
MANUAL TÉCNICO

Plataforma de Gestión de Proyectos con Integración de IA

Fecha: 26 de junio de 2025
Versión: 1.0
Contacto: Braulio Cesar Ortega Batall
E-mail: 75142209@continental.edu.pe

Índice

1. **Introducción**
 - 1.1. Descripción General del Sistema
 - 1.2. Alcance del Manual
2. **Requisitos del Sistema**
 - 2.1. Requisitos de Hardware
 - 2.2. Requisitos de Software
 - 2.3. Dependencias
3. **Arquitectura del Sistema**
 - 3.1. Diagrama de Arquitectura
 - 3.2. Descripción de Componentes
4. **Tecnologías Utilizadas**
 - 4.1. Stack Tecnológico
 - 4.2. Justificación de Tecnologías
5. **Estructura del Código**
 - 5.1. Organización de Carpetas
 - 5.2. Módulos Clave
6. **Instalación y Configuración**
 - 6.1. Clonado del Repositorio
 - 6.2. Configuración de Variables de Entorno
 - 6.3. Levantamiento del Entorno
7. **Base de Datos**
 - 7.1. Modelo de Datos
 - 7.2. Conexión a la Base de Datos
8. **Seguridad**
 - 8.1. Autenticación y Autorización
 - 8.2. Protección de API
9. **Pruebas**
 - 9.1. Tipos de Pruebas Realizadas
 - 9.2. Ejecución de Pruebas

- 9.3. Herramientas Utilizadas
 - 10. Mantenimiento y Actualizaciones**
 - 10.1. Buenas Prácticas
 - 10.2. Agregar Nuevas Funcionalidades
 - 11. Solución de Problemas Comunes**
 - 12. Apéndices**
 - 12.1. APIs Externas Utilizadas
 - 12.2. Enlaces a Documentación
-

1. Introducción

1.1. Descripción General del Sistema

La "Plataforma de Gestión de Proyectos Colaborativos con Integración de Inteligencia Artificial" es una aplicación web desarrollada para los estudiantes del curso "Dirección de proyectos" de la Universidad Continental. Su objetivo es proporcionar un entorno práctico y moderno para aplicar los conocimientos teóricos del curso, utilizando una pila tecnológica MERN (MongoDB, Express.js, React, Node.js) y funcionalidades avanzadas de IA. ¹

1.2. Alcance del Manual

Este documento está dirigido a desarrolladores, administradores de sistemas y personal técnico encargado del mantenimiento, despliegue y soporte de la plataforma. Proporciona una guía detallada sobre la arquitectura, configuración, estructura del código, procedimientos de prueba y solución de problemas.

2. Requisitos del Sistema

2.1. Requisitos de Hardware

- **Servidor:**
 - CPU: 2+ núcleos
 - RAM: 4 GB+
 - Almacenamiento: 20 GB+ de espacio en disco
- **Cliente:**
 - Cualquier dispositivo con un navegador web moderno y conexión a internet.

2.2. Requisitos de Software

- **Servidor:**
 - Sistema Operativo: Linux (recomendado), Windows o macOS.
 - Docker y Docker Compose. ²
- **Cliente:**
 - Navegador web: Google Chrome, Mozilla Firefox, Microsoft Edge o Safari en sus versiones más recientes.

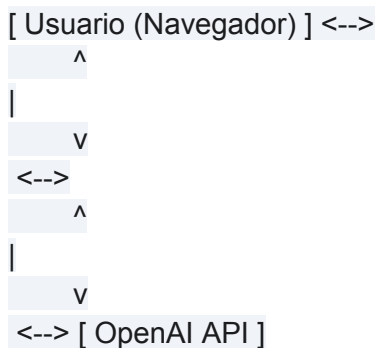
2.3. Dependencias

- **Base de Datos:** MongoDB (versión 5.0 o superior), preferiblemente alojada en un servicio como MongoDB Atlas.
- **Librerías Principales:** Node.js, React, Express.js, Mongoose, Socket.io, LangChain, Jest, Cypress, Postman.
- **Servicios Externos:**
 - OpenAI API (para funcionalidades de IA).
 - Firebase Cloud Messaging (para notificaciones push).

3. Arquitectura del Sistema

3.1. Diagrama de Arquitectura

El sistema sigue una arquitectura de microservicios desacoplada, compuesta por tres componentes principales que se comunican a través de APIs REST.



3.2. Descripción de Componentes

- **gestion-APP (Frontend):** Es la aplicación cliente construida con React, Vite y estilizada con Tailwind CSS. Se encarga de renderizar la interfaz de usuario, gestionar el estado local y comunicarse con el backend para obtener y enviar datos.
- **backend (API Principal):** Es el núcleo del sistema, desarrollado con Node.js y Express.js. Gestiona la lógica de negocio, la autenticación de usuarios (JWT), las operaciones CRUD sobre la base de datos, y la comunicación en tiempo real a través de Socket.io.
- **langchain-api (Servicio de IA):** Un microservicio aislado que orquesta las interacciones con modelos de lenguaje externos (LLMs) a través de LangChain. Procesa las solicitudes de IA del backend y devuelve respuestas estructuradas.
- **Base de Datos (MongoDB):** Almacena toda la información persistente del sistema, incluyendo usuarios, proyectos, tareas, mensajes, etc.

4. Tecnologías Utilizadas

4.1. Stack Tecnológico

- **Stack Principal:** MERN (MongoDB, Express.js, React, Node.js) con TypeScript. ²
- **Inteligencia Artificial:** LangChain, OpenAI API.
- **Estilos y UI:** Tailwind CSS. ¹
- **Comunicación en Tiempo Real:** Socket.io, Firebase Cloud Messaging.
- **Contenerización:** Docker, Docker Compose.
- **Pruebas:** Jest, Postman, Cypress.

4.2. Justificación de Tecnologías

- **MERN con TypeScript:** Se eligió por ofrecer un ecosistema de desarrollo unificado basado en JavaScript/TypeScript, lo que mejora la productividad y la robustez del código gracias al tipado estático.
- **LangChain:** Permite construir aplicaciones de IA complejas y orquestadas, en lugar de simples llamadas a una API, facilitando la implementación de asistentes virtuales y otras funcionalidades avanzadas.
- **Docker:** Se adoptó para garantizar entornos de desarrollo consistentes y reproducibles, eliminando el problema de "funciona en mi máquina" y simplificando el despliegue. ²

5. Estructura del Código

5.1. Organización de Carpetas

El repositorio está organizado en tres directorios principales a nivel raíz, correspondiendo a cada microservicio:

- **gestion-APP/:** Contiene todo el código fuente del frontend de React.
- **backend/:** Contiene el código de la API principal de Node.js/Express.
- **langchain-api/:** Contiene el código del servicio dedicado a la IA.

5.2. Módulos Clave

- **Backend:**
 - **controllers/:** Lógica para manejar las solicitudes HTTP.
 - **models/:** Esquemas de Mongoose para la base de datos.
 - **routes/:** Definición de las rutas de la API.
 - **middleware/:** Funciones para autenticación y validación.
- **Frontend:**
 - **src/components/:** Componentes reutilizables de React (ej. TaskCard, KanbanColumn).
 - **src/pages/:** Componentes que representan las páginas de la aplicación (ej. LoginPage, ProjectDashboard).
 - **src/context/ o src/store/:** Lógica para la gestión del estado global.
 - **src/api/:** Funciones para realizar las llamadas a la API del backend.

6. Instalación y Configuración

6.1. Clonado del Repositorio

Bash

```
git clone https://github.com/GiovannyLESM/PGP-IA-TP2.git  
cd PGP-IA-TP2
```

6.2. Configuración de Variables de Entorno

Cada servicio (backend y langchain-api) requiere un archivo .env en su respectivo directorio. Cree los archivos basándose en los ejemplos (.env.example) que deberían existir en cada carpeta.

Ejemplo para backend/.env:

```
DATABASE_URL=mongodb://mongo:27017/proyectos  
JWT_SECRET=tu_secreto_jwt
```

Ejemplo para langchain-api/.env:

```
OPENAI_API_KEY=tu_api_key_de_openai
```

6.3. Levantamiento del Entorno

El proyecto está contenerizado con Docker. Para levantar cada servicio:

1. **Navegue al directorio del servicio** (ej. cd backend o cd gestion-APP).
2. **Construya e inicie los contenedores:**

Bash

```
docker-compose up --build
```

3. Para detener los servicios, use docker-compose down en el mismo directorio.

7. Base de Datos

7.1. Modelo de Datos

El sistema utiliza una base de datos NoSQL (MongoDB). Las entidades principales son:

- **User:** Almacena datos de los usuarios y credenciales.
- **Project:** Contiene la información del proyecto, el creador y los miembros.
- **List:** Representa las columnas del tablero Kanban.
- **Card:** Representa las tareas individuales, con detalles como descripción, asignados, checklists, etc.
- **Message:** Almacena los mensajes del chat de cada proyecto.

- **Invitation:** Gestiona las invitaciones a proyectos.

Las relaciones se manejan mediante referencias (ObjectId) y documentos embebidos para optimizar las consultas.

7.2. Conexión a la Base de Datos

La conexión a MongoDB se configura en el archivo .env del servicio backend a través de la variable DATABASE_URL. Mongoose gestiona el pool de conexiones.

8. Seguridad

8.1. Autenticación y Autorización

- **Autenticación:** Se implementa un sistema basado en JSON Web Tokens (JWT). Tras un login exitoso, el backend genera un token que el frontend almacena y envía en el encabezado Authorization de las solicitudes a rutas protegidas.
- **Autorización:** Se definen roles (Administrador, Miembro) que se verifican mediante middleware en el backend para restringir el acceso a ciertas funcionalidades (ej. el panel de administración).¹ⁱ

8.2. Protección de API

- Las contraseñas de los usuarios se hashean con bcryptjs antes de ser almacenadas en la base de datos.¹
- Se utilizan variables de entorno para gestionar claves secretas y de API, las cuales no deben ser subidas al control de versiones.

