



# Manual Técnico

11/7/2025

DWF

## Herramientas y tecnologías que se utilizaron

**Spring Boot:** es un framework de código abierto que da soporte para el desarrollo de aplicaciones y páginas webs basadas en Java. Se trata de uno de los entornos más populares y ayuda a los desarrolladores back-end a crear aplicaciones con un alto rendimiento empleando objetos de java sencillos.

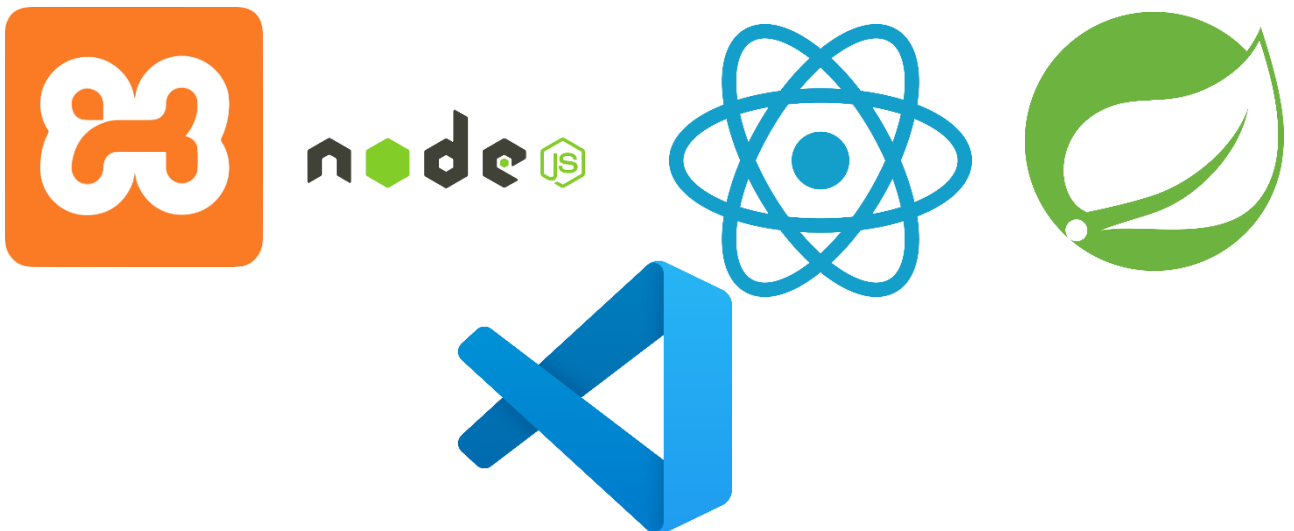
**React** native biblioteca de JavaScript de código abierto para crear interfaces de usuario interactivas.

**Node.js** es un entorno de ejecución de código JavaScript del lado del servidor, lo que permite a los desarrolladores usar JavaScript para crear aplicaciones web y herramientas de línea de comandos que no necesitan un navegador

**XAMPP** es un paquete de software gratuito y de código abierto que crea un servidor web local en tu computadora. El nombre es un acrónimo de Apache, MariaDB (antes MySQL), PHP y Perl, lo que permite a los desarrolladores crear y probar sitios y aplicaciones web en su máquina local antes de publicarlos en línea.

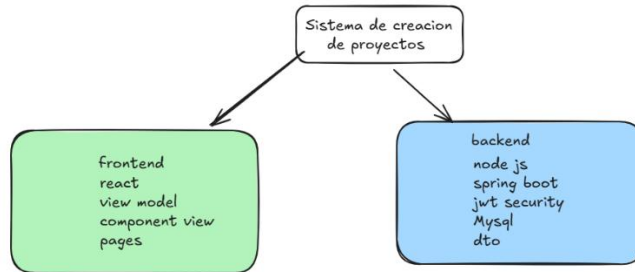
**Visual studio code:** es un editor de código fuente gratuito, ligero y potente para Windows, macOS y Linux. Es muy personalizable y extensible gracias a un amplio ecosistema de extensiones, lo que le permite soportar una gran variedad de lenguajes de programación y entornos de desarrollo. Está diseñado para ser eficiente e incluye funciones como resaltado de sintaxis, depuración, control de versiones con integración

de Git y autocompletado de código

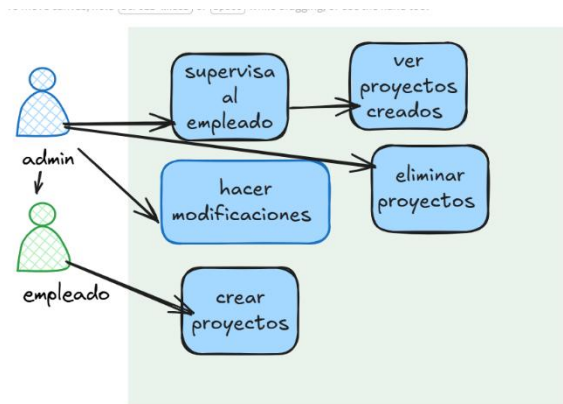


# Diagramas

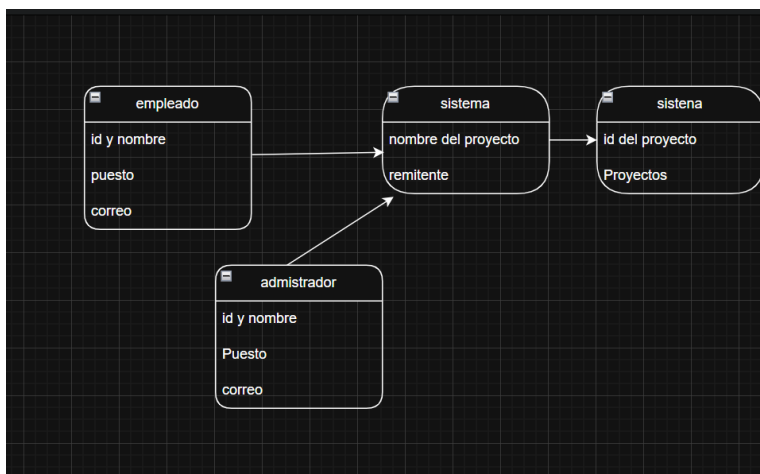
## Arquitectura



## Diagrama de caso de uso

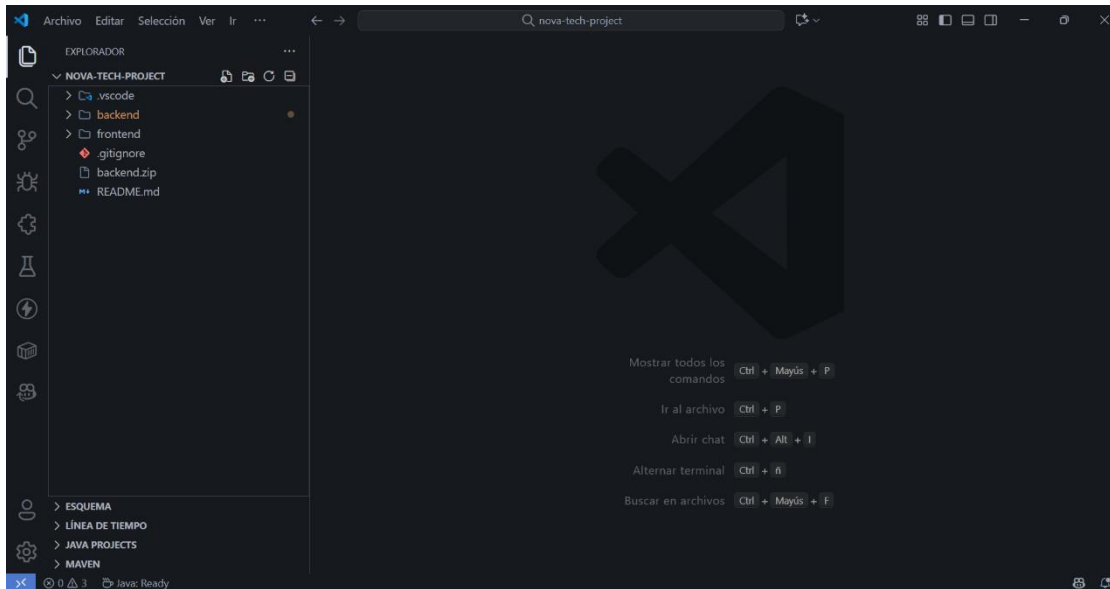


