UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Objetivos de la sesión:

- *Crear **entidades JPA** asociadas a una tabla que tenga relaciones **@ManyToMany**.*
- *Aprender a **securizar una API REST** con una autentificación basada en **tokens (JWT)**.*

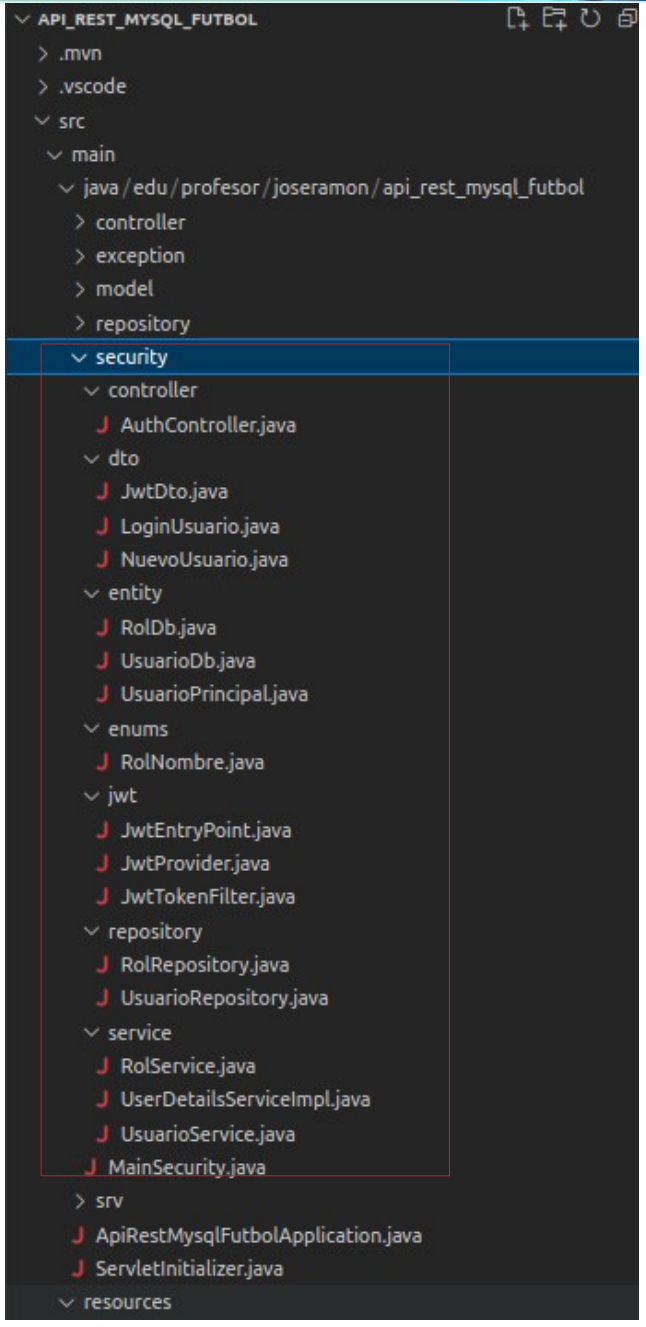
UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

Los **pasos** que realizaremos en esta práctica son:

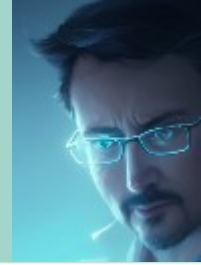
1º Crear tablas en mySql para securizar la API Rest e introducir roles básicos.
2º Crear las Entidades y enlazarlas con @ManyToMany.
3º Crear los Repositorios.
4º Crear los Servicios.
5º Configurar Spring Security y JWT.
6º Configurar la implementación de la interfaz UserDetails: UsuarioPrincipal.
7º Configurar la implementación de la interfaz UserDetailsService: UserDetailsServiceImpl.
8º Configurar el filtro que valida el token JWT.
9º Configurar la clase que gestiona la seguridad : MainSecurity.
10º API Rest: Crear nuevo Usuario.
11º API Rest: Login Usuario.
12º Pruebas.
13º Método con permisos de Admin.

Para poder ***exportar más fácilmente las seguridad otros proyectos todas las clases nuevas de esta práctica estarán dentro del paquete "security"***.



UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



1º Crear tablas en mySql para securizar la API Rest e introducir roles básicos:

Vamos a crear las tablas necesarias para la autenticación del usuario: ***"usuarios","roles" y "usuarios_roles"***.

La tabla "usuarios_roles" contendrá una clave ajena a "usuarios" y otra a "roles" para permitir que un usuario pueda tener varios roles y un rol pueda ser asignado a varios usuarios.

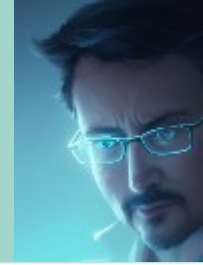
Un usuario puede ***autenticarse* en la API Rest** con un nickname y password correcto, pero puede que no esté ***autorizado* a realizar ciertas tareas** si no tiene el rol adecuado.

El alumno puede utilizar el ***script de creación en el DRIVE***.

En dicho script también se incluye la inserción de los ***roles básicos "ROLE_ADMIN" y "ROLE_USER"***.

Abrir — DWES_UD3_09_TablasNuevasParaImplementarSeguridad.sql — ~/Dropbox/2022.Simarro/DWES/UD3 — Guardar

```
DROP TABLE IF EXISTS `usuarios` ;

CREATE  TABLE IF NOT EXISTS `usuarios` (
  id INT(11)  NOT NULL AUTO_INCREMENT,
  `nombre` VARCHAR(255) NOT NULL ,
  `nickname` VARCHAR(255) NOT NULL ,
  `email` VARCHAR(255) NOT NULL ,
  `password` VARCHAR(255) NOT NULL ,
  PRIMARY KEY (`id`),
  CONSTRAINT usuario_uk_nickname UNIQUE KEY (`nickname`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------
-- Table `roles`
-- -----------------------------------------------------

DROP TABLE IF EXISTS `roles` ;

CREATE  TABLE IF NOT EXISTS `roles` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `nombre` VARCHAR(255) NOT NULL ,
  PRIMARY KEY (`id`) )
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;

-- -----------------------------------------------------
-- Table `usuarios_roles`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `usuarios_roles` ;

CREATE  TABLE IF NOT EXISTS `usuarios_roles` (
  idUsuario INT(11)  NOT NULL,
  idRol INT(11)  NOT NULL,
  PRIMARY KEY (`idUsuario`,`idRol`),
  CONSTRAINT `usuarios_roles_fk_usuarios`
    FOREIGN KEY (`idUsuario` )
    REFERENCES `usuarios` (`id` ),
  CONSTRAINT `usuarios_roles_fk_roles`
    FOREIGN KEY (`idRol` )
    REFERENCES `roles` (`id` ) )
ENGINE = InnoDB
DEFAULT CHARACTER SET = latin1;
```



2º Crear las Entidades y enlazarlas con @ManyToMany:

No es un olvido, no hace falta crear la tabla “usuarios_roles” porque gracias a la relación ***@ManyToMany*** podemos saber que roles tiene un usuario sin mapear dicha tabla. Fijate como le decimos que el ***nickname debe ser único***:

RolNombre.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.enums;

public enum RolNombre {
    ROLE_ADMIN, ROLE_USER
}
```

RolDb.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.entity;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.enums.RolNombre;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@Table(name = "roles")
public class RolDb {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @NotNull
    @Enumerated(EnumType.STRING) //Si no por defecto seria numérico
    private RolNombre nombre;
}
```

UsuarioDb.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.entity;
import javax.persistence.*;
import javax.validation.constraints.NotNull;
import java.util.HashSet;
import java.util.Set;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@NoArgsConstructor
@AllArgsConstructor
@Data
@Entity
@Table(name = "usuarios")
public class UsuarioDb {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotNull
    private String nombre;
    @NotNull
    @Column(unique = true)
    private String nickname;
    @NotNull
    private String email;
    @NotNull
    private String password;
    @NotNull
    @ManyToMany(fetch = FetchType.EAGER)
    //En la tabla 'usuarios_roles' queremos sacar todos los 'idRol' correspondientes
    //al 'idUsuario' actual para poder generar la lista de 'roles' en 'UsuarioDb'.
    @JoinTable(name = "usuarios_roles", joinColumns = @JoinColumn(name = "idUsuario"),
    inverseJoinColumns = @JoinColumn(name = "idRol"))
    private Set<RolDb> roles = new HashSet<>();
    //Constructor con todos los campos menos 'id'
    public UsuarioDb(@NotNull String nombre, @NotNull String nickname,
                     @NotNull String email, @NotNull String password) {
        this.nombre = nombre;
        this.nickname = nickname;
        this.email = email;
        this.password = password;
    }
}
```

3º Crear los Repositorios:

De “Roles” nos intereresa poder buscar por el nombre del rol:

RolRepository.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > repository > RolRepository.java

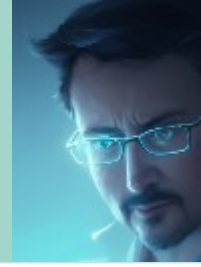
```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import edu.profesor.joseramon.api_rest_mysql_futbol.security.entity.RolDb;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.enums.RolNombre;

import java.util.Optional;

public interface RolRepository extends JpaRepository<RolDb, Integer> {
    Optional<RolDb> findByNombre(RolNombre rolNombre);
}
```

De los usuarios nos interesa poder buscar por nickname, y saber si un nickname o un email existe :

UsuarioRepository.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > repository > UsuarioRepository.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import edu.profesor.joseramon.api_rest_mysql_futbol.security.entity.UsuarioDb;

import java.util.Optional;


public interface UsuarioRepository extends JpaRepository<UsuarioDb, Integer> {
    Optional<UsuarioDb> findByNickname(String nickname);
    boolean existsByNickname(String nickname);
    boolean existsByEmail(String email);
}
```



4º Crear los Servicios:

Fijate que aparece una notación nueva ***@Transactional*** para evitar problemas de acceso concurrentes a la base de datos y mantener la coherencia de los datos si más de un usuario escribe al mismo tiempo sobre la tabla.

Puesto que las tablas relacionadas con usuarios y roles se utilizan con ***fines de seguridad*** vamos a simplificar y ***no utilizaremos DTOS*** para estas clases:

RolService.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import edu.profesor.joseramon.api_rest_mysql_futbol.security.entity.RolDb;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.enums.RolNombre;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.repository.RolRepository;

import java.util.Optional;

@Service
@Transactional//Mantiene la coherencia de la BD si hay varios accesos
public class RolService {

    @Autowired
    RolRepository rolRepository;

    public Optional<RolDb> getByRolNombre(RolNombre rolNombre){
        return rolRepository.findByNombre(rolNombre);
    }

    public void save(RolDb rol){
        rolRepository.save(rol);
    }
}
```

UsuarioService.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.service;

import org.springframework.beans.factory.annotation.Autowired;

@Service
@Transactional
public class UsuarioService {

    @Autowired
    UsuarioRepository usuarioRepository;

    public Optional<UsuarioDb> getByNickname(String nickname){
        return usuarioRepository.findByNickname(nickname);
    }

    public boolean existsByNickname(String nickname){
        return usuarioRepository.existsByNickname(nickname);
    }

    public boolean existsByEmail(String email){
        return usuarioRepository.existsByEmail(email);
    }

    public void save(UsuarioDb usuario){
        usuarioRepository.save(usuario);
    }
}
```

5º Configurar Spring Security y JWT:

En esta práctica vamos a ***securizar nuestra API Rest de futbol con Spring security y Json Web Token (JWT)*** . Para ello añadimos las dependencias ***"spring-boot-starter-security"*** y ***"jjwt"*** en el fichero de configuración "pom.xml":

pom.xml > project > dependencies > dependency

```
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--  DTOs  -->
        <dependency>
            <groupId>org.mapstruct</groupId>
            <artifactId>mapstruct</artifactId>
            <version>${org.mapstruct.version}</version>
            <scope>compile</scope>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-security -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
        <!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - Joseramon.profesor@gmail.com

5º Configurar Spring Security y JWT:

En ***application.properties*** debemos Añadir 2 nuevas variables:

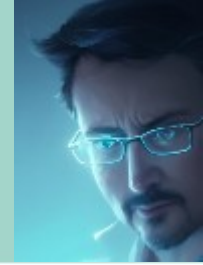
jwt.secret: Variable necesaria para la firma de seguridad.

jwt.expiration: Tiempo de expiración del token. 1 hora seria 3600, si le ponemos 36000 estamos indicando 10 horas.

application.properties 2

src > main > resources > application.properties

```
#Cambiamos puerto para evitar solape con Xampp
server.port=8090

# url a la base de datos MySQL
# futbol=nombre de la base de datos que hemos creado en MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/futbol
# nombre de usuario y contraseña
spring.datasource.username=futbol
spring.datasource.password=futbolSimarro

#En H2 utilizabamos el dialecto H2Dialect, aquí MySQL8Dialect
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
#Para evitar que nombreLargo lo mapee como nombre_largo y no lo e
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.

