TECNOLÓGICO NACIONAL DE MEXICO
CAMPUS ENSENADA

INGENIERÍA DE SOFTWARE

MANUAL TÉCNICO

ALUMNOS:

Alonso Hernandez Devorah 22760237

Angulo Martinez Angel Gabriel 22760239

Castillo Escareño Coral 22760231

Gaynor Zurita Cindy Geraldine 22760927

Zerega Navarro Eduardo Isidro 22760601

6SF

DOCENTE:

Luis Armando Cárdenas Florido

Ensenada, Baja California a 13 de febrero de 2025.

ÍNDICE

1. INTRODUCCIÓN

RespondITE es un sistema de chat inteligente diseñado como un “blank canvas” adaptable a distintos dominios, en este caso personalizado para el área de Servicios Escolares dentro del ITE. El objetivo principal es apoyar a los trabajadores de dicho departamento, reduciendo la carga repetitiva de responder constantemente las mismas preguntas sobre trámites como constancias, credenciales, pagos, horarios, etc.

El sistema funciona como un asistente complementario, no como un reemplazo del personal. Utiliza una IA conectada a una base de datos Supabase con toda la información oficial del área de Servicios Escolares. Cuando un usuario realiza una consulta en la página web, el sistema accede a la base de datos y proporciona la mejor respuesta posible, utilizando un modelo IA de lenguaje natural.

Además:

- Permite registro de usuarios y almacena el historial de conversaciones (con límite) para su consulta posterior.
- Ofrece una interfaz de administración para que los responsables de Servicios Escolares gestionen la información y las cuentas.

El sistema está dirigido a dos públicos:

- Usuarios finales: Estudiantes, padres o tutores que necesiten información sobre los distintos procesos.
- Clientes (usuarios internos): Personal del área, quienes gestionan y administran el sistema.

2. ESTRUCTURA DEL PROYECTO

RespondITE se organiza en varios componentes principales que trabajan en conjunto para ofrecer un sistema de chat inteligente, integrando un frontend web, un backend con FastAPI, una base de datos Supabase, y un modelo IA (Meta-Llama-3.1-8B-Instruct) alojado en Hugging Face.

El proyecto tiene la siguiente estructura:

```
RespondITE-main/
├─── backend/                    # Lógica del servidor (FastAPI)
│   ├─── main.py                 # Punto de entrada de la API REST
│   ├─── config.py               # Configuración global del servidor
│   ├─── models.py               # Definición de modelos de datos
│   ├─── textProcessor.py        # Procesamiento y análisis del texto para preguntas
│   ├─── ai/
│   │   ├─── aiEngine.py         # Motor IA, conexión a Hugging Face
│   ├─── conexion/
│   │   ├─── dataBase.py         # Conexión a Supabase
│   ├─── logic/                  # Lógica separada por módulos (admin, auth, chat, user)
│   ├─── routers/                # Endpoints expuestos por la API (admin, chat, login)
├─── cache_texts/                # Archivos TXT con la información de la base de datos
├─── database_postgresql.sql     # Script SQL de referencia (Supabase)
├─── README.md                   # Instrucciones generales del proyecto
├─── Img/                        # Imágenes usadas en el README
├─── ResponditeFront/            # Frontend web (HTML, CSS, JS)
│   ├─── html/
│   │   ├─── index.html          # Página inicial de inicio de sesión
│   │   ├─── register.html       # Pantalla de registro de cuenta
│   │   ├─── ChatUser.html       # Pantalla principal del chat IA
│   │   ├─── VentanaAdmin.html   # Interfaz administrativa
│   ├─── images/                 # Recursos visuales del frontend
│   ├─── src/components/         # Componentes JS (admin.js, chat.js, login.js)
│   ├─── src/services/           # Servicios JS para interactuar con la API y Supabase
└─── .env                        # Variables de entorno con claves para Hugging Face y Supabase
```

DESCRIPCIÓN DETALLADA

2.1. BACKEND

- Se encarga de recibir preguntas desde el frontend y procesarlas mediante “textProcessor.py” y “aiEngine.py”, conectándose a Supabase para obtener datos actualizados, devolviendo la mejor respuesta generada por la IA.
- Los archivos “routers/” definen los endpoints (chat, login, admin) que son consumidos por los servicios JS del frontend.
- “conexion/dataBase.py” gestiona la conexión con Supabase usando claves definidas en “.env”.
- El archivo “main.py” es el punto de entrada del backend, ejecutando con “uvicorn”.

