Desafío 03

Tercer desafío practico BookApi

Estudiantes:

Fernando Josué Anzora Aquino AA222744

César Daniel Chevez Zepeda CZ230902

Nombre del docente:

Miguel Alejandro Meléndez Martínez

Materia:

Desarrollo de Aplicaciones con Web Frameworks

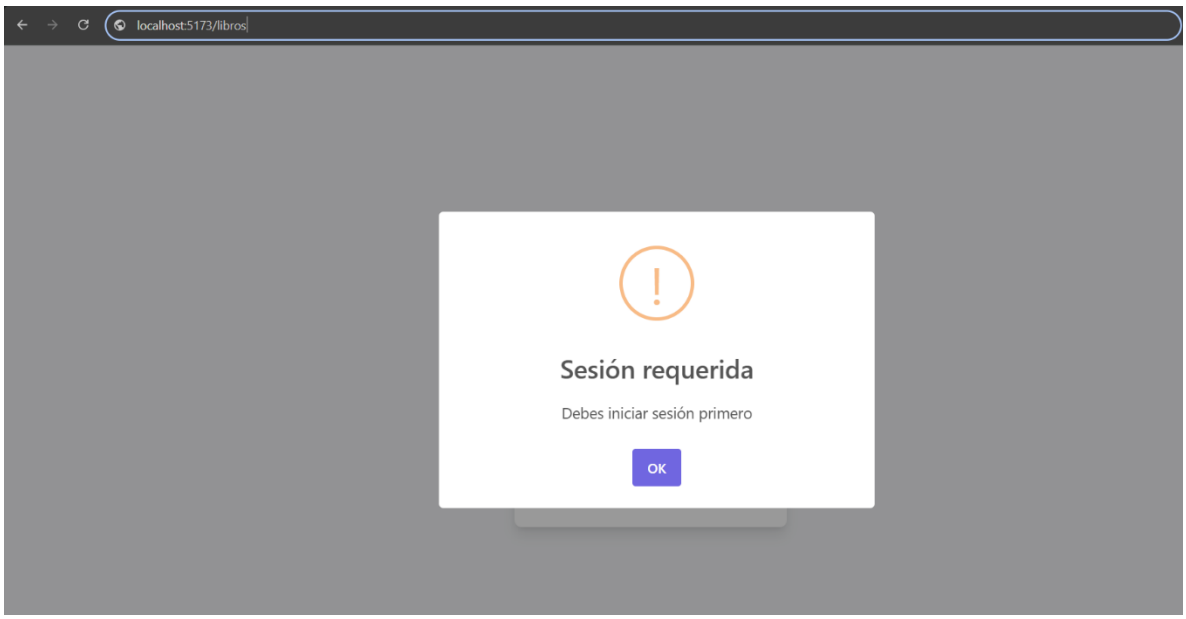
Grupo de laboratorio: 01

Link GitHub:

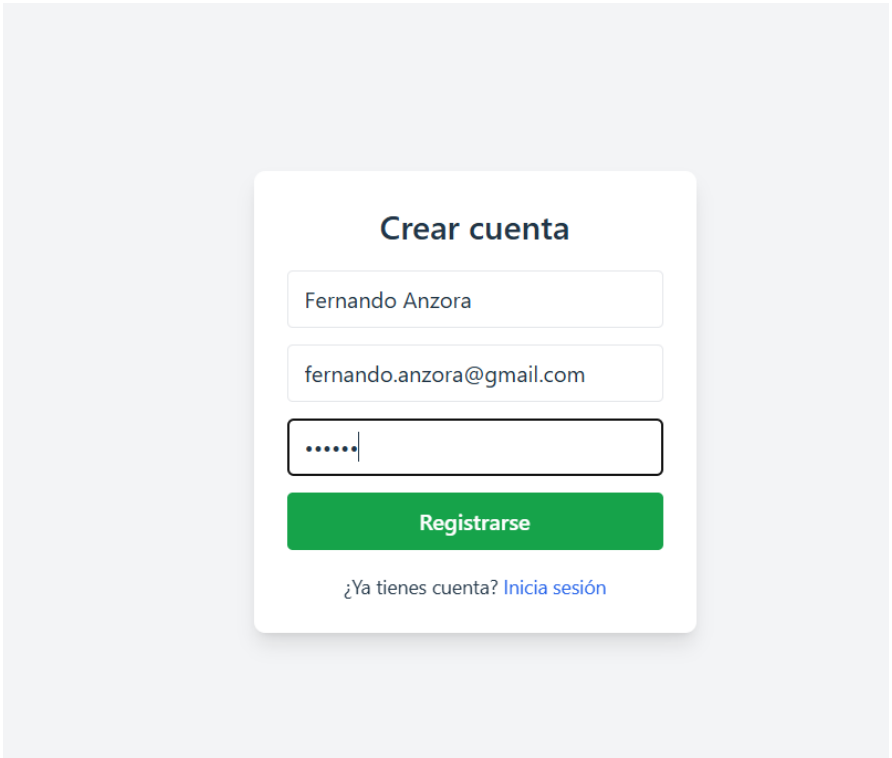
https://github.com/Fernando-Anzora/DWF-01L-Desafio3.git

