ссылается на другое представление с именем login. Далее мы создадим это представление.

4. Обеспечение безопасности:

В pom.xml добавляем следующие зависимости:

```
<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>
```

Создадим класс WebSecurityConfig.java.

Следующая конфигурация безопасности (WebSecurityConfig.java) гарантирует, что только аутентифицированные пользователи смогут видеть данные файла /hello:

```
package com.company.springSecurity;

import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapte
r;
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;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http
			.authorizeRequests()
				.antMatchers("/", "/home").permitAll()
				.anyRequest().authenticated()
				.and()
			.formLogin()
				.loginPage("/login")
				.permitAll()
				.and()
			.logout()
				.permitAll();
	}

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

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

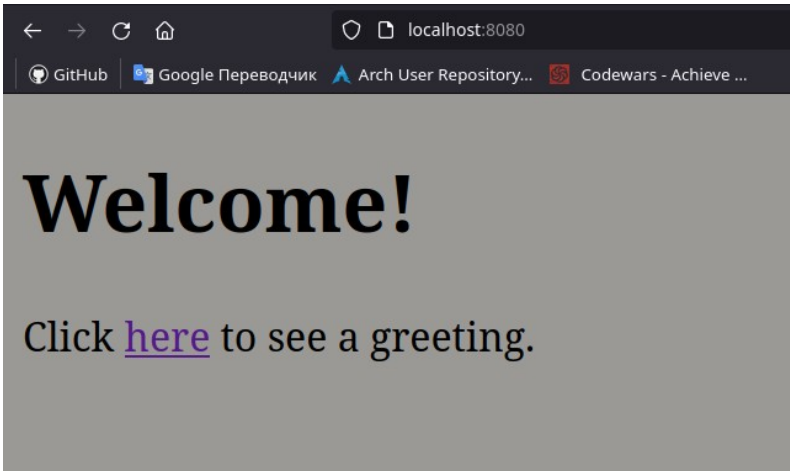
WebSecurityConfigкласс аннотируется @EnableWebSecurity чтобы включить поддержку веб-безопасности Spring Security и обеспечить интеграцию Spring MVC. Он также расширяет WebSecurityConfigurerAdapter и переопределяет несколько своих методов, чтобы установить некоторые особенности конфигурации веб-безопасности.

configure(HttpSecurity)Метод определяет, какие пути URL должны быть защищены, а какие нет. В частности, / и /home пути настроены так, чтобы не требовать аутентификации. Все остальные пути должны быть аутентифицированы.

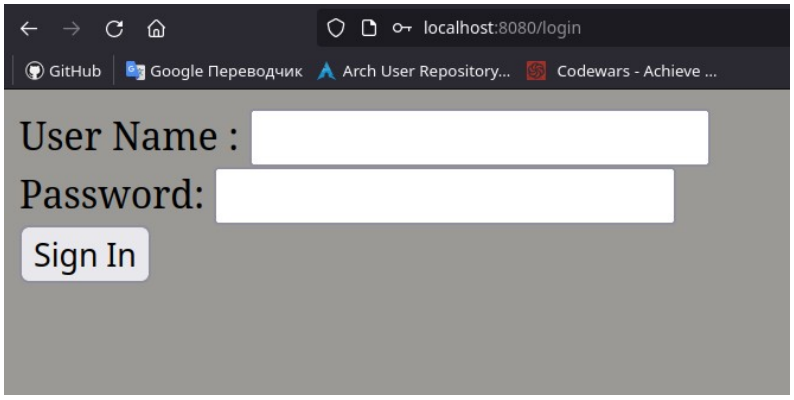
Когда пользователь успешно входит в систему, он перенаправляется на ранее запрошенную страницу, требующую аутентификации. /login страница (которая указана loginPage()), и всем разрешено просматривать его.

userDetailsService()Метод устанавливает пользовательское хранилище в памяти с одним пользователем. Этому пользователю дается имя пользователя user, пароль password, и роль USER.

Запускаем прогамму.
Указываем в браузере http://localhost:8080.



Переходим по сслыке:



Вводим в поле User Name : user. Password : password: