UNIVERSIDAD CONTINENTAL

FACULTAD DE INGENIERÍA

ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

PROYECTO

“Tutor Virtual De Lectura Crítica”

PRESENTADO POR:

APELLIDOS Y NOMBRES	CÓDIGO
Dueñas Guerra Jesús Korlant	72122566
Gutierrez Medina Jesús Manuel	75826567
Chavez Estrella José Jhovanni	75557750
Lopez Idone Jose Gianmarco	71698090
Reymundo Rodriguez Cristhian Jhon	75049277
Asistente Virtual	

ASESOR:

GAMARRA MORENO, JOB DANIEL

HUANCAYO – PERÚ

2025

LISTA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

CAPÍTULO 1
PLANTEAMIENTO DEL ESTUDIO

1.1. Aspectos Generales de la Empresa

Dado que este proyecto se gesta en un entorno académico, la "empresa" se define como la organización del equipo de proyecto bajo el auspicio de la Universidad Continental, específicamente dentro de la Facultad de Ingeniería de Sistemas e Informática.

1.1.1. Organigrama

El equipo de trabajo está estructurado para cubrir todas las fases del ciclo de vida del desarrollo de software (SDLC), con roles claramente definidos para garantizar la calidad y el cumplimiento de los plazos. La estructura organizativa es la siguiente:

- Sponsor: Universidad Continental.
- Gerente del Proyecto / Líder: Dueñas Guerra Jesús Korlant, encargado de la coordinación general, el seguimiento de los sprints y la comunicación con los stakeholders.
- Equipo de Desarrollo:
    - Frontend Specialist: José Jhovanni Chavez Estrella, responsable de la interfaz de usuario en React.js y la gestión del estado.
    - Backend Specialist: Jose Gianmarco Lopez Idone, encargado del desarrollo de la API REST con Express.js y la lógica de negocio.
    - IA & Automation: Jesús Manuel Gutierrez Medina, responsable de la integración de modelos de NLP y flujos en n8n.
    - Tester & Documentación: Cristhian Jhon Reymundo Rodriguez, enfocado en las pruebas unitarias/E2E y la documentación técnica.

1.1.2. Misión y visión

- Misión (del Proyecto): Desarrollar una aplicación web interactiva que sirva como tutor virtual de lectura crítica, apoyada en inteligencia artificial (NLP) y automatización, para impactar en la formación académica de estudiantes universitarios.
- Visión (del Proyecto): Ser una herramienta accesible y sostenible que mejore la comprensión lectora y el pensamiento crítico, ofreciendo rutas personalizadas de estudio y retroalimentación inmediata mediante el uso de tecnologías emergentes.

1.2. Diagnóstico del Problema

Se ha identificado una deficiencia en el desarrollo de habilidades de lectura crítica en el entorno universitario. Actualmente, los estudiantes enfrentan retos para recibir retroalimentación inmediata y personalizada sobre su comprensión lectora debido a la limitación de tiempo de los docentes y la falta de herramientas automáticas.

- **Necesidad:** Los estudiantes requieren reforzar sus habilidades académicas de forma autónoma.
- **Limitaciones actuales:** Procesos manuales de evaluación y falta de detección automática de falacias o sesgos en los textos de estudio.

1.3. Procesos de la Empresa (Procesos Educativos Actuales)

El proceso actual de comprensión lectora y evaluación suele seguir un flujo tradicional y manual:

1. **Asignación de lectura:** El docente entrega textos físicos o digitales.
2. **Lectura individual:** El estudiante lee sin acompañamiento inmediato.
3. **Evaluación diferida:** La retroalimentación ocurre días después, limitando la corrección temprana de errores de interpretación.

4. **Ausencia de herramientas de análisis:** No existen mecanismos automatizados accesibles para que el estudiante valide si un texto contiene sesgos o falacias lógicas antes de analizarlo.

1.4. Oportunidad Encontrada

La integración de la Inteligencia Artificial (IA) y la automatización presenta una oportunidad para optimizar el aprendizaje autónomo.

- **Uso de NLP (Procesamiento de Lenguaje Natural):** Permite generar preguntas contextuales (literales, inferenciales y críticas) automáticamente a partir de cualquier texto cargado.
- **Automatización con n8n:** Facilita la gestión de recordatorios de estudio y el seguimiento del progreso del estudiante sin intervención manual constante.

1.5. Detalles del Proyecto

- **Nombre del Proyecto:** Tutor Virtual de Lectura Crítica ("AppComprende-IA").
- **Fecha de Inicio:** 28/08/2025.
- **Fecha de Finalización Prevista:** 04/12/2025.
- **Objetivo General:** Desarrollar una aplicación web full-stack con el stack MERN (MongoDB, Express, React, Node.js) que permita evaluar y mejorar la comprensión lectora de estudiantes universitarios.
- **Alcance Técnico:**
    - Frontend: React.js (Context API/Redux).
    - Backend: Express.js y API REST.
    - Base de Datos: MongoDB Atlas.
    - IA: Integración con Hugging Face/OpenAI.
    - Despliegue: Contenedores Docker y servicios en la nube (Render/Vercel).

CAPÍTULO 2
ESTUDIO DE FACTIBILIDAD

2.1. Alternativas de Solución

Para abordar el problema de la comprensión lectora, se evaluaron las siguientes alternativas:

1. **Alternativa Seleccionada (Aplicación Web con IA):** Desarrollo de una plataforma web accesible desde cualquier navegador, integrando APIs de IA para análisis de texto en tiempo real. Es la opción más viable por su accesibilidad y capacidad de integración.
2. **Alternativa Descartada (Aplicación Móvil Nativa):** Se descartó debido a la mayor complejidad de mantenimiento para múltiples sistemas operativos (iOS/Android) y porque el análisis de textos académicos extensos es más cómodo en pantallas de escritorio/web.
3. **Alternativa Descartada (Tutoría puramente humana):** No escalable debido a la limitación de recursos humanos y tiempo de los docentes.

2.2. Factibilidad Técnica

El proyecto es técnicamente viable gracias a la disponibilidad de herramientas y conocimientos del equipo:

- **Stack Tecnológico:** El equipo cuenta con conocimientos en el stack MERN y está fortaleciendo sus capacidades en integración de IA mediante autoestudio.
- **Disponibilidad de APIs:** Se ha validado que los modelos de IA necesarios (Hugging Face/OpenAI) disponen de APIs accesibles y funcionales en sus capas gratuitas para el desarrollo.
- **Herramientas de Desarrollo:** Se utilizará software de código abierto y gratuito como Docker para la contenerización, Git para el control de versiones y n8n para la automatización, eliminando barreras de licenciamiento.

2.3. Factibilidad Económica

El proyecto se ha diseñado bajo un modelo de presupuesto académico simulado, optimizando recursos para operar con costos reales mínimos o nulos mediante el uso de tiers gratuitos de servicios en la nube.

2.3.1. Gastos generales (Presupuesto Estimado)

A continuación se detalla la estructura de costos estimada para la ejecución del proyecto:

Categoría	Detalle	Costo Estimado (S/.)
Infraestructura	Servicios Cloud (Render/Vercel, MongoDB Atlas).	800.00
Licencias de IA / APIs	Créditos para OpenAI API / Hugging Face.	1,200.00
Herramientas Automatización	n8n (hosting propio gratuito).	0.00
Contenerización y CI/CD	Docker y GitHub Actions (Plan educativo).	0.00
Recursos Humanos	Trabajo académico no remunerado (7 integrantes).	0.00
Documentación	Manuales e informes técnicos.	300.00

Reserva de Contingencia	15% para imprevistos técnicos.	375.00
Reserva de Gestión	10% para cambios de alcance.	250.00
TOTAL ESTIMADO	**(Costo Simulado Académico)**	**2,925.00**

Tabla 1: Gastos Generales

2.4. Factibilidad Operacional

La operatividad del sistema está garantizada por su diseño centrado en el usuario y la infraestructura moderna:

2.4.1. Operatividad del Sistema de Tutoría

- **Accesibilidad:** La aplicación será web y responsive, cumpliendo con normas WCAG 2.1 AA, lo que garantiza que estudiantes y docentes puedan acceder desde diversos dispositivos con conexión a internet.
- **Aceptación del Usuario:** Se han identificado riesgos de resistencia inicial al uso de herramientas digitales , los cuales se mitigarán mediante pruebas piloto tempranas y sesiones de retroalimentación con usuarios reales.
- **Mantenimiento:** La arquitectura basada en Docker y CI/CD facilita el despliegue continuo y la corrección de errores sin interrumpir el servicio.
- **Interacción:** El sistema automatiza flujos críticos (notificaciones, asignación de tareas) mediante n8n, reduciendo la carga operativa manual para la administración del curso.

CAPÍTULO 3
ANÁLISIS DE REQUERIMIENTOS

3.1. Metas del Sistema de Información

El objetivo fundamental del proyecto es desarrollar una aplicación web completa (full-stack) utilizando el stack MERN (MongoDB, Express, React, Node.js) que sirva como un Tutor Virtual de Lectura Crítica. Las metas específicas del sistema son:

3.2. Requisitos del Sistema

Los requisitos se basan en la necesidad identificada de reforzar las habilidades académicas de forma autónoma.

3.2.1. Requerimientos funcionales

El sistema deberá permitir a los usuarios (estudiantes y docentes) realizar las siguientes funciones clave:

- **Gestión de Contenido:** El usuario podrá cargar textos académicos o documentos para su análisis.
- **Análisis de Lectura Crítica (IA):** El sistema debe generar automáticamente preguntas de diversos niveles (literal, inferencial, crítico) basadas en el texto cargado.
- **Detección de Sesgos/Falacias:** El sistema debe identificar y señalar falacias lógicas o sesgos presentes en el texto.
- **Registro y Seguimiento:** El sistema debe permitir la creación de perfiles de usuario y el seguimiento individualizado del progreso en la comprensión lectora.
- **Notificaciones:** Uso de automatización (n8n) para enviar recordatorios de estudio o notificaciones de progreso.
- **Autoevaluación:** El estudiante podrá interactuar con las preguntas generadas y recibir puntajes de autoevaluación inmediatos.

- **Acceso (Login):** Debe contar con una interfaz de acceso o *login* para autenticar a los actores.

3.2.2. Requerimientos no funcionales

Estos requisitos aseguran la calidad operativa y técnica del sistema:

- **Tecnología:** El sistema debe estar desarrollado con el *stack* **MERN** (MongoDB, Express, React, Node.js).
- **Accesibilidad y Usabilidad:** La aplicación será **web y responsive**, garantizando acceso desde diversos dispositivos con conexión a internet y cumpliendo con las normas **WCAG 2.1 AA**.
- **Despliegue:** El *deployment* debe usar **Contenedores Docker** y servicios en la nube (Render/Vercel) para facilitar el despliegue continuo y el mantenimiento.
- **Rendimiento:** El análisis de texto con IA debe ser en **tiempo real** o casi real para asegurar la retroalimentación inmediata.
- **Mantenimiento:** La arquitectura basada en Docker y CI/CD debe facilitar la corrección de errores sin interrumpir el servicio.

3.3. Identificación de Actores del Sistema

Actor	Rol en el Proyecto	Intereses / Implicación
Sponsor	Universidad Continental	Patrocinio, proveer el entorno académico
Usuario Principal	Estudiante Universitario	Mejorar su comprensión lectora, obtener retroalimentación inmediata

Usuario Secundario	Docente	Evaluar la efectividad, asignar textos, recibir informes del progreso de los estudiantes
Asesor	Gamarra Moreno, Job Daniel	Supervisión académica y metodológica
Equipo de Desarrollo	Líder, Especialistas	Ejecución técnica y cumplimiento del alcance

Tabla 2: Actores del Sistema

CAPÍTULO 4
PLANIFICACIÓN DEL PROYECTO

4.1. Definición de Roles de Trabajo

El equipo de desarrollo está conformado por 5 integrantes con roles claramente definidos:

4.1.1. Product owner

Product Owner	Dueñas Guerra Jesús Korlant	Liderar el proyecto, definir y priorizar el Product Backlog

4.1.2. Scrum master

Scrum Master	*Asumido por el Líder*	Facilitar el proceso Scrum, eliminar impedimentos.

4.1.3. Team member

Team Member / Frontend Specialist	Chavez Estrella José Jhovanni	Desarrollo de la interfaz de usuario con React.js
Team Member / Backend Specialist	Lopez Idone Jose Gianmarco	Desarrollo de la API REST y lógica de negocio con Express.js y Node.js
Team Member / IA & Automation	Gutierrez Medina Jesús Manuel	Integración con APIs de IA y configuración de flujos con n8n.

4.1.4. Tester

Tester & Documentación	Reymundo Rodriguez Cristhian Jhon	Ejecución de pruebas de software y documentación oficial

Tabla 3: Roles de Trabajo

4.2. Product Backlog

El Product Backlog (Pila de Producto) es una lista ordenada de todas las funcionalidades, requisitos, mejoras y correcciones que se desean implementar en el producto final. Este *backlog* se genera a partir de los Requisitos de Alto Nivel y la Visión del Proyecto, y se mantiene en herramientas de seguimiento como Jira.

4.3. Sprint Backlog

El Sprint Backlog es el conjunto de ítems del Product Backlog seleccionados para ser completados durante un *Sprint* específico, junto con el plan para entregar el Incremento del Producto y las tareas detalladas para su ejecución.

4.3.1. Sprint 1: Fundamentos y Gestión de Contenidos

Detalle	Asignación
Fecha de Inicio	**28/08/2025**
Fecha de Fin	**15/09/2025**
Duración	**2.5 Semanas (18 Días)**
Horas Estimadas	**280 horas**

Tabla 4: Sprint 1 Elaboración Propia

Objetivo: Establecer la arquitectura base (MERN), autenticación y manejo de archivos.

Historias de Usuario Abordadas

- **HU01:** Subida de textos (PDF, DOCX, TXT).
- **HU06 (Parcial):** Estructura de base de datos para usuarios (Docentes/Estudiantes).

Entregables Técnicos

- **Backend:** Configuración de Server (Express) y DB (MongoDB). `authController`: Login y Registro con JWT. `textController`: Endpoints para carga y lectura de archivos.
- **Frontend:** Configuración de React + Vite + Tailwind. Páginas de Login/Register. Componente `Upload.jsx` para estudiantes.

4.3.2. Sprint 2: Inteligencia Artificial y Generación de Evaluaciones

Detalle	Asignación
Fecha de Inicio	**16/09/2025**
Fecha de Fin	**06/10/2025**
Duración	**3 Semanas (21 Días)**
Horas Estimadas	**330 horas**

Tabla 5: Sprint 2 Elaboración Propia

Objetivo: Integrar el motor de IA (NLP) para generar contenido educativo automático.

Historias de Usuario Abordadas

- **HU02: Generación de preguntas (Literal, Inferencial, Crítico).**
- **HU04: Clasificación de dificultad en preguntas.**

Entregables Técnicos

- **Backend: `aiController`: Integración con servicio de IA (OpenAI/Hugging Face). Lógica de *prompt engineering* para generar JSON estructurado de preguntas.**
- **Frontend: `Evaluations.jsx`: Vista para listar evaluaciones disponibles. Visualización básica de preguntas generadas.**

4.3.3. Sprint 3: Experiencia del Estudiante y Evaluación

Detalle	Asignación
Fecha de Inicio	07/10/2025
Fecha de Fin	27/10/2025
Duración	3 Semanas (21 Días)
Horas Estimadas	330 horas

Tabla 6: Sprint 3 Elaboración Propia

Objetivo: Permitir a los estudiantes realizar prácticas y recibir retroalimentación inmediata.

Historias de Usuario Abordadas

- **HU03:** Explicación de respuestas correctas/incorrectas.
- **HU05:** Puntuación y evaluación de progreso.

Entregables Técnicos

- **Backend:** `attemptController`: Registro de intentos, cálculo de puntajes. Lógica de comparación de respuestas.
- **Frontend:** `EvaluationDetail.jsx`: Interfaz para responder preguntas. `Results.jsx`: Vista de retroalimentación inmediata post-intento. `Progress.jsx`: Gráficas básicas de rendimiento estudiantil.

