



# Whatsapp ChatBot - Youtube

## 1. Primeros pasos

### Introducción

Bienvenido a la guía para crear tu propio chatbot de WhatsApp utilizando NodeJS. Esta guía está diseñada con un propósito muy claro: proporcionarte un camino claro y estructurado para adentrarte en el mundo de estos chatbots, desde la concepción inicial hasta la implementación de tu proyecto.

Si estás aquí, probablemente ya sabes que los chatbots han revolucionado la forma en que las empresas interactúan con sus clientes, automatizando las comunicaciones y proporcionando respuestas instantáneas a preguntas frecuentes, entre muchas otras funcionalidades. La capacidad de crear soluciones personalizadas en este ámbito representa una habilidad tremendamente valiosa en el mercado actual.

Está dirigida tanto a principiantes que están dando sus primeros pasos en el mundo de la programación como a desarrolladores más experimentados que buscan expandir sus habilidades en el desarrollo de chatbots de WhatsApp. A través de explicaciones detalladas, ejemplos prácticos y consejos estratégicos para la comercialización de tus chatbots, te proporcionaré todas las herramientas necesarias para tener éxito en este campo.

Ya sea que tu objetivo sea mejorar la operación de tu negocio actual, ofrecer servicios de desarrollo de chatbots como freelancer, o incluso iniciar tu propia empresa en este nicho, esta guía está diseñada para ayudarte a lograrlo. Estoy emocionado de acompañarte en este viaje, compartiendo con vos todo lo que aprendí y ayudándote a evitar los obstáculos con los que me encontré en el camino.

Sin mas, comencemos con esta guía. Espero que te sirva!

### Objetivos de la Guía

El principal objetivo de esta guía es brindarte una comprensión completa y detallada de cómo desarrollar y desplegar chatbots de WhatsApp utilizando NodeJS. Para asegurar que obtengas el máximo beneficio de este recurso, hemos establecido los siguientes objetivos específicos:

1. **Entender los Fundamentos:** Adquirir un sólido entendimiento de NodeJS y las tecnologías asociadas necesarias para el desarrollo de chatbots.
2. **Desarrollo Práctico:** Aprender a construir un chatbot de WhatsApp desde cero, utilizando prácticas de programación efectivas y eficientes.
3. **Personalización y Funcionalidades Avanzadas:** Dominar cómo personalizar tu chatbot para satisfacer necesidades específicas, incluyendo el manejo de mensajes entrantes/salientes, integración con APIs externas como ChatGPT, y manejo de archivos multimedia.
4. **Implementación y Pruebas:** Conocer las mejores prácticas para testear tu chatbot y desplegarlo en un entorno de nube, garantizando su óptimo funcionamiento.

5. **Aprovechamiento de Recursos:** Utilizar plantillas y ejemplos de código proporcionados para acelerar el desarrollo y mejorar la comprensión de los conceptos.

## Cómo Usar la Guía

Para aprovechar al máximo esta guía, te recomendamos seguir estas pautas:

- **Lee Secuencialmente:** Aunque cada sección está diseñada para aportar valor por sí misma, te recomendamos seguir el orden propuesto para construir una comprensión integral paso a paso.
- **Practica con los Ejemplos:** Acompañamos conceptos teóricos con plantillas y ejemplos de código. Te animamos a probar estos ejemplos por tu cuenta, modificándolos y experimentando con ellos para ver cómo afectan al comportamiento de tu chatbot.
- **Entiende Cada Línea de Código:** Mientras trabajas con los ejemplos y plantillas, toma un momento para entender qué hace cada línea de código. Esto enriquecerá tu aprendizaje y te proporcionará una base más sólida para la resolución de problemas y la innovación.
- **Utiliza ChatGPT4:** Si tienes acceso a ChatGPT4, considera subir esta guía en formato PDF y utiliza la herramienta para hacer preguntas específicas o solicitar ayuda para construir partes del código. ChatGPT4 puede ser un recurso invaluable para clarificar dudas y proporcionar ejemplos de código personalizados.
- **Aplica lo Aprendido:** Intenta aplicar los conceptos aprendidos en proyectos reales o ejercicios prácticos. La mejor manera de consolidar tu conocimiento es a través de la práctica en situaciones del mundo real.

Recuerda, el objetivo de esta guía no es solo enseñarte a construir un chatbot, sino también a entender profundamente cómo funciona y cómo puede ser adaptado y mejorado. Al final de este camino, tendrás no solo las habilidades técnicas, sino también la confianza para emprender proyectos de chatbot de WhatsApp por tu cuenta o para clientes, y llevar tus ideas innovadoras al mercado.

