

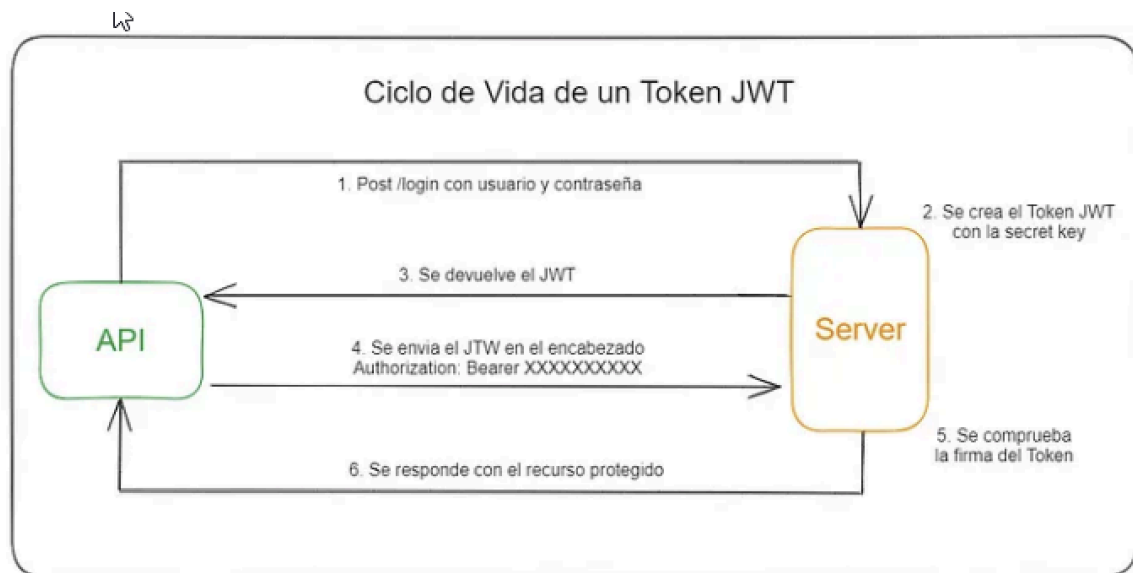
Laboratorio 2

Profundización Programación Orientada a Objetos II

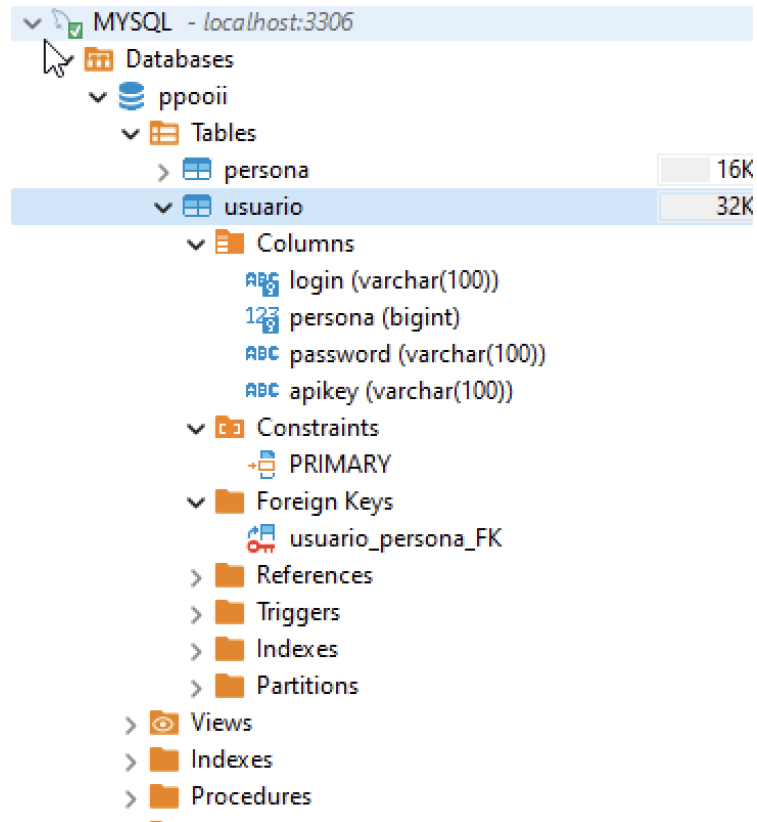
Creación de un proyecto API Rest con el framework Spring Boot, motor de base de datos MySQL, JAVA, JPA y repositorio de dependencia Maven.

A través del IDE Eclipse se realizará la creación de la aplicación Laboratorio2 con las anteriores tecnologías mencionadas y como base inicial el Laboratorio1.