# actualizar base de datos y crear entidades
spring.jpa.hibernate.ddl-auto=update

#CONFIGURACIÓN SOLO durante las pruebas:
# Habilitar estadísticas hibernate
spring.jpa.properties.hibernate.generate_statistics=true
# Habilitar LOGGER de las estadísticas de hibernate
logging.level.org.hibernate.stat=debug
# Mostrar que consultas esta realizando Hibernate
spring.jpa.show-sql=true
# Formatear las consultas
spring.jpa.properties.hibernate.format_sql=true
# Mostrar los parametros que estan enviandose a las consultas
logging.level.org.hibernate.type=debug
#FIN CONFIGURACIÓN SOLO durante las pruebas

# SPRING SECURITY Y JWT:
#Variable necesaria para la firma de seguridad
jwt.secret = firmaSeguridadSimarro
#Tiempo de expiración del token
jwt.expiration = 36000
```



6º Configurar la implementación de la interfaz UserDetails: UsuarioPrincipal:

Para que ***cada clase tenga una única responsabilidad*** implementamos la seguridad en ***UsuarioPrincipal*** (implementa los privilegios de cada usuario), evitando que UsuarioDb tenga más de una función (solo accederá a la BD).

UsuarioPrincipal no es una entidad (porque no se va a crear una tabla en la BD), pero implementará los métodos de la interface ***"UserDetails" de Spring Security***.

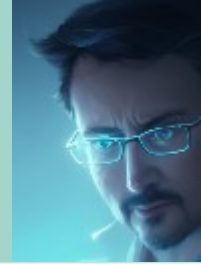
Puesto que tenemos que implementar los métodos de UserDetails ***no utilizaremos Lombok.***

Build() asignará los privilegios a cada usuario en base a la BD.

UsuarioPrincipal.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > entity > UsuarioPrincipal.java > UsuarioPrincipal > build(UsuarioDb)

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.entity;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import java.util.Collection;
import java.util.List;
import java.util.stream.Collectors;

public class UsuarioPrincipal implements UserDetails {//Clase encargarda de generar la seguridad: Implementa los privilegios de cada usuario
    private String nombreCompleto;
    private String nickname;
    private String email;
    private String password;
    private Collection<? extends GrantedAuthority> authorities;

    public UsuarioPrincipal(String nombreCompleto, String nickname, String email, String password, Collection<? extends GrantedAuthority> authorities) {
        this.nombreCompleto = nombreCompleto;
        this.nickname = nickname;
        this.email = email;
        this.password = password;
        this.authorities = authorities;
    }

    public static UsuarioPrincipal build(UsuarioDb usuarioDb){ //Convertimos un UsuarioDb es un UsuarioPrincipal con sus privilegios
        List<GrantedAuthority> authorities =
                usuarioDb.getRoles().stream().map(rol -> new SimpleGrantedAuthority(rol
                .getNombre().name())).collect(Collectors.toList()); //Convertimos los roles de la BD en una lista de 'GrantedAuthority'
        return new UsuarioPrincipal(usuarioDb.getNombre(), usuarioDb.getNickname(), usuarioDb.getEmail(), usuarioDb.getPassword(), authorities);
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

```



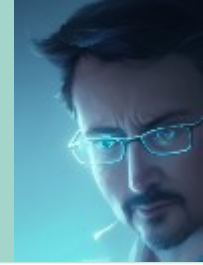
6º Configurar la implementación de la interfaz UserDetails: UsuarioPrincipal:

Como vemos UsuarioPrincipal tiene todos los campos de UsuarioDb y en vez de roles va a tener ***Authorities***, que son de tipo ***GrantedAuthorities***, clases propias de la ***seguridad de Spring Boot***.

Al ***implementar la interface UserDetails*** podemos decirle que añada los métodos que faltan con el botón derecho sobre el nombre de la clase y luego modificar la implementación de los métodos como se muestra en la imagen.

No es un error ***"getUsername()" devuelve el nickname*** porque UserDetails necesita tener Implementado "getUsername()".

```
37      @Override
38      public String getPassword() {
39          return password;
40      }
41
42      @Override
43      public String getUsername() {
44          return nickname;
45      }
46
47      @Override
48      public boolean isAccountNonExpired() {
49          return true;
50      }
51
52      @Override
53      public boolean isAccountNonLocked() {
54          return true;
55      }
56
57      @Override
58      public boolean isCredentialsNonExpired() {
59          return true;
60      }
61
62      @Override
63      public boolean isEnabled() {
64          return true;
65      }
66
67      public String getNombreCompleto() {…
70
71      public String getEmail() {
72          return email;
73      }
74  }
```



7º Configurar la implementación de la interfaz UserDetailsService:

Creamos ***UserDetailsServiceImpl*** como implementación de UserDetailsService. Por ello debemos de sobreescribir el método ***"loadUserByUserName()"*** y en nuestro caso utilizamos el nickname para encontrar el usuario en la BD y devolver un UsuarioPrincipal (implementación de la interfaz UserDetails):

UserDetailsServiceImpl.java

ramon > api_rest_mysql_futbol > security > service > UserDetailsServiceImpl.java > Language Support for Java(TM) by Red Hat > UserDetailsServiceImpl >

```java
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import edu.profesor.joseramon.api_rest_mysql_futbol.security.entity.UsuarioDb;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.entity.UsuarioPrincipal;

@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    UsuarioService usuarioService;

    @Override
    public UserDetails loadUserByUsername(String nickname) throws UsernameNotFoundException {
        //Método que debemos sobreescribir  (debe tener este nombre) de la interfaz UserDetailsService.
        //En nuestro caso buscamos por nickname en la BD y devolvemos un UsuarioPrincipal,
        //que es una implementación de la interfaz UserDetails.
        UsuarioDb usuario = usuarioService.getByNickname(nickname).get();
        return UsuarioPrincipal.build(usuario);
    }
}
```

8º Configurar el filtro que valida el token JWT:

Para configurar el filtro de las llamadas a la API Rest utilizando un token JWT debemos configurar ***3 clases***:

· ***JwtEntryPoint:***

Implementará la interfaz ***AuthenticationEntryPoint*** de Spring Security. Comprobará si hay un token válido y si no lanzará excepción 401.

