



# Tema 4. Tecnologías del servidor. Seguridad con Spring Security

Programación web

Boni García  
Curso 2016/2017

# Índice

---

1. Java EE y Spring
2. Spring MVC y Thymeleaf
3. Bases de datos con Spring
4. Seguridad con Spring Security
5. Pruebas con JUnit y Selenium

# Índice

---

1. Java EE y Spring
2. Spring MVC y Thymeleaf
3. Bases de datos con Spring
4. Seguridad con Spring Security
  - Seguridad en redes de datos
  - Seguridad en aplicaciones web
  - Autenticación y autorización en Spring Security
  - Confidencialidad (HTTPS) en Spring Security
5. Pruebas con JUnit y Selenium

# Seguridad en redes de datos

---

## Servicios de seguridad

- Un **servicio de seguridad** protege las comunicaciones de los usuarios ante determinados ataques. Los principales son:
  - Autenticación (*authentication*): sirve para garantizar que una entidad (persona o máquina) es quien dice ser
  - Autorización (*authorization*): sirve para discernir si una entidad tiene acceso a un recurso determinado
  - Integridad (*data integrity*): garantiza al receptor del mensaje que los datos recibidos coinciden exactamente con los enviados por el emisor
  - Confidencialidad (*data confidentiality*) proporciona protección para evitar que los datos sean revelados a un usuario no autorizado

# Seguridad en redes de datos

---

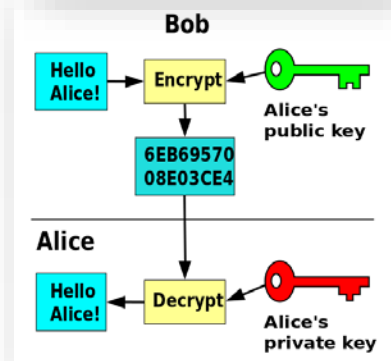
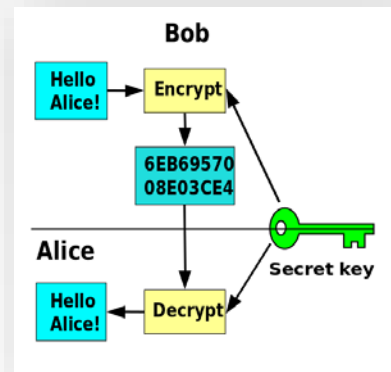
## Servicios de seguridad

- La **autenticación** se consigue mediante:
  - Algo que sabes. Por ejemplo, unas credenciales login-password
  - Algo que tienes. Por ejemplo, una tarjeta de acceso
  - Algo que eres. Por ejemplo, cualidades biométricas (huella digital...)
- La **autorización** discrimina el acceso a un determinado recurso en base a permisos (*grants*), roles de usuario, tokens, ...
  - A veces requiere autenticación previa (es decir, confirmar la identidad)
- La **integridad** se consigue típicamente con funciones Hash (resumen)
  - Son funciones computables mediante un algoritmo que convierte una entrada binaria (típicamente un fichero o un mensaje digital) a un rango de salida finito (típicamente una cadenas alfanumérica)
  - La posibilidad de colisión (diferentes entradas con mismo hash) es muy pequeña

# Seguridad en redes de datos

## Servicios de seguridad

- La **confidencialidad** se consigue típicamente usando técnicas criptográficas (cifrado de mensajes). Tipos de sistemas criptográficos:
  - Criptosistemas de **clave secreta**. En ellos, la clave de cifrado y de descifrado es la misma: es una clave secreta que comparten el emisor y el receptor del mensaje. Debido a esta característica son denominados también criptosistemas **simétricos**
  - Criptosistemas de **clave pública**. Se distinguen porque cada usuario o sistema final dispone de dos claves: una privada, que debe mantener secreta, y una pública, que debe ser conocida por todas las restantes entidades que van a comunicar con ella. Se los conoce también como criptosistemas **asimétricos**



# Seguridad en redes de datos

---

## TLS

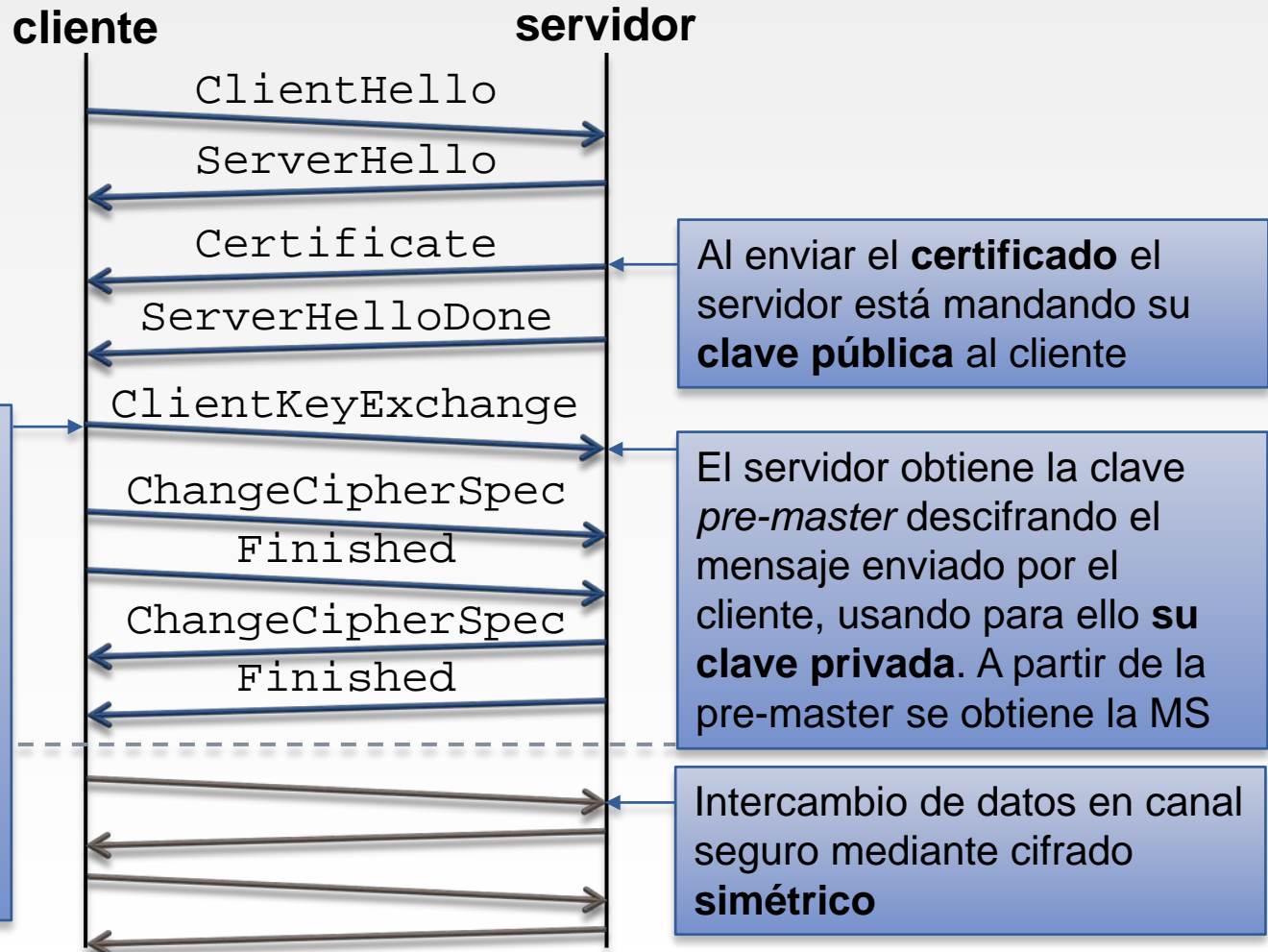
- TLS (*Transport Layer Security*) es un protocolo criptográfico de nivel de transporte que proporciona comunicaciones seguras (cifradas) a una conexión TCP
  - Es la versión evolucionada de SSL (*Secure Sockets Layer*)
  - En octubre de 2014 se descubrió una vulnerabilidad crítica en SSL 3.0 que hace que su uso esté desaconsejado
- Los servicios de seguridad ofrecidos por TLS son confidencialidad, (cifrado el intercambio de datos a nivel de transporte), autenticación (entidades pueden confirmar su identidad), e integridad (mediante una función hash)
- Para establecer un canal seguro cifrado, las entidades tienen que llegar a un acuerdo (***handshake***)

# Seguridad en redes de datos

## TLS

- Handshake TLS en aplicaciones cliente-servidor (por ejemplo en HTTPS)

El cliente genera una clave *pre-master* que será usada generar la clave maestra MS con la que se cifrarán todos los datos de la sesión. Esta clave se envía cifrada con la **clave pública del servidor**, obtenida a partir del certificado



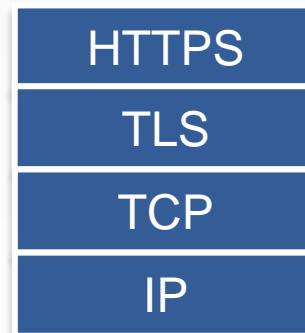


# Seguridad en redes de datos

---

## HTTPS

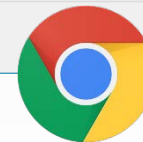
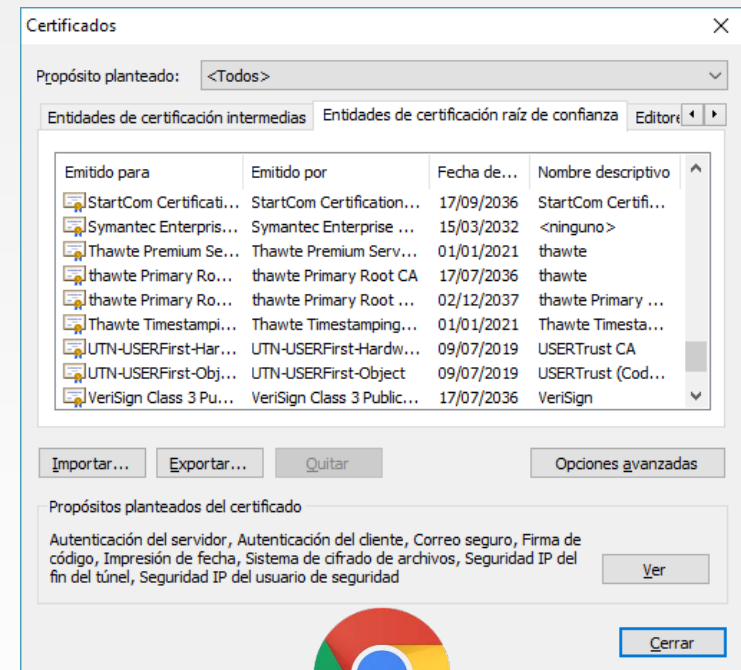
- *Hypertext Transfer Protocol Secure*. Versión segura de HTTP
- HTTPS no es más que HTTP sobre TLS
- Con HTTPS se consigue que la información sensible (claves, etc) no pueda ser interceptada por un atacante, ya que lo único que obtendrá será un flujo de datos cifrados que le resultará imposible de descifrar
- Puerto TCP por defecto en servidores HTTPS: 443



# Seguridad en redes de datos

## HTTPS

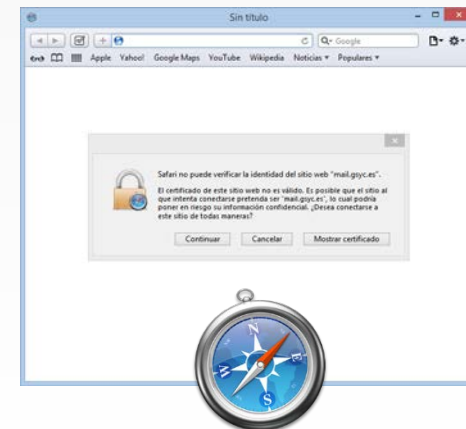
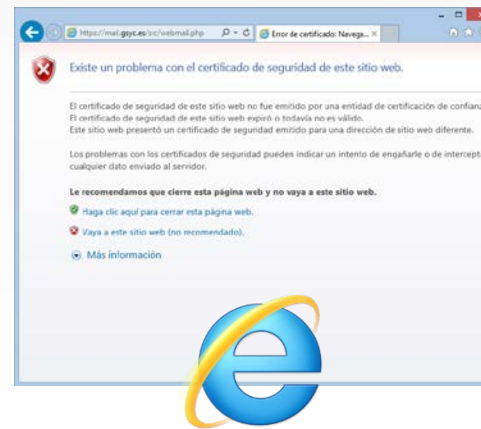
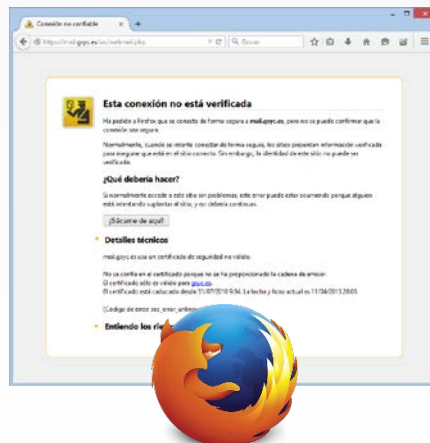
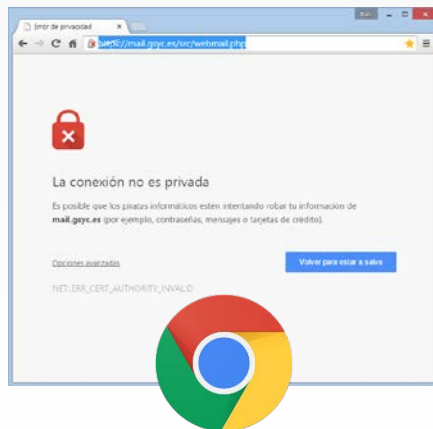
- Un **certificado** es emitidos por Autoridad de Certificación (CA)
  - Ejemplos: FNMT, GlobaSign, Symantec, Verisign, ...
- Los navegadores tienen una lista de CAs conocidas



# Seguridad en redes de datos

## HTTPS

- Al recibir un certificado emitido por una CA desconocida, el navegador muestra una alerta de seguridad
- Por ejemplo, al recibir un certificado autofirmado
  - Certificado firmado por la propia entidad cuya identidad se quiere autenticar
  - Se puede generar por ejemplo con OpenSSL o KeyTool (para Java)



# Seguridad en aplicaciones web

---

## Vulnerabilidades web

- Algunas de las **vulnerabilidades** web más importantes son:
  - *Injection*: Método de infiltración de código malicioso, por ejemplo SQL en el lado servidor
  - *Cross-Site Scripting* (XSS): Inyección de código (típicamente JavaScript) malicioso en la lado cliente (por ejemplo para robar cookies)
  - *Cross Site Request Forgery* (CSRF): Falsificación de petición. Ocurre cuando a un usuario legítimo (típicamente con sesión en una web) se le fuerza a hacer una petición no deseada (por ejemplo cambiar el password)
- **OWASP** (*Open Web Application Security Project*) es una organización sin ánimo de lucro dedicada la seguridad de aplicaciones web
  - Tutoriales, herramienta, metodologías, ...

[https://www.owasp.org/index.php/Top\\_10\\_2013-Top\\_10](https://www.owasp.org/index.php/Top_10_2013-Top_10)



# Seguridad en aplicaciones web

---

## Política del mismo origen

- La **política del mismo origen** (*same-origin policy*) es una medida de seguridad implementada en los navegadores para evitar cargar scripts cuyo *origen* sea diferente
  - Se entiende origen como la parte inicial de la URL (combinación de protocolo + host + puerto)
- A veces esta política es demasiado restrictiva, y se puede relajar activando lo que se conoce como **CORS** (*Cross-origin Resource Sharing*) en los servidores web
  - Es un estándar del W3C que permite saltarse la política del mismo origen usando ciertas cabeceras HTTP (`Access-Control-Allow-Origin`)

<https://www.w3.org/TR/cors/>

# Autenticación y autorización en Spring Security

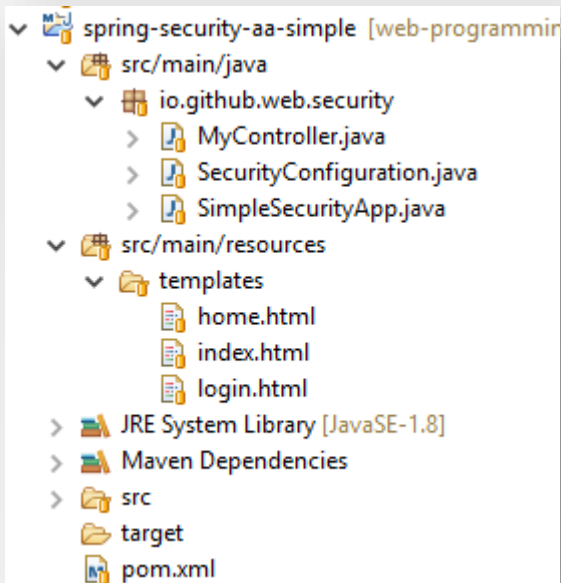
---

- Vamos a ver como implementar los servicios de seguridad de autenticación y autorización en Spring Boot + Security estudiando tres ejemplos:
  1. Sencillo (proyecto `spring-security-aa-simple`)
    - Usuarios en memoria, rol único
  2. Medio (proyecto `spring-security-aa-medium`)
    - Usuarios en memoria, varios roles
  3. Avanzado (proyecto `spring-security-aa-advanced`)
    - Usuarios en base de datos, varios roles

# Autenticación y autorización en Spring Security

## Ejemplo AA sencillo: proyecto spring-security-aa-simple

Fork me on GitHub



```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.2.RELEASE</version>
</parent>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
</dependencies>
```

# Autenticación y autorización en Spring Security

## Ejemplo AA sencillo: proyecto `spring-security-aa-simple`

```
@Controller
public class MyController {

    @RequestMapping("/")
    public ModelAndView index() {
        return new ModelAndView("index");
    }

    @RequestMapping("/login")
    public ModelAndView login() {
        return new ModelAndView("login");
    }

    @RequestMapping("/home")
    public ModelAndView home() {
        return new ModelAndView("home");
    }
}
```

Controlador muy  
sencillo: sólo asocia  
URLs con vistas



# Autenticación y autorización en Spring Security

## Ejemplo AA sencillo: proyecto spring-security-aa-simple

```
@Configuration
@EnableGlobalMethodSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // Paths that can be visited without authentication
        http.authorizeRequests().antMatchers("/").permitAll();
        http.authorizeRequests().antMatchers("/login").permitAll();
        http.authorizeRequests().antMatchers("/logout").permitAll();

        // Paths that cannot be visited without authentication
        http.authorizeRequests().anyRequest().authenticated();

        // Login form
        http.formLogin().loginPage("/login");
        http.formLogin().usernameParameter("username");
        http.formLogin().passwordParameter("password");
        http.formLogin().defaultSuccessUrl("/home");
        http.formLogin().failureUrl("/login?error");

        // Logout
        http.logout().logoutUrl("/logout");
        http.logout().logoutSuccessUrl("/login?logout");
    }
    // ...
}
```

← Páginas que tendrán el acceso permitido

← El resto de páginas requerirán de autenticación

← Autenticación basada en formulario

← Página para la desconexión

# Autenticación y autorización en Spring Security

## Ejemplo AA sencillo: proyecto spring-security-aa-simple

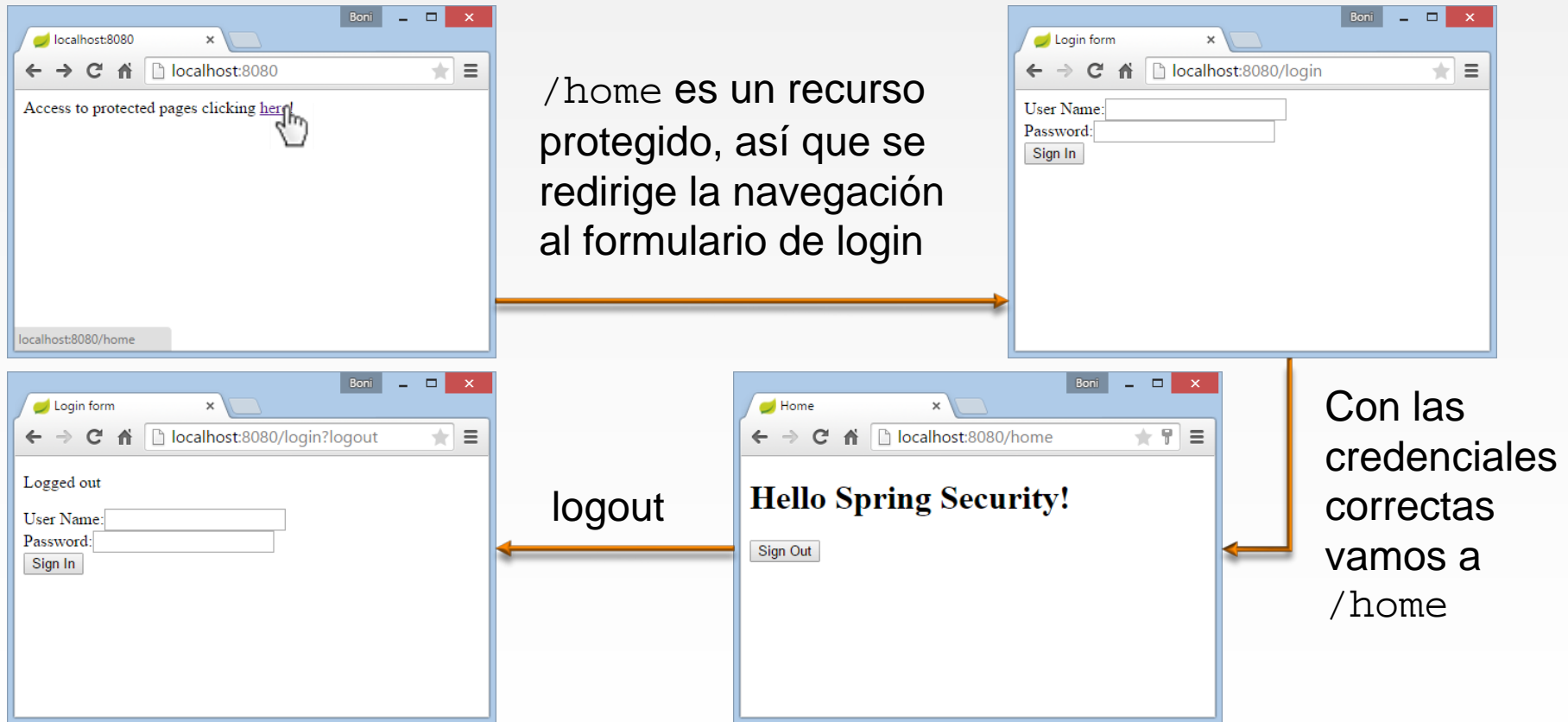
```
// ...

@Override
protected void configure(AuthenticationManagerBuilder auth)
    throws Exception {
    // Authorization
    auth.inMemoryAuthentication().withUser("user").password("p1")
        .roles("USER");
}
}
```

Un único usuario (en memoria)

# Autenticación y autorización en Spring Security

## Ejemplo AA sencillo: proyecto `spring-security-aa-simple`



# Autenticación y autorización en Spring Security

## Ejemplo AA sencillo: proyecto `spring-security-aa-simple`

- Todas las vistas incorporan una medida de seguridad automática: un token para evitar ataques CSRF (*Cross Site Request Forgery*)
- Este token lo genera el servidor para cada petición y es requerido para poder recibir datos del cliente (hay que usar la etiqueta `th:action`)

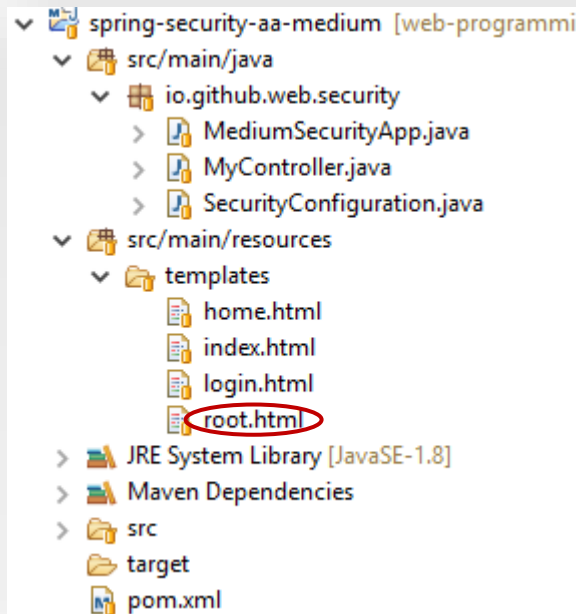
```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>Home</title>
</head>
<body>
<h1>Hello Spring Security!</h1>
<form th:action="@{/logout}" method="post">
<input type="submit" value="Sign Out" />
</form>
</body>
</html>
```



```
<!DOCTYPE html>
<html>
<head>
<title>Home</title>
</head>
<body>
<h1>Hello Spring Security!</h1>
<form method="post" action="/logout">
<input type="submit" value="Sign Out" />
<input type="hidden" name="_csrf" value="c54a70a7-
1586-4dc3-8e64-4fac09625ce2" /></form>
</body>
</html>
```

# Autenticación y autorización en Spring Security

Ejemplo AA medio: proyecto `spring-security-aa-medium`



Proyecto con la misma estructura salvo que tiene una vista más

# Autenticación y autorización en Spring Security

## Ejemplo AA medio: proyecto `spring-security-aa-medium`

Cambiamos la anotación que define el método de seguridad para poder restringir la autorización de los métodos controladores a ciertos roles de usuario

```
@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    // Same authentication schema than example before

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {
        // Authorization
        auth.inMemoryAuthentication().withUser("user").password("p1")
            .roles("USER");
        auth.inMemoryAuthentication().withUser("root").password("p2")
            .roles("USER", "ADMIN");
    }
}
```

Dos usuarios  
en memoria de  
diferente tipo  
(rol)

# Autenticación y autorización en Spring Security

## Ejemplo AA medio: proyecto spring-security-aa-medium

```
@Controller
public class MyController {

    @RequestMapping("/")
    public ModelAndView index() {
        return new ModelAndView("index");
    }

    @RequestMapping("/login")
    public ModelAndView login() {
        return new ModelAndView("login");
    }

    @Secured({ "ROLE_USER", "ROLE_ADMIN" })
    @RequestMapping("/home")
    public ModelAndView home() {
        return new ModelAndView("home");
    }

    @Secured("ROLE_ADMIN")
    @RequestMapping("/root")
    public ModelAndView root() {
        return new ModelAndView("root");
    }
}
```

