

# Investigación JWT

Realizado por:

Dilan Esteban Rey Sepulveda -2190397

Silvia Alejandra Cárdenas Santos - 2210102

1. Construir un login completo usando Spring Boot para el Backend, asignando la seguridad a través de JWT y consumirlo con Java Script Genérico usando FETCH u otra tecnología a la aplicada por el profesor (XMLHttpRequest).
  - Anexar una explicación de la forma como se desarrolla la solución
  - Compartir el proyecto backend y el frontend de la solución

## Desarrollo

### Backend

Para comenzar con el desarrollo del backend, lo primero que debemos hacer es agregar las dependencias necesarias en el archivo pom.xml. Estas dependencias permitirán la implementación de la autenticación mediante JSON Web Tokens (JWT), proporcionando una capa de seguridad a nuestra aplicación.

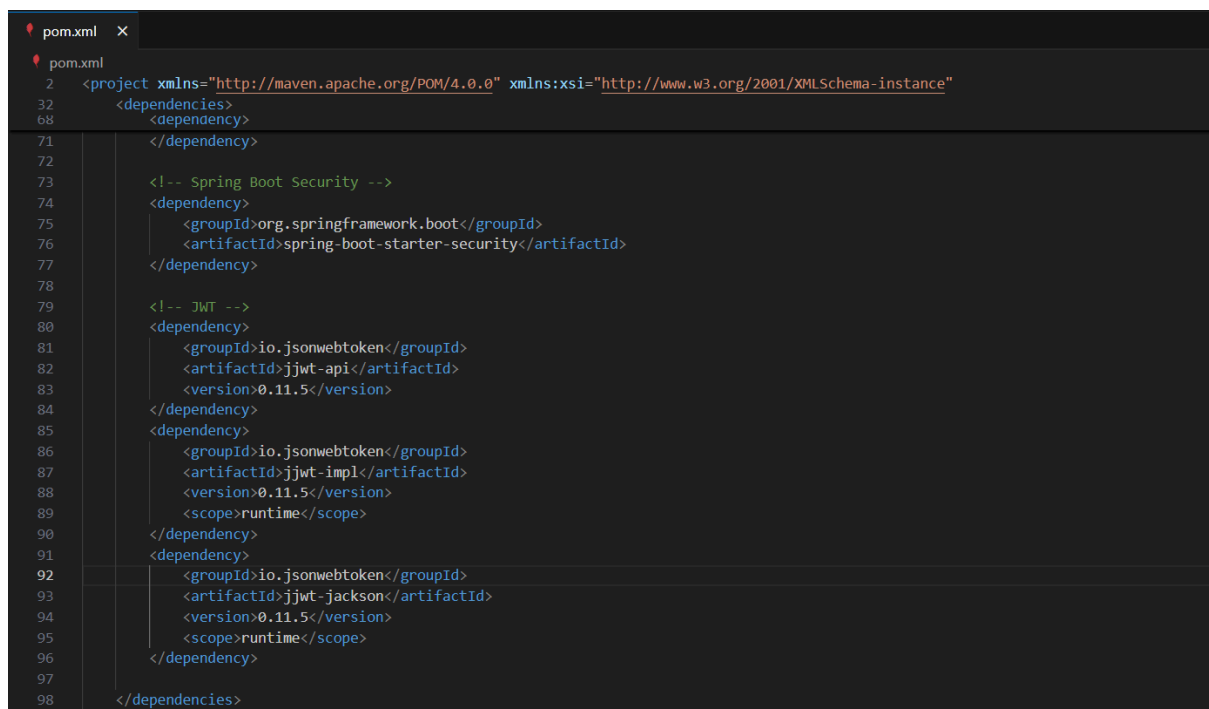


Imagen 1. Dependencias en el pom.xml

El siguiente paso es la creación del modelo Usuario, el cual representará a los usuarios en la base de datos. Este modelo define los atributos esenciales y se encarga de mapear los datos almacenados en MySQL.

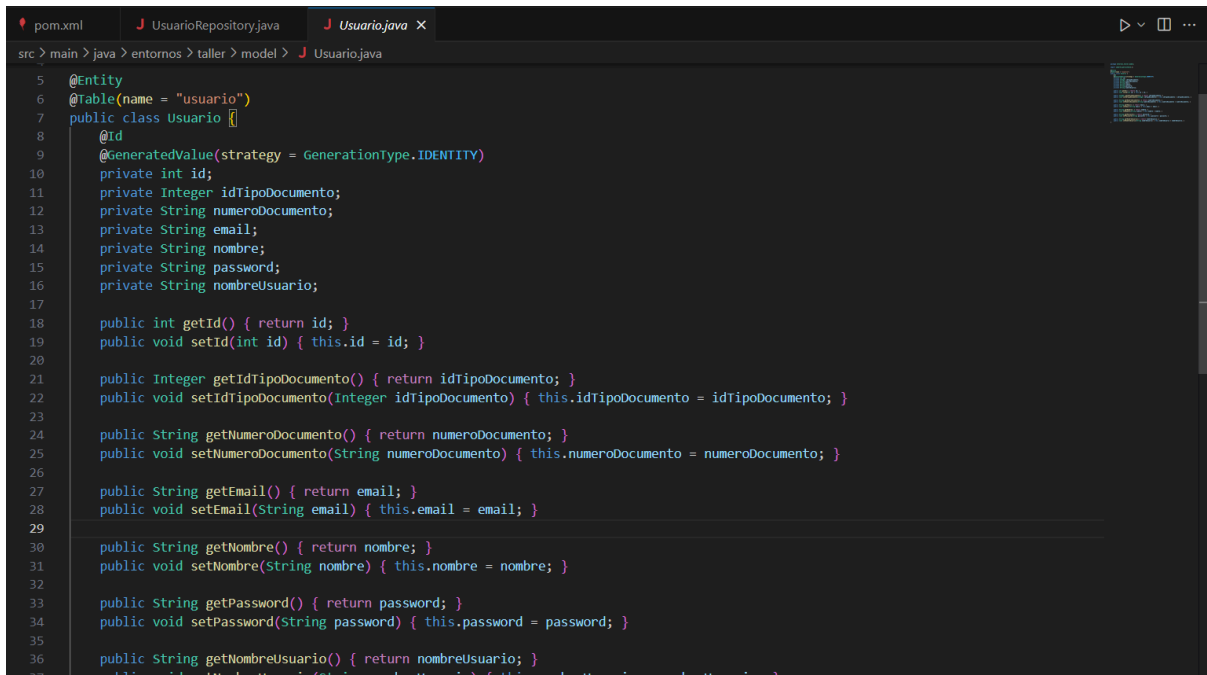


Imagen 2. Modelo Usuario.java

Una vez definido el modelo, es necesario crear el repositorio correspondiente. La clase UsuarioRepository.java se encargará de la interacción con la base de datos, permitiendo realizar operaciones como la búsqueda de usuarios por diferentes criterios.

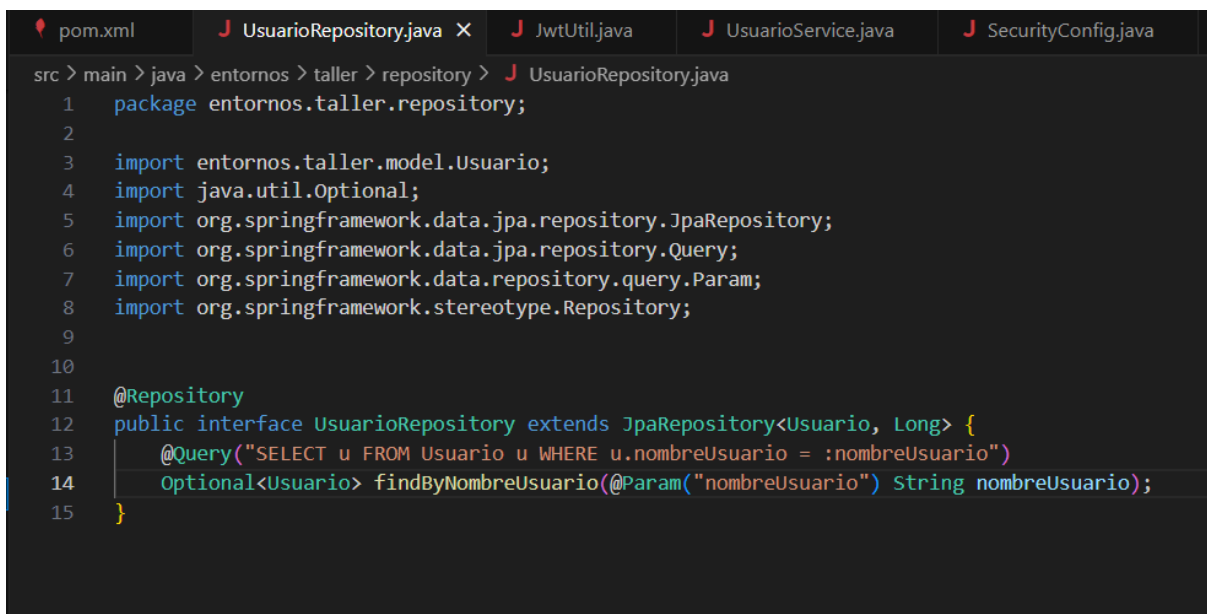
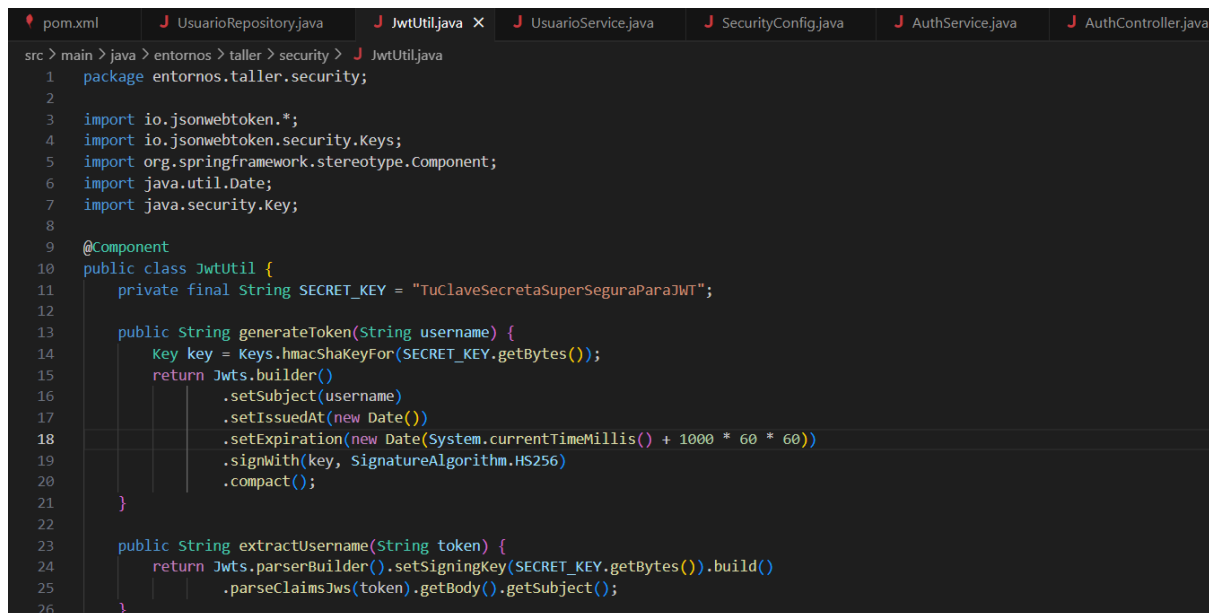


Imagen 3. Repositorio UsuarioRepository.java

A continuación, se implementa la clase JWTUtil dentro del directorio entornos.taller.security. Esta clase es fundamental, ya que contiene la lógica para generar y validar tokens JWT, los cuales tendrán una duración de 1 hora. Mediante

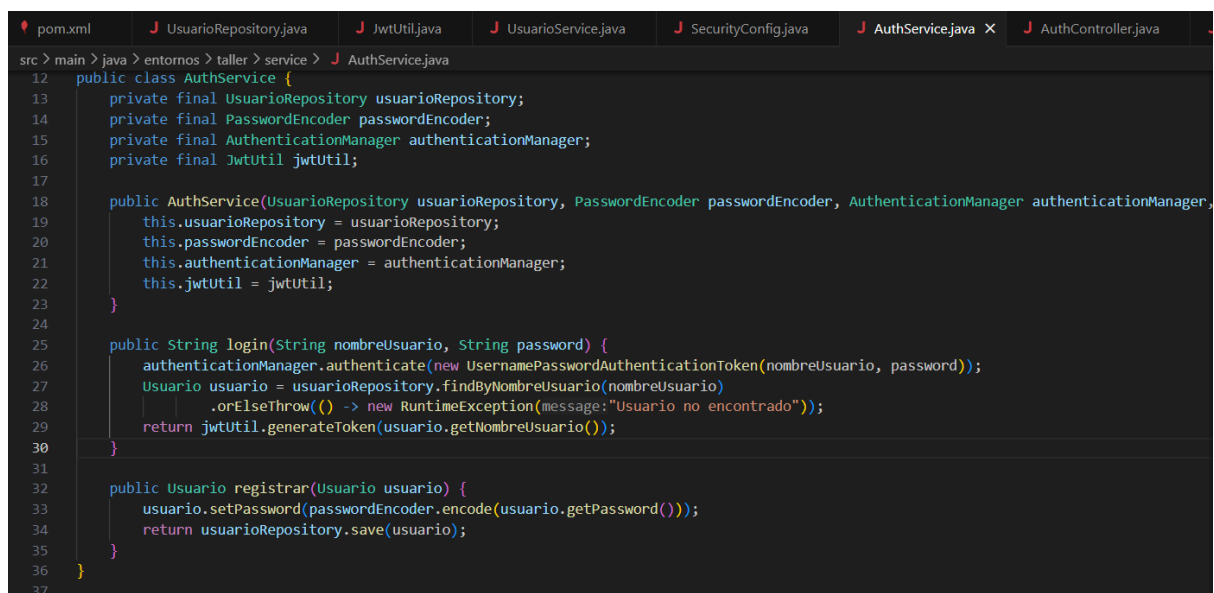
esta utilidad, podremos garantizar que las solicitudes a los endpoints protegidos sean realizadas por usuarios autenticados.



```
src > main > java > entornos > taller > security > JwtUtil.java
1 package entornos.taller.security;
2
3 import io.jsonwebtoken.*;
4 import io.jsonwebtoken.security.Keys;
5 import org.springframework.stereotype.Component;
6 import java.util.Date;
7 import java.security.Key;
8
9 @Component
10 public class JwtUtil {
11     private final String SECRET_KEY = "TuClaveSecretaSuperSeguraParaJWT";
12
13     public String generateToken(String username) {
14         Key key = Keys.hmacShaKeyFor(SECRET_KEY.getBytes());
15         return Jwts.builder()
16             .setSubject(username)
17             .setIssuedAt(new Date())
18             .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 60))
19             .signWith(key, SignatureAlgorithm.HS256)
20             .compact();
21     }
22
23     public String extractUsername(String token) {
24         return Jwts.parserBuilder().setSigningKey(SECRET_KEY.getBytes()).build()
25             .parseClaimsJws(token).getBody().getSubject();
26     }
27 }
```