· ***JwtProvider:***

Se encarga de generar el token y validarlo cuando se solicite.

· ***JwtTokenFilter:***

Se ejecutará en cada petición de la API Rest y utilizando JwtProvider comprobará que sea válido para permitir el acceso al recurso solicitado.

8º Configurar el filtro que valida el token JWT:

JwtEntryPoint.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > jwt > JwtEntryPoint.java > Language Support for Java(TM) by Red Hat > JwtEntryPoint

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.jwt;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@Component
public class JwtEntryPoint implements AuthenticationEntryPoint {
    //Comprueba si hay un toquen válido y si no lanza error de autenticación 401

    //SOLO EN DESARROLLO: utilizamos un loggger para ver que tipo de error nos da
    private final static Logger logger = LoggerFactory.getLogger(clazz: JwtEntryPoint.class);

    @Override
    public void commence(HttpServletRequest req, HttpServletResponse res, AuthenticationException e) throws IOException, ServletException {
        logger.error(msg: "Fallo en el método commence");
        res.sendError(HttpServletResponse.SC_UNAUTHORIZED, "No autorizado");
    }
}
```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

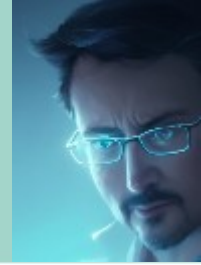
Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

8º Configurar el filtro que valida el token JWT:

JwtProvider.java

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.jwt;

import io.jsonwebtoken.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;

import edu.profesor.joseramon.api_rest_mysql_futbol.security.entity.UsuarioPrincipal;

import java.util.Date;

@Component
public class JwtProvider { //Se encargará de generar el token y comprobar su validez
    private final static Logger logger = LoggerFactory.getLogger(clazz: JwtProvider.class);

    @Value("${jwt.secret}")//valor en application.properties
    private String secret;
    @Value("${jwt.expiration}")//valor en application.properties
    private int expiration;

    public String generateToken(Authentication authentication){ //genera el token
        //Obtenemos el usuario principal (UserDetails)
        UsuarioPrincipal usuarioPrincipal = (UsuarioPrincipal) authentication.getPrincipal();
        //Configuramos nickname (getUsername), fecha de expedición, fecha de expiración y firmamos
        return Jwts.builder().setSubject(usuarioPrincipal.getUsername())
                .setIssuedAt(new Date())
                .setExpiration(new Date(new Date().getTime() + expiration * 1000))
                .signWith(SignatureAlgorithm.HS512, secret)
                .compact();
    }
```

```java
    public String getNicknameUsuarioFromToken(String token){//extrae el nickname del token
        return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody().getSubject();
    }

    public boolean validateToken(String token){//valida el token
        try {
            Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
            return true;
        }catch (MalformedJwtException e){
            logger.error(msg: "Token mal formado");
        }catch (UnsupportedJwtException e){
            logger.error(msg: "Token no soportado");
        }catch (ExpiredJwtException e){
            logger.error(msg: "Token expirado");
        }catch (IllegalArgumentException e){
            logger.error(msg: "Token vacío");
        }catch (SignatureException e){
            logger.error(msg: "Fallo en la firma");
        }
        return false;
    }
}
```



8º Configurar el filtro que valida el token JWT:

JwtTokenFilter se ejecuta una vez por cada petición por heredar de OncePerRequestFilter:

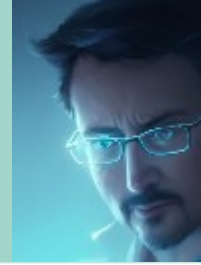
JwtTokenFilter.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > jwt > JwtTokenFilter.java > Language Support for Java(TM) by Red Hat > JwtTokenFilter > jwtProvider

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.jwt;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.filter.OncePerRequestFilter;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.service.UserDetailsServiceImpl;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class JwtTokenFilter extends OncePerRequestFilter {
//Se ejecutará en cada petición de la API Rest (por heredar de OncePerRequestFilter) y comprobará que sea válido el token utilizando el provider
    private final static Logger logger = LoggerFactory.getLogger(clazz: JwtTokenFilter.class);

    @Autowired
    JwtProvider jwtProvider;
    @Autowired
    UserDetailsServiceImpl userDetailsService;

```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



8º Configurar el filtro que valida el token JWT:

JwtTokenFilter.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > jwt > JwtTokenFilter.java > Language Support for Java(TM) by Red Hat > JwtTokenFilter > doFilterInternal(HttpServletReque

```java
25
26      @Override
27      protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain filterChain) throws ServletException, IOException {
28          try { //Comprueba el token y si es valido permite el acceso al recurso.
29              String token = getToken(req);
30              if(token != null && jwtProvider.validateToken(token)){//Token valido
31                  String nicknameUsuario = jwtProvider.getNicknameUsuarioFromToken(token);//Extraer nickname del token
32                  UserDetails userDetails = userDetailsService.loadUserByUsername(nicknameUsuario); //Buscamos UsuarioPrincipal(UserDetails) logeado
33                  //Obtenemos el UserNamePasswordAuthenticationToken en base al userDetails y sus autorizaciones
34                  UsernamePasswordAuthenticationToken auth =
35                          new UsernamePasswordAuthenticationToken(userDetails, credentials: null, userDetails.getAuthorities());
36                  SecurityContextHolder.getContext().setAuthentication(auth);//aplicamos autorización al contexto
37              }
38          } catch (Exception e){ //Si falla la autentificación
39              logger.error("Fallo de autentifación del token JWT:  " + e.getMessage());
40          }
41          //No falla autentificación y permitimos la petición
42          filterChain.doFilter(req, res);
43      }
44
45      private String getToken(HttpServletRequest request){
46          String header = request.getHeader(name: "Authorization");
47          if(header != null && header.startsWith("Bearer"))
48              return header.replace("Bearer ", "");
49          return null;
50      }
51  }
```

9º Configurar la clase que gestiona la seguridad: MainSecurity:

Creamos la clase ***MainSecurity*** para gestionar la seguridad de la API Rest:

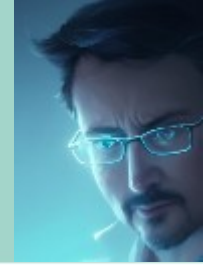
MainSecurity.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > MainSecurity.java > Language Support for Java(TM) by Red Hat > MainSecurity >

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.jwt.JwtEntryPoint;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.jwt.JwtTokenFilter;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.service.UserDetailsServiceImpl;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true, jsr250Enabled = true)
                            //'prePostEnabled' permite configurar que métodos solo tiene acceso el administrador
public class MainSecurity  {

    @Autowired
    UserDetailsServiceImpl userDetailsService; //Convierte la clase UsuarioDb en UsuarioPrincipal (UserDetails)

    @Autowired
    JwtEntryPoint jwtEntryPoint;//Si no hay token o no es válido devuelve error 401 "No Autorizado"

    @Bean
    public JwtTokenFilter jwtTokenFilter(){
        return new JwtTokenFilter();
    }

    @Bean
    public PasswordEncoder passwordEncoder(){//permite cifrar la contraseña
        return new BCryptPasswordEncoder();
    }

```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



9º Configurar la clase que gestiona la seguridad: MainSecurity:

MainSecurity.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > MainSecurity.java > Language Support for Java(TM) by Red Hat > MainSecurity >

```
41
42      @Bean
43      public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
44          //Inhabilitamos csrf con csrf().disable(). Si necesitaramos cookies no sería buena idea
45          //Todo lo que sea '/auth/' estará autorizado y para el resto hará falta el token JWT de autenticación
46          //Para controllar las excepciones utilizamos exceptionHandling indicando que utilice jwtEntryPoint
47          //Sesión sin estados (sin cookies) mediante STATELESS
48          //Añadimos el jwtTokenFilter para que se compruebe el token en cada petición y va a pasar el usuario al
49          //contexto de autenticación
50          http.cors().and().csrf().disable()
51          .authorizeRequests()
52          .antMatchers(...antPatterns: "/auth/**").permitAll()
53          .anyRequest().authenticated()
54          .and()
55          .exceptionHandling().authenticationEntryPoint(jwtEntryPoint)
56          .and()
57          .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
58          http.addFilterBefore(jwtTokenFilter(), beforeFilter: UsernamePasswordAuthenticationFilter.class);
59          return http.build();
60      }
61
62      @Bean
63      public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration)
64              throws Exception {
65          return authenticationConfiguration.getAuthenticationManager();
66      }
```

10º API Rest: Crear nuevo Usuario:

Para crear un nuevo Usuario desde la API Rest deberemos empezar creando el ***DTO NuevoUsuario***:

NuevoUsuario.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > dto > NuevoUs

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.dto;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.HashSet;
import java.util.Set;

@NoArgsConstructor
@AllArgsConstructor
@Data
public class NuevoUsuario {
    @NotBlank
    private String nickname;
    @NotBlank
    private String nombre;
    @Email
    private String email;
    @NotBlank
    private String password;
    //Al utilizar API Rest utilizamos objetos tipo Json y
    //es mejor que sean cadenas para agilizar el tráfico
    private Set<String> roles = new HashSet<>();
}
```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

10º API Rest: Crear nuevo Usuario:

Adicionalmente también hace falta los ***DTOS LoginUsuario y JwtDto***.

LoginUsuario.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > dto > LoginUs

```java
package edu.profesor.joseramon.api_rest_mysql_futbol.security.dto;

import javax.validation.constraints.NotBlank;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
@Data
public class LoginUsuario {
    @NotBlank
    private String nickname;
    @NotBlank
    private String password;
}
```

JwtDto.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > dto > JwtDto.java > JwtDto > JwtDto(String, String, Collect

```java

import java.util.Collection;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@NoArgsConstructor
@AllArgsConstructor
@Data
public class JwtDto {
    private String token;
    private String bearer = "Bearer";
    private String nickname;
    private Collection<? extends GrantedAuthority> authorities;
    //Constructor sin el atributo 'bearer'
    public JwtDto(String token, String nickname, Collection<? extends GrantedAuthority> authorities) {
        this.token = token;
        this.nickname = nickname;
        this.authorities = authorities;
    }
}
```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

10º API Rest: Crear nuevo Usuario:

Necesitamos ahora el procedimiento que atienda la llamada al API Rest de creación del nuevo usuario desde ***AuthController***:

AuthController.java

```
main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > controller > AuthController.java > Language Supp
package edu.profesor.joseramon.api_rest_mysql_futbol.security.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;

import edu.profesor.joseramon.api_rest_mysql_futbol.model.dto.Mensaje;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.dto.JwtDto;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.dto.LoginUsuario;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.dto.NuevoUsuario;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.entity.RolDb;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.entity.UsuarioDb;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.enums.RolNombre;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.jwt.JwtProvider;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.service.RolService;
import edu.profesor.joseramon.api_rest_mysql_futbol.security.service.UsuarioService;

import javax.validation.Valid;
import java.util.HashSet;
import java.util.Set;

@RestController
@RequestMapping("/auth")
@CrossOrigin //Al no poner nada más permitimos acceder desde
//cualquier origen
public class AuthController {

    @Autowired
    PasswordEncoder passwordEncoder;

    @Autowired
    AuthenticationManager authenticationManager;

    @Autowired
    UsuarioService usuarioService;

    @Autowired
    RolService rolService;
```

AuthController.java

```
src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > controller > AuthController.java > Language Support for Java(TM) by Red Hat > AuthController
    @Autowired
    JwtProvider jwtProvider;


