DocSupport

Índice

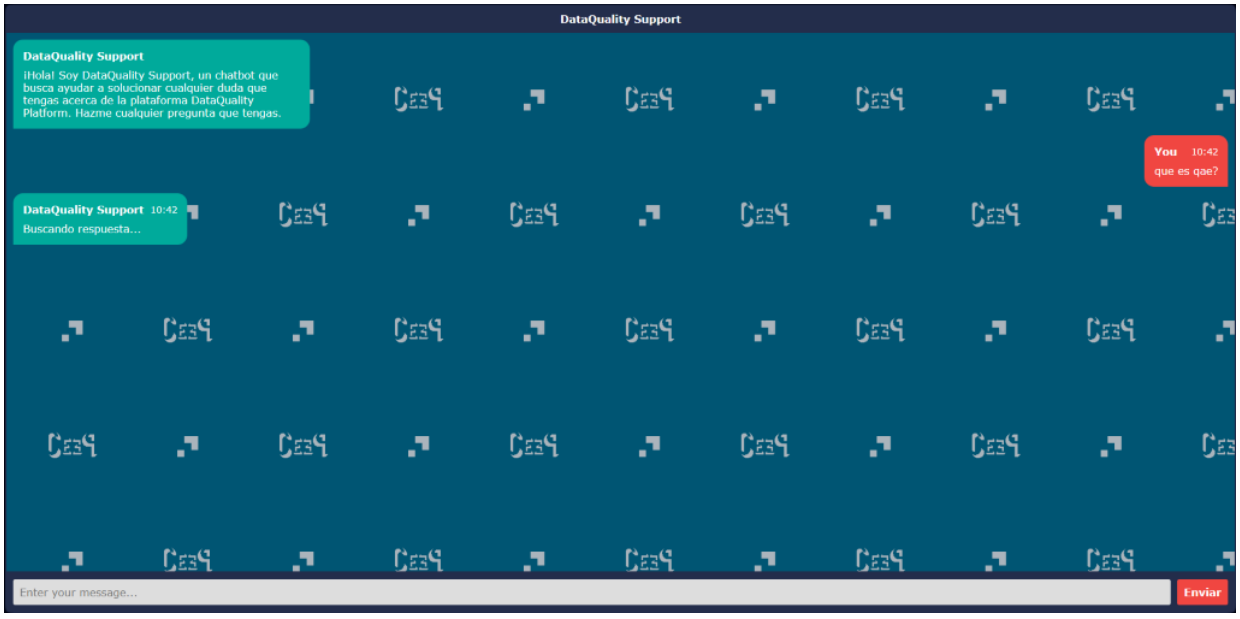
Plataforma

Es un **asistente virtual de apoyo al cliente** donde el usuario puede hacer preguntas y resolver dudas de distintos dominios y niveles de complejidad construido sobre la plataforma de **DataQuality**, además de preguntas acerca de diversas áreas de Data, como Gobierno del Dato o Calidad del Dato.