Imagen 4. Utilidad de JWT

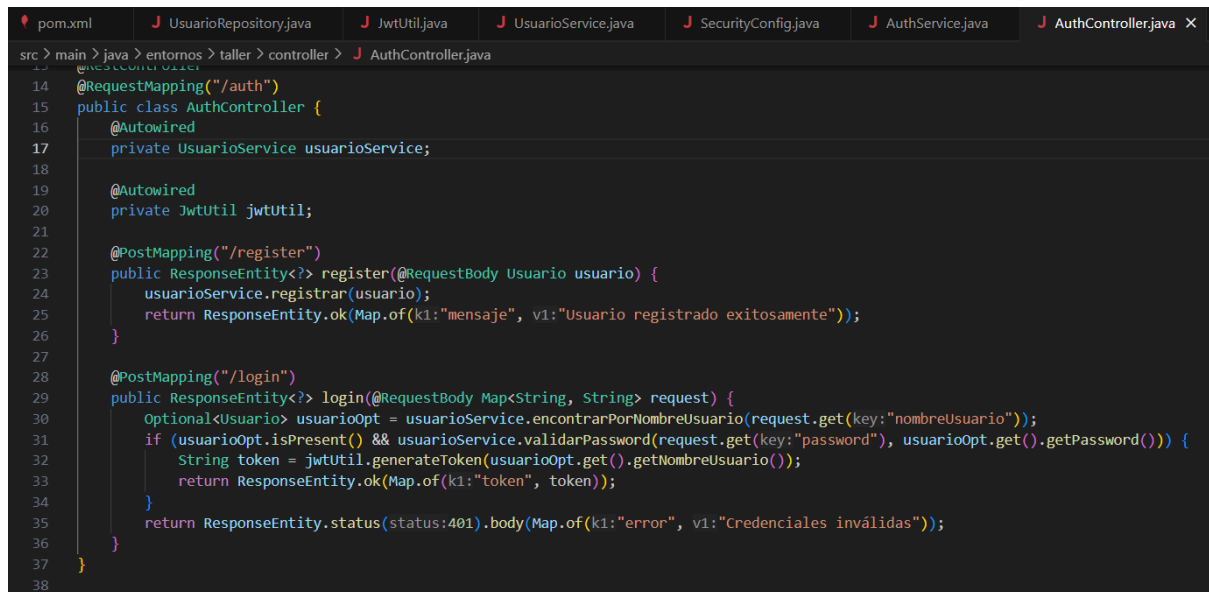
El siguiente paso es la creación del servicio de autenticación. Este servicio es el encargado de gestionar tanto el inicio de sesión como el registro de nuevos usuarios. Aquí se implementa la lógica para validar credenciales, generar tokens JWT y almacenar nuevos usuarios en la base de datos.



```
src > main > java > entornos > taller > service > AuthService.java
12 public class AuthService {
13     private final UsuarioRepository usuarioRepository;
14     private final PasswordEncoder passwordEncoder;
15     private final AuthenticationManager authenticationManager;
16     private final JwtUtil jwtUtil;
17
18     public AuthService(UsuarioRepository usuarioRepository, PasswordEncoder passwordEncoder, AuthenticationManager authenticationManager,
19         this.usuarioRepository = usuarioRepository;
20         this.passwordEncoder = passwordEncoder;
21         this.authenticationManager = authenticationManager;
22         this.jwtUtil = jwtUtil;
23     }
24
25     public String login(String nombreUsuario, String password) {
26         authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(nombreUsuario, password));
27         Usuario usuario = usuarioRepository.findByNombreUsuario(nombreUsuario)
28             .orElseThrow(() -> new RuntimeException(message:"Usuario no encontrado"));
29         return jwtUtil.generateToken(usuario.getNombreUsuario());
30     }
31
32     public Usuario registrar(Usuario usuario) {
33         usuario.setPassword(passwordEncoder.encode(usuario.getPassword()));
34         return usuarioRepository.save(usuario);
35     }
36 }
37 }
```

Imagen 5. Servicio de autenticación

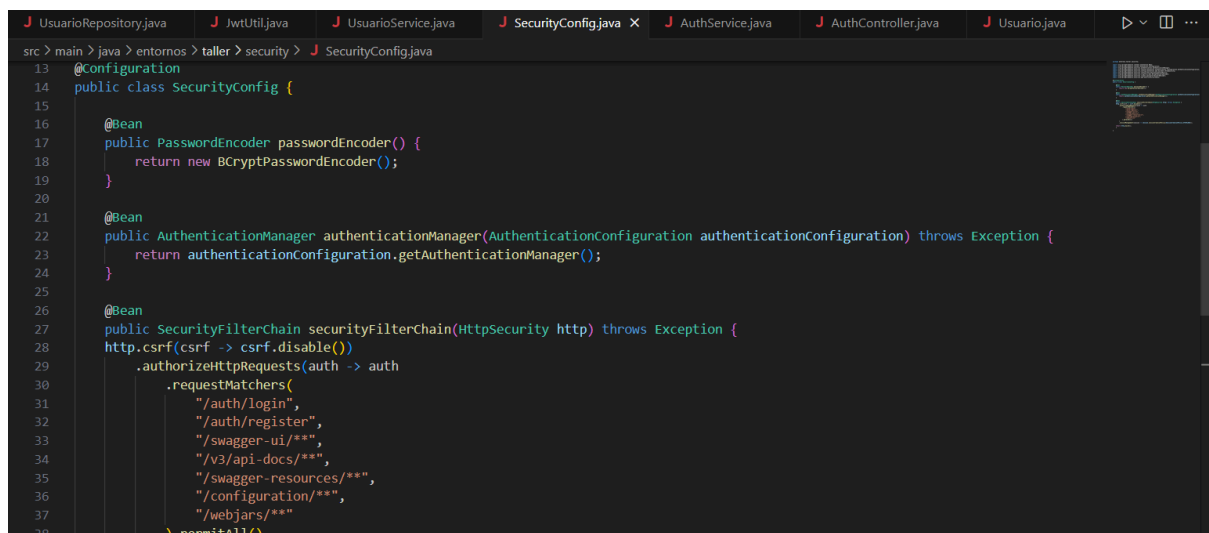
Después, configuramos el controlador de autenticación. A través de este controlador se exponen los endpoints necesarios para que los usuarios puedan registrarse e iniciar sesión.



```
src > main > java > entornos > taller > controller > J AuthController.java
13 @RestController
14 @RequestMapping("/auth")
15 public class AuthController {
16     @Autowired
17     private UsuarioService usuarioService;
18
19     @Autowired
20     private JwtUtil jwtUtil;
21
22     @PostMapping("/register")
23     public ResponseEntity<> register(@RequestBody Usuario usuario) {
24         usuarioService.registrar(usuario);
25         return ResponseEntity.ok(Map.of(k1:"mensaje", v1:"Usuario registrado exitosamente"));
26     }
27
28     @PostMapping("/login")
29     public ResponseEntity<> login(@RequestBody Map<String, String> request) {
30         Optional<Usuario> usuarioOpt = usuarioService.encontrarPorNombreUsuario(request.get(key:"nombreUsuario"));
31         if (usuarioOpt.isPresent() && usuarioService.validarPassword(request.get(key:"password"), usuarioOpt.get().getPassword())) {
32             String token = jwtUtil.generateToken(usuarioOpt.get().getNombreUsuario());
33             return ResponseEntity.ok(Map.of(k1:"token", token));
34         }
35         return ResponseEntity.status(status:401).body(Map.of(k1:"error", v1:"Credenciales inválidas"));
36     }
37 }
38
```

