# Documento Inicial: Decisiones de Diseño y Tecnologías Seleccionadas

## Recopilación de Preguntas y Respuestas de Discord para crear una base de conocimiento

En este proyecto se busca construir una base de conocimiento a partir de mensajes intercambiados entre alumnos y docentes en Discord, exportados en archivos JSON. Para analizar y procesar estos mensajes, se eligió utilizar el lenguaje de programación Python junto con la librería pandas, que permite cargar los datos en una estructura tipo tabla llamada DataFrame.

Esta elección se debe a que pandas ofrece una manera eficiente y flexible de manipular datos estructurados, facilitando tareas como el filtrado, la limpieza o el análisis, que serían más complejas si se trabajara directamente con listas o diccionarios.

Una vez cargados los datos en el DataFrame, se aplican distintos filtros para identificar y eliminar mensajes no útiles, tales como:

* Mensajes vacíos o con solo espacios.
* Mensajes compuestos únicamente por emojis, GIFs o enlaces (por ejemplo, de Tenor o Giphy).
* Mensajes con símbolos o números sin texto (como “+1”).
* Mensajes que no contienen letras ni números (como “?”).

Cada uno de estos criterios de filtrado requiere una lógica particular. Por eso, además de aplicar los filtros, es necesario:

* Registrar qué mensajes se eliminan según cada criterio.
* Exportar los resultados para su posterior análisis.
* Contar con un sistema flexible y fácil de extender si surgen nuevos criterios.

Para resolver este problema de forma escalable, se implementó el patrón de diseño Strategy, que permite definir un conjunto de filtros encapsulados e intercambiables. Esto promueve la flexibilidad y la reutilización del código.

Diagrama

El contenido generado por IA puede ser incorrecto.

En la implementación, cada filtro está representado por una clase distinta que implementa una interfaz común llamada EstrategiaFiltro, con dos métodos:

* aplicar(texto: str) → bool: determina si el mensaje cumple el criterio de filtrado.
* nombre() → str: devuelve el nombre del filtro, útil para registrar logs y exportar resultados.

Además, se desarrolló una función principal llamada aplicar\_filtros\_mensajes\_json, que:

* Recibe el DataFrame con los mensajes.
* Recibe una lista de estrategias de filtrado.
* Aplica cada filtro en forma desacoplada, guardando por separado los mensajes eliminados por cada uno.

Ventajas del uso del patrón Strategy en este contexto:

* Desacoplamiento: cada filtro es independiente; la función principal no conoce los detalles internos de cada uno.
* Extensibilidad: se puede agregar un nuevo filtro simplemente creando una clase nueva y sumándola a la lista de estrategias.
* Trazabilidad: los mensajes filtrados se registran por separado, lo que facilita su análisis.
* Reutilización y control de orden: aunque en este proyecto no se requiere aplicar los filtros en un orden específico, el sistema permite hacerlo si fuera necesario, o reutilizar los filtros en otros contextos.

Una vez aplicados los filtros al conjunto de mensajes de un JSON, para que quede limpio, se va a crear una instancia de la clase Procesador para que se ocupe de la clasificación de mensajes.

Tabla

El contenido generado por IA puede ser incorrecto.

En Discord, se tiene un hilo de conversación, es decir, un intercambio de preguntas y respuestas, que puede ser entre un alumno y un docente o varios participantes de ambos roles. Esos mensajes pueden ser consecutivos, o en algunos casos mezclarse con otras consultas. Por este motivo se han tenido que determinar ciertas condiciones para definir:

* cuando un mensaje es pregunta,
* cuando es respuesta
* en caso de ser respuesta, si la misma es de cierre o no de la pregunta a la que se corresponde.

Entonces las situaciones que se pueden tener:

* mensaje de un docente: se considera que siempre es una respuesta. La misma se asigna a la/las pregunta que están abiertas. Si no las hay, se analiza si hay al menos una pregunta cerrada, y se asigna a ella, y si hay más de una, a las últimas dos preguntas cerradas. Y en caso de no cumplirse lo anterior, se tendrá un mensaje suelto de un docente.
  + Ejemplo de mensaje suelto de un docente: el primer mensaje del JSON que da la bienvenida a los alumnos al canal de consultas: “Este canal será utilizado para realizar consultas (TPA, temas de la asignatura, parciales, etc.). Esperamos que sea lo más colaborativo posible.

Siéntanse libres de contestar las dudas de sus compañeros y/o aportar nuevas ideas 🙂”

* mensaje de alumno: puede ser tanto una pregunta como una respuesta. Para determinar en qué categoría corresponde se analiza:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Llega un mensaje de cierto autor para procesar | | | | | | | | |
| ¿Hay preguntas abiertas? | | | | | | | | |
| SI | | | | | | NO | | |
| ¿Hay preguntas del mismo autor del mensaje? | | | | | | ¿Es pregunta? | | |
| SI | | | | NO | | SI | NO | |
| ¿Es parte de la pregunta a analizar?  (la pregunta es del mismo autor del mensaje y no tiene respuestas) | | | | ¿Es pregunta? | | El mensaje es una Nueva Pregunta | ¿Hay preguntas cerradas? | |
| NO | | | SI | SI | NO |  | SI | NO |
| ¿Es una respuesta de cierre? | | | El mensaje es la continuación de una pregunta (se concatena con ella) | El mensaje es una Nueva Pregunta | Se agrega respuesta a cada una de las que está en preguntas abiertas |  | El mensaje se asigna como respuesta a la última o las últimas dos preguntas cerradas | El mensaje del alumno queda suelto ya que no hay lista de preguntas abiertas y cerradas |
| SI | | NO |  |  |  |  |  |  |
| ¿Tiene Respuesta Validada? | | Es una respuesta a la pregunta |  |  |  |  |  |  |
| SI | NO |  |  |  |  |  |  |  |
| Se cierra pregunta, se guarda en lista de preguntas cerradas, se quita de preguntas abiertas | Es una respuesta a la pregunta |  |  |  |  |  |  |  |

La clase Procesador mantiene una relación de asociación con las clases Mensaje, Pregunta y Respuesta, ya que las utiliza para llevar a cabo el procesamiento semántico de los datos extraídos desde archivos JSON.

