Documentación del proyecto “Chat IA – Tony”

Este proyecto fue realizado usando el siguiente stack tecnológico:

- Frontend: React, TailwindCSS (Next.js)
- Backend: Typescript (Next.js)
- API IA: Google Gemini API

Se tomó la decisión de usar Nextjs por ser un framework completo que implementa herramientas como Tailwind y Typescript además de elementos y componentes de React, este framework permite crear proyectos rápidamente usando todas estas tecnologías por default, por lo que es más sencillo crear MVP’s en situaciones en las que por ejemplo, es necesario enfocarse en otros aspectos del desarrollo como lo es aprender a usar una nueva herramienta.

En cuanto al cliente de IA, se tomó la decisión de usar Gemini de Google puesto que a diferencia de OpenAI, ésta ofrece un API gratuita, además, revisando detalladamente la documentación es posible encontrar fácilmente que esta API ofrece la opción de retornar respuestas estructuradas en formato JSON, lo que facilitó la obtención de las respuestas con el formato solicitado.

Para conectar el API de Gemini solo se necesita una llave de API, que se puede generar desde la consola de desarrollo de Gemini. Una vez obteniendo esta llave, es posible llamar el API directamente desde peticiones REST a una URL ó también se puede usar el SDK de Gemini para Javascript o Python, esta fue la opción seleccionada puesto que de esta forma es mucho más sencillo y limpio la forma de configurar tanto los parámetros de entrada como los de salida del API.

Para lograr obtener los parámetros correctos en la respuesta del API se usan los siguientes parámetros de configuración en el llamado del API:

```
config: {
responseMimeType: "application/json",
responseSchema: {
    type: Type.ARRAY,
    items: {
    type: Type.OBJECT,
    properties: {
        readable: {
        type: Type.STRING,
        },
        jsonresponse: {
        type: Type.OBJECT,
        properties: {
            date:{ type:Type.STRING },
            location:{ type: Type.STRING },
            description:{ type: Type.STRING },
            injuries:{ type: Type.BOOLEAN },
            owner:{ type: Type.BOOLEAN },
            complete: { type: Type.BOOLEAN },
            question:{ type: Type.STRING }
        },
        },
    },
    propertyOrdering: ["readable", "jsonresponse"],
    },
},
},
```

Con lo que se logra obtener una respuesta que contenga tanto un texto legible como un JSON con los parámetros especificados.

Para lograr el seguimiento, lo que se hace es enviar un prompt como el siguiente:

```
    contents:
    `1.  Genera una respuesta amigable, concisa y legible que confirme la recepción de la información y haga un breve resumen.
    2.  Extrae los siguientes datos del texto proporcionado y devuélvelos en formato JSON.
    Si algún campo no se puede determinar, usa un valor nulo para el JSON.
    Infiere todos los datos posibles solo por el contexto del texto que manda el usuario,
    la locación puede ser una descripción relativa de dónde sucedió el acontecimiento,
    por ejemplo el domicilio/casa del usuario o la casa de X persona, y en cuanto a la fecha,
    si el usuario habla en presente, usa la fecha actual, si es una fecha relativa por ejemplo, ayer, calcula la fecha,
    para tu referencia, la fecha actual en la que se encuentra el usuario es: ${clientTime}.
    si el usuario habla en primera persona del singular, entonces el es el owner, si habla en primera persona del plural o si se incluye en los
    acontecimientos, entonces el tambien figura como el owner pues está involucrado en el incidente,
    si no hay suficiente información para completar los datos, setea 'complete' en false y haz una pregunta para obtener la información faltante.
    Formato JSON requerido (el orden de los parámetros debe ser estrictamente de esta forma):
    {
        "date": "yyyy-mm-dd",
        "location": "Lugar del suceso",
        "description": "Descripción en una sola oración",
        "injuries": true | false,
        "owner": true | false,
        "complete": true | false,
        "question": "Pregunta para el usuario si es necesario"
    }
    El texto del usuario es:
    "${ userQuestion }"
    ${ followUpAnswer ? 'Este es un dato extra que envía el usuario:' + followUpAnswer + ', úsala para complementar la información.' : '' }
    ${ followUpData ?
        'Y aquí hay un arreglo de JSONs con datos previamente recopilados sobre el mismo incidente que recibe el usuario:' + JSON.stringify(followUpData) + ', úsalos para complementar la pregunta del usuario.'
        : ''
    }`,
    config: {
```

Donde además de la pregunta o situación del usuario, se mandan los JSON previamente generados en caso de que existan y las respuestas del usuario a las preguntas de seguimiento de la IA, con esto se logra proveer del contexto necesario a la IA para que en caso de que la situación que describe el usuario no esté completa, en siguientes iteraciones se pueda completar con las siguientes respuestas del usuario y la IA y entonces completar la situación.