Imagen 6. Controlador de autenticación

Finalmente es necesario configurar la seguridad de la aplicación. En este punto se definen las reglas de acceso a los distintos endpoints, permitiendo el acceso público a las rutas de login y registro, así como a la documentación generada por Swagger.



```
J UsuarioRepository.java J JwtUtil.java J UsuarioService.java J SecurityConfig.java X J AuthService.java J AuthController.java J Usuario.java
src > main > java > entornos > taller > security > J SecurityConfig.java
13 @Configuration
14 public class SecurityConfig {
15
16     @Bean
17     public PasswordEncoder passwordEncoder() {
18         return new BCryptPasswordEncoder();
19     }
20
21     @Bean
22     public AuthenticationManager authenticationManager(AuthenticationConfiguration authenticationConfiguration) throws Exception {
23         return authenticationConfiguration.getAuthenticationManager();
24     }
25
26     @Bean
27     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
28         http.csrf(csrf -> csrf.disable())
29             .authorizeHttpRequests(auth -> auth
30                 .requestMatchers(
31                     "/auth/login",
32                     "/auth/register",
33                     "/swagger-ui/**",
34                     "/v3/api-docs/**",
35                     "/swagger-resources/**",
36                     "/configuration/**",
37                     "/webjars/**"
38                 ).permitAll()
39             );
40     }
41 }
```

Imagen 7. Configuración de seguridad

Antes de proceder con la implementación del frontend, es importante verificar que la autenticación mediante JWT funciona correctamente. Para ello utilizamos Postman y realizamos una prueba de registro de usuario, si la configuración es correcta, el usuario se almacenará en la base de datos exitosamente.

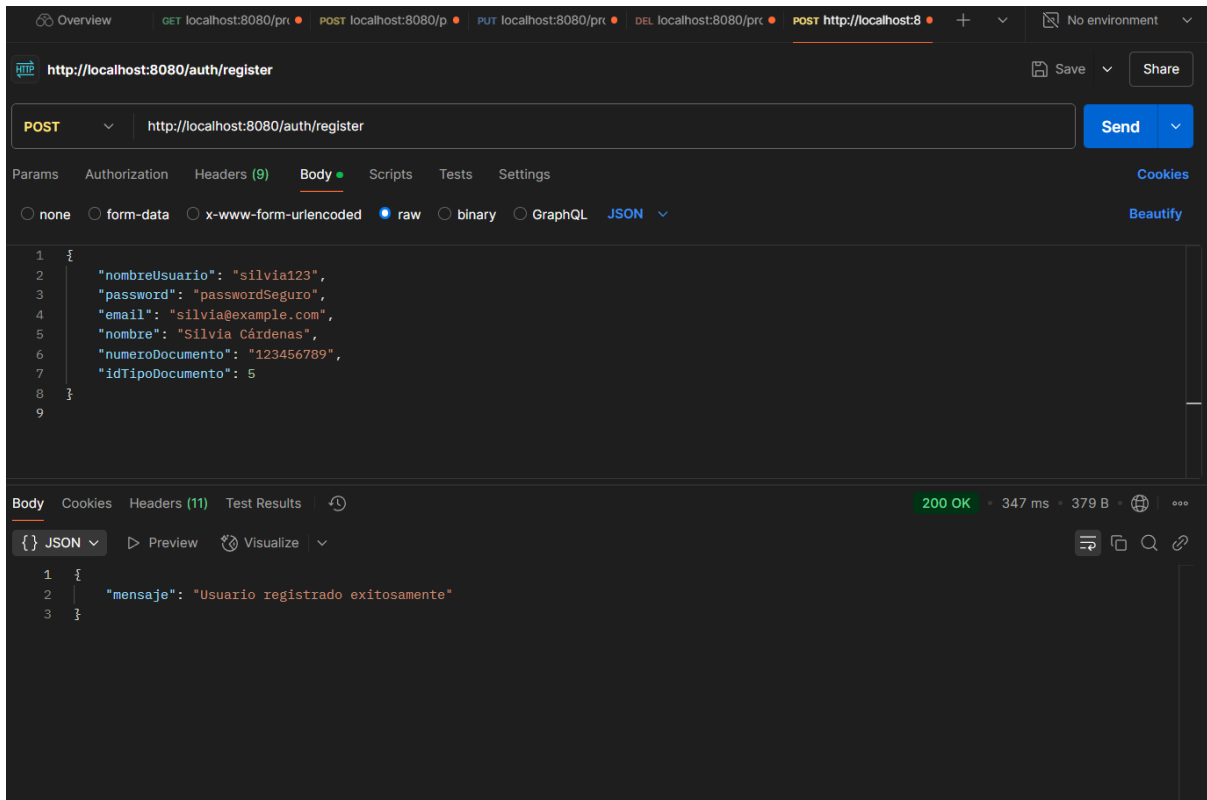


Imagen 8. Registro usando Postman

The screenshot shows a MySQL database interface with a table named 'usuario'. The table has 7 columns: id, idTipoDocumento, numeroDocumento, email, nombre, password, and nombreUsuario. The table contains 19 rows of data, including a row for the user 'silvia123'.

id	idTipoDocumento	numeroDocumento	email	nombre	password	nombreUsuario
1	5	63124561	pedro@laforesta.com	Pedro Alonso Paquetiva	123	admin
2	5	9123455	pepe@gmail.com	Pepe	123456	cliente
4	6	890123445	palonso@floresta.edu.co	Pedro Alonso Paquetiva	pepito	pepe
9	5	91239999	JAnono@uis.edu.co	Jorge Arturo Cifuentes Velez	nono123456	elnono
10	5	91234190	llopez@gmail.com	Luis Pedro López	123456	llopez
11	5	63124561	Sninio@uis.edu.co	Sandra Milena Niño	ssndra123	sninio
12	5	1098123451	Falarcon@gmail.com	Fernando José Alarcón Suarez	12345	FAlarcon
13	6	65081136415	lcalderon@gmail.com	Luis Fernando Calderón	12311	lcalderon
14	5	90213778	jsuarez@gmail.com	Jorge Ernesto Suarez	123456	jsuarez
19	5	123456789	silvia@example.com	Silvia Cárdenas	\$2a\$10\$z9d64V6DH7aOw17NmR9BeU3MEnktj...	silvia123
	NULL	NULL	NULL	NULL	NULL	NULL

Imagen 9. Base de datos MySQL

De la misma forma, se prueba el login con un usuario registrado. Al ser las credenciales correctas, el sistema responde con un token JWT válido, lo que confirma que la autenticación ha sido implementada correctamente.