Los métodos protegidos se anotan con `@Secured` y el nombre del rol (con prefijo `ROLE_`)

# Autenticación y autorización en Spring Security

## Ejemplo AA medio: proyecto spring-security-aa-medium

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Home</title>
</head>
<body>
  <h1>
    Hello <span sec:authentication="name">user</span>
  </h1>

  <div sec:authorize="hasRole('ROLE_ADMIN')">
    <a th:href="@{/root}">Administration page</a>
  </div>

  <form th:action="@{/logout}" method="post">
    <input type="submit" value="Sign Out" />
  </form>
</body>
</html>
```

Podemos acceder a las propiedades de autenticación y autorización mediante las etiquetas Thymeleaf:

- `sec:authentication`
- `sec:authorize`

Para ello es necesario añadir la siguiente dependencia en el pom.xml

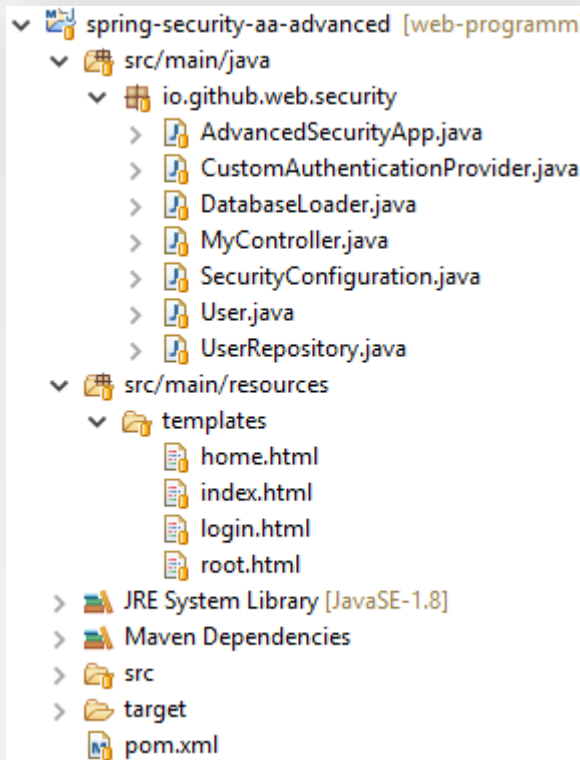
```
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity4</artifactId>
</dependency>
```



# Autenticación y autorización en Spring Security

Ejemplo AA avanzado: proyecto spring-security-aa-advanced

Fork me on GitHub



```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity4</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
  </dependency>
</dependencies>
```

# Autenticación y autorización en Spring Security

## Ejemplo AA avanzado: proyecto spring-security-aa-advanced

```
@Entity
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private String user;
    private String password;

    @ElementCollection(fetch = FetchType.EAGER)
    private List<GrantedAuthority> roles;

    public User() {
    }
    public User(String user, String password, List<GrantedAuthority> roles) {
        this.user = user;
        this.password = new BCryptPasswordEncoder().encode(password);
        this.roles = roles;
    }

    // getters, setters
}
```

Entidad persistente  
que almacenará las  
credenciales de  
usuario y sus roles

Las contraseñas  
nunca se deben  
almacenar en claro  
(hay que cifrarlo o  
usar función hash)

# Autenticación y autorización en Spring Security

## Ejemplo AA avanzado: proyecto spring-security-aa-advanced

```
@Component
public class DatabaseLoader {

    @Autowired
    private UserRepository userRepository;

    @PostConstruct
    private void initDatabase() {
        // User #1: "user", with password "p1" and role "USER"
        GrantedAuthority[] userRoles = { new SimpleGrantedAuthority("ROLE_USER") };
        userRepository.save(new User("user", "p1", Arrays.asList(userRoles)));

        // User #2: "root", with password "p2" and roles "USER" and "ADMIN"
        GrantedAuthority[] adminRoles = { new SimpleGrantedAuthority("ROLE_USER"),
            new SimpleGrantedAuthority("ROLE_ADMIN") };
        userRepository.save(new User("root", "p2", Arrays.asList(adminRoles)));
    }
}
```

Componente usado  
para popular la base  
de datos (se  
ejecutará al iniciar la  
aplicación)

# Autenticación y autorización en Spring Security

## Ejemplo AA avanzado: proyecto spring-security-aa-advanced