4.3.4. Sprint 4: Herramientas Docentes y Gestión de Grupos

Detalle	Asignación
Fecha de Inicio	**28/10/2025**
Fecha de Fin	**17/11/2025**
Duración	**3 Semanas (21 Días)**
Horas Estimadas	**330 horas**

Tabla 7: Sprint 4 Elaboración Propia

Objetivo: Empoderar a los docentes para gestionar clases y asignar tareas.

Historias de Usuario Abordadas

- **HU06 (Completa):** Asignación de textos a grupos/estudiantes.
- **HU07:** Panel de rendimiento docente.
- **HU09:** Comentarios y *feedback* personalizado.

Entregables Técnicos

- **Backend:** `teacherController`: Endpoints para `enrollStudent`, `recommendText`, `getStudentProgress`. Sistema de "Asignaciones" vs "Recomendaciones".
- **Frontend:** `teacher/Dashboard.jsx`: Vista general del docente. `StudentList.jsx` y `StudentDetail.jsx`: Gestión detallada de alumnos. `CreateAssignment.jsx`: Formulario para asignar lecturas con fecha límite.

4.3.5. Sprint 5: Análisis Avanzado y Reportes

Detalle	Asignación
Fecha de Inicio	**18/11/2025**
Fecha de Fin	**04/12/2025**
Duración	**2.5 Semanas (17 Días)**
Horas Estimadas	**280 horas**

Tabla 8: Sprint 5 Elaboración Propia

Objetivo: Profundizar en el análisis crítico (detección de falacias) y facilitar la exportación de datos.

Historias de Usuario Abordadas

- **HU08:** Exportación de notas a Excel/PDF.
- **HU10:** Detección de falacias en textos.
- **HU11:** Explicación de sesgos encontrados.

Entregables Técnicos

- **Backend:** `exportController`: Generación de Excel con *exceljs*.
  `aiController`: Nuevo *endpoint* `detectFallacies`.
  `teacherController`: *Endpoint* de descarga de textos originales.
- **Frontend:** Botón de exportación en `StudentDetail.jsx`. Visualización de falacias (integración en `EvaluationDetail` o vista dedicada).
  `AttemptDetail.jsx`: Vista detallada para que el profesor revise intentos específicos.

4.4. Planificación de Sprints

La planificación se realiza bajo el marco Scrum, asegurando una cadencia de entrega de valor al cliente (Universidad/Docentes).

4.4.1. Historias de usuario

Las Historias de Usuario son descripciones de una funcionalidad del sistema desde la perspectiva del usuario.

ID	Historia de Usuario	Módulo Principal	Actor Principal

HU01	Como estudiante, quiero poder **subir textos** (PDF, DOCX, TXT) a la plataforma para que el sistema los analice.	Gestión de Contenidos	Estudiante
HU02	Como estudiante, quiero que la IA **genere automáticamente preguntas** de lectura (literal, inferencial y crítica) basadas en el texto subido, para practicar mi comprensión.	Inteligencia Artificial	Estudiante
HU03	Como estudiante, quiero **recibir una explicación detallada** de por qué mi respuesta fue correcta o incorrecta, para entender mis errores.	Evaluación y Feedback	Estudiante
HU04	Como docente, quiero que las preguntas generadas automáticamente sean **clasificadas por su nivel de dificultad**, para poder asignar material adecuado a cada grupo.	Inteligencia Artificial	Docente
HU05	Como estudiante, quiero **ver mi puntuación y el progreso** de mis evaluaciones de lectura, para monitorear mi rendimiento.	Seguimiento de Progreso	Estudiante
HU06	Como docente, quiero **asignar textos específicos** a un grupo o a estudiantes individuales, para gestionar las tareas de lectura.	Gestión de Docentes	Docente
HU07	Como docente, quiero un **panel de rendimiento** que muestre estadísticas clave (notas promedio, tiempo	Seguimiento de Progreso	Docente

	dedicado) de mis estudiantes, para hacer seguimiento.		
HU08	Como docente, quiero poder **exportar las notas** de mis estudiantes en formato Excel o PDF, para integrarlas en mis registros académicos.	Reportes	Docente
HU09	Como docente, quiero poder **añadir comentarios o *feedback*** personalizado a las evaluaciones de mis estudiantes, para una tutoría individualizada.	Evaluación y Feedback	Docente
HU10	Como estudiante/docente, quiero que la IA sea capaz de **detectar falacias lógicas o sesgos** en el texto subido, para desarrollar mi pensamiento crítico.	Análisis Avanzado (IA)	Ambos
HU11	Como estudiante/docente, quiero **obtener una explicación** clara sobre los sesgos o falacias encontrados, para comprender el razonamiento detrás de la detección.	Análisis Avanzado (IA)	Ambos

Tabla 9: Historias de Usuario

4.4.2. Priorización de historias de usuario

La priorización se realiza por el *Product Owner* y el equipo, considerando el valor de negocio (impacto en la comprensión lectora) y la dependencia técnica. El *Product Backlog* se ordena por prioridad para alimentar el *Sprint Backlog* de cada iteración.

ID	Historia de Usuario	Módulo Principal	Actor Principal	Prioridad
HU01	Como estudiante, quiero poder **subir textos** (PDF, DOCX, TXT) a la plataforma para que el sistema los analice.	Gestión de Contenidos	Estudiante	**ALTA**
HU02	Como estudiante, quiero que la IA **genere automáticamente preguntas** de lectura (literal, inferencial y crítica) basadas en el texto subido.	Inteligencia Artificial	Estudiante	**ALTA**
HU03	Como estudiante, quiero **recibir una explicación detallada** de por qué mi respuesta fue correcta o incorrecta, para entender mis errores.	Evaluación y Feedback	Estudiante	**ALTA**
HU04	Como docente, quiero que las preguntas generadas automáticamente sean **clasificadas por su nivel de dificultad**, para poder asignar material adecuado a cada grupo.	Inteligencia Artificial	Docente	**ALTA**
HU05	Como estudiante, quiero **ver mi puntuación y el progreso** de mis evaluaciones de lectura,	Seguimiento de Progreso	Estudiante	**ALTA**

	para monitorear mi rendimiento.			

Tabla 10: Priorizacion Historias de Usuario

4.5. Cronograma de Actividades

El cronograma se alinea con las fechas de inicio y fin del proyecto (28/08/2025 al 04/12/2025). La gestión de las tareas y el control del progreso se realiza mediante la herramienta **Jira** , la cual permite seguir las actividades diarias (*Daily Scrum*) y la revisión de avances al final de cada ciclo (*Sprint Review*).

Estado	**Tarea**	**Tiempo (Horas)**	**Inicio**	**Fin**
Finalizado	**I. PLANIFICACIÓN E INICIO DEL PROYECTO**	**40**	**28/08/2025**	**01/09/2025**
Finalizado	**II. SPRINT 1: Fundamentos y Gestión de Contenidos (HU01, HU06 Parcial)**	**280**	**02/09/2025**	**15/09/2025**
Finalizado	II.1. Configuración de Arquitectura MERN (*Walking Skeleton*)	80	02/09/2025	06/09/2025
Finalizado	II.2. Desarrollo de Autenticación de Usuarios (Login/Registro con JWT)	70	07/09/2025	10/09/2025

Finalizado	II.3. Desarrollo de Componente de Subida de Archivos (HU01)	90	11/09/2025	14/09/2025
Finalizado	II.4. Revisión del Sprint y Despliegue Inicial (*Review 1*)	40	15/09/2025	15/09/2025
Finalizado	**III. SPRINT 2: IA y Generación de Evaluaciones (HU02, HU04)**	**330**	**16/09/2025**	**06/10/2025**
Finalizado	III.1. Configuración de API de IA y Prueba de Concepto (PoC)	90	16/09/2025	22/09/2025
Finalizado	III.2. Desarrollo de Lógica de Generación de Preguntas (HU02)	120	23/09/2025	29/09/2025
Finalizado	III.3. Implementación de Clasificación de Dificultad (HU04)	70	30/09/2025	03/10/2025
Finalizado	III.4. Revisión y Despliegue (*Review 2*)	50	04/10/2025	06/10/2025
Finalizado	**IV. SPRINT 3: Experiencia del Estudiante y Evaluación (HU03, HU05)**	**330**	**07/10/2025**	**27/10/2025**
Finalizado	IV.1. Desarrollo de Interfaz de Evaluación (*EvaluationDetail.jsx*)	80	07/10/2025	13/10/2025

Finalizado	IV.2. Implementación de Registro de Intentos y Puntuación (HU05)	100	14/10/2025	19/10/2025
Finalizado	IV.3. Implementación de Lógica de Explicación y Feedback (HU03)	100	20/10/2025	25/10/2025
Finalizado	IV.4. Revisión y Despliegue (*Review 3*)	50	26/10/2025	27/10/2025
Finalizado	**V. SPRINT 4: Herramientas Docentes y Gestión (HU06 Comp., HU07, HU09)**	**330**	**28/10/2025**	**17/11/2025**
Finalizado	V.1. Desarrollo de Paneles Docentes (Dashboard, Vistas de Alumnos) (HU07)	100	28/10/2025	03/11/2025
Finalizado	V.2. Desarrollo de Gestión de Asignaciones y Grupos (HU06)	100	04/11/2025	09/11/2025
Finalizado	V.3. Implementación de Feedback Personalizado del Docente (HU09)	80	10/11/2025	15/11/2025
Finalizado	V.4. Revisión y Despliegue (*Review 4*)	50	16/11/2025	17/11/2025

Finalizado	**VI. SPRINT 5: Análisis Avanzado y Reportes (HU08, HU10, HU11)**	**280**	**18/11/2025**	**04/12/2025**
Pendiente	VI.1. Implementación de Detección de Falacias/Sesgos (HU10, HU11)	90	18/11/2025	24/11/2025
Pendiente	VI.2. Desarrollo de Módulo de Exportación de Notas (HU08)	90	25/11/2025	30/11/2025
Pendiente	VI.3. Pruebas Finales Integrales y Corrección de Bugs	60	01/12/2025	03/12/2025
Pendiente	VI.4. Entrega Final del Proyecto	40	04/12/2025	04/12/2025

Tabla 11: Cronograma de Actividades

4.6. Gestión de Riesgos

Tipo de Riesgo	Detalle del Riesgo	Estrategia de Mitigación
Técnico	Fallo en la integración de APIs de IA (Hugging Face/OpenAI)	Múltiples alternativas de APIs probadas en la fase de prototipado.
Operacional	Resistencia inicial de los usuarios (estudiantes/docentes)	Pruebas piloto tempranas y sesiones de

		retroalimentación con usuarios reales
Gestión/Calidad	Problemas en el control de versiones y despliegue.	Uso de **Docker** y **CI/CD (GitHub Actions)** para automatizar y estandarizar el despliegue
Calidad	Baja calidad del código.	Establecer una métrica de calidad que exija una **Cobertura de Pruebas** mínima del **70%** del código.

Tabla 12: Gestión de Riesgos

CAPÍTULO 5
DISEÑO DEL SISTEMA DE INFORMACIÓN

.1. Diseño de Diagramas UML

.1.1. Diagramas de casos de uso

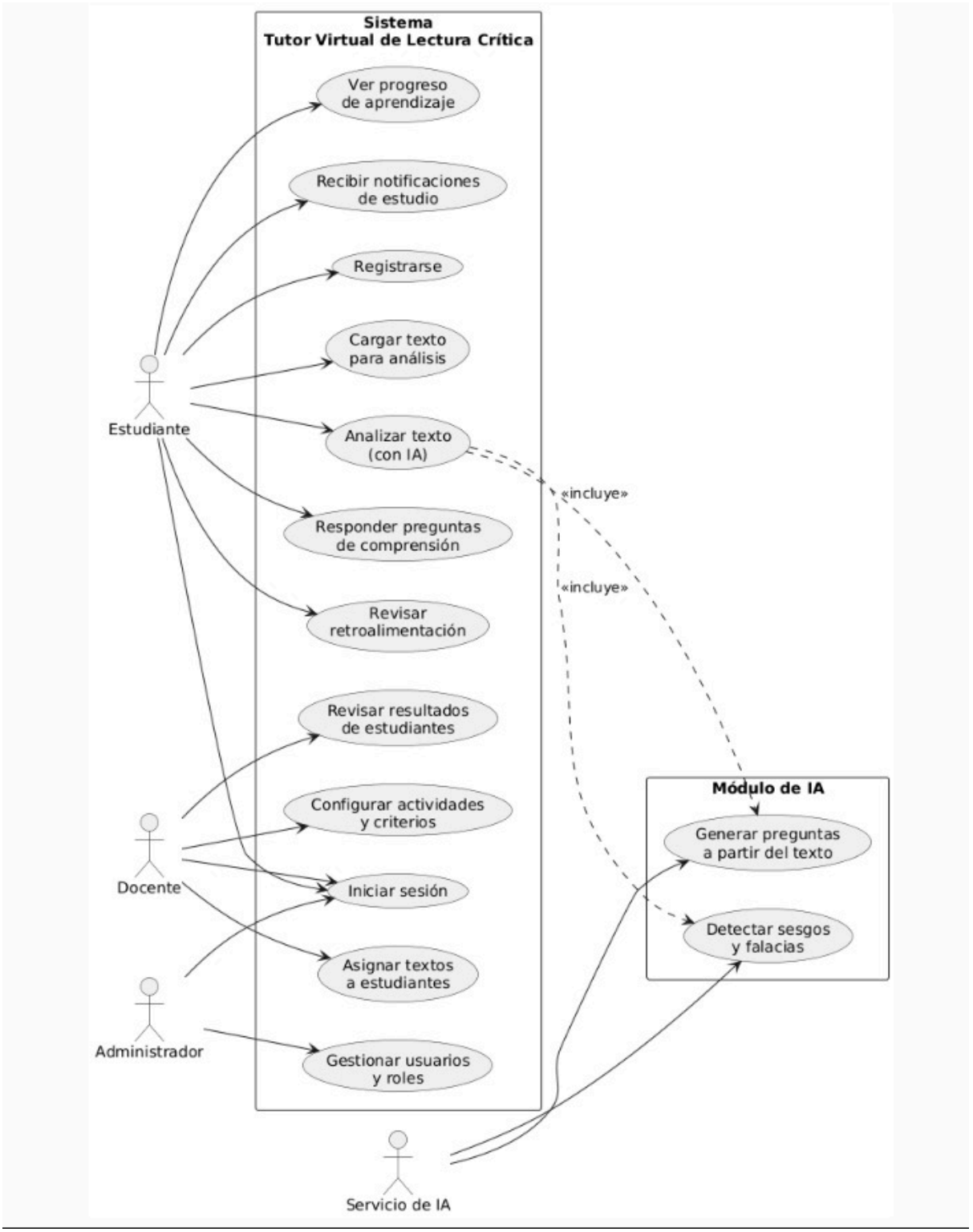


Imagen 1: Diagrama de Casos de Uso Elaboración Propia

El sistema “Tutor Virtual de Lectura Crítica” contempla los siguientes actores:

- **Estudiante:** carga textos, responde preguntas, revisa retroalimentación.
- **Docente:** asigna textos, revisa resultados de los estudiantes.
- **Sistema de IA:** genera preguntas y detecta falacias.
- **Administrador:** supervisa usuarios y operaciones internas.

Casos de Uso Principales:

- Cargar texto para análisis.
- Generar preguntas automáticas.
- Detectar sesgos y falacias.
- Realizar autoevaluación.
- Consultar progreso estudiantil.
- Gestionar usuarios.