    @PostMapping("/nuevo")
    public ResponseEntity<?> nuevo(@Valid @RequestBody NuevoUsuario nuevoUsuario, BindingResult bindingResult){
        if(bindingResult.hasErrors())
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Mensaje(mensaje: "Datos incorrectos o email inválido"));
        if(usuarioService.existsByNickname(nuevoUsuario.getNickname()))
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Mensaje(mensaje: "El nickname del usuario ya existe"));
        if(usuarioService.existsByEmail(nuevoUsuario.getEmail()))
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Mensaje(mensaje: "El email del usuario ya existe"));
        UsuarioDb usuarioDb =
                new UsuarioDb(nuevoUsuario.getNombre(), nuevoUsuario.getNickname(), nuevoUsuario.getEmail(),
                        passwordEncoder.encode(nuevoUsuario.getPassword()));
        Set<RolDb> rolesDb = new HashSet<>();
        rolesDb.add(rolService.getByRolNombre(RolNombre.ROLE_USER).get());
        if(nuevoUsuario.getRoles().contains("admin"))
            rolesDb.add(rolService.getByRolNombre(RolNombre.ROLE_ADMIN).get());
        usuarioDb.setRoles(rolesDb);
        usuarioService.save(usuarioDb);
        return ResponseEntity.status(HttpStatus.CREATED).body(new Mensaje(mensaje: "Usuario creado"));
    }
```

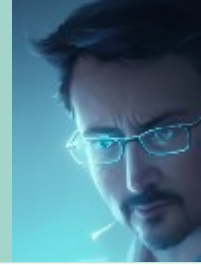
11º API Rest: Login Usuario:

Para finalizar implementamos en AuthController el método para logearnos:

AuthController.java

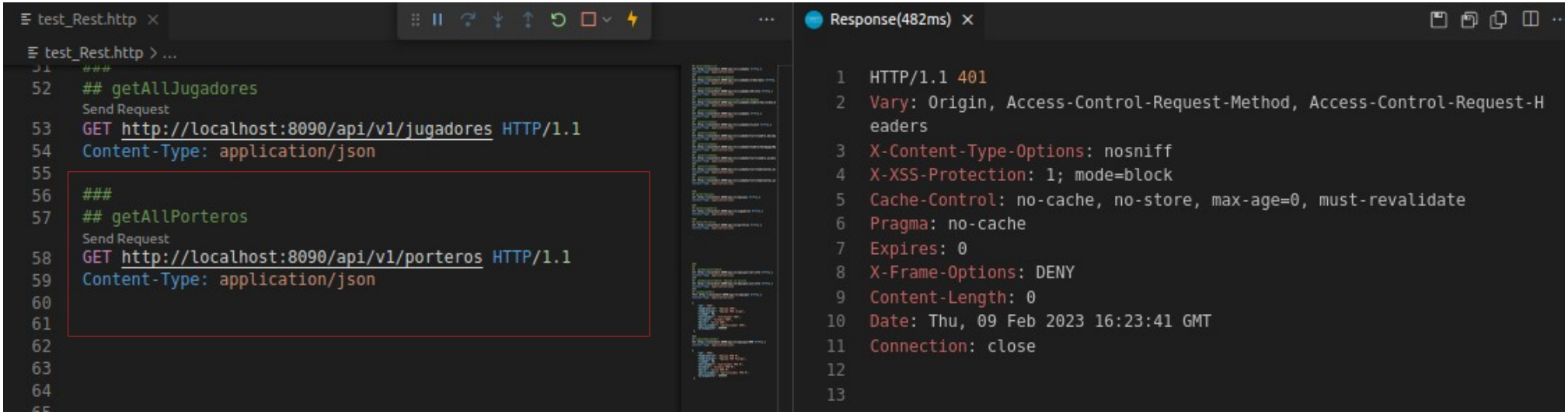
src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > security > controller > AuthController.java > Language Support for Java(TM) by Red Hat > AuthCon

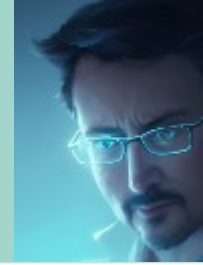
```
72      @PostMapping("/login")
73      public ResponseEntity<?> login(@Valid @RequestBody LoginUsuario loginUsuario, BindingResult bindingResult){
74          if(bindingResult.hasErrors())
75              return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(new Mensaje(mensaje: "Datos incorrectos"));
76          Authentication authentication =
77                  authenticationManager.authenticate(
78                      new UsernamePasswordAuthenticationToken(loginUsuario.getNickname(), loginUsuario.getPassword()));
79          SecurityContextHolder.getContext().setAuthentication(authentication);
80          String jwt = jwtProvider.generateToken(authentication);
81          UserDetails userDetails = (UserDetails)authentication.getPrincipal();
82          JwtDto jwtDto = new JwtDto(jwt, userDetails.getUsername(), userDetails.getAuthorities());
83          return ResponseEntity.status(HttpStatus.OK).body(jwtDto);
84      }
85  }
```



12º Pruebas:

Para realizar las pruebas empezamos por probar una llamada sin autenticarnos y comprobamos que nos deniega la operación:





12º Pruebas:

Creamos un usuario incorrecto para ver si se queja:

test_Rest.http

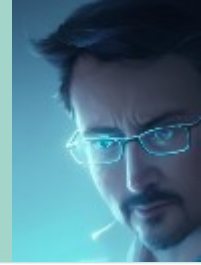
```
52  ## getAllJugadores
    Send Request
53  GET http://localhost:8090/api/v1/jugadores HTTP/1.1
54  Content-Type: application/json
55
56  ###
57  ## getAllPorteros
    Send Request
58  GET http://localhost:8090/api/v1/porteros HTTP/1.1
59  Content-Type: application/json
60
61  ###
62  ## nuevo incorrecto
    Send Request
63  POST http://localhost:8090/auth/nuevo HTTP/1.1
64  Content-Type: application/json
65
66  {
67      "nickname": "prueba",
68      "email": "@",
69      "password": "a",
70      "roles": ["admin"]
71  }
```

Response(207ms)

```
1   HTTP/1.1 400
2   Vary: Origin, Access-Control-Request-Method, Access-Control-Request-H
    eaders
3   X-Content-Type-Options: nosniff
4   X-XSS-Protection: 1; mode=block
5   Cache-Control: no-cache, no-store, max-age=0, must-revalidate
6   Pragma: no-cache
7   Expires: 0
8   X-Frame-Options: DENY
9   Content-Type: application/json
10  Transfer-Encoding: chunked
11  Date: Thu, 09 Feb 2023 16:32:32 GMT
12  Connection: close
13
14  {
15    "mensaje": "Datos incorrectos o email inválido"
16  }
```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



12º Pruebas:

Creamos un usuario correctamente:

test_Rest.http

```
###
## nuevo incorrecto
Send Request
POST http://localhost:8090/auth/nuevo HTTP/1.1
Content-Type: application/json

{
    "nickname": "prueba",
    "email": "@",
    "password": "a",
    "roles": ["admin"]
}

###
## nuevo ok
Send Request
POST http://localhost:8090/auth/nuevo HTTP/1.1
Content-Type: application/json

{
    "nickname": "joserra",
    "nombre": "Profesor Jose Ramón",
    "email": "joseramon.profesor@gmail.com",
    "password": "NoTeLoDigo@1",
    "roles": ["admin"]
}
```

Response(2919ms)

```
HTTP/1.1 201
Vary: Origin, Access-Control-Request-Method, Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 09 Feb 2023 16:45:41 GMT
Connection: close

