Documentación Proyecto 1: RAG

Historias de Usuario

1. **Registro de Usuario:**
    - Como nuevo usuario, quiero poder registrarme en la aplicación para poder acceder a sus funcionalidades.
    - **Criterios de Aceptación**:
        1. El usuario debe poder acceder al formulario de registro en la vista de registro.
        2. El formulario debe solicitar un nombre de usuario, correo electrónico y contraseña.
        3. Al enviar el formulario, se debe realizar una solicitud al endpoint /accounts.
        4. Si el registro es exitoso, el usuario debe ser redirigido al inicio de sesión.
        5. Si ocurre un error durante el registro, el usuario debe recibir un mensaje.
2. **Inicio de Sesión:**
    - Como usuario registrado, quiero poder iniciar sesión en la aplicación para acceder a mis archivos y chats.
    - **Criterios de Aceptación**:
        1. El usuario debe poder acceder a un formulario de inicio de sesión.
        2. El formulario debe solicitar el correo electrónico y la contraseña del usuario.
        3. Al enviar el formulario, se debe realizar una solicitud al endpoint /accounts/login.
        4. Si las credenciales son correctas, el token de acceso debe almacenarse en sessionStorage.
        5. El usuario es redirigido a la vista principal una vez se inicie la sesión con éxito.
        6. Si ocurre un error durante el inicio de sesión, el usuario debe recibir un mensaje.
3. **Carga de documentos:**
    - Como usuario autenticado, quiero subir documentos en varios formatos (PDF, TXT, MD, DOCX) para utilizarlos en conversaciones contextuales con un LLM
    - **Criterios de aceptación:**
        1. Debo poder cargar documentos en formatos PDF, TCT, MD, DOCX.
        2. Debo poder arrastrar y soltar archivos en el área de subida, así mismo, seleccionar un archivo desde el sistema de archivos local.
        3. Una vez el archivo sea cargado con éxito, este aparecerá en la lista.
        4. El sistema debe poder extraer el texto del documento según el formato.
        5. El sistema debe iniciar automáticamente el proceso de indexación del documento.

6. Debo recibir errores si hay problemas con el formato o el contenido del archivo.

4. **Gestion de Documentos:**
    - Como usuario autenticado quiero ver la lista de documentos que he subido para poder gestionar los recursos.
    - **Criterios de aceptación:**
        1. Debo ver una lista de todos mis documentos cargados.
        2. Para cada documento debo poder ver su fecha de creación.
        3. Debo poder eliminar cualquier documento que haya subido.
        4. Al eliminar un documento, este debe desaparecer de la lista mencionada anteriormente.
5. **Creación de Conversaciones:**
    - Como usuario autenticado, quiero iniciar nuevas conversaciones para interactuar con mis documentos mediante un asistente basado en LLMs.
    - **Criterios de aceptación:**
        1. Debo poder crear una nueva conversación vacía.
        2. La nueva conversación debe estar asociada a mi cuenta de usuario.
        3. La nueva conversación debe aparecer en mi lista de conversaciones.
        4. Debo poder eliminar una conversación cuando lo requiera.
        5. Al eliminar la conversación esta debe desaparecer de la lista.
6. **Interacción con Asistente IA:**
    - Como usuario autenticado quiero enviar mensajes en una conversación para obtener respuestas contextuales basadas en mis documentos.
    - **Criterios de aceptación:**
        1. Debo poder enviar mensajes de texto en una conversación existente.
        2. El sistema debe procesar mi mensaje y generar una respuesta.
        3. La respuesta debe basarse en información contenida en mis documentos.
        4. Tanto mi mensaje como la respuesta del RAG deben quedar registrados en el historial de la conversación.

Descripción General

Esta solución proporciona un servicio de Generación Aumentada por Recuperación (RAG) completo que integra:

1. Una API backend en Go para gestión de usuarios, documentos y chats
2. Servicios de procesamiento asíncrono en Python para:
    - Indexación y embeddings de documentos
    - Generación de respuestas con IA basadas en el contexto documental

El sistema utiliza una arquitectura de microservicios con:

- PostgreSQL para almacenamiento de datos
- RabbitMQ para procesamiento asíncrono de mensajes
- Sentence Transformers para generación de embeddings vectoriales
- Ollama como motor LLM para generación de respuestas

Arquitectura del Sistema

Componentes Principales

1. **API Backend (Go)**
    - Gestión de cuentas y autenticación
    - Subida y almacenamiento de documentos
    - Sistema de chat y mensajería
    - Enrutamiento de solicitudes a colas de procesamiento
2. **Servicio de Indexación de Documentos (Python)**
    - Procesamiento de texto de documentos
    - Segmentación (chunking) de documentos largos
    - Generación de embeddings vectoriales
    - Almacenamiento de embeddings en la base de datos
3. **Servicio de Asistente IA (Python)**
    - Recuperación de documentos relevantes
    - Generación de respuestas contextuales
    - Integración con modelos LLM (Ollama)
4. **Infraestructura**
    - Base de datos PostgreSQL (almacenamiento estructurado)
    - RabbitMQ (mensajería asíncrona entre componentes)

Flujo de Datos

1. **Procesamiento de Documentos:**
    - Usuario sube documento mediante la API Go
    - API extrae texto según formato (PDF, TXT, MD, DOCX)
    - API envía mensaje a cola `document-indexing-queue`
    - Servicio Python procesa el documento:
        - Divide el texto en chunks con solapamiento
        - Genera embeddings usando Sentence Transformers
        - Almacena embeddings en la base de datos
2. **Procesamiento de Consultas:**
    - Usuario envía mensaje en un chat mediante la API Go
    - API almacena el mensaje y envía a cola `ai-assistant-queue`
    - Servicio Python procesa la consulta:
        - Genera embedding de la consulta
        - Recupera documentos relevantes comparando similitud vectorial
        - Construye contexto a partir de documentos recuperados
        - Genera respuesta usando Ollama con el contexto
        - Almacena respuesta en la base de datos

API Backend (Go + ECHO)

Modelos de Datos

Account

```
{
"id": int32,
"createdAt": timestamp,
"username": string,
"email": string
}
```

Document (Documento)

```
{
"id": int32,
"createdAt": timestamp,
"name": string,
"text": string,
"filePath": string,
"embedding": object,
"accountId": int32
}
```

Chat

```
{
"id": int32,
"createdAt": timestamp,
"messages": array,
"accountId": int32,
"unreadMessages": boolean
}
```

Message (Mensaje)

```
{
"id": string,
"timestamp": timestamp,
"sender": string ("user", "assistant", o "system"),
"text": string
}
```

Endpoints de la API

Autenticación

Crear Cuenta de Usuario

- **Endpoint**: `POST /accounts`
- **Descripción**: Crea una nueva cuenta de usuario
- **Cuerpo de la Solicitud**:
    - `username`: Nombre de usuario

    - `email`: Email del usuario
    - `password`: Password del usuario

Login

- **Endpoint**: `POST /accounts/login`
- **Descripción**: Autentica a un usuario y devuelve un token JWT
- **Parámetros de Formulario**:
    - `username`: Nombre de usuario
    - `password`: Contraseña del usuario

Conseguir usuario

- **Endpoint**: `GET /accounts`
- **Descripción**: Consigue los datos de un usuario a partir de su token JWT
- **Autenticación**: Token JWT requerido

Conseguir documentos del usuario

- **Endpoint**: `GET /accounts/documents`
- **Descripción**: Consigue los documentos de un usuario a partir de su token JWT
- **Autenticación**: Token JWT requerido

Conseguir chats de usuario

- **Endpoint**: `GET /accounts/chats`
- **Descripción**: Consigue los chats de un usuario a partir de su token JWT
- **Autenticación**: Token JWT requerido