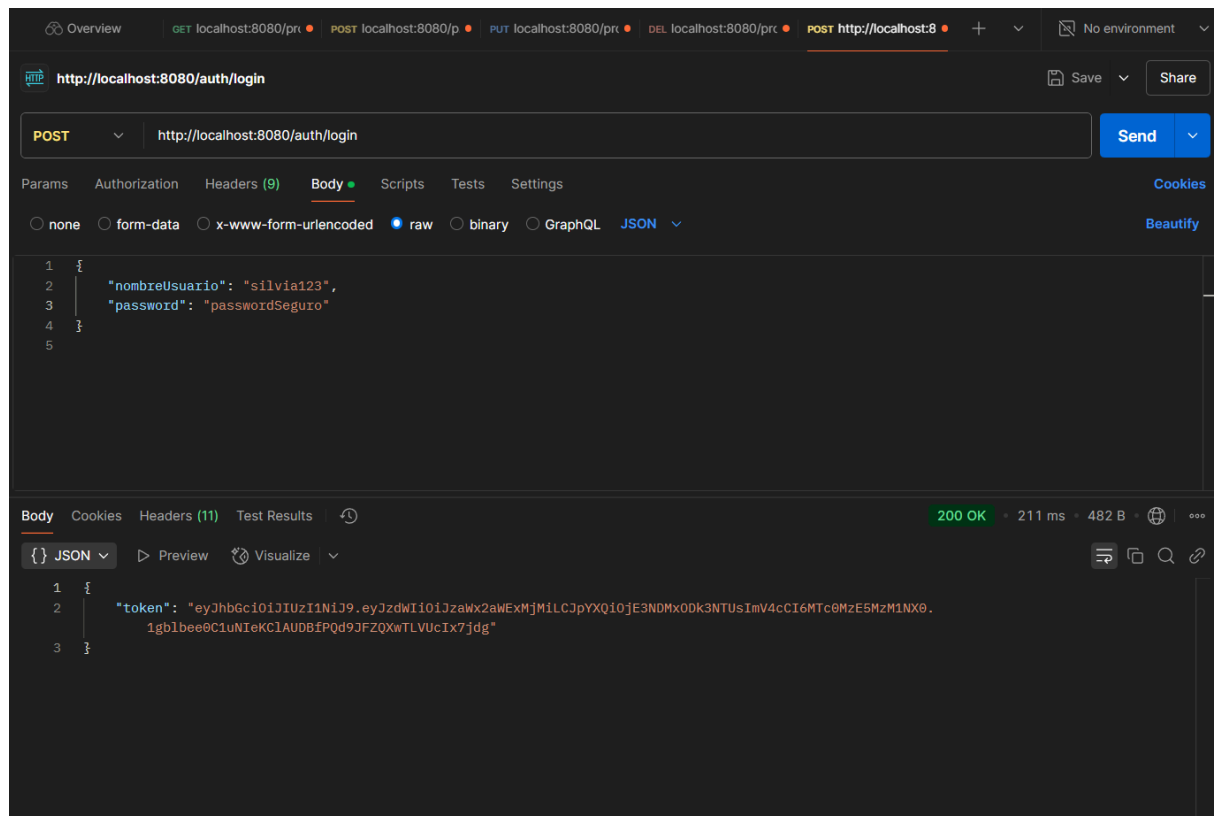


Imagen 10. Login usando Postman

También verificamos que, al intentar iniciar sesión con credenciales incorrectas, el sistema no permite el acceso, garantizando así la seguridad del proceso de autenticación.

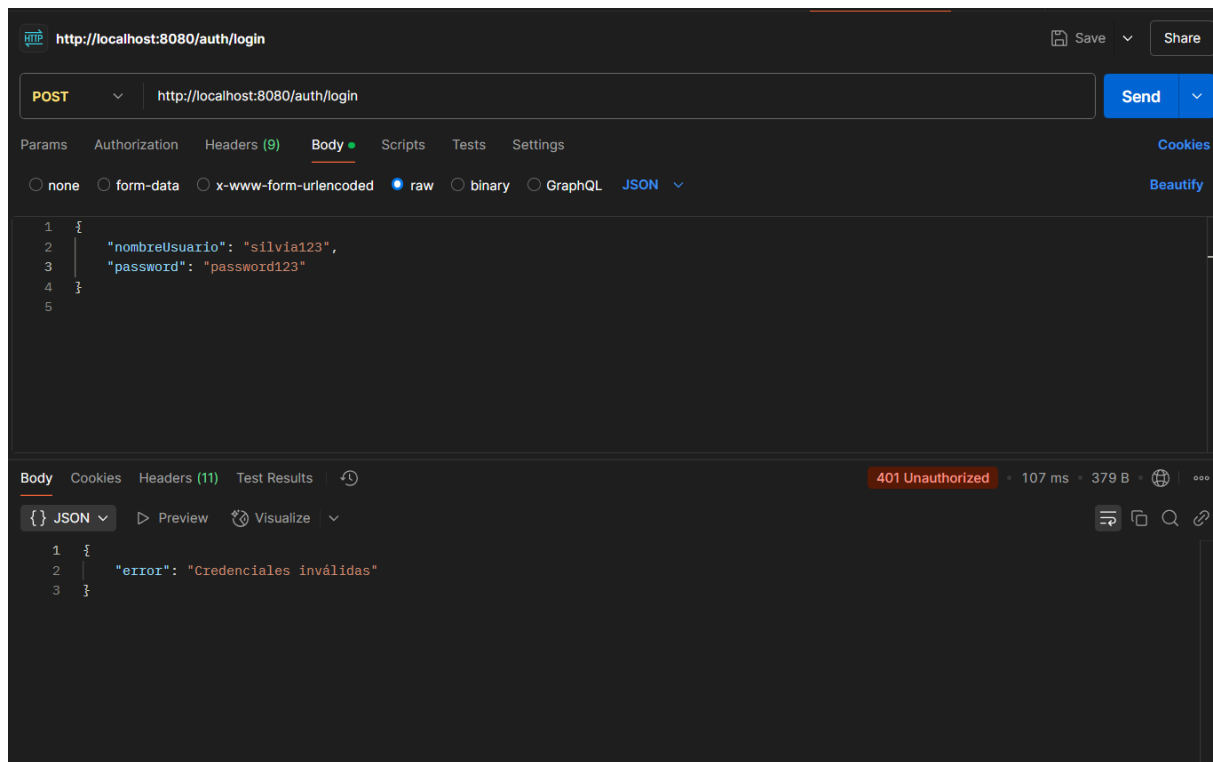


Imagen 11. Login invalido usando Postman

Para facilitar la documentación y prueba de la API, se configuró Swagger con acceso público. A través de esta herramienta, se pueden visualizar los endpoints disponibles y realizar pruebas de autenticación de manera sencilla.

