# AI Coding Tutor:

# Desarrollo de un Asistente Inteligente para Programación

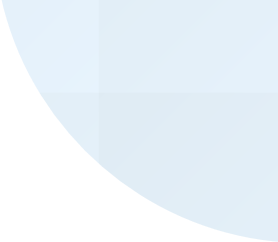


**Título del Proyecto:** AI Coding Tutor

**Autor/es:** Carlos Arroyo Caballero

**Ciclo Formativo:** Desarrollo de Aplicaciones Multiplataforma

**Curso Académico:** 2024-2025



**Fecha de Entrega:** [No me acuerdo]

****

## Índice

1. [Introducción](#X0e2973c2e3c317285bf344b2034a52754788c6d)
2. [Estado del Arte](#X19ee09d8e2e9703f46f5f45001f1ce70beae54d)
3. [Análisis de Requisitos](#Xed4393332d9ab9786ea04f6697ba2bddff126c1)
4. [Diseño del Sistema](#Xa5ef823f284acd3d03769d74ddc98ba243b17a7)
5. [Base de Datos](#X540976099df20f679173496aab20b800f3a85ac)
6. [Backend](#X905a47bb82999ab1aef9e919d34994ed7c6ddd2)
7. [Frontend](#X8630f05130a485e6b5800911d44d53fe231f72e)
8. [Integración con IA](#Xa97e35203ec989f0a3f6beb1f7399eaa30594a8)
9. [Seguridad](#Xefcb36cdb9f9ac41b4061a38e42b42d83715ce7)
10. [Pruebas](#X1732d2506b4b2444dba36ae8345d33d16fbd828)
11. [Despliegue](#Xcb9a17ea45480a3ac485e68613ee960bf673166)
12. [Conclusiones y Trabajo Futuro](#X1d3fb8f183439b026c69bed400b0a89a319c307)

1

Introducción

## 1. Introducción

### 1.1 Contexto y Motivación

Desde hace un tiempo me he visto fascinado con el veloz y constante avance de la IA, y habiendo hecho cursos relacionados al mundo de la inteligencia artificial como son por ejemplo, el fascinante curso que tiene Harvard en su Plataforma de aprendizaje: [CS50-Harvard-University-AI-Course](https://pll.harvard.edu/course/cs50s-introduction-artificial-intelligence-python), o la famosa serie de videos subidos por uno de los cofundadores de OpenAI, [Andrej Karpathy](https://www.youtube.com/@AndrejKarpathy). Yo tenía claro que mi proyecto tendría que circular en torno a la Inteligencia Artificial.

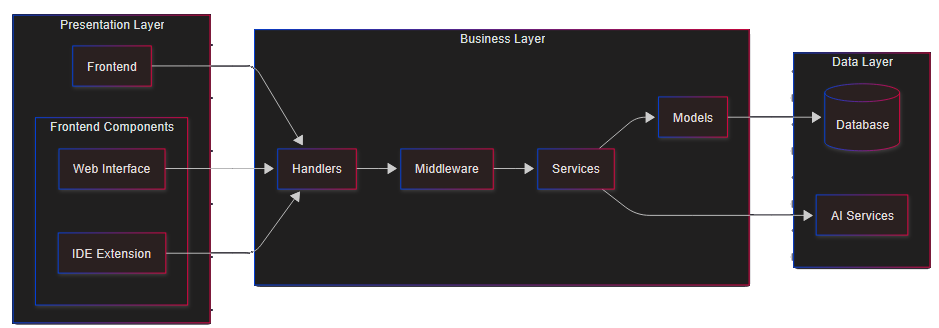
Mi primera idea fue una salida de pasión hacia un mundo que ha sido una gran parte de mi vida, el mundo de los videojuegos, pero que al final terminé desechando dicha idea debido a una falta de implicaciones reales en el mundo laboral. En un principio, iba a hacer un videojuego, o más bien una demo, mi intención era crear un mundo cruel, que se sintiese vivo y donde hubieran depredadores y una cantidad limitada de recursos, a dicho mundo luego le introducería una nueva raza o criaturas que funcionasen a través de una red neuronal y cuya IA sería entrenada con anterioridad para que estos pudiesen vivir y florecer en dicho mundo, una gran parte de mi inspiración para dicho proyecto sería el famoso juego 2D, Rainworld, más concretamente los scavengers, cuyo diseño aparece en la imagen, quería ver que tan bien podría adaptarse una IA a dicho mundo.

Finalmente, opté por un TFG que, pese a no ser necesariamente menos díficil, creí que tenía un mayor valor empresarial, menos riesgo de entregas, y una mucho más amplia gama de aprendizaje. Me decidí por hacer un proyecto Full-Stack.

### 1.2 Resumen del proyecto

**¡El AI Coding Tutor!**

Un asistente de programación, personalizable para instituciones públicas y empresas privadas, que contaría de un backend programado en Go o Golang para gestionar la capa de negocio (llamadas a los diferentes modelos, acceso a la base de datos…), un frontend dividido en dos partes, la extensión dentro de visual studio code y una página web, en la que los administradores podrán iniciar sesión y modificar, añadir o eliminar los modelos de IA, asi como el promt introducido por el mismo. Una base de datos Postgress para almacenar los diferentes modelos, feedback para posiblemente entrenar bia fine-tunning nuevos modelos de IA y los usuarios.



Finalmente, todo quedará concatenado haciendo uso Docker, con un Docker-compose file para la puesta en producción.

### 1.3 Objetivos del Proyecto

El objetivo principal de este proyecto es desarrollar un plugin para entornos de desarrollo integrado (IDE) que actúe como tutor de programación inteligente, proporcionando asistencia personalizada a los usuarios según su nivel de experiencia. Además añadiendo un nivel extra de personalización y testing directo para dichos modelos dentro de una página web propia.

Los objetivos específicos incluyen:

1. Desarrollar un sistema de análisis de código en tiempo real
2. Implementar un sistema de consultas basado en IA
3. Crear una interfaz de usuario intuitiva y accesible
4. Garantizar la seguridad y privacidad de los usuarios
5. Proporcionar retroalimentación personalizada según el nivel de experiencia
6. TODO

Los objetivos complementarios incluyen:

1. Modelo de IA personalizado bia fine-tunning (no solamente modificación el promt)
2. Puesta en producción dentro de un servicio cloud IaaS .

### 1.4 Estructura del Documento

Este documento está organizado en 12 capítulos principales que cubren todos los aspectos del desarrollo del proyecto, desde el análisis inicial hasta la implementación y conclusiones. Cada capítulo se centra en un aspecto específico del sistema, proporcionando detalles técnicos y justificaciones de diseño.

2

Estado del Arte

## 2. Estado del Arte

### 2.1 Herramientas Existentes

En el mercado actual existen varias herramientas que ofrecen asistencia en la programación:

1. **GitHub Copilot**
   * Asistente de código basado en IA
   * Integración con múltiples IDEs
   * Generación de código basada en contexto
2. **Amazon CodeWhisperer**
   * Sugerencias de código en tiempo real
   * Soporte para múltiples lenguajes
   * Integración con AWS
3. **Tabnine**
   * Autocompletado inteligente
   * Aprendizaje del estilo de código del usuario
   * Soporte para múltiples lenguajes

Pero ninguna de estas herramientas “mainstream” permite un nivel de configuración adaptado, forzándote a utilizar sus modelos de IA.

La principal razón que me llevo a elegir este proyecto finalmente, fue que fui incapaz de encontrar un servicio que ofreciese exactamente lo que yo buscaba, un nivel de **personalización independiente** de los **proveedores** de IA, que permitiese desde un nivel superior (administradores) modificar dichos modelos directamente, afectando así al sistema perteneciente en su totalidad, esto genera un nivel de control y seguridad superior, además de evitar posibles dependencias con los diferentes proveedores, siempre dando al cliente la posibilidad de elegir su mejor herramienta.

### 2.2 Tecnologías Elegidas para el Proyecto

1. **Go (Golang):**

Se trata de un lenguaje de programación desarrollado por Google, conocido por su rendimiento, simplicidad y excelente soporte para programación concurrente, haciéndolo ideal para servidores y microservicios.

1. **PostgreSQL**:

Sistema de gestión de bases de datos relacional y de código abierto, reconocido por su robustez, extensibilidad y cumplimiento de estándares SQL avanzados.

1. **Docker**:

Plataforma que permite empaquetar aplicaciones y sus dependencias en contenedores, garantizando que funcionen igual en cualquier entorno (ideal para despliegue y desarrollo).

1. **React-router v7:**

framework basado en React que permite crear navegación entre páginas en aplicaciones de una sola página (SPA), con mejoras en rendimiento y rutas anidadas más simples.

1. **Typescript:**

Superset de JavaScript que añade tipado estático y herramientas de desarrollo más sólidas, facilitando la detección de errores antes de ejecutar el código.

1. **Tailwind css:**

Framework de CSS basado en clases utilitarias, que permite diseñar interfaces de forma rápida y coherente directamente desde el HTML o JSX.

1. **Swagger:**

Herramienta para documentar y probar APIs REST. Permite generar documentación interactiva que facilita a los desarrolladores entender y consumir servicios web.

1. **JWT (JSON Web Token):**Estándar para transmitir datos de forma segura entre partes como un token firmado, comúnmente usado para autenticación y autorización en aplicaciones web.