[1. Primeros pasos](#)  
[2. Beneficios del chatbot](#)  
[3. Teoría Chatbot WhatsApp](#)  
[4. Instalar chatbot WhatsApp](#)  
[5. Mensajes](#)  
[6. Eventos](#)  
[7. Menu de opciones](#)  
[8. ChatGPT](#)  
[9. Whisper voice2text](#)  
[10. Errores comunes](#)  
[Links y referencias](#)

## 2. Beneficios del chatbot

### ¿Cuál es la propuesta de valor de un chatbot?

Los chatbots ofrecen una gama impresionante de beneficios, facilitando la interacción entre empresas y usuarios. Te dejamos algunos puntos clave que destacan su propuesta de valor:

- **Disponibilidad constante:** Están listos para responder preguntas en cualquier momento del día, los 365 días del año, sin necesidad de descanso.
- **Capacidad de interpretar audios:** Pueden entender y procesar comandos de voz, haciendo la interacción más natural para el usuario.
- **Envío de imágenes y enlaces:** Enriquecen la conversación enviando material visual o links relevantes para ofrecer una respuesta más completa.
- **Delegación efectiva:** Si el chatbot no puede resolver una consulta, puede transferir al usuario a un asesor humano sin interrupciones.
- **Gestión de múltiples usuarios:** Manejan varias conversaciones al mismo tiempo, aumentando la eficiencia en la atención al cliente.

Estas características hacen de los chatbots una herramienta invaluable para mejorar la experiencia de usuario, optimizando procesos y brindando un servicio de atención al cliente sin igual.

### ¿Cuál es el alcance de este chatbot?

Gracias a la flexibilidad que ofrece NodeJS, el alcance de este chatbot es prácticamente ilimitado. Puede integrarse con una amplia variedad de servicios y APIs para ampliar sus funcionalidades. Aquí algunos ejemplos:

- **Integración con Google Calendar:** Permite crear y gestionar eventos directamente desde el chat.
- **Uso de Zapier:** Facilita la conexión con cientos de aplicaciones para acciones como enviar correos electrónicos o recibir notificaciones.
- **Conexión con bases de datos de productos:** Ofrece información actualizada sobre el stock disponible.

En esencia, el límite de lo que el chatbot puede hacer está dado solo por tu creatividad y las necesidades específicas que busques cubrir.

### ¿Cuáles son las limitaciones del chatbot?

Las limitaciones de un chatbot dependen en gran medida del proveedor de servicios que elijas. Aquí algunos puntos a considerar:

- **Proveedores gratuitos como Baileys:** Aunque suelen ser eficientes, pueden presentar dificultades al momento de implementar ciertas funcionalidades, como el uso de botones, y en ocasiones pueden enfrentar problemas aislados.
- **Proveedores oficiales como Meta y Twilio:** Ofrecen soluciones robustas ideales para entornos de producción, pero pueden implicar costos adicionales. Ideales para manejar un volumen alto de conversaciones y suelen requerir una configuración un poco más compleja.
- **Limitaciones de WhatsApp:** Utilizar proveedores gratuitos puede exponerte a riesgos de seguridad, como la posibilidad de que WhatsApp considere tu actividad como spam y bloquee el número. Es crucial mantener una buena gestión de contactos para evitar estos inconvenientes. Agendar nuevos clientes y solicitar que los clientes te agenden es esencial para evitar este problema.

Seleccionar el proveedor adecuado y estar consciente de estas limitaciones es esencial para asegurar el éxito y la eficiencia de tu chatbot en WhatsApp. No te preocupes, en el capítulo siguiente esta la explicación mas en detalle de que es un provider y que opciones hay disponibles.

## 3. Teoría Chatbot WhatsApp

### ¿Qué es este chatbot?



Imagínate tener un ayudante virtual que trabaja para ti en WhatsApp, respondiendo preguntas y enviando mensajes a cualquier hora del día. Este chatbot es exactamente eso: un programa que vive en tu computadora o en la nube y actúa como un puente entre tú y tus contactos de WhatsApp. Cuando escaneas un código QR con tu teléfono, es como darle la llave de tu WhatsApp a este programa para que pueda manejar los mensajes por ti. Es una forma genial de automatizar tareas, responder consultas y estar siempre disponible para quien te necesite, sin tener que estar pegado al teléfono. Una vez loguado, en caso de usar un proveedor gratis, vas a poder ver desde tu teléfono los mensajes que envía y recibe el bot

### ¿Cuáles son los elementos mas importantes?


Para que este chatbot funcione correctamente, se apoya en tres pilares fundamentales:

1. **Flow (Flujo):** Este es el camino de la conversación, donde decidís cómo quieres que tu chatbot converse. Imagina programarlo para que cuando alguien diga "hola", él responda con un amigable "¡Hola! ¿En qué puedo ayudarte hoy?". Acá es donde se definen esas interacciones. Los flujos hacen referencia al hecho de construir un flujo de conversación. Esto es un flow podemos observar que están presentes dos métodos importantes **addKeyword** y **addAnswer**.
2. **Provider (Proveedor):** Piensa en esto como el mensajero. Es el que lleva y trae los mensajes entre tu chatbot y WhatsApp. Dependiendo de con quién quieras que hable tu bot (como WhatsApp, Twilio, o incluso la API oficial de WhatsApp), puedes cambiar este proveedor sin tener que alterar el resto de tu configuración.
3. **Database (Base de Datos):** Es como el diario de vida de tu chatbot, donde guarda toda la información importante para recordar conversaciones pasadas. Esto le permite a tu bot ser más inteligente y personalizado en sus respuestas, ya que puede recordar con quién está hablando y qué se ha dicho antes. Esta base de datos es muy simple de implementar ya que esta pre configurada en la librería. Sin embargo, no vas a poder modificar lo que se guarda en la base de datos. Eso lo vamos a explicar mas adelante. A diferencia de JSON y MOCK (Memoria local), Mongo necesita unas configuraciones adicionales, te dejo un ejemplo de como se debería ver:

Referencia:

**Conviértete en un Programador Backend** aprendiendo todo de Cloud y Nodejs  
Crear chatbot WhatsApp en minutos — Servicio de chatbot para whatsapp gratis proyecto OpenSource  
 <https://bot-whatsapp.netlify.app/docs/essential/>

**Crear chatbot WhatsApp en minutos**  
Con esta librería, puedes configurar respuestas automatizadas para preguntas frecuentes, recibir y responder mensajes de manera automatizada, y hacer un seguimiento de las interacciones con los clientes. Además, nuestro Chatbot se integra fácilmente con otros sistemas y herramientas que ya esté utilizando en su negocio.  
[Ver documentación](#)



## 4. Instalar chatbot WhatsApp

### Introducción al capítulo

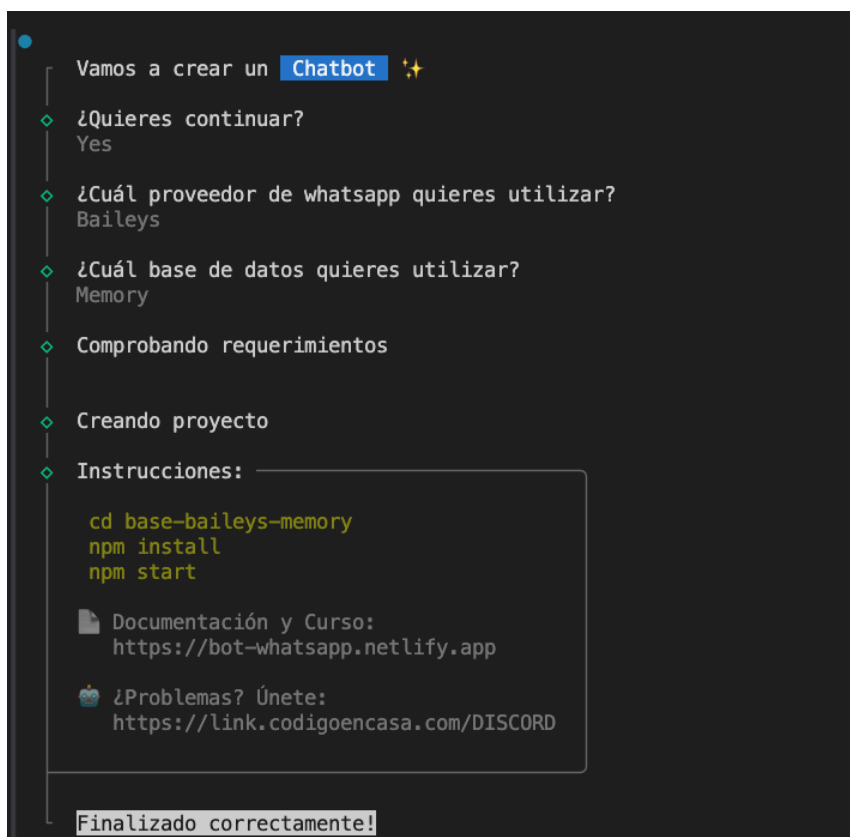
En el capítulo anterior, exploramos la teoría detrás de la creación de un bot de WhatsApp. Ahora que tenemos una base sólida, es momento de poner manos a la obra. Vamos a programar y configurar nuestro propio bot. A continuación, te guiaré a través de una serie de pasos prácticos para hacerlo.

### ¿Cómo comenzar?

Para comenzar hay que correr el siguiente mensaje en la terminal

```
npm create bot-whatsapp@latest
```

Luego de correr este mensaje, en la terminal nos va a salir una serie de preguntas. Para una primera prueba puedes seguir la siguiente configuración:



```
Vamos a crear un Chatbot ✨
◇ ¿Quieres continuar?
  Yes
◇ ¿Cuál proveedor de whatsapp quieres utilizar?
  Baileys
◇ ¿Cuál base de datos quieres utilizar?
  Memory
◇ Comprobando requerimientos
◇ Creando proyecto
◇ Instrucciones:
  cd base-baileys-memory
  npm install
  npm start
  Documentación y Curso:
  https://bot-whatsapp.netlify.app
  🤖 ¿Problemas? Únete:
  https://link.codigoencasa.com/DISCORD
Finalizado correctamente!
```

Una vez instalado hay que seguir las instrucciones que nos figuran ahí en la terminal

```
cd base-baileys-memory
npm install
```

## ¿Como vincular Whatsapp?

Una vez instaladas las dependencias, es hora de correr el bot y vincular nuestro Whatsapp con esa instancia. Para iniciarlo corremos:

```
npm start
```