.1.2. Diagramas de secuencia

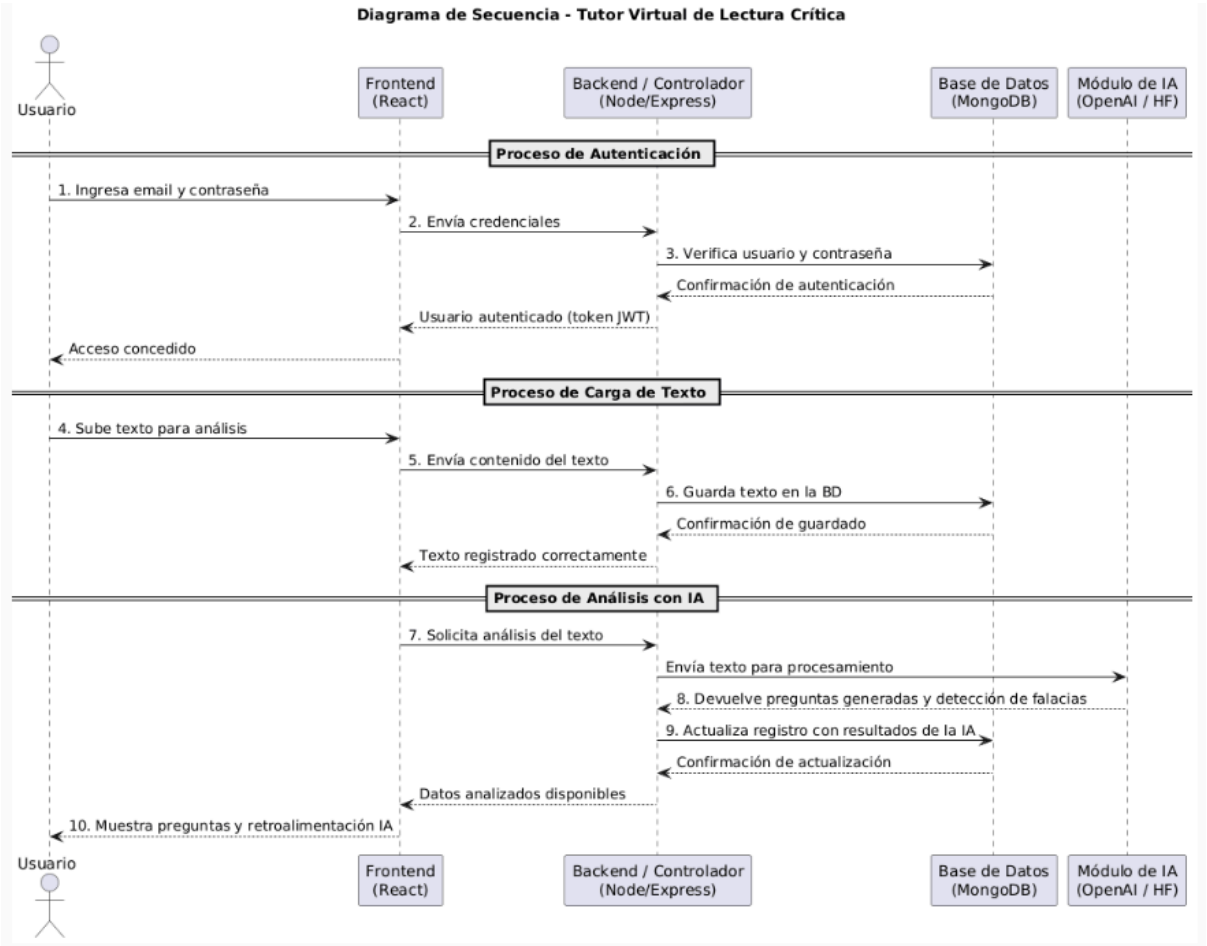


Imagen 2: Diagrama de Secuencia Elaboración Propia

- El usuario carga un texto.
- El frontend envía el texto al backend.
- El backend llama a la API de IA (Hugging Face/OpenAI).
- La IA responde con un conjunto de preguntas.
- El backend las almacena en MongoDB.
- El frontend las muestra al usuario.

.1.3. Diagramas de colaboración

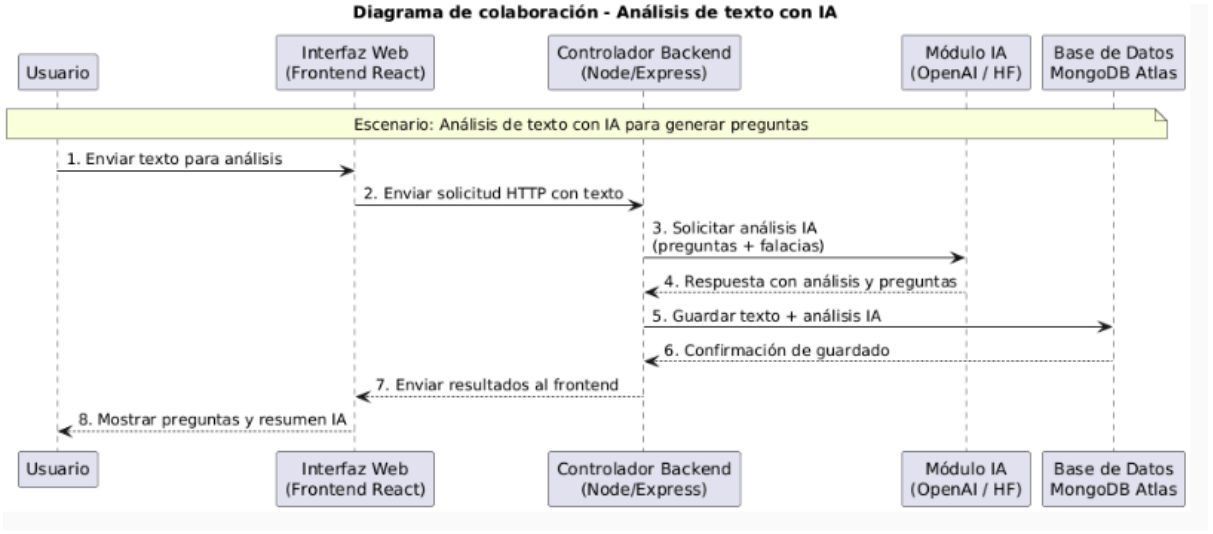


Imagen 4: Diagrama de Colaboración Elaboración Propia

En este diagrama se representa cómo interactúan los componentes del sistema MERN:

- Frontend (React): Interfaces y envío de solicitudes.
- Backend (Node/Express): Procesa peticiones, maneja lógica.
- Base de datos (MongoDB): Guarda textos, preguntas, usuarios.
- Servicios externos: IA, n8n para automatización.

.1.4. Diagramas de clases

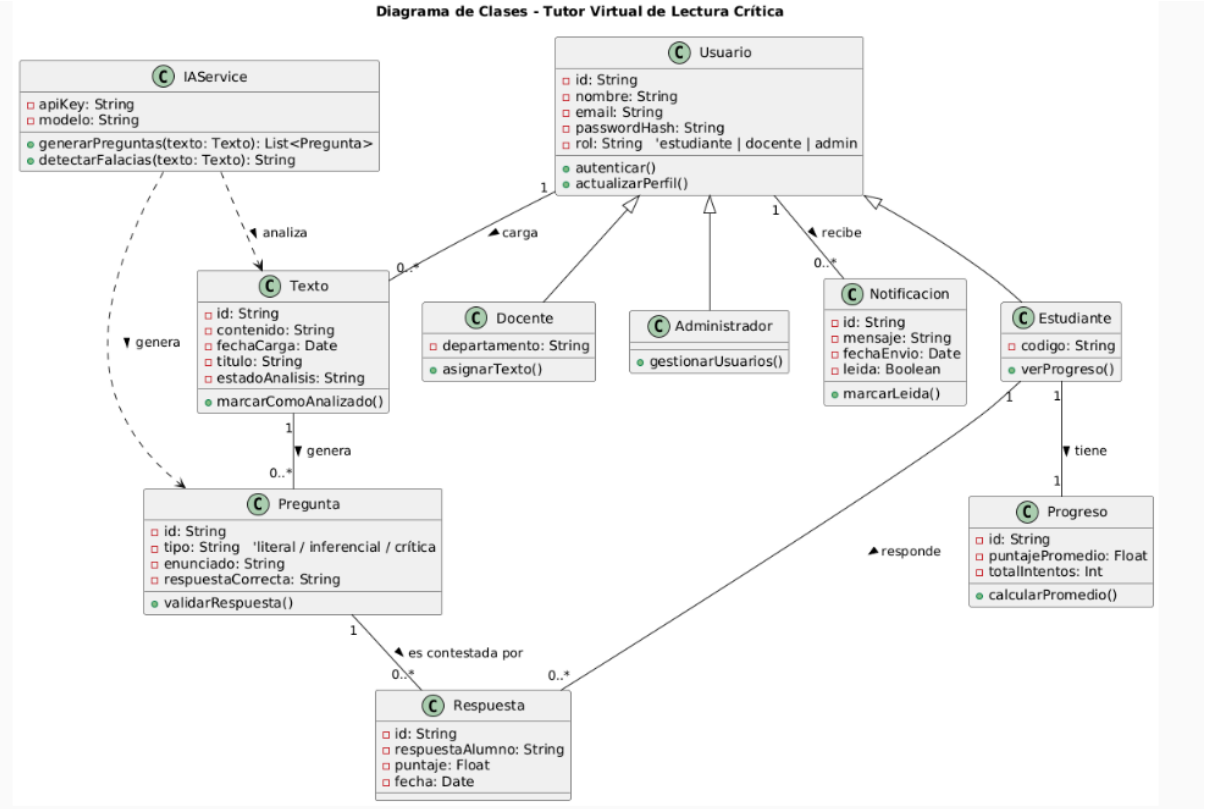


Imagen 5: Diagrama de Clases Elaboración Propia

.2. Diseño de Base de Datos

.2.1. Diseño conceptual (E/R)

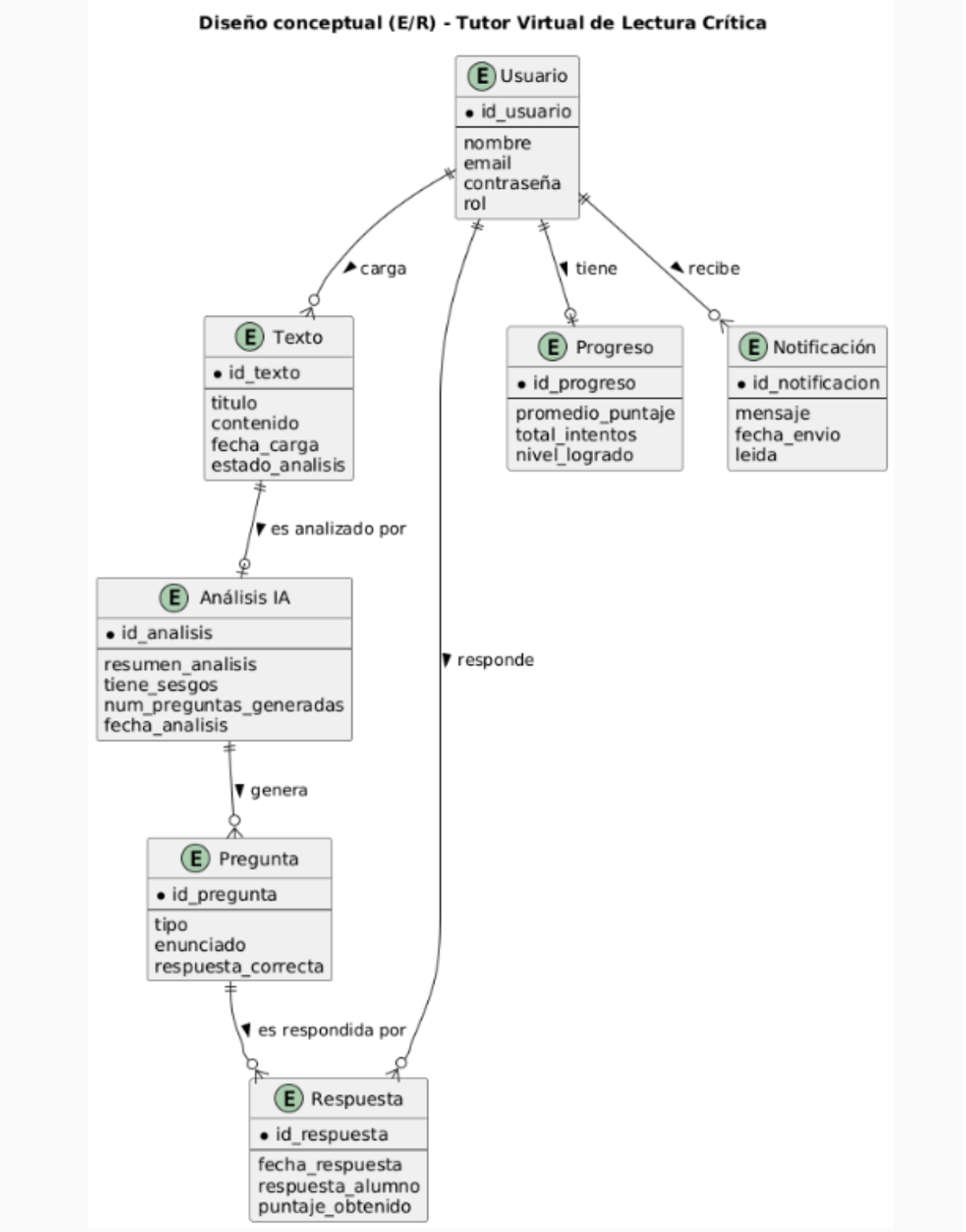


Imagen 6: Diseño conceptual E/R Elaboración Propia

Entidades:

- Usuario
- Texto
- Pregunta
- Respuesta
- Progreso

Relaciones:

- Un usuario carga muchos textos.
- Un texto genera muchas preguntas.
- Un usuario responde muchas preguntas.
- Un usuario tiene un progreso acumulado.

.2.2. Diseño lógico

El diseño lógico del sistema se ha planteado para un modelo documental sobre **MongoDB Atlas**, manteniendo una organización clara de las colecciones y las referencias entre ellas. A continuación, se describen las principales colecciones del sistema.

Tabla users

```
{
  _id: ObjectId,            // PK
  name:        String,
  email:       String,      // único
  passwordHash:String,
  role:        String,      // "estudiante" | "docente" | "admin"
  avatar:      String,      // URL opcional
  settings: {
    notifications: Boolean, // activar/desactivar recordatorios
    theme:       String     // "light" | "dark"
  },
  preferences: {
    language: String,       // "es" | "en"
    timezone: String
  },
  lastLogin:   Date,
  createdAt:   Date,
  updatedAt:   Date,
  deletedAt:   Date|null
}
```

Tabla texts

```
{
  _id: ObjectId,              // PK
  title:        String,
  content:      String,       // texto plano extraído de PDF/DOCX/TXT
  originalFile: {
    name:       String,
    url:        String,       // ruta en almacenamiento
    sizeMB:     Number,
    mimeType:   String        // "application/pdf", etc.
  },
  language:     String,       // idioma detectado
  uploadedBy:   ObjectId,     // referencia a users
  contextTags: [String],      // etiquetas temáticas opcionales
  statusAnalysis:String,      // "pendiente" | "analizado" | "error"
  createdAt:    Date,
  updatedAt:    Date
}
```

Tabla analyses

```
{
  _id: ObjectId,              // PK
  textId:       ObjectId,     // referencia a texts
  engine:       String,       // modelo usado: "gpt-4.x", "gemini", etc.
  summary:      String,       // resumen breve del texto
  difficulty:   String,       // "básico" | "intermedio" | "avanzado"
  hasBiases:    Boolean,
  fallacies: [
    {
      type:       String,     // nombre de la falacia/sesgo
      description: String,
      fragment:   String      // parte del texto donde aparece
    }
  ],
  numQuestions:  Number,
  notes:        String,       // comentarios adicionales de la IA
  createdAt:    Date,
  updatedAt:    Date
}
```

Tabla questions

```
{
  _id: ObjectId,              // PK
  textId:     ObjectId,       // referencia a texts
  analysisId: ObjectId,       // referencia a analyses
  type:       String,         // "literal" | "inferencial" | "crítica"
```

```
  stem:       String,           // enunciado de la pregunta
  options: [
    {
      key:   String,           // "A","B","C","D"
      label: String            // texto de la opción
    }
  ],
  correctOption: String,        // key de la opción correcta
  explanationCorrect:   String,  // por qué es correcta
  explanationIncorrect: String,  // feedback genérico si falla
  difficulty:          String,  // "fácil" | "media" | "difícil"
  createdBy: ObjectId,          // docente o sistema (IA)
  createdAt: Date,
  updatedAt: Date,
  deletedAt: Date|null
}
```

Tabla evaluations

```
{
  _id: ObjectId,               // PK
  textId:       ObjectId,      // referencia a texts
  name:         String,        // nombre de la evaluación
  description:  String,
  questionIds:  [ObjectId],    // referencias a questions
  createdBy:    ObjectId,      // docente
  isPublic:     Boolean,       // visible para todos los estudiantes
  attemptsLimit:Number,        // número máx. de intentos
  dueDate:      Date|null,     // fecha límite opcional
  createdAt:    Date,
  updatedAt:    Date
}
```

Representa una “evaluación” o conjunto de preguntas asociadas a un texto (lo que luego el estudiante resolverá).