Gestión de Documentos

Subir Documento

- **Endpoint**: `POST /documents`
- **Descripción**: Sube un nuevo documento y activa la indexación para RAG
- **Autenticación**: Token JWT requerido
- **Parámetros de Formulario**:
    - `file`: El archivo del documento (PDF, TXT, MD, DOCX)

Eliminar Documento

- **Endpoint**: `DELETE /documents/:documentID`
- **Descripción**: Elimina un documento propiedad del usuario actual
- **Autenticación**: Token JWT requerido

Sistema de Chat

Crear Chat Vacío

- **Endpoint**: `POST /chats`
- **Descripción**: Crea un nuevo chat vacío para el usuario actual
- **Autenticación**: Token JWT requerido

Enviar Mensaje

- **Endpoint**: `POST /chats/:chatID/messages`
- **Descripción**: Añade un nuevo mensaje a un chat y activa la respuesta del asistente de IA
- **Autenticación**: Token JWT requerido
- **Cuerpo de la Solicitud:**
  - `text`: Cuerpo del mensaje
  - `sender`: “user”

Conseguir chat

- **Endpoint**: `GET /chats/:chatID`
- **Descripción**: Conseguir los datos del chat por id
- **Autenticación**: Token JWT requerido

Borrar chat

- **Endpoint**: `DELETE /chats/:chatID`
- **Descripción**: Borrar el chat con el ID correspondiente
- **Autenticación**: Token JWT requerido

Conseguir chat unread

- **Endpoint**: `GET /chats/:chatID/unread`
- **Descripción**: Conseguir si un chat tiene mensajes sin leer
- **Autenticación**: Token JWT requerido

Marcar chat leido

- **Endpoint**: `POST /chats/:chatID/mark-as-read`
- **Descripción**: Marcar un chat como leído
- **Autenticación**: Token JWT requerido

Servicios de Procesamiento (Python)

Servicio de Indexación de Documentos

El servicio de indexación de documentos se encarga de:

1. **Extracción de Texto**:
   - Recibe mensajes de la cola `document-indexing-queue`
   - Procesa el texto del documento

2. **Segmentación de Documentos**:
    - Divide documentos largos en chunks de tamaño configurable (por defecto 500 caracteres)
    - Mantiene solapamiento entre chunks (por defecto 50 caracteres)
    - Intenta dividir en límites de oraciones para preservar contexto
3. **Generación de Embeddings**:
    - Utiliza el modelo `all-MiniLM-L6-v2` de Sentence Transformers
    - Genera embeddings para cada chunk de texto
    - Calcula el embedding promedio y lo normaliza
    - Almacena el vector resultante en la base de datos

Servicio de Asistente IA

El servicio de asistente IA se encarga de:

1. **Procesamiento de Mensajes**:
    - Recibe mensajes de la cola `ai-assistant-queue`
    - Extrae la consulta del usuario del último mensaje
2. **Recuperación de Documentos**:
    - Genera embedding de la consulta del usuario
    - Busca documentos similares en la base de datos usando distancia vectorial
    - Selecciona los documentos más relevantes (top_k=3 por defecto)
3. **Generación de Respuestas**:
    - Construye un contexto con los fragmentos de texto relevantes
    - Utiliza Ollama como LLM para generar respuestas basadas en el contexto
    - Configura un prompt específico para respuestas basadas en contexto
    - Limita respuestas a la información proporcionada en el contexto
    - Almacena la respuesta generada en el chat

Frontend

Aplicación de una sola página (SPA) estructurada con Vue.js que utiliza Vue Router para el manejo de vistas y JavaScript para las integraciones con los endpoints.

