Добавление Spring Security в проект – настройки по умолчанию

В этой статье рассказывается, что будет, если добавить в проект Spring Security – какие настройки включатся по умолчанию.

**Подготовка**

Сгенерируем на <https://start.spring.io/> Spring Boot приложение с зависимостью Web:

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-web</**artifactId**>

</**dependency**>

Напишем в нем единственный REST-контроллер:

@RestController

**public** **class** HelloController {

@GetMapping("/api/hello")

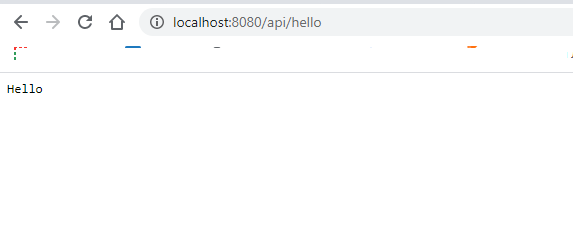
**public** **String** hello(){

**return** "Hello";

}

}

Сейчас к нему имеют доступ все:

Доступный всем контроллер

**Добавление Spring Security**

Для того, чтобы включить Spring Security, достаточно добавить Maven-зависимость:

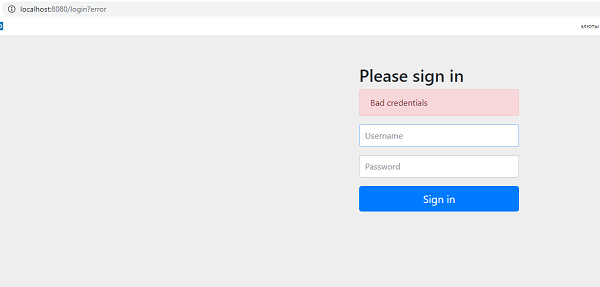
<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-security</**artifactId**>

</**dependency**>

И сразу же мы столкнемся с неожиданностью. Теперь при попытке ввести в браузере *http://localhost:8080/api/hello* мы перенаправляемся на страницу логина *http://localhost:8080/login*. Если ввести в нее любые наугад взятые данные, получим ошибку:

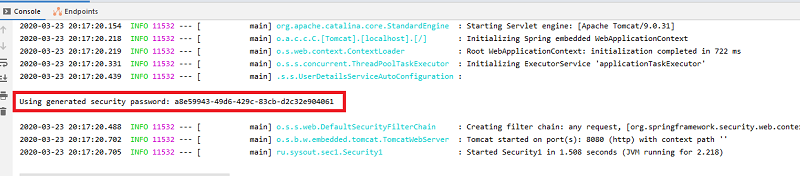


Отсюда очевидно, что некая проверка выполняется. Но какая?

**Что дает зависимость *spring-boot-starter-security***

Обычно включение любого стартера в POM-файл ничего не дает: чтобы что-то запрограммировать, все равно надо написать дополнительный код. В случае Spring Security все иначе.

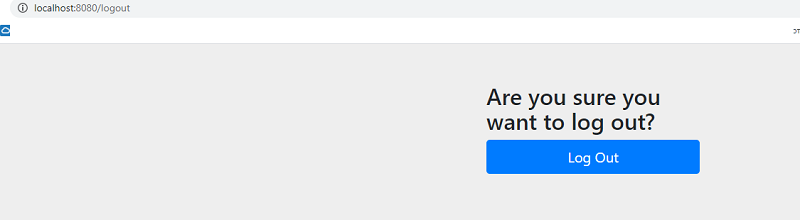
Давайте заглянем в консоль. В ней видно, что генерируется некий пароль:

В консоли выдается сгенерированный пароль

Да, Spring Security создал некоего пользователя по умолчанию. Имя его ***user***, а пароль генерируется автоматически при запуске программы.

Итак, что происходит при одном только добавлении *spring-boot-starter-security* в POM-файл:

* Spring Security создает пользователя с именем *user* и автоматически сгенерированным паролем,  который можно посмотреть в консоли.
* Создается страница с формой для ввода имени и пароля -имеем Form-based аутентификацию.
* Имя и пароль реально проверяются.
* Все url оказываются недоступны, пока мы не “залогинимся” под этим пользователем.
* И еще создается страница, где можно “разлогиниться”. Она находится по адресу *logout*. Выглядит так:

Страница “разлогина”

Кстати, страница входа генерируется в классе *org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter*

**In-Memory аутентификация**

С точки зрения получения параметров пользователя из запроса, продемонстрированная выше аутентификация является Form-Based – имя и пароль отправляются через форму и берутся на сервере из запроса как POST-параметры.

С точки зрения же хранения пользователей на стороне сервера, продемонстрированная выше аутентификация в Spring Security называется*In-Memory authentication*. Она означает, что пользователь хранится не в базе, не на LDAP-сервере и не где-либо еще, а в оперативной памяти приложения до тех пор, пока оно запущено. И чтобы отредактировать пользователя, придется заново запускать приложение. Разумеется, этот вариант не годится для среды Production, зато он прост и полезен для экспериментов во время разработки.

**Как задать своего пользователя в In-Memory аутентификации**

Итак, приложение при запуске генерирует и хранит имя и пароль пользователя в памяти, мы можем подсмотреть пароль в консоли.

Но чтобы не подсматривать пароль в консоли, можно воспользоваться файлом настроек  *application.yml – з*афиксировать имя/пароль там.

**Переопределение пользователя и пароля в настройках**

Для этого в настройках *application.yml*нужно задать свойства:

spring:

security:

user:

name: myname

password: mypassword

Теперь пароль не генерируется и в консоль не выводится – используется пользователь с именем и паролем, заданным в  *application.yml*.

Можно задать и несколько пользователей – давайте сделаем это в коде. Перейдем наконец к написанию кода – напишем класс-конфигурацию для Spring Security и настроим в нем аутентификацию явно.

**Настройка In-Memory аутентификации в коде**

Итак, создадим класс *SecurityConfig*, который расширяет класс *WebSecurityConfigurerAdapter*. Сделаем его бином с помощью *@EnableWebSecurity:*

@EnableWebSecurity

**public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {

...

}

Аннотацию *@EnableWebSecurity* необходимо прописывать при настройке аутентификации, а иначе, как сказано в документации, поведение будет непредсказуемым. Хотя наш пример работает и просто с*@Configuration*.

Аутентификацию выполняет *AuthenticationManager*, но определять этот бин явно нам не надо. Вместо этого надо переопределить метод*configure(AuthenticationManagerBuilder auth)*класса *WebSecurityConfigurerAdapter* – так мы получим доступ к билдеру *AuthenticationManagerBuilder*, а уж через него настроим нужный нам *AuthenticationManager.*Делается это так:

* Во первых, в билдере надо задать тип аутентификации – она может быть не In-Memory, а другой: например, Jdbc, LDAP или кастомной  (тип аутинтификации задает где в принципе хранится пользователь). У нас In-Memory аутентификация – этот факт задается строкой *auth.inMemoryAuthentication().*
* Далее идут специфические настройки выбранного *AuthenticationManager.*В них уточняется, как *AuthenticationManager*извлекает хранимого пользователя, чтоб потом сравнить его с введенным. В случае In-Memory аутентификации менеджеру далеко ходить не надо, реальные имя и пароль задаются тут же с помощью *withUser()* и*password():*

@EnableWebSecurity

**public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {

@Bean

**public** PasswordEncoder passwordEncoder() {

**return** NoOpPasswordEncoder.getInstance();

}

@Override

**public** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.inMemoryAuthentication()

.withUser("u1")

.password("p1")

.authorities("ROLE\_USER")

.and()

.withUser("u2")

.password("p2")

.authorities("ROLE\_USER");

}

}

На самом деле *AuthenticationManager*достает не только реальные имя и пароль, но еще разрешение пользователя (что ему разрешено делать в приложении). Мы задали двух пользователей с разрешением*ROLE\_USER*. В данном примере разрешения не используются, мы будем их использовать в примере про авторизацию.

Итак, мы настроили *AuthenticationManager*, который сравнивает переданные имя и пароль со значениями имени и пароля*u1 p1* и*u2 p2.*В случае совпадения с любым из пользователей, аутентификация проходит успешно.

Обратите внимание на бин *PasswordEncoder –* в нем задается, как шифровать пароль. Мы задали *NoOpPasswordEncoder*, который не делает ничего – оставляет пароль в первоначальном виде. Это выбрано в учебных целях, чтобы было наглядно, что требуется вводить в форму логина при запуске примера – ведь в методе *password(“p2”)* задается уже **зашифрованный**пароль. Конечно, в реальном приложении *NoOpPasswordEncoder*не пригоден – пароль нужно шифровать, например, с помощью *BCryptPasswordEncoder.*

**Итоги**

Таким образом, в примере мы вручную воспроизвели In-Memory аутентификацию, которая вообще-то предоставляется по умолчанию при добавлении security-стартера в проект. Правда, немного видоизменили ее, добавив двух своих пользователей.