Guía de implementación de spring security en el api rest

Henry Antonio Mendoza Puerta

TABLA DE CONTENIDO

AGREGAR DEPENDENCIAS DE SPRING SECURITY Y JSON WEB TOKEN

Paso 1: Agregar las dependencias de Spring Security y JSON Web Token en el archivo pom.xml

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

<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>
```

Paso 2: Guarda los cambios. Luego selecciona la opción Load Change Maven

Paso 3: Ingresa al archivo application.properties y agrega las propiedades de jwt

```
jwt.secret=chLhMF9w3mwDutysbQxsX8x4CGwZef4mayTGSmbAG2BUsXbYFKvXrVfnPCa62P
Jxp9TuHxx4PQAS2yGUTBAPy3Dy53j8Uj2wb2AQ3nK8VLg7tUx9HCzHATEp
jwt.validity-in-seconds=2592000
```

Crear Enum Role y Entidad User

Paso 4: En el package model.entities crea el package enums. Luego agrega el enum Role

```
public enum Role {
    ADMIN,
    USER
}
```

Paso 5: En el package model.entities crea la clase User

```
import com.hampcode.bankingservice.model.entities.enums.Role;
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;

@Data
@Entity
@Table(name = "users")
@AllArgsConstructor
@NoArgsConstructor
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
    @Column(name = "full_name")
    private String fullName;
    private String email;
    private String password;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    @Enumerated(EnumType.STRING)
    private Role role;
}
```

CREA LA INTERFACE USERREPOSITORY

Paso 6: En el package repository debes crear la interface UserRepository

```java
import com.hampcode.bankingservice.model.entities.User;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findOneByEmail(String email);
    boolean existsByEmail(String email);
}
```

CREA LAS CLASES DTO

Paso 7: En el package model.dto debes crear las siguientes clases

```java
import lombok.Data;

@Data
public class AuthRequestDTO {
    private String email;
    private String password;
}
```

```java
import com.hampcode.bankingservice.model.entities.enums.Role;
import lombok.Data;

@Data
public class UserProfileDTO {
    private String firstName;
    private String lastName;
    private String fullName;
    private String email;
    private Role role;
}
```

```java
import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class AuthResponseDTO {
    private String token;
    private UserProfileDTO user;
}
```

```java
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;
import lombok.Data;

@Data
public class SignupFormDTO {
    @NotBlank
    private String firstName;
    @NotBlank
    private String lastName;
    @Email
    @NotBlank
    private String email;
    @NotNull
    @Size(min = 4)
    private String password;
    public String getFullName() {
        return firstName + " " + lastName;
    }
}
```

```java
import com.hampcode.bankingservice.model.entities.enums.Role;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import lombok.Data;

@Data
public class UserFormDTO {
    @NotBlank
    private String firstName;
    @NotBlank
    private String lastName;
    @Email
    @NotBlank
    private String email;
    @NotBlank
    private String password;
    @NotNull
    private Role role;
    public String getFullName() {
        return firstName + " " + lastName;
    }

}
```

CREAR CLASE MAPPER

Paso 8: En el package mapper debes crear la clase UserMapper

```
import com.hampcode.bankingservice.model.dto.SignupFormDTO;
import com.hampcode.bankingservice.model.dto.UserProfileDTO;
import com.hampcode.bankingservice.model.entities.User;
import lombok.AllArgsConstructor;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

@Component
@AllArgsConstructor
public class UserMapper {
    private final ModelMapper modelMapper;

    public User convertToEntity(SignupFormDTO signupFormDTO){
        return  modelMapper.map(signupFormDTO, User.class);
    }

    public UserProfileDTO convertToDTO(User user){
        return  modelMapper.map(user, UserProfileDTO.class);
    }
}
```

CREAR CLASES SERVICE

Paso 9: En el package debe crear la clase UserService

```
@AllArgsConstructor
@Service
public class UserService {

    private UserRepository userRepository;
    private PasswordEncoder passwordEncoder;
    private UserMapper userMapper;

    public UserProfileDTO signup(SignupFormDTO signupFormDTO) {
        boolean emailAlreadyExists =
userRepository.existsByEmail(signupFormDTO.getEmail());

        if (emailAlreadyExists) {
            throw new BadRequestException("El email ya está siendo usado
por otro usuario.");
        }

        User user = userMapper.convertToEntity(signupFormDTO);
```

```
user.setPassword(passwordEncoder.encode(signupFormDTO.getPassword()));
        user.setRole(Role.USER);
        user.setCreatedAt(LocalDateTime.now());

        userRepository.save(user);

        // TODO: generar un token de verificación
        // TODO: envía un email con el token de verificación
        // el estado actual sería no verificado

        return  userMapper.convertToDTO(user);
    }

    public UserProfileDTO findByEmail(String email) {
        User user = userRepository
                .findOneByEmail(email)
                .orElseThrow(ResourceNotFoundException::new);

        return userMapper.convertToDTO(user);
    }

}
```

IMPLEMENTACIÓN DE CLASE USERDETAILSSERVICEIMPL

Paso 10: Crea el package securiy. Luego en este package debes crear la clase UserDetailsServiceImpl

```
import com.hampcode.bankingservice.model.entities.User;
import com.hampcode.bankingservice.repository.UserRepository;
import lombok.RequiredArgsConstructor;
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;
@RequiredArgsConstructor
@Service
public class UserDetailsServiceImpl implements UserDetailsService {
    private final UserRepository userRepository;
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository
                .findOneByEmail(username)
                .orElseThrow(() -> new
UsernameNotFoundException(username));
        return org.springframework.security.core.userdetails.User
                .withUsername(user.getEmail())
                .password(user.getPassword())
                .roles(user.getRole().name())
                .build();
    }

}
```

IMPLEMENTAR LAS CLASES GENERAR JSON WEB TOKEN

Paso 11: Crea la clase TokenProvider en el package security

```
import io.jsonwebtoken.*;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Value;
import
org.springframework.security.authentication.UsernamePasswordAuthenticatio
nToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.stereotype.Component;

import java.security.Key;
import java.util.Collections;
import java.util.Date;
import java.util.List;

@Component
public class TokenProvider {

    @Value("${jwt.secret}")
    private String jwtSecret;

    @Value("${jwt.validity-in-seconds}")
    private long jwtValidityInSeconds;

    private Key key;
    private JwtParser jwtParser;

    @PostConstruct
    public void init() {
        key = Keys.hmacShaKeyFor(Decoders.BASE64.decode(jwtSecret));
        jwtParser = Jwts
                .parserBuilder()
                .setSigningKey(key)
                .build();
    }
```

```
public String createAccessToken(Authentication authentication) {
    String role = authentication
            .getAuthorities()
            .stream()
            .findFirst()
            .orElseThrow(RuntimeException::new)
            .getAuthority();

    return Jwts
            .builder()
            .setSubject(authentication.getName())
            .claim("role", role)
            .signWith(key, SignatureAlgorithm.HS512)
            .setExpiration(new Date(System.currentTimeMillis() +
jwtValidityInSeconds * 1000))
            .compact();
}
```

```
public Authentication getAuthentication(String token) {
        Claims claims = jwtParser.parseClaimsJws(token).getBody();

        String role = claims.get("role").toString();
        List<GrantedAuthority> authorities =
Collections.singletonList(new SimpleGrantedAuthority(role));

        User principal = new User(claims.getSubject(), "", authorities);
        return new UsernamePasswordAuthenticationToken(principal, token,
authorities);
    }

    public boolean validateToken(String token) {
        try {
            jwtParser.parseClaimsJws(token);
            return true;
        } catch (JwtException e) {
            return false;
        }
    }

}
```

Paso 12: Crea la clase JWTFilter en el package security

```
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.http.HttpServletRequest;
import org.springframework.http.HttpHeaders;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.GenericFilterBean;

import java.io.IOException;

public class JWTFilter extends GenericFilterBean {
    private final TokenProvider tokenProvider;
    public JWTFilter(TokenProvider tokenProvider) {
        this.tokenProvider = tokenProvider;
    }

    @Override
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain) throws IOException,
ServletException {
        HttpServletRequest httpServletRequest = (HttpServletRequest)
request;

        String bearerToken =
httpServletRequest.getHeader(HttpHeaders.AUTHORIZATION);

        if (StringUtils.hasText(bearerToken) &&
bearerToken.startsWith("Bearer ")) {
            String token = bearerToken.substring(7);

            Authentication authentication =
tokenProvider.getAuthentication(token);

SecurityContextHolder.getContext().setAuthentication(authentication);
        }
        chain.doFilter(request, response);
    }
}
```

Paso 13: Crea la clase JWTConfigurer en el package security

```
import
org.springframework.security.config.annotation.SecurityConfigurerAdapter;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.DefaultSecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthentic
ationFilter;

public class JWTConfigurer extends
SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {

    private final TokenProvider tokenProvider;

    public JWTConfigurer(TokenProvider tokenProvider) {
        this.tokenProvider = tokenProvider;
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        JWTFilter jwtFilter = new JWTFilter(tokenProvider);
        http.addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class);
    }
}
```

IMPLEMENTA LAS CLASES DE CONFIGURACIÓN

Paso 14: Crea la clase SecurityConfig en el package config

```
import com.hampcode.bankingservice.security.JWTConfigurer;
import com.hampcode.bankingservice.security.TokenProvider;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configurers.AbstractHt
tpConfigurer;
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;

@RequiredArgsConstructor
@Configuration
public class SecurityConfig {
    private final TokenProvider tokenProvider;
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        http
                .cors(Customizer.withDefaults())
                .csrf(AbstractHttpConfigurer::disable)
                .authorizeHttpRequests((authz) -> authz
                        .requestMatchers("/users/signup",
"/auth/token").permitAll() // Permitir sin autenticación
                        // Incluir rutas adicionales
                        .requestMatchers("/api/v1/swagger-ui/**",
"/v3/api-docs/**", "/swagger-ui.html", "/swagger-ui/**",
"/webjars/**").permitAll()
                        .anyRequest()
                        .authenticated() // Cualquier otra solicitud
requiere autenticación

                )
                .sessionManagement(h ->
h.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .with(new JWTConfigurer(tokenProvider),
Customizer.withDefaults());

        return http.build();
    }
```

```
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Habilitamos CORS

Paso 15: En el package config crea la clase CorsConfig

```
import jakarta.servlet.*;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.core.Ordered;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
import java.io.IOException;

@Component
@Order(Ordered.HIGHEST_PRECEDENCE)
public class CorsConfig implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        // TODO Auto-generated method stub

    }
  @Override
    public void doFilter(ServletRequest req, ServletResponse res,
FilterChain chain)
            throws IOException, ServletException {
        HttpServletResponse response = (HttpServletResponse) res;
        HttpServletRequest request = (HttpServletRequest) req;

        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "DELETE, GET,
OPTIONS, PATCH, POST, PUT");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers", "x-requested-
with, authorization, Content-Type, Authorization, credential, X-XSRF-
TOKEN");

        if ("OPTIONS".equalsIgnoreCase(request.getMethod())) {
            response.setStatus(HttpServletResponse.SC_OK);
        } else {
            chain.doFilter(req, res);
        }
        // chain.doFilter(req, res);
    }

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
    }
}
```

INTEGRAMOS SECURITY EN LA CLASE OPENAPICONFIG

Paso 16: En el metodo myOpenAPI deben agrega los objetos de security

```
// Configuración de seguridad JWT
SecurityScheme securityScheme = new SecurityScheme()
        .type(SecurityScheme.Type.HTTP)
        .scheme("bearer")
        .bearerFormat("JWT")
        .name("JWT Authentication");
Components components = new Components()
        .addSecuritySchemes("bearerAuth", securityScheme);

// Requerimiento de seguridad para utilizar en las operaciones
SecurityRequirement securityRequirement = new
SecurityRequirement().addList("bearerAuth");


return new OpenAPI()
        .info(info)
        .servers(List.of(devServer))
        .addSecurityItem(securityRequirement)
        .components(components);
```

CREANDO CLASES CONTROLLER

Paso 17: En el package controller crea la clase UserController

```
@RequiredArgsConstructor
@RestController
@RequestMapping("/users")
public class UserController {

    private final UserService userService;

    @ResponseStatus(HttpStatus.CREATED)
    @PostMapping("/signup")
    public UserProfileDTO signup(@RequestBody @Validated SignupFormDTO
signupFormDTO) {
        return userService.signup(signupFormDTO);
    }
}
```

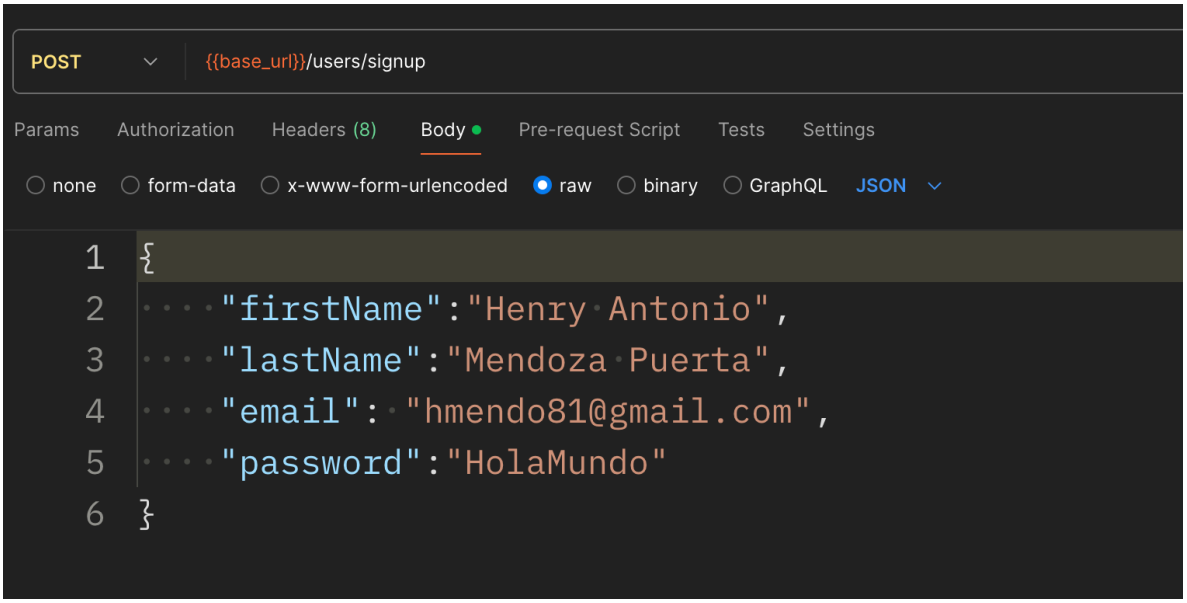
Paso 18: En el package controller crea la clase JWTController

```
@RequiredArgsConstructor
@RestController
@RequestMapping("/auth")
public class JWTController {

    private final AuthenticationManagerBuilder
authenticationManagerBuilder;
    private final TokenProvider tokenProvider;
    private final UserService userService;

    @PostMapping("/token")
    public ResponseEntity<AuthResponseDTO> getAccessToken(@RequestBody
AuthRequestDTO authRequest) {
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(
                authRequest.getEmail(),
                authRequest.getPassword()
        );
        Authentication authentication =
authenticationManagerBuilder.getObject().authenticate(authenticationToken
);

SecurityContextHolder.getContext().setAuthentication(authentication);

        String accessToken =
tokenProvider.createAccessToken(authentication);
        UserProfileDTO userProfileDTO =
userService.findByEmail(authRequest.getEmail());
        AuthResponseDTO authResponse = new AuthResponseDTO(accessToken,
userProfileDTO);
```

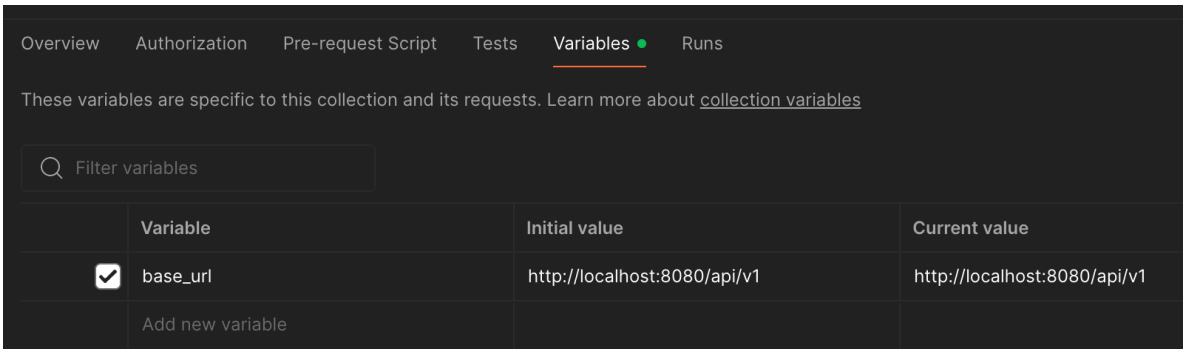
```
        return ResponseEntity
                .ok()
                .header(HttpHeaders.AUTHORIZATION, "Bearer " +
accessToken)
                .body(authResponse);
    }
}
```

