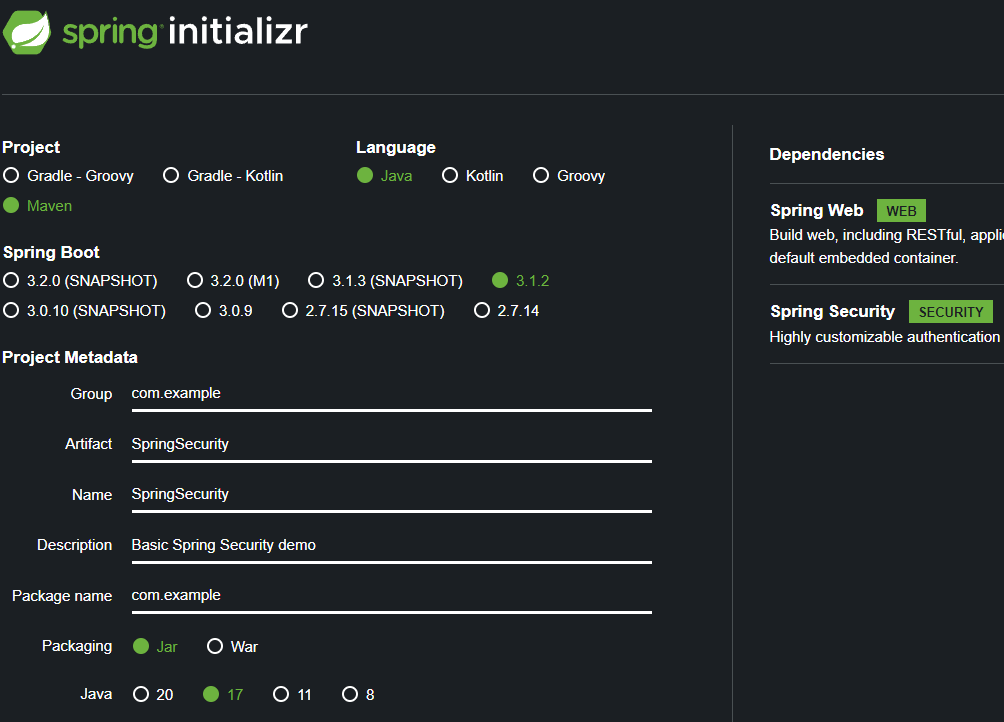
Proyecto Spring Security

1. Vamos a generar el Proyecto desde Spring Initializr (start.spring.io)

Seleccionamos Maven, luego dejamos la version de Springboot que recomienda la web (ya que es la estable).

Lo mas importante es seleccionar siempre la version 17 del JDK de Java ya que es la compatible con Spring

Solo agregamos las dependencias de Spring web y Spring security.



2 – Una vez importado el Proyecto a Eclipse, vamos a comentar las dependencias instaladas de Spring security por el momento

<!--<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> -->

3 – Como agregado estetico vamos a crear en la carpeta resources un archive llamado banner.txt y buscamos en el navegador la web de Spring banner. En la misma escribimos Spring Security y copiamos el resultado en el archive creado

,---. ,--. ,---. ,--. ,--.

' .-' ,---. ,--.--.`--',--,--, ,---. ' .-' ,---. ,---.,--.,--.,--.--.`--',-' '-.,--. ,--.

`. `-.| .-. || .--',--.| \| .-. | `. `-.| .-. :| .--'| || || .--',--.'-. .-' \ ' /

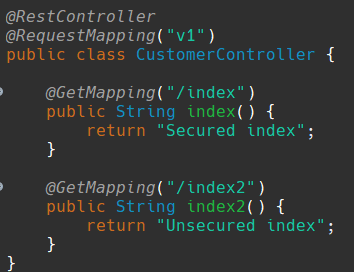
.-' | '-' '| | | || || |' '-' ' .-' \ --.\ `--.' '' '| | | | | | \ '

`-----'| |-' `--' `--'`--''--'.`- / `-----' `----' `---' `----' `--' `--' `--' .-' /

`--' `---' `---'

4 – En la carpeta de src/main > example creamos el paquete “controller”.  
Creamos la clase CustomerController y le agregamos las anotaciones @RestController (esto marca a la clase como un controller y le indica a la app que aca estaran los metodos a ejecutar segun la url a la que se accede) y @RequestMapping(“v1”) (hara que cada metodo responda a una url determinada e indicara el tipo de request que envía al servidor. Tener en cuenta que el “v1” indicara la url base… http://localhost:8080/v1)