## Diagrama de la base datos

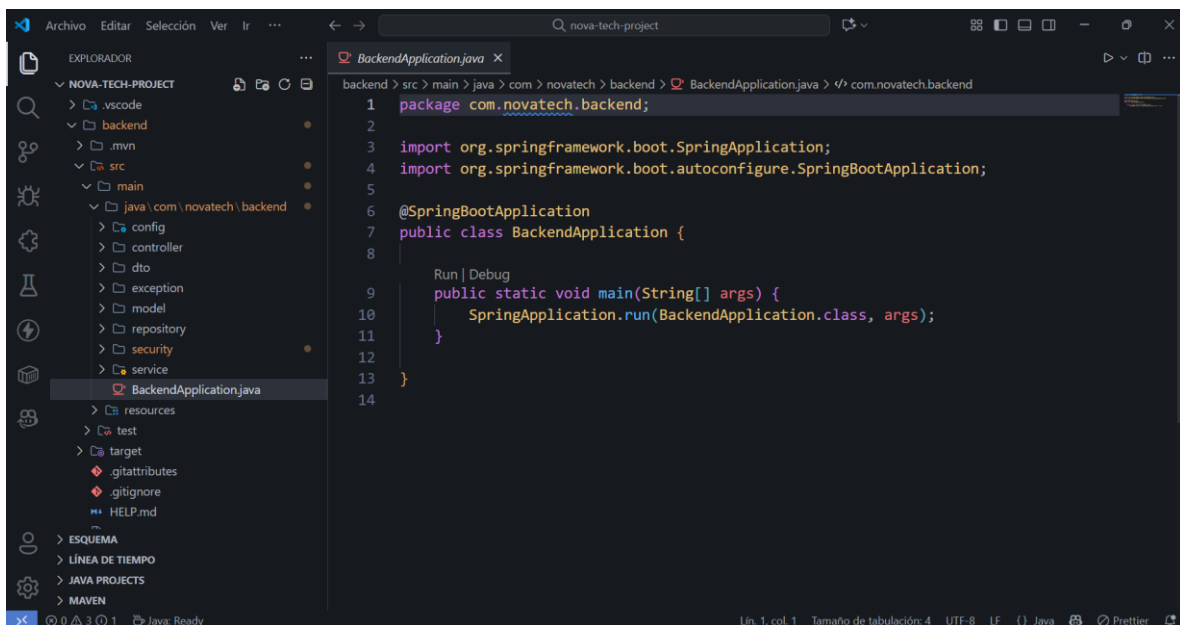


## Estructura del proyecto

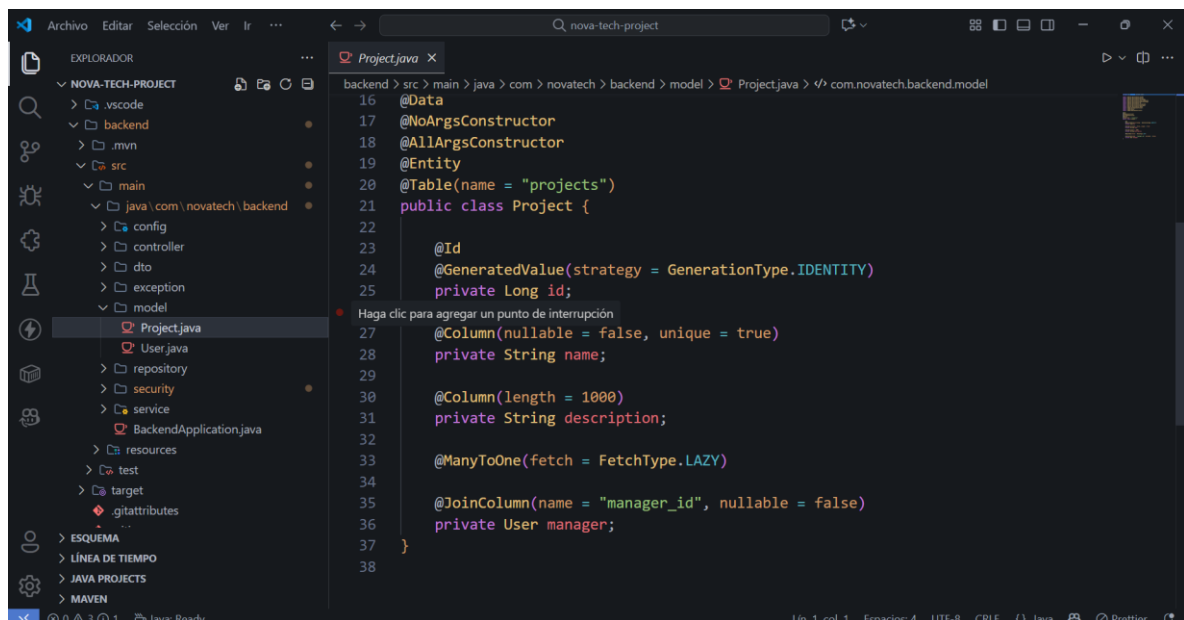
Esta separado por carpetas el front y backend