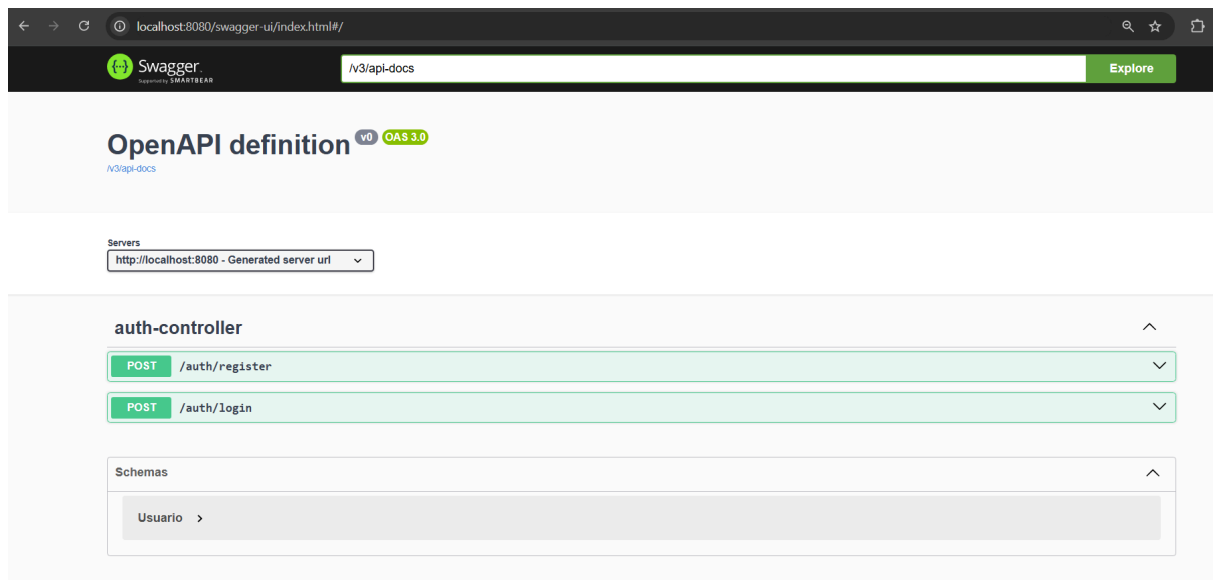


Imagen 12. API en Swagger

En Swagger, también es posible probar la creación de usuarios y el proceso de inicio de sesión, asegurando que todas las funcionalidades del backend operan correctamente antes de continuar con el desarrollo del frontend.

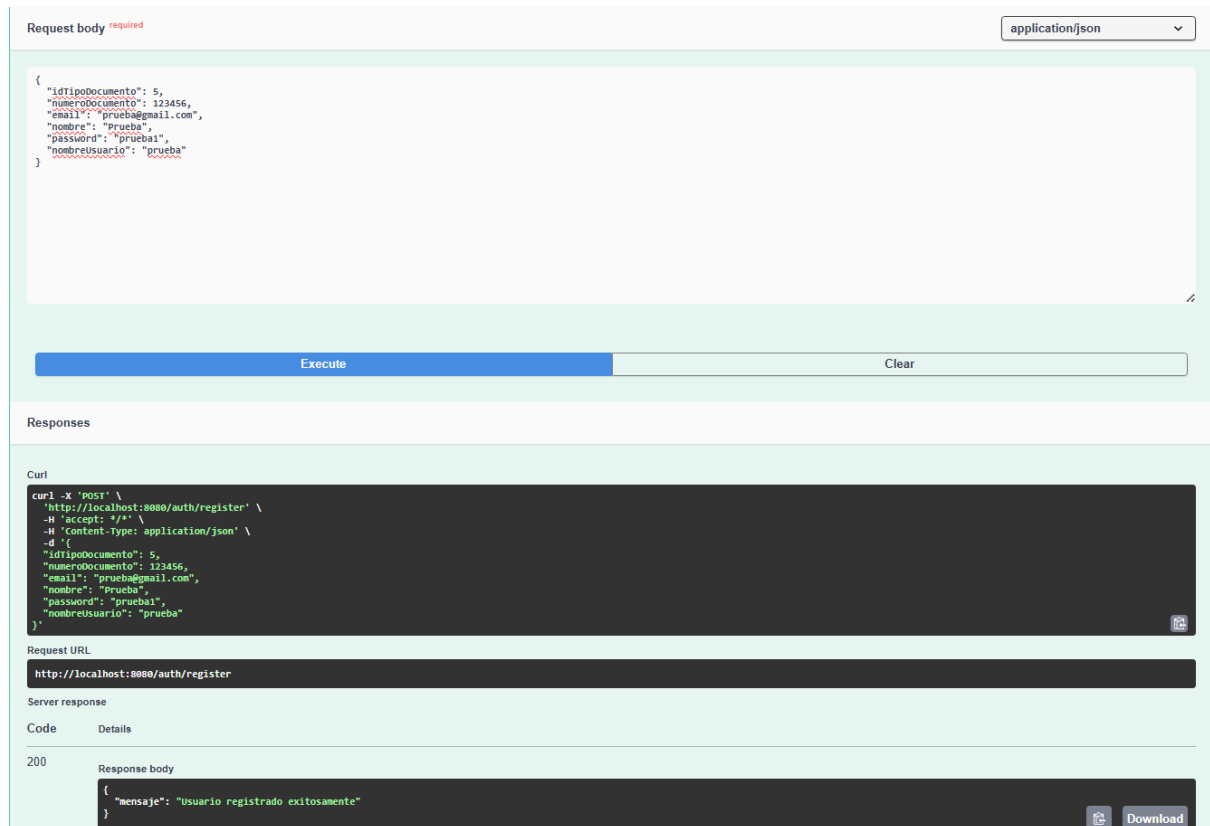


Imagen 13. Creación usuario en Swagger



Request body required application/json

```
{  "nombreUsuario": "prueba1",  "password": "prueba1"}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \  'http://localhost:8080/auth/login' \  -H 'accept: */*' \  -H 'Content-Type: application/json' \  -d '{  "nombreUsuario": "prueba1",  "password": "prueba1"  }'
```

Request URL

`http://localhost:8080/auth/login`

Server response

Code	Details
401 <small>Undocumented</small>	Error: response status is 401 Response body <pre>{  "error": "credenciales inválidas"}</pre> <span>Download</span>

Imagen 14. Login incorrecto en Swagger

Request body required application/json

```
{  "nombreUsuario": "prueba",  "password": "prueba"}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \  'http://localhost:8080/auth/login' \  -H 'accept: */*' \  -H 'Content-Type: application/json' \  -d '{  "nombreUsuario": "prueba",  "password": "prueba"  }'
```

Request URL

`http://localhost:8080/auth/login`

Server response

Code	Details
401 <small>Undocumented</small>	Error: response status is 401 Response body <pre>{  "error": "credenciales inválidas"}</pre> <span>Download</span>

