Puedes encontrar toda la documentación del proyecto Spring Security en **https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/**

Puedes encontrar toda la documentación del proyecto Spring Security en https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/

Autenticación y autorización
- La seguridad de una aplicación suele reducirse a dos problemas más o menos independientes
  - Autenticación: ¿quién es usted?
  - Autorización: ¿qué se le permite hacer?

Las preguntan quien eres y que permisos tienes

Usa patrón de diseño Estrategia

¿Qué es un `Principal`?

Según la wikipedia), un *Principal*, en el ámbito de la seguridad informática, es una entidad (persona, grupo, servicio, proceso, ...) que puede ser autenticado en un sistema informático o red. En la literatura de Java y Microsoft suele encontrarse como *security principal*.

En Java, viene representado por la interfaz java.security.Principal.

¿Y un `Authentication`?

En Spring Security, la interfaz org.springframework.security.core.Authentication, que es nombrada en repetidas ocasiones durante esta lección, es una subinterfaz de `Principal`, proporcionando además de la información de esta, otra referente a *credenciales*, *authorities* y mecanismos para identificar si está autenticado o no.

En el mecanismo de autenticación, se utiliza en el método `AuthenticationManager.authenticate(Authentication)`.

A continuación podemos ver el listado de enlaces a la documentación de las clases, interfaces y anotaciones que se explican en esta lección:

- `WebSecurityConfigurerAdapter`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/web/configuration/WebSecurityConfigurerAdapter.html

- `@EnableWebSecurity`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/web/configuration/EnableWebSecurity.html
- `Authentication`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/Authentication.html
- `AuthenticationManagerBuilder`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/authentication/builders/AuthenticationManagerBuilder.html
- `UserDetails`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetails.html
- `User`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/User.html
- `GrantedAuthority`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/GrantedAuthority.html
- `SimpleGrantedAuthority`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/authority/SimpleGrantedAuthority.html
- `UserDetailsService`: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetailsService.html

@EnableWebSecurity
● Sirve para conmutar (apagar) la configuración por defecto aplicada por Spring Boot, y añadir la nuestra.
● Se utiliza anotando una clase que extienda a WebSecurityConfigurerAdapter.

Posibles implementaciones!..

Si quieres saber más sobre los diferentes esquemas de autenticación, puedes visitar la siguiente documentación:

Autenticación básica

Puedes leer algunos RFCs

- **RFC 1945: Hypertext Transfer Protocol**
- **RFC 2617: HTTP Authentication: Basic and Digest Access Authentication**

Autenticación con JWT

JWT no es en sí un mecanismo de autenticación, aunque veremos que utilizando este estándar para la gestión de token podemos arreglárnoslas para poder hacer dicho proceso. Puedes leer más en:

- **RFC 7519: JSON Web Token (JWT)**
- **RFC 7797: JSON Web Signature (JWS) Unencoded Payload Option**

OAuth 2.0

Puedes leer algunos RFCs

- **RFC 6749: The OAuth 2.0 Authorization Framework**
- **RFC 8252: OAuth 2.0 for Native Apps**
- **RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage**

También puedes encontrar más información en la web **https://oauth.net/2/**

Gestion de Usuarios

Seguridad Basica

Estas podrían ser algunas pruebas realizadas sobre nuestra API Rest con cURL

*Si queremos codificar el `username:password` en base64, lo podemos hacer en alguna web como **https://www.base64encode.org/***

Petición GET para obtener los productos

Nos autenticamos como `marialopez:Marialopez1`, que en Base64 equivale a `bWFyaWFsb3BlejpNYXJpYWxvcGV6MQ==`

Al encauzar la salida de curl a `jq .` conseguimos que esta aparezca prettyficada.

```
curl -v -H "Authorization: Basic bWFyaWFsb3BlejpNYXJpYWxvcGV6MQ==" http://localhost:8080/producto/ | jq .
```