En la carpeta del backend esta toda la lógica del proyecto



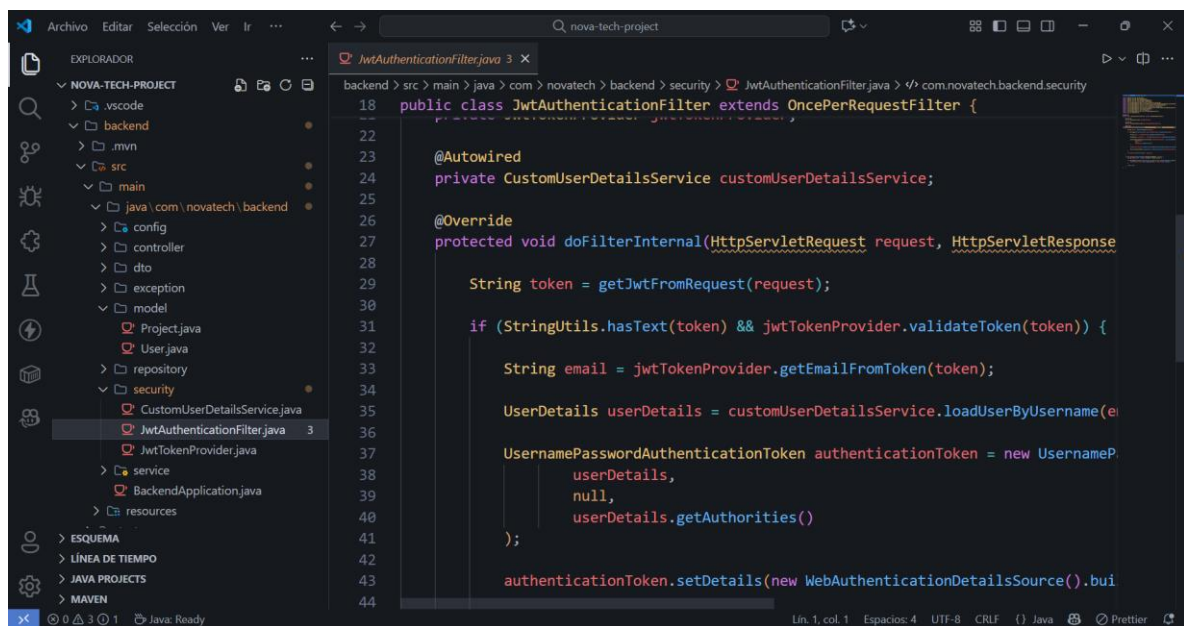
Observando algunas tenemos los modelos



The screenshot shows the Visual Studio Code editor with the 'nova-tech-project' workspace. The Explorer sidebar on the left shows the project structure, with the 'model' package selected under 'src/main/java/com/novatech/backend'. The main editor displays the 'Project.java' file, which is a JPA entity. The code includes annotations for @Data, @NoArgsConstructor, @AllArgsConstructor, @Entity, and @Table(name = "projects"). It defines a public class Project with private fields for id (annotated with @Id and @GeneratedValue), name (annotated with @Column), and description (annotated with @Column). There is also a many-to-one relationship with the User entity, annotated with @ManyToOne and @JoinColumn.