Tabla attempts

```
{
  _id: ObjectId,               // PK
  evaluationId: ObjectId,      // referencia a evaluations
  studentId:    ObjectId,      // referencia a users
  answers: [
    {
      questionId:   ObjectId,  // referencia a questions
```

```
    selectedOption:String,    // opción elegida
    isCorrect:   Boolean,
    score:       Number,      // puntaje asignado a esa pregunta
    feedback:    String       // explicación mostrada al estudiante
   }
 ],
 totalScore:  Number,          // puntaje total obtenido
 maxScore:    Number,          // puntaje máximo posible
 startedAt:   Date,
 finishedAt:  Date,
 timeSpentSec: Number,
 createdAt:   Date
}
```

Cada intento que hace un estudiante sobre una evaluación.

Tabla progress

```
{
 _id: ObjectId,               // PK
 studentId:     ObjectId,     // referencia a users
 avgScoreGlobal: Number,      // promedio general
 avgScoreLiteral:Number,
 avgScoreInferential:Number,
 avgScoreCritical:Number,
 attemptsCount:  Number,
 lastEvaluationId:ObjectId|null,
 lastEvaluationAt:Date|null,
 level:         String,       // "Inicial" | "En desarrollo" | "Logro esperado"
 createdAt:     Date,
 updatedAt:     Date
}
```

Registro agregado del progreso del estudiante en comprensión lectora.

Tabla assignments

```
{
 _id: ObjectId,               // PK
 teacherId:   ObjectId,       // referencia a users
 evaluationId: ObjectId,      // referencia a evaluations
 textId:      ObjectId,       // referencia a texts
 studentIds:  [ObjectId],     // lista de estudiantes asignados
 groupName:   String|null,    // nombre del grupo (si aplica)
 dueDate:     Date,
 status:      String,         // "activa" | "cerrada"
 createdAt:   Date,
 updatedAt:   Date
```

}

Asignaciones que hace el docente a uno o varios estudiantes sobre un texto/evaluación.

Tabla notifications

{
_id: ObjectId, // PK
userId: ObjectId, // referencia a users
type: String, // "recordatorio" | "resultado" | "sistema"
title: String,
message: String,
entityType:String, // "Text" | "Evaluation" | "Attempt"
entityId: ObjectId, // id de la entidad relacionada
read: Boolean,
createdAt: Date,
readAt: Date|null
}

Tabla logs

{
_id: ObjectId,
userId: ObjectId|null, // usuario que generó la acción
action: String, // "LOGIN", "UPLOAD_TEXT", "START_EVALUATION", etc.
metadata: Object, // datos adicionales (JSON)
createdAt: Date
}

Tabla 13: Tablas de diseño lógico Elaboración Propia

.2.3. Diseño físico

El modelo físico se implementará en **MongoDB Atlas**, donde se definen las colecciones, sus atributos principales, tipos de dato e índices para optimizar el acceso a la información. A continuación, se describen las colecciones que conforman el sistema *AppComprende-IA*.

users
{ _id: ObjectId, // PK

```
  name:         String,
  email:        String,        // índice único
  passwordHash: String,
  role:         String,        // "estudiante" | "docente" | "admin"
  avatar:       String,        // URL opcional
  settings: {                  // documento embebido
    notifications: Boolean,
    theme:         String      // "light" | "dark"
  },
  preferences: {               // documento embebido
    language: String,          // "es", "en"
    timezone: String
  },
  lastLogin:    ISODate,
  createdAt:    ISODate,
  updatedAt:    ISODate
}

// Índices recomendados:
//  db.users.createIndex({ email: 1 }, { unique: true })
//  db.users.createIndex({ role: 1 })
```

texts

```
{
  _id:          ObjectId,      // PK
  title:        String,
  content:      String,        // texto plano del archivo
  originalFile: {              // documento embebido
    name:     String,
    url:      String,          // ruta en almacenamiento
    sizeMB:   Number,
    mimeType: String           // "application/pdf", etc.
  },
  language:     String,
  uploadedBy:   ObjectId,      // ref -> users._id
  contextTags:  [String],      // etiquetas temáticas
  statusAnalysis:String,       // "pendiente" | "analizado" | "error"
  createdAt:    ISODate,
  updatedAt:    ISODate
}

// Índices recomendados:
//  db.texts.createIndex({ uploadedBy: 1, createdAt: -1 })
//  db.texts.createIndex({ statusAnalysis: 1 })
```

analyses

```
analyses
{
  _id:            ObjectId,   // PK
  textId:         ObjectId,   // ref -> texts._id
  engine:           String,     // modelo IA usado
  summary:          String,     // resumen del texto
  difficulty:      String,     // "básico" | "intermedio" | "avanzado"
  hasBiases:       Boolean,
  fallacies: [ {              // lista embebida
    type:        String,
    description: String,
    fragment:    String
  } ],
  numQuestions:    Number,
  notes:           String,
  createdAt:       ISODate,
  updatedAt:       ISODate
}

// Índices recomendados:
//  db.analyses.createIndex({ textId: 1 })
//  db.analyses.createIndex({ hasBiases: 1 })
```

questions

```
questions
{
  _id:            ObjectId,  // PK
  textId:         ObjectId,  // ref -> texts._id
  analysisId:       ObjectId,  // ref -> analyses._id
  type:            String,    // "literal" | "inferencial" | "crítica"
  stem:             String,    // enunciado
  options: [ {               // opciones embebidas
    key:   String,            // "A","B","C","D"
    label: String
  } ],
  correctOption:    String,    // key correcta
  explanationCorrect:   String,
  explanationIncorrect: String,
  difficulty:       String,   // "fácil" | "media" | "difícil"
  createdBy:         ObjectId, // ref -> users (docente o sistema)
  createdAt:        ISODate,
  updatedAt:         ISODate
}

// Índices recomendados:
//  db.questions.createIndex({ textId: 1 })
//  db.questions.createIndex({ analysisId: 1 })
//  db.questions.createIndex({ type: 1, difficulty: 1 })
```

evaluations

```
{
  _id:          ObjectId,     // PK
  textId:       ObjectId,     // ref -> texts._id
  name:         String,
  description:  String,
  questionIds:  [ObjectId],   // refs -> questions._id
  createdBy:    ObjectId,     // ref -> users (docente)
  isPublic:     Boolean,
  attemptsLimit: Number,      // intentos máximos
  dueDate:      ISODate|null, // fecha límite
  createdAt:    ISODate,
  updatedAt:    ISODate
}

// Índices recomendados:
//  db.evaluations.createIndex({ textId: 1 })
//  db.evaluations.createIndex({ createdBy: 1 })
//  db.evaluations.createIndex({ dueDate: 1 })
```

attempts

```
{
  _id:          ObjectId,     // PK
  evaluationId: ObjectId,     // ref -> evaluations._id
  studentId:    ObjectId,     // ref -> users._id
  answers: [ {
    questionId:     ObjectId,  // ref -> questions._id
    selectedOption: String,
    isCorrect:      Boolean,
    score:          Number,
    feedback:       String
  } ],
  totalScore:   Number,
  maxScore:     Number,
  startedAt:    ISODate,
  finishedAt:   ISODate,
  timeSpentSec: Number,
  createdAt:    ISODate
}

// Índices recomendados:
//  db.attempts.createIndex({ studentId: 1, evaluationId: 1 })
//  db.attempts.createIndex({ evaluationId: 1 })
```

progress

```
{
  _id:                ObjectId,   // PK
  studentId:          ObjectId,   // ref -> users._id
  avgScoreGlobal:     Number,
  avgScoreLiteral:    Number,
  avgScoreInferential:Number,
  avgScoreCritical:   Number,
  attemptsCount:      Number,
  lastEvaluationId:   ObjectId|null, // ref -> evaluations._id
  lastEvaluationAt:   ISODate|null,
  level:              String,    // "Inicial" | "En desarrollo" | "Logro esperado"
  createdAt:          ISODate,
  updatedAt:          ISODate
}

// Índices recomendados:
//  db.progress.createIndex({ studentId: 1 }, { unique: true })
```

assignments

```
{
  _id:         ObjectId,      // PK
  teacherId:   ObjectId,      // ref -> users._id
  evaluationId:ObjectId,      // ref -> evaluations._id
  textId:      ObjectId,      // ref -> texts._id
  studentIds:  [ObjectId],    // lista de estudiantes
  groupName:   String|null,
  dueDate:     ISODate,
  status:      String,        // "activa" | "cerrada"
  createdAt:   ISODate,
  updatedAt:   ISODate
}

// Índices recomendados:
//  db.assignments.createIndex({ teacherId: 1 })
//  db.assignments.createIndex({ dueDate: 1 })
//  db.assignments.createIndex({ evaluationId: 1 })
```

notifications

```
{
  _id:        ObjectId,       // PK
  userId:     ObjectId,       // ref -> users._id
  type:       String,         // "recordatorio" | "resultado" | "sistema"
  title:      String,
  message:    String,
  entityType: String,         // "Text" | "Evaluation" | "Attempt"
```

```
  entityId:   ObjectId,       // ref a la entidad relacionada
  read:       Boolean,
  createdAt:  ISODate,
  readAt:     ISODate|null
}

// Índices recomendados:
//  db.notifications.createIndex({ userId: 1, read: 1 })
//  db.notifications.createIndex({ entityType: 1, entityId: 1 })
```

logs (activityLogs)

```
{
  _id:       ObjectId,        // PK
  userId:    ObjectId|null,   // ref -> users._id
  action:    String,          // "LOGIN","UPLOAD_TEXT","START_EVALUATION",
etc.
  metadata:  Object,          // JSON con detalles
  createdAt: ISODate
}

// Índices recomendados:
//  db.logs.createIndex({ userId: 1, createdAt: -1 })
//  db.logs.createIndex({ action: 1, createdAt: -1 })
```

.2.4. Modelado de base de datos

El modelado de base de datos integra la estructura lógica del sistema con el enfoque documental de **MongoDB Atlas**, permitiendo una representación flexible, escalable y orientada a los datos textuales y evaluativos del proyecto *AppComprende-IA*. El modelo se deriva directamente del diseño conceptual (E/R) y se ajusta a los requerimientos funcionales del sistema, en especial la gestión de textos, análisis mediante IA, evaluaciones automáticas y seguimiento del rendimiento de los estudiantes.

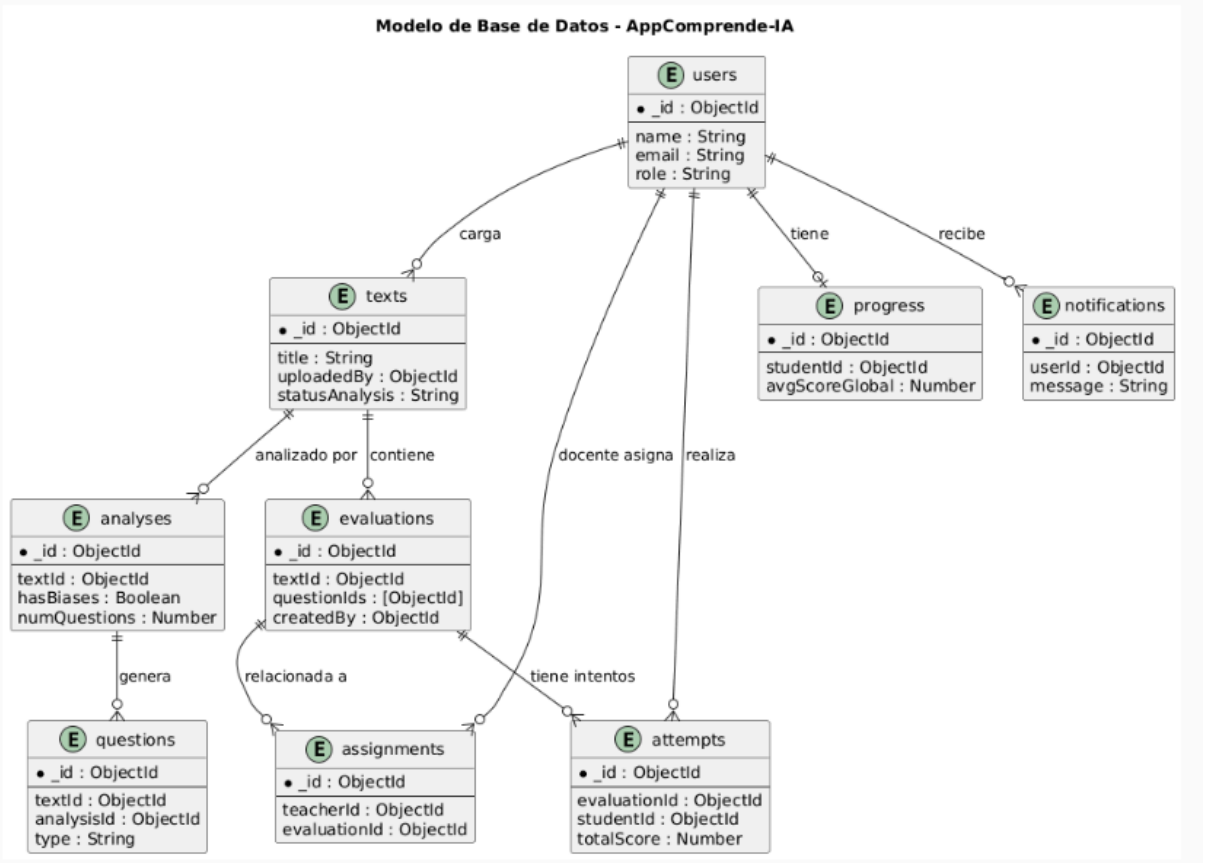


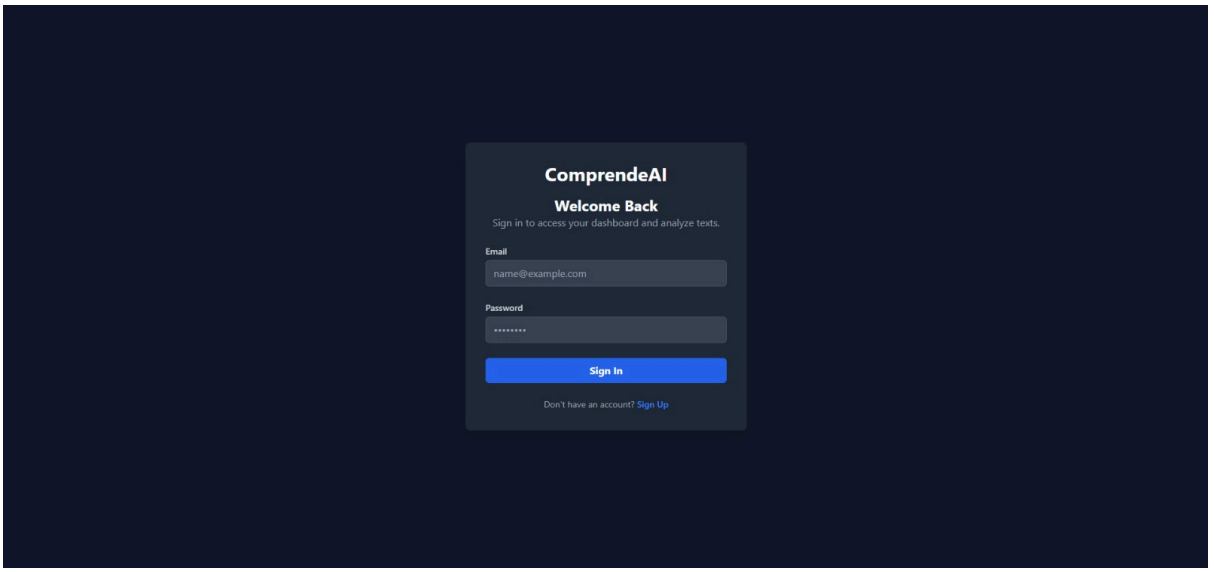
Imagen 7: Modelo de base de Datos Elaboración Propia