```
public interface UserRepository extends CrudRepository<User, Long> {  
    User findByUser(String user);  
}
```

Repositorio de  
usuarios

```
@Configuration  
@EnableGlobalMethodSecurity(securedEnabled = true)  
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {  
  
    @Autowired  
    public CustomAuthenticationProvider authenticationProvider;  
  
    // Same authentication schema than example before  
  
    @Override  
    protected void configure(AuthenticationManagerBuilder auth)  
        throws Exception {  
        // Database authentication provider  
        auth.authenticationProvider(authenticationProvider);  
    }  
}
```

El gestor de  
autenticación ya no  
son credenciales en  
memoria

# Autenticación y autorización en Spring Security

## Ejemplo AA avanzado: proyecto spring-security-aa-advanced

```
@Component
public class CustomAuthenticationProvider implements AuthenticationProvider {

    @Autowired
    private UserRepository userRepository;

    @Override
    public Authentication authenticate(Authentication authentication)
        throws AuthenticationException {
        String username = authentication.getName();
        String password = (String) authentication.getCredentials();
        User user = userRepository.findByUser(username);
        if (user == null) {
            throw new BadCredentialsException("User not found");
        }
        if (!new BCryptPasswordEncoder().matches(password, user.getPasswordHash())) {
            throw new BadCredentialsException("Wrong password");
        }
        List<GrantedAuthority> roles = user.getRoles();
        return new UsernamePasswordAuthenticationToken(username, password, roles);
    }
}
```

Se inyecta repositorio de usuario

Lectura de credenciales del formulario

Se comprueba usuario y contraseña

Lectura de lista de roles

# Confidencialidad (HTTPS) en Spring Security

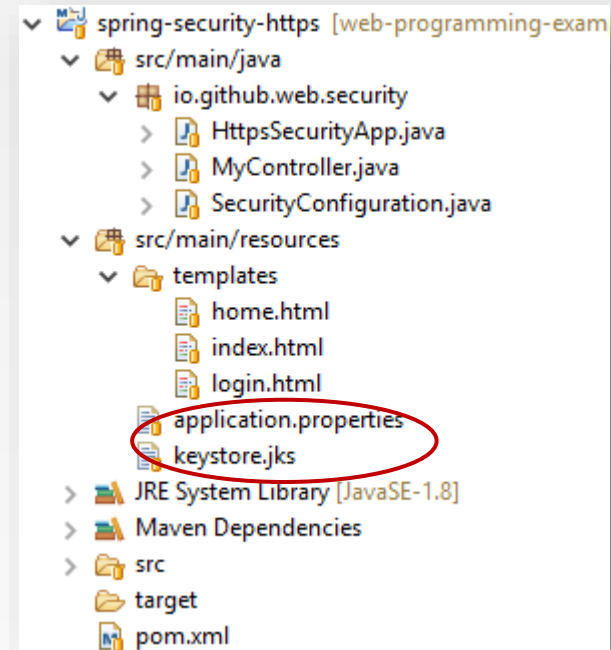
Ejemplo: proyecto `spring-security-https`

- Exactamente igual que proyecto `spring-security-aa-simple` excepto:

- `application.properties`:

```
server.port = 8443  
server.ssl.key-store = classpath:keystore.jks  
server.ssl.key-store-password = password  
server.ssl.key-password = secret
```

- `keystore.jks`: Repositorio de certificados Java



# Confidencialidad (HTTPS) en Spring Security

## Ejemplo: proyecto `spring-security-https`

- `keystore.jks` se crea con herramienta `keytool` (incorporada en JRE)

```
$ cd $JAVA_HOME/bin
$ keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass
password -validity 360 -keysize 2048
¿Cuáles son su nombre y su apellido?
[Unknown]: Boni Garcia
¿Cuál es el nombre de su unidad de organización?
[Unknown]: Web Programming
¿Cuál es el nombre de su organización?
[Unknown]: GitHub
¿Cuál es el nombre de su ciudad o localidad?
[Unknown]: Madrid
¿Cuál es el nombre de su estado o provincia?
[Unknown]: Madrid
¿Cuál es el código de país de dos letras de la unidad?
[Unknown]: ES
¿Es correcto CN=Boni Garcia, OU=Programacion Web, O=GitHub, L=Madrid, ST=Madrid,
C=ES?
[no]: si
Introduzca la contraseña de clave para <selfsigned>
(INTRO si es la misma contraseña que la del almacén de claves): secret
Volver a escribir la contraseña nueva: secret
```

# Confidencialidad (HTTPS) en Spring Security

Ejemplo: proyecto `spring-security-https`