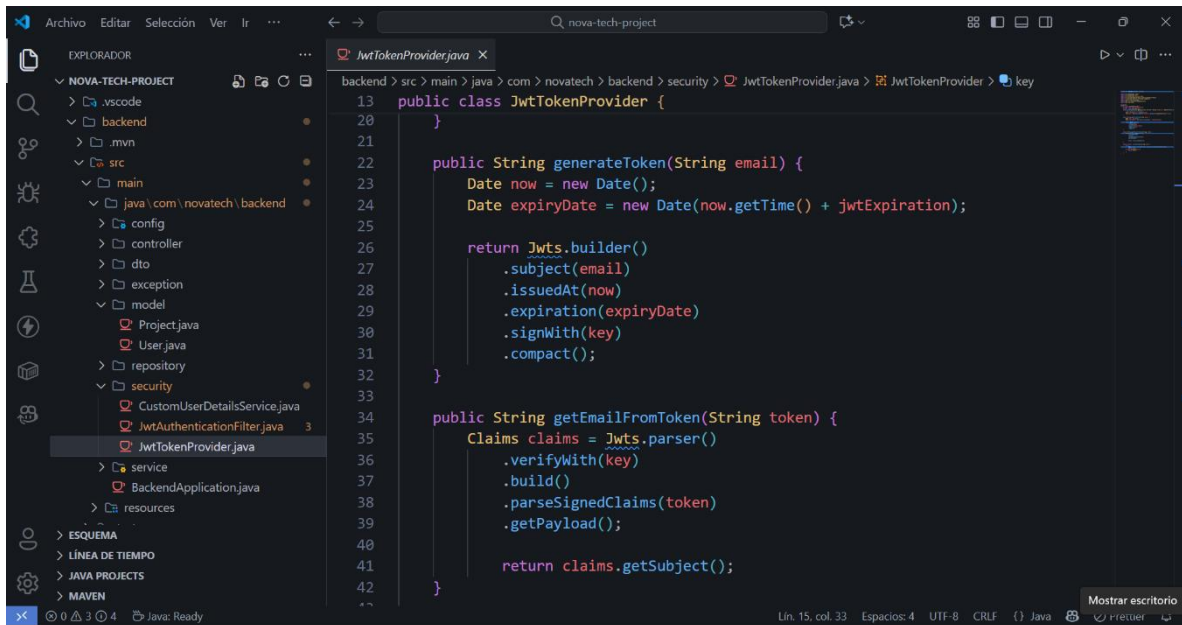
```
16 @Data
17 @NoArgsConstructor
18 @AllArgsConstructor
19 @Entity
20 @Table(name = "projects")
21 public class Project {
22
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY)
25     private Long id;
26
27     @Column(nullable = false, unique = true)
28     private String name;
29
30     @Column(length = 1000)
31     private String description;
32
33     @ManyToOne(fetch = FetchType.LAZY)
34
35     @JoinColumn(name = "manager_id", nullable = false)
36     private User manager;
37 }
38
```

Tenemos también la carpeta de security es la seguridad de nuestro proyecto  
utilizamos jwt



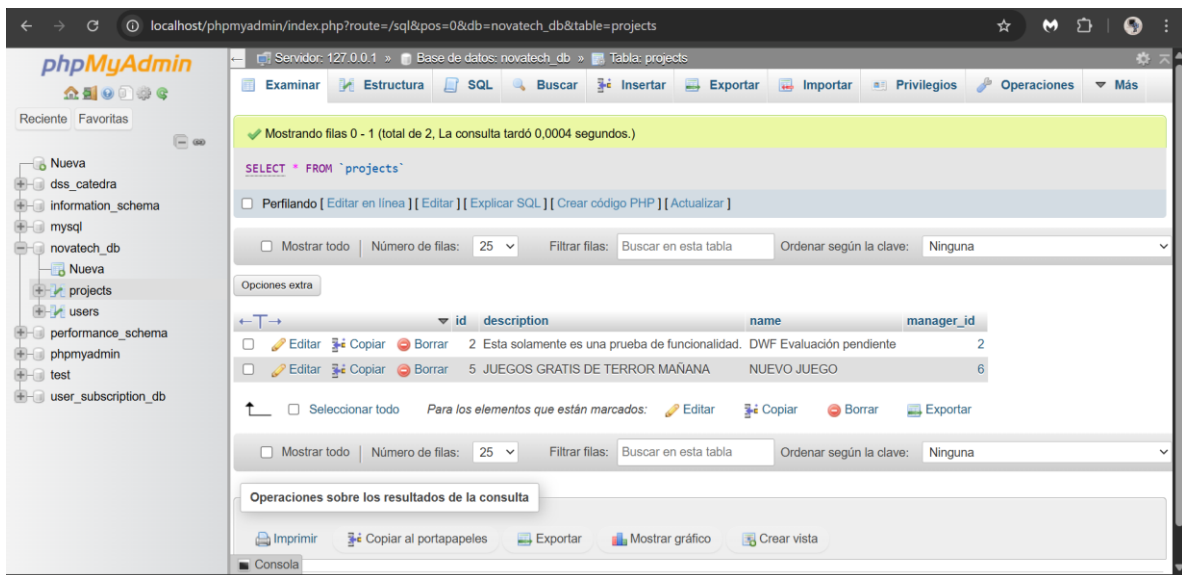
The screenshot shows the Visual Studio Code editor with the 'nova-tech-project' workspace. The Explorer sidebar on the left shows the project structure, with the 'security' package selected under 'src/main/java/com/novatech/backend'. The main editor displays the 'JwtAuthenticationFilter.java' file, which is a Spring Security filter. The code includes annotations for @Autowired and @Override. It implements the doFilterInternal method, which extracts the JWT token from the request, validates it using the JwtTokenProvider, and then loads the user details from the CustomUserDetailsService. Finally, it creates a UsernamePasswordAuthenticationToken and sets its details using the WebAuthenticationDetailsSource.

```
18 public class JwtAuthenticationFilter extends OncePerRequestFilter {
19
20     @Autowired
21     private CustomUserDetailsService customUserDetailsService;
22
23     @Override
24     protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws ServletException, IOException {
25
26         String token = getJwtFromRequest(request);
27
28         if (StringUtils.hasText(token) && jwtTokenProvider.validateToken(token)) {
29
30             String email = jwtTokenProvider.getEmailFromToken(token);
31
32             UserDetails userDetails = customUserDetailsService.loadUserByUsername(email);
33
34             UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(
35                 userDetails,
36                 null,
37                 userDetails.getAuthorities()
38             );
39
40             authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
41
42             chain.doFilter(request, response);
43         }
44     }
45 }
```



