MANUAL TÉCNICO – SNAP MATH

Versión: 1.0.0
Fecha de publicación: 2025
Desarrollado por TEDICH

Indicie

1. INTRODUCCIÓN

Snap Math es una plataforma educativa diseñada para asistir a estudiantes y profesores en la resolución,
gestión y supervisión de actividades académicas. El sistema integra Inteligencia Artificial basada en Gemini,
un robusto backend en Node.js con Express, y un frontend moderno construido en React.

Este manual técnico proporciona todos los detalles internos del sistema, necesarios para mantenimiento,
actualización, escalamiento, auditoría y comprensión profunda de la arquitectura de software.

2. OBJETIVOS DEL SISTEMA

El propósito principal de Snap Math es ofrecer una herramienta académica inteligente capaz de:
- Resolver problemas paso a paso mediante IA sin ofrecer la respuesta final al estudiante.
- Permitir a profesores administrar clases, tareas, ejercicios y recursos educativos.
- Crear entornos interactivos para fortalecer el aprendizaje.
- Permitir interacción entre módulos educativos, base de datos y algoritmos de inteligencia artificial.

OBJETIVOS TÉCNICOS:
- Escalabilidad total del backend.
- Modularidad en controladores, rutas, middleware y servicios.
- Seguridad a través de JWT, Helmet, validaciones y sanitización.
- Optimización en la comunicación con Gemini.

3. ARQUITECTURA GENERAL

Snap Math utiliza una arquitectura cliente–servidor compuesta por tres capas principales:

1. Capa de Presentación (Frontend React):
   - Maneja la interfaz del usuario.
   - Implementa rutas, componentes, contextos y consumo de API REST.
   - Administra sesiones mediante tokens y protección de rutas.

2. Capa Lógica (Backend Node.js + Express):
   - Exposición de endpoints REST.
   - Gestión de seguridad, autenticación y autorización.
   - Conexión con base de datos MySQL mediante mysql2.
   - Módulo dedicado a comunicación con GEMINI.
   - Subida de archivos con Multer.
   - Validación mediante express-validator.

3. Capa de Datos (MySQL):
   - Bases de datos relacionales.
   - Relaciones N:M entre estudiantes y clases.
   - Índices para optimización de consultas.

4. TECNOLOGÍAS DETALLADAS

Frontend React (Iteración 1):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 1):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 1):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 1):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 2):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 2):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 2):

- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 2):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 3):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 3):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 3):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 3):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 4):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 4):
- Enrutamiento modular con separación por controladores.

- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 4):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 4):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 5):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 5):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 5):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 5):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 6):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.

- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 6):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 6):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 6):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 7):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 7):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 7):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 7):
- Procesamiento de lenguaje natural.

- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 8):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 8):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 8):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 8):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 9):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 9):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 9):

- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 9):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 10):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 10):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 10):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 10):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 11):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 11):
- Enrutamiento modular con separación por controladores.

- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 11):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 11):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 12):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 12):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 12):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 12):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 13):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.

- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 13):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 13):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 13):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 14):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 14):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 14):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 14):
- Procesamiento de lenguaje natural.

- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 15):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 15):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 15):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 15):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 16):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 16):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 16):

- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 16):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 17):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 17):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 17):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 17):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 18):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 18):
- Enrutamiento modular con separación por controladores.

- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 18):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 18):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 19):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 19):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 19):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 19):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 20):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.

- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 20):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 20):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 20):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 21):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 21):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 21):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 21):
- Procesamiento de lenguaje natural.

- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 22):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 22):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 22):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 22):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 23):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 23):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 23):

- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 23):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 24):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 24):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 24):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 24):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 25):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 25):
- Enrutamiento modular con separación por controladores.

- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 25):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 25):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 26):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 26):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 26):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 26):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 27):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.

- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 27):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 27):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 27):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 28):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 28):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 28):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 28):
- Procesamiento de lenguaje natural.

- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 29):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 29):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 29):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 29):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 30):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 30):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 30):

- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 30):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 31):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 31):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 31):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 31):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 32):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 32):
- Enrutamiento modular con separación por controladores.

- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 32):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 32):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 33):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 33):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 33):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 33):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 34):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.

- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 34):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 34):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 34):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 35):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 35):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 35):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 35):
- Procesamiento de lenguaje natural.

- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 36):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 36):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 36):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 36):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 37):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 37):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 37):

- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 37):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 38):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 38):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 38):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 38):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 39):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 39):
- Enrutamiento modular con separación por controladores.

- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 39):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 39):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

Frontend React (Iteración 40):
- React 18 con componentes funcionales y Hooks.
- React Router para navegación multipágina.
- Context API para manejo global de estado.
- Axios para peticiones HTTP seguras con header Bearer Token.
- Uso de componentes desacoplados bajo arquitectura limpia.

Backend Node.js + Express (Iteración 40):
- Enrutamiento modular con separación por controladores.
- Middlewares para validación, seguridad, roles y permisos.
- Librerías clave: helmet, jsonwebtoken, multer, express-validator.
- Control de errores centralizado.
- Logs para monitoreo y auditoría del sistema.

Base de Datos MySQL (Iteración 40):
- Relaciones entre tablas.
- Índices, claves primarias, claves foráneas.
- Procedimientos almacenados opcionales.

IA Gemini (Iteración 40):
- Procesamiento de lenguaje natural.
- Análisis de imágenes enviado desde frontend.
- Respuestas formateadas paso a paso.

5. DOCUMENTACIÓN COMPLETA DE ENDPOINTS API

ENDPOINT #1:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #1:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #1:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #2:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #2:
POST /classes
Body:
{{

"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #2:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #3:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #3:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #3:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #4:
POST /auth/login
Descripción: Login de usuario, generación de JWT.

Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #4:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #4:
POST /assistant/ask
Body:
{{
 "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #5:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #5:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #5:

POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #6:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #6:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #6:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #7:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #7:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #7:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #8:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #8:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #8:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #9:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #9:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #9:
POST /assistant/ask
Body:
{{
 "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #10:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #10:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",

"description": "Clase introductoria"
}}

ENDPOINT IA #10:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #11:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #11:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #11:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #12:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{

"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #12:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #12:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #13:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #13:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #13:
POST /assistant/ask
Body:

{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #14:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #14:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #14:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #15:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #15:
POST /classes

Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #15:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #16:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #16:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #16:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #17:

POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #17:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #17:
POST /assistant/ask
Body:
{{
 "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #18:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #18:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #18:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #19:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #19:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #19:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #20:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"

}}

ENDPOINT CLASS MANAGEMENT #20:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #20:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #21:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #21:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #21:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"

}}

ENDPOINT #22:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #22:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #22:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #23:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #23:
POST /classes
Body:
{{

"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #23:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #24:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #24:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #24:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #25:
POST /auth/login
Descripción: Login de usuario, generación de JWT.

Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #25:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #25:
POST /assistant/ask
Body:
{{
 "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #26:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #26:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #26:

POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #27:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #27:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #27:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #28:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #28:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #28:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #29:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #29:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #29:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #30:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #30:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #30:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #31:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #31:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",

"description": "Clase introductoria"
}}

ENDPOINT IA #31:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #32:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #32:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #32:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #33:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{

"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #33:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #33:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #34:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #34:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #34:
POST /assistant/ask
Body:

{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #35:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #35:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #35:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #36:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #36:
POST /classes

Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #36:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #37:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #37:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #37:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #38:

POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #38:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #38:
POST /assistant/ask
Body:
{{
 "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #39:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #39:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #39:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #40:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #40:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #40:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #41:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"

}}

ENDPOINT CLASS MANAGEMENT #41:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #41:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #42:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #42:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #42:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"

}}

ENDPOINT #43:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #43:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #43:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #44:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #44:
POST /classes
Body:
{{

"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #44:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #45:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
"email": "usuario@example.com",
"password": "********"
}}

ENDPOINT CLASS MANAGEMENT #45:
POST /classes
Body:
{{
"name": "Álgebra",
"subject": "Matemáticas",
"description": "Clase introductoria"
}}

ENDPOINT IA #45:
POST /assistant/ask
Body:
{{
"question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #46:
POST /auth/login
Descripción: Login de usuario, generación de JWT.

Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #46:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #46:
POST /assistant/ask
Body:
{{
 "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #47:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
 "email": "usuario@example.com",
 "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #47:
POST /classes
Body:
{{
 "name": "Álgebra",
 "subject": "Matemáticas",
 "description": "Clase introductoria"
}}

ENDPOINT IA #47:

POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #48:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #48:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #48:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #49:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #49:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #49:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

ENDPOINT #50:
POST /auth/login
Descripción: Login de usuario, generación de JWT.
Body esperado:
{{
  "email": "usuario@example.com",
  "password": "********"
}}

ENDPOINT CLASS MANAGEMENT #50:
POST /classes
Body:
{{
  "name": "Álgebra",
  "subject": "Matemáticas",
  "description": "Clase introductoria"
}}

ENDPOINT IA #50:
POST /assistant/ask
Body:
{{
  "question": "Explica el teorema de Pitágoras"
}}

6. MODELO DE BASE DE DATOS (DETAILED)

Tabla users (Iteración 1):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 1):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 1):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 1):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 2):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 2):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 2):
- id INT PK

- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 2):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 3):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 3):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 3):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 3):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 4):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 4):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 4):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 4):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 5):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 5):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 5):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 5):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT

- created_at TIMESTAMP

Tabla users (Iteración 6):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 6):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 6):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 6):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 7):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 7):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 7):
- id INT PK
- class_id INT FK

- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 7):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 8):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 8):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 8):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 8):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 9):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 9):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 9):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 9):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 10):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 10):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 10):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 10):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 11):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 11):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 11):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 11):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 12):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 12):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 12):
- id INT PK
- class_id INT FK
- title VARCHAR(160)

- description TEXT

Tabla ai_logs (Iteración 12):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 13):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 13):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 13):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 13):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 14):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 14):

- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 14):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 14):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 15):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 15):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 15):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 15):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 16):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 16):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 16):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 16):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 17):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 17):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 17):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 17):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 18):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 18):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 18):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 18):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 19):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 19):
- id INT PK AI

- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 19):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 19):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 20):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 20):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 20):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 20):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 21):

- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 21):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 21):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 21):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 22):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 22):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 22):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 22):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 23):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 23):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 23):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 23):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 24):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 24):
- id INT PK AI
- name VARCHAR(120)

- subject VARCHAR(120)

Tabla tasks (Iteración 24):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 24):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 25):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 25):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 25):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 25):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 26):
- id INT PK AI

- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 26):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 26):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 26):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 27):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 27):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 27):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 27):

- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 28):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 28):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 28):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 28):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 29):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 29):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 29):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 29):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 30):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 30):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 30):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 30):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 31):
- id INT PK AI
- name VARCHAR(120)

- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 31):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 31):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 31):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 32):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 32):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 32):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 32):
- id INT PK

- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 33):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 33):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 33):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 33):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 34):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 34):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 34):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 34):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 35):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 35):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 35):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 35):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 36):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE

- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 36):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 36):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 36):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 37):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 37):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 37):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 37):
- id INT PK
- user_id FK

- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 38):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 38):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 38):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 38):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 39):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 39):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 39):

- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 39):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 40):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 40):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 40):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 40):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 41):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)

- role ENUM('student','teacher')

Tabla classes (Iteración 41):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 41):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 41):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 42):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 42):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 42):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 42):
- id INT PK
- user_id FK
- input TEXT

- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 43):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 43):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 43):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 43):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 44):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 44):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 44):
- id INT PK

- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 44):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 45):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 45):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 45):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 45):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 46):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 46):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 46):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 46):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 47):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 47):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 47):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 47):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT

- created_at TIMESTAMP

Tabla users (Iteración 48):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 48):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 48):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 48):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 49):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 49):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 49):
- id INT PK
- class_id INT FK

- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 49):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 50):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 50):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 50):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 50):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 51):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 51):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 51):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 51):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 52):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 52):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 52):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 52):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 53):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 53):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 53):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 53):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 54):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 54):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 54):
- id INT PK
- class_id INT FK
- title VARCHAR(160)

- description TEXT

Tabla ai_logs (Iteración 54):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 55):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 55):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 55):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 55):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 56):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 56):

- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 56):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 56):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 57):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 57):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 57):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 57):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 58):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 58):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 58):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 58):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 59):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 59):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 59):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 59):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

Tabla users (Iteración 60):
- id INT PK AI
- name VARCHAR(120)
- email VARCHAR(160) UNIQUE
- password VARCHAR(255)
- role ENUM('student','teacher')

Tabla classes (Iteración 60):
- id INT PK AI
- name VARCHAR(120)
- subject VARCHAR(120)

Tabla tasks (Iteración 60):
- id INT PK
- class_id INT FK
- title VARCHAR(160)
- description TEXT

Tabla ai_logs (Iteración 60):
- id INT PK
- user_id FK
- input TEXT
- output LONGTEXT
- created_at TIMESTAMP

7. EJEMPLOS DE CÓDIGO COMPLETO DEL BACKEND

--- Código ejemplo 1 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 2 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 3 ---

```
const express = require('express');
const router = express.Router();
```

```
router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 4 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 5 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
```

```
});

module.exports = router;
```

--- Código ejemplo 6 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 7 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 8 ---

```
const express = require('express');
```

```
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 9 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 10 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
```

```
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 11 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 12 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 13 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 14 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 15 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});
```

```
  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 16 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 17 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 18 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 19 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 20 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});
```

```
  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 21 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 22 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 23 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 24 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 25 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
```

```
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 26 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 27 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 28 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 29 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 30 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
```

```
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 31 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 32 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 33 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 34 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 35 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
```

```
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 36 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});

module.exports = router;
```

--- Código ejemplo 37 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
  const {{ email, password }} = req.body;
  const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
  if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

  const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
  res.json({{ token }});
}});
```

```
module.exports = router;
```

--- Código ejemplo 38 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 39 ---

```
const express = require('express');
const router = express.Router();

router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

--- Código ejemplo 40 ---

```
const express = require('express');
const router = express.Router();
```

```
router.post('/login', async (req, res) => {
   const {{ email, password }} = req.body;
   const user = await db.query("SELECT * FROM users WHERE email = ?", [email]);
   if (!user) return res.status(404).json({{ msg: "Usuario no encontrado" }});

   const token = jwt.sign({{ id: user.id, role: user.role }}, process.env.JWT_SECRET);
   res.json({{ token }});
});

module.exports = router;
```

8. DOCUMENTACIÓN DETALLADA DEL FRONTEND (REACT)

React Component Rendering Cycle (Iteración 1):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:

```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 2):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:

```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 3):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props

- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 4):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 5):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

```
   </div>
  );
}

<b>React Component Rendering Cycle (Iteración 6):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}

<b>React Component Rendering Cycle (Iteración 7):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}

<b>React Component Rendering Cycle (Iteración 8):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
```

- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 9):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 10):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

```
    <h2>Nombre de Clase</h2>
   </div>
  );
}

<b>React Component Rendering Cycle (Iteración 11):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
   <div className='card'>
    <h2>Nombre de Clase</h2>
   </div>
  );
}

<b>React Component Rendering Cycle (Iteración 12):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
   <div className='card'>
    <h2>Nombre de Clase</h2>
   </div>
  );
}

<b>React Component Rendering Cycle (Iteración 13):</b>
- Módulo de clases
- Módulo de tareas
```

- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 14):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 15):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

```
   <div className='card'>
     <h2>Nombre de Clase</h2>
   </div>
 );
}

<b>React Component Rendering Cycle (Iteración 16):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
 return (
   <div className='card'>
     <h2>Nombre de Clase</h2>
   </div>
 );
}

<b>React Component Rendering Cycle (Iteración 17):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
 return (
   <div className='card'>
     <h2>Nombre de Clase</h2>
   </div>
 );
}

<b>React Component Rendering Cycle (Iteración 18):</b>
- Módulo de clases
```

- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 19):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 20):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {

```
 return (
   <div className='card'>
     <h2>Nombre de Clase</h2>
   </div>
 );
}

<b>React Component Rendering Cycle (Iteración 21):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
 return (
   <div className='card'>
     <h2>Nombre de Clase</h2>
   </div>
 );
}

<b>React Component Rendering Cycle (Iteración 22):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
 return (
   <div className='card'>
     <h2>Nombre de Clase</h2>
   </div>
 );
}

<b>React Component Rendering Cycle (Iteración 23):</b>
```

- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 24):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 25):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:

```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 26):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:

```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 27):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:

```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 28):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 29):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 30):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 31):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 32):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 33):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 34):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
```
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}
```

React Component Rendering Cycle (Iteración 35):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 36):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 37):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );

```
}

<b>React Component Rendering Cycle (Iteración 38):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}

<b>React Component Rendering Cycle (Iteración 39):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}

<b>React Component Rendering Cycle (Iteración 40):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
```

- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 41):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 42):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

```
  );
}

<b>React Component Rendering Cycle (Iteración 43):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}

<b>React Component Rendering Cycle (Iteración 44):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}

<b>React Component Rendering Cycle (Iteración 45):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
```

- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 46):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

React Component Rendering Cycle (Iteración 47):
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

```
   </div>
  );
}

<b>React Component Rendering Cycle (Iteración 48):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}

<b>React Component Rendering Cycle (Iteración 49):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

<b>Ejemplo de componente React:</b>
function ClassCard() {
  return (
    <div className='card'>
      <h2>Nombre de Clase</h2>
    </div>
  );
}

<b>React Component Rendering Cycle (Iteración 50):</b>
- Módulo de clases
- Módulo de tareas
- Vista de ejercicios
```

- Componentes dinámicos usando props
- Renderizado condicional
- Manejo de estado con Hooks

Ejemplo de componente React:
function ClassCard() {
  return (

      <h2>Nombre de Clase</h2>

  );
}

9. SEGURIDAD DEL SISTEMA

JWT Security Layer (Iteración 1):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 1):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 1):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 1):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 2):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 2):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 2):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 2):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 3):

- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 3):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 3):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 3):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 4):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 4):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 4):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 4):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 5):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 5):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 5):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 5):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 6):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 6):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 6):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 6):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 7):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 7):
- Prevención de XSS.
- Protección de cabeceras.

- HSTS.

Rate Limit (Iteración 7):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 7):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 8):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 8):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 8):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 8):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 9):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 9):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 9):
- Evita fuerza bruta.

- Límites configurables.

Sanitización (Iteración 9):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 10):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 10):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 10):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 10):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 11):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 11):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 11):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 11):
- Entradas validadas.

- Prevención de SQL Injection.

JWT Security Layer (Iteración 12):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 12):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 12):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 12):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 13):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 13):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 13):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 13):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 14):
- Validez del token.

- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 14):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 14):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 14):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 15):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 15):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 15):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 15):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 16):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 16):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 16):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 16):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 17):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 17):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 17):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 17):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 18):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 18):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 18):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 18):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 19):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 19):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 19):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 19):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 20):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 20):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 20):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 20):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 21):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 21):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 21):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 21):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 22):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 22):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 22):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 22):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 23):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 23):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 23):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 23):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 24):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 24):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 24):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 24):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 25):
- Validez del token.
- Protección contra expiración.

- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 25):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 25):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 25):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 26):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 26):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 26):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 26):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 27):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 27):

- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 27):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 27):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 28):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 28):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 28):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 28):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 29):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 29):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 29):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 29):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 30):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 30):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 30):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 30):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 31):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 31):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 31):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 31):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 32):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 32):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 32):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 32):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 33):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 33):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 33):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 33):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 34):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 34):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 34):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 34):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 35):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 35):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 35):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 35):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 36):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.

- Validación en middleware.

Helmet Security (Iteración 36):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 36):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 36):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 37):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 37):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 37):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 37):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 38):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 38):
- Prevención de XSS.

- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 38):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 38):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 39):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 39):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 39):
- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 39):
- Entradas validadas.
- Prevención de SQL Injection.

JWT Security Layer (Iteración 40):
- Validez del token.
- Protección contra expiración.
- Firmado con clave secreta.
- Validación en middleware.

Helmet Security (Iteración 40):
- Prevención de XSS.
- Protección de cabeceras.
- HSTS.

Rate Limit (Iteración 40):

- Evita fuerza bruta.
- Límites configurables.

Sanitización (Iteración 40):
- Entradas validadas.
- Prevención de SQL Injection.