Creamos el metodo index():String y por ahora solo retornara un “hola galaxia”. Dicho metodo llevara la anotacion de @GetMapping para mostrar que es una peticion get y la url sera “/index”. (el “/index” es lo que se llama ENDPOINT)



5 – Lo siguiente es Volver al pom.xml y descomentar las dependencias de security, luego sobre la carpeta del Proyecto damos click derecho, en la linea de Maven le damos a update project para que se implementen los cambios. Luego volvemos a correr la app y automaticamente ya tendremos las url encriptadas; cuando querramos ingresar a cualquier endpoint se nos pedira un usuario y una contraseña (por defecto, SpringSecurity trae un usuario que se llama “user” y el password figura en los datos al levaltar la app. Esto abre una session que se mantendra abierta hasta que llamemos a la url /logout; tambien se puede configurar un tiempo de inactividad)

Una forma de configurar un usuario y un password distinto al generico (no es recomendable ya que el hardcodear datos pone en riesgo los datos sensibles) es ir a application.properties y escribir lo siguiente

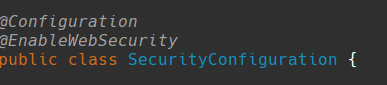
“spring.security.user.name=example”

“spring.security.user.password=example”

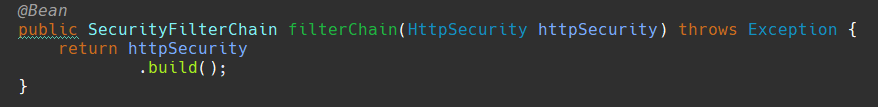


6 – dentro del paquete de example creamos el paquete Configuration, este llevara la configuracion de seguridad ya que al haber bloqueados las url de la app, nadie sin autenticaciones puede acceder a la misma y eso no es lo que queremos.

Dentro del nuevo paquete creamos la clase SecurityConfiguration y le agregamos las anotaciones @Configuration y @EnableSecurityWeb

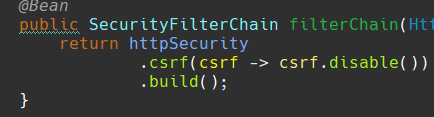


Ahora vamos a crear un metodo public llamado filterChain del tipo SecurityFilterChain (esto es una interfaz) que recibe por parametron una HttpSecurity y la retorna con un .build (dicho metodo debe tener un trhow Exception y una anotacion de @Bean



La httpSecurity es una conexion segura que por defecto viene con varias implementaciones que protegen las vulnerabilidades de la conexion via URL

La primera que vamos a modificar/remover es la “.csrf.disable()” (la cual ahora esta discontinuada, pero por ser el caso de este curso vamos a usar la modificacion que nos permite utilizarla)  
“csrf” significa CROSS-SITE-REFQUEST-FORGERY” y lo que hace es proteger el envio de archivos xml, json y todo tipo de formularios, previniendo que al momento de enviar las peticiones al servidor no se filtren nuestros datos (ya sea nombre de usuario, password u otras informaciones sensibles)  
En este caso vamos a deshabilitar esta proteccion de la siguiente manera



Para lo que es este Proyecto, dicha vulnerabilidad no debe estar desprotegida por lo que volveremos a habilitar la proteccion de formularios (tomar esto como un mero ejempo de vulnerabilidades protegidas por defecto)

Vamos a agregar un filtro para habilitar ciertas urls al publico con el metodo “.authorizedHttpRequest()”.  
Esto nos va a habilitar dos nuevos metodos (puede haber mas pero en este caso usaremos estos dos) “.requestMatchers(/v1/index2).permitAll()” (con el cual podremos ingresar todas las url a las cuales querramos que esten habilitadas para el publico sin log-in) y “anyRequest().authenticated()” (esto restringira el ingreso al resto de las url de la app); por ultimo agregaremos un “.and()” con este metodo podremos concatenar nuevos metodos ya que tanto el requestMatchers como el anyRequest son metodos que vienen de .auto….. y luego del .and() llamaremos a .formLogin().permitAll() (con esto le permitiremos a todos aquellos que quieran ir a dicha url poder utilizar el login para ingresar con autenticaciones a la url deseada)

(ACLARACION: Aparentemente todos estos metodos quedaron deprecados en la v6.1 pero para este curso seran utilizados de como si estuvieran en actividad)