Inicio de sesión y registro de usuarios:

Autenticación requerida:



Registro de usuario:

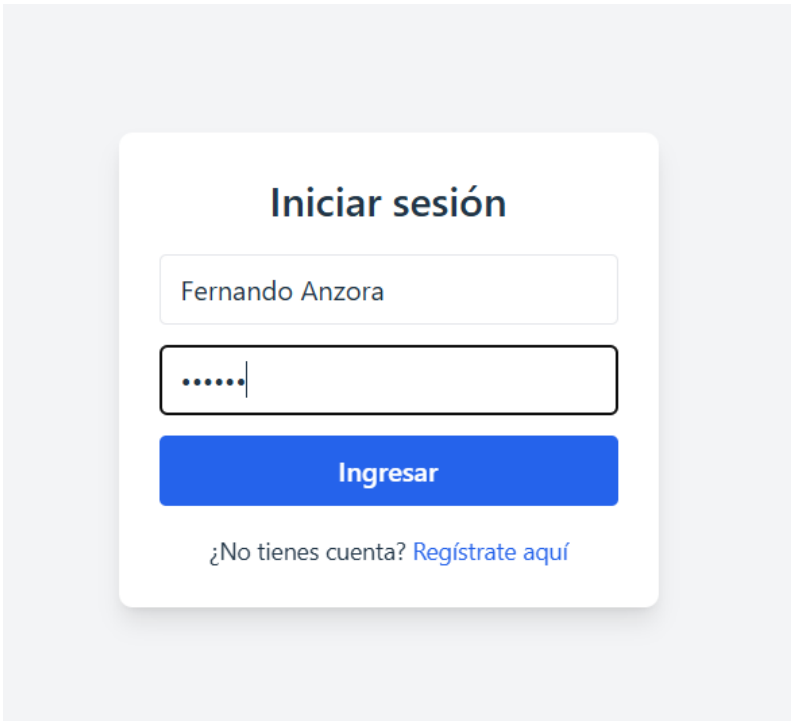


Éxito

Cuenta creada correctamente

OK

Inicio de sesión:

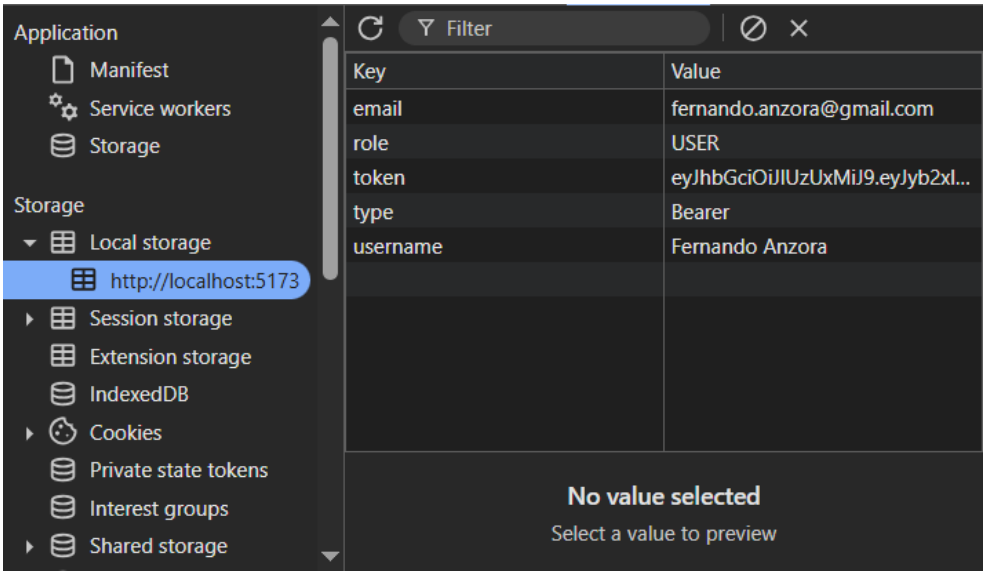


Bienvenido

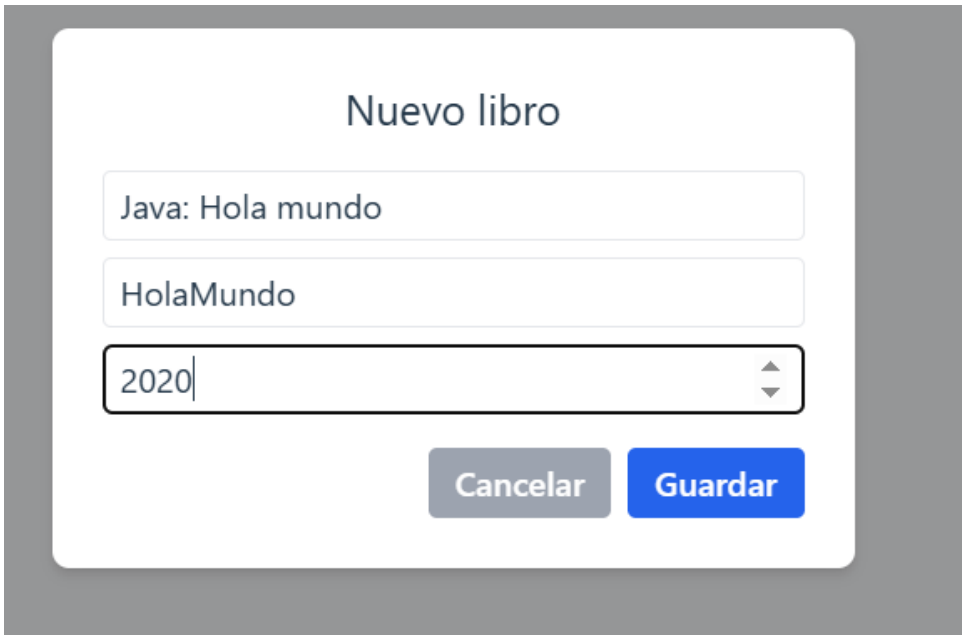
Inicio de sesión exitoso

OK

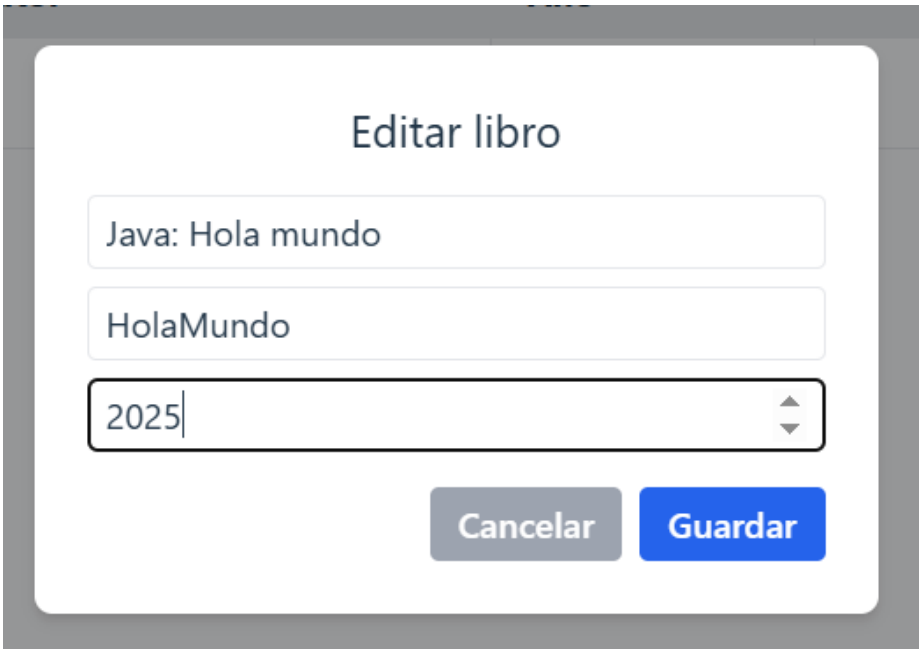
Datos del usuario guardados en local storage:



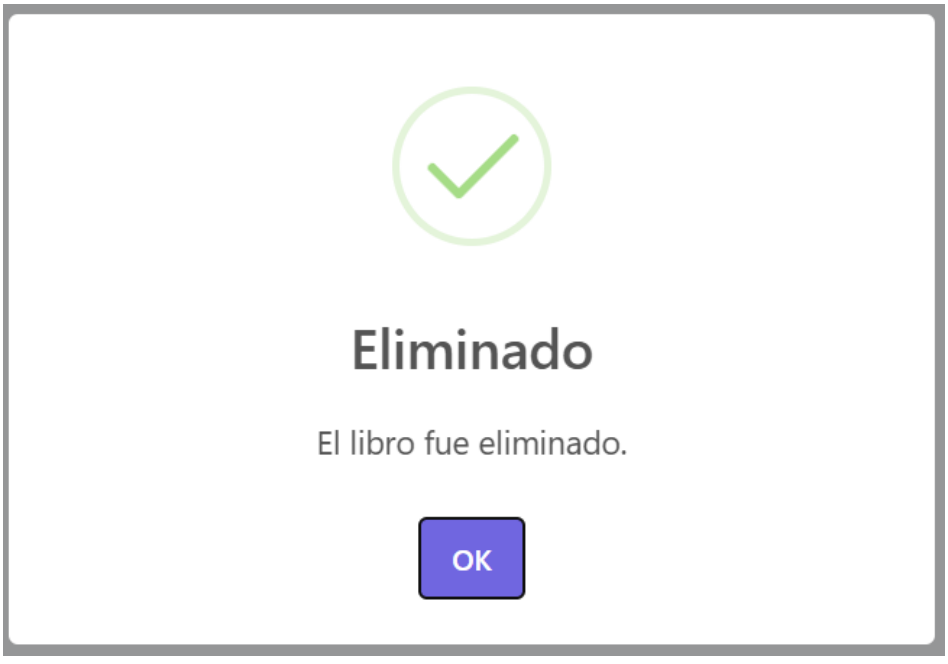
Nuevo libro:



Editar libro



Eliminar libro



Lista Completa de EndPoints con Postman:

https://fernandoanzora2004-8928736.postman.co/workspace/Fernando-Anzora's-Workspace~af18db97-31e2-4017-845f-b820cadc38e2/collection/48828671-11dccbbe-44f9-4cf7-9d80-f34bce6898c3?action=share&creator=48828671

flujo JWT implementado:

Componentes principales involucrados

AuthController: expone endpoints públicos (/auth/login, /auth/register) para generar tokens.

AuthService: realiza la autenticación (verifica credenciales) y delega la generación del token.

JwtUtil: utilitario para crear, firmar, parsear y validar tokens (incluye expiración y extracción del username/claims).

JwtAuthenticationFilter: filtro que intercepta cada petición entrante, extrae el token del encabezado, lo valida y, si es válido, coloca la Authentication en el SecurityContext.

UserDetailsServiceImpl: carga datos del usuario (username, password, roles) para crear el UserDetails requerido por Spring Security.

SecurityConfig: configura Spring Security: registra el JwtAuthenticationFilter (antes de UsernamePasswordAuthenticationFilter), marca la política de sesión como STATELESS y define reglas de autorización.

Flujo

Registro: el cliente llama /api/v1/auth/register con los datos del usuario; el servidor guarda el usuario (contraseña hasheada) y devuelve confirmación.

Login: el cliente envía POST /api/v1/auth/login con username y password.

Autenticación: AuthService (o AuthenticationManager) verifica las credenciales usando UserDetailsServiceImpl y PasswordEncoder.

Generación del JWT: si la autenticación es correcta, JwtUtil crea un token firmado (por ejemplo HS256) que contiene al menos el sub (username), fecha de emisión y expiración, y opcionalmente roles/claims.

Respuesta al cliente: el servidor devuelve el token en LoginResponseDto (o en el body). El cliente lo guarda (p.ej. localStorage o memoria segura).

Peticiones subsecuentes: el cliente añade el encabezado Authorization: Bearer <token> en cada petición a endpoints protegidos.

Filtro de validación: JwtAuthenticationFilter se ejecuta en cada petición, extrae el token del encabezado, invoca JwtUtil.validateToken(token) y, si es válido, obtiene el username y carga UserDetails.

Contexto de seguridad: el filtro crea un UsernamePasswordAuthenticationToken con los roles y lo guarda en SecurityContextHolder.getContext().setAuthentication(...). Spring Security usa esto para autorizar acceso a controladores/recursos.

Expiración y re-autenticación: si el token está expirado o inválido, la petición no tendrá Authentication válida y se retornará 401 Unauthorized. El cliente debe re-login (o usar un mecanismo de refresh si está implementado).