El objetivo principal de este laboratorio es realizar la configuración de seguridad sprint boot y JWT, en donde para cada servicio CRUD ya creado requiera doble autenticación a través de la solicitud de token previamente generado.



1. Adicionar a la base de datos “ppooii” la tabla usuario de la siguiente forma.

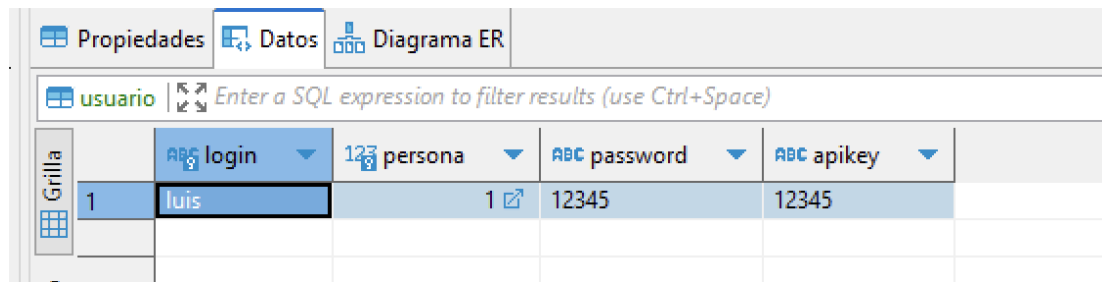


Con el siguiente scrip, teniendo como requerimiento la clave primaria debe ser compuesta por el campo login e id de la tabla persona y la creación de llave foránea con referencia a la tabla persona.

-- ppooii.usuario definition

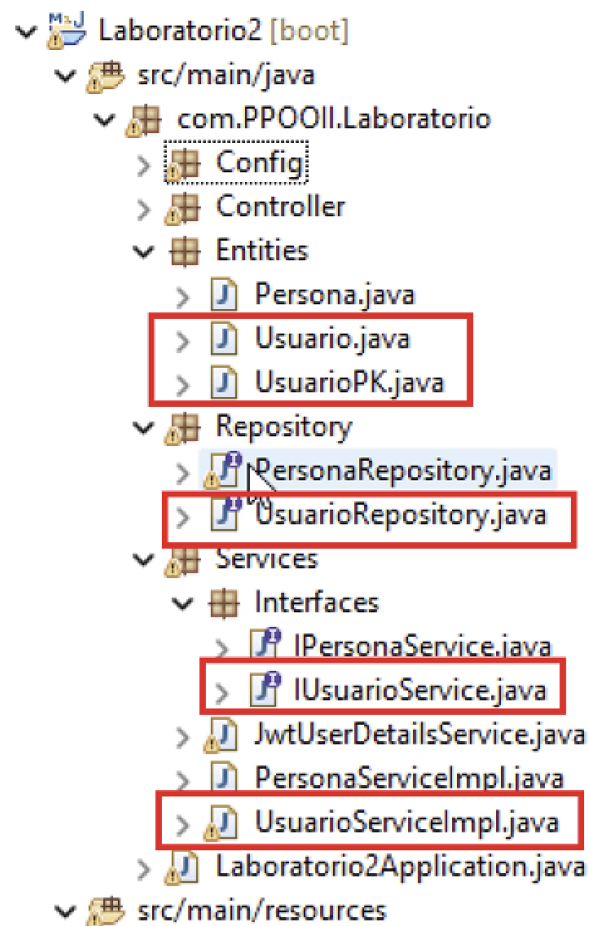
```
CREATE TABLE `usuario` (  
  `login` varchar(100) NOT NULL,  
  `persona` bigint NOT NULL,  
  `password` varchar(100) NOT NULL,  
  `apikey` varchar(100) DEFAULT NULL,  
  PRIMARY KEY (`login`, `persona`),  
  KEY `usuario_persona_FK` (`persona`),  
  CONSTRAINT `usuario_persona_FK` FOREIGN KEY (`persona`)  
  REFERENCES `persona` (`ID`)
```

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;



	login	persona	password	apikey
1	luis	1	12345	12345

2. Implementar el siguiente modelo con la entidad “usuario” creada en el punto 1, como se visualiza en la siguiente imagen.



