PROYECTO APLICACIÓN FEEDBACK CON LLM

1. PROYECTO Y SETUP INICIAL

1.1. Descripción del MVP

La idea inicial para el proyecto será que el usuario añada un texto que sea el feedback sobre una web o aplicación para que un LLM clasifique ese feedback y responda al usuario adecuadamente en función del feedback aportado.

1.2. Historias de usuario

- Como usuario,
  Quiero leer en la home de la aplicación una descripción e instrucciones
  Para entender cómo aportar mi feedback.

- Como usuario,
  Quiero enviar mi feedback en texto (ya llegará audio) respecto a una aplicación o web
  Para que mi feedback se guarde y se le haga llegar a los desarrolladores.

- Como empresa de desarrollo de software,
  Quiero guardar y clasificar el feedback enviado por el usuario
  Para saber a qué departamento o persona enviárselo.

- Como usuario,
  Quiero recibir una respuesta adecuada al feedback enviado
  Para confirmar que se está entendiendo y que llegará a los desarrolladores.

- Como usuario,
  Quiero ver un histórico de feedbacks enviados y respuestas recibidas
  Para recordar la respuesta.

1.3. Repositorio Git

1. Abrir Github, crear repositorio Git: https://github.com/xdelrey/feedback
2. Crear una carpeta en local para el proyecto.
3. Clonar repositorio en Visual Studio Code.

1.4. Estructura de Carpetas

```
feedback/
├── data/
├── notebooks/
│   ├── borrador.ipynb???????????????
├── src/
├── backend/
│   ├── app.py
│   ├── llm_client.py
│   ├── model_test.py
├── frontend/
├── .env
├── .gitignore
└── README.md
```

1.5. Configuración de Entorno Virtual y instalación de Dependencias

```
cd C:\ruta\al\proyecto
python -m venv .venv
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
.\.venv\Scripts\Activate.ps1
pip install flask streamlit groq python-dotenv requests
pip list
```

Comprobación para trabajar en el entorno virtual:

- Selección de interprete de Python de `.venv`
  - `Python: Select Interpreter`
- Comprobar activación del .venv
  - `& .\.venv\Scripts\Activate.ps1`
- Cuando quiera desactivar el entorno virtual:
  - `(.venv) PS> deactivate`

1.6. Configuración de `.gitignore`

`.gitignore` es el fichero de texto que le dice a Git qué carpetas o archivos no debe guardar en el repositorio.

- Creación de `.gitignore` con `.venv`
- Creación de `requirements.txt` (o también se usa para actualizar dependencias):
  `pip freeze > requirements.txt`
- Archivo: `/.gitignore`

1.7. Archivo `.env`

- Creación del archivo `.env` con la API Key de Groq.
- Añadimos también otras variables de Groq como Modelo o puerto.
- Comprobamos que las llamadas al modelo funcionan (`/backend/model_test.py`)
- Archivo: `/.env`
- Añadimos el archivo `.env` a `.gitignore`.

2. BACKEND FLASK

2.1. Creación de API con Flask

- Creación de la estructura de la API con Flask
- Archivo: `/backend/app.py`

2.2. Endpoint `/feedback`

- Creación del Endpoing `/feedback` que reciba un JSON y responda 'Gracias!' (hardcodeado)
- Validaciones básicas (vacío, es json...) para responder con un error 400.
- Formato del json (clase 'texto': `{"texto": "Hola"}`
- Comprobar con Thunder Client.
- Archivo: `/backend/app.py`

3. INTEGRACIÓN GROQ + LLM

3.1. Función de respuesta del LLM

- Configuraciones (api_key de Groq, modelo de LLM a utilizar,...)
- Definición del System Prompt
- Declaración de la función `responder_llm()` para enviar el feedback del usuario a Groq y devolver la respuesta del LLM
- Archivo: `/backend/llm_client.py`

2.1. Actualizar endpoint `/feedback`

- Actualización del Endpoing `/feedback` para sustituir la respuesta hardcodeada por la generada por el LLM
- Archivo: `/backend/app.py`

tecnologías

- Github
- Visual Studio Code
- Groq
- Meta Llama 3.3 70B
- Thunder Client
- Flask

requirements.txt

- pandas
- groq
- flask
- os
- dotenv