Vistas:

1. **Register**:
    - **Endpoint**: `/accounts`
    - **Funcionalidad**: Permite la creación de nuevos usuarios mediante un formulario.
2. **Login**:
    - **Endpoint:** `/accounts/login`
    - **Funcionalidad:** Permite el ingreso de usuarios registrados y almacena el token de acceso en ***sessionStorage***, asegurando la permanencia del usuario mientras la sesión del navegador esté activa.
3. **HomeView**:

- **Funcionalidad:** Vista principal que permite la carga y eliminación de archivos, así como la creación y eliminación de chats nuevos cuando el usuario ha iniciado sesión

Componentes principales:

1. **FileUploader**:
   - **Endpoint**: `/documents`
   - **Funcionalidad**: Maneja la subida de nuevos archivos, permite arrastrar y soltar documentos, refresca la vista y hace un fetch al endpoint ***/accounts/documents*** para mostrar los nuevos documentos cargados. Acepta únicamente los formatos de archivo permitidos por el backend.
2. **FileReader:**
   - **Endpoint**: `/documents/:documentID`
   - **Funcionalidad**: Muestra todos los archivos creados por el usuario y permite su eliminación.
3. **ChatBox**:
   - **Endpoints**: `/chats, /accounts/chats, /chats/:chatID`
   - **Funcionalidad**: Crea nuevos chats, muestra los chats existentes y permite eliminar los que ya no son necesarios.
4. **messagesHandler:**
   - **Endpoints**: `/chats/:chatID/messages, /chats/:chatID/unread`
   - **Funcionalidad**: Maneja la comunicación a través de mensajes entre el usuario y la aplicación. Envía mensajes nuevos y hace un ping al endpoint de mensajes no leídos, esperando la respuesta de la aplicación para imprimir el mensaje.

Tecnologías y Componentes

Modelos de Embedding

El sistema utiliza el modelo `all-MiniLM-L6-v2` de Sentence Transformers para generar embeddings vectoriales de:

- Documentos (para indexación)
- Consultas del usuario (para recuperación)

Características del modelo:

- Dimensionalidad: 384
- Normalización: Embeddings normalizados para comparación de similitud coseno
- Eficiencia: Buen equilibrio entre tamaño/rendimiento para aplicaciones RAG

Motor LLM

El sistema utiliza Ollama como motor LLM para la generación de respuestas:

- **Modelo predeterminado**: `phi` (configurable mediante variables de entorno)
- **Parámetros ajustables**:
    - Temperature: 0.7 (configurable)
    - Max tokens: 2048 (configurable)

Comunicación Asíncrona

RabbitMQ gestiona la comunicación asíncrona entre componentes:

- **Colas**:
    - `document-indexing-queue`: Para procesamiento de documentos
    - `ai-assistant-queue`: Para generación de respuestas IA
- **Intercambios**:
    - `document-indexing-exchange`: Para mensajes de indexación
    - `ai-assistant-exchange`: Para mensajes de consultas

Configuración

Variables de Entorno

Base de datos:

- `DB_HOST`: Host de la base de datos
- `DB_PORT`: Puerto de la base de datos
- `DB_USER`: Usuario de la base de datos
- `DB_PASSWORD`: Contraseña de la base de datos
- `DB_NAME`: Nombre de la base de datos

RabbitMQ:

- `CLOUDAMQP_URL`: URL de conexión a RabbitMQ

Ollama:

- `OLLAMA_API_BASE`: URL base de la API de Ollama (por defecto: http://localhost:11434/api)
- `OLLAMA_MODEL`: Modelo a utilizar (por defecto: phi)
- `OLLAMA_TEMPERATURE`: Temperatura para generación (por defecto: 0.7)
- `OLLAMA_MAX_TOKENS`: Tokens máximos por respuesta (por defecto: 2048)

Logging:

- `LOG_LEVEL`: Nivel de detalle de logs (por defecto: INFO)

Seguridad

- **Autenticación**: Tokens JWT con expiración de 24 horas
- **Contraseñas**: Hasheadas con bcrypt
- **Autorización**: Middleware para verificar propiedad de recursos
- **Validación**: Verificación de propiedad para documentos y chats

Procesamiento de Documentos

El sistema soporta varios formatos de documentos:

- PDF (.pdf): Texto extraído usando la biblioteca go-fitz
- Texto (.txt): Texto plano leído directamente
- Markdown (.md): Contenido leído como texto plano
- Documentos Word (.docx): Texto extraído usando la biblioteca docx

Optimizaciones

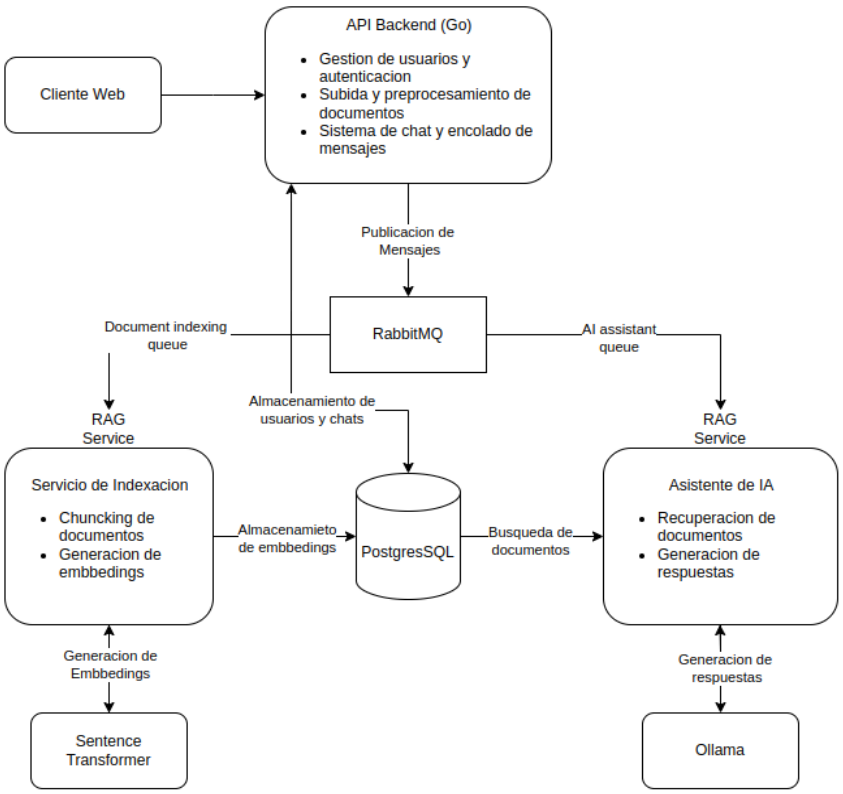
Indexación de Documentos

- División en chunks para manejar documentos largos
- Solapamiento entre chunks para preservar contexto
- Normalización de embeddings para comparación consistente

Generación de Respuestas

- Recuperación de top-k documentos más relevantes
- Contextualización del LLM con fragmentos recuperados
- Instrucción explícita para basarse solo en la información proporcionada

Arquitectura Detallada de Microservicios



1. Servicio Principal (Go)

El servicio principal en Go maneja:

- API REST para interacción con clientes
- Autenticación y gestión de usuarios
- Almacenamiento inicial de documentos y mensajes
- Publicación de mensajes a colas RabbitMQ

2. Servicio de Indexación (Python)

El servicio consume mensajes de `document-indexing-queue` y:

- Procesa el texto del documento
- Genera embeddings vectoriales con Sentence Transformers
- Actualiza el documento en la base de datos con el embedding

3. Servicio de IA (Python)

El servicio consume mensajes de `ai-assistant-queue` y:

- Genera embeddings de consultas
- Busca documentos relevantes por similitud
- Integra con Ollama para generar respuestas

- Almacena respuestas en el chat

Escalabilidad

El sistema está diseñado para escalar horizontalmente:

- Procesamiento asíncrono con RabbitMQ
- Semáforo para controlar concurrencia (`MAX_WORKERS = 12`)
- Pool de conexiones para base de datos
- Patrón Singleton para clientes y conectores