La salida debe ser algo así como:

```
{
  "content": [
    {
      "id": 1,
      "nombre": "Juice - Orange, Concentrate",
      "imagen": "http://dummyimage.com/139x103.bmp/5fa2dd/ffffff",
      "precio": 91,
```

```
    "categoria": "Bebida"
  },
  {
    "id": 2,
    "nombre": "Beef - Ground, Extra Lean, Fresh",
    "imagen": "http://dummyimage.com/206x125.bmp/cc0000/ffffff",
    "precio": 87,
    "categoria": "Comida"
  },
  {
    "id": 3,
    "nombre": "Cheese - Parmesan Grated",
    "imagen": "http://dummyimage.com/133x134.bmp/dddddd/000000",
    "precio": 39,
    "categoria": "Comida"
  },
  {
    "id": 4,
    "nombre": "Cups 10oz Trans",
    "imagen": "http://dummyimage.com/245x246.jpg/dddddd/000000",
    "precio": 67,
    "categoria": "Comida"
  },
  {
    "id": 5,
    "nombre": "Wine - Beringer Founders Estate",
    "imagen": "http://dummyimage.com/139x103.bmp/5fa2dd/ffffff",
    "precio": 27,
    "categoria": "Bebida"
  },
  {
    "id": 6,
    "nombre": "Bread - Wheat Baguette",
    "imagen": "http://dummyimage.com/206x125.bmp/cc0000/ffffff",
    "precio": 82,
    "categoria": "Bebida"
  },
  {
    "id": 7,
    "nombre": "Un producto nuevo",
    "imagen": "http://dummyimage.com/133x134.bmp/dddddd/000000",
    "precio": 123.4,
    "categoria": "Bebida"
  },
  {
    "id": 8,
    "nombre": "Cheese - Mascarpone",
    "imagen": "http://dummyimage.com/245x246.jpg/dddddd/000000",
    "precio": 97,
    "categoria": "Bebida"
  },
  {
    "id": 9,
    "nombre": "Mace",
    "imagen": "http://dummyimage.com/139x103.bmp/5fa2dd/ffffff",
    "precio": 25,
    "categoria": "Bebida"
  },
  {
    "id": 10,
    "nombre": "Oil - Shortening - All - Purpose",
    "imagen": "http://dummyimage.com/206x125.bmp/cc0000/ffffff",
```

```
    "precio": 63,
    "categoria": "Bebida"
  },
  {
    "id": 11,
    "nombre": "Marjoram - Fresh",
    "imagen": "http://dummyimage.com/133x134.bmp/dddddd/000000",
    "precio": 60,
    "categoria": "Bebida"
  },
  {
    "id": 12,
    "nombre": "Turnip - White",
    "imagen": "http://dummyimage.com/245x246.jpg/dddddd/000000",
    "precio": 74,
    "categoria": "Bebida"
  },
  {
    "id": 13,
    "nombre": "Pork Salted Bellies",
    "imagen": "http://dummyimage.com/139x103.bmp/5fa2dd/ffffff",
    "precio": 38,
    "categoria": "Bebida"
  },
  {
    "id": 14,
    "nombre": "Longos - Greek Salad",
    "imagen": "http://dummyimage.com/206x125.bmp/cc0000/ffffff",
    "precio": 15,
    "categoria": "Bebida"
  },
  {
    "id": 15,
    "nombre": "Amaretto",
    "imagen": "http://dummyimage.com/133x134.bmp/dddddd/000000",
    "precio": 85,
    "categoria": "Bebida"
  },
  {
    "id": 16,
    "nombre": "Godiva White Chocolate",
    "imagen": "http://dummyimage.com/245x246.jpg/dddddd/000000",
    "precio": 97,
    "categoria": "Bebida"
  },
  {
    "id": 17,
    "nombre": "Tomatoes - Roma",
    "imagen": "http://dummyimage.com/139x103.bmp/5fa2dd/ffffff",
    "precio": 61,
    "categoria": "Bebida"
  },
  {
    "id": 18,
    "nombre": "Oven Mitt - 13 Inch",
    "imagen": "http://dummyimage.com/206x125.bmp/cc0000/ffffff",
    "precio": 1,
    "categoria": "Complementos"
  },
  {
    "id": 19,
    "nombre": "Vermouth - White, Cinzano",
```

```
      "imagen": "http://dummyimage.com/133x134.bmp/dddddd/000000",
      "precio": 72,
      "categoria": "Bebida"
    },
    {
      "id": 20,
      "nombre": "Club Soda - Schweppes, 355 Ml",
      "imagen": "http://dummyimage.com/245x246.jpg/dddddd/000000",
      "precio": 38,
      "categoria": "Bebida"
    }
  ],
  "pageable": {
    "sort": {
      "sorted": false,
      "unsorted": true,
      "empty": true
    },
    "pageSize": 20,
    "pageNumber": 0,
    "offset": 0,
    "paged": true,
    "unpaged": false
  },
  "last": false,
  "totalPages": 2,
  "totalElements": 30,
  "first": true,
  "sort": {
    "sorted": false,
    "unsorted": true,
    "empty": true
  },
  "numberOfElements": 20,
  "size": 20,
  "number": 0,
  "empty": false
}
```