{
  "mensaje": "Usuario creado"
}
```

Servidor: localhost » Base de datos: futbol » Tabla: usuarios

Examinar | Estructura | SQL | Buscar | Insertar | Exportar | Importar | Privilegios | Operaciones | Seguimiento

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código PHP] [Actualizar]

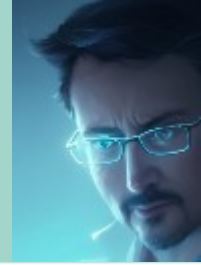
Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla

+ Opciones

	id	nombre	nickname	email	password
Editar Copiar Borrar	1	Profesor Jose Ramón	joserra	joseramon.profesor@gmail.com	$2a$10$ITXY3ECxSvpZPvXC1r3rGeuB6zV/4qwHsivLi2V1mRM...

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



12º Pruebas:

Ahora nos logeamos :

test_Rest.http

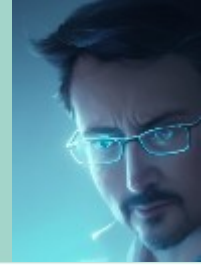
```
    "nickname": "prueba",
    "email": "@",
    "password": "a",
    "roles": ["admin"]
}

###
## nuevo ok
Send Request
POST http://localhost:8090/auth/nuevo HTTP/1.1
Content-Type: application/json

{
    "nickname": "joserra",
    "nombre": "Profesor Jose Ramón",
    "email": "joseramon.profesor@gmail.com",
    "password": "NoTeLoDigo@1",
    "roles": ["admin"]
}

###
## login ok
Send Request
POST http://localhost:8090/auth/login HTTP/1.1
Content-Type: application/json

{
    "nickname": "joserra",
    "password": "NoTeLoDigo@1"
}
```

(4195ms)

```
HTTP/1.1 200
Vary: Origin, Access-Control-Request-Method, Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 09 Feb 2023 17:12:31 GMT
Connection: close

{
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJqb3NlcnJhIiwiaWF0IjoxNjc1OTYyNzUxLCJleHAiOjE2NzU5OTg3NTF9.B2wdrtgOo7yvj9YkiUeW9BBdGyv0L8f72R9LH2mKRhxeOg1eYfZXs4kf8sUrD8Tebe6QHgjs_yUtwtCtd2W2Ig",
  "bearer": "Bearer",
  "nickname": "joserra",
  "authorities": [
    {
      "authority": "ROLE_USER"
    },
    {
      "authority": "ROLE_ADMIN"
    }
  ]
```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



12º Pruebas:

Y volvemos a realizar la operación con el token anterior:

test_Rest.http

```
67      "nickname": "prueba",
68      "email": "@",
69      "password": "a",
70      "roles": ["admin"]
71    }
72
73    ###
74    ## nuevo ok
      Send Request
75    POST http://localhost:8090/auth/nuevo HTTP/1.1
76    Content-Type: application/json
77
78    {
79        "nickname": "joserra",
80        "nombre": "Profesor Jose Ramón",
81        "email": "joseramon.profesor@gmail.com",
82        "password": "NoTeLoDigo@1",
83        "roles": ["admin"]
84    }
85
86    ###
87    ## login ok
      Send Request
88    POST http://localhost:8090/auth/login HTTP/1.1
89    Content-Type: application/json
90
91    {
92        "nickname": "joserra",
93        "password": "NoTeLoDigo@1"
94    }
95
96    ###
97    ## getAllPorteros con JWGT
      Send Request
98    GET http://localhost:8090/api/v1/porteros HTTP/1.1
99    Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJqb3NlcnJh
100
101
102
103
104
105
```

(3076ms)