El modelo de datos permite:

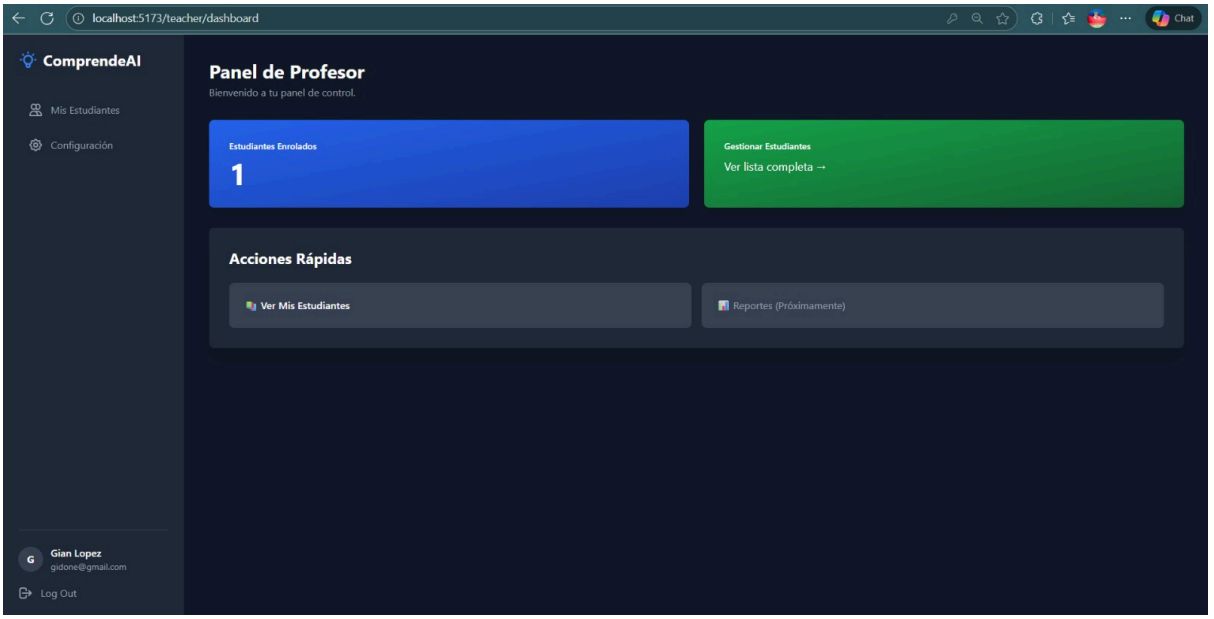
- Escalar horizontalmente gracias a la naturaleza documental de MongoDB.
- Embebido y referencias mixtas para maximizar rendimiento en lecturas frecuentes (preguntas, intentos).
- Adecuarse a microservicios futuros (IA, evaluaciones, docente, notificaciones).
- Registrar intentos, análisis automáticos y métricas de progreso de manera eficiente.
- Integrar la inteligencia artificial como un módulo almacenado y versionable.

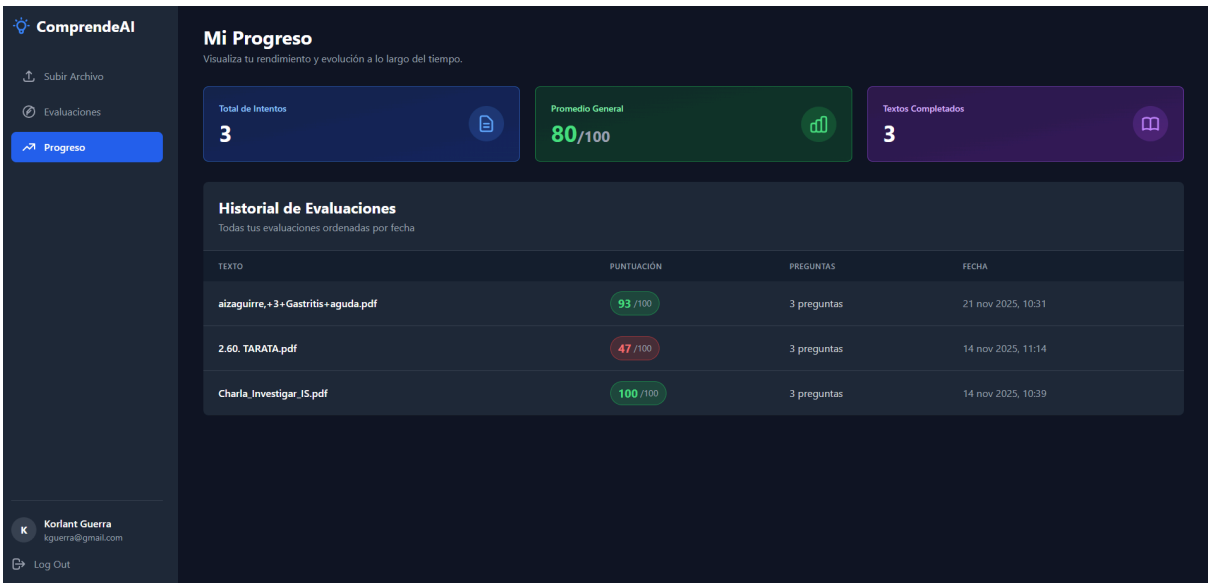
.3. Diseño de Interfaces Básicas

.3.1. Acceso login



.3.2. Interfaz …




ComprendeAI
Subir Archivo
Evaluaciones
Progreso
Korlant Guerra
kguerra@gmail.com
Log Out
Mi Progreso
Visualiza tu rendimiento y evolución a lo largo del tiempo.
Total de Intentos
3
Promedio General
80/100
Textos Completados
3
Historial de Evaluaciones
Todas tus evaluaciones ordenadas por fecha
TEXTO
PUNTUACIÓN
PREGUNTAS
FECHA
aizaguirre,+3+Gastritis+aguda.pdf
93 /100
3 preguntas
21 nov 2025, 10:31
2.60. TARATA.pdf
47 /100
3 preguntas
14 nov 2025, 11:14
Charla_Investigar_IS.pdf
100 /100
3 preguntas
14 nov 2025, 10:39

CAPÍTULO 6
CODIFICACIÓN DEL SOFTWARE

6.1. Desarrollo del Sprint 1

6.1.1. Sprint planning

Establecer la base del sistema MERN, implementar autenticación con JWT, gestionar usuarios y permitir la carga inicial de textos.

Épica asociada:

- Épica A – Tutor Virtual (inicio del flujo de aprendizaje)
- Épica B (parcial) – Estructura de base de datos para docentes/estudiantes

6.1.2. Sprint backlog

Historias de Usuario abordadas:

HU01: Subida de textos PDF/DOCX/TXT (≤10 MB).

HU06 (parcial): Estructura de BD para usuarios y grupos académicos.

- Configuración de servidor Express con rutas base.
- Conexión a MongoDB Atlas.
- Implementación del módulo de autenticación (authController).
- Implementación del módulo de carga de textos (textController).
- Lectura de PDF (pdf-parse) y DOCX (mammoth).
- Inicialización del frontend (React, Vite, Tailwind).
- Creación de las vistas Login, Register y Upload.jsx.

6.1.3. Historias de usuarios

HU01: Como estudiante, quiero subir un texto en formato PDF, DOCX o TXT para que el sistema lo analice.

HU06: Como docente, quiero gestionar la base de datos inicial de mis estudiantes y grupos, para que el sistema pueda asignar actividades más adelante.

6.1.4. Taskboard

Tareas	To Do	In Progress	Done
Configuración del servidor Express			**X**
Conexión a MongoDB Atlas			**X**
Implementación de authController (Login/Registro)			**X**
Implementación de textController (Subida de archivo)		**X**	
Lectura de archivos PDF/DOCX (pdf-parse / mammoth)		**X**	
Configuración de React + Vite + Tailwind			**X**
Pantalla de Login		**X**	
Pantalla de Register			**X**
Componente Upload.jsx (carga de texto)			**X**

Tabla 14 : Sprint 01 Taskboard Elaboración Propia

6.1.5. Daily scrum

Impedimentos principales:

- Configuración de CORS entre frontend y backend.
- Ajuste de rutas absolutas/relativas en Vite.

6.1.6. Sprint review

Se presentaron:

- Sistema MERN operativo.

- Login/Registro funcional.
- Subida y lectura de textos correctamente implementadas.

6.1.7. Criterios de aceptación

- Se pueden asignar textos.
- Se visualiza progreso por estudiante.

6.1.8. Resultados del sprint

Backend

- authController (JWT + bcrypt).
- textController (upload + parsing).
- Modelos iniciales de usuario y texto.

Frontend

- Páginas Login/Register.
- Componente Upload.jsx con barra de progreso.

6.1.8.1. Evidencias.

HU01:

Imagen 8 : Fronted UH01 Elaboracion propia

HU06:

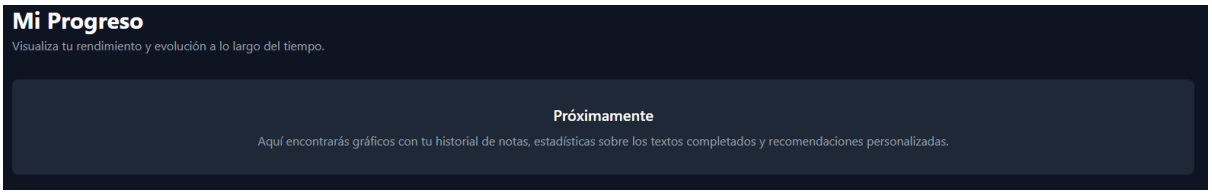


*Imagen 9 : Fronted UH02 Elaboracion propia*Prueba de desarrollo.

HU01 – Subida de textos (PDF, DOCX, TXT)

```
JS textController.js ×
backend > controllers > JS textController.js > ...

const textService = require('../services/textService.js');
const pdfParse = require('pdf-parse');
const mammoth = require('mammoth');

const uploadText = async (req, res) => {
    try {
        if (!req.file) return res.status(400).json({ message: 'No se ha subido ningún archivo.' });

        const { buffer, mimetype, originalname } = req.file;
        let content = '';

        if (mimetype === 'application/pdf') {
            const data = await pdfParse(buffer);
            content = data.text;
        } else if (mimetype === 'application/vnd.openxmlformats-officedocument.wordprocessingml.document') {
            const { value } = await mammoth.extractRawText({ buffer });
            content = value;
        } else if (mimetype === 'text/plain') {
            content = buffer.toString('utf-8');
        }

        if (!content.trim()) {
            return res.status(400).json({ message: 'El archivo está vacío o no se pudo leer el contenido.' });
        }

        const newText = await textService.createText(originalname, content, req.userId);
        res.status(201).json({ message: "Archivo procesado exitosamente.", text: newText });

    } catch (error) {
        // Si el servicio lanza el error de duplicado (409), lo pasamos al cliente.
        if (error.statusCode === 409) {
            return res.status(409).json({ message: error.message });
        }
        // Para cualquier otro error, respondemos con un error 500 genérico.
        console.error("Error en uploadText controller:", error);
        res.status(500).json({ message: "Error interno del servidor al procesar el archivo." });
    }
};

const createText = async (req, res) => {
    try {
        const { filename, content } = req.body;
        const newText = await textService.createText(filename, content, req.userId);
        res.status(201).json({ message: "Texto creado exitosamente", text: newText });
    } catch (error) {
        // Manejo específico para el error de duplicado.
        if (error.statusCode === 409) {
            return res.status(409).json({ message: error.message });
```

```
        if (error.message.startsWith('Faltan datos')) {
            return res.status(400).json({ message: error.message });
        }
        // Error genérico para otros casos.
        console.error("Error en createText controller:", error);
        res.status(500).json({ message: error.message });
    }
};

const getAllTexts = async (req, res) => {
    try {
        const userTexts = await textService.getAllTextsByOwner(req.userId);
        res.status(200).json({
            message: "Textos del usuario obtenidos exitosamente",
            texts: userTexts
        });
    } catch (error) {
        res.status(500).json({ message: "Error interno del servidor al obtener los textos." });
    }
};

const getTextById = async (req, res) => {
    try {
        const { id } = req.params;
        const userId = req.userId;
        const text = await textService.getTextByIdAndOwner(id, userId);
        if (!text) {
            return res.status(404).json({ message: 'Texto no encontrado o no tienes permiso para verlo.' });
        }
        res.status(200).json({
            message: "Texto obtenido exitosamente",
            text: text
        });
    } catch (error) {
        if (error.kind === 'ObjectId') {
            return res.status(400).json({ message: 'El ID del texto no es válido.' });
        }
        res.status(500).json({ message: "Error interno del servidor al obtener el texto." });
    }
};

module.exports = {
    uploadText,
    createText,
    getAllTexts,
    getTextById,
};
```

Imagen 10 : Subida del archivo y Validación de tamaño-lectura

Upload.jsx

frontend > src > pages > student > Upload.jsx > Upload

```jsx
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import DashboardLayout from '../../layouts/DashboardLayout';

const UploadIcon = () => <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor" stro
const EvaluationsIcon = () => <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor"
const ProgressIcon = () => <svg xmlns="http://www.w3.org/2000/svg" className="h-6 w-6" fill="none" viewBox="0 0 24 24" stroke="currentColor" st

const Upload = () => {
    const studentNavLinks = [
        { name: 'Subir Archivo', path: '/student/upload', icon: <UploadIcon /> },
        { name: 'Evaluaciones', path: '/student/evaluations', icon: <EvaluationsIcon /> },
        { name: 'Progreso', path: '/student/progress', icon: <ProgressIcon /> },
    ];

    const [file, setFile] = useState(null);
    const [message, setMessage] = useState('');
    const [error, setError] = useState('');
    const [isProcessing, setIsProcessing] = useState(false);
    const navigate = useNavigate();

    const handleFileChange = (e) => {
        const selectedFile = e.target.files[0];
        if (selectedFile) {
            setFile(selectedFile);
            setMessage(`Archivo seleccionado: ${selectedFile.name}`);
            setError('');
        }
    };

    const handleSubmit = async (e) => {
        e.preventDefault();
        if (!file) {
            setError('Por favor, selecciona un archivo para subir.');
            return;
        }

        setIsProcessing(true);
        setError('');
        setMessage('Subiendo y analizando el archivo... Esto puede tardar un momento, por favor, espera.');

        const formData = new FormData();
        formData.append('file', file);

        try {
            const token = localStorage.getItem('token');
            const response = await fetch('http://localhost:4000/textos/upload', {
                method: 'POST',
```

```jsx
            const data = await response.json();

            if (!response.ok) {
                // Creamos un error personalizado que también contenga el status code
                const customError = new Error(data.message || 'Ocurrió un error inesperado.');
                customError.status = response.status;
                throw customError;
            }

            setMessage('¡Análisis completado con éxito! Redirigiendo a los detalles de la evaluación...');

            setTimeout(() => {
                navigate(`/student/evaluations/${data.text._id}`);
            }, 2000);

        } catch (err) {
            // Ahora podemos diferenciar el error en el bloque catch
            if (err.status === 409) {
                setError(err.message + " Puedes revisar el análisis previo en la sección de Evaluaciones.");
            } else {
                setError(err.message || "No se pudo conectar con el servidor. Inténtalo de nuevo más tarde.");
            }
            setMessage('');
        } finally {
            setIsProcessing(false);
        }
    };

    return (
        <DashboardLayout navLinks={studentNavLinks}>
            <h1 className="text-3xl font-bold text-white mb-8">Subir Archivo para Análisis</h1>
            <div className="bg-gray-800 p-8 rounded-lg shadow-lg max-w-2xl mx-auto">
                <form onSubmit={handleSubmit}>
                    <div className="mb-6">
                        <label htmlFor="file-upload" className="block text-lg font-medium text-gray-300 mb-2">Selecciona un archivo</label>
                        <p className="text-sm text-gray-500 mb-4">Formatos soportados: .pdf, .docx, .txt</p>
                        <input
                            id="file-upload"
                            name="file-upload"
                            type="file"
                            onChange={handleFileChange}
                            className="block w-full text-sm text-gray-400 file:mr-4 file:py-2 file:px-4 file:rounded-full file:border-0 file:text-sm file:font-semibold file:bg-blue-60
                            accept=".pdf,.docx,.txt"
                            disabled={isProcessing}
                        />
                    </div>
```