Petición GET para obtener los pedidos de un usuario

Nos volvemos a autenticar como `marialopez:Marialopez1`, que en Base64 equivale a `bWFyaWFsb3BlejpNYXJpYWxvcGV6MQ==`

```
curl -v -H "Authorization: Basic bWFyaWFsb3BlejpNYXJpYWxvcGV6MQ==" http://localhost:8080/pedido/ | jq .
```

Petición GET para obtener todos los pedidos (como ADMIN)

Un administrador si será capaz de obtener todos los pedidos, independientemente del usuario que los haya realizado. Ejecutamos la misma petición, autenticados como un administrador.

Para autenticarnos como administrador, lo hacemos con `admin:Admin1`, que en Base64 equivale a `YWRtaW46QWRtaW4x`

```
curl -v -H "Authorization: Basic YWRtaW46QWRtaW4x" http://localhost:8080/pedido/ | jq .
```

Petición PUT para modificar un producto

Si en esta petición nos autenticamos como `marialopez:Marialopez1`, obtenedremos un error `403 Forbidden`.

Creamos un fichero json con el siguiente contenido (o uno parecido):

```
{
    "nombre": "Producto nuevo",
    "precio": 12,
    "categoriaId": 1
}
```

Lo más fácil es ejecutar este comando desde la ubicación del archivo

```
curl -X PUT -H "Authorization: Basic YWRtaW46QWRtaW4x" -H "Content-Type:
application/json" -d "@modificaproducto.json" http://localhost:8080/producto/7 |
jq .
```

JWT

Puedes leer la información más completa sobre la especificación JWT aquí: **https://tools.ietf.org/html/rfc7519**.

Para depurar tus tokens JWT, puedes utilizar la siguiente herramienta: **https://jwt.io/#debugger-io**.

Las dependencias a incluir para trabajar con JWT es:

```
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.10.7</version>
    <scope>runtime</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.10.7</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.10.7</version>
    <scope>runtime</scope>
</dependency>
```

Puedes encontrar una documentación más extensa sobre esta librería en **https://github.com/jwtk/jjwt**. Tal y como indica dicha documentación, la mayor parte de la complejidad se oculta detrás de una interfaz fluida basada en un generador. Valga el siguiente ejemplo:

```
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;
import java.security.Key;

// We need a signing key, so we'll create one just for this example. Usually
// the key would be read from your application configuration instead.
Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);

String jws = Jwts.builder().setSubject("Joe").signWith(key).compact();
```

El resultado parecerá algo así:

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKb2UifQ.1KP0SsvENi7Uz1oQc07aXTL7kpQG5jBNIybqr60A
lD4
```

Token `signWith`

La librería JJWT ha cambiado en sus últimas versiones, sobre todo con respecto a firmar el token.

Con el siguiente código, podemos generar una clave aleatoria:

```
Key key = Keys.secretKeyFor(SignatureAlgorithm.HS256);
String jws = Jwts.builder().setSubject("Joe").signWith(key).compact();
```

Según la documentación, podemos la librería puede encargarse de determinar el mejor algoritmo en función de la longitud en bytes de la misma. Por ejemplo, si llamamos a `signWith` con una `SecretKey` de 32 bytes, no será suficientemente fuerte para el algoritmo `HS384` o `HS512`, por lo que la librería aplicará el algoritmo `HS256`. Además, cuando se firma, automáticamente se añade el algoritmo en el parámetro `alg` en el encabezado.

El método utilizado en el ejemplo es `HS512`, porque la longitud de la clave lo permite:

```
return Jwts.builder.signWith(Keys.hmacShaKeyFor(jwtSecreto.getBytes()),
SignatureAlgorithm.HS512)...
```

El método `hmacShaKeyFor` crea una nueva `SecretKey` para usar con algoritmos basados en HMAC-SHA, recibiendo la clave como un array de bytes.

Descifrado del token

Podemos descifrar el token aplicamos de nuevo la clave secreta, y a partir de ahí podemos obtener el *claim* que necesitemos:

```
public Long getUserIdFromJWT(String token) {
    Claims claims = Jwts.parser()
```