Imagen 15. Login incorrecto en Swagger

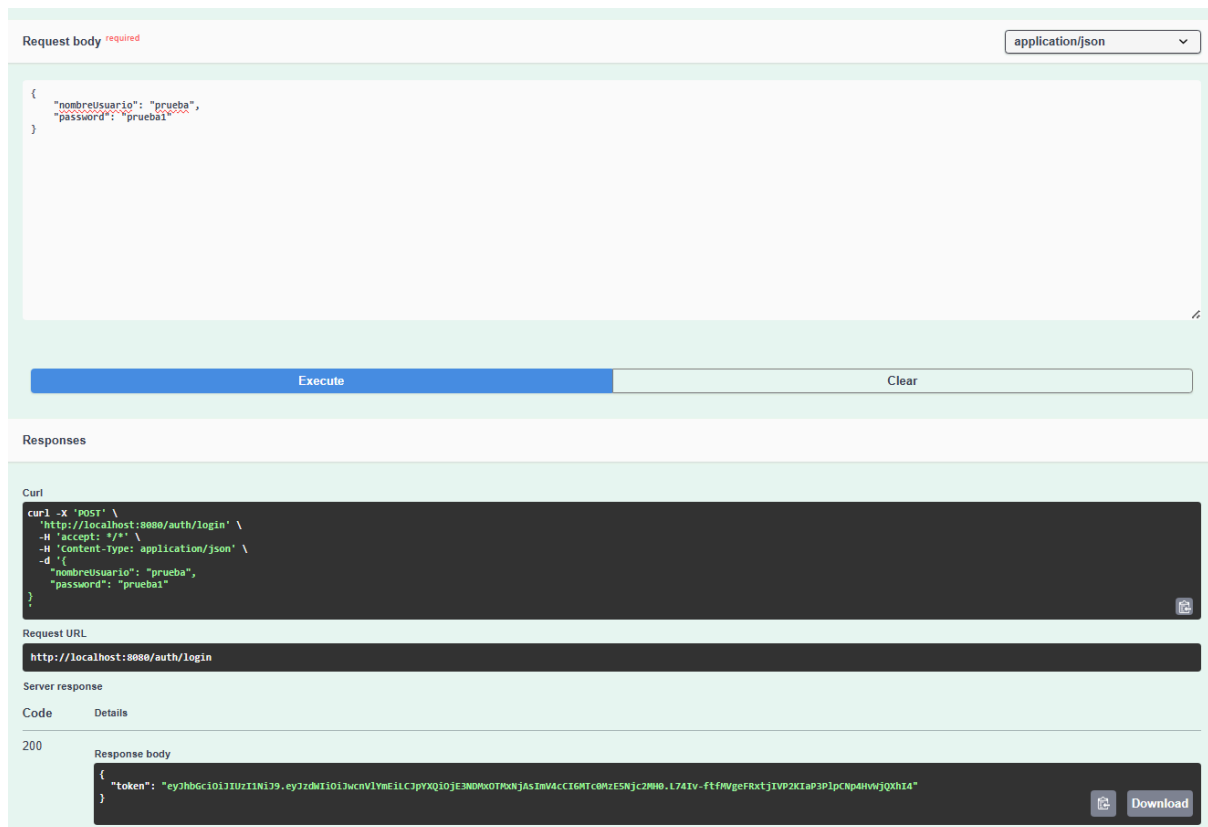


Imagen 16. Login correcto en Swagger

## Frontend

La estructura es muy básica ya que se implementa un frontend en vanilla javascript y estilizado mediante bootstrap. Consta de un archivo index.html en donde se alojan los dos paneles principales (login y registro), un archivo panel.html que emula la página que se renderiza al loguearse con éxito, y un archivo script.js donde se maneja toda la logica de programacion del frontend.

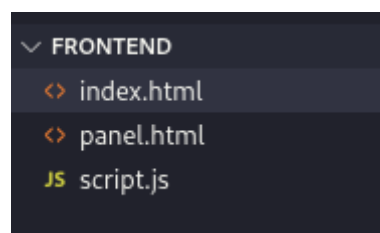


Imagen 17. Estructura del frontend

**Login panel - index.html:** Visual que se renderiza inicialmente en la página, consta de un form que solicita usuario y contraseña de ingreso, un botón para ingresar, un campo de texto para notificaciones y un link para renderizar el panel de registro (en caso de que se quiera registrar un usuario)

```
index.html X JS script.js panel.html
index.html > html > body > div.container.mt-5 > div#registerPanel > form#registerForm > div.mb-3 > select#registerDocumentType.form-control
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Login y Registro</title>
7   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
8 </head>
9 <body>
10   <div class="container mt-5">
11     <div id="loginPanel">
12       <h2>Iniciar Sesión</h2>
13       <form id="loginForm">
14         <div class="mb-3">
15           <label for="loginUser" class="form-label">Usuario</label>
16           <input type="text" class="form-control" id="loginUser" required>
17         </div>
18         <div class="mb-3">
19           <label for="loginPassword" class="form-label">Contraseña</label>
20           <input type="password" class="form-control" id="loginPassword" required>
21         </div>
22         <button type="submit" class="btn btn-primary">Ingresar</button>
23       </form>
24       <p id="loginMessage" class="text-muted mt-3"></p>
25       <button id="showRegister" class="btn btn-link mt-3">¿No tienes cuenta? Regístrate</button>
26     </div>
27   </div>
```

Imagen 18. Login en html

### Register panel - index.html:

Consta de un form que solicita todos los campos requeridos (Todos los campos son required por defecto) para crear un usuario tales como:

- Nombre - campo tipo texto
- Tipo de documento - campo tipo select
- Número de documento - campo tipo texto
- Correo electrónico - campo tipo email
- Nombre de usuario - campo tipo texto
- Contraseña - campo tipo password

Adicionalmente se cuenta con un campo de texto para notificaciones, un botón para registrar usuario y un link para volver al panel de iniciar sesión.

```

<div id="registerPanel" style="display: none;">
  <h2>Registro</h2>
  <form id="registerForm">
    <div class="mb-3">
      <label for="registerName" class="form-label">Nombre</label>
      <input type="text" class="form-control" id="registerName" required>
    </div>
    <div class="mb-3">
      <label for="registerDocumentType" class="form-label">Tipo de Documento</label>
      <select class="form-control" id="registerDocumentType" required>
        <option value="5">Cédula</option>
        <option value="6">Tarjeta</option>
        <option value="7">Pasaporte</option>
      </select>
    </div>
    <div class="mb-3">
      <label for="registerDocumentNumber" class="form-label">Número de Documento</label>
      <input type="text" class="form-control" id="registerDocumentNumber" required>
    </div>
    <div class="mb-3">
      <label for="registerEmail" class="form-label">Correo Electrónico</label>
      <input type="email" class="form-control" id="registerEmail" required>
    </div>
    <div class="mb-3">
      <label for="registerUsername" class="form-label">Nombre de Usuario</label>
      <input type="text" class="form-control" id="registerUsername" required>
    </div>
    <div class="mb-3">
      <label for="registerPassword" class="form-label">Contraseña</label>
      <input type="password" class="form-control" id="registerPassword" required>
    </div>
    <p id="registerMessage" class="text-muted mt-3"></p>
    <button type="submit" class="btn btn-success">Registrarse</button>
  </form>
  <button id="showLogin" class="btn btn-link mt-3">¿Ya tienes cuenta? Inicia sesión</button>
</div>

```

Imagen 19. Registro en html

Por último en el index.html se cuenta con una sección para linkear los archivos .js necesarios para el frontend, como lo son el index.js (lógica de programación) y el archivo de importación de bootstrap.

```

<script src="script.js"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>

```

Imagen 20. Importe de Bootstrap

**panel.html:** Es un archivo .html plano muy básico que renderiza un encabezado con el texto “Ingreso válido” con el fin de emular una página que se renderiza después de la autenticación satisfactoria del usuario. A su vez, cuenta con un hipervínculo que le permite al usuario cerrar sesión y así volver a la página principal de login (en el index.html)

```

index.html  JS script.js  panel.html X
panel.html > html
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Panel</title>
7    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
8  </head>
9  <body class="d-flex justify-content-center align-items-center vh-100 bg-light">
10   <div class="text-center">
11     <h1 class="text-success">Ingreso Válido</h1>
12     <a href="index.html" class="btn btn-primary mt-3">Cerrar sesión</a>
13   </div>
14 </body>
15 </html>