Nos debería salir el siguiente mensaje

```
gonza@Gonzas-MacBook-Pro base-baileys-memory % npm start

> base-bailey-memory@1.0.0 prestart
> npx eslint . --no-ignore

> base-bailey-memory@1.0.0 start
> node app.js

▶ ESCANEAR QR ▶
Existen varias maneras de escanear el QR code
- Tambien puedes visitar http://localhost:3000
- Se ha creado un archivo que finaliza qr.png

⚡ ⚡ ACCIÓN REQUERIDA ⚡ ⚡
Debes escanear el QR Code 📱 bot.qr.png
Recuerda que el QR se actualiza cada minuto
Necesitas ayuda: https://link.codigoencasa.com/DISCORD
```

Ahora desde el teléfono que quieras que sea el Host de este bot, osea al numero que hay que escribirle para que conteste nuestro bot, tenemos que seguir los siguientes pasos:

1. Abrir Whatsapp
2. Configuración → Dispositivos vinculados → Vincular dispositivo
3. Escanear el archivo QR.png (Se resetea cada 60 segundos).

Si te lograste conectar correctamente debería aparecer un cartel de "proveedor conectado y listo". Ahora puedes escribirle a ese numero un simple "Hola" y ver que responda correctamente.

Felicidades! Ya pudiste correr tu primer bot. Te voy a dejar algunas plantillas básicas con distintas configuraciones de provider y databases para que puedas elegir la que te guste y usarlo de punto de partida. En el próximo capítulo te vamos a enseñar como editar los flujos y poder activar estos flujos de distintas maneras.

## Plantillas para comenzar

bot-whatsapp/starters/apps at main · codigoencasa/bot-whatsapp

👤 Crear Chatbot WhatsApp en minutos. Únete a este proyecto OpenSource (Typescript Version Pronto) - [codigoencasa/bot-whatsapp](#)

🌐 <https://github.com/codigoencasa/bot-whatsapp/tree/main/starters/apps>

**codigoencasa/bot-whatsapp**

👤 Crear Chatbot WhatsApp en minutos. Únete a este proyecto OpenSource (Typescript Version Pronto)

👤 50 Contributors    🗨️ 15 Issues    💬 36 Discussions    ⭐ 2k Stars    🍴 746 Forks

## 5. Mensajes

En nuestro chatbot, la interacción comienza con la configuración de cómo debe responder a ciertos mensajes. Este proceso utiliza dos métodos fundamentales de nuestra librería: `addKeyword` y `addAnswer`. A continuación, explicaremos qué hace cada uno de estos métodos y cómo funcionan en conjunto para manejar las conversaciones.

### Método `addKeyword`

El método `addKeyword` es esencial para definir cómo el chatbot identifica los mensajes que requieren una respuesta. Este método espera una lista de "palabras clave" que, cuando son detectadas en un mensaje entrante, activan el chatbot para responder. Las palabras clave son señales para el chatbot, indicándole que preste atención y se prepare para actuar. Por ejemplo, si configuras `addKeyword` con las palabras `['hola', 'ole', 'alo']`, cualquier mensaje entrante que contenga alguna de estas palabras activará el flujo asociado a este método.

### Método `addAnswer`

Una vez que se activa un flujo mediante `addKeyword`, el siguiente paso es determinar qué debería responder el chatbot. Aquí es donde entra en juego el método `addAnswer`. Este método simplemente toma un texto que será enviado como respuesta al mensaje que activó el chatbot. La respuesta no necesita ser compleja; es simplemente lo que quieres que tu chatbot diga una vez que reconoce una de las palabras clave.

### Ejemplo Práctico

Consideremos un ejemplo para ilustrar cómo se utilizan estos dos métodos en la práctica:

```
const flowPrincipal = addKeyword(['hola', 'ole', 'alo'])
  .addAnswer('Buanas como estas?')
```

En este ejemplo, el `flowPrincipal` es configurado inicialmente con el método `addKeyword`, donde las palabras clave son `['hola', 'ole', 'alo']`. Esto significa que si un mensaje entrante contiene cualquiera de estas palabras, el flujo se activará. Inmediatamente después, utilizamos el método `addAnswer` para definir la respuesta del chatbot: 'Buenas, ¿cómo estás?'.

Así, cuando el chatbot recibe un mensaje que dice "hola", "ole" o "alo", automáticamente responderá con "Buenas, ¿cómo estás?". Este mecanismo simple pero poderoso permite que el chatbot interactúe de manera relevante y oportuna con los usuarios, proporcionando una experiencia de conversación fluida y natural.

## 6. Eventos

En la programación, un **evento** es una acción o suceso, como un clic del usuario, una entrada de texto, o un mensaje recibido, que puede ser detectado y manejado por el software. Los eventos son fundamentales para crear aplicaciones interactivas, como chatbots, porque permiten al software responder dinámicamente a lo que está ocurriendo en el entorno de ejecución.

En el contexto de un chatbot, los eventos pueden incluir recibir un mensaje, recibir un tipo específico de archivo, o incluso acciones específicas definidas por el programador. La capacidad de manejar estos eventos de manera efectiva es clave para diseñar chatbots que sean responsivos y útiles.

## Importación de Eventos desde la Librería

Para manejar eventos en nuestro chatbot que utiliza la librería `@bot-whatsapp/bot`, necesitamos primero importar los eventos disponibles. Esto se hace de la siguiente manera:

```
const {
  createBot,
  createProvider,
  createFlow,
  addKeyword,
  EVENTS,
} = require("@bot-whatsapp/bot");
```

Aquí, `EVENTS` es un objeto que contiene varios tipos de eventos que nuestro chatbot puede manejar.

## Plantillas de Eventos

La librería define varios eventos estándar que podemos utilizar para activar diferentes flujos en nuestro chatbot. Vamos a describir algunos de ellos y cómo se utilizan:

1. **EVENTS.WELCOME:** Este evento se activa cuando el chatbot recibe un mensaje que no coincide con ninguna de las palabras clave configuradas. Es útil para dar una respuesta inicial o de bienvenida a los usuarios que comienzan la interacción sin un comando específico.

```
const flowWelcome = addKeyword(EVENTS.WELCOME)
  .addAnswer("Este es el flujo Welcome");
```

2. **EVENTS.MEDIA:** Activado cuando se recibe un archivo multimedia, como una imagen o video. Este evento permite al chatbot responder específicamente a entradas multimedia.

```
const flowMedia = addKeyword(EVENTS.MEDIA)
  .addAnswer("Este es el flujo Media");
```

3. **EVENTS.LOCATION:** Se dispara cuando el usuario envía una ubicación. Esto es especialmente útil para aplicaciones que necesitan procesar o responder basadas en la ubicación del usuario.

```
const flowLocation = addKeyword(EVENTS.LOCATION)
  .addAnswer("Este es el flujo Location");
```

4. **EVENTS.DOCUMENT:** Utilizado cuando se recibe un documento, como un PDF o Word. El chatbot puede responder de manera adecuada, tal vez procesando o almacenando el documento.

```
const flowDocument = addKeyword(EVENTS.DOCUMENT)
  .addAnswer("Este es el flujo Document");
```

5. **EVENTS.VOICE\_NOTE:** Este evento es para notas de voz. El chatbot podría iniciar un flujo para transcribir la nota de voz o simplemente reconocer que se recibió y pedir más información.



```
const flowVoice = addKeyword(EVENTS.VOICE_NOTE)
  .addAnswer("Este es el flujo Voice Note");
```

6. **EVENTS.ACTION:** Este evento es especial porque no se activa por una acción directa del usuario, sino que debe ser llamado explícitamente desde otra función dentro del código. Es útil para realizar acciones internas complejas.

```
const flowAction = addKeyword(EVENTS.ACTION)
  .addAnswer("Este es el flujo Action");
```

Cada uno de estos eventos permite al chatbot manejar diferentes tipos de interacciones de manera específica, lo que hace que el bot sea más interactivo y útil para el usuario. Configurar correctamente estos eventos es crucial para aprovechar al máximo las capacidades del chatbot.

## 7. Menu de opciones

Ahora que comprendes cómo funcionan los métodos básicos como `addKeyword` y `addAction`, podemos pasar a un ejemplo más avanzado que ilustra cómo crear un menú de opciones interactivo en un chatbot. Este tipo de funcionalidad es esencial para guiar a los usuarios a través de diferentes funcionalidades que tu chatbot puede ofrecer.

```
const menu = "Este es el menu de opciones, elegi opciones 1,2,3,4,5 o 0"
const menuFlow = addKeyword(EVENTS.ACTION).addAnswer(
  menu,
  { capture: true },
  async (ctx, { gotoFlow, fallBack, flowDynamic }) => {
    if (!["1", "2", "3", "4", "5", "0"].includes(ctx.body)) {
      return fallBack(
        "Respuesta no válida, por favor selecciona una de las opciones."
      );
    }
    switch (ctx.body) {
      case "1":
        return await gotoFlow("menu1");
      case "2":
        return await gotoFlow("menu2");
      case "3":
        return await gotoFlow("menu3");
      case "4":
        return await gotoFlow("menu4");
      case "5":
        return await gotoFlow("menu5");
      case "0":
        return await flowDynamic(
          "Saliendo... Puedes volver a acceder a este menú escribiendo '*Menú'"
        );
    }
  });
```

```
}  
}  
);
```

## Descripción del Código

### 1. Configuración del Flujo:

- Utilizamos `addKeyword(EVENTS.ACTION)` para iniciar este flujo mediante una acción programada. Este tipo de evento se activa no por una entrada directa del usuario, sino por una llamada de función desde otra parte del código, ideal para menús que se presentan en respuesta a comandos específicos.

### 2. Definición de la Respuesta:

- El método `addAnswer` se configura con un parámetro de `capture: true`. Esto indica que el flujo debe "capturar" la próxima entrada del usuario, que se espera sea su selección del menú.
- Se proporciona una función asincrónica como callback, que procesa la entrada del usuario una vez recibida.

### 3. Validación de la Entrada:

- Dentro de la función, primero se verifica si el mensaje (`ctx.body`) está entre las opciones válidas (`["1", "2", "3", "4", "5", "6"]`). Si no es así, se utiliza `fallBack` para enviar un mensaje de error y pedir al usuario que elija una opción válida.

### 4. Procesamiento de la Elección:

- Utilizamos una declaración `switch` para manejar las diferentes opciones. Según el número que el usuario envíe, se llama a `gotoFlow` con el flujo correspondiente (`menu1`, `menu2`, etc.). Esto redirige al usuario al flujo asociado con la opción elegida.
- Si el usuario selecciona "0" (generalmente usado para salir), se utiliza `flowDynamic` para enviar un mensaje de despedida y terminar la interacción actual.

## Consideraciones Adicionales

### • Personalización del Menú:

- El `menu` en `addAnswer(menu, ...)` debería ser una variable o constante que contenga el texto del menú que se mostrará al usuario. Asegúrate de definirlo con todas las opciones disponibles listadas claramente.

### • Extensibilidad:

- Este método hace fácil añadir más opciones al menú simplemente extendiendo los casos en la declaración `switch` y añadiendo los flujos correspondientes.

### • Mantenimiento:

- Gracias a la modularidad y la clara separación de las lógicas de flujo, este código es fácil de mantener y actualizar. Los flujos individuales (`menu1`, `menu2`, etc.) pueden modificarse independientemente sin afectar la lógica del menú principal.

Este ejemplo ilustra cómo puedes construir un sistema de menú robusto y fácil de navegar para tus usuarios, mejorando significativamente la interactividad y la funcionalidad de tu chatbot.

### Trae el mensaje "Menu" desde un archivo txt

Para manejar archivos y datos externos en un proyecto Node.js, es común utilizar el módulo `path` que proporciona utilidades para trabajar con rutas de archivos y directorios. Aquí, te explico cómo puedes traer una variable de texto desde un archivo externo de manera eficiente y segura utilizando este módulo.

### Uso del Módulo `path`

El módulo `path` ayuda a construir rutas de archivo que son compatibles con cualquier sistema operativo donde se ejecute Node.js. Esto es particularmente útil para evitar problemas con los separadores de ruta que difieren entre sistemas (por ejemplo, Windows usa `\` mientras que UNIX usa `/`).

### Código para Traer una Variable de Texto

Aquí tienes un ejemplo de cómo puedes cargar el contenido de un archivo de texto en tu aplicación Node.js:

```
// Importa el módulo 'path' para manejar rutas de archivo
const path = require('path');

// Construye la ruta al archivo que deseas leer
const pathMenu = path.join(__dirname, 'mensajes', "menu.txt");

// Importa el módulo 'fs' para operaciones de archivo
const fs = require('fs');

// Lee el contenido del archivo de manera síncrona
const menuText = fs.readFileSync(pathMenu, 'utf8');

// Utiliza 'menuText' como la variable de texto que contiene el contenido
del archivo
console.log(menuText);
```

### Explicación del Código

#### 1. Importar Módulos Necesarios:

- `path`: Necesario para manejar rutas de archivo de manera segura.
- `fs`: Módulo de sistema de archivos de Node.js que permite leer y escribir en archivos.

#### 2. Construir la Ruta del Archivo:

- `__dirname`: Una variable global en Node.js que contiene la ruta del directorio del archivo actual.

- `path.join()` : Método para unir varios segmentos de una ruta de manera que resulte en una ruta bien formada. Previene errores comunes como duplicación de separadores.

### 3. Leer el Contenido del Archivo:

- `fs.readFileSync()` : Método para leer archivos de manera síncrona. Bloquea el hilo de Node.js hasta que el archivo es leído completamente. Esto es útil para configuraciones o datos que necesitan cargarse al inicio del script.
- `'utf8'` : Especifica que el contenido del archivo se debe interpretar como texto UTF-8.

Este método para cargar una variable de texto desde un archivo es especialmente útil para configuraciones, menús interactivos, o cualquier otro dato que preferirías mantener fuera del código principal por razones de organización o facilidad de mantenimiento. Al almacenar este tipo de datos en archivos externos, facilitas la actualización y la gestión del contenido sin necesidad de modificar y re-deployar el código de la aplicación.

---

## 8. ChatGPT

### ¿Cómo vincular la API de ChatGPT?

Para integrar la API de ChatGPT de OpenAI en tu aplicación Node.js, primero deberás registrar y obtener una API key de OpenAI. Visita [OpenAI](#), crea una cuenta, navega al panel de control de API, y genera una nueva clave API. Guarda esta clave de manera segura.

### Configuración del Archivo .env

Crear un archivo `.env` en el directorio raíz de tu proyecto Node.js para manejar configuraciones sensibles como tu clave API. Dentro de este archivo, añade tu clave API de la siguiente manera:

```
OPENAI_API_KEY=tu_clave_api_aquí
```

Utilizar variables de entorno ayuda a mantener tu código seguro, especialmente si compartes el código en repositorios públicos o trabajas en entornos de equipo.

### Instalación de Dependencias

Necesitarás instalar el paquete de OpenAI para Node.js. Abre tu terminal y ejecuta el siguiente comando en el directorio de tu proyecto:

```
npm install openai
```

### Crear el Archivo chatgpt.js

Crea un archivo llamado `chatgpt.js` en tu proyecto para manejar las interacciones con la API de ChatGPT. En este archivo, implementa la función para enviar peticiones a la API:

```

const { Configuration, OpenAIApi } = require("openai");

const chat = async (prompt, text) => {
  try {
    const configuration = new Configuration({
      apiKey: process.env.OPENAI_API_KEY,
    });
    const openai = new OpenAIApi(configuration);
    const completion = await openai.createChatCompletion({
      model: "gpt-3.5-turbo",
      messages: [
        { role: "system", content: prompt },
        { role: "user", content: text },
      ],
    });
    return completion.data.choices[0].message;
  } catch (err) {
    console.error("Error al conectar con OpenAI:", err);
    return "ERROR";
  }
};

module.exports = chat;

```

### Importación y Uso en app.js

Importa y utiliza la función `chat` directamente en tu archivo `app.js`. Aquí te muestro cómo implementar la solicitud a la API sin encapsularla en una función adicional:

```

const chatGPT = require('./chatgpt');

const prompt = "El sistema debería ser capaz de responder preguntas acerca de Node.js.";
const question = "¿Qué es Node.js?";
chatGPT(prompt, question)
  .then(response => console.log("Respuesta de ChatGPT:", response))
  .catch(error => console.error("Error al obtener respuesta:", error));

```

Con estos ajustes, tu aplicación Node.js está configurada para comunicarse con la API de ChatGPT de OpenAI, permitiéndote enviar preguntas y recibir respuestas generadas por inteligencia artificial. Esta configuración es segura, modular y fácil de mantener, ideal para aplicaciones que requieren capacidades avanzadas de procesamiento de lenguaje natural.

## 9. Whisper voice2text

Este tutorial se dirige a desarrolladores que buscan implementar una funcionalidad en sus chatbots de WhatsApp para convertir notas de voz a texto usando la API de OpenAI, específicamente el modelo Whisper. A continuación, se describe cómo configurar el entorno y el código necesario para esta tarea.

### Configuración del Entorno y Preparación del Código

Antes de comenzar a escribir el código, es crucial preparar el entorno de desarrollo realizando las siguientes acciones:

- **Instalación de Dependencias:** Utilizar npm para instalar los módulos necesarios: `fs` para operaciones de archivos, `openai` para interactuar con la API de OpenAI, y `@adiwajshing/baileys` para la conexión con WhatsApp.
- **Configuración de Variables de Entorno:** Asegurarse de que el archivo `.env` está creado y contiene la clave API de OpenAI ( `OPENAI_API_KEY` ).
- **Creación de Directorio Temporal:** Establecer una carpeta `tmp` en el directorio del proyecto para almacenar temporalmente los archivos de audio.
- **Actualización de Dependencias:** Modificar el archivo `package.json` para asegurar que la versión de `openai` sea `^3.3.0`.

En el archivo `app.js`, es necesario configurar la ejecución del script para procesar las notas de voz recibidas:

```
require("dotenv").config();
const { handlerAI } = require("./voice2text")

const flowVoice = addKeyword(EVENTS.VOICE_NOTE).addAction(
  async (ctx, ctxFn) => {
    const text = await handlerAI(ctx);
    console.log(text);
  }
);
```

### Script para Convertir Voz a Texto

El script `voice2text.js` es el corazón de la operación de conversión. Incluye todo el código necesario para descargar la nota de voz, convertirla de formato OGG a MP3, y finalmente usar Whisper para transcribir el contenido de audio a texto.

```
const { downloadMediaMessage } = require("@adiwajshing/baileys");
const { Configuration, OpenAIApi } = require("openai");
const ffmpegPath = require("@ffmpeg-installer/ffmpeg").path;
const ffmpeg = require("fluent-ffmpeg");
const fs = require("fs");
ffmpeg.setFfmpegPath(ffmpegPath);
```

```

const voiceToText = async (path) => {
  if (!fs.existsSync(path)) {
    throw new Error("No se encuentra el archivo");
  }
  try {
    const configuration = new Configuration({
      apiKey: process.env.OPENAI_API_KEY,
    });
    const openai = new OpenAIApi(configuration);
    const resp = await openai.createTranscription(
      fs.createReadStream(path),
      "whisper-1"
    );
    return resp.data.text;
  } catch (err) {
    console.log(err.response);
    return "ERROR";
  }
};

const convertOggMp3 = async (inputStream, outputStream) => {
  return new Promise((resolve, reject) => {
    ffmpeg(inputStream)
      .audioQuality(96)
      .toFormat("mp3")
      .save(outputStream)
      .on("progress", (p) => null)
      .on("end", () => {
        resolve(true);
      });
  });
};

const handlerAI = async (ctx) => {
  const buffer = await downloadMediaMessage(ctx, "buffer");
  const pathTmpOgg = `${process.cwd()}/tmp/voice-note-${Date.now()}.ogg`;
  const pathTmpMp3 = `${process.cwd()}/tmp/voice-note-${Date.now()}.mp3`;
  await fs.writeFileSync(pathTmpOgg, buffer);
  await convertOggMp3(pathTmpOgg, pathTmpMp3);
  const text = await voiceToText(pathTmpMp3);
  fs.unlink(pathTmpMp3, (error) => {
    if (error) throw error;
  });
  fs.unlink(pathTmpOgg, (error) => {
    if (error) throw error;
  });
  return text;
};

```

```
};  
  
module.exports = { handlerAI };
```

### ¿Cómo optimizar el consumo de la API?

La implementación eficiente de ChatGPT en un chatbot de WhatsApp puede significar un balance entre costo y efectividad. Es crucial optimizar el uso de la API no solo para mantener los costos controlados, sino también para garantizar una experiencia de usuario fluida y eficiente. Aquí se detallan algunas estrategias para optimizar el consumo de la API de ChatGPT.

#### Uso de Respuestas Estáticas vs. Dinámicas

Muchas interacciones dentro de un chatbot no requieren el procesamiento avanzado de lenguaje que proporciona ChatGPT. Por ejemplo, respuestas a preguntas frecuentes, saludos iniciales, o confirmaciones pueden ser manejadas eficientemente mediante respuestas estáticas codificadas directamente en el chatbot. Esto no solo reduce los costos evitando llamadas innecesarias a la API, sino que también disminuye la latencia, mejorando la respuesta del bot.

##### Implementación Práctica:

- **Definir un mapeo de respuestas:** Crea un diccionario o una base de datos de respuestas estáticas para las preguntas más comunes o los comandos más frecuentes.
- **Evaluación de la consulta:** Antes de hacer una llamada a la API, evalúa si la pregunta del usuario puede ser respondida con una respuesta predefinida.

#### Selección de Modelos: ChatGPT-3.5 vs. ChatGPT-4

Dadas las diferencias de costos entre las versiones de los modelos ChatGPT, es prudente seleccionar el modelo basado en la necesidad específica de precisión y detalle en la respuesta:

- **ChatGPT-3.5:** Este modelo es significativamente más barato y sigue siendo altamente efectivo para una amplia gama de tareas, especialmente para generar respuestas donde la precisión extrema no es crítica. Es ideal para responder preguntas basadas en un FAQ, donde las respuestas ya están bien definidas y solo necesitan ser formuladas correctamente.
- **ChatGPT-4:** Aunque más costoso, ChatGPT-4 es más adecuado para tareas que requieren una mayor comprensión del contexto o la generación de respuestas complejas y detalladas. Debe utilizarse para las interacciones más importantes, como las respuestas finales a los clientes que requieren alta calidad y precisión, o cuando la consulta involucra un entendimiento profundo y una generación de contenido novedoso.

##### Implementación Práctica:

- **Segmentación de uso:** Decide cuál modelo utilizar en función del tipo de pregunta. Si la consulta es directa o forma parte de un FAQ, utiliza ChatGPT-3.5. Para consultas que requieren creatividad, detalles específicos del contexto o calidad premium en la comunicación, opta por ChatGPT-4.



- **Precarga de FAQs:** Si el chatbot maneja muchas FAQs, considera enviar tanto la pregunta como la respuesta predefinida a ChatGPT-3.5 para reformular la respuesta de manera más natural, manteniendo bajo el costo.

Optimizar el uso de la API de ChatGPT para un chatbot de WhatsApp implica una planificación cuidadosa del tipo de respuestas y la selección del modelo apropiado para cada tipo de consulta. Implementar respuestas estáticas para las preguntas frecuentes y segmentar el uso de los modelos de ChatGPT según la complejidad y la importancia de las preguntas no solo reduce costos, sino que también mejora la eficiencia y la experiencia del usuario.

---

## 10. Errores comunes

Desarrollar chatbots de WhatsApp utilizando NodeJS es una tarea que implica varios desafíos técnicos, especialmente para los desarrolladores menos experimentados. A continuación, exploraremos los errores más comunes que puedes enfrentar, cómo evitarlos y qué hacer si te encuentras atascado en un problema complicado.

### ¿Cuáles son los errores que puedo tener?

Al desarrollar un chatbot de WhatsApp, puedes encontrarte con una serie de errores comunes, que incluyen:

1. **No instalar las dependencias necesarias:** Un error frecuente es omitir la instalación de paquetes necesarios que tu proyecto requiere para funcionar. Esto puede llevar a errores de módulo no encontrado y otros problemas relacionados al inicio de la aplicación.
2. **Conflictos de versiones:** Usar versiones incompatibles de NodeJS o de paquetes npm puede causar incompatibilidades y fallos. Es crucial asegurarse de que todas las dependencias estén en versiones que funcionen bien juntas y sean compatibles con tu versión de NodeJS.
3. **Poco manejo de errores:** No manejar adecuadamente los errores que pueden surgir durante la ejecución de tu chatbot puede llevar a fallos inesperados y a una mala experiencia del usuario. Es fundamental implementar un manