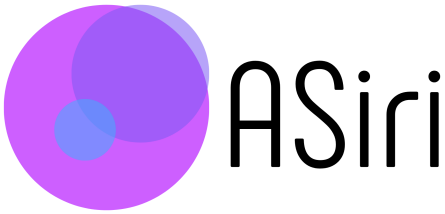


ASiri es un asistente personal creado con el objetivo de programar, manejar y dar seguimiento a las tareas de la vida cotidiana. La interacción con **ASiri** es mediante lenguaje natural, que bien puede ser por entrada de texto o mediante comandos de voz. El objetivo fundamental de este asistente es mostrar cuan útil y poderoso puede ser el uso de un **modelo del lenguaje** en cualquier parte del desarrollo de un producto de software.

Arquitectura e Implementación

Servidor de backend

El servidor de nuestro asistente corre sobre un servidor de **Flask**, desarrollado en **python**, que provee las API's necesarias para hacer peticiones desde clientes, ya sean de texto o mediante audio. Ambas solicitudes son manejadas de la misma forma en cuanto a la lógica de procesamiento, pues un audio es convertido previamente a texto para su análisis. Para la conversión de una solicitud de audio a texto hemos utilizado **Whisper**, un modelo de red neuronal entrenado en el reconocimiento de voz.

> Whisper es un sistema de reconocimiento automático de voz (ASR) desarrollado por **OpenAI**, y entrenado con 680,000 horas de datos multilingües y multitarea supervisados, recopilados de la web. El uso de un conjunto de datos tan grande y diverso lleva a una mayor robustez frente a acentos, ruido de fondo y lenguaje técnico.

Whisper cuenta con 5 modelos disponibles (*tini*, *base*, *small*, *medium*, *large*), de los cuales hemos escogido la versión *base*, entrenada con 74 millones de parámetros; pero esta elección es perfectamente modificable para obtener mejores resultados de transcripción, en dependencia del poder de cómputo con que se dispone.

Dado que gran parte del procesamiento de las solicitudes se realiza utilizando modelos del lenguaje, la interacción del usuario acepta casi cualquier idioma, sin embargo es recomendable interactuar con **ASiri** en **inglés**.

El **modelo de lenguaje** que se utilizó durante el desarrollo del proyecto, y por tanto para el **prompt engineering** fue **Gemini**, a través de su API.

> Gemini es un modelo de inteligencia artificial desarrollado por DeepMind, la división de IA de Google. Se basa en grandes modelos de lenguaje (LLMs) y está enfocado en tareas como generación de texto, razonamiento, y posiblemente integración con otras herramientas de IA para resolver problemas complejos.

Una vez obtenida la solicitud del usuario, el primer paso es determinar si la solicitud constituye una acción específica sobre el almacenamiento de tareas, o es una solicitud que puede ser respondida sin el acceso a la base de datos, es decir, utilizando el entrenamiento del modelo de lenguaje. En este último caso se pasa la solicitud a Gemini, para que genere una respuesta.

Si la solicitud requiere acceso a la base de datos, lo que hacemos es analizar por separado para los tipos `GET`, `POST` o `REMOVE`. La base de información se almacena en un archivo `csv`, al que se accede a través de la biblioteca `pandas`. De forma general, el funcionamiento de estas solicitudes es mediante generación de código pandas.

Cuando se hace una solicitud `GET`, primeramente se extrae información relevante de la query del usuario, a partir de la cual se genera la consulta en pandas, se ejecuta el código, y se obtiene un dataframe de salida. El cual es posteriormente interpretado por Gemini para responder al usuario.