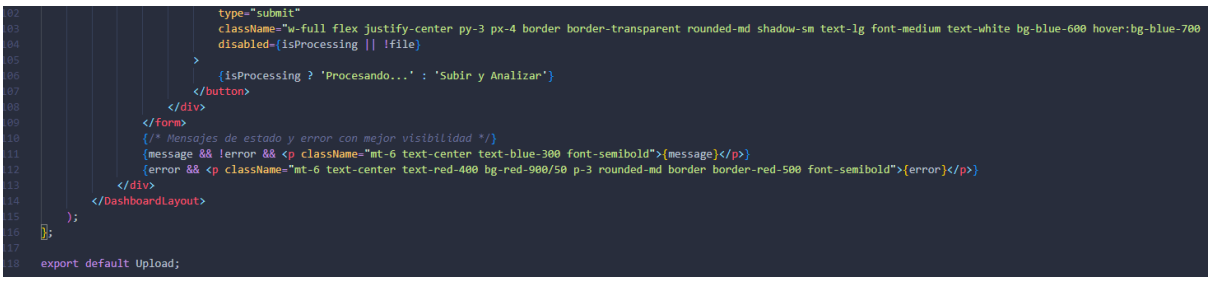


Imagen 11 : Selección del archivo-Vista previa del nombre-Estado-Notificación

HU06 – Base de datos para usuarios (rol estudiante/docente)

```
const register = async (req, res) => {
    try {
        // 1. Llama al servicio para ejecutar la lógica de negocio.
        const registeredUser = await authService.registerUser(req.body);

        // 2. Si todo va bien, envía una respuesta exitosa.
        res.status(201).json({
            message: "Usuario registrado exitosamente",
            usuario: registeredUser
        });

    } catch (error) {
        // 3. Si el servicio lanza un error, lo captura y envía una respuesta de error apropiada.
        // Usamos un switch para un manejo de errores más limpio y escalable.
        switch (error.message) {
            case "El email ya está registrado":
                res.status(409).json({ message: error.message });
                break;
            case "Faltan datos: nombre, email o password son requeridos":
                res.status(400).json({ message: error.message });
                break;
            default:
                res.status(500).json({ message: "Error interno del servidor al registrar." });
                break;
        }
    }
};

/**
 * Controlador para manejar el login de un usuario.
 */
const login = async (req, res) => {
    try {
        // 1. Llama al servicio para la autenticación.
        const loginData = await authService.loginUser(req.body);

        // 2. Envía la respuesta con el token y los datos del usuario.
        res.status(200).json({
            message: "Usuario logueado exitosamente",
            usuario: loginData
        });

    } catch (error) {
        // 3. Para el login, por seguridad, siempre devolvemos un error genérico 401.
        res.status(401).json({ message: "Credenciales incorrectas" });
    }
};
```

Imagen 12 : Registro correcto-Rol asignado-Validación de email duplicado

6.1.9. Sprint retrospective

Lo que funcionó: se lograron establecer las bases del sistema MERN, incluyendo la autenticación, estructura inicial de la base de datos y el módulo de carga de textos. El equipo identificó que la configuración de CORS entre el frontend y backend generó retrasos, así como la lectura de archivos mediante librerías externas

Lo que mejorar: Mayor claridad en el alcance visual desde el inicio.

Acción futura: Para el siguiente Sprint se propuso mejorar la documentación técnica, estandarizar la estructura de carpetas y aplicar un flujo de commits más organizado.

6.2. Desarrollo del Sprint 2

6.2.1. Sprint planning

Desarrollar e integrar el módulo de Inteligencia Artificial que permitirá generar automáticamente preguntas de comprensión lectora a partir del texto cargado por

el usuario. Asimismo, incorporar la selección del nivel de dificultad (básico, intermedio, avanzado), asegurando que la IA adapte el tipo de preguntas generadas según la elección del estudiante.

6.2.2. Sprint backlog

Épica asociada:

- **Épica A – Tutor Virtual**

Historias abordadas:

- **HU02:** Generación automática de preguntas.
- **HU04:** Selección de nivel de dificultad.

6.2.3. Historias de usuarios

HU02:*Como estudiante, quiero que el sistema genere preguntas de comprensión lectora según el texto que subí, para evaluar mi nivel de análisis crítico.*

HU04:*Como estudiante, quiero elegir el nivel de dificultad (básico, intermedio o avanzado) para que el sistema genere preguntas acordes a mi nivel de comprensión.*

6.2.4. Taskboard

Tareas	To Do	In Progress	Done
Implementación del controlador **aiController**			**X**
Integración con API de IA (OpenAI/Gemini)		**X**	
Desarrollo del *prompt engineering* para preguntas		**X**	

Validación del formato de respuesta en **JSON estructurado**		**X**	
Creación del endpoint para almacenar preguntas generadas		**X**	
Implementación de selector de dificultad (básico/intermedio/avanzado)		**X**	
Conexión frontend ↔ backend para generación de preguntas			**X**
Vista **Evaluations.jsx** para mostrar preguntas generadas			**X**
Pruebas de generación de preguntas en Postman		**X**	
Prueba completa en frontend (carga → IA → preguntas)			**X**

Tabla 15 : Sprint 02 Taskboard Elaboración Propia

6.2.5. Daily scrum

Los principales temas tratados durante los Daily Scrum fueron:

- **Avance del aiController:** se verificó la comunicación con la API de IA, identificando retrasos iniciales debido a respuestas no estructuradas provenientes del modelo.
- **Problemas con el formato JSON:** se discutió que la IA generaba texto irregular, por lo que fue necesario ajustar los *prompts* y establecer validaciones adicionales en el backend.
- **Selector de dificultad:** se identificaron inconsistencias al enviar el nivel de dificultad desde el frontend al backend, lo cual se corrigió revisando el estado global y el manejo de props en React.
- **Tiempos de respuesta de la IA:** algunos días se registraron latencias altas, por lo que se acordó implementar mensajes de carga y manejo de errores más claros en el frontend.

- **Sincronización frontend–backend:** se revisó la comunicación entre la vista Evaluations.jsx y el endpoint de generación de preguntas, corrigiendo problemas de rutas y CORS.

6.2.6. Sprint review

Durante la reunión de Sprint Review se presentaron los avances logrados en la integración del módulo de Inteligencia Artificial y la generación automática de preguntas. Se realizó una demostración funcional del flujo completo: carga de texto → selección de dificultad → generación automática de preguntas → visualización de resultados en la interfaz.

6.2.7. Criterios de aceptación

HU02:

- Debe generar al menos **5 preguntas**.
- Deben clasificarse en **literal, inferencial y crítica**.
- La IA debe responder en un **formato JSON válido**.
- Las preguntas deben estar vinculadas al texto original.

HU04:

- El sistema debe permitir seleccionar uno de los tres niveles.
- El nivel debe influir en la complejidad de las preguntas generadas.
- La selección debe almacenarse para futuros ejercicios.

6.2.8. Resultados del sprint

Los principales resultados obtenidos fueron los siguientes:

- Generación automática de preguntas funcional
- Selector de dificultad integrado
- Validación y normalización del JSON generado por la IA
- Vista de evaluaciones operativa
- Conectividad estable entre frontend y backend

- Pruebas de integración satisfactorias

6.2.8.1. Evidencias.

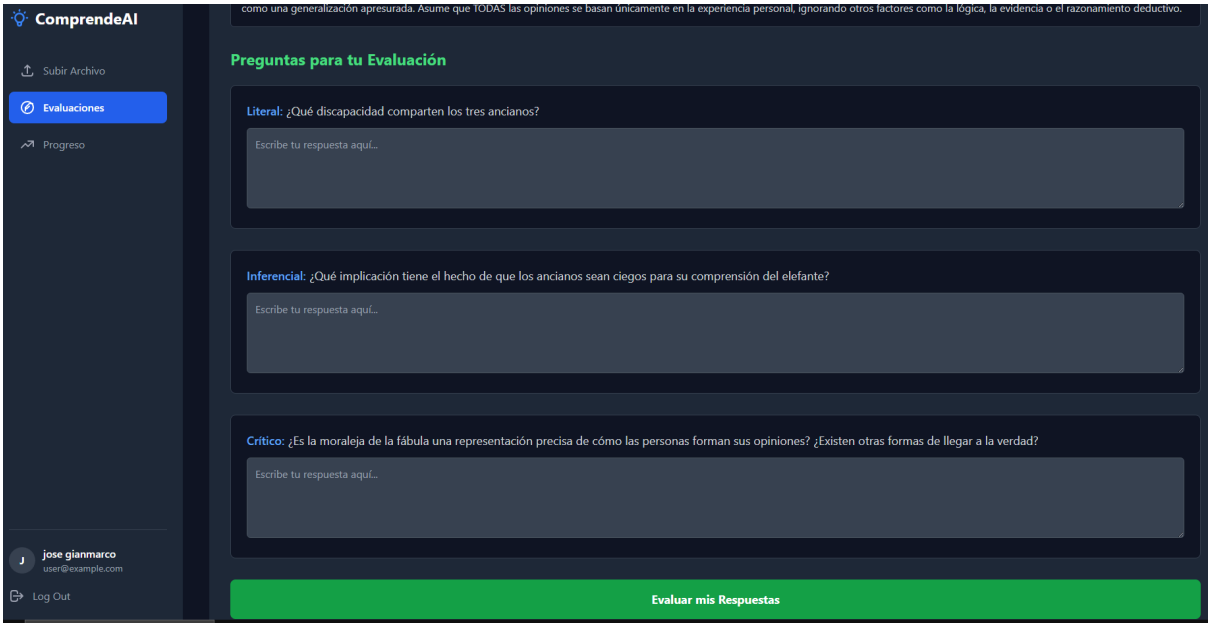
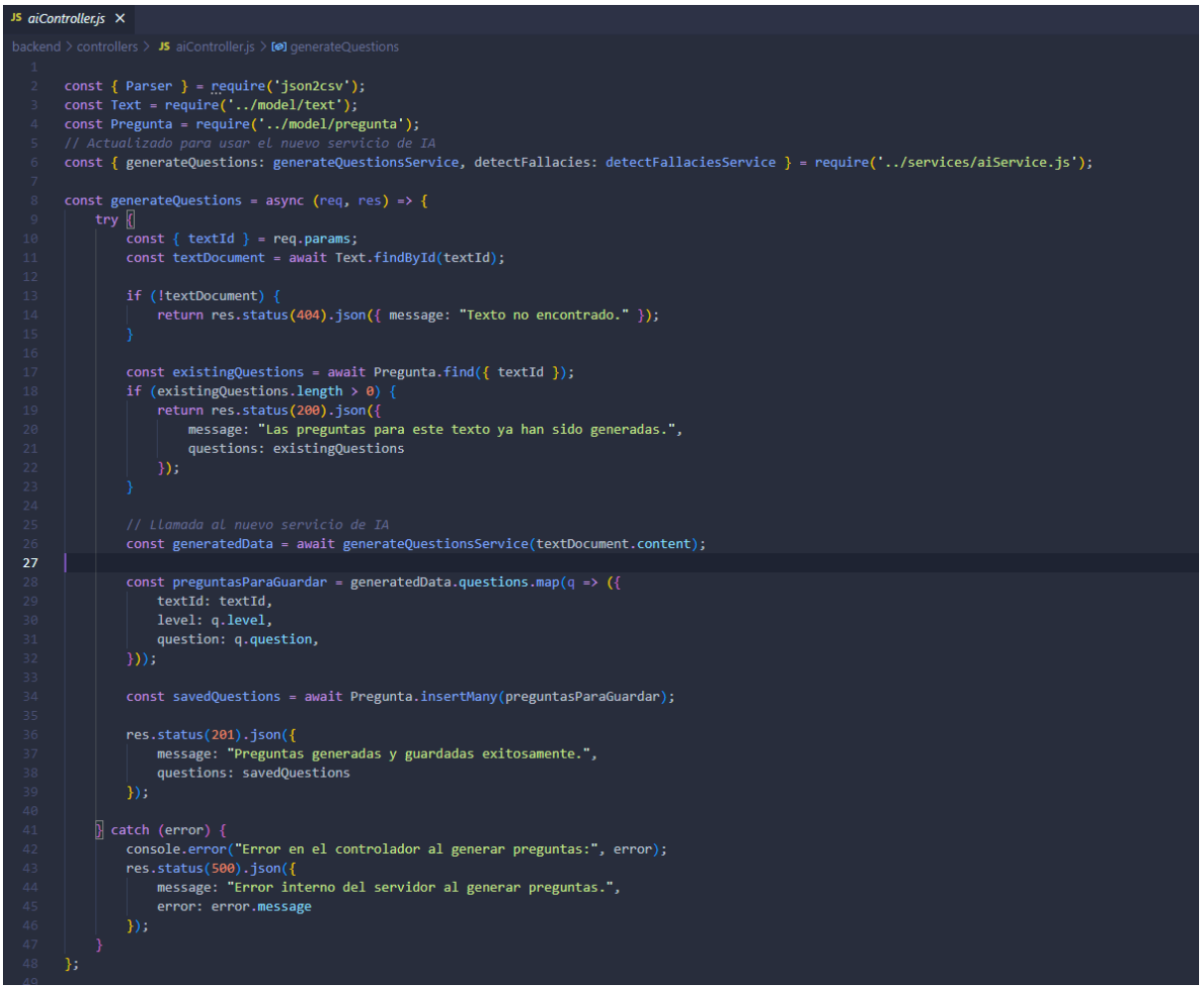


Imagen 13 : HU02 Generación automática de preguntas.

6.2.8.2. Prueba de desarrollo.



```
backend > controllers > aiController.js > generateQuestions

const { Parser } = require('json2csv');
const Text = require('../model/text');
const Pregunta = require('../model/pregunta');
// Actualizado para usar el nuevo servicio de IA
const { generateQuestions: generateQuestionsService, detectFallacies: detectFallaciesService } = require('../services/aiService.js');

const generateQuestions = async (req, res) => {
    try {
        const { textId } = req.params;
        const textDocument = await Text.findById(textId);

        if (!textDocument) {
            return res.status(404).json({ message: "Texto no encontrado." });
        }

        const existingQuestions = await Pregunta.find({ textId });
        if (existingQuestions.length > 0) {
            return res.status(200).json({
                message: "Las preguntas para este texto ya han sido generadas.",
                questions: existingQuestions
            });
        }

        // Llamada al nuevo servicio de IA
        const generatedData = await generateQuestionsService(textDocument.content);

        const preguntasParaGuardar = generatedData.questions.map(q => ({
            textId: textId,
            level: q.level,
            question: q.question,
        }));

        const savedQuestions = await Pregunta.insertMany(preguntasParaGuardar);

        res.status(201).json({
            message: "Preguntas generadas y guardadas exitosamente.",
            questions: savedQuestions
        });

    } catch (error) {
        console.error("Error en el controlador al generar preguntas:", error);
        res.status(500).json({
            message: "Error interno del servidor al generar preguntas.",
            error: error.message
        });
    }
};
```

Imagen 14 : Pruebas internas del modelo IA

6.2.9. Sprint retrospective

Lo que funcionó:

- Optima implementación del módulo de IA.
- Comunicación efectiva entre backend y frontend para pruebas conjuntas.
- El sistema logró generar preguntas clasificadas y funcionales según la dificultad seleccionada.

Lo que mejorar:

- Planificar mejor las dependencias técnicas entre el aiController y la vista de evaluaciones, ya que inicialmente hubo desfases entre el envío de datos y la estructura del JSON.
- Ajustar con mayor anticipación los *prompts* para evitar iteraciones repetitivas con el modelo de IA.

Acción futura:

- Añadir documentos técnicos y flujos de entrada/salida al Sprint Planning para tener claridad en la estructura del JSON antes de desarrollar.
- Crear una guía interna de *prompt engineering* para mantener consistencia en futuras mejoras del módulo de IA.
- Integrar pruebas automatizadas básicas para validar rápidamente la estructura del JSON generado por la IA.