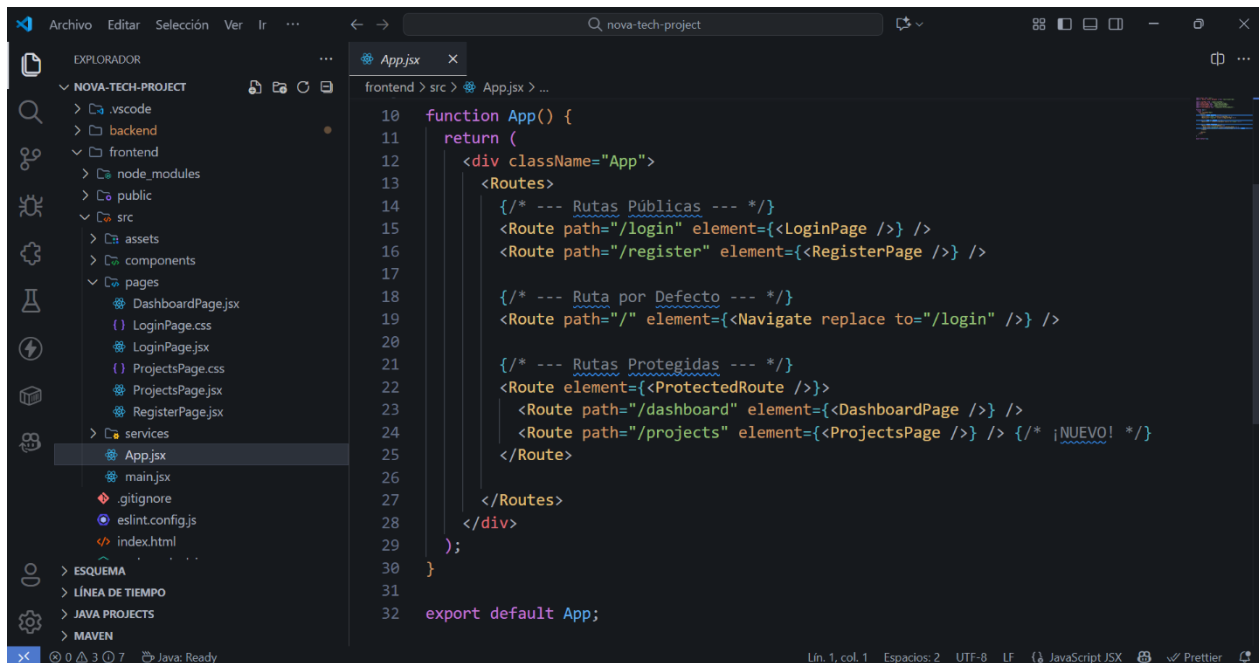
```
13 public class JwtTokenProvider {
20 }
21
22 public String generateToken(String email) {
23     Date now = new Date();
24     Date expiryDate = new Date(now.getTime() + jwtExpiration);
25
26     return Jwts.builder()
27         .subject(email)
28         .issuedAt(now)
29         .expiration(expiryDate)
30         .signWith(key)
31         .compact();
32 }
33
34 public String getEmailFromToken(String token) {
35     Claims claims = Jwts.parser()
36         .verifyWith(key)
37         .build()
38         .parseSignedClaims(token)
39         .getPayload();
40
41     return claims.getSubject();
42 }
```