Aunque Pregunta y Respuesta comparten algunos atributos con Mensaje (autor, contenido, id, attachments, timestamp y origen), se decidió **no aplicar herencia**, dado que sus comportamientos son diferentes y su propósito dentro del procesamiento es específico.

Esta separación permite mantener cada clase enfocada en una única responsabilidad, de acuerdo con el principio de responsabilidad única del diseño orientado a objetos.

Una captura de pantalla de un celular con letras

El contenido generado por IA puede ser incorrecto.

Por cada JSON analizado, se va a tener una instancia de la clase Procesador. Cada una de ellas, se encarga del procesamiento de los mensajes (determinar cuál es pregunta y respuesta y cómo se corresponden) y se almacenan en una lista de procesadores. Cada instancia tiene:

* la lista de pregunta cerradas, con sus respuestas, necesarias para la persistencia de datos en la base de datos relacional que se va a implementar en postgresql.
* Lista de preguntas abiertas (en este caso no debería contener elementos).
* Contador de preguntas, cantidad de concatenaciones, cantidad de mensajes de cierre de alumnos y contador de mensajes que son respuestas. Son necesarios para la trazabilidad ya que permite determinar la cantidad total de mensajes analizados.

Luego, por cada elemento en la lista de procesadores, se va a aplicar una función que se encarga de analizar las preguntas y las respuestas.

Es decir, se encarga de determinar si:

* Si una pregunta tiene o no contexto (es decir, es una pregunta muy corta, menos de 6 palabras)
* Si una pregunta es administrativa (por ejemplo, es una pregunta que se refiere a la hora de un parcial o final o a qué equipo le tienen que corregir el trabajo práctico anual)
* Si una respuesta es o no corta ( se trata de textos que tiene menos de 14 caracteres)

Este análisis es necesario, dado a que las preguntas que sean administrativas y no tengan contexto, no se van a convertir en embeddings (en vectores).

Se procede a conectar la base de datos con Python para que puedan persistirse las preguntas y respuestas.

Diagrama

El contenido generado por IA puede ser incorrecto.

* mensajes: todos los mensajes del JSON a excepción de lo mensajes de cierre de alumnos.
* preguntas: solo los mensajes que son preguntas (relación 1:1 con mensajes)
* respuestas: solo los mensajes que son respuestas (relación 1:1 con mensajes, n:1 con preguntas)

Ventajas del modelo de datos elegido:

* Separación de preguntas y respuestas para aplicar reglas distintas, evitar ambigüedades y permitir trazabilidad:
  + las validaciones las tienen las respuestas
  + la clasificación de si es administrativa o es sin contexto es para las preguntas
  + los embeddings se construyen a partir de las preguntas
  + se puede reconstruir la conversación identificando claramente qué es pregunta y qué es respuesta
* Lograr escalabilidad y mantenibilidad
  + Si más adelante hay que agregar otra categoría para los mensajes, por ejemplo, calificaciones, preguntas de trabajo práctico, etcétera, el modelo lo permite.
  + Resulta fácil de leer y mantener para otros desarrolladores.
* **Alineación con el flujo de tu chatbot**
  + El sistema necesita saber qué es una pregunta y sus respuestas válidas para pasárselas a Mistral. Tener las entidades explícitas facilita todo el flujo de RAG.

Desventajas

* Más joins en consultas: las consultas requieren combinar tablas para recuperar contenido completo.

Luego del procesamiento de los mensajes proporcionados en los archivos JSON y la persistencia de datos en la base de datos relacional PostgreSQL, se procede a la generación de embeddings.

Para ello, se ha elegido el modelo de generación de embeddings all-MiniLM-L6-v2, disponible a través de la librería HuggingFaceEmbeddings, para la generación de representaciones vectoriales (embeddings) de preguntas. Este modelo fue seleccionado luego de un análisis comparativo que consideró precisión y velocidad y uso:

|  |  |  |  |
| --- | --- | --- | --- |
| **Modelo** | **Precisión semántica** | **Velocidad** | **Notas** |
| all-MiniLM-L6-v2 | Muy buena | Muy rápida | Búsquedas semánticas eficientes |
| all-MiniLM-L12-v2 | Mejor que L6 | Más lenta | Más precisión, más costo |
| all-mpnet-base-v2 | Excelente | Más lenta | Alta precisión semántica |
| multi-qa-MiniLM-L6-cos-v1 | Buena | Rápida | QA, recuperación de preguntas |

Razones para elegir all-MiniLM-L6-v2:

1. Balance óptimo entre velocidad y precisión: ideal para aplicaciones que necesitan buena performance en entornos sin GPU o con recursos limitados.
2. Buen rendimiento semántico: ha demostrado obtener muy buenos resultados en tareas de búsqueda de preguntas similares.
3. Adecuado para producción: se puede escalar sin costos altos de cómputo.

Una vez que se han generado los embeddings, se pueden hacer pruebas de búsqueda semántica utilizando Python. Para eso se necesita determinar la pregunta con la que se van a realizar las pruebas y un parámetro que indique la cantidad de resultados similares de la búsqueda semántica. No existe un "valor óptimo universal", pero los más comunes son entre 2 y 5. Notar que si es muy bajo, se corre el riesgo de **omitir resultados relevantes**, y si es muy alto, puede traer resultados **irrelevantes o ambiguos**.