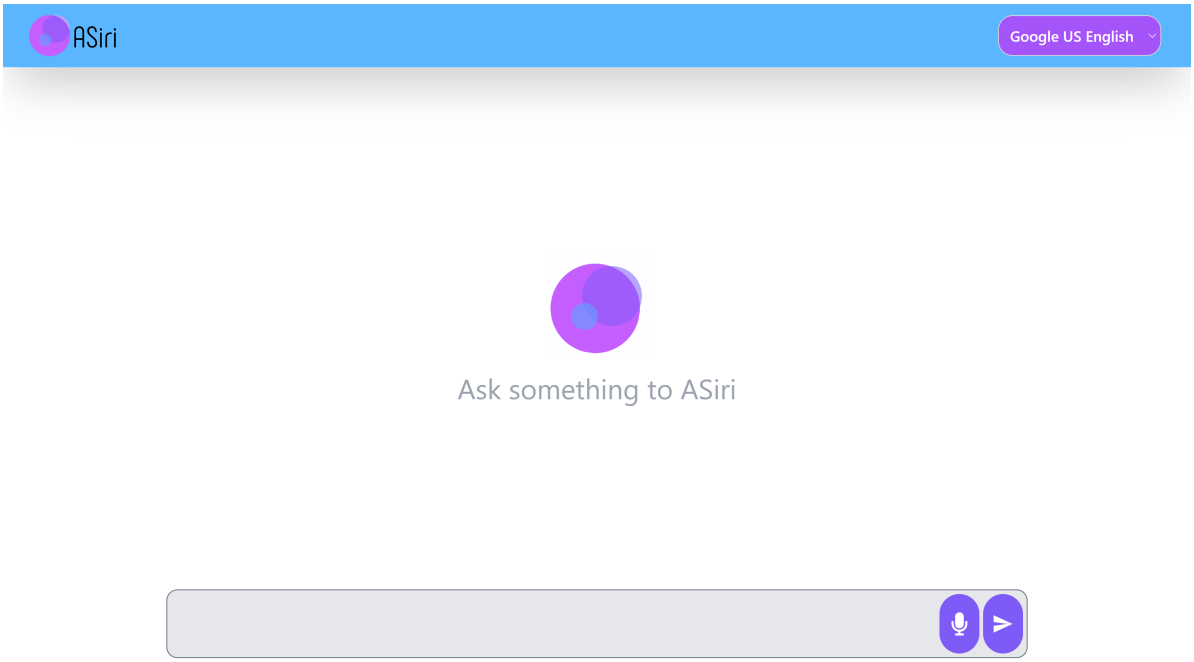
Cuando se hace una solicitud `POST`, se extrae información relevante de la query, a partir de la cual se llevan a cabo comprobaciones sobre la base de datos (como puede ser que esa tarea ya se encuentre programada ese día), y posteriormente se genera con Gemini el código pandas correspondiente a la creación de la tarea. Finalmente se responde al usuario con un mensaje positivo o negativo.

Cuando se hace una solicitud `REMOVE`, se extrae información relevante de la query, se comprueba la consistencia de la tarea que se quiere borrar con la información de la base de datos, se genera el código pandas correspondiente y luego se ejecuta la acción, respondiendo finalmente al usuario.

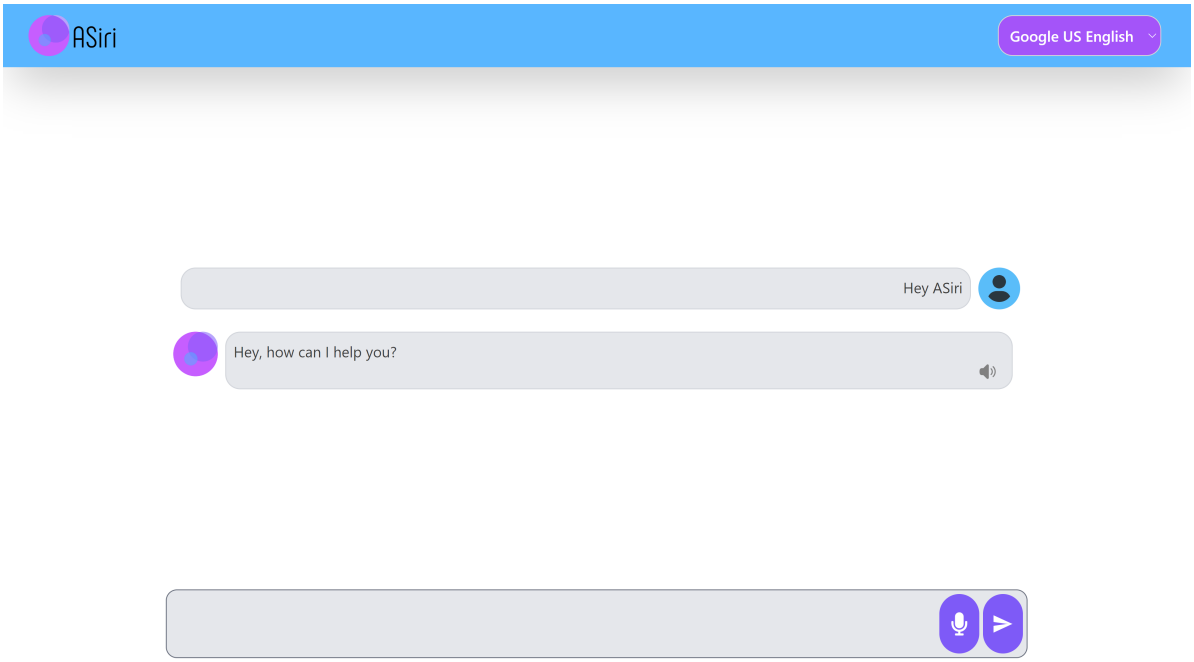
UI

Para el desarrollo de la interfaz de usuario, optamos por desarrollar una aplicación de **React**. El servidor de React se ejecuta, y se puede acceder a él desde cualquier dispositivo de la red local.

> Al ser un proyecto con fines educativos, no se manejan cuentas de usuarios independientes, por lo que todos los dispositivos de la red estarían accediendo como un usuario único.



La interfaz es simple y fácil de usar. Cuenta con un cuadro de texto para la entrada de texto, y el botón de grabación de audio para la solicitud. Al enviar la solicitud, esta pasa al servidor, y se obtiene la respuesta del asistente. La respuesta es obtenida en forma de texto, si este fue el método de entrada del usuario; y si se utilizó un comando de voz la respuesta se lee en voz alta por la aplicación de forma automática, aunque se muestra en texto también. Si la solicitud es mediante texto, también se muestra la opción de leerse en voz alta.



En la esquina superior derecha se muestran opciones de voces que se pueden elegir para **ASiri**. Las voces disponibles varían en dependencia del navegador, así como el idioma que estas utilizan.

Ejecución de desarrollo

A continuación se detallan las instrucciones y dependencias necesarias para ejecutar un servidor de desarrollo de prueba para el proyecto.

Servidor backend

Se deben instalar las dependencias de **python** para ejecutar el servidor.

```
pip install -r /path/to/requirements.txt
```

Luego se ejecuta el servidor de flask simplemente ejecutando el archivo `server.py`

```
python server.py
```

Servidor de React

Primeramente es necesario tener instalado `node.js` y `npm`. Luego debemos configurar la dirección del ordenador que ejecuta el servidor de `flask` en la red. Esto lo hacemos a través de las variables de entorno en `gui/.env.development`, y se debe sustituir la dirección IP y PORT. La mejor forma de obtener esto es ejecutando el servidor de `flask`, que nos muestra el siguiente log:

```
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:8888
* Running on http://192.168.1.105:8888
Press CTRL+C to quit
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 144-216-133
```

En este caso serían: `VITE_SERVER_IP='192.168.1.105'` y `VITE_SERVER_PORT='8888'`.

Una vez configurada la dirección, necesitamos instalar todas las dependencias utilizadas. Para ello ejecutamos el siguiente comando en una terminal en la dirección: `/gui`

```
npm install
```

Ahora solo quedaría ejecutar el servidor de React:

- Local:

  ```
  npm run dev
  ```

- En la red:

  ```
  npm run dev -- --host
  ```

Y ya podemos interactuar con **ASiri**.