2.2. FRONTEND

- La página inicial “index.html” valida la cuenta ya existente en Supabase mediante servicios JS.
- “register.html”permite a los usuarios registrarse, creando nuevas cuentas en Supabase.
- Una vez autenticado, se accede a “ChatUser.html”, donde se despliega el chat y pueden interactuar con el chat inteligente.
- La interfaz administrativa está en “VentanaAdmin.html”, permitiendo a los usuarios internos gestionar cuentas e información.
- La comunicación con el backend se realiza a través de servicios JS definidos en “src/services/”, que interactúan con la API REST.

2.3. BASE DE DATOS

- Almacena toda la información oficial del área de Servicios Escolares, incluyendo datos de usuarios e historial de chats.
- La base de datos se accede mediante claves “SUPABASE_URL” y “SUPABASE_KEY” definidas en “.env”, gestionadas desde el backend.

2.4. IA

- Utiliza el modelo “Meta-Llama-3.1-8B-Instruct”, configurado mediante “API_URL” y “API_KEY” en el archivo “.env”.
- Las preguntas se procesan en “aiEngine.py”, que invoca funciones de “textProcessor.py” para:
    - “get_knowledge_data”: Obtiene los datos de conocimiento desde Supabase.
    - “get_text_from_url”: Obtiene y procesa textos desde URLs asociadas.
- El resultado se almacena temporalmente y es usado por la IA para generar respuestas.

3. REQUISITOS DEL SISTEMA

El proyecto RespondITE requiere una combinación de herramientas y tecnologías para su correcto funcionamiento. A continuación se detallan los requisitos de sistema y software necesarios para ejecutar tanto el backend, como el frontend.

3.1. REQUISITOS GENERALES

- LENGUAJE Y FRAMEWORKS
    - Python 3.10 o superior: Lenguaje principal para el desarrollo del backend con FastAPI.
    - FastAPI: Framework para construir APIs RESTful.
    - Uvicorn: Servidor ASGI necesario para ejecutar el backend.
    - HTML5, CSS3, JavaScript: Tecnologías empleadas en el frontend.

- HERRAMIENTAS ADICIONALES
    - pip: Gestor de paquetes para instalar dependencias en Python.
    - Navegador web compatible: Firefox, Google Chrome, Microsoft Edge, etc.

3.2. DEPENDENCIAS DEL BACKEND

El backend necesita instalar las siguientes bibliotecas, que se pueden agregar usando “pip”:

```
pip install fastapi uvicorn bcrypt==3.2.2 passlib python-multipart python-jose[cryptography]
```

3.3. VARIABLES DE ENTORNO (.env)

El archivo “.env” debe crearse en la raíz del proyecto para almacenar las claves y configuraciones necesarias:

- IA (Hugging Face)
    - API_URL: URL del modelo “Meta-Llama-3.1-8B-Instruct” en Hugging Face.
    - API_KEY: Clave de acceso al modelo en Hugging Face.

- BASE DE DATOS (Supabase)
    - SUPABASE_URL: URL del proyecto Supabase.
    - SUPABASE_KEY: Clave de acceso a la base de datos Supabase.
    - SUPABASE_SERVICE_ROLE_KEY: Clave con permisos de servicio para operaciones administrativas.

- SEGURIDAD
    - SECRET_KEY: Clave para generación de tokens JWT.
    - ALGORITHM: Algoritmo para firma de JWT.
    - ACCESS_TOKEN_EXPIRE_MINUTES: Tiempo de expiración del token de acceso.

3.4. CONFIGURACIÓN RECOMENDADA

- Archivo “.env” bien configurado y protegido, con las claves mencionadas.
- Backend corriendo un entorno local con Uvicorn, ejecutando con:
  `uvicorn backend.main:app --reload`
- Frontend cargado a través de “index.html”, asegurando que el backend esté activo para poder autenticarse y usar el sistema.