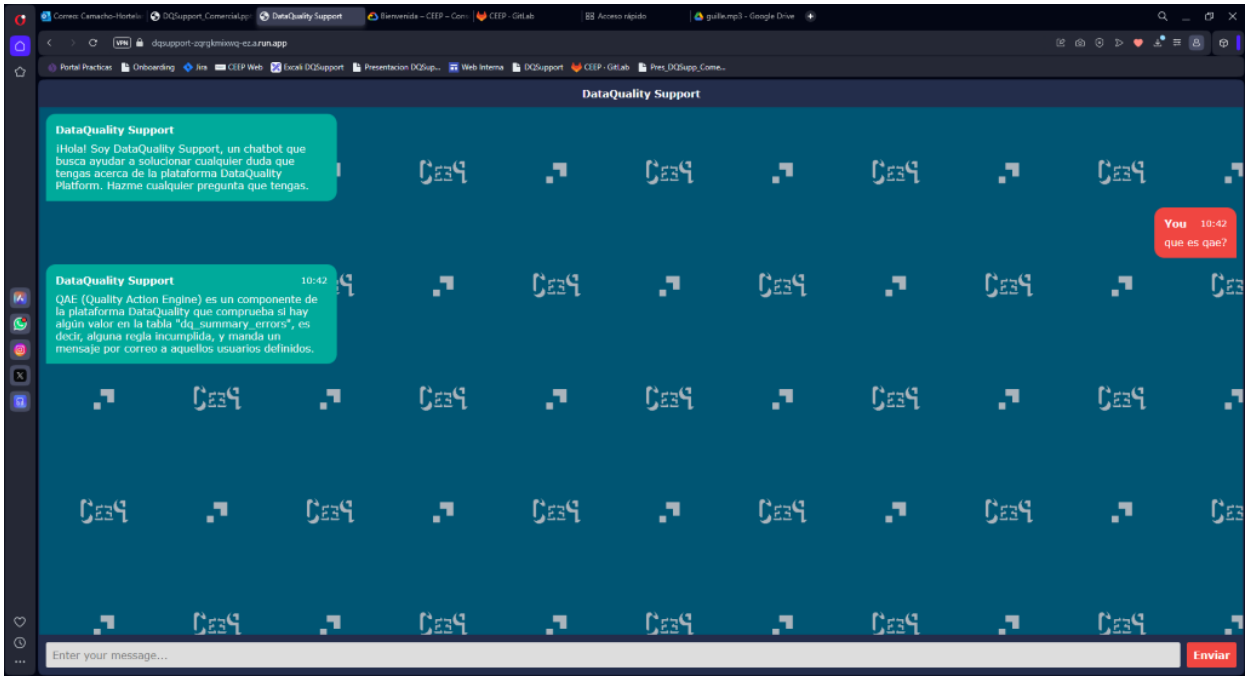
La plataforma está alojada en un Cloud Run, con el siguiente enlace: https://dqsupport-zqrgkmixwq-ez.a.run.app

Nada más abrir la página, el usuario se encontrará con un mensaje de bienvenida.

Cada vez que el usuario haga una pregunta, el asistente virtual mostrará un mensaje temporal mientras busca la respuesta (La primera pregunta siempre tarda un poco más que las siguientes).



Una vez encontrada la respuesta, el asistente virtual la mostrará.



Además, se le pueden hacer preguntas contextuales sobre sus respuestas.

La plataforma tiene distintos componentes, que se pueden distinguir en el siguiente diagrama.

Los colores distinguen las distintas partes y comportamientos de los componentes:

- Las partes roja y azul son componentes **OneShot** que sólo se ejecutan cuando se modifica la base de documentos. La diferencia es que la parte roja tarda entre 30 y 40 minutos en ejecutarse (Asociado a la acción de 'implementar'), mientras que la parte azul tarda mucho menos.
- La parte amarilla es donde está alojado lo de su derecha. Simplemente actúa como **Hosting** de la plataforma, y está ejecutándose continuamente mientras esté implementado. A diferencia de los índices, esta parte no gasta presupuesto cuando no está siendo usada.
- La parte verde engloba el funcionamiento recursivo de la plataforma. Se ejecuta mientras esté abierta la página. Esta parte consta de la creación de la conversación y de la parte de RetrievalQA.

Para explicar cada una de las partes del proyecto se han creado distintas secciones, que se pueden acceder mediante los siguientes enlaces:

- Arquitectura
- Ingesta e Indexación
- Acciones de Administración
- Cloud Run (Docker)
- Interfaz
- Creador de Conversación
- RetrievalQA

Arquitectura

Antes de hablar de la arquitectura de nuestra plataforma, el usuario debería familiarizarse con la arquitectura **RAG** (Retrieval Augmented Generation)

Esta arquitectura, usada en muchas plataformas de **QA** (Question & Answer) con base de conocimiento en documentos, se basa en dos partes:

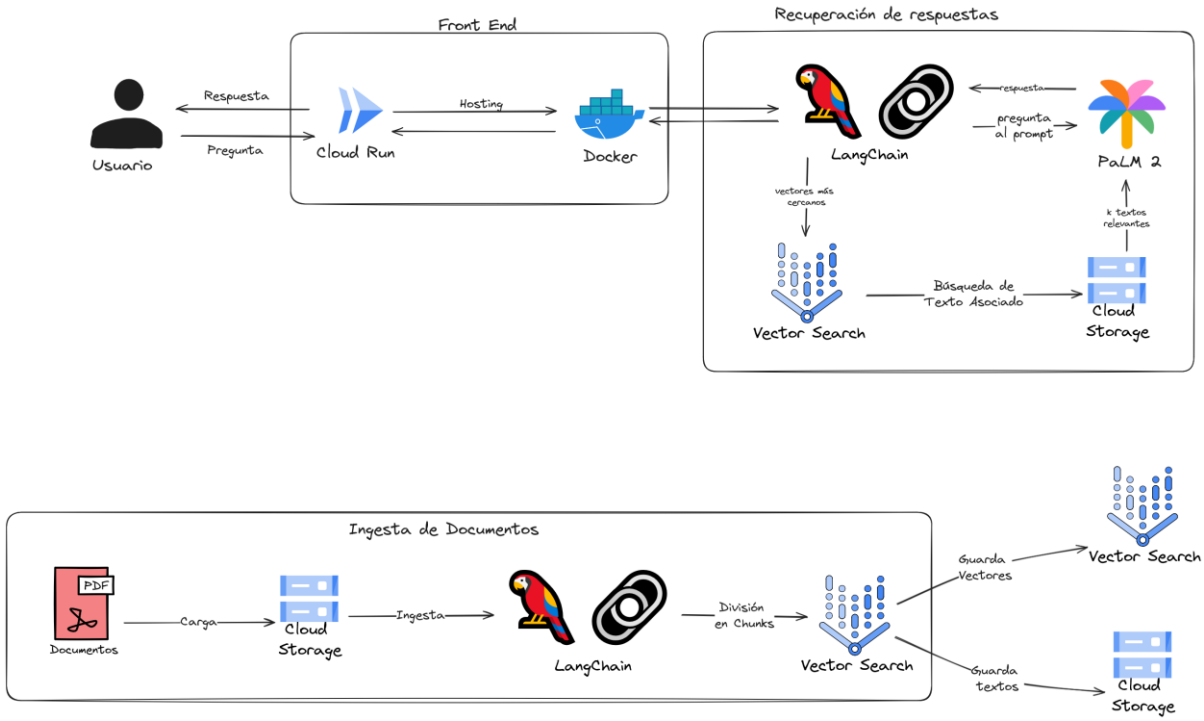
- El **Generator** es un modelo de IA Generativa. El usuario le envía una pregunta, si el modelo puede responderlo, lo hace, pero si no, le pide contexto al Retriever.
- El **Retriever** puede ser varias cosas distintas, pero en esencia se basa en una búsqueda de la pregunta en la base de datos vectorial (o similar) donde se almacena la información.

Como se ve en el diagrama, cada pregunta y respuesta sería:

1. El usuario hace una pregunta al modelo
2. El modelo busca a ver si puede responder con su propio contexto
    a. Si puede, el modelo contesta, y aquí acabaría la cadena
3. Si el modelo no puede responder con su propio contexto, le pregunta al Retriever.

4. El Retriever busca y devuelve las 'x' piezas de contexto de una base de datos vectorial que mejor coincidan con la pregunta.
5. El modelo coge este contexto y lo usa para responder a la pregunta inicial.

Una vez entendido esto, podemos hablar de la arquitectura concreta de nuestra plataforma



Como se puede ver, en nuestro caso hemos usado el modelo de Google **text-bison@002*** de **PaLM 2** como Generator y la herramienta de código abierto **LangChain** como Retriever.

*Actualmente se está explorando el cambio de modelo a Claude 3 Opus o ChatGPT-4 Turbo

Ingesta e Indexación

Para ingerir e indexar, la plataforma hace lo siguiente:

1. Se dividen los documentos (Previamente introducidos por el usuario) en chunks, de tal manera que, en vez de pocos documentos de muchas páginas, nos queden muchos documentos de uno o dos párrafos cada uno, lo cual hace extremadamente más fácil y eficaz la búsqueda vectorial.
2. Dichos documentos, ya divididos, se indexan en un bucket específico para la plataforma, el cual servirá más adelante para la indexación.
3. Tras esto, se crearán índices de Vector Search asociados a estos documentos, creando así una base de datos vectorial a la que puede acceder el Retriever para buscar la información y, con esto, la plataforma estaría lista para usarse.

Estos pasos habría que volver a ejecutarlos cada vez que se modifican o eliminan los documentos originales. Para ello, haremos uso de las Acciones de Administración.

