Configuración para la validación de tokens para asegurar endpoints y mantener la sesión del usuario sin mostrar datos sensibles.

Paso 1: Configurar Dependencias

Utilizando Gradle, se agrgan las líneas: build.gradle.kts

```
dependencias {
   implementation("io.jsonwebtoken:jjwt-api:0.11.2")
   implementation("io.jsonwebtoken:jjwt-impl:0.11.2")
   implementation("io.jsonwebtoken:jjwt-jackson:0.11.2") // Para manejar JSON
}
```

Paso 2: Clase JwtUtil

Se crea la clase JwtUtil para generar y validar tokens JWT, un ejemplo podrías ser

```
package com.gamerly.projectgamerly.security

import io.jsonwebtoken.Claims
import io.jsonwebtoken.Jwts
import io.jsonwebtoken.SignatureAlgorithm
import java.util.*

class JwtUtil {
   private val secretKey = "mySecretKey" // Debe ser segura y secreta
   private val expirationTime = 86400000 // 24 horas en milisegundos

   fun generateToken(username: String): String {
      val claims = Jwts.claims().setSubject(username)
      val now = Date()
      val expiryDate = Date(now.time + expirationTime)

      return Jwts.builder()
         .setClaims(claims)
         .setIssuedAt(now)
         .setExpiration(expiryDate)
         .signWith(SignatureAlgorithm.HS512, secretKey)
         .compact()
   }

   fun validateToken(token: String): Boolean {
      return try {
         val claims = getClaimsFromToken(token)
```

```
            !claims.expiration.before(Date())
        } catch (e: Exception) {
            false
        }
    }

    fun getUsernameFromToken(token: String): String {
        return getClaimsFromToken(token).subject
    }

    private fun getClaimsFromToken(token: String): Claims {
        return Jwts.parser()
            .setSigningKey(secretKey)
            .parseClaimsJws(token)
            .body
    }
}
```

Paso 3: Crear una clase Servicio para la autenticación

```
package com.gamerly.projectgamerly.service

import com.gamerly.projectgamerly.dto.UsuarioLoginDTO
import com.gamerly.projectgamerly.security.JwtUtil
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.security.authentication.AuthenticationManager
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken
import org.springframework.security.core.userdetails.UserDetailsService
import org.springframework.stereotype.Service

@Service
class AuthService {

    @Autowired
    private lateinit var authenticationManager: AuthenticationManager

    @Autowired
    private lateinit var jwtUtil: JwtUtil

    @Autowired
    private lateinit var userDetailsService: UserDetailsService
```

```
    fun login(loginDTO: UsuarioLoginDTO): String {
        val authentication = authenticationManager.authenticate(
            UsernamePasswordAuthenticationToken(loginDTO.email, loginDTO.password)
        )

        val userDetails = userDetailsService.loadUserByUsername(loginDTO.email)
        return jwtUtil.generateToken(userDetails.username)
    }
}
```

Paso 4: Configurar Seguridad de Spring

```
package com.gamerly.projectgamerly.config

import com.gamerly.projectgamerly.security.JwtFilter
import com.gamerly.projectgamerly.service.CustomUserDetailsService
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.builders.AuthenticationManagerBuilder
import org.springframework.security.config.annotation.web.builders.HttpSecurity
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter
import org.springframework.security.config.http.SessionCreationPolicy
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter

@Configuration
@EnableWebSecurity
class SecurityConfig : WebSecurityConfigurerAdapter() {

    @Autowired
    private lateinit var customUserDetailsService: CustomUserDetailsService

    @Autowired
    private lateinit var jwtFilter: JwtFilter

    override fun configure(auth: AuthenticationManagerBuilder) {
        auth.userDetailsService(customUserDetailsService)
    }
```

```
    @Bean
    override fun authenticationManagerBean(): AuthenticationManager {
        return super.authenticationManagerBean()
    }

    override fun configure(http: HttpSecurity) {
        http.csrf().disable()
            .authorizeRequests().antMatchers("/login").permitAll()
            .anyRequest().authenticated()
            .and()
            .exceptionHandling()
            .and()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

        http.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter::class.java)
    }
}
```

Paso 5: Crear un filtro JWT para interceptar las solicitudes y validar los tokens

```
package com.gamerly.projectgamerly.security

import com.gamerly.projectgamerly.service.CustomUserDetailsService
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.security.core.context.SecurityContextHolder
import org.springframework.security.core.userdetails.UserDetails
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource
import org.springframework.stereotype.Component
import org.springframework.web.filter.OncePerRequestFilter
import javax.servlet.FilterChain
import javax.servlet.http.HttpServletRequest
import javax.servlet.http.HttpServletResponse

@Component
class JwtFilter : OncePerRequestFilter() {

    @Autowired
    private lateinit var jwtUtil: JwtUtil

    @Autowired
    private lateinit var userDetailsService: CustomUserDetailsService
```

```
    override fun doFilterInternal(request: HttpServletRequest, response: HttpServletResponse,
filterChain: FilterChain) {
        val authorizationHeader = request.getHeader("Authorization")
        var username: String? = null
        var token: String? = null

        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
            token = authorizationHeader.substring(7)
            username = jwtUtil.getUsernameFromToken(token)
        }

        if (username != null && SecurityContextHolder.getContext().authentication == null) {
            val userDetails: UserDetails = userDetailsService.loadUserByUsername(username)

            if (jwtUtil.validateToken(token!!)) {
                val authenticationToken = UsernamePasswordAuthenticationToken(
                    userDetails, null, userDetails.authorities
                )
                authenticationToken.details = WebAuthenticationDetailsSource().buildDetails(request)
                SecurityContextHolder.getContext().authentication = authenticationToken
            }
        }
        filterChain.doFilter(request, response)
    }
}
```

Paso 6: Crear controlador para manejar las solicitudes de inicio de sesión y devolver el token JWT

```
package com.gamerly.projectgamerly.controller

import com.gamerly.projectgamerly.dto.UsuarioLoginDTO
import com.gamerly.projectgamerly.service.AuthService
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.http.ResponseEntity
import org.springframework.web.bind.annotation.*

@RestController
@RequestMapping("/api")
class AuthController {
```

```
    @Autowired
    private lateinit var authService: AuthService

    @PostMapping("/login")
    fun login(@RequestBody loginDTO: UsuarioLoginDTO): ResponseEntity<String> {
        val token = authService.login(loginDTO)
        return ResponseEntity.ok(token)
    }
}
```

Cuando el usuario inicie sesión recibirá un token jwt que se usará para acceder a los enpoints protegidos.

1- Solicitar Token:
   Envía un POST a la api con el cuerpo que contenga usaurio y password.
   Recibe el token jwt en la respuesta.

2- Accede a endpoints protegidos
   Añade el token jwt en el encabezado “Autirization” de las solicitudes.