Tenemos nuestra base de datos que almacena los usuarios y proyectos



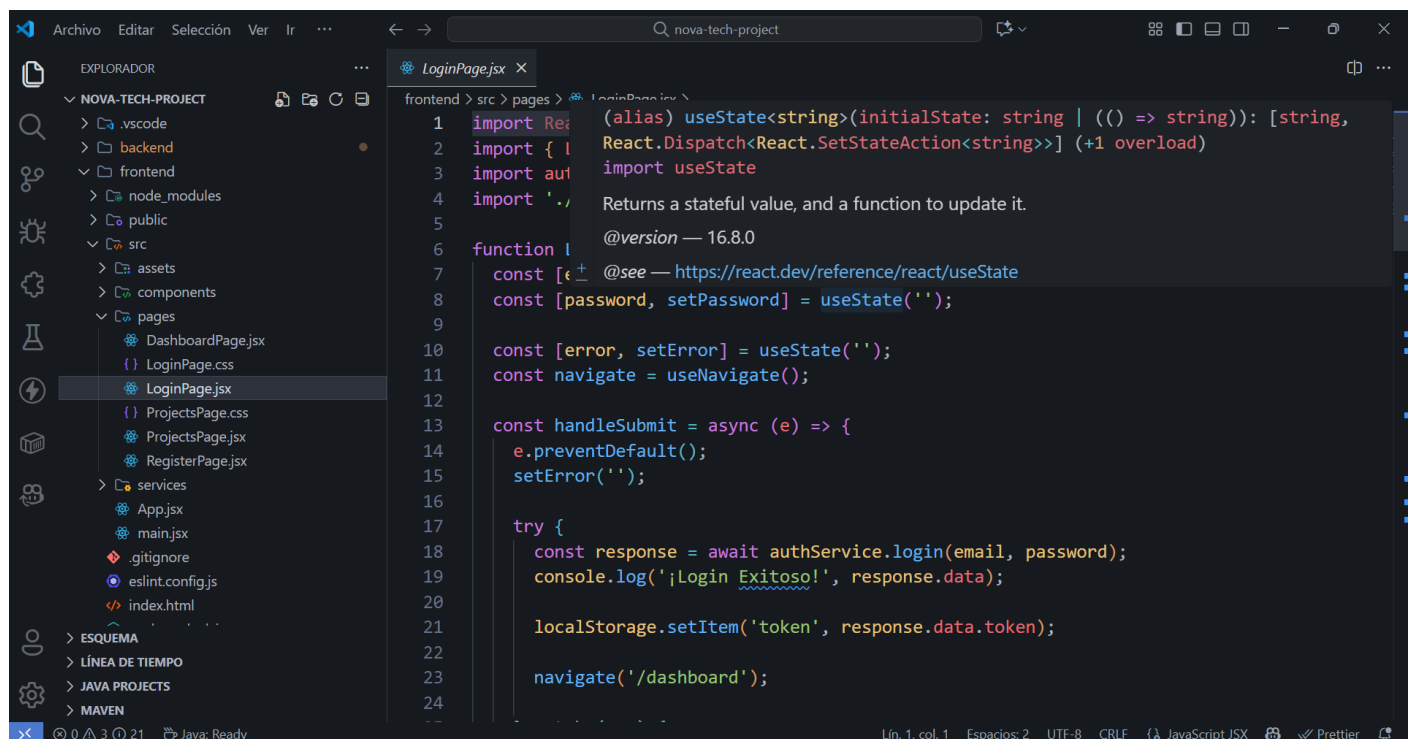
id	description	name	manager_id
2	Esta solamente es una prueba de funcionalidad. DWF Evaluación pendiente		2
5	JUEGOS GRATIS DE TERROR MAÑANA	NUEVO JUEGO	6

Y por parte del front tenemos lo siguiente aquí almacenamos todas nuestras vistas



```
10 function App() {
11   return (
12     <div className="App">
13       <Routes>
14         /* --- Rutas Públicas --- */
15         <Route path="/login" element={<LoginPage />} />
16         <Route path="/register" element={<RegisterPage />} />
17
18         /* --- Ruta por Defecto --- */
19         <Route path="/" element={<Navigate replace to="/login" />} />
20
21         /* --- Rutas Protegidas --- */
22         <Route element={<ProtectedRoute />}>
23           <Route path="/dashboard" element={<DashboardPage />} />
24           <Route path="/projects" element={<ProjectsPage />} /> /* ¡NUEVO! */
25         </Route>
26       </Routes>
27     </div>
28   );
29 }
30
31
32 export default App;
```

Como la del login



```
1 import React, { useState } from 'react';
2 import { useNavigate } from 'react-router-dom';
3 import authService from '../services/authService';
4
5 Returns a stateful value, and a function to update it.
6 @version — 16.8.0
7
8 function Login() {
9   const [password, setPassword] = useState('');
10   const [error, setError] = useState('');
11   const navigate = useNavigate();
12
13   const handleSubmit = async (e) => {
14     e.preventDefault();
15     setError('');
16
17     try {
18       const response = await authService.login(email, password);
19       console.log('¡Login Exitoso!', response.data);
20
21       localStorage.setItem('token', response.data.token);
22
23       navigate('/dashboard');
24     } catch (error) {
25       setError(error.message);
26     }
27   }
28
29   return (
30     <div>
31       <input type="password" value={password} onChange={setPassword} />
32       <button type="button" value="Login" onClick={handleSubmit} />
33     </div>
34   );
35 }
```