### 2.3 Estructura Elegida para el Proyecto

1. Backend:
   * Go (Golang) para el servidor
   * PostgreSQL para la base de datos
   * Docker para contenedorización
2. Frontend:
   * React-router v7 para la interfaz web
   * TypeScript para el desarrollo de la extensión
   * Tailwind CSS para estilos
3. IA:
   * Integración con múltiples proveedores (Groq, GPT)
   * Modelos de lenguaje avanzados
   * Sistema de prompts personalizable

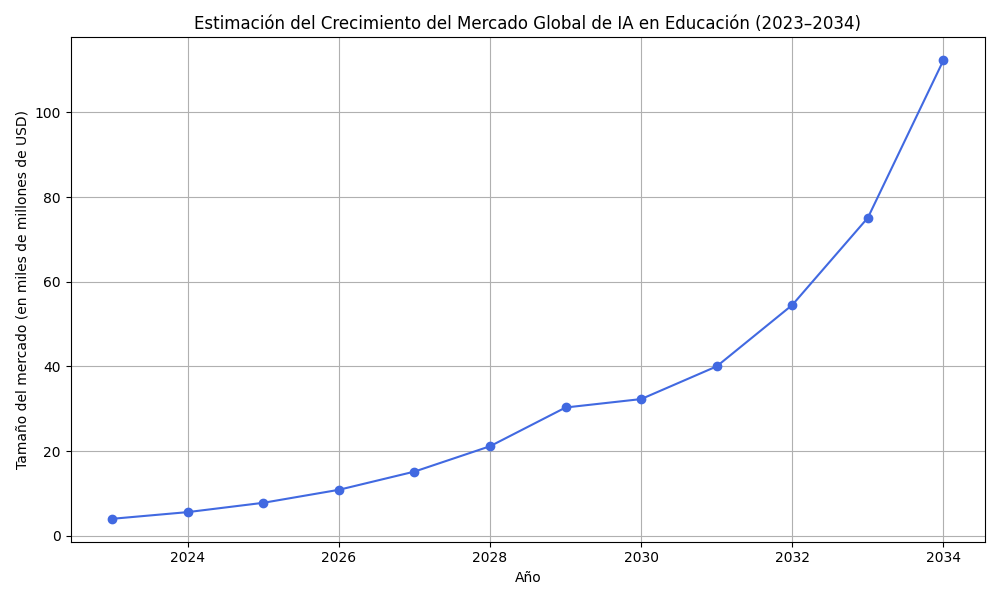
### 2.4 Investigación en IA Aplicada a la Programación en el Ámbito Educativo

**Modelos de IA en Programación:** La IA, especialmente aprendizaje automático y profundo, mejora la enseñanza de programación al generar código, detectar errores y personalizar el aprendizaje. Desde 2020, se enfoca en aulas y evaluaciones ([Artificial intelligence in computer programming education](https://www.sciencedirect.com/science/article/pii/S2666920X25000438)).

**Técnicas de NLP:** Claves para el aprendizaje adaptativo, permiten entender y generar texto de código, mejorando la interacción estudiante-herramienta ([Artificial intelligence in education](https://www.sciencedirect.com/science/article/pii/S0957417424010339)).

**Sistemas de Tutoría Inteligente:** Ofrecen asistencia personalizada, beneficiando a estudiantes e instructores, pero enfrentan debates sobre evaluación y ética ([Systematic literature review on opportunities, challenges](https://www.sciencedirect.com/science/article/pii/S2666920X2200073X" \t "_blank)).

**Evaluación de Efectividad:** Se mide por diseño de cursos, implementación, retroalimentación y monitoreo. Hay beneficios claros, pero controversias sobre métodos y desafíos éticos persisten ([A meta systematic review of artificial intelligence](https://educationaltechnologyjournal.springeropen.com/articles/10.1186/s41239-023(nil)436-z)).

En resumen, es indudable que la IA es una herramienta con un gran potencial tanto en el mundo laboral como el educativo, pero conlleva una serie de problemas que este proyecto espera poder solucionar o al menos aliviar en gran medida.

* [Fuentes](#_11._Bibliografia)

3

Análisis

## 3.1 Usuarios

La aplicación, por temas de seguridad, deberá de admitir la existencia de diferentes roles.

Es imprescindible marcar unos niveles de acceso, de manera que cada rol solo pueda utilizar y/o modificar aquello que le ha sido asignado.

Por defecto, la aplicación contendrá tres tipos de usuarios/clientes, pero esto podrá verse ampliado de forma dinámica si fuese necesario, en la actualidad existen los usuarios administradores, que pueden modificar los diferentes modelos de IA usados en la aplicación, añadir más o eliminar los ya existentes, además de poder acceder a todas las subpáginas dentro de la web. A consiguiente estaría el usuario normal (user) que también tiene acceso a la página web y ocupa un espacio en la base de datos, pero no a la configuración donde se pueden modificar las distintas configuraciones.