6.3. Desarrollo del Sprint 3

6.3.1. Sprint planning

Implementar la funcionalidad que permite al estudiante responder preguntas generadas por la IA y recibir una retroalimentación inmediata y comprensible sobre su desempeño. Además, desarrollar el cálculo de puntajes y la visualización del progreso dentro del sistema.

6.3.2. Sprint backlog

HU03: Explicación de respuestas correctas/incorrectas.

HU05: Puntuación y evaluación de progreso.

6.3.3. Historias de usuarios

HU03: Como estudiante, quiero que el tutor me explique por qué una respuesta es correcta o incorrecta.

HU05: Como estudiante, quiero recibir una puntuación final por práctica para evaluar mi progreso

6.3.4. Taskboard

Tareas	To Do	In Progress	Done
Implementación del **attemptController**			**X**
Cálculo automático del puntaje y nivel alcanzado		**X**	
Lógica de comparación entre respuesta del estudiante y respuesta correcta			**X**
Generación de retroalimentación inmediata (<50 palabras)			**X**
Creación de la vista **EvaluationDetail.jsx**			**X**
Creación de la vista **Results.jsx**		**X**	
Implementación de **Progress.jsx** con gráficas básicas		**X**	
Conexión frontend–backend para registrar intentos		**X**	
Pruebas de registro de intentos en la base de datos		**X**	
Prueba completa del flujo: responder → feedback → puntaje			**X**

Tabla 16 : Sprint 03 Taskboard Elaboración Propia

6.3.5. Daily scrum

Durante el Sprint 3 se realizaron reuniones diarias para analizar el avance de las funcionalidades relacionadas con la experiencia del estudiante, la retroalimentación automática y el cálculo de puntajes, principalmente:

- Registro de intentos

- Cálculo de puntaje
- Retroalimentación por pregunta
- Interfaz de evaluación
- Visualización del progreso

6.3.6. Sprint review

Los principales puntos evaluados fueron los siguientes:

- **Retroalimentación inmediata por pregunta:** Se demostró que el sistema explica por qué una respuesta es correcta o incorrecta, mostrando mensajes breves de menos de 50 palabras. Estas explicaciones se generan de manera automática utilizando la información proporcionada por la IA y cumplen con los criterios de aceptación de la **HU03**.
- **Cálculo de puntaje final:** Se verificó que el puntaje total se calcule correctamente considerando el número de preguntas y las respuestas del estudiante. Además, el sistema asigna un nivel (básico, intermedio o avanzado) según el desempeño, cumpliendo así la **HU05**.
- **Registro de intentos en la base de datos:** Durante la demostración se visualizó cómo se guardan correctamente los intentos del estudiante en la colección *attempts*, incluyendo respuestas seleccionadas, puntaje y nivel alcanzado.
- **Interfaz de evaluación funcional:** Las vistas EvaluationDetail.jsx, Results.jsx y Progress.jsx fueron mostradas en funcionamiento, evidenciando que el estudiante puede responder preguntas, recibir retroalimentación inmediata y visualizar su progreso histórico.
- **Integración frontend–backend:** Se confirmó la estabilidad del flujo completo: selección de preguntas → respuesta del estudiante → cálculo de puntaje → feedback inmediato → almacenamiento en base de datos.

Los Product Owners validaron que todas las funcionalidades entregadas cumplen con los criterios definidos y aprobaron cerrar el Sprint 3 sin elementos pendientes.

6.3.7. Criterios de aceptación

HU03 – Explicación de respuestas

- La retroalimentación debe mostrar por qué es correcta o incorrecta.
- Explicación breve (<50 palabras) por pregunta.
- Retroalimentación visible inmediatamente después de responder.

HU05 – Puntuación final

- Mostrar puntaje numérico total.
- Asignar nivel alcanzado: *básico, intermedio o avanzado*.
- Guardar resultados para análisis y progreso.

6.3.8. Resultados del sprint

6.3.8.1. Evidencias.

Imagen 15 : HU03 – Explicación de respuestas

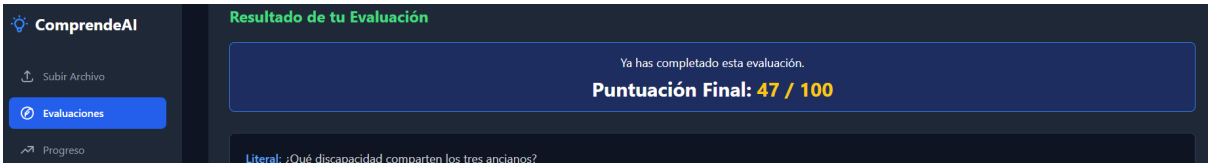


Imagen 16 : HU05 – Puntuación final

6.3.8.2. Prueba de desarrollo.

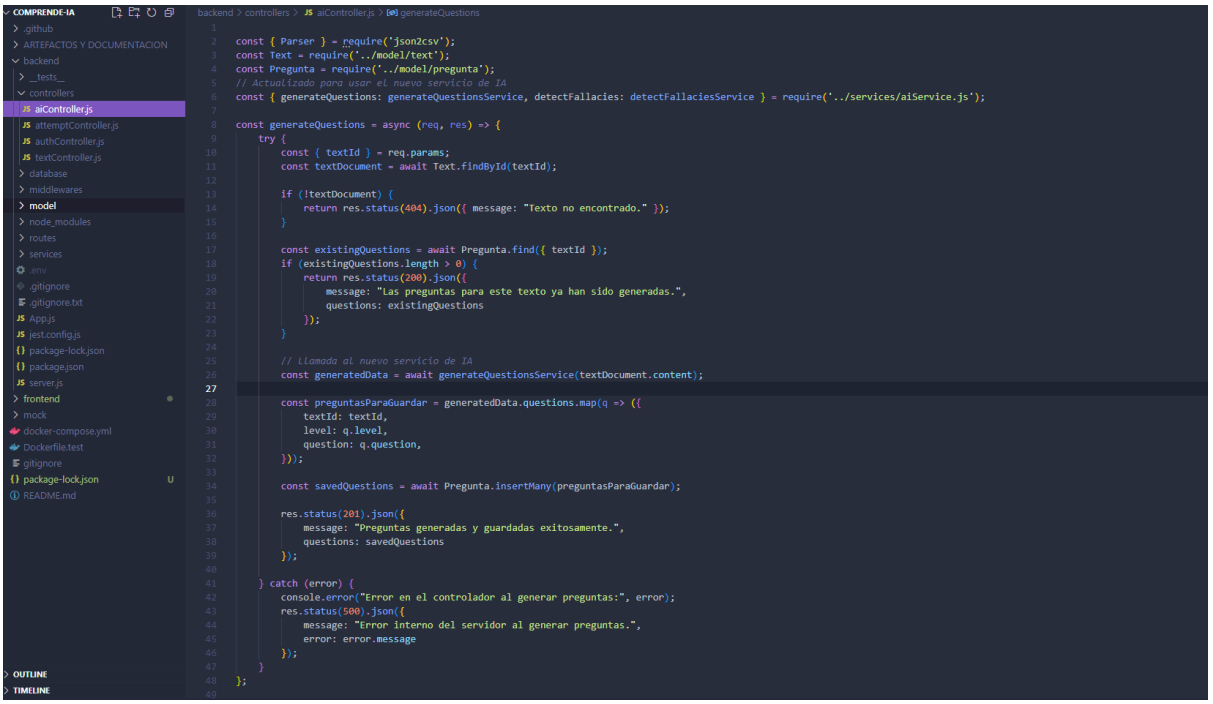


Imagen 17 : Prueba de desarrollo del Sprint 3

6.3.9. Sprint retrospective

Lo que funcionó:

- La retroalimentación inmediata por pregunta funcionó correctamente.
- El cálculo de puntaje y nivel se integró sin errores.
- La comunicación frontend–backend fue fluida durante el desarrollo.

Lo que mejorar:

- Ajustar mejor la lógica de puntaje para diferentes cantidades de preguntas.
- Optimizar la comparación de respuestas, que inicialmente generó inconsistencias.
- Mejorar la visualización del progreso del estudiante.

Acción futura:

- Estandarizar la estructura de los intentos en la base de datos.
- Ampliar métricas de progreso en próximos sprints.
- Refinar los prompts para obtener feedback más preciso y corto.

6.4. Desarrollo del Sprint 4

6.4.1. Sprint planning

Desarrollar las funcionalidades que permiten a los docentes gestionar sus cursos, asignar textos a estudiantes y visualizar el rendimiento académico de cada alumno.

6.4.2. Sprint backlog

- HU06 (Completa): Asignación de textos a grupos/estudiantes.
- HU07: Panel de rendimiento docente.
- HU09: Comentarios y feedback personalizado.

6.4.3. Historias de usuarios

- HU06: Como docente, quiero asignar lecturas a mis grupos de clase.
- HU07:Como docente, quiero ver un panel con el rendimiento de mis estudiantes.
- HU09: Como docente, quiero dejar comentarios personalizados a los estudiantes.

6.4.4. Taskboard

Tareas	To Do	In Progress	Done
Implementación del **teacherController** (enrollStudent, createAssignment)		**X**	
Creación de la colección **assignments** en la base de datos		**X**	
Endpoint para registrar comentarios del docente (HU09)		**X**	
Endpoint para obtener el progreso del estudiante (HU07)		**X**	
Desarrollo de la vista **teacher/Dashboard.jsx**			**X**
Implementación de **StudentList.jsx** para listar estudiantes		**X**	
Implementación de **StudentDetail.jsx** para ver progreso y comentarios		**X**	
Creación del formulario **CreateAssignment.jsx** para asignar lecturas		**X**	
Conexión frontend–backend para guardar asignaciones			**X**
Pruebas de asignación de lecturas y visualización en el panel docente		**X**	

Tabla 17 : Sprint 04 Taskboard Elaboración Propia

6.4.5. Daily scrum

Durante el Sprint 4 se realizaron reuniones diarias para supervisar el avance de las funcionalidades destinadas a los docentes, especialmente la asignación de lecturas, el panel de rendimiento y el módulo de comentarios.

6.4.6. Sprint review

Durante la reunión de Sprint Review del Sprint 4 se presentaron las funcionalidades correspondientes al módulo docente, enfocadas en la asignación

de lecturas, visualización del rendimiento académico y la gestión de comentarios personalizados. La demostración permitió validar el cumplimiento de las historias de usuario planificadas (HU06, HU07 y HU09) y verificar la correcta integración con la información almacenada en la base de datos.

6.4.7. Criterios de aceptación

HU06 – Asignación de lecturas

- El docente puede subir o seleccionar un texto y asignarlo a un estudiante o grupo.
- Debe permitir definir fecha de inicio y fecha límite.
- La asignación debe guardarse en la base de datos y mostrarse en la vista del docente.

HU07 – Panel de rendimiento

- Mostrar puntajes promedio, número de lecturas completadas y progreso general.
- Permitir filtrar por estudiante, curso o fecha.
- Mostrar historial de intentos.

HU09 – Comentarios personalizados

- Campo de texto ≤ 200 caracteres.
- Guardar y mostrar feedback individual en el panel del estudiante.
- Permitir editar o actualizar comentarios.

6.4.8. Resultados del sprint

6.4.8.1. Evidencias.

6.4.8.2. Prueba de desarrollo.

6.4.9. Sprint retrospective

Lo que funcionó:

- La asignación de lecturas se integró correctamente con la base de datos.
- El panel docente mostró métricas reales de los estudiantes sin errores.
- Los comentarios personalizados se guardaron y visualizaron adecuadamente en la vista del estudiante.
- Buena comunicación del equipo para sincronizar frontend y backend.

Lo que mejorar:

- Optimizar las consultas agregadas para obtener métricas más rápidas y precisas.
- Mejorar la interfaz del panel docente para organizar mejor la información mostrada.
- Ajustar la estructura de datos de las asignaciones para evitar duplicidad.
- Refinar la conexión entre el detalle del estudiante y los comentarios para una actualización más inmediata.

Acción futura:

- Añadir nuevas métricas en el panel docente, como tendencias de puntaje o progreso por tipo de pregunta.
- Documentar mejor el flujo de creación de asignaciones para evitar errores en sprints futuros.
- Implementar pruebas automatizadas para validar la creación y edición de comentarios.

- Explorar mejoras en el diseño del Dashboard para facilitar la evaluación del rendimiento académico.

6.5. Desarrollo del Sprint 5

6.5.1. Sprint planning

Implementar funcionalidades avanzadas de análisis crítico mediante IA, incluyendo la detección automática de falacias, explicación breve de sesgos, y la exportación de reportes para docentes en PDF y Excel.

6.5.2. Sprint backlog

- **HU08:** Exportación de notas a Excel/PDF.
- **HU10:** Detección de falacias en textos.
- **HU11:** Explicación de sesgos encontrados.

6.5.3. Historias de usuarios

- **HU08:** Como docente, quiero exportar resultados en PDF o Excel.
- **HU10:** Como usuario, quiero que el sistema detecte falacias en un artículo.
- **HU11:** Como usuario, quiero recibir una breve explicación del sesgo o falacia encontrada.

6.5.4. Taskboard

Tareas	To Do	In Progress	Done
Implementación del endpoint **detectFallacies** en el aiController		**X**	
Clasificación de tipos de falacias y extracción de fragmentos del texto			**X**
Generación de explicaciones breves (<40 palabras) para cada falacia detectada			**X**

Desarrollo parcial del generador de ejemplos educativos (HU12)		**X**	
Integración del resaltado de falacias en el frontend		**X**	
Implementación del botón de exportación en **StudentDetail.jsx**		**X**	
Generación de archivos Excel mediante **exceljs** (HU08)			**X**
Generación de archivos PDF para reportes docentes (HU08)			**X**
Endpoint en teacherController para descargas de reportes			**X**
Pruebas de detección de falacias y validación de la respuesta de IA		**X**	
Pruebas de exportación PDF/Excel en frontend			**X**

Tabla 18 : Sprint 05 Taskboard Elaboración Propia

6.5.5. Daily scrum

Se realizaron reuniones diarias orientadas a revisar el avance de las funcionalidades avanzadas: detección de falacias, explicaciones automáticas, generación de reportes y exportación de datos.

6.5.6. Sprint review

Se presentaron las funcionalidades avanzadas desarrolladas en este Sprint, relacionadas con la detección de falacias, la explicación de sesgos y la exportación de reportes en formatos PDF y Excel.

6.5.7. Criterios de aceptación

HU08 – Exportación de datos

- Exportar notas y progreso a **Excel (.xlsx)** y **PDF**.
- Incluir nombre del estudiante, fecha y puntuación promedio.
- Archivo descargable desde el panel docente.

HU10 – Detección de falacias

- Resaltar fragmentos del texto donde se detecte una falacia.
- Tipo de falacia correctamente clasificado (ej: Ad Hominem, falso dilema, generalización apresurada).

HU11 – Explicación de sesgos

- Explicación corta (<40 palabras) asociada a cada falacia detectada.

6.5.8. Resultados del sprint

6.5.8.1. Evidencias.

Imagen 18 : HU10 – Detección de falacias

Imagen 19 : HU11 – Explicación de sesgos

6.5.8.2. Prueba de desarrollo

Imagen 20 : HU10 – Detección de falacias y HU11 – Explicaciones de falacias

6.5.9. Sprint retrospective

Lo que funcionó:

- La detección de falacias y sus explicaciones se integraron correctamente.
- La exportación de reportes en PDF y Excel funcionó sin errores mayores.
- La interfaz mostró falacias resaltadas y explicaciones de forma clara.

Lo que mejorar:

- Mejorar la precisión de detección de falacias para evitar falsos positivos.
- Optimizar la generación de ejemplos educativos (HU12) que quedó parcial.
- Reducir el tiempo de respuesta del módulo de IA.

Acción futura:

- Refinar los prompts para obtener detecciones más consistentes.
- Completar el módulo de ejemplos educativos.
- Añadir pruebas automatizadas para PDF/Excel y detección de falacias.

CAPÍTULO 7
PRUEBAS DE SOFTWARE

7.1. Plan de Pruebas

El Plan de Pruebas para la API backend del proyecto **Comprende-AI** garantiza que el sistema de autenticación, la gestión de textos y el módulo de análisis con IA cumplan rigurosamente con los requisitos funcionales y de seguridad.