4. INSTALACIÓN Y CONFIGURACIÓN

El proceso de instalación y configuración se realiza en varios pasos para preparar tanto el backend como el frontend.

4.1. CLONAR O DESCARGAR EL PROYECTO

Para obtener el proyecto, puede optar por clonarlo directamente desde el repositorio o descargarlo como archivo ZIP.

- Clonación del repositorio.
  Recomendable si se desea mantener una conexión con el repositorio para futuras actualizaciones. Para ello, abra una terminal en su equipo y ejecute el siguiente comando:
  git clone https://github.com/debby019/RespondITE.git
- Descarga en formato ZIP.
  También se puede descargar el proyecto en un archivo comprimido. Para ello, acceda al siguiente enlace:
  https://github.com/debby019/RespondITE
  Luego, haga clic en el botón "Code" y seleccione la opción "Download ZIP":

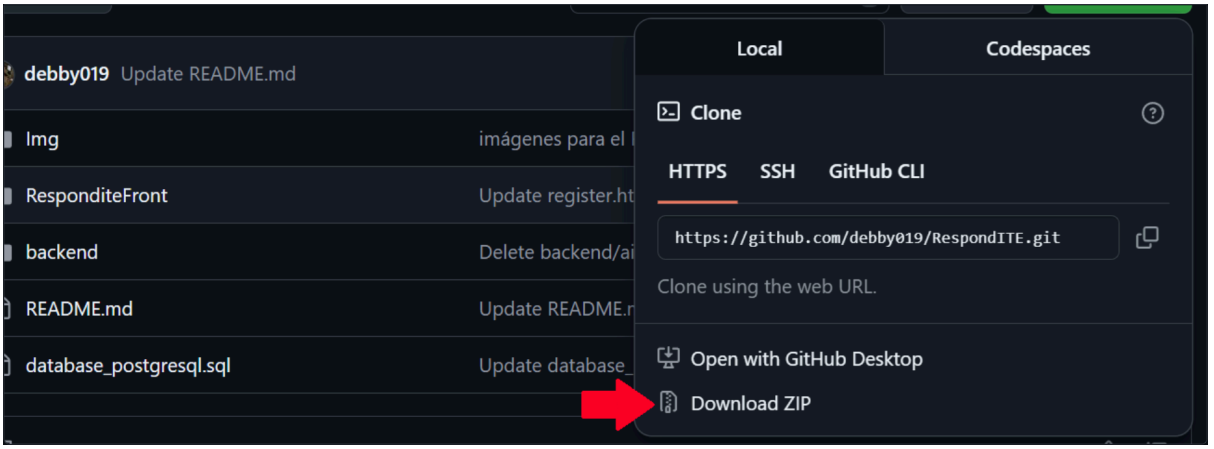

Ilustración 1. Descarga del proyecto.

Una vez descargado o clonado el proyecto, descomprima (si es ZIP) y acceda a la carpeta principal para continuar con la configuración del entorno.

4.2. CONFIGURAR EL ARCHIVO .env

Crear un archivo ".env" en la raíz del proyecto, incluyendo las siguientes variables:

- API_URL, API_KEY (Hugging Face).
- SUPABASE_URL, SUPABASE_KEY, SUPABASE_SERVICE_ROLE_KEY.
- SECRET_KEY, ALGORITHM, ACCESS_TOKEN_EXPIRE_MINUTES.

4.3. INSTALAR DEPENDENCIAS DEL BACKEND

Para garantizar el correcto funcionamiento del programa, es necesario instalar una serie de dependencias dentro de la carpeta del backend mediante la terminal. A continuación, se presenta cada una de ellas junto con su utilidad y el comando correspondiente para su instalación:

- **fastapi:** Framework principal para construir la API web de forma rápida, moderna y eficiente.

  pip install fastapi

- **supabase:** Cliente oficial de Supabase para Python, permite interactuar con la base de datos y el almacenamiento en la nube.

  pip install supabase

- **uvicorn:** Servidor ASGI ligero y rápido, utilizado para ejecutar la aplicación FastAPI.

  pip install uvicorn

- **bcrypt==3.2.2**: Librería para el cifrado seguro de contraseñas utilizando el algoritmo bcrypt.