9. Pruebas

9.1. Tipos de Pruebas Realizadas

Se ha implementado una estrategia de pruebas multinivel para asegurar la calidad del software, tal como lo exige la consigna del proyecto:

1. **Pruebas Unitarias (Backend):** Pruebas a nivel de función para cada controlador, validando la lógica de negocio, manejo de errores y permisos.
2. **Pruebas de API (Backend):** Pruebas funcionales sobre los endpoints de la API para verificar el comportamiento completo de las solicitudes y respuestas.
3. **Pruebas de Interfaz de Usuario (E2E - Frontend):** Pruebas que simulan flujos de usuario completos en el navegador para validar la experiencia del usuario final.

9.2. Ejecución de Pruebas

- **Pruebas Unitarias (Jest):**

[Bash](#)

```
# Dentro del directorio 'backend'  
npm test
```

- **Pruebas de API (Postman):**

- Importar la colección de Postman del proyecto y ejecutar las solicitudes manualmente o usando el Collection Runner.

- **Pruebas E2E (Cypress):**

```
Bash  
# Dentro del directorio 'gestion-APP'  
npm run cypress:open
```

9.3. Herramientas Utilizadas

- **Jest:** Para las pruebas unitarias del backend.
- **Postman:** Para las pruebas manuales y de validación de la API.
- **Cypress:** Para las pruebas End-to-End del frontend.

10. Mantenimiento y Actualizaciones

10.1. Buenas Prácticas

- Actualizar las dependencias del proyecto periódicamente (npm update) y ejecutar la suite de pruebas para detectar regresiones.
- Mantener la documentación (READMEs, manuales) actualizada con cada cambio significativo.

10.2. Agregar Nuevas Funcionalidades

1. Crear una nueva rama desde develop siguiendo el flujo de Git (feature/nombre-funcionalidad).
2. Desarrollar la funcionalidad.
3. Añadir las pruebas unitarias y E2E correspondientes.
4. Actualizar la documentación si es necesario.
5. Crear un Pull Request hacia develop para la revisión del código.

11. Solución de Problemas Comunes

- **Error: Connection refused al conectar frontend con backend.**
 - **Solución:** Asegurarse de que todos los contenedores de Docker (backend, gestion-APP, mongo) estén en ejecución y en la misma red de Docker. Verificar que las URLs de la API en los archivos .env del frontend sean correctas.
- **Error: 401 Unauthorized en la API.**
 - **Solución:** Verificar que el token JWT se esté enviando correctamente en el encabezado Authorization: Bearer <token>. El token puede haber expirado, requiriendo un nuevo inicio de sesión.

- **Error: Fallo en la construcción de Docker (docker-compose up --build).**
 - **Solución:** Revisar la sintaxis del Dockerfile y docker-compose.yml. Ejecutar docker system prune -a para limpiar contenedores, imágenes y volúmenes antiguos que puedan estar causando conflictos.

12. Apéndices

12.1. APIs Externas Utilizadas

- **OpenAI API:** Para las funcionalidades del asistente virtual y predicción de tiempos.
- **Firebase Cloud Messaging API:** Para el envío de notificaciones push.

12.2. Enlaces a Documentación

- (<https://react.dev/>)
- (<https://nodejs.org/en/docs>)
- (<https://www.mongodb.com/docs/>)
- (<https://docs.docker.com/>)
- (<https://js.langchain.com/docs/>)
- (<https://github.com/GiovannyLESM/PGP-IA-TP2>)
- (<https://continental-team-up7irt2y.atlassian.net/jira/software/projects/SCRUM/boards/1/>)

PGP-IA-TP2

Taller de Proyectos 2 - Plataforma de Gestión de Proyectos con Inteligencia Artificial - MERN

Gestión APP – Entorno de Desarrollo con Docker

Este proyecto usa React + Vite + Tailwind CSS con un entorno de desarrollo totalmente aislado en Docker para facilitar la colaboración entre desarrolladores sin conflictos de versiones ni configuraciones locales.

Requisitos

- Tener instalado [Docker](#) y Docker Compose

Iniciar el entorno de desarrollo front


```
# Clona el proyecto
git clone https://github.com/usuario/gestion-APP.git
cd gestion-APP


# Construye e inicia el contenedor
docker-compose up --build
```


Iniciar el entorno de desarrollo back

```
# Clona el proyecto
git clone https://github.com/usuario/gestion-APP.git
cd backend

# Construye e inicia el contenedor
docker-compose up --build
```

Luego abre tu navegador en:  <http://localhost:5173>

 Este comando compila la imagen, instala dependencias y ejecuta npm run dev dentro del contenedor.

 Reiniciar sin volver a construir Una vez construido, puedes simplemente iniciar con:

```
docker-compose up
```

Solo usa --build si cambiaste package.json, Dockerfile, o agregaste nuevas dependencias.

▼ Detener el contenedor Para detener el entorno de desarrollo:

```
docker-compose down
```