```
1   HTTP/1.1 200
2   Vary: Origin, Access-Control-Request-Method, Access-Control-Request-Hea
    ders
3   X-Content-Type-Options: nosniff
4   X-XSS-Protection: 1; mode=block
5   Cache-Control: no-cache, no-store, max-age=0, must-revalidate
6   Pragma: no-cache
7   Expires: 0
8   X-Frame-Options: DENY
9   Content-Type: application/json
10  Transfer-Encoding: chunked
11  Date: Thu, 09 Feb 2023 17:15:04 GMT
12  Connection: close
13
14  {
15    "totalItems": 44,
16    "data": [
17      {
18        "idEquipo": "ath",
19        "dorsal": 1,
20        "nombre": "Gorka Iraizoz"
21      },
22      {
23        "idEquipo": "ath",
24        "dorsal": 13,
25        "nombre": "Raúl Fernández"
26      },
27      {
28        "idEquipo": "atm",
29        "dorsal": 1,
30        "nombre": "Sergio Asenjo"
31      }
32    ],
33    "totalPages": 15,
34    "pageSize": 3,
35    "currentPage": 0
36  }
```

13º Método con permisos de 'Admin':

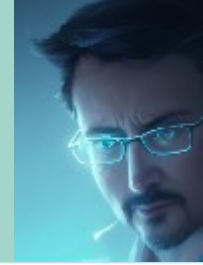
Para permitir hacer ciertas operaciones solo a Administradores podemos utilizar la notación **@*PreAuthorize***. Probemoslo con :

PorteroRestController.java

src > main > java > edu > profesor > joseramon > api_rest_mysql_futbol > controller > PorteroRestContro

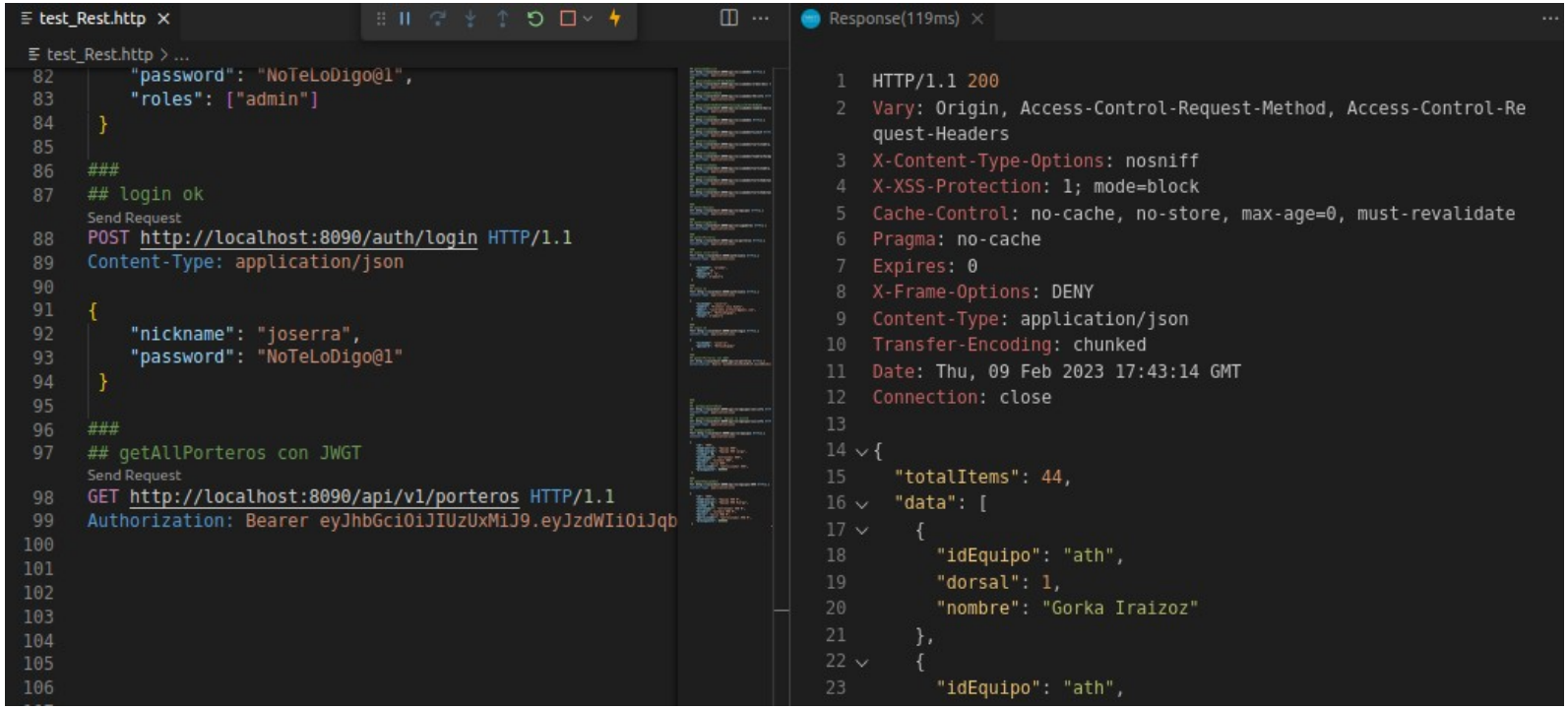
```
1   package edu.profesor.joseramon.api_rest_mysql_futbol.controller;
2
3   import org.springframework.security.access.prepost.PreAuthorize;…
22
23
24
25  @RestController
26  @RequestMapping("/api/v1/")
27  public class PorteroRestController {
28      private PorteroService porteroService;
29
30      public PorteroRestController(PorteroService porteroService){
31          this.porteroService=porteroService;
32      }
33
34      @PreAuthorize("hasRole['ROLE_ADMIN']")
35      @GetMapping("/porteros")
36      public ResponseEntity<Map<String, Object>> getAllPorteros(
37          @RequestParam(defaultValue = "0") int page,
38          @RequestParam(defaultValue = "3") int size,
39          @RequestParam(defaultValue = "idEquipo,asc") String[] sort) {
40
41          try {
```

Puedes consultar más información sobre permisos sobre métodos en Spring Security en : https://www.baeldung.com/spring-security-method-security



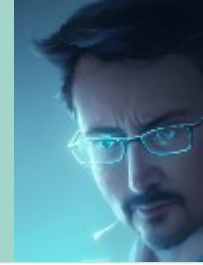
13º Método con permisos de 'Admin':

Probemos primero con el usuario que tenia permisos de admin:



UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com



13º Método con permisos de 'Admin':

Ahora probemos a crear un usuario que no sea adminstrador y comprobemos que no nos deja consultar la lista de porteros (error 403):

test_Rest.http

```
## nuevo ok
Send Request
POST http://localhost:8090/auth/nuevo HTTP/1.1
Content-Type: application/json

{
    "nickname": "prueba",
    "nombre": "Prueba Profesor",
    "email": "prueba.profesor@gmail.com",
    "password": "NoTeLoDigo@1"
}

###
## login ok
Send Request
POST http://localhost:8090/auth/login HTTP/1.1
Content-Type: application/json

{
    "nickname": "prueba",
    "password": "NoTeLoDigo@1"
}
```

Response(121ms)

```
HTTP/1.1 200
Vary: Origin, Access-Control-Request-Method, Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Type: application/json
Transfer-Encoding: chunked
Date: Thu, 09 Feb 2023 17:45:36 GMT
Connection: close

{
  "token": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwcnVlYmEiLCJpYXQiOjE2NzU5NjQ3MzYsImV4cCI6MTY3NjAwMDczNn0.DCRXYtXwpsk6Dah6QKLLMJawlK06o-bLtSVrtU76VpwyaafqBOb9DSN5c7Rk84W7DK4tuJQGMQ5AqBuXUGnQ6w",
```

test_Rest.http

```
}

###
## login ok
Send Request
POST http://localhost:8090/auth/login HTTP/1.1
Content-Type: application/json

{
    "nickname": "prueba",
    "password": "NoTeLoDigo@1"
}

###
## getAllPorteros con JWT
Send Request
GET http://localhost:8090/api/v1/porteros HTTP/1.1
Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwc
```

Response(49ms)

```
HTTP/1.1 403
Vary: Origin, Access-Control-Request-Method, Access-Control-Request-Headers
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Content-Length: 0
Date: Thu, 09 Feb 2023 17:48:09 GMT
Connection: close
```

UD 3: Bases de datos y servicios REST
7.- Autentificación JWT

Desarrollo Web en Entorno Servidor - joseramon.profesor@gmail.com

EJERCICIO:

Sube la aplicación final al moodle.
Para ello:

1º Haz un “Run As \Maven Clean” para dejar solo los fichero fuentes y quitar momentaneamente los necesarios para ejecutar la aplicación (dependencias).
2º Comprime la carpeta de tu aplicación y ponle UD3_practica9_nombreAlumno.tar.gz como nombre al fichero comprimido donde nombreAlumno es el nombre del alumno que entrega la práctica.
3º Súbela al moodle.

IMPORTANTE: No comprimir en RAR, porque Ubuntu no lo lee bien y en clase tenemos Ubuntu. Si tuviésemos Windows, podemos comprimir en ZIP.