```
                        .setSigningKey(Keys.hmacShaKeyFor(jwtSecreto.getBytes()))
                        .parseClaimsJws(token)
                        .getBody();

    return Long.parseLong(claims.getSubject());
    }
```

Según la documentación de Spring Security, para realizar la autenticación basada en *usuario* y *contraseña*, podemos utilizar una instancia de `UsernamePasswordAuthenticationToken`. Se trata de una implementación de `Authentication`.

Si la autenticación se realiza correctamente (en nuestro caso, si el token es validado), debemos almacenar en el contexto de seguridad una instancia de dicha clase:

```
if (StringUtils.hasText(token) && tokenProvider.validateToken(token)) {
    Long userId = tokenProvider.getUserIdFromJWT(token);

    UserEntity user = (UserEntity) userDetailsService.loadUserById(userId);
    UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(user, user.getRoles(),
user.getAuthorities());
    authentication.setDetails(new WebAuthenticationDetails(request));
    SecurityContextHolder.getContext().setAuthentication(authentication);

}
```

La documentación oficial sobre OAuth 2.0 la podemos encontrar en **https://oauth.net/2/**.

En ella, se indican algunos RFCs útiles sobre dicho framework:

- RFC 6749: OAuth 2.0 Framework **https://tools.ietf.org/html/rfc6749**
- RFC 6750: Bearer Tokens **https://oauth.net/2/bearer-tokens/**
- RFC 6819: Threat Model and Security Considerations **https://oauth.net/2/security-considerations/**

Sobre Tokens y su gestión

- RFC 7662: Token Introspection **https://oauth.net/2/token-introspection/**
- RFC 7009: Token revocation **https://oauth.net/2/token-revocation/**
- RFC 7519: Json Web token **https://oauth.net/2/jwt/**

La dependencia necesaria para implementar nuestro servidor de autenticación es la siguiente:

```
<dependency>
    <groupId>org.springframework.security.oauth.boot</groupId>
<artifactId>spring-security-oauth2-autoconfigure</artifactId>
    <version>2.X.Y.RELEASE</version>
</dependency>
```

donde `2.X.Y.RELEASE` es la versión de Spring Boot.

En el siguiente enlace podemos encontrar el DDL necesario para generar las tablas para persistir la información asociada a OAuth2.0: clientes, tokens, ...

https://github.com/spring-projects/spring-security-oauth/blob/master/spring-security-oauth2/src/test/resources/schema.sql

```
-- used in tests that use HSQL
create table oauth_client_details (
  client_id VARCHAR(256) PRIMARY KEY,
  resource_ids VARCHAR(256),
  client_secret VARCHAR(256),
  scope VARCHAR(256),
  authorized_grant_types VARCHAR(256),
  web_server_redirect_uri VARCHAR(256),
  authorities VARCHAR(256),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additional_information VARCHAR(4096),
  autoapprove VARCHAR(256)
);

create table oauth_client_token (
  token_id VARCHAR(256),
  token LONGVARBINARY,
  authentication_id VARCHAR(256) PRIMARY KEY,
  user_name VARCHAR(256),
  client_id VARCHAR(256)
);

create table oauth_access_token (
  token_id VARCHAR(256),
  token LONGVARBINARY,
  authentication_id VARCHAR(256) PRIMARY KEY,
  user_name VARCHAR(256),
  client_id VARCHAR(256),
  authentication LONGVARBINARY,
  refresh_token VARCHAR(256)
);

create table oauth_refresh_token (
  token_id VARCHAR(256),
  token LONGVARBINARY,
  authentication LONGVARBINARY
);

create table oauth_code (
  code VARCHAR(256), authentication LONGVARBINARY
);

create table oauth_approvals (
    userId VARCHAR(256),
    clientId VARCHAR(256),
    scope VARCHAR(256),
    status VARCHAR(10),
    expiresAt TIMESTAMP,
```

```
    lastModifiedAt TIMESTAMP
);


-- customized oauth_client_details table
create table ClientDetails (
  appId VARCHAR(256) PRIMARY KEY,
  resourceIds VARCHAR(256),
  appSecret VARCHAR(256),
  scope VARCHAR(256),
  grantTypes VARCHAR(256),
  redirectUrl VARCHAR(256),
  authorities VARCHAR(256),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additionalInformation VARCHAR(4096),
  autoApproveScopes VARCHAR(256)
);
```

En esta web se indica que se ha probado con HSQL, y también funciona en H2.