```

Imagen 21. Panel en html

**script.js:** Inicialmente se cuenta con los dos listener que permiten manejar el control de cuando se muestra cada uno de los paneles de index.html (login y register), si se hace uso de showRegister, se renderiza el panel register y se oculta el panel login, y viceversa.



```
JS script.js > addEventListener('submit') callback
1 // Alternar entre login y registro
2 document.getElementById('showRegister').addEventListener('click', function() {
3   document.getElementById('loginPanel').style.display = 'none';
4   document.getElementById('registerPanel').style.display = 'block';
5 });
6
7 document.getElementById('showLogin').addEventListener('click', function() {
8   document.getElementById('registerPanel').style.display = 'none';
9   document.getElementById('loginPanel').style.display = 'block';
10 });
11
```

Imagen 22. JavaScript de mostrar login o registro

En cuanto al login se crea la función listener del botón iniciar sesión con el fin de capturar los datos ingresados en los campos de usuario y contraseña, que posteriormente serán enviados en el body de una petición POST (haciendo uso de la herramienta fetch) al endpoint /auth/login del backend, con el fin de autenticar el usuario, si la respuesta del servidor es correcta se renderiza el panel.html (lo que indicaría que el usuario se pudo autenticar exitosamente). Dado el caso de que la autenticación sea errónea, se mostrará un mensaje de error en el apartado de notificaciones, y no avanzara el usuario a panel.html sino se mantendrá en el panel de login actual.



```
// Login
document.getElementById('loginForm').addEventListener('submit', async function(event) {
  event.preventDefault();
  const user = document.getElementById('loginUser').value;
  const password = document.getElementById('loginPassword').value;
  let modalMessage = '';
  const response = await fetch('http://localhost:8080/auth/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ nombreUsuario: user, password })
  });

  if (response.ok) {
    document.getElementById("loginMessage").innerHTML = '';
    window.location.href = 'panel.html';
  } else {
    document.getElementById("loginMessage").innerHTML = 'Usuario o contraseña incorrectos. Intenta nuevamente.'
  }
});
```

Imagen 23. Función de login

En cuanto a la lógica del registro, se crea el listener para el botón registrarse. Lo primero que hace el listener es capturar todos los valores de los campos del form de registro, que serán enviados en el body de una petición POST (haciendo uso de la herramienta fetch) al endpoint /auth/register del backend, con el fin de registrar el

usuario, si la respuesta del backend es exitosa se renderiza en el campo de notificaciones la confirmación del registro exitoso, y se mantiene el panel de registro por si se desea realizar un nuevo registro, el usuario tiene la posibilidad de volver al panel de login si así lo desea haciendo uso del link habilitado para tal caso. Dada la situación de que la respuesta del backend sea errónea al momento de registrar el usuario (o exista un error al momento de conectarse con el backend en sí) igualmente se notificará en el apartado de notificaciones y se mantendrá renderizado el panel de registro.

```
// Registro
document.getElementById('registerForm').addEventListener('submit', async function(event) {
    event.preventDefault();
    const name = document.getElementById('registerName').value;
    const documentType = document.getElementById('registerDocumentType').value;
    const documentNumber = document.getElementById('registerDocumentNumber').value;
    const email = document.getElementById('registerEmail').value;
    const username = document.getElementById('registerUsername').value;
    const password = document.getElementById('registerPassword').value;

    try {
        const response = await fetch('http://localhost:8080/auth/register', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({
                nombre: name,
                idTipoDocumento: parseInt(documentType),
                numeroDocumento: documentNumber,
                email: email,
                nombreUsuario: username,
                password: password
            })
        });

        if (response.ok) {
            document.getElementById('registerMessage').innerHTML = 'Registro Exitoso!';
        } else {
            document.getElementById('registerMessage').innerHTML = 'Ha habido un error y el usuario no pudo registrarse. Por favor, inténtalo de nuevo.';
        }
    } catch (error) {
        console.error(error);
        document.getElementById('registerMessage').innerHTML = 'Error de conexión. Inténtelo más tarde.'; // Error de conexión
    }
});
```

Imagen 24. Función de registro

El frontend se vería pues de la siguiente forma:

The screenshot shows a web browser window with the address bar displaying 'file:///home/theyrent/workspace/universidad/2025/entornos/segundo\_taller/frontend/index.html'. The page content includes a heading 'Iniciar Sesión', a label 'Usuario' above a text input field, a label 'Contraseña' above another text input field, a blue button labeled 'Ingresar', and a link at the bottom that reads '¿No tienes cuenta? [Registrate](#)'.

Imagen 25. Panel login

Iniciar Sesión

Usuario

dilanexample

Contraseña

••••

Ingresar

Usuario o contraseña incorrectos. Intenta nuevamente.

[¿No tienes cuenta? Regístrate](#)

Imagen 26. Panel login (error al iniciar sesión)

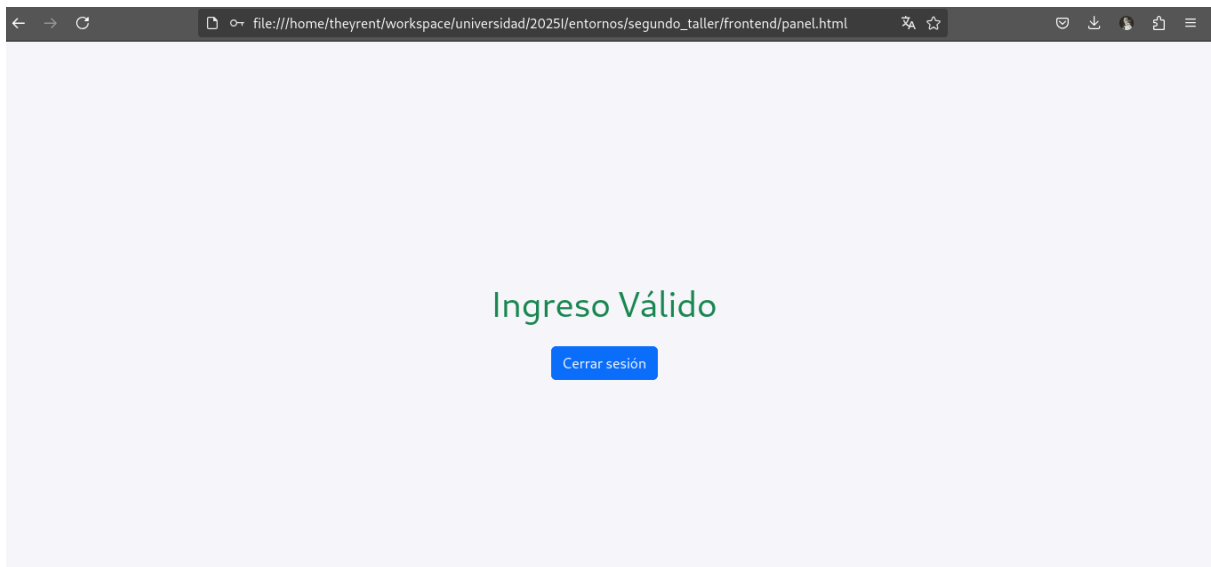


Imagen 27. Panel.html (ingreso exitoso)

Registro

Nombre

Tipo de Documento

Cédula

Número de Documento

Correo Electrónico

Nombre de Usuario

Contraseña

Registrarse

[¿Ya tienes cuenta? Inicia sesión](#)

Imagen 27. Panel de registro

Registro

Nombre

Marco Tulio

Tipo de Documento

Tarjeta

Número de Documento

65901238

Correo Electrónico

marcot@uis.edu.co

Nombre de Usuario

soyMarcoTulio

Contraseña

•••••

Registro Exitoso!

Registrarse

Imagen 28. Panel de registro (registro exitoso)