  pip install bcrypt==3.2.2

- **passlib:** Proporciona funciones para el manejo y verificación de contraseñas cifradas, compatible con bcrypt.

  pip install passlib

- **python-multipart:** Necesaria para manejar formularios que incluyen archivos (por ejemplo, al subir archivos a la API).

  pip install python-multipart

- **python-jose[cryptography]:** Permite generar, firmar y verificar tokens JWT, esenciales para la autenticación de usuarios.

  pip install python-jose[cryptography]

- **python-dotenv:** Carga variables de entorno desde archivos .env, facilitando la configuración segura del entorno.

  pip install python-dotenv

- **pydantic:** Valida y define los modelos de datos usados en las solicitudes y respuestas del API.

  pip install pydantic

- **requests:** Permite realizar peticiones HTTP desde el backend, útil para obtener datos externos o enviar solicitudes a otras APIs.

  pip install requests

4.4. EJECUTAR EL SERVIDOR BACKEND

En la terminal desde la carpeta principal del proyecto ejecuta el siguiente comando para ejecutar el servidor:

uvicorn backend.main:app --reload

Ejemplo:

```
PS C:\Users\Angel\Documents\ITE\6SF\AdminBD\RespondITE-main> uvicorn backend.main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\Angel\\Documents\\ITE\\6SF\\
AdminBD\\RespondITE-main']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [23420] using StatReload
DEBUG: ACCESS_TOKEN_EXPIRE_MINUTES = 120
INFO:     Started server process [368]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Ilustración 2. Ejecución del servidor.

4.5. ACCEDER AL FRONTEND

Abrir el archivo “index.html” en un navegador.

Iniciar sesión con una cuenta válida (almacenada en Supabase) para interactuar con el chat.

5. ARQUITECTURA DEL SISTEMA

RespondITE se organiza en una arquitectura modular compuesta por:

5.1. FLUJO DEL SISTEMA

Usuarios interactúan a través del frontend envían consultas al backend (API REST) el backend accede a Supabase y al modelo IA se genera la respuesta y se envía de vuelta al usuario.

5.2. COMPONENTES CLAVE

- Frontend: Interfaz web en HTML, CSS, JS, con servicios JS para llamadas a la API.
- Backend: FastAPI con módulos separados (auth, chat, admin, user), lógica para acceder a IA y base de datos.
- IA: Modelo Meta-Llama-3.1-8B-Instruct alojado en Hugging Face, configurado mediante ".env".
- Base de datos: Supabase, basada en la nube, almacena usuarios, historial de chats, y toda la información.

5.3. DIAGRAMAS

- Diagrama de arquitectura general del sistema.
  Este diagrama muestra la arquitectura general del sistema y cómo interactúan sus componentes principales:

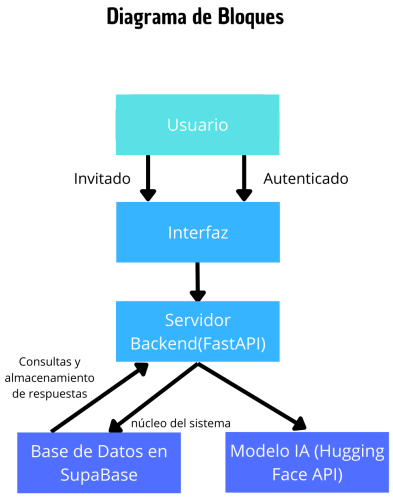

Ilustración 3. Diagrama de bloques de la arquitectura del sistema.

Usuario: Puede acceder como invitado o como usuario autenticado.
Interfaz: Es el puente visual que conecta al usuario con el sistema, permitiéndole realizar funciones como ingresar mensajes y visualizar respuestas.
Servidor Backend (FastAPI): Se encarga de recibir y procesar las solicitudes desde la interfaz. Es el núcleo lógico del sistema.
Modelo IA (Hugging Face API): Se encarga de interpretar y generar respuestas a los mensajes del usuario.
Base de Datos en Supabase: Almacena las respuestas, registros de usuarios y demás datos necesarios para el funcionamiento del sistema.

- Diagrama entidad-relación (ERD) de la base de datos.
  Este diagrama representa la estructura de la base de datos que gestiona las interacciones entre usuarios, chats, solicitudes de ayuda y la base de conocimiento que consulta la inteligencia artificial (IA).

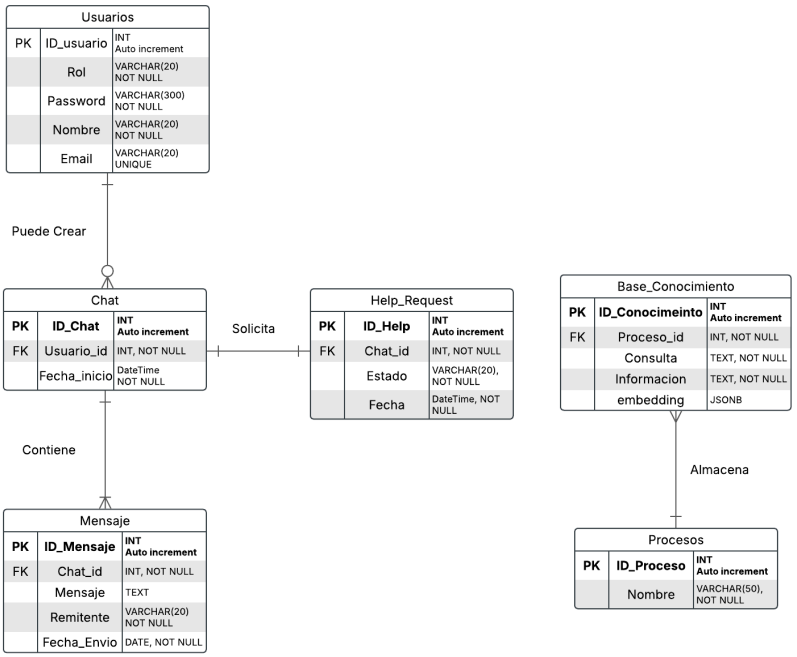

Ilustración 4. Diagrama ERD de la base de datos.

Entidades Principales:

- **Usuarios:** Contiene información de los usuarios registrados.
- **Chat:** Cada usuario puede iniciar un chat que contiene mensajes y puede estar asociado a una solicitud de ayuda.
- **Mensaje:** Representa cada mensaje enviado dentro de un chat. Incluye información del remitente, el contenido del mensaje y la fecha de envío.
- **Help_Request**: Contiene las solicitudes de ayuda generadas por los estudiantes.
- **Procesos:** Contiene los nombres de los diferentes procesos escolares (por ejemplo, credencialización, reinscripción, etc.) para los cuales se registran las consultas.
- **Base_Conocimiento:** Almacena consultas y sus respuestas (Información). Aunque por el momento no está en uso, cuenta con soporte para embeddings semánticos (embedding). Esta tabla está asociada con la tabla Procesos, que clasifica las consultas por proceso administrativo.

- Diagrama de casos de uso.

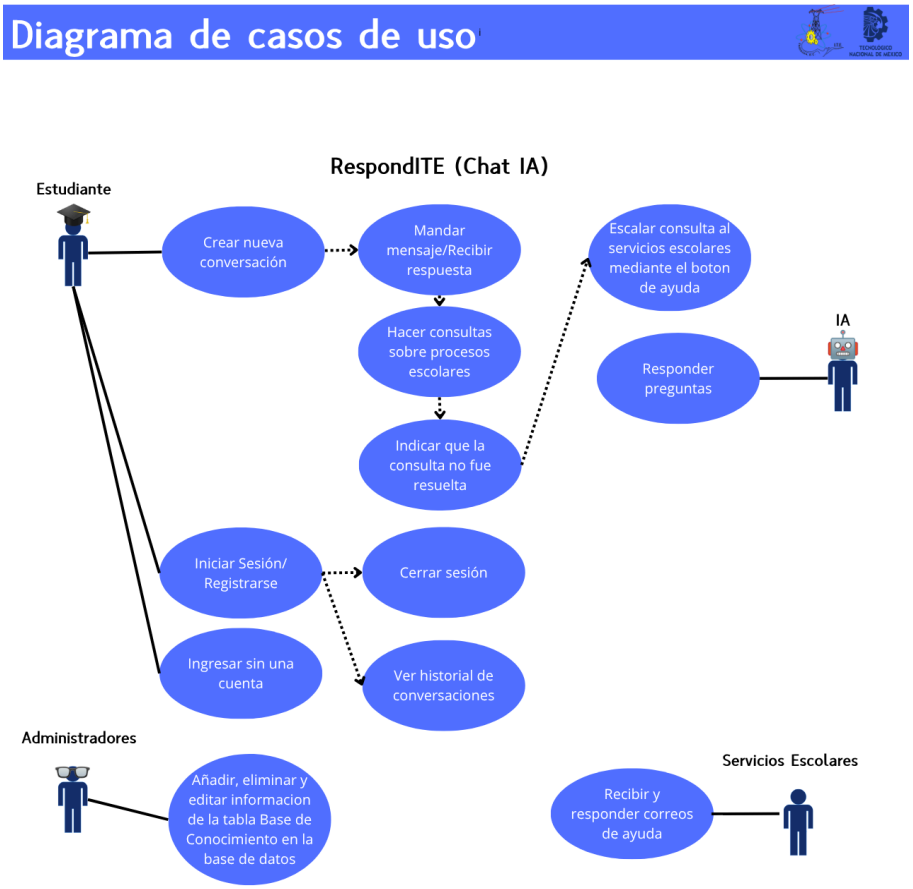

Ilustración 5. Diagrama de casos de uso.

Este diagrama de casos de casos de uso incluye:

Estudiantes: Pueden iniciar sesión, crear conversaciones, hacer consultas escolares, ver historial y mandar correo a servicios escolares en caso de ser requerido.

Asistente (IA) : Responde automáticamente las preguntas de los estudiantes.

Administradores: Editan, agregan o eliminan información en la base de conocimientos del sistema.

Servicios Escolares: Atienden las consultas que no puede resolver la IA a través de correos de ayuda.

6. DETALLE TÉCNICO DEL CÓDIGO Y MÓDULOS

6.1. MÓDULOS PRINCIPALES

- aiEngine.py: Configura la conexión con Hugging Face y procesa el historial del chat para generar respuestas usando el modelo de IA.
- textProcessor.py: Descarga textos desde la base de datos, organizando la información y alimentando a la IA.
- routers/: Define los endpoints REST, incluyendo “/chat”, “/login”, “/register” y “/chats/nuevo”.
- logic/: Contiene la lógica separada para autenticación, chat, usuarios y administradores.
- conexion/dataBase.py: Gestiona conexión a Supabase usando las claves de “.env”.

6.2. BUENAS PRÁCTICAS

- Comentarios claros y precisos en el código.
- Convenciones de nombres descriptivos para funciones y variables.
- Separación de lógica para mayor mantenibilidad.

7. DOCUMENTACIÓN DE LA API

A continuación se presenta una descripción de los principales endpoints disponibles en la API de RespondITE. Estos permiten la autenticación de usuarios, el manejo de sesiones de chat, y la gestión de procesos administrativos.

MÉTODO	ENDPOINT	DESCRIPCIÓN
POST	/login	Autenticación del usuario
POST	/register	Registro de nuevo usuario
POST	/chat	Enviar consulta al chat y obtener respuesta
POST	/chats/nuevo	Crear nuevo chat
POST	/admin/procesos	Crear un nuevo proceso con sus consultas e información correspondiente.
POST	/chats,{usuario_id}	Obtener todos los chats de un usuario.
GET	/chats/mensajes	Obtener los mensajes de un chat en específico.
GET	/Admin/procesos	Obtener los procesos e información correspondiente.

Tabla 1. Endpoints Principales.

7.1. EJEMPLO DE USO

POST /chat

```
{
        “chat_id”: chat_id,
        “user_input”: “¿Cómo solicito una credencial?”,
        “tone”: “neutral”
}
```

RESPUESTA

```
{
        “respuesta”: “Para solicitar una credencial…”
}
```

8. DESPLIEGUE Y MANTENIMIENTO

8.1. EJECUCIÓN LOCAL

- Backend con Uvicorn: “uvicorn backend.main:app --reload”.
- Frontend accedido vía “index.html”.

8.2. ESCALABILIDAD FUTURA

- Opcional contenerización con Docker.
- Uso de Nginx como proxy inverso para producción.

8.3. MANTENIMIENTO

- Respaldos automáticos de Supabase.
- Monitoreo de logs del backend para detectar errores o caídas.
- Rotación de claves de acceso y tokens.

8.4. COSTOS MENSUALES.

En la siguiente tabla se muestra un desglose de los costos mensuales estimados por cada servicio utilizado en el sistema. Estos costos se aplican una vez implementado el sistema en la institución:

SERVICIOS	COSTO MENSUAL
Hugging Face Pro	$ 192.75
Supabase Pro	$ 500.00
Dominio Web	$ 40.00
Total	$ 732.75

Tabla 2. Costo Mensual del Sistema.

Hugging Face Pro: Plataforma de modelos de lenguaje usada para responder preguntas de estudiantes. Se utiliza el plan Pro por mayor capacidad de procesamiento.

Supabase Pro (Base de datos): Servicio en la nube que ofrece base de datos, autenticación y almacenamiento. Se usa por su rápida integración y soporte a tiempo real.

Dominio Web: Incluye un dominio personalizado junto con un certificado SSL para garantizar una conexión segura. Aunque en la tabla se presenta como un costo mensual estimado, este servicio generalmente se paga de forma anual, con un valor aproximado de entre $250 y $400 MXN al año.

Alternativas:

Aunque actualmente se utiliza Supabase Pro por su facilidad de uso y despliegue rápido, se considera viable en el futuro migrar a otras soluciones más económicas como PostgreSQL autogestionado o incluso utilizar servidores propios de la institución si estos están disponibles. Esta decisión podría reducir significativamente los costos mensuales y facilitar el cumplimiento de normas de seguridad.

También existe la posibilidad de utilizar un subdominio proporcionado por la institución (por ejemplo: respondi.ite.edu.mx), lo cual permitiría eliminar por completo este costo, siempre y cuando la institución lo autorice.

9. PRUEBAS Y VALIDACIÓN

9.1. PRUEBAS FUNCIONALES

9.1.1. VALIDACIÓN DE REGISTRO E INICIO DE SESIÓN

Se realizaron pruebas para validar el correcto funcionamiento de los mecanismos de autenticación y registro. A continuación se describen los principales comportamientos esperados:

Pruebas de Inicio de sesión y registro:

- Si se intenta acceder al sitio con un correo electrónico no registrado aparece el siguiente mensaje en pantalla y no permite el acceso al sitio.

Ilustración 6. Correo electrónico incorrecto.

- Si se intenta acceder al sitio con una contraseña incorrecta aparece el siguiente mensaje en pantalla y no permite el acceso al sitio.

Ilustración 7.Contraseña incorrecta.

- Si al registrarse el usuario coloca un correo electrónico invalido aparecerá el siguiente mensaje y no se realizará el proceso del registro:

Ilustración 8. Formato de correo incorrecto.

- Al intentar entrar por el url a la interfaz del chat sin haber iniciado sesión, el usuario será redirigido a la ventana de inicio (Index.html), solo se puede ingresar a las diferentes páginas del sistema iniciando sesión o por medio del botón “Acceder sin Cuenta”

9.1.2. ENVÍO Y RECEPCIÓN DE MENSAJES EN “/CHAT”

- En caso de solicitar información sobre algún tema no relacionado con la institución o que no se encuentre en la base de datos, el asistente responderá algo similar a lo que se muestra en la *Ilustración 9*, podemos ver del lado izquierdo el chat con el asistente y del lado derecho los procesos realizados por el backend:

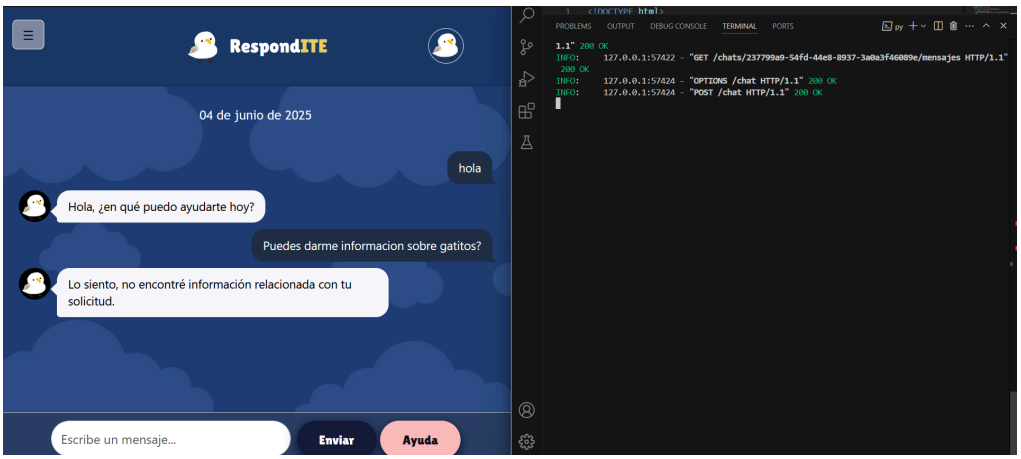

Ilustración 9. El asistente no encuentra información del tema solicitado.

9.1.3. VERIFICACIÓN DEL HISTORIAL DE CHATS

El historial de chats se visualiza correctamente desde el menú lateral izquierdo:

Ilustración 10. Historial de chats.

Entre los elementos identificados para futuras mejoras se encuentran:

- Evitar la creación de chats vacíos (aquellos que no contienen mensajes).
- Mejorar la presentación visual del historial para una navegación más intuitiva.

9.2. HERRAMIENTAS USADAS

- Navegador para pruebas del frontend.
- Validación visual y funcional directa sobre el frontend.

9.3. RESULTADOS ESPERADOS

- Respuestas precisas basadas en información oficial.
- Tiempos de respuesta adecuados.
- Integridad y seguridad de las transacciones.

10. RECOMENDACIONES PARA NUEVOS DESARROLLADORES

10.1. RECOMENDACIONES

- Revisar “main.py”, “routers/”, “aiEngine.py” y “textProcessor.py” para entender el flujo completo.
- Utilizar “print()” para depuración.
- Respetar convenciones y buenas prácticas para facilitar la mantenibilidad.

10.2. DIAGRAMAS

- Arquitectura general (usuarios frontend backend IA/Supabase).

Ilustración 11. Arquitectura general.

- Flujo del proceso (desde una consulta hasta una respuesta):
  Este diagrama representa el flujo lógico que sigue el usuario desde que inicia sesión hasta que recibe una respuesta del sistema:

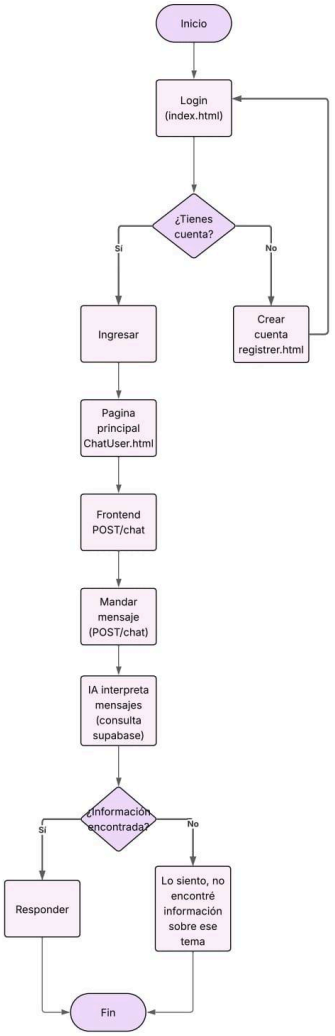

Ilustración 12. Flujo del proceso al enviar un mensaje y una respuesta.

1. El proceso inicia con la pantalla de inicio de sesión.
2. Si el usuario no posee una cuenta, se le redirige al formulario de registro.
3. Una vez autenticado, accede a la interfaz principal, desde donde puede enviar mensajes.
4. Estas solicitudes son procesadas por el backend, que interpreta el contenido y realiza consultas en la base de datos (Supabase).
5. Si se encuentra información relevante, se genera una respuesta; de lo contrario, se notifica al usuario que no se hallaron resultados relacionados.