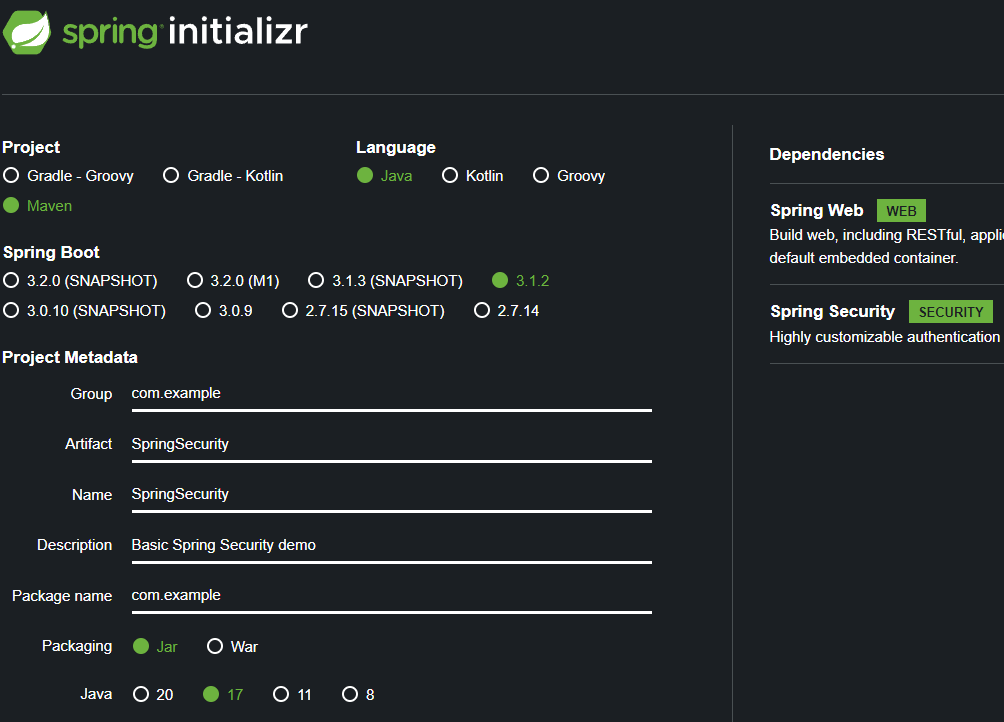
Proyecto Spring Security

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

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

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

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

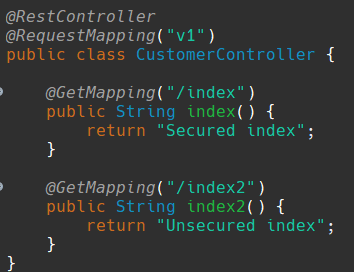
.-' | '-' '| | | || || |' '-' ' .-' \ --.\ `--.' '' '| | | | | | \ '

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

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

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

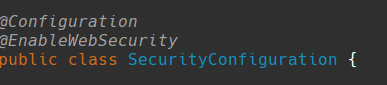
“spring.security.user.name=example”

“spring.security.user.password=example”

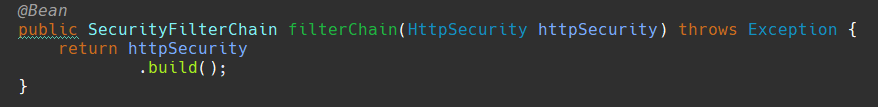


6 – dentro del paquete de example creamos el paquete Configuration, este llevara la configuracion de seguridad ya que al haber bloqueados las url de la app, nadie sin autenticaciones puede acceder a la misma y eso no es lo que queremos.

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

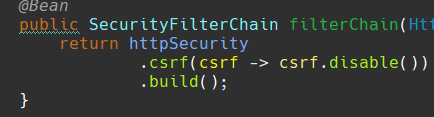


Ahora vamos a crear un metodo public llamado filterChain del tipo SecurityFilterChain (esto es una interfaz) que recibe por parametron una HttpSecurity y la retorna con un .build (dicho metodo debe tener un trhow Exception y una anotacion de @Bean



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)



Otra forma de escribir este método es con expresiones LAMBDA

*@Bean*

public SecurityFilterChain filterChain(HttpSecurity httpSecurity) throws Exception {

return httpSecurity

.authorizeHttpRequests(auth -> {

auth.requestMatchers("/v1/index2").permitAll();

auth.anyRequest().authenticated();

})

.~~formLogin~~().permitAll()

.~~and~~()

.build();

}

\*Otra de las opciones que podemos modificar es el que al loguear, la app nos redireccione:

Creamos un public SuccesHandler (interfaz) que manejara la re direccion en caso de que se acceda satisfactoriamente a la web (osea, se loguee)

public AuthenticationSuccessHandler succesHandler() {

return ((request, response, authentication) -> {

response.sendRedirect("/v1/index");

});

}

Al metodo filterChain() le agregamos el metodo

.formLogin(auth -> {

auth.successHandler(succesHandler());

auth.permitAll();

})

\*Ahora vamos a establecer la forma en que se maneja la creacion de las sesiones . Existen 4 posibilidades

ALWAYS: Siempre que se ingrese al enpoint creara una nueva session

IF\_REQUIRED: Solo creara una session si no existe una en uso, caso contrario, utilizara la existente

NEVER: No creara ninguna session pero si existe una en uno, la utilizara; caso contrario, intentara acceder al ENDPOINT sin session