Finalmente existirá el usuario dentro del IDE (En este caso, Visual Studio Code), que no pertenecerá a la base de datos, este usuario solo hará llamadas al backend desde el IDE y recibirá las respuestas automáticamente para ayudar en la programación, no tendrá un espacio en la base de datos ni mantendrá ningún tipo de sesión de momento.

| **Tipo de Usuario** | **Acceso a Web** | **Acceso a Configuración de IA** | **Presencia en Base de Datos** | **Descripción** |
| --- | --- | --- | --- | --- |
| **Administrador** | Sí | Sí | Sí | Puede modificar modelos de IA, añadir/eliminar modelos y acceder a toda la web. |
| **Usuario Normal (user)** | Sí | No | Sí | Puede acceder a la web, pero no a configuraciones avanzadas. |
| **Usuario en IDE (VS Code)** | No | No | No | Hace llamadas al backend desde el IDE sin iniciar sesión ni persistencia. |

Es imprescindible tener en cuenta que dependiendo del cliente, se adaptarían los usuarios y roles adecuadamente, como por ejemplo, que el usuario dentro del IDE tenga que iniciar sesión con anterioridad para una institución pública de forma que se pudiese ver exactamente que queries o asistencia la IA le ha proporcionado durante un examen o si la ha usado en primer lugar.

## 3.2 Análisis de Requisitos

### 3.2.1 Requisitos Funcionales

**1. Requisitos Funcionales del Backend**

El backend, implementado en Go con el framework Gin, actúa como intermediario entre la extensión/web, la base de datos y los servicios de IA externos.

1. Gestión de Consultas de Usuarios
   * **RF-B1**: El backend debe aceptar solicitudes POST en el endpoint /query con un cuerpo JSON que incluya query (consulta del usuario) y level (nivel de competencia: novice, medium, expert).
   * **RF-B2**: Debe generar un prompt específico basado en el nivel de competencia del usuario y enviarlo al servicio de IA seleccionado.
   * **RF-B3**: Debe devolver una respuesta JSON con un id único y el response del servicio de IA.
2. Análisis de Código
   * **RF-B4**: El backend debe aceptar solicitudes POST en el endpoint /analyze con un cuerpo JSON que incluya code (código a analizar) y level.
   * **RF-B5**: Debe enviar el código al servicio de IA con un prompt que solicite análisis y sugerencias, ajustado al nivel del usuario.
   * **RF-B6**: Debe devolver una respuesta JSON con una lista de sugerencias (formato: [{ "line": número, "message": texto }]).
3. Gestión de Retroalimentación
   * **RF-B7**: El backend debe aceptar solicitudes POST en el endpoint /feedback con un cuerpo JSON que incluya id (de la consulta) y feedback (positiva o negativa).
   * **RF-B8**: Debe actualizar la retroalimentación asociada a la consulta en la base de datos.
4. Gestión de Configuraciones
   * **RF-B9**: El backend debe aceptar solicitudes POST en el endpoint /settings con un cuerpo JSON que incluya service (e.g., query, analyze) y config (proveedor de IA, modelo, clave API cifrada, prompts).
   * **RF-B10**: Debe permitir configuraciones específicas por servicio (query y analyze pueden usar diferentes proveedores y modelos de IA).
   * **RF-B11**: Debe recargar las configuraciones dinámicamente tras una actualización sin reiniciar el servidor.
5. Autenticación y Autorización
   * **RF-B12**: El backend debe aceptar solicitudes POST en el endpoint /login con un cuerpo JSON que incluya username y password.
   * **RF-B13**: Debe validar las credenciales contra la base de datos y devolver un token JWT si son correctas.
   * **RF-B14**: Debe restringir el acceso al endpoint /settings a usuarios con rol admin, verificado mediante el token JWT.
6. Documentación de API
   * **RF-B15**: El backend debe proporcionar documentación interactiva de la API en /swagger/index.html utilizando Swagger/OpenAPI, detallando todos los endpoints, parámetros y respuestas.

**2. Requisitos Funcionales de la Metodología Docker**

El sistema utiliza Docker para contenerizar y desplegar los servicios en Google Cloud.

1. Contenerización
   * **RF-D1**: El backend debe estar contenido en una imagen Docker basada en un archivo Dockerfile que compile y ejecute la aplicación Go.
   * **RF-D2**: La base de datos PostgreSQL debe estar contenida en una imagen oficial (postgres:13) con volúmenes para persistencia de datos.
   * **RF-D3**: El archivo docker-compose.yml debe definir servicios para el backend (backend) y la base de datos (db), con dependencias y configuraciones de red.
2. Variables de Entorno
   * **RF-D4**: El contenedor del backend debe leer variables de entorno como DB\_HOST, DB\_PORT, DB\_USER, DB\_PASSWORD, DB\_NAME, y ENCRYPTION\_KEY para conectar a la base de datos y cifrar claves API.
   * **RF-D5**: Las claves API iniciales y otras configuraciones sensibles deben inyectarse como variables de entorno durante el despliegue.
3. Despliegue
   * **RF-D6**: El sistema debe ser desplegable en Google Cloud utilizando Docker Compose o Kubernetes, asegurando escalabilidad y alta disponibilidad.
   * **RF-D7**: Los contenedores deben exponer puertos específicos (e.g., 8080 para el backend) mapeados a la máquina host.

**3. Requisitos Funcionales del Frontend (Extensión de VS Code)**

La extensión de VS Code es el punto de interacción principal para los usuarios.

1. Interfaz de Consultas
   * **RF-F1**: La extensión debe proporcionar un comando (e.g., AI Coding Tutor: Ask a Question) que abra un campo de entrada para que el usuario escriba una consulta.
   * **RF-F2**: Debe permitir al usuario seleccionar su nivel de competencia (novice, medium, expert) mediante un menú desplegable o configuración.
   * **RF-F3**: Debe enviar la consulta al endpoint /query y mostrar la respuesta en una ventana emergente o panel lateral.
2. Análisis de Código
   * **RF-F4**: La extensión debe proporcionar un comando (e.g., AI Coding Tutor: Analyze Code) que envíe el código seleccionado o el archivo activo al endpoint /analyze.
   * **RF-F5**: Debe mostrar las sugerencias devueltas como decoraciones en el editor (e.g., subrayados con mensajes en cada línea relevante).
3. Retroalimentación
   * **RF-F6**: La extensión debe permitir al usuario calificar la respuesta del IA (positiva o negativa) mediante botones o un formulario simple.
   * **RF-F7**: Debe enviar la retroalimentación al endpoint /feedback con el id de la consulta correspondiente.
4. Configuración
   * **RF-F8**: La extensión debe permitir configurar la URL del backend (e.g., http://localhost:8080) en las preferencias de VS Code.
5. Usabilidad
   * **RF-F9**: La extensión debe ser fácil de instalar desde el Marketplace de VS Code y no requerir configuraciones complejas para usuarios básicos.

**4. Requisitos Funcionales de la Base de Datos**

La base de datos PostgreSQL almacena datos críticos para el sistema.

1. Almacenamiento de Consultas
   * **RF-DB1**: La tabla queries debe almacenar cada consulta con columnas: id (UUID), query (texto), level (varchar), response (texto), feedback (varchar nullable), y created\_at (timestamp).
   * **RF-DB2**: Debe soportar restricciones para garantizar que level sea novice, medium, o expert, y feedback sea positive, negative, o null.
2. Almacenamiento de Configuraciones
   * **RF-DB3:** La tabla settings debe almacenar configuraciones por servicio con columnas: id (serial), service (varchar único), config (JSONB), y updated\_at (timestamp).
   * **RF-DB4**: El campo config debe incluir ai\_provider, ai\_model, encrypted\_api\_key, y prompts como un objeto JSON.
3. Autenticación
   * **RF-DB5**: La tabla users debe almacenar usuarios con columnas: id (serial), username (varchar único), password\_hash (varchar), role (varchar: admin o user), y created\_at (timestamp).
4. Integridad y Seguridad
   * **RF-DB6**: Las claves API en la tabla settings deben almacenarse cifradas usando una clave de cifrado externa (ENCRYPTION\_KEY).
   * **RF-DB7**: La base de datos debe soportar transacciones para operaciones como la actualización de retroalimentación y configuraciones.

**5. Requisitos Funcionales del Sitio Web Administrativo**

El sitio web, implementado en Remix, proporciona una interfaz para administradores.

1. Autenticación
   * **RF-W1**: El sitio debe ofrecer una página de inicio de sesión (/login) que acepte username y password y redirija al dashboard tras la autenticación.
   * **RF-W2**: Debe generar un token JWT y almacenarlo en una cookie segura (HttpOnly).
2. Gestión de Configuraciones
   * **RF-W3**: El sitio debe ofrecer una página /admin/settings que muestre formularios para configurar cada servicio (query, analyze), incluyendo proveedor, modelo, clave API, y prompts.
   * **RF-W4**: Debe enviar las actualizaciones al endpoint /settings y mostrar una confirmación de éxito.
3. Seguridad
   * **RF-W5**: Todas las rutas administrativas deben estar protegidas, requiriendo un rol admin verificado por el token JWT.
   * **RF-W6**: El sitio debe usar HTTPS para todas las comunicaciones.
4. Diseño
   * **RF-W7**: El sitio debe tener un diseño formal con gradientes sutiles (e.g., en fondos o botones), tipografía clara (e.g., Inter), y ser responsivo.

### 3.2.2 Requisitos No Funcionales

1. **Rendimiento**:

* **RNF-R1:** Tiempo de respuesta < 2 segundos
* **RNF-R2:** Soporte para múltiples usuarios

1. **Seguridad**:

* **RNF-S1:** Encriptación de datos
* **RNF-S2:** Uso de tokens JWT en la página web

1. **Escalabilidad**:

* **RNF-E1:** Arquitectura modular

## 3.3 Casos de Uso

**1. Registro de Usuario**

**Actores:** Usuario no registrado  
**Precondiciones:** Ninguna

**Flujo Principal:**

1. El usuario accede a la página de registro.
2. El sistema muestra el formulario de registro.
3. El usuario introduce:
   * Nombre
   * Apellido
   * Email
   * Nombre de usuario
   * Contraseña
4. El frontend valida el formato de los datos.
5. El frontend envía los datos al backend (/api/v1/register).
6. El backend:
   * Verifica la unicidad de email y nombre de usuario.
   * Valida la fortaleza de la contraseña.
   * Encripta la contraseña.
   * Crea el usuario en la base de datos.
7. El sistema muestra un mensaje de éxito.
8. El usuario es redirigido a la página de inicio de sesión.

**Flujos Alternativos:**

* 4a. Datos inválidos: El sistema muestra errores de validación.
* 6a. Email o nombre de usuario duplicado: El sistema muestra error de duplicación.
* 6b. Error del servidor: El sistema muestra un error genérico.

**2. Inicio de Sesión**

**Actores:** Usuario registrado  
**Precondiciones:** Usuario debe estar registrado

**Flujo Principal:**

1. El usuario accede a la página de inicio de sesión.
2. El sistema muestra el formulario de login.
3. El usuario introduce:
   * Nombre de usuario o email
   * Contraseña
4. El frontend envía las credenciales al backend (/api/v1/login).
5. El backend:
   * Valida las credenciales
   * Genera un token JWT
6. El frontend almacena el token en localStorage.
7. El sistema redirige al dashboard.

**Flujos Alternativos:**

* 5a. Credenciales inválidas: El sistema muestra un error.
* 5b. Error del servidor: El sistema muestra un error genérico.

**3. Gestión de Configuración**

**3.1 Configuración de Servicios AI**

**Actores:** Usuario autenticado  
**Precondiciones:** Usuario debe estar logueado

**Flujo Principal:**

1. El usuario accede a la sección de configuración.
2. El frontend obtiene la configuración actual (/api/v1/settings).
3. El sistema muestra el formulario de configuración.
4. El usuario configura:
   * Proveedor de IA
   * Modelo
   * API Key
   * Prompts personalizados
5. El frontend envía la configuración al backend (/api/v1/settings).
6. El backend:
   * Valida el token
   * Encripta la API Key
   * Guarda la configuración
7. El sistema muestra un mensaje de confirmación.

**Flujos Alternativos:**

* 2a. Error al obtener configuración: El sistema muestra un error.
* 5a. Datos inválidos: El sistema muestra errores de validación.
* 6a. Error del servidor: El sistema muestra un error genérico.

**4. Interacción con IA**

**4.1 Consulta al Tutor AI**

**Actores:** Usuario autenticado  
**Precondiciones:** Usuario logueado, configuración válida

**Flujo Principal:**

1. El usuario accede a la interfaz de consultas.
2. El usuario introduce:
   * Consulta
   * Nivel de experiencia
   * Contexto (opcional)
3. El frontend envía la consulta al backend (/api/v1/query).
4. El backend:
   * Valida el token
   * Obtiene la configuración de IA
   * Procesa la consulta con el servicio de IA
   * Guarda la interacción
5. El frontend muestra la respuesta.
6. El usuario puede:
   * Copiar la respuesta
   * Proporcionar feedback
   * Realizar seguimiento

**Flujos Alternativos:**

* 4a. Error de IA: El sistema muestra un error específico.
* 4b. Error del servidor: El sistema muestra un error genérico.

**4.2 Análisis de Código**

**Actores:** Usuario autenticado  
**Precondiciones:** Usuario logueado, configuración válida

**Flujo Principal:**

1. El usuario accede a la interfaz de análisis.
2. El usuario:
   * Sube o escribe el código
   * Selecciona el nivel de experiencia
3. El frontend envía el código al backend (/api/v1/analyze).
4. El backend:
   * Valida el token
   * Analiza el código
   * Genera sugerencias
   * Guarda el análisis
5. El frontend muestra:
   * Sugerencias
   * Mejoras propuestas
   * Ejemplos de código
6. El usuario puede:
   * Aplicar cambios
   * Guardar el análisis
   * Proporcionar feedback

**Flujos Alternativos:**

* 4a. Código inválido: El sistema muestra error de sintaxis.
* 4b. Error de análisis: El sistema muestra un error específico.

**5. Gestión de Feedback**

**5.1 Proporcionar Feedback**

**Actores:** Usuario autenticado  
**Precondiciones:** Usuario debe tener interacciones previas

**Flujo Principal:**

1. El usuario accede al historial de interacciones.
2. El usuario selecciona una interacción.
3. El usuario proporciona feedback:
   * Positivo
   * Negativo
   * Neutral
4. El frontend envía el feedback al backend (/api/v1/feedback).
5. El backend:
   * Valida el token
   * Actualiza el feedback
6. El sistema muestra un mensaje de confirmación.

**Flujos Alternativos:**

* 4a. Error al enviar feedback: El sistema muestra un error.
* 5a. Error del servidor: El sistema muestra un error genérico.

**6. Gestión de Perfil**

**6.1 Ver Perfil**

**Actores:** Usuario autenticado  
**Precondiciones:** Usuario logueado

**Flujo Principal:**

1. El usuario accede a la sección de perfil.
2. El frontend obtiene los datos del perfil (/api/v1/profile).
3. El sistema muestra:
   * Información personal
   * Estadísticas de uso
   * Historial de interacciones
4. El usuario puede:
   * Ver detalles
   * Exportar datos
   * Gestionar preferencias

**Flujos Alternativos:**

* 2a. Error al obtener perfil: El sistema muestra un error.
* 2b. Token expirado: El sistema redirige a la página de login.

**7. Seguridad**

**7.1 Verificación de Token**

**Actores:** Sistema  
**Precondiciones:** Usuario con sesión activa

**Flujo Principal:**

1. El frontend realiza una petición autenticada.
2. El sistema verifica el token (/api/v1/verify-token).
3. El backend:
   * Valida la firma
   * Verifica la expiración
4. Si el token es válido:
   * La operación continua
5. Si el token no es válido:
   * El sistema redirige al login

**Flujos Alternativos:**

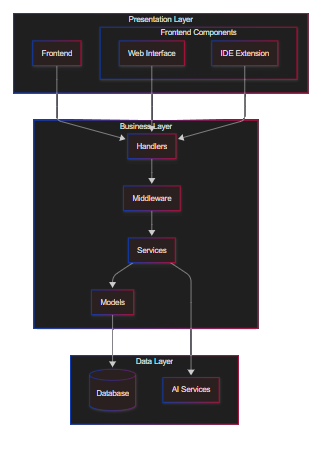
* 3a. Token expirado: El sistema solicita renovación.
* 3b. Token inválido: El sistema cierra la sesión.

4

Diseño

## 4. Diseño del Sistema

### 4.1 Arquitectura General

El sistema sigue una arquitectura de microservicios con los siguientes componentes principales:

1. Backend API:
   * Servidor Go
   * Base de datos PostgreSQL
   * Servicios de autenticación
   * Servicios de IA
2. Frontend Web:
   * Aplicación React-router v7
   * Sistema de autenticación
   * Sesión segura por medio de JWT
   * Panel de administración
3. Extensión IDE:
   * Plugin VS Code
   * Integración con backend
   * Interfaz de usuario

### 4.2 Diagrama de Componentes

### 4.3 Patrones de Diseño

1. Repository Pattern:
   * Separación de acceso a datos
   * Abstracción de operaciones de base de datos
2. Service Pattern:
   * Lógica de negocio centralizada
   * Reutilización de código
3. Factory Pattern:
   * Creación de servicios IA
   * Gestión de proveedores

[PLACEHOLDER: Incluir más detalles sobre patrones de diseño]

## 5. Base de Datos

### 5.1 Modelo de Datos

El sistema utiliza las siguientes tablas principales:

1. Users:
   * Campos: id, first\_name, last\_name, email, username, password\_hash, role
   * Relaciones: Queries (one-to-many)
2. Queries:
   * Campos: id, query, provider\_name, level, response, feedback
   * Relaciones: Users (many-to-one)
3. Settings:
   * Campos: id, service, config, updated\_at
   * Configuración de servicios IA

### 5.2 Esquema de Base de Datos

CREATE TABLE users (  
 id SERIAL PRIMARY KEY,  
 first\_name VARCHAR(255) NOT NULL,  
 last\_name VARCHAR(255) NOT NULL,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 username VARCHAR(255) UNIQUE NOT NULL,  
 password\_hash VARCHAR(255) NOT NULL,  
 role VARCHAR(50) NOT NULL DEFAULT 'user',  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP  
);  
  
CREATE TABLE queries (  
 id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),  
 query TEXT NOT NULL,  
 provider\_name VARCHAR(50) NOT NULL,  
 level VARCHAR(10) NOT NULL,  
 response TEXT NOT NULL,  
 feedback VARCHAR(10),  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
CREATE TABLE settings (  
 id SERIAL PRIMARY KEY,  
 service VARCHAR(50) NOT NULL,  
 config JSONB NOT NULL,  
 updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  
 UNIQUE(service)  
);

### 5.3 Optimización y Rendimiento

[PLACEHOLDER: Incluir detalles sobre: - Índices utilizados - Estrategias de optimización - Consideraciones de rendimiento]

## 6. Backend

### 6.1 Estructura del Proyecto

El backend está organizado en los siguientes directorios:

backend/  
├── cmd/  
│ └── server/  
│ └── main.go  
├── internal/  
│ ├── handlers/  
│ ├── models/  
│ └── services/  
├── config/  
└── docs/

### 6.2 Servicios Principales

1. AuthService:
   * Gestión de autenticación
   * Generación de tokens JWT
   * Validación de credenciales
2. AIService:
   * Integración con proveedores IA
   * Procesamiento de consultas
   * Gestión de respuestas
3. SettingsService:
   * Configuración de IA
   * Gestión de prompts
   * Almacenamiento seguro de claves

### 6.3 API REST

Endpoints principales:

1. Autenticación:
   * POST /api/v1/register
   * POST /api/v1/login
   * GET /api/v1/profile
2. Consultas IA:
   * POST /api/v1/query
   * POST /api/v1/analyze
   * POST /api/v1/feedback
3. Configuración:
   * GET /api/v1/settings
   * PUT /api/v1/settings

[PLACEHOLDER: Incluir documentación completa de la API]

## 7. Frontend

### 7.1 Aplicación Web

La interfaz web está desarrollada con React y TypeScript, utilizando:

1. Componentes principales:
   * Sistema de autenticación
   * Panel de administración
   * Interfaz de consultas
2. Estilos:
   * Tailwind CSS
   * Diseño responsive
   * Tema claro/oscuro

### 7.2 Extensión VS Code

La extensión proporciona:

1. Funcionalidades:
   * Análisis de código en tiempo real
   * Consultas contextuales
   * Sugerencias de mejora
2. Integración:
   * API del editor
   * Comunicación con backend
   * Persistencia de datos

[PLACEHOLDER: Incluir más detalles sobre la implementación]

## 8. Integración con IA

### 8.1 Configuración de Modelos

El sistema soporta múltiples proveedores de IA:

1. Groq:
   * Modelo: llama-3.3-70b-versatile
   * Configuración personalizable
   * Gestión de API keys
2. GPT:
   * Modelos disponibles
   * Configuración específica
   * Gestión de tokens

### 8.2 Procesamiento de Consultas

Flujo de procesamiento:

1. Recepción de consulta
2. Análisis de contexto
3. Selección de modelo
4. Generación de respuesta
5. Formateo y entrega

[PLACEHOLDER: Incluir detalles sobre optimización de respuestas]

## 9. Seguridad

### 9.1 Autenticación y Autorización

1. JWT:
   * Generación de tokens
   * Validación de firmas
   * Gestión de expiración
2. Roles:
   * Administrador
   * Usuario regular
   * Permisos específicos

### 9.2 Protección de Datos

1. Encriptación:
   * Claves API
   * Contraseñas
   * Datos sensibles
2. Validación:
   * Entrada de usuario
   * Parámetros de API
   * Configuración

[PLACEHOLDER: Incluir más detalles sobre medidas de seguridad]

## 10. Pruebas

### 10.1 Estrategia de Pruebas

1. Pruebas unitarias:
   * Servicios backend
   * Componentes frontend
   * Utilidades
2. Pruebas de integración:
   * API endpoints
   * Comunicación entre servicios
   * Flujos completos

### 10.2 Casos de Prueba

[PLACEHOLDER: Incluir casos de prueba detallados]

## 11. Despliegue

### 11.1 Configuración del Entorno

1. Requisitos:
   * Docker
   * Docker Compose
   * Variables de entorno
2. Configuración:
   * Base de datos
   * Servicios backend
   * Frontend

### 11.2 Docker y Contenedores

version: '3.8'  
services:  
 db:  
 image: postgres:latest  
 environment:  
 POSTGRES\_PASSWORD: ${DB\_PASSWORD}  
 volumes:  
 - postgres\_data:/var/lib/postgresql/data  
  
 backend:  
 build: ./backend  
 environment:  
 DB\_HOST: db  
 DB\_PASSWORD: ${DB\_PASSWORD}  
 depends\_on:  
 - db  
  
 frontend:  
 build: ./frontend/webpage  
 ports:  
 - "3000:3000"

[PLACEHOLDER: Incluir más detalles sobre despliegue en producción]

## 12. Conclusiones y Trabajo Futuro

### 12.1 Logros Alcanzados

1. Implementación exitosa de:
   * Sistema de autenticación
   * Integración con IA
   * Interfaz de usuario
2. Características destacables:
   * Análisis de código en tiempo real
   * Respuestas personalizadas
   * Interfaz intuitiva

### 12.2 Limitaciones Identificadas

[PLACEHOLDER: Discutir las limitaciones encontradas durante el desarrollo]

### 12.3 Trabajo Futuro

1. Mejoras planificadas:
   * Soporte para más lenguajes
   * Mejora de la precisión de IA
   * Optimización de rendimiento
2. Nuevas características:
   * Colaboración en tiempo real
   * Análisis de código más avanzado
   * Integración con más IDEs

[PLACEHOLDER: Incluir más propuestas de mejora]

## Referencias

[PLACEHOLDER: Incluir todas las referencias utilizadas en el documento]

## Anexos

### A. Diagramas UML

[PLACEHOLDER: Incluir diagramas UML completos]

### B. Manual de Usuario

[PLACEHOLDER: Incluir guía de usuario detallada]

### C. Manual de Desarrollador

[PLACEHOLDER: Incluir guía de desarrollo]

### D. Documentación de la API

[PLACEHOLDER: Incluir documentación completa de la API]

## 11. Bibliografia

1. Estimaciones del crecimiento del mercado Global de la IA en el sector educativo.

* **Open2Study** – Datos estimados del tamaño del mercado entre 2023 y 2028.
* **Research and Markets** – Proyecciones para 2029 y 2032.
* **Grand View Research** – Estimación para 2030.
* **IMARC Group** – Datos proyectados para 2033.
* **Precedence Research** – Proyección del mercado para 2034

1. [Go (Golang)](https://www.digitalocean.com/community/tutorial-series/how-to-code-in-go)