Para esto se empezara por la creación de la entidad Usuario y UsuarioPK en cumplimiento al requerimiento.

```
package com.PPOOII.Laboratorio.Entities;

import java.io.Serializable;

import jakarta.persistence.Column;
import jakarta.persistence.Embeddable;

@Embeddable
public class UsuarioPK implements Serializable{

    private static final long serialVersionUID = 1L;

    @Column(name = "login")
    private String login;

    @Column(name = "persona")
    private int persona;

    public UsuarioPK () {}

    public UsuarioPK (String login, int persoana) {
        this.login = login;
        this.persona = persoana;
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public int getPersona() {
        return persona;
    }

    public void setPersona(int persona) {
        this.persona = persona;
    }

}
```

```

package com.PPOOII.Laboratorio.Entities;

import java.io.Serializable;
import jakarta.persistence.Column;
import jakarta.persistence.EmbeddedId;
import jakarta.persistence.Entity;
import jakarta.persistence.Table;

@Entity(name = "UsrANDPer")
@Table(schema = "PPOOII", name = "usuario")
public class Usuario implements Serializable{
    private static final long serialVersionUID = 1L;
    @EmbeddedId
    private UsuarioPK id;
    @Column(name = "password")
    private String password;
    @Column(name = "apikey")
    private String apikey;

    public Usuario() {}
    public Usuario(String login, String password) {
        this.id.setLogin(login);
        this.password = password;
    }

    public UsuarioPK getId() {
        return id;
    }
    public void setId(UsuarioPK id) {
        this.id = id;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getApikey() {
        return apikey;
    }
    public void setApikey(String apikey) {
        this.apikey = apikey;
    }
}

```

Seguido a esto se crean las interfaces donde se definirán los métodos a ser implementados en el servicio. En el Repositorio se crearán métodos basados en JPQL.

```
package com.PP00II.Laboratorio.Repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import com.PP00II.Laboratorio.Entities.Usuario;
import com.PP00II.Laboratorio.Entities.UsuarioPK;

@Repository("IUsuarioRepository")
public interface UsuarioRepository extends JpaRepository<Usuario, UsuarioPK>, CrudRepository<Usuario, UsuarioPK>{

    //Hay Métodos que JPA ya los tiene desarrollados, se pueden crear para tener
    //una manipulación más específica a la hora de usarlos en el service

    @Query("SELECT usr FROM UsrANDPer usr WHERE usr.id.login = :login AND usr.id.persona = :persona")
    public abstract Usuario getUsuarioANDPersona(@Param("login") String login, @Param("persona") int persona);

    //usr.id.login, usr.id.persona, usr.apikey, usr.password
    @Query("SELECT usr FROM UsrANDPer usr WHERE usr.id.login = :login")
    public abstract Usuario findByUsername(@Param("login") String login);
}

package com.PP00II.Laboratorio.Services.Interfaces;

import java.util.List;

import org.springframework.data.domain.Pageable;

import com.PP00II.Laboratorio.Entities.Usuario;
import com.PP00II.Laboratorio.Entities.UsuarioPK;

public interface IUsuarioService {

    //METODOS CRUD
    boolean guardar(Usuario usuario);

    boolean actualizar(Usuario usuario);

    boolean eliminar(UsuarioPK id);

    List<Usuario> consultarUsuario(Pageable pageable);

    Usuario getUsuarioById(UsuarioPK id);
}
```

Por último se crea la clase de servicio que implementa los métodos definidos en las interfaces anteriormente creadas. Adicional a esto y como objetivo de este laboratorio la implementación de seguridad JWT, para el caso de servicio se implementa el método

UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;

de la interface “UserDetailsService”

```

package com.PPOOII.Laboratorio.Services;

import java.util.ArrayList;
import java.util.List;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.data.domain.Pageable;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import com.PPOOII.Laboratorio.Entities.Usuario;
import com.PPOOII.Laboratorio.Entities.UsuarioPK;
import com.PPOOII.Laboratorio.Repository.UsuarioRepository;
import com.PPOOII.Laboratorio.Services.Interfaces.IUsuarioService;

@Service("UsuarioService")
public class UsuarioServiceImpl implements IUsuarioService, UserDetailsService {
    // ===== INYECCIÓN DE DEPENDENCIAS =====
    @Autowired
    @Qualifier("IUsuarioRepository")
    private UsuarioRepository IUsuarioRepository;
    //===== LOGS =====
    //LOGS DE ERROR
    private static final Logger logger = org.apache.logging.log4j.LogManager.getLogger(PersonaServiceImpl.class);
    //INSERT
    @Override
    public boolean guardar(Usuario usuario) {
        try {
            if (usuario == null) {
                logger.error("ERROR AGREGAR_USUARIO: EL USUARIO ES NULO!");
                return false;
            }
            else {
                IUsuarioRepository.save(usuario);
                return true;
            }
        } catch (Exception e) {
            logger.error("ERROR AGREGAR_USUARIO: EL USUARIO NO SE HA GUARDADO!");
            return false;
        }
    }
}

```

```

//UPDATE
@Override
public boolean actualizar(Usuario usuario) {
    try {
        if ((usuario == null)
            || (usuario.getId().getPersona() == 0)
            || (usuario.getId().getLogin().isEmpty())
            || (usuario.getId().getLogin() == null))
        {
            Logger.error("ERROR EDITAR_PERSONA: EL USUARIO ES NULO O EL ID ES 0 O EL LOGIN ES NULL!");
            return false;
        }
        else {
            IUsuarioRepository.save(usuario);
            return true;
        }
    } catch (Exception e) {
        Logger.error("ERROR EDITAR_PERSONA: EL USUARIO NO SE HA EDITADO!");
        return false;
    }
}

//DELETE
@Override
public boolean eliminar(UsuarioPK id) {
    try {
        if ((id.getPersona() == 0) || (id.getLogin().isEmpty()) || id.getLogin() == null) {
            Logger.error("ERROR ELIMINAR_PERSONA: EL ID DEL USUARIO ES 0 O NULL!");
            return false;
        }
        else {
            Usuario usuario = IUsuarioRepository.getUsuarioANDPersona(id.getLogin(), id.getPersona());
            IUsuarioRepository.delete(usuario);
            return true;
        }
    } catch (Exception e) {
        Logger.error("ERROR ELIMINAR_PERSONA: LA PERSONA NO SE HA ELIMINADO!");
        return false;
    }
}

//LISTA DE PRODUCTOS
@Override
public List<Usuario> consultarUsuario(Pageable pageable) {
    return IUsuarioRepository.findAll(pageable).getContent();
}

//LISTA DE PRODUCTOS
@Override
public List<Usuario> consultarUsuario(Pageable pageable) {
    return IUsuarioRepository.findAll(pageable).getContent();
}

@Override
public Usuario getUsuarioById(UsuarioPK id) {
    return IUsuarioRepository.getUsuarioANDPersona(id.getLogin(), id.getPersona());
}

//CONSULTA DE CREDENCIALES
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    System.out.println("Buscar el usuario con el repositorio y si no existe lanzar una excepcion. ");
    //Buscar el usuario por el login y basados en la consulta JPQL
    Usuario appUser = IUsuarioRepository.findByUsername(username);
    List<String> grantList = new ArrayList(); // Este objeto es usado para manejar roles de usuario. en este caso NO APLICA.
    System.out.println("Crear El objeto UserDetails que va a ir en sesion y retornarlo.");
    //Crear El objeto UserDetails que va a ir en sesion y retornarlo.
    UserDetails user = (UserDetails) new User(appUser.getId().getLogin(), appUser.getPassword(), grantList);
    System.out.println("user:[" + user + "]");
    return user;
}

```

Activar Windows

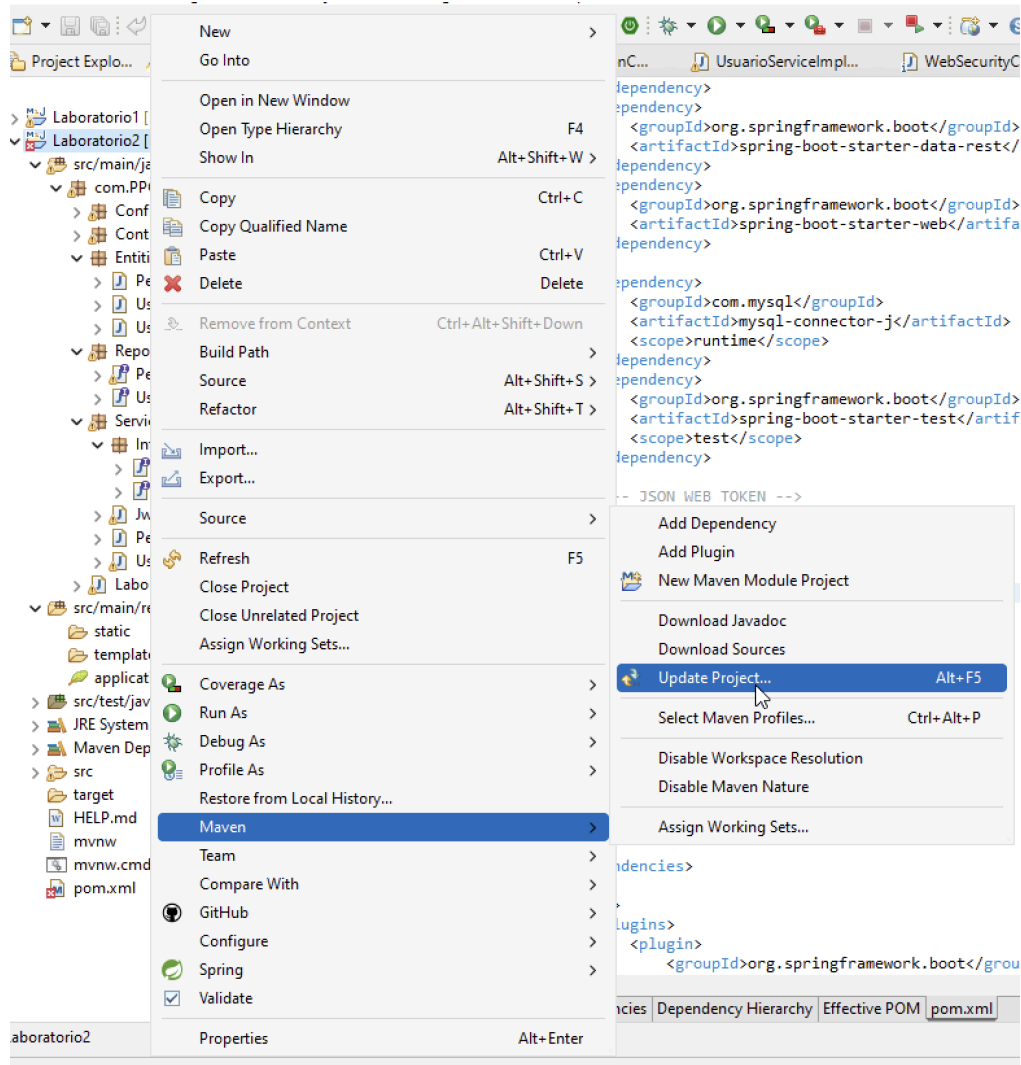
- Implementación de seguridad JWT, configurando los servicios CRUD de persona creados en Laboratorio1 autenticación Bearer Token.

Como primer paso se realizara la configuración de dependencias de seguridad spring y JWT requeridas para la implementación.

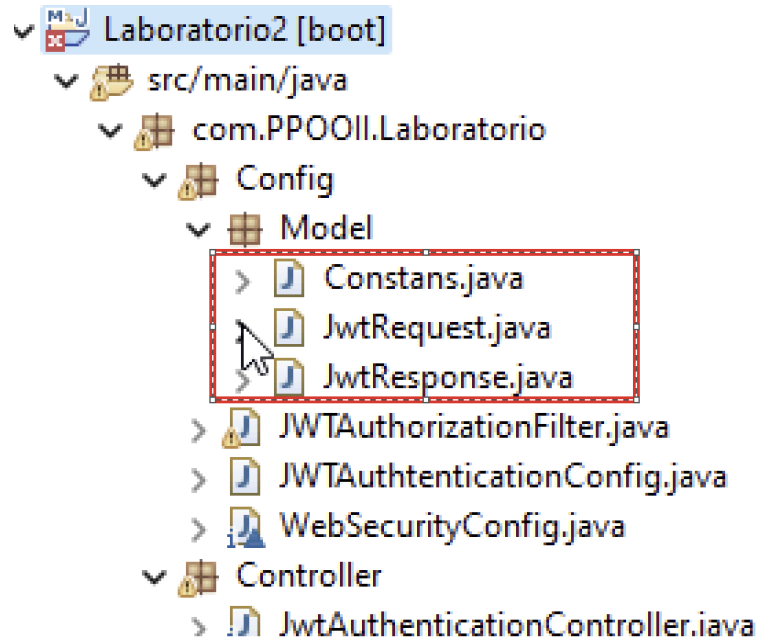

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<!-- JSON WEB TOKEN -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.5</version>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.11.5</version>
  <scope>runtime</scope>
</dependency>
```

Para completar dicha configuración se realiza Update Project del repositorio Maven.



Seguido de las configuraciones de dependencias de seguridad sprint boot y JWT, se crean objetos Response, Request y Constantes en la siguiente estructura de paquetes.



```
package com.PPOOII.Laboratorio.Config.Model;

import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;

import java.nio.charset.StandardCharsets;
import java.security.Key;

public class Constans {

    // Spring Security
    public static final String LOGIN_URL = "/authenticate";
    public static final String HEADER_AUTHORIZACION_KEY = "Authorization";
    public static final String TOKEN_BEARER_PREFIX = "Bearer ";

    // JWT
    public static final String SUPER_SECRET_KEY = "ZnJhc2VzbGFyZ2ZzcGFyYWNvbG9jYXJjb21vY2xhdmV1bnVucHJvamVjdG9kZWVtZXE";
    public static final long TOKEN_EXPIRATION_TIME = 864_000_000; // 10 day

    public static Key getSigningKeyB64(String secret) {
        byte[] keyBytes = Decoders.BASE64.decode(secret);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    public static Key getSigningKey(String secret) {
        byte[] keyBytes = secret.getBytes(StandardCharsets.UTF_8);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}
```

```
package com.PP00II.Laboratorio.Config.Model;

import java.io.Serializable;

public class JwtRequest implements Serializable{

    private static final long serialVersionUID = 1L;

    private String username;
    private String password;

    public JwtRequest () {}

    public JwtRequest (String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

}
```

```
package com.PP00II.Laboratorio.Config.Model;

import java.io.Serializable;

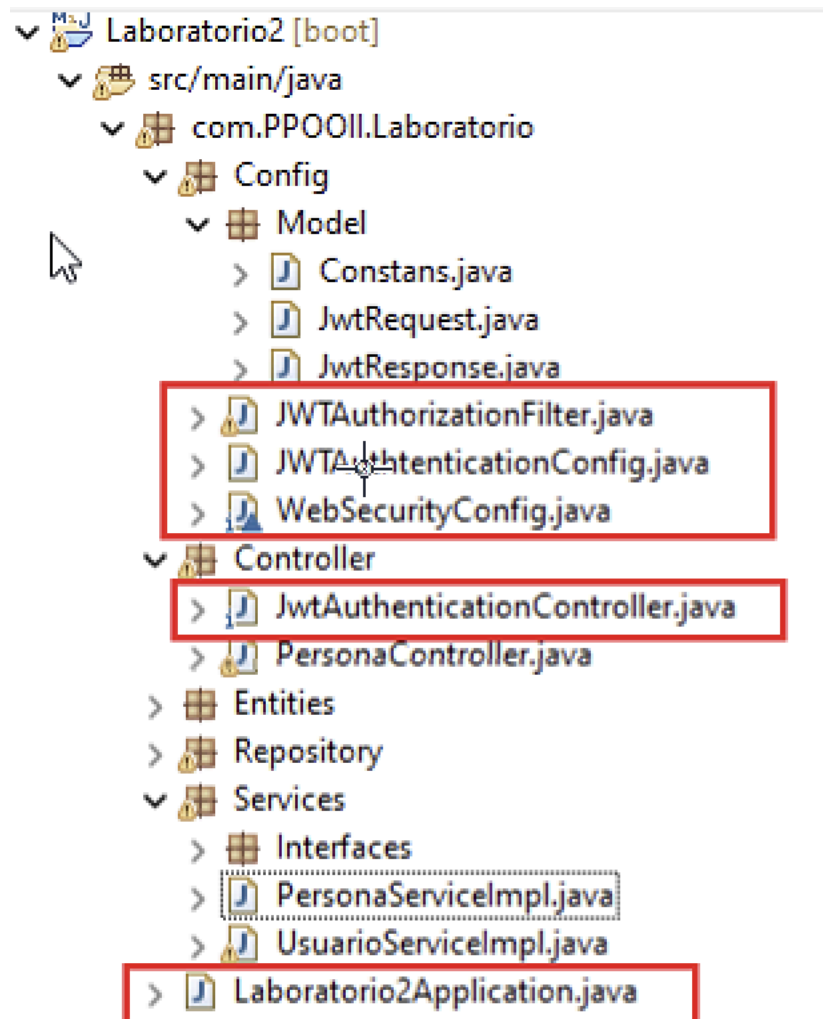
public class JwtResponse implements Serializable {

    private static final long serialVersionUID = -8091879091924046844L;
    private final String jwttoken;

    public JwtResponse(String jwttoken) {
        this.jwttoken = jwttoken;
    }

    public String getToken() {
        return this.jwttoken;
    }
}
```

A continuación se crea los objetos de configuración de seguridad spring boot y JWT.



```

package com.PPOOII.Laboratorio.Config;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;

import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;

import static com.PPOOII.Laboratorio.Config.Model.Constants.*;

@Configuration
public class JWTAuthenticationConfig {

    public String getJWTToken(String username) {
        List<GrantedAuthority> grantedAuthorities = AuthorityUtils
            .commaSeparatedStringToAuthorityList("ROLE_USER");

        String token = Jwts
            .builder()
            .setId("PPOOII_JWT")
            .setSubject(username)
            .claim("authorities",
                grantedAuthorities.stream()
                    .map(GrantedAuthority::getAuthority)
                    .collect(Collectors.toList()))
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + TOKEN_EXPIRATION_TIME))
            .signWith(getSigningKey(SUPER_SECRET_KEY), SignatureAlgorithm.HS512).compact();

        return "Bearer " + token;
    }
}

```

```

package com.PP00II.Laboratorio.Config;

import io.jsonwebtoken.*;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;
import java.util.List;
import java.util.stream.Collectors;

import static com.PP00II.Laboratorio.Config.Model.Constants.*;

@Component
public class JWTAuthorizationFilter extends OncePerRequestFilter {

    private Claims setSigningKey(HttpServletRequest request) {
        String jwtToken = request.
            getHeader(HEADER_AUTHORIZACION_KEY).
            replace(TOKEN_BEARER_PREFIX, "");

        return Jwts.parserBuilder()
            .setSigningKey(getSigningKey(SUPER_SECRET_KEY))
            .build()
            .parseClaimsJws(jwtToken)
            .getBody();
    }

    private void setAuthentication(Claims claims) {

        List<String> authorities = (List<String>) claims.get("authorities");

        UsernamePasswordAuthenticationToken auth =
            new UsernamePasswordAuthenticationToken(claims.getSubject(), null,
                authorities.stream().map(SimpleGrantedAuthority::new).collect(Collectors.toList()));

        SecurityContextHolder.getContext().setAuthentication(auth);
    }

    private boolean isValid(HttpServletRequest request, HttpServletResponse res) {
        String authenticationHeader = request.getHeader(HEADER_AUTHORIZACION_KEY);
        if (authenticationHeader == null || !authenticationHeader.startsWith(TOKEN_BEARER_PREFIX))
            return false;
        return true;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {
        try {
            if (isValid(request, response)) {
                Claims claims = setSigningKey(request);
                if (claims.get("authorities") != null) {
                    setAuthentication(claims);
                } else {
                    SecurityContextHolder.clearContext();
                }
            } else {
                SecurityContextHolder.clearContext();
            }
            filterChain.doFilter(request, response);
        } catch (ExpiredJwtException | UnsupportedJwtException | MalformedJwtException e) {
            response.setStatus(HttpServletResponse.SC_FORBIDDEN);
            response.sendError(HttpServletResponse.SC_FORBIDDEN, e.getMessage());
            return;
        }
    }
}

```



```

package com.PPO0II.Laboratorio.Config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import com.PPO0II.Laboratorio.Config.Model.Constants;

@EnableWebSecurity
@Configuration
class WebSecurityConfig{

    @Autowired
    JWTAuthorizationFilter jwtAuthorizationFilter;

    @Bean
    public SecurityFilterChain configure(HttpSecurity http) throws Exception {
        http
            .csrf((csrf) -> csrf
                .disable())
            .authorizeHttpRequests( authz -> authz
                // other configuration options
                .requestMatchers(Constants.LOGIN_URL).permitAll().requestMatchers(HttpMethod.OPTIONS, "**")
                .permitAll()
                // all other requests need to be authenticated
                .anyRequest().authenticated())
            .addFilterAfter(jwtAuthorizationFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}

```

Seguido a la creación de las tres anteriores clases, se crea el servicio en el controlador y se realiza una configuración adicional de inicialización de la aplicación.

```

package com.PP00II.Laboratorio.Controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import com.PP00II.Laboratorio.Config.JWTAuthenticationConfig;
import com.PP00II.Laboratorio.Config.Model.JwtRequest;
import com.PP00II.Laboratorio.Config.Model.JwtResponse;

@RestController
@CrossOrigin
public class JwtAuthenticationController {
    @Autowired
    JWTAuthenticationConfig jwtAuthenticationConfig;
    @Autowired
    private UserDetailsService jwtInMemoryUserDetailsService;

    @RequestMapping(
        value = "/authenticate",
        method = RequestMethod.POST,
        consumes = MediaType.APPLICATION_JSON_VALUE,
        produces = MediaType.APPLICATION_JSON_VALUE
    )
    public ResponseEntity<> createAuthenticationToken(@RequestBody JwtRequest authenticationRequest)
        throws Exception {
        System.out.println("*****");
        System.out.println("authenticationRequest.getUsername():["+authenticationRequest.getUsername()+"]");
        System.out.println("authenticationRequest.getPassword():["+authenticationRequest.getPassword()+"]");
        System.out.println("*****");
        final UserDetails userDetails = jwtInMemoryUserDetailsService
            .loadUserByUsername(authenticationRequest.getUsername());
        final String token = jwtAuthenticationConfig.getJWTToken(userDetails.getUsername());
        System.out.println("*****");
        System.out.println("token:["+token+"]");
        System.out.println("*****");
        return ResponseEntity.ok(new JwtResponse(token));
    }
}

```

Activar!

```

package com.PP00II.Laboratorio;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.security.servlet.UserDetailsServiceAutoConfiguration;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

@SpringBootApplication(exclude = {UserDetailsServiceAutoConfiguration.class})
public class Laboratorio2Application extends SpringBootServletInitializer {

    public static void main(String[] args) {
        SpringApplication.run(Laboratorio2Application.class, args);
    }
}

```

4. Correr el proyecto y probar servicios desde postman.

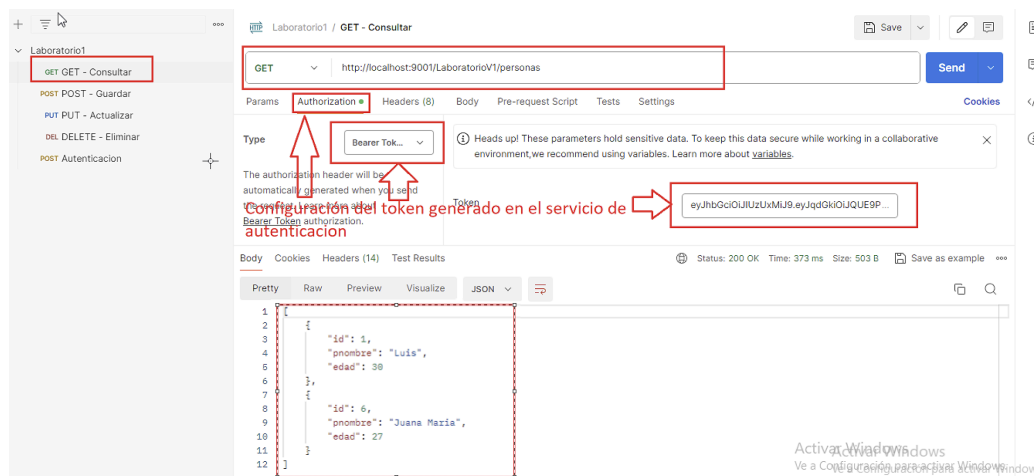
The image shows a Spring Boot application running in a terminal window. The application is named 'Laboratorio2' and is running on port 9001. The terminal output shows the application starting successfully, with various components like Spring Data, Tomcat, and Hibernate initialized.

Below the terminal, a REST client (Postman) is shown with a POST request to 'http://localhost:9001/authenticate'. The request body is a JSON object containing 'username' and 'password'.

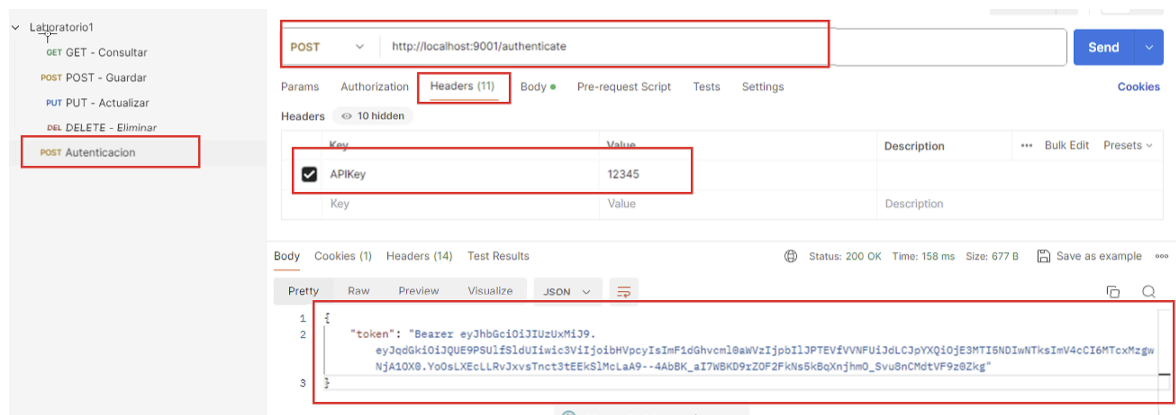
The response is a 200 OK status with a 'Bearer' token in the 'Authorization' header. The token is a long string of characters.

Annotations in red highlight the request body and the response body, with a red arrow pointing from the request body to the response body, indicating the flow of data.

Configuración de servicio con el token de seguridad.



5. Como actividad complementaria configurar los demás servicio con el token de seguridad generado y verificar su funcionamiento.
6. Implementar la validación del campo APIKey de la entidad usuario, en el servicio de autenticación enviando el valor como parámetro en el header de la petición.



Ejemplos de referencia

- <https://blog.softtek.com/es/autenticando-apis-con-spring-y-jwt>
- <https://medium.com/@espinozajge/protegiendo-tu-aplicaci%C3%B3n-web-con-spring-security-y-autenticaci%C3%B3n-basada-en-tokens-jwt-1321cbe4c4c3>