# Manual Tecnico de ProjeX

**1. Introducción**

* **Nombre del proyecto**: ProjeX
* **Descripción**: Aplicación web para la administración integral de proyectos: creación, asignación de tareas, seguimiento y reportes.
* **Objetivo del manual técnico**: Documentar la arquitectura, tecnologías, estructura del código, configuración del entorno y pautas de mantenimiento.

**2. Arquitectura del Sistema**

* **Tipo**: Web cliente-servidor.
* **Componentes**:
  + Frontend: interfaz de usuario (por ejemplo React/Next.js).
  + Backend/API: lógica y endpoints (Node.js/Express o similar).
  + Base de datos: PostgreSQL/Supabase.
* Incluye un **diagrama de bloques** que muestre interacción entre UI, servidor, BD y autenticación (si aplica).

**3. Tecnologías Utilizadas**

* **Lenguajes**: JavaScript/TypeScript.
* **Frontend**: Next.js (u otro framework), librerías UI como Tailwind, Bootstrap, Material‑UI.
* **Backend**: Node.js + Express.
* **Base de Datos**: Supabase.
* **Autenticación**: Integración con Supabase Auth.
* **Herramientas adicionales**: ESLint, Prettier, Jest/Testing Library.

**4. Instalación del Entorno de Desarrollo**

1. Clonar el repositorio:

bash

git clone https://github.com/CarlosVilllalvaQuispe/G505-Gestion-de-proyectos.git

1. Instalar dependencias:

bash

cd G505-Gestion-de-proyectos

npm install # o yarn

1. Crear y configurar variables de entorno (.env):

* RESEND\_API\_KEY=link
* NEXT\_PUBLIC\_SUPABASE\_URL=link
* NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=link
* NEXT\_PUBLIC\_SUPABASE\_STORAGE\_URL=link
* NEXT\_PUBLIC\_API\_URL=http://localhost:3000

1. Iniciar los servicios:

bash

npm run dev

(o npm start, npm run backend, según tu configuración)

**5. Estructura del Código**

**📁 Raíz del proyecto**

* /frontend (o en raíz si es monorepo)
* /backend (si aplica)
* /scripts, /tests

**Frontend**

* pages/ – rutas de la app
* components/ – elementos UI reutilizables
* services/ – funciones de llamadas a la API
* styles/ – CSS/Tailwind

**Backend**

* routes/ – endpoints (GET, POST…)
* controllers/ – lógica por recurso
* models/ – definiciones de tablas con ORM (Prisma/Sequelize) o consultas SQL
* middlewares/ – autenticación, validaciones

**6. Base de Datos**

* **Diagrama ER**: entidades como Proyecto, Tarea, Usuario, Asignación.
* **Descripciones**:
  + Proyecto: id, nombre, descripcion, fecha\_inicio, fecha\_fin.
  + Tarea: id, proyecto\_id, titulo, descripcion, estado, fecha\_entrega.
  + Usuario: id, nombre, correo, rol.
* **Relaciones**: Usuario ↔ Tarea (asignación), Proyecto ↔ Tarea.
* **Scripts** SQL o migraciones incluidas en /migrations.

**7. API y Endpoints**

| **Ruta** | **Método** | **Función** | **Entrada** | **Salida** |
| --- | --- | --- | --- | --- |
| /api/proyectos | GET | Listar proyectos | — | Array de proyectos |
| /api/proyectos | POST | Crear proyecto | {nombre, descripcion…} | Proyecto creado |
| /api/proyectos/:id | PUT | Editar | {…} | Proyecto actualizado |
| /api/tareas | GET | Tareas de un proyecto | proyecto\_id | Array de tareas |
| /api/tareas | POST | Crear tarea | {...} | Tarea creada |
| ... | ... | ... | ... | ... |

Incluye ejemplos de request/response JSON y códigos de error.

**8. Seguridad y Autenticación**

* Uso de JWT o tokens de Supabase Auth
* Middleware de validación de token en rutas protegidas
* Manejo de errores de acceso no autorizado (401/403)
* Encriptación de contraseñas (por ejemplo con bcrypt)

**9. Pruebas**

**🔹 Pruebas End-to-End (E2E) con Playwright**

Se utilizaron pruebas E2E con [**Playwright**](https://playwright.dev/) para garantizar que el flujo completo de la aplicación funcione correctamente desde la perspectiva del usuario.

**✅ Funcionalidades cubiertas:**

* Autenticación (login/logout)
* Creación de proyectos y tareas
* Asignación de tareas a usuarios
* Edición y eliminación de elementos
* Validación de errores del sistema

**🛠️ Instalación de dependencias:**

bash

npm install --save-dev @playwright/test

npx playwright install

**▶️ Comando para ejecutar pruebas:**

bash

npx playwright test

**📁 Estructura de pruebas:**

bash

/tests

└── e2e

├── login.spec.ts

├── proyectos.spec.ts

├── tareas.spec.ts

**📄 Ejemplo básico (login.spec.ts):**

ts

import { test, expect } from '@playwright/test';

test('login exitoso con credenciales válidas', async ({ page }) => {

await page.goto('http://localhost:3000/login');

await page.fill('#email', 'usuario@demo.com');

await page.fill('#password', 'contrasena123');

await page.click('button[type="submit"]');

await expect(page).toHaveURL('http://localhost:3000/dashboard');

});

**🧪 Reportes:**

* Playwright genera un informe automático en HTML con capturas de pantalla y estado de cada prueba:

bash

npx playwright show-report

**✅ Ventajas del uso de Playwright:**

* Soporte multi-navegador (Chromium, Firefox, WebKit)
* Capturas automáticas en fallos
* Fácil integración con CI/CD

**10. Despliegue y Producción**

* Comandos:

bash

npm run build

npm run start

* Configuración de variables en entorno de producción
* Consideraciones: migraciones, backups de BD, certificados HTTPS (Let's Encrypt)
* Logs y monitoreo (p. ej. con PM2, Logrotate)

**11. Mantenimiento y Buenas Prácticas**

* Normas de estilo: ESLint, Prettier
* Convenciones de Git: ramas features y PR, git-flow
* Cómo agregar una nueva entidad/tarea:
  1. Crear migración
  2. Definir modelo en backend
  3. Crear rutas/endpoints
  4. Añadir UI/form
  5. Escribir tests correspondientes