STATELESS: Intentara acceder a todos los ENDPOINT sin session alguna y retornara los resultados de dicha request

.sessionManagement(auth -> {

auth.sessionCreationPolicy(*SessionCreationPolicy*.***IF\_REQUIRED***);

\*Ahora le vamos a agregar el metodo “invalidSessionUrl(“/login”)” que marcara al ENDPOINT que se enviara al usuario en caso de que el login no sea satisfactorio.  
\*Luego “.maximumSessions(1)” que dictaminara la cantidad de sessions permitidas por cuenta (casi siempre es 1 salvo en las app multi-plataforma)  
\*En caso de desconexion se utilizara el metodo “expiredUrl(“/login”)” que redireccionara al user al login

.sessionManagement(auth -> {

auth.sessionCreationPolicy(*SessionCreationPolicy*.***IF\_REQUIRED***);

auth.invalidSessionUrl("/login");

auth.maximumSessions(1).expiredUrl("/login");

})

\*Otro de los riesgos a proteger es el “sessionFixation()” que permite que una session este logueada por tiempo indeterminado por lo que si el atacante accede a una session, podra tener acceso por siempre a la app. Hay 3 posibilidades para resolver esta vulnerabilidad:

.migrateSession() – Cuando Spring detecta el intento de ataque mediante esta vulnerabilidad, cambia el id de la session; por lo que el atacante cuando intenta usar la session, la misma ya tiene un id invalid (este metodo migra todos los datos del usuario a una nueva session por lo que no se pierde informacion almacenada, solo se modifica el id. Es la configuracion por defecto de Spring)

.newSession() – Difiere de .migrateSession() en que no migra los datos de la session anterior y los mismos se pierden (si no me confundo esto es normalmente visto cuando se aprieta F5 en un formulario)

.none() – Inactiva la seguridad de SpringSecurity por lo que permite al atacante utilizar esta vulnerabilidad (no recomendable)

auth.sessionFixation()

.migrateSession();

\*Ahora, del metodo “auth.maximumSession(1)” vamos a llamar al metodo “sessionRegistry(sessionRegistry)” el cual nos permitira ver los registros de la session active.

auth.maximumSessions(1)

.sessionRegistry(sessionRegistry())

Dentro de la clase Security Configuration creamos el metodo sessionRegistry (es un @Bean) que nos devolvera un SessionRegistry. (en el return va un return SessionRegistryImpl(); )

*@Bean*

public SessionRegistry sessionRegistry() {

return new SessionRegistryImpl();

}

Este metodo sera pasado por parametron al “sessionRegistry” de auth.  
Ahora en el controlador vamos a crear un metodo por el cual podamos ver los detalles requeridos.  
Primero creamos un atributo del tipo SessionRegistry que tendrá la anotacion @Autowired(esto indica que sera inyectada dicha informacion)

Creamos un metodo del tipo ResponseEntity<?> llamado getDetailsSession()  
Instanceamos un String “sessionId = “ ”;” y un User llamado userObject( esta clase la importamos de Spring details) y la igualamos a null  
Luego creamos una lista que contenga objetos llamada sessions y esta obtendra los datos mediante el metodo getAllPrincipalls() de la sessionsRegistry inyectada

List<Object> sessions = this.sessionRegistry.getAllPrincipals();

Ahora recorremos la lista con un foreach y, si el dato session es una instancia de User, igualamos el userObject a la session.

for (Object session : sessions) {

if (session instanceof User) {

userObject = (User) session;

}

Luego creamos una nueva lista que llevara la SessionInformation y obtendra los datos con el metodo getAllSessions(session, false) que se invoca desde el sessionRegistry inyectado (el param session hace referencia a la session active obtenida y el false hace referencia a que no tome las sesiones no activas)

List<SessionInformation> sessionInformations = this.sessionRegistry.getAllSessions(session, false);

Recorremos con un foreach nuevamente la lista de sessionInformations obtenido previamente e igualamos el sessionId a la id obetnida mediante el metodo getSessionId();

for(SessionInformation sessionInformation : sessionInformations) {

sessionId = sessionInformation.getSessionId();

}

Por ultimo creamos un Map<String Object> llamado responde = new HashMap<>; (ya que Spring nos devuelve la info a modo de hashmap)  
y creamos 3 put para mostrar la info en formato Json

Map<String, Object> response = new HashMap<>();

response.put("response", "Hello galaxy");

response.put("sessionId", sessionId);

response.put("sessionUser", userObject);

y ahora retornamos un ResponseEntity.ok(response)

return ResponseEntity.*ok*(response);

Ahora modificamos el metodo de SecurityConfioguration “succesHandler” para que en vez de que nos redireccione al index, nos envie a la url “/session” que nos mostrara la info de la session

public AuthenticationSuccessHandler succesHandler() {

return ((request, response, authentication) -> {

response.sendRedirect("/v1/session");

});

}



\*A modo de mencionar mas info, se puede utilizar el metodo .httpBasic() que nos habilitara a enviar las credenciales via header. Esto significa que si utilziamos postman, podemos enviar el usuario y la contraseña sin complete el formulario, simplemente enviando la info “por header” lo cual no es muy recomendable ya que es informacion sensible y la estariamos enviando de forma sencilla sin seguridad…

.~~httpBasic~~()

En caso de que quisieramos utilizar postman y enviar la info por header sin el httpBasic(), la respuesta sería el html que nos solicita las credenciales (osea la web del formulario de acceso)