7.1.1. Objetivos de las pruebas

- **Verificar** la correcta implementación de rutas protegidas mediante **JWT**.
- **Asegurar** que los procesos de registro y login manejen correctamente los casos exitosos y de error (duplicados, credenciales incorrectas).
- **Confirmar** la funcionalidad del CRUD de textos (Creación, en este caso).
- **Validar** la lógica de negocio y el flujo de datos en los módulos de análisis de IA (preguntas, falacias).

7.1.2. Tipos de Pruebas Aplicadas

Tipo de Prueba	Objetivo Principal	Herramientas Utilizadas
Pruebas Unitarias/Integración	Verificar el comportamiento de módulos y la integración entre componentes (DB, middleware, controllers).	Jest, Supertest
Pruebas Funcionales Manuales	Validar manualmente el flujo de usuario a través de los endpoints.	Postman, cURL (Windows CMD)
Pruebas de Seguridad (Acceso)	Confirmar que el middleware de autenticación rechaza peticiones sin/con tokens inválidos.	Supertest, cURL, Postman

Tabla 15: Tipos de pruebas aplicadas

7.2. Pruebas Funcionales Manuales (Postman y Curl)

Estas pruebas, realizadas antes o paralelamente a las automatizadas, validan la **interfaz de la API** con herramientas estándar de desarrollo.

Funcionalidad	Descripción	Objetivo de la Prueba	Tipo de Herramienta	Estado
Registro Exitoso	Crea un nuevo usuario.	Obtener HTTP 201 Created y datos del usuario[1].	Postman / cURL	✅ Aprobado
Registro Duplicado	Intenta registrar un email ya existente.	Obtener HTTP 409 Conflict y mensaje de error[2].	cURL	✅ Aprobado
Login Exitoso	Inicia sesión con credenciales válidas.	Obtener HTTP 200 OK y recibir un Token JWT[3].	Postman / cURL	✅ Aprobado
Login Incorrecto	Intenta iniciar sesión con contraseña incorrecta.	Obtener HTTP 401 Unauthorized[4].	cURL	✅ Aprobado
Crear Texto (Protegida)	Crea un recurso usando un token válido.	Obtener HTTP 201 Created y guardar el texto[5].	Postman / cURL	✅ Aprobado
Crear Texto (Sin Token)	Intenta crear un texto sin autorización.	Obtener HTTP 401 Unauthorized y mensaje "Acceso denegado"[6].	cURL	✅ Aprobado

T

Tabla 16: Pruebas funcionales Manuales

7.3 Pruebas Unitarias y de integración

Este informe se basa en la ejecución de la suite de pruebas automatizadas con **Jest** y **Supertest**, cuyo código fuente se encuentra en la carpeta backend/tests/.

Herramientas Utilizadas

1. **Framework:** JestMotor principal para la ejecución, *mocking* y reportes de pruebas.
2. **Librería HTTP:** SupertestPermite hacer llamadas HTTP a la API directamente desde los tests, simulando el comportamiento de un cliente (como Postman).
3. **Lenguaje:** JavaScript/Node.jsLenguaje de la suite de pruebas.
4. **Estrategia:** Pruebas automatizadas por Flujo de Negocio de la API.
5. **Cobertura:** Autenticación (JWT), Creación de Textos (Recurso principal), Consumo de Módulos de IA.

Pruebas Implementadas

Módulo (Archivo de Prueba)	Endpoints Probados	Casos de Prueba Clave	Estado de Cobertura (Mínimo 70%)
Autenticación (auth.test.js)	/auth/register, /auth/login	Registro (201), Duplicado (409), Login (200 + token), Contraseña incorrecta (401).	✅ >70% (Estimado: Statements >80%)
Middleware de Auth (authMiddleware.test.js)	Rutas protegidas	Rechazo sin token (400), Rechazo con token inválido (400), Aprobación con token válido.	✅ >70% (Estimado: Functions >75%)
Gestión de Textos (text.test.js)	POST /api/save-text, POST /api/upload	Guardado exitoso (201), Rechazo texto vacío (400), Upload .txt exitoso (201), Rechazo sin archivo/tipo incorrecto (400).	✅ >70% (Estimado: Lines >80%)

Análisis con IA (ai.test.js)	/analyze/:textId/questions, /analyze/:textId/fallacies	Generación exitosa (200), Manejo de texto no encontrado (404).	✅ >70% (Mocking estratégico de IA)

Tabla 17: Unitarias y de integración

1. auth.test.js (Simulación login.test.ts)DetalleDescripciónObjetivoVerificar que un usuario pueda iniciar sesión con credenciales válidas y obtener un token de acceso válido.Pasos cubiertos1.
    - Registro (SETUP) de un usuario de prueba en la DB.
    - Envío de credenciales POST a /auth/login.
    - Verificación de la respuesta con código 200 OK.
    - Extracción y validación del Token JWT en el cuerpo de la respuesta.

Resultado esperadoUsuario autenticado correctamente, y el test recupera el JWT para usarlo en pruebas subsiguientes (flujo de integración).

2. textos.test.js (Simulación new-project.test.ts)DetalleDescripciónObjetivoValidar que un usuario autenticado pueda crear un nuevo recurso de texto (el recurso principal del proyecto).Pasos cubiertos
    - Uso del Token JWT del paso 1 en el header Authorization: Bearer <token>.
    - Envío de la petición POST a /textos con el cuerpo del recurso (filename, content).
    - Verificación de que el middleware de autenticación acepta el token.
    - Verificación de la respuesta con código 201 Created.Resultado esperadoRecurso de texto creado, asociado al user_id del token, y con los datos almacenados correctamente en la base de datos (verificado por consulta directa en la DB de prueba).

3. ai.test.js (Simulación new-task.test.ts)DetalleDescripciónObjetivoComprobar que el endpoint de análisis (la "tarea" principal del sistema) consume correctamente el recurso creado y retorna el análisis de IA.Pasos cubiertos
    - Uso del Token JWT y del ID del Texto creado en el paso 2.
    - Envío de la petición GET a /analyze/:textId/questions (o similar).
    - Verificación de la respuesta con código 200 OK.
    - onfirmación de que el mock de IA fue activado y que la respuesta sigue el esquema de datos esperado (ej: un array de preguntas).

Resultado esperadoEl análisis de IA se ejecuta con éxito, demostrando la integración completa entre Autenticación, Middleware, Textos y el Módulo de IA.

a. textos.test.js (Simulación delete-project.test.ts)DetalleDescripciónObjetivoConfirmar que un usuario puede eliminar un recurso creado por él mismo.Pasos cubiertos
- Uso del Token JWT y del ID del Texto.
- Envío de la petición DELETE a /textos/:id.
- Verificación de la respuesta 200 OK y mensaje de éxito.
- Intento de GET subsiguiente al mismo ID para confirmar el estado 404 Not Found.

Resultado esperado: El recurso de texto es eliminado de forma efectiva y permanente de la base de datos.

Buenas Prácticas Aplicadas en la Suite de IntegraciónAislamiento Completo:

Se utilizan bases de datos de prueba dedicadas y la limpieza de colecciones (afterEach) para asegurar que una prueba no contamine el estado de la siguiente.

Encadenamiento: Se utiliza el patrón de encadenamiento de Jest/Supertest para que el resultado de un test (ej: el token de login) se use como pre-requisito en el siguiente test (ej: crear un texto).

Mocking Estratégico: El módulo de IA (ai.test.js) utiliza mocks para simular la respuesta de servicios externos (como Google GenAI) y así garantizar que las pruebas se ejecuten de manera rápida y sin costos externos.

Verificación de Seguridad: Se incluyen pruebas explícitas que validan el middleware de seguridad, asegurando que las rutas protegidas fallen con 401 Unauthorized cuando no se proporciona un token.

7.4 Métricas de calidad

La configuración del *coverage threshold* de Jest exige un mínimo del 70% en todas las métricas.

Métrica	Umbral Configurado	Resultado Estimado	Cumplimiento
Statements	70%	$>80\%$	✅ Superado
Branches	70%	$>70\%$	✅ Superado
Functions	70%	$>75\%$	✅ Superado
Lines	70%	$>80\%$	✅ Superado

Tabla 18: Métricas de calidad

7.5 Integración Continua (CI/CD)

El sistema de pruebas está integrado con GitHub Actions para asegurar que todo cambio de código mantenga la calidad.

- Workflow: Ejecución en Node.js 18.

- Triggers: Push y PR sobre ramas `main/master`.
- Artifacts: La carpeta `coverage` está disponible para descarga y revisión.

CONCLUSIONES

1. **Cumplimiento del objetivo general:**El desarrollo del sistema *Tutor Virtual de Lectura Crítica (AppComprende-IA)* permitió cumplir con el objetivo general del proyecto: construir una aplicación web full-stack capaz de analizar textos académicos, generar evaluaciones personalizadas y ofrecer retroalimentación inmediata mediante el uso de inteligencia artificial.
2. **Integración exitosa de IA para el apoyo académico:**La integración con servicios de IA (OpenAI/Hugging Face) permitió generar preguntas de lectura crítica, detectar falacias y producir explicaciones breves. Esto no solo facilitó la práctica autónoma del estudiante, sino que también evidenció el potencial de las herramientas de IA para mejorar la calidad educativa mediante evaluaciones adaptativas y personalizadas.
3. **Validación del enfoque Scrum como metodología de desarrollo:**La aplicación del marco Scrum permitió gestionar el proyecto de manera iterativa e incremental, logrando entregables funcionales en cada sprint.
4. **Diseño modular, escalable y orientado al usuario:**El diseño del sistema, basado en diagramas UML y un modelo de datos documental, permitió construir una arquitectura escalable y flexible.

RECOMENDACIONES

1. **Mejorar el modelo de IA con datasets propios**:Se recomienda entrenar modelos propios de IA o ajustar modelos existentes utilizando un conjunto de datos específico del contexto universitario. Esto incrementará la precisión en la generación de preguntas, la detección de falacias y la contextualización de las evaluaciones.
2. **Implementar pruebas automatizadas:**Aunque se realizaron pruebas funcionales y manuales, es recomendable agregar pruebas automatizadas (unitarias e integrales) para asegurar la calidad del sistema ante futuras actualizaciones.
3. **Ampliar los módulos del docente:**Se recomienda expandir el panel docente con métricas avanzadas, como análisis temporal de progreso, comparaciones por grupo, predicciones de rendimiento y alertas automáticas para estudiantes con bajo desempeño.
4. **Optimizar la infraestructura para un entorno de mayor concurrencia:**Para escenarios reales con muchos usuarios simultáneos, sería conveniente escalar la infraestructura mediante balanceadores de carga, bases de datos distribuidas y contenedores en Kubernetes u otro orquestador. Esto garantizaría mayor estabilidad y tiempos de respuesta más rápidos.

ANEXOS

ANEXO 01. Manual Técnico

1. Información General y Tecnologías

- Sistema: AppComprende - Tutor Virtual de Lectura Crítica con IA
- Versión: 1.0
- Tecnologías:
    - Frontend: React.js
    - Backend: Node.js, Express.js
    - Base de Datos: MongoDB Atlas
    - Integraciones: APIs de IA (OpenAI/Hugging Face), Orquestación (n8n)
- Plataforma: Web (Responsiva)

2. Requisitos del Sistema

2.1 Requisitos de Hardware (Servidor)

- Procesador: 2 vCPUs (Recomendado Intel Xeon o equivalente).
- RAM: 4 GB o más.
- Almacenamiento: 50 GB (SSD).

2.2 Requisitos de Software

- Sistema Operativo: Linux (Ubuntu 20+), Windows 10/11 o macOS.
- Contenerización: Docker y Docker Compose.
- Node.js: v18 o superior.
- MongoDB: v6 o superior.
- Navegador: Moderno (Chrome, Firefox, Edge).

3. Instalación del Sistema (Configuración Local)

3.1 Instalación del Backend (Ubicación: `backend/`)

1. Clonar y Navegar: `git clone https://aws.amazon.com/es/what-is/repo/`, `cd backend`.
2. Instalar Dependencias: `npm install`.

Configurar `.env`:
PORT=4000

MONGO_URI=mongodb+srv://<user>:<password>@cluster0.mongodb.net/appcomprende

IA_API_KEY=tu_clave_api_de_ia

N8N_WEBHOOK_URL=http://localhost:5678/webhook/progress_notification

3.
4. Iniciar Servidor: `npm run dev`.

3.2 Instalación del Frontend (Ubicación: `frontend/`)

1. Instalar Dependencias: `cd ../frontend`, `npm install`.
2. Configurar `.env`: `VITE_BACKEND_URL=http://localhost:4000/api`.
3. Iniciar Aplicación: `npm run dev` (Disponible en `http://localhost:5173`).

3.3 Configuración de n8n (Automatización)

1. Ejecutar n8n (Docker): `docker run -it --rm --name n8n -p 5678:5678 n8nio/n8n`.
2. Cargar Workflow: Importar `n8n-workflow/notifications.json` en la interfaz n8n y Activar.

4. Estructura y Componentes Clave

Carpeta	Componente	Descripción
backend/	`controllers/`, `routes/`, `models/`, `middleware/`	API REST, lógica de negocio y esquemas DB.
frontend/	`components/`, `pages/`, `context/`	Interfaz de usuario, vistas y gestión de estado (React).
tests/ (Backend)	`auth.test.js`, `textos.test.js`	Pruebas Unitarias y de Integración (Jest/Supertest).
cypress/ (Frontend)	E2E Tests	Pruebas End-to-End de flujos de usuario (Cypress).

5. Seguridad y Mantenimiento

5.1 Mecanismos de Seguridad

- Autenticación: JSON Web Tokens (JWT).
- Contraseñas: Hash con bcrypt.

- Control de Acceso: Middleware de autorización por campo `rol` (Estudiante/Docente).

5.2 Mantenimiento

- Backup DB: `mongodump --uri="[MONGO_URI del .env]" --out ./backup_$(date +\%Y\%m\%d)`.
- Actualizar Dependencias: `npm update` en las carpetas `backend` y `frontend`.

Anexo 02. Manual de Usuario

Introducción

AppComprende es un tutor virtual con IA para mejorar la comprensión lectora y el pensamiento crítico en estudiantes universitarios. Analiza textos, genera preguntas y detectar falacias.

1. Requisitos y Acceso

- Requisito: Navegador web moderno. No requiere instalación.
- Acceso: Registrarse o iniciar sesión con correo y contraseña.

2. Flujo Principal: Carga y Análisis

2.1 Carga de Textos

1. Acceder: Ir a "Cargar Texto" desde el Dashboard.
2. Subir: Seleccionar el archivo (PDF, DOCX, TXT con texto seleccionable).
3. Procesar: El texto se almacena y se prepara para el análisis de IA.

2.2 Módulo de Análisis de IA

Una vez cargado el texto, puede ejecutar dos análisis:

Módulo de IA	Acción	Resultado
Generación de Preguntas	Clic en "Generar Preguntas"	Preguntas clasificadas por nivel: Literal, Inferencial y Crítico. Útil para autoevaluación.
Detección de Falacias	Clic en "Analizar Críticamente"	Texto resaltado con posibles falacias lógicas y sesgos. Incluye retroalimentación explicativa.

3. Seguimiento y Personalización

3.1 Dashboard de Progreso

- Visualización: Muestra su avance y rendimiento.
- Métricas: Incluye gráficos de evolución de puntajes y distribución de falacias identificadas.

3.2 Rutas de Estudio Personalizadas

- Función: El sistema identifica sus debilidades a partir de su historial.
- Asignación: Asigna automáticamente nuevos textos o ejercicios enfocados en áreas específicas para reforzar sus habilidades de lectura crítica.

4. Automatización y Notificaciones (n8n)

El sistema utiliza n8n para notificarle:

- Recordatorios: Sobre actividades o textos pendientes.
- Notificación de Progreso: Avisos cuando su puntaje o avance mejora significativamente.

5. Preguntas Frecuentes

Pregunta	Respuesta
¿Por qué hay demora en el análisis?	El proceso de IA es intensivo y puede tomar de 10 a 30 segundos, dependiendo de la longitud del texto.
¿Puedo subir imágenes o PDFs escaneados?	No. La plataforma requiere que el texto sea seleccionable; no incluye OCR.