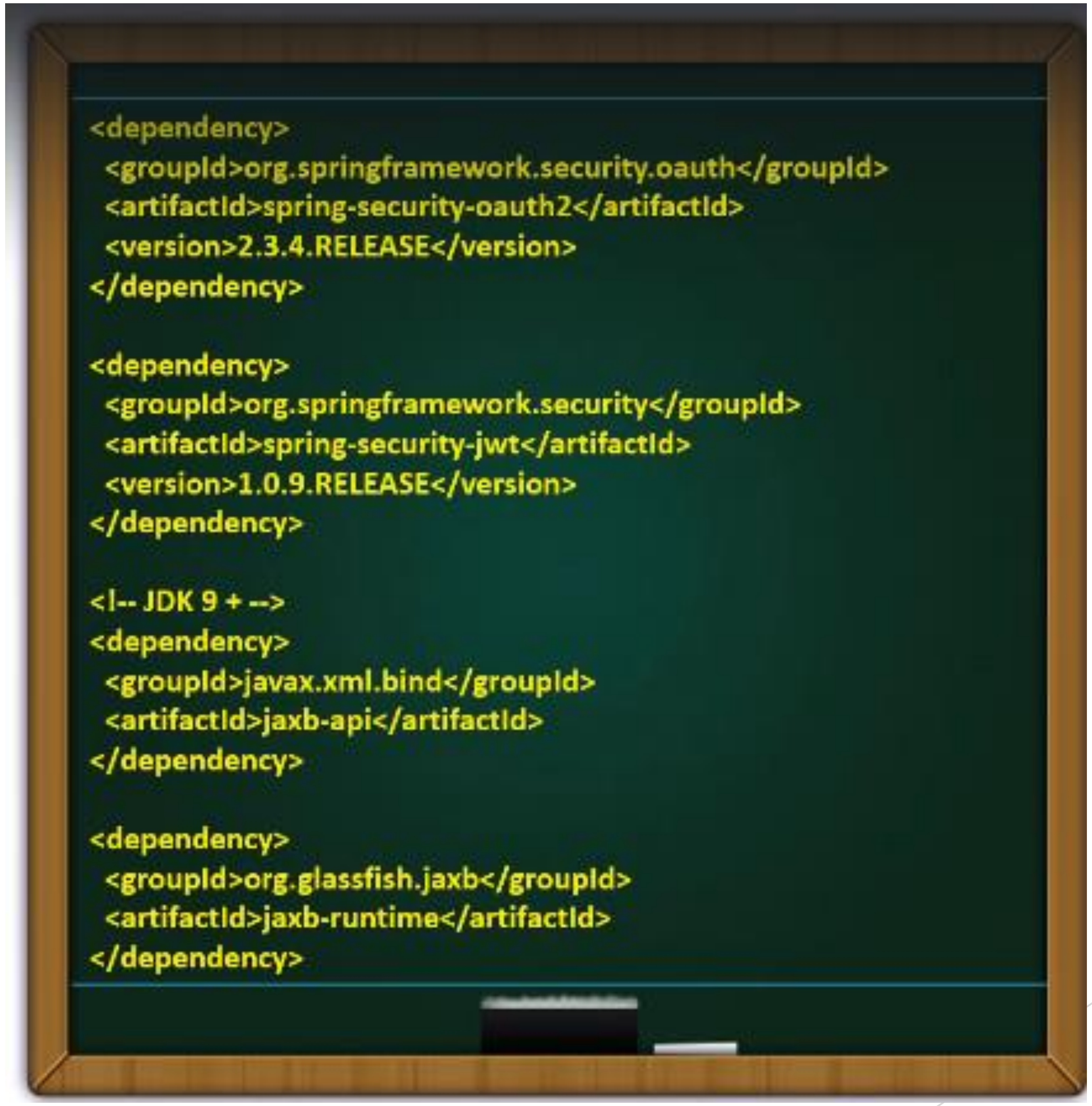
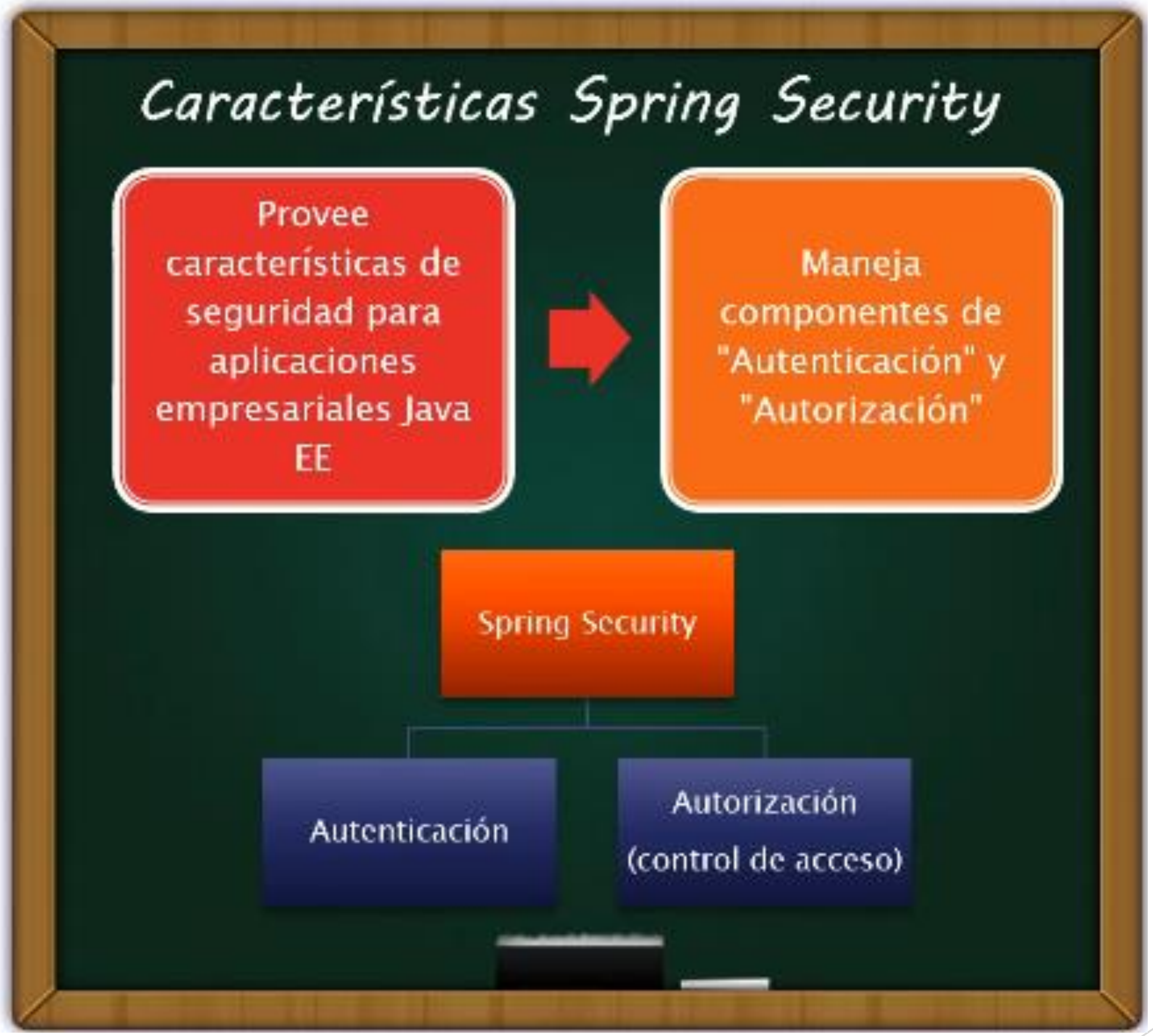
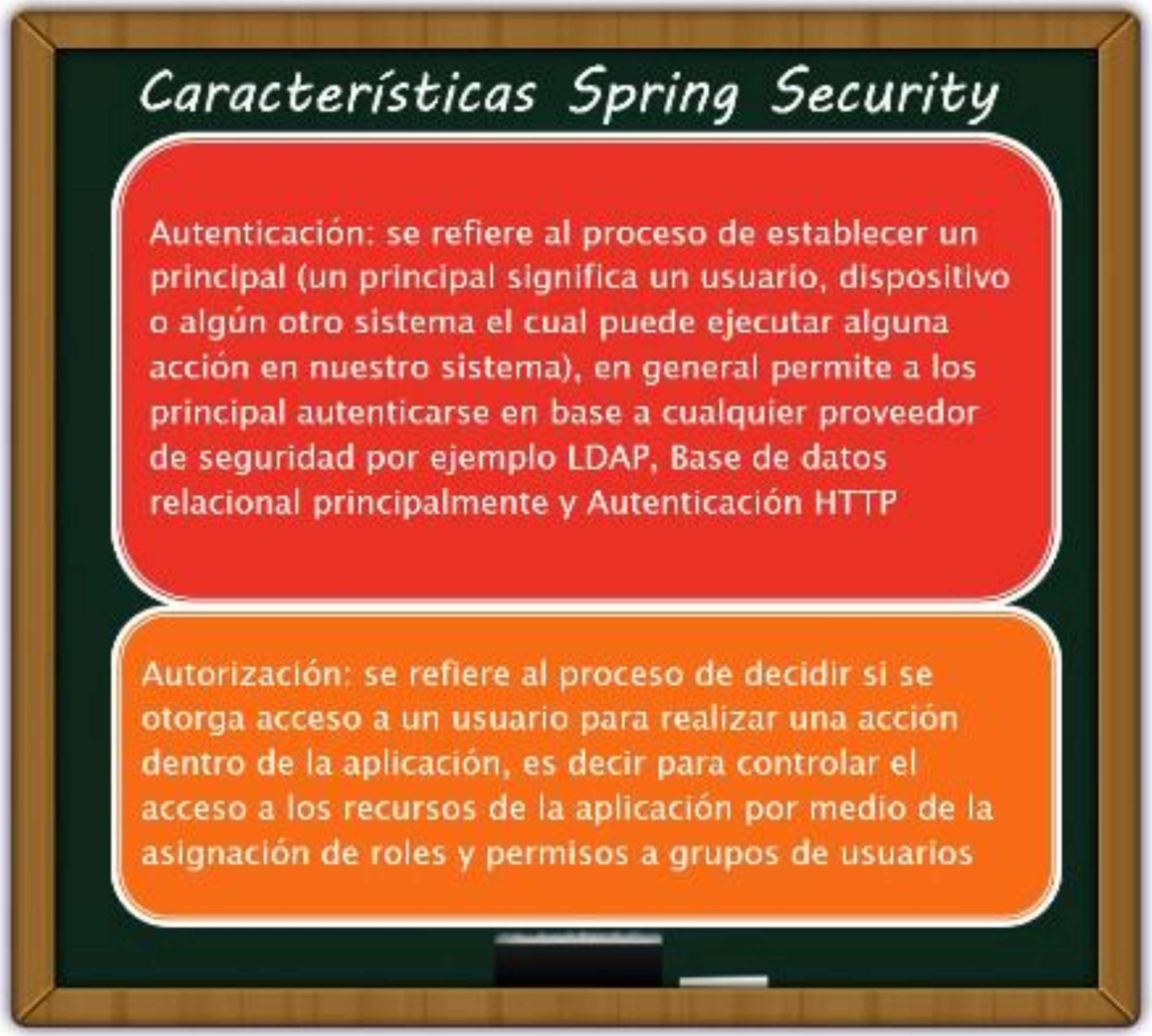
Módulo de Autenticación JWT Spring boot



```xml
<dependency>
  <groupId>org.springframework.security.oauth</groupId>
  <artifactId>spring-security-oauth2</artifactId>
  <version>2.3.4.RELEASE</version>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-jwt</artifactId>
  <version>1.0.9.RELEASE</version>
</dependency>

<!-- JDK 9 + -->
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
</dependency>

<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
</dependency>
```


Características Spring Security
Provee características de seguridad para aplicaciones empresariales Java EE
Maneja componentes de "Autenticación" y "Autorización"
Spring Security
Autenticación
Autorización (control de acceso)

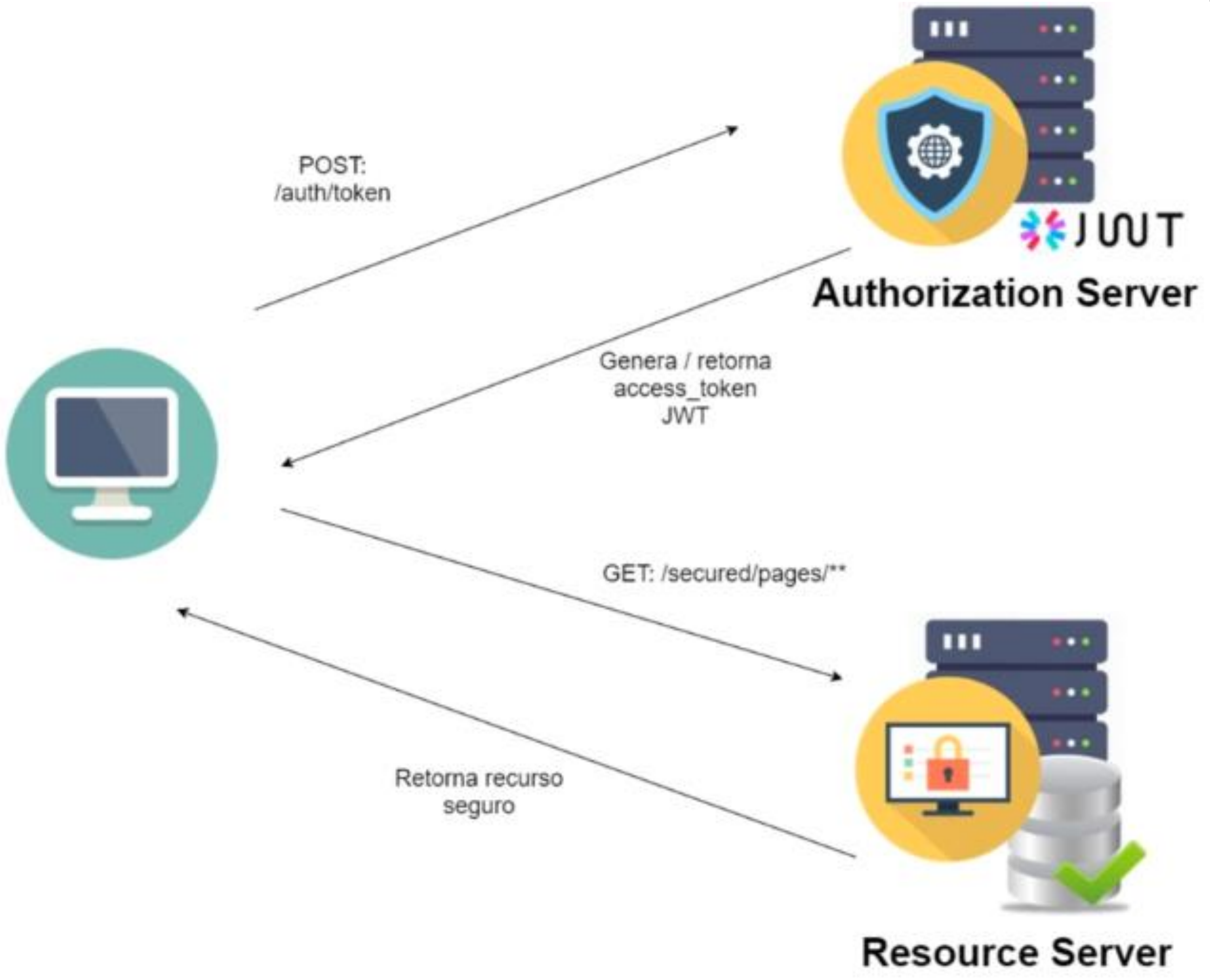


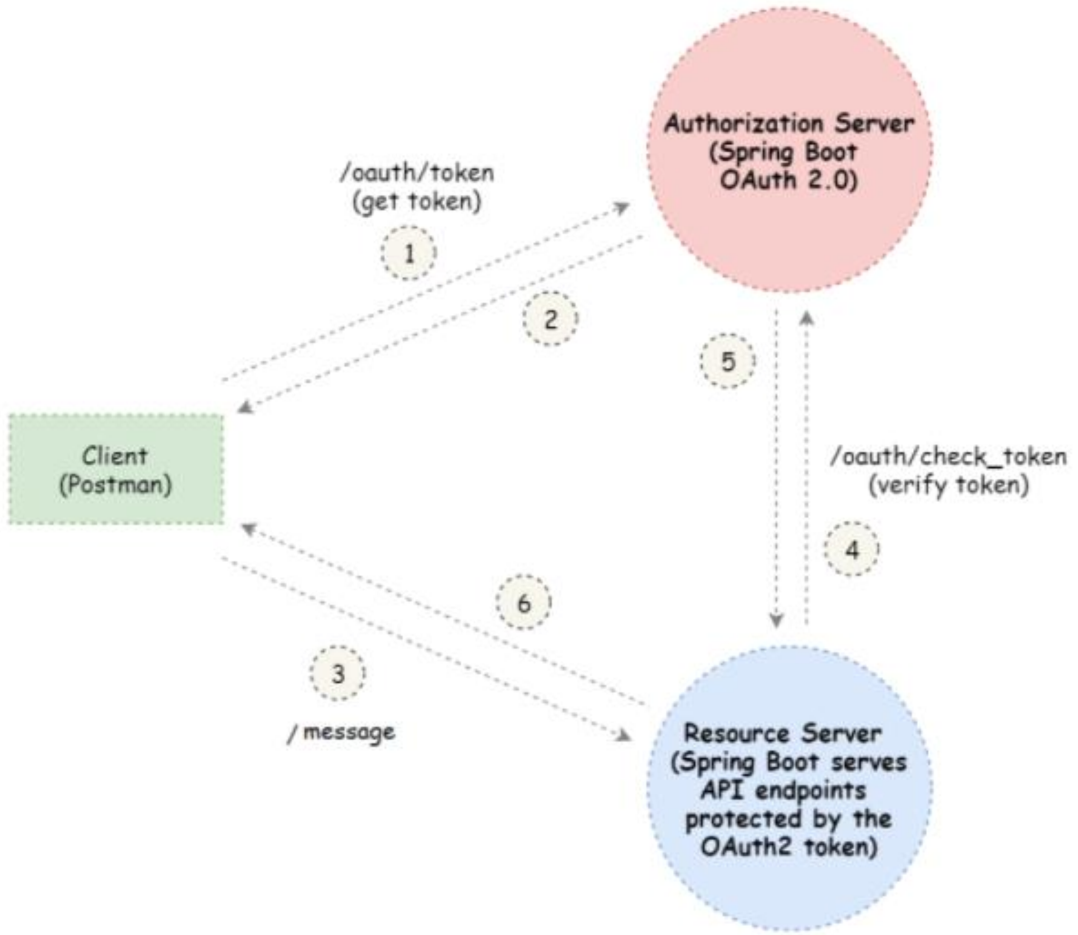
Características Spring Security

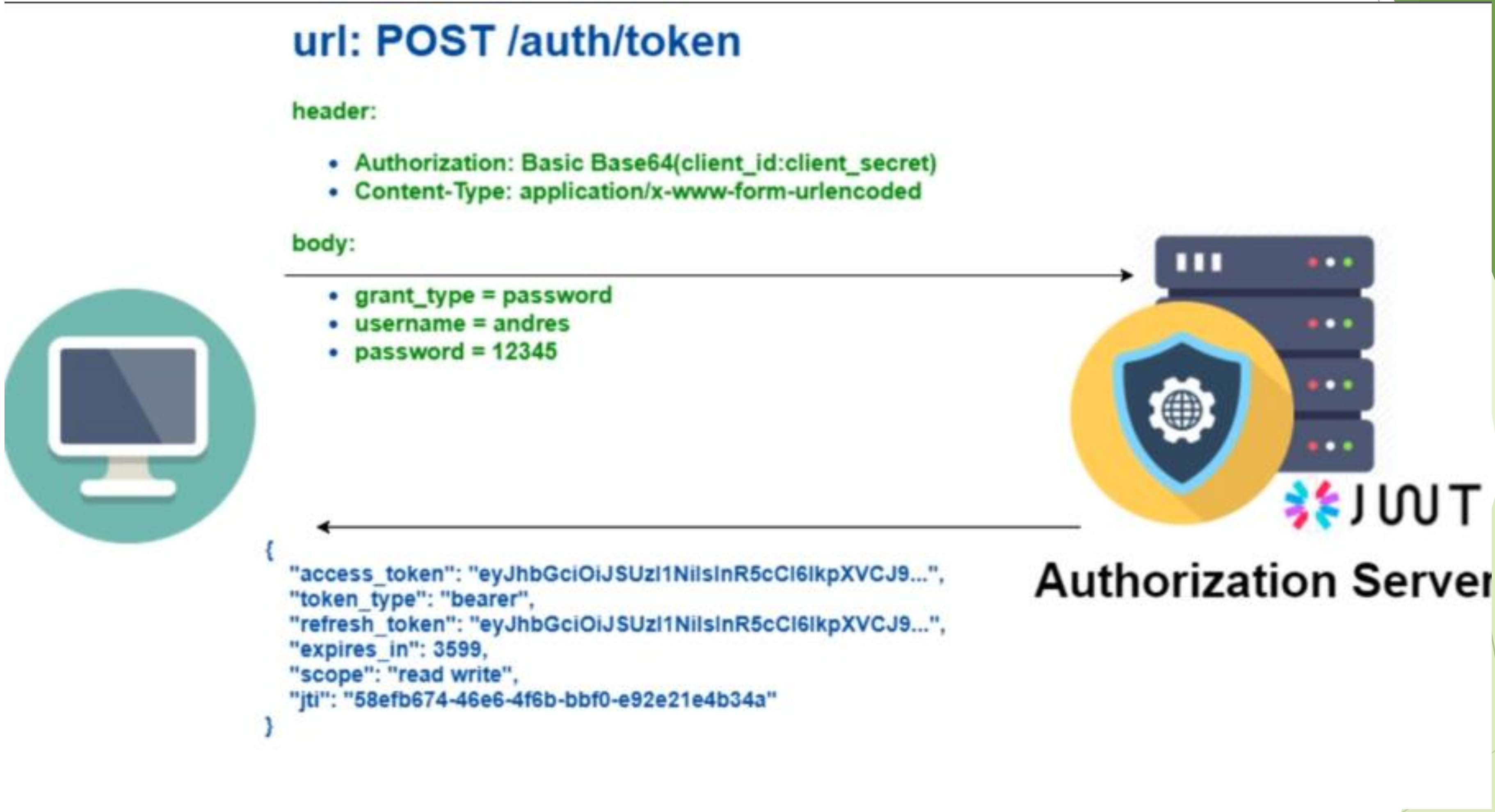
Autenticación: se refiere al proceso de establecer un principal (un principal significa un usuario, dispositivo o algún otro sistema el cual puede ejecutar alguna acción en nuestro sistema), en general permite a los principal autenticarse en base a cualquier proveedor de seguridad por ejemplo LDAP, Base de datos relacional principalmente y Autenticación HTTP

Autorización: se refiere al proceso de decidir si se otorga acceso a un usuario para realizar una acción dentro de la aplicación, es decir para controlar el acceso a los recursos de la aplicación por medio de la asignación de roles y permisos a grupos de usuarios

Roles


POST:
/auth/token
JWT
Authorization Server
Genera / retorna
access_token
JWT
GET: /secured/pages/**
Retorna recurso
seguro
Resource Server


Authorization Server (Spring Boot OAuth 2.0)
/oauth/token (get token)
1
2
5
Client (Postman)
/oauth/check_token (verify token)
4
6
3
/ message
Resource Server (Spring Boot serves API endpoints protected by the OAuth2 token)



url: POST /auth/token

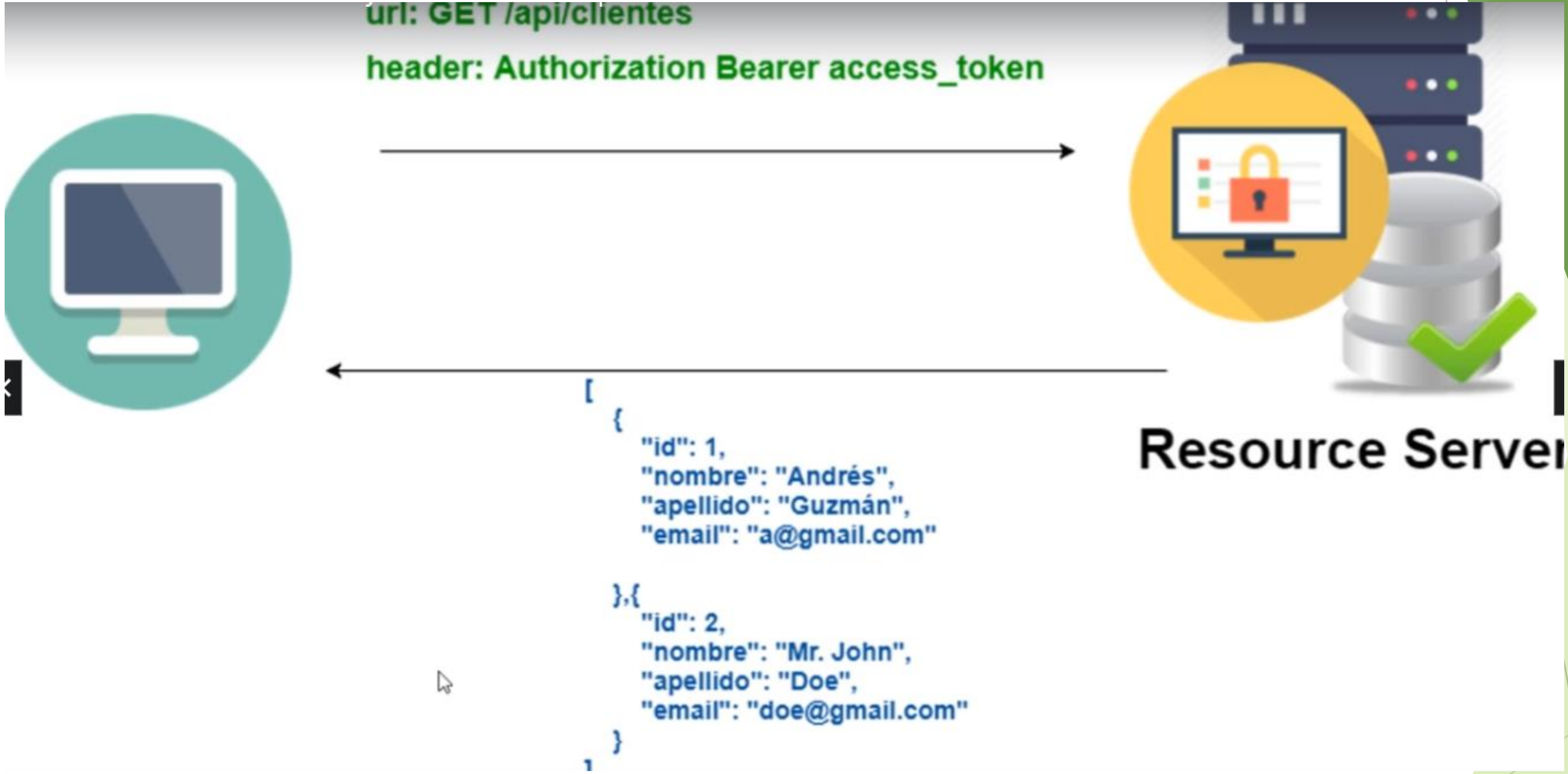
header:

- Authorization: Basic Base64(client_id:client_secret)
- Content-Type: application/x-www-form-urlencoded

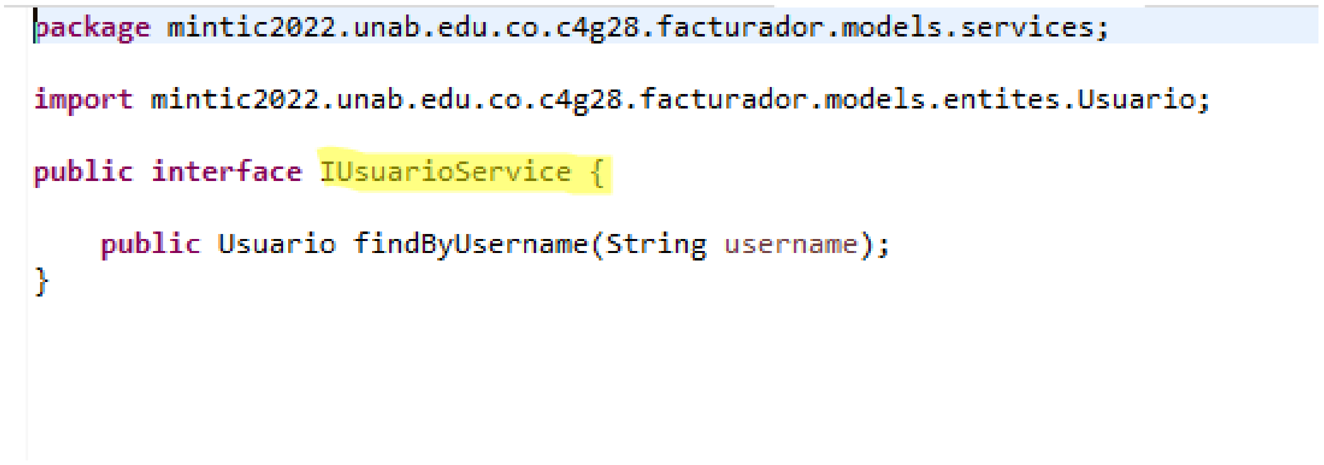
body:

- grant_type = password
- username = andres
- password = 12345

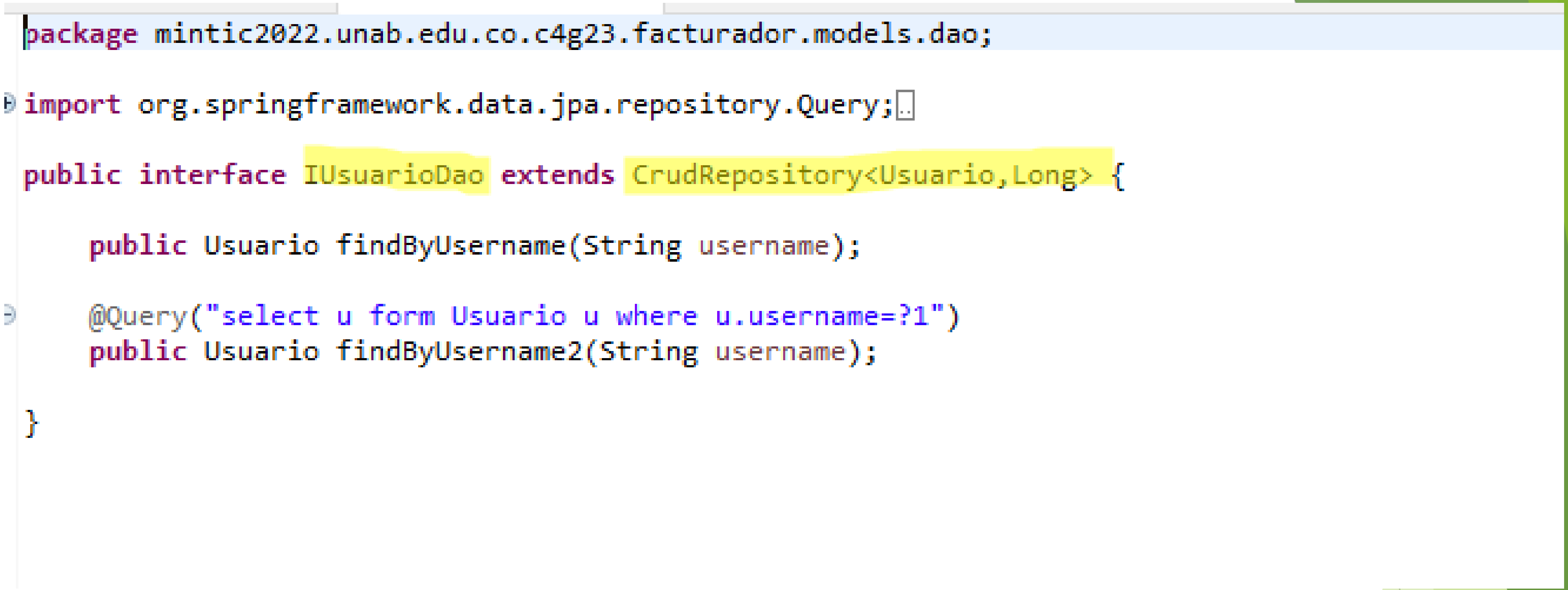
```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "token_type": "bearer",
  "refresh_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9...",
  "expires_in": 3599,
  "scope": "read write",
  "jti": "58efb674-46e6-4f6b-bbf0-e92e21e4b34a"
}
```



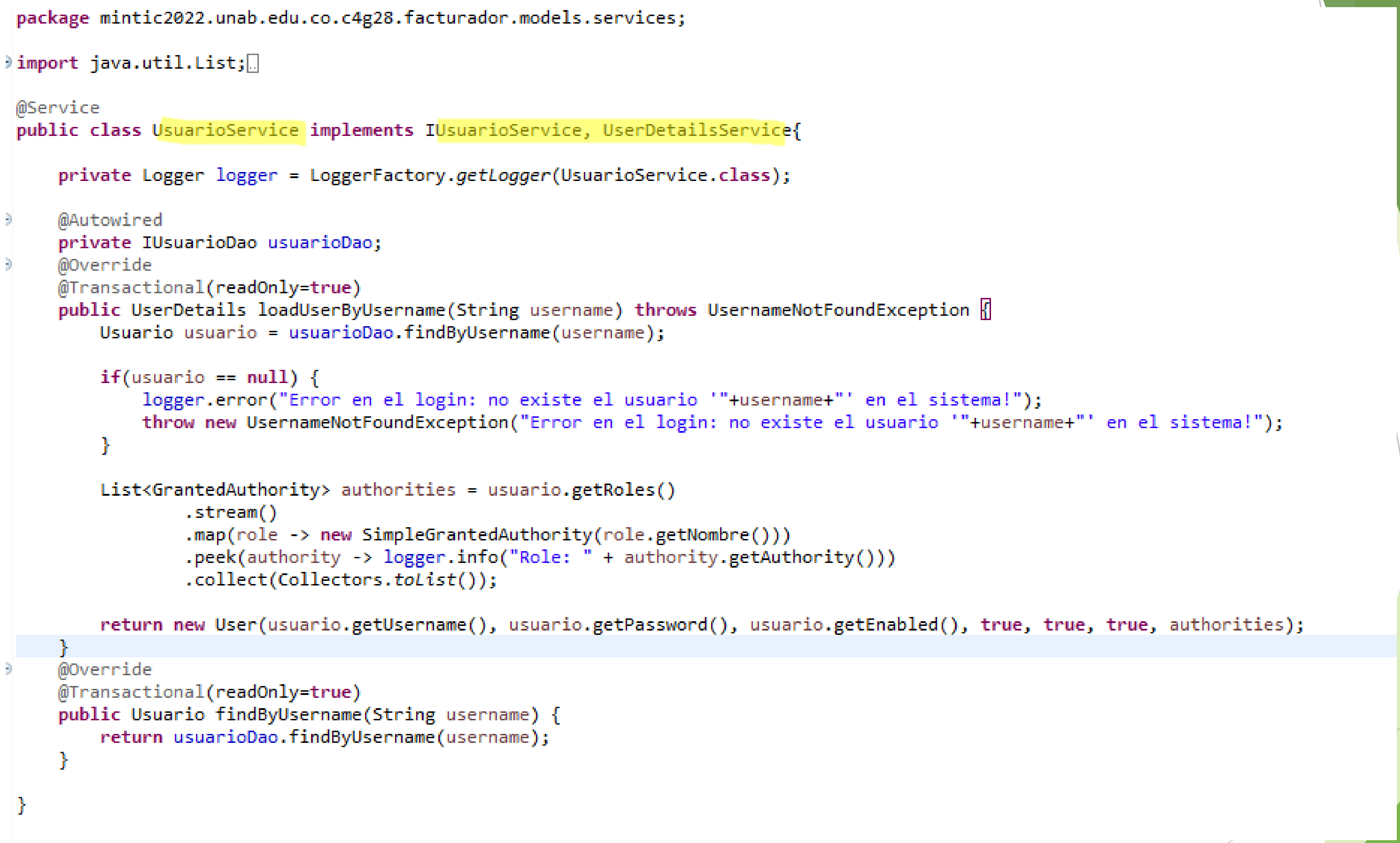
```
[
  {
    "id": 1,
    "nombre": "Andrés",
    "apellido": "Guzmán",
    "email": "a@gmail.com"

  },{
    "id": 2,
    "nombre": "Mr. John",
    "apellido": "Doe",
    "email": "doe@gmail.com"
  }
]
```



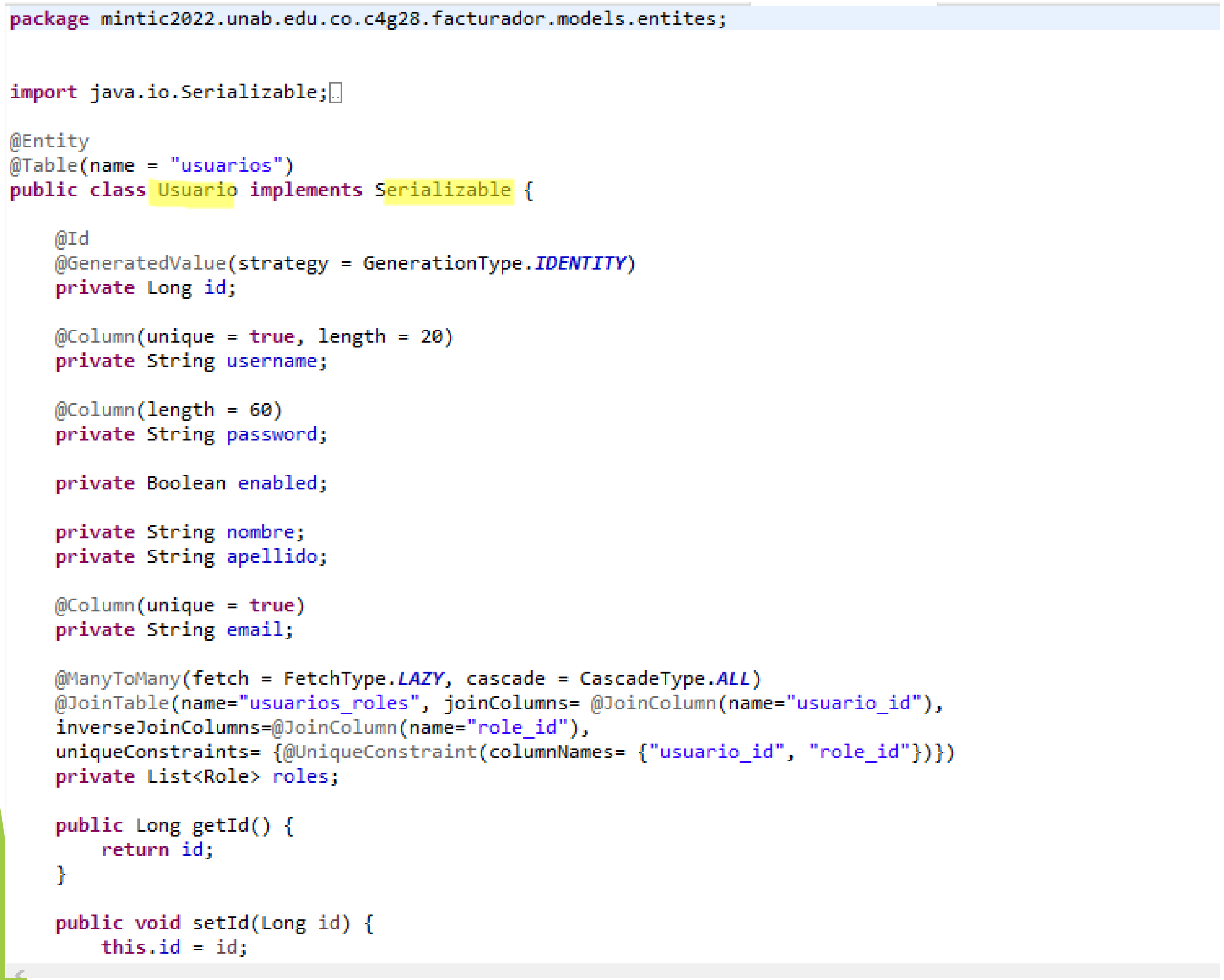
```java
package mintic2022.unab.edu.co.c4g28.facturador.models.services;

import mintic2022.unab.edu.co.c4g28.facturador.models.entites.Usuario;

public interface IUsuarioService {

    public Usuario findByUsername(String username);
}
```



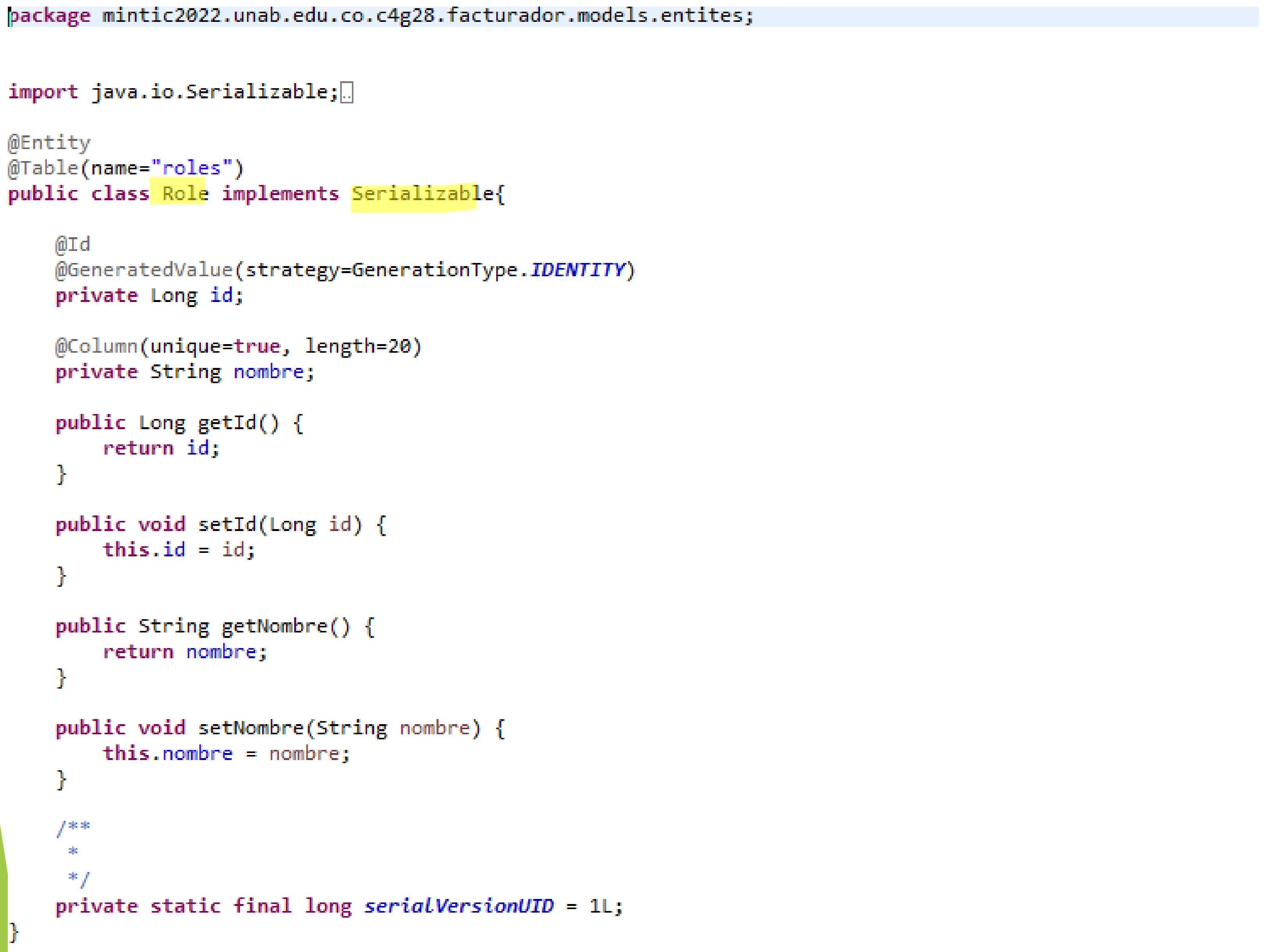
```java
package mintic2022.unab.edu.co.c4g23.facturador.models.dao;

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

public interface IUsuarioDao extends CrudRepository<Usuario,Long> {

    public Usuario findByUsername(String username);

    @Query("select u form Usuario u where u.username=?1")
    public Usuario findByUsername2(String username);

}
```



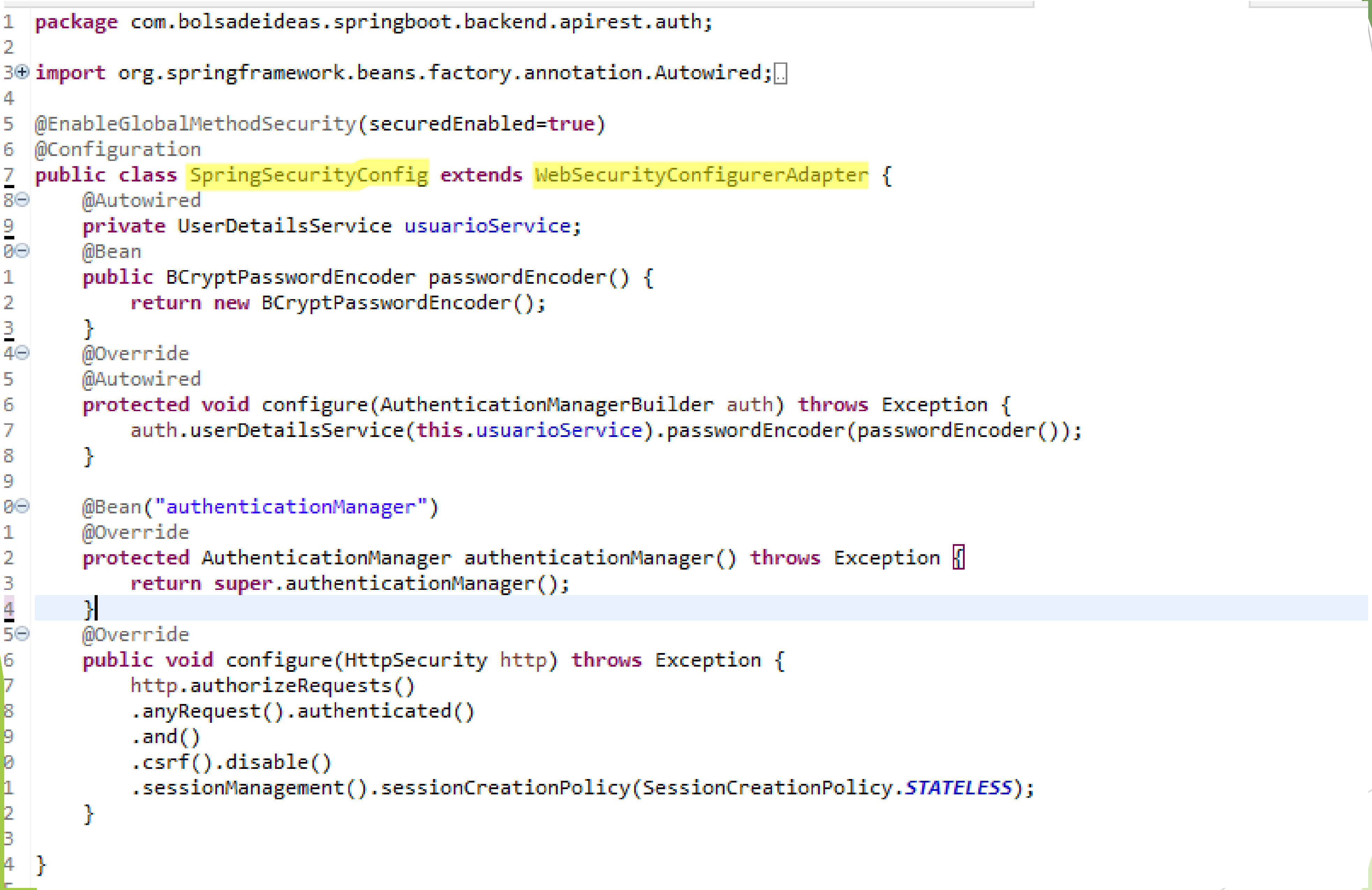
```java
package mintic2022.unab.edu.co.c4g28.facturador.models.services;

import java.util.List;

@Service
public class UsuarioService implements IUsuarioService, UserDetailsService{

    private Logger logger = LoggerFactory.getLogger(UsuarioService.class);

    @Autowired
    private IUsuarioDao usuarioDao;
    @Override
    @Transactional(readOnly=true)
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        Usuario usuario = usuarioDao.findByUsername(username);

        if(usuario == null) {
            logger.error("Error en el login: no existe el usuario '"+username+"' en el sistema!");
            throw new UsernameNotFoundException("Error en el login: no existe el usuario '"+username+"' en el sistema!");
        }

        List<GrantedAuthority> authorities = usuario.getRoles()
                .stream()
                .map(role -> new SimpleGrantedAuthority(role.getNombre()))
                .peek(authority -> logger.info("Role: " + authority.getAuthority()))
                .collect(Collectors.toList());

        return new User(usuario.getUsername(), usuario.getPassword(), usuario.getEnabled(), true, true, true, authorities);
    }
    @Override
    @Transactional(readOnly=true)
    public Usuario findByUsername(String username) {
        return usuarioDao.findByUsername(username);
    }

}
```



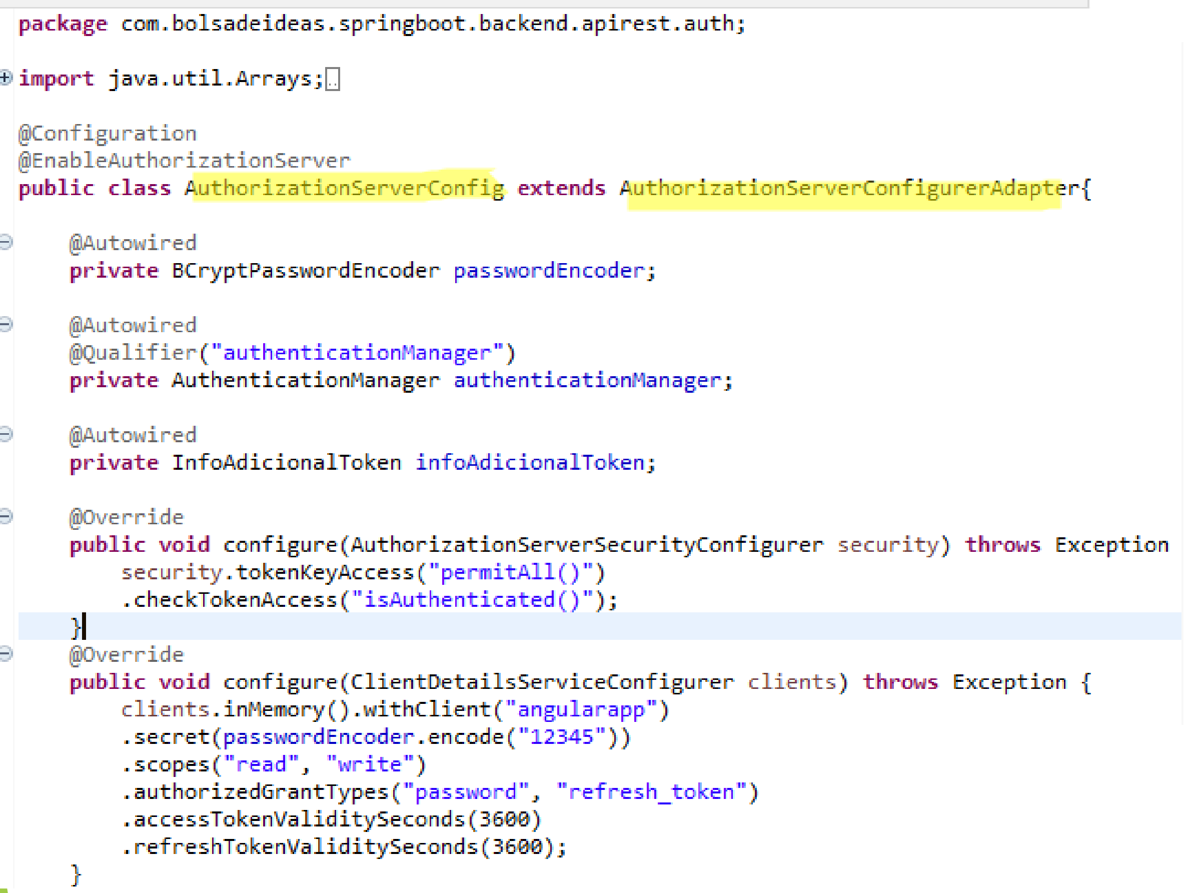
```
package mintic2022.unab.edu.co.c4g28.facturador.models.entites;

import java.io.Serializable;

@Entity
@Table(name = "usuarios")
public class Usuario implements Serializable {

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

    @Column(unique = true, length = 20)
    private String username;

    @Column(length = 60)
    private String password;

    private Boolean enabled;

    private String nombre;
    private String apellido;

    @Column(unique = true)
    private String email;

    @ManyToMany(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinTable(name="usuarios_roles", joinColumns= @JoinColumn(name="usuario_id"),
    inverseJoinColumns=@JoinColumn(name="role_id"),
    uniqueConstraints= {@UniqueConstraint(columnNames= {"usuario_id", "role_id"})})
    private List<Role> roles;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
```



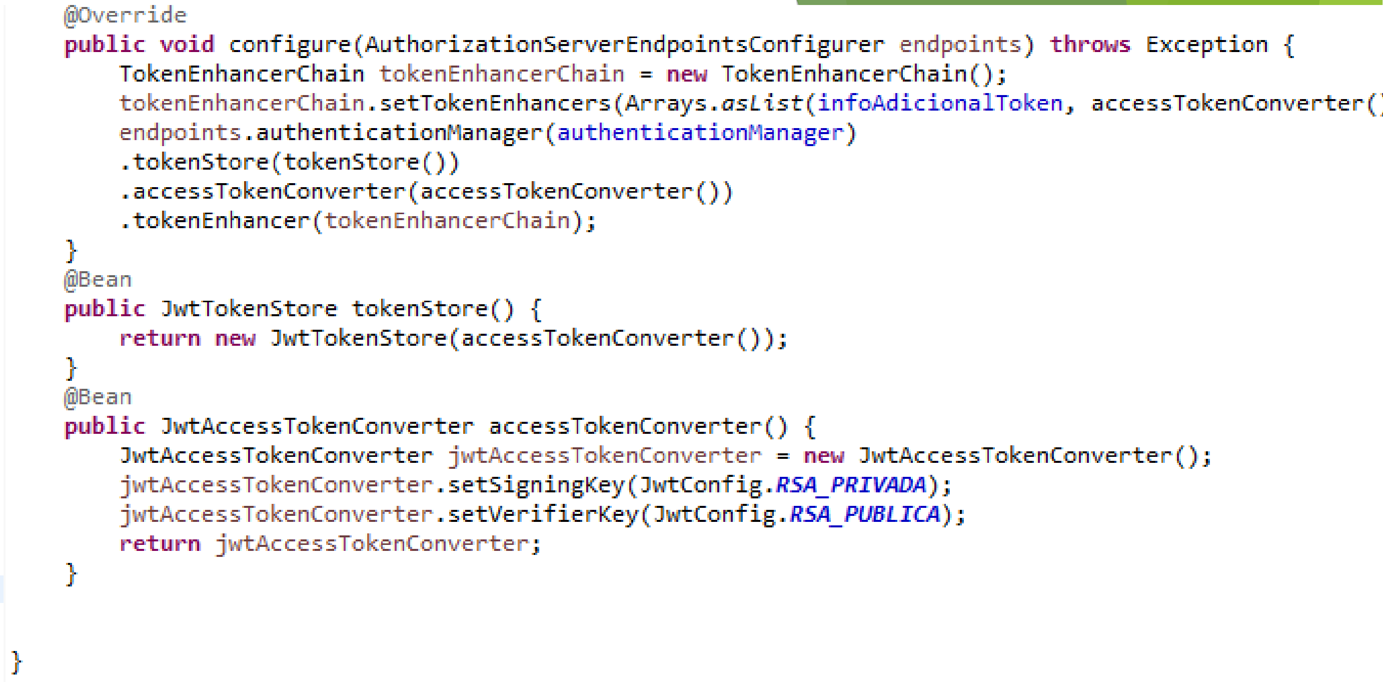
```java
package mintic2022.unab.edu.co.c4g28.facturador.models.entites;


import java.io.Serializable;

@Entity
@Table(name="roles")
public class Role implements Serializable{

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

    @Column(unique=true, length=20)
    private String nombre;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * 
     */
    private static final long serialVersionUID = 1L;
}
```



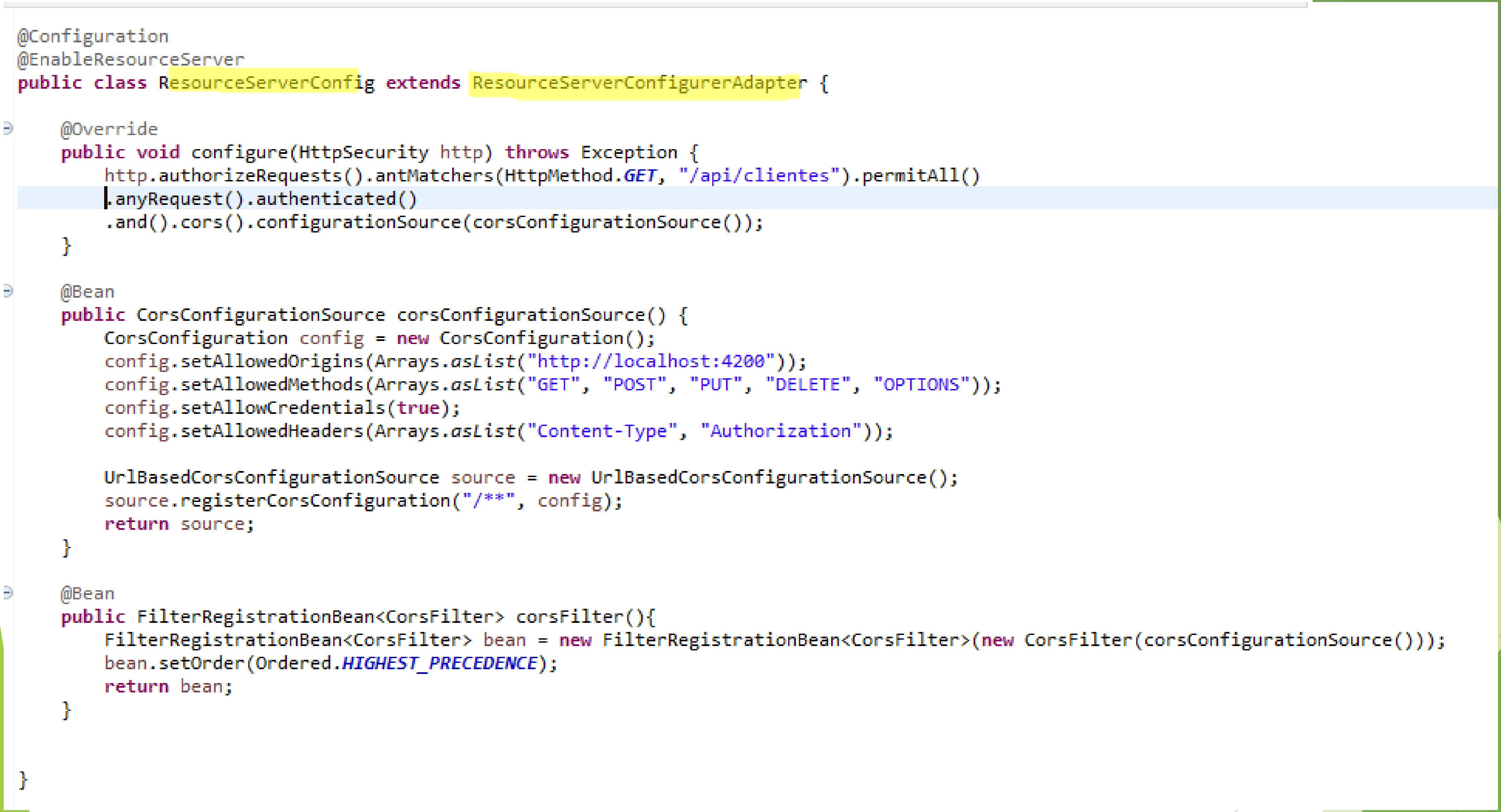
```
package com.bolsadeideas.springboot.backend.apirest.auth;

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

@EnableGlobalMethodSecurity(securedEnabled=true)
@Configuration
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService usuarioService;
    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Override
    @Autowired
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(this.usuarioService).passwordEncoder(passwordEncoder());
    }

    @Bean("authenticationManager")
    @Override
    protected AuthenticationManager authenticationManager() throws Exception {
        return super.authenticationManager();
    }
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
        .anyRequest().authenticated()
        .and()
        .csrf().disable()
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    }

}
```



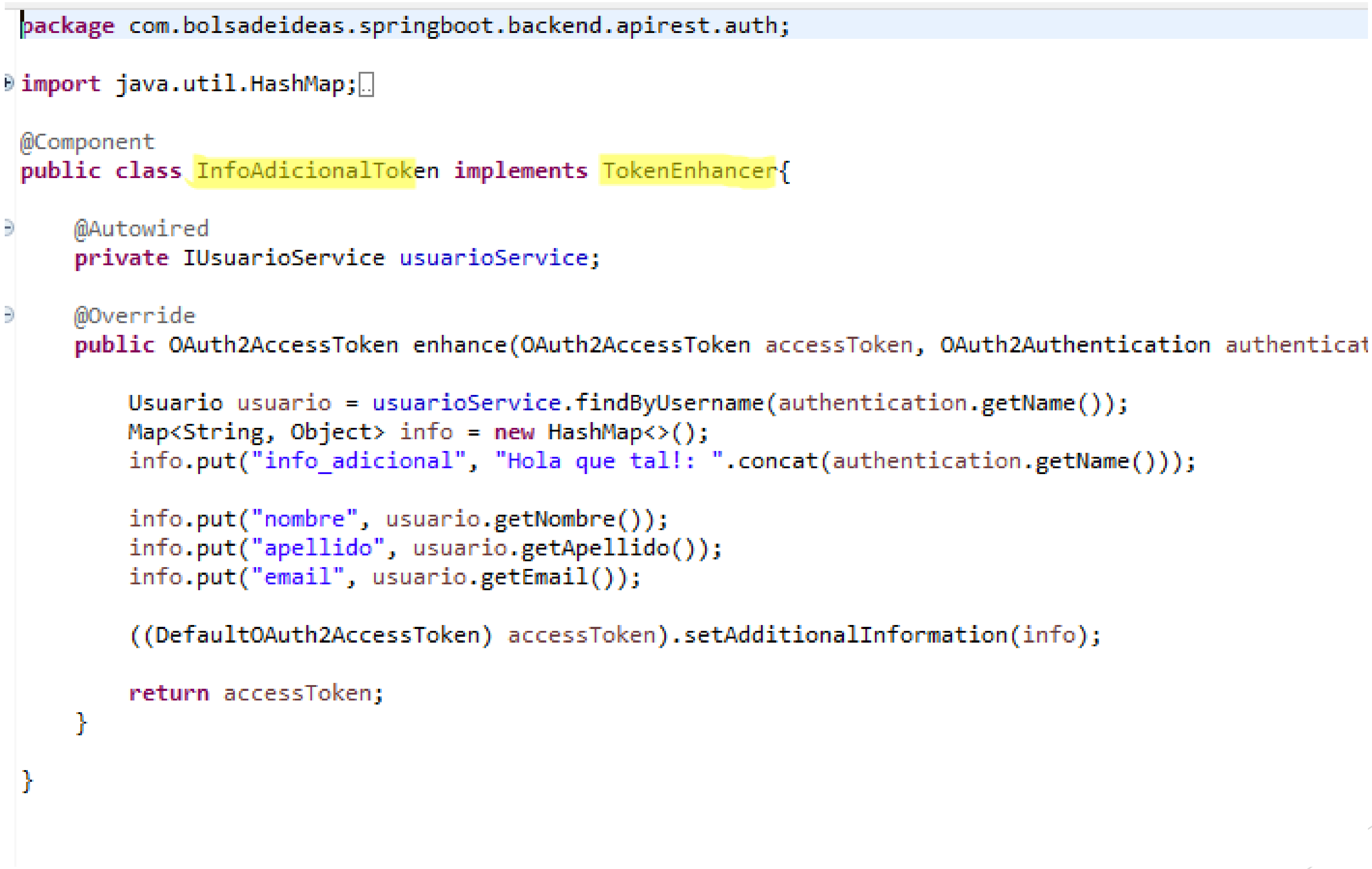
```java
package com.bolsadeideas.springboot.backend.apirest.auth;

import java.util.Arrays;

@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter{

    @Autowired
    private BCryptPasswordEncoder passwordEncoder;

    @Autowired
    @Qualifier("authenticationManager")
    private AuthenticationManager authenticationManager;

    @Autowired
    private InfoAdicionalToken infoAdicionalToken;

    @Override
    public void configure(AuthorizationServerSecurityConfigurer security) throws Exception
        security.tokenKeyAccess("permitAll()")
        .checkTokenAccess("isAuthenticated()");
    }
    @Override
    public void configure(ClientDetailsServiceConfigurer clients) throws Exception {
        clients.inMemory().withClient("angularapp")
        .secret(passwordEncoder.encode("12345"))
        .scopes("read", "write")
        .authorizedGrantTypes("password", "refresh_token")
        .accessTokenValiditySeconds(3600)
        .refreshTokenValiditySeconds(3600);
    }
```



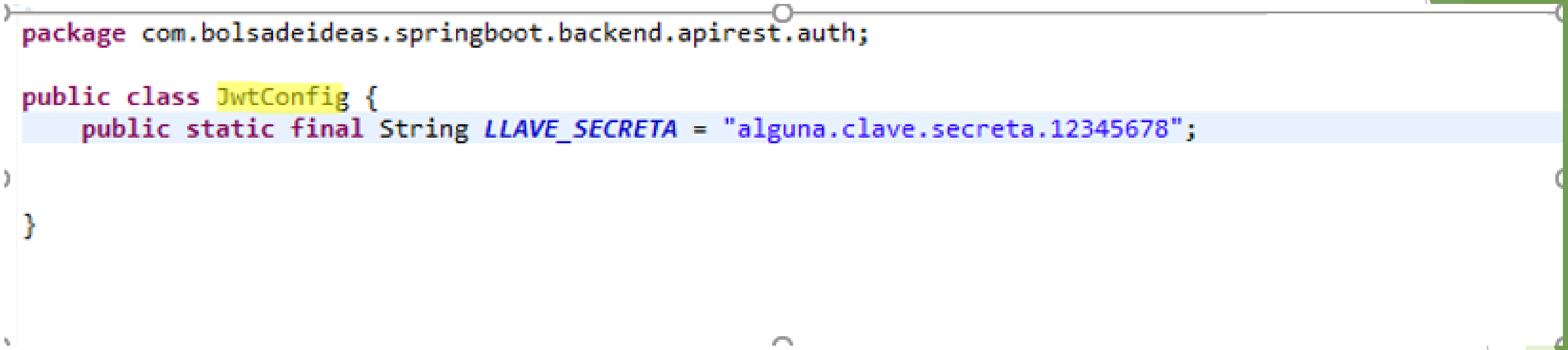
```java
    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
        tokenEnhancerChain.setTokenEnhancers(Arrays.asList(infoAdicionalToken, accessTokenConverter(
        endpoints.authenticationManager(authenticationManager)
        .tokenStore(tokenStore())
        .accessTokenConverter(accessTokenConverter())
        .tokenEnhancer(tokenEnhancerChain);
    }
    @Bean
    public JwtTokenStore tokenStore() {
        return new JwtTokenStore(accessTokenConverter());
    }
    @Bean
    public JwtAccessTokenConverter accessTokenConverter() {
        JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();
        jwtAccessTokenConverter.setSigningKey(JwtConfig.RSA_PRIVADA);
        jwtAccessTokenConverter.setVerifierKey(JwtConfig.RSA_PUBLICA);
        return jwtAccessTokenConverter;
    }


}
```



```
@Configuration
@EnableResourceServer
public class ResourceServerConfig extends ResourceServerConfigurerAdapter {

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests().antMatchers(HttpMethod.GET, "/api/clientes").permitAll()
        .anyRequest().authenticated()
        .and().cors().configurationSource(corsConfigurationSource());
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOrigins(Arrays.asList("http://localhost:4200"));
        config.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
        config.setAllowCredentials(true);
        config.setAllowedHeaders(Arrays.asList("Content-Type", "Authorization"));

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return source;
    }

    @Bean
    public FilterRegistrationBean<CorsFilter> corsFilter(){
        FilterRegistrationBean<CorsFilter> bean = new FilterRegistrationBean<CorsFilter>(new CorsFilter(corsConfigurationSource()));
        bean.setOrder(Ordered.HIGHEST_PRECEDENCE);
        return bean;
    }


}
```



```
package com.bolsadeideas.springboot.backend.apirest.auth;

import java.util.HashMap;

@Component
public class InfoAdicionalToken implements TokenEnhancer{

    @Autowired
    private IUsuarioService usuarioService;

    @Override
    public OAuth2AccessToken enhance(OAuth2AccessToken accessToken, OAuth2Authentication authenticat

        Usuario usuario = usuarioService.findByUsername(authentication.getName());
        Map<String, Object> info = new HashMap<>();
        info.put("info_adicional", "Hola que tal!: ".concat(authentication.getName()));

        info.put("nombre", usuario.getNombre());
        info.put("apellido", usuario.getApellido());
        info.put("email", usuario.getEmail());

        ((DefaultOAuth2AccessToken) accessToken).setAdditionalInformation(info);

        return accessToken;
    }

}
```



```
package com.bolsadeideas.springboot.backend.apirest.auth;

public class JwtConfig {
    public static final String LLAVE_SECRETA = "alguna.clave.secreta.12345678";


}
```