Acciones de Administración

Las acciones de administración sirven para administrar la plataforma. Hay cinco acciones disponibles:

- 'crear': Crea e implementa todo lo necesario para el funcionamiento de la plataforma. Es necesario que exista anteriormente un bucket con los documentos originales en el proyecto pero, aparte de esto, el resto se crea desde 0.
- 'borrar': Borra todo lo relacionado con la plataforma, menos el bucket de los documentos originales citado anteriormente.
- 'implementar': Implementa los índices de Vector Search.
- 'suspender': Anula la implementación de los índices de Vector Search.
- 'reiniciar': Reinicia la plataforma, útil para cuando se haya modificado la documentación original. Es equivalente a hacer una acción 'suspender' seguida de una acción 'implementar'.

Estas acciones se ejecutarán mediante una petición POST a una **Cloud Function**, cuyo único parámetro es el nombre de la acción que se desee ejecutar. Por ejemplo, si se quiere ejecutar la acción 'crear', se mandará una petición similar a la siguiente:

```
curl -m 3610 -X POST https://europe-west4-ceep-394706.cloudfunctions.net/dqsupport-reset-index \
-H "Authorization: bearer $(gcloud auth print-identity-token)" \
-H "Content-Type: application/json" \
-d '{
  "accion": "crear"
}'
```

Esta petición se puede programar, de forma que se suspenda e implemente la plataforma cuando se desee. Por ejemplo, si se decide que la plataforma esté disponible sólo en días de diario en horario lectivo, se puede declarar que se implemente de lunes a viernes a las 8:00 y se suspenda los mismos días a las 17:30.

Esto último es muy importante, y se recomienda encarecidamente hacerlo ya que, al hacer uso de Vertex AI, la plataforma tiene unos costes elevados con tan sólo estar implementada.

Cloud Run (Docker)

Una vez inicializado todo, la plataforma ya estaría preparada para su uso, el cual se hace en un **Docker** alojado en un **Cloud Run** de Google. Este Cloud Run tiene CI/CD automático con el repositorio, lo cual significa que cada vez que se hace un commit cambiando el código al repositorio, la plataforma se actualiza automáticamente.

Interfaz

La interfaz está hecha en HTML y css, usando los colores de la compañía y, como fondo, una malla con los logos del CEEP y de inetum.

Es una interfaz bastante simple y fácil de entender, por lo que no habría mucho más que destacar aquí.

*Actualmente se está revisando la migración de esta interfaz a Streamlit.

Creador de Conversación

Una conversación se crea en el momento que un usuario manda el primer prompt. Esto desencadena un conjunto de acciones:

1. Se llama al **megine**, del cual consultamos los índices del vector search y de su endpoint.
2. Se crea el **me** (matching engine), al que establece la conexión dados los índices del punto anterior, la región y los embedding.
3. Se crea el QA (question & answer), que es un objeto de la clase RetrivalQA, al cual se le pasa el template (formato del prompt), LLM (modelo de machine learning seleccionado) y Retriver (creado a partir del me).

En esencia, el QA es el objeto al que el usuario mandará los prompt y de donde obtendremos las respuestas.

RetrievalQA

RetrievalQA es una clase importada desde **langchain.chains**. Esta clase se usa para crear el QA.

RetrivalQA recibe las preguntas del usuario y sigue una serie de pasos:

1. Se ejecuta el Retriver, que:
    a. Convierte la consulta en un vector.
    b. Calcula la distancia con los vectores de la base de conocimientos.
    c. Devuelve el texto asociado a los más cercanos.
2. Se contruye el prompt con el uso de una plantilla que tiene unas variables a rellenar:
    a. **Context**: Son los vectores extraídos por el Retriver, es decir, el contexto que necesitaría el LLM para responder correctamente a la pregunta.
    b. **Question**: Es la consulta introducida por el usuario.
    c. **History**: Es el historial de mensajes de la conversación.
3. El prompt se pasa al LLM (el modelo generativo) el cual devuelve una respuesta.

Con esto, RetrievalQA devuelve la respuesta generada por el LLM, el texto asociado a los vectores más cercanos y los metadatos de origen de dichos vectores.

A continuación, se muestra el código correspondiente a la creación de RetrievalQA:

```
qa = RetrievalQA.from_chain_type(
    llm=llm,
    chain_type="stuff",
    retriever=retriever,
    return_source_documents=True,
    verbose=verbose,
    chain_type_kwargs={
        "prompt": PromptTemplate(
            template=template,
            input_variables=["context", "question", "history"],
        ),
        "memory": ConversationBufferMemory(
            memory_key="history",
            input_key="question"),
    },
)
```