REALIZANDO PRUEBAS DE INICIO DE SESSION Y REGISTRO DE USUARIO CON POSTMAN

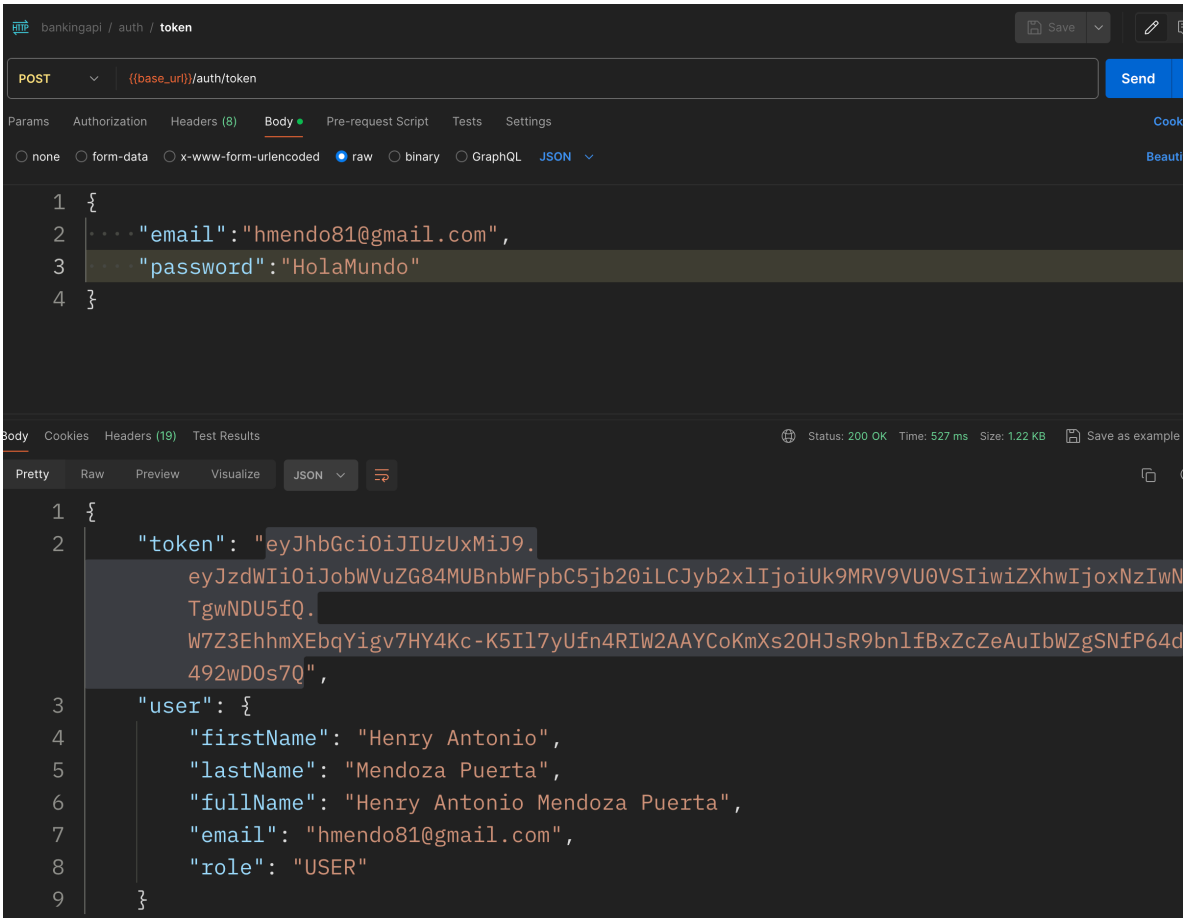
Paso 19: Crea el siguiente request registro de usuario (signup)



Recuerda que la variable base_url debe ser creado en la colección



Paso 20: Crea el siguiente request inicio de sesion (auth/token)



Uitlizando token JWT en los request

Paso 21: Cuando ejecutas el request auth/token se genera un token el cual debes de utilizar para probar los demas request. En la siguiente imagen utilizamos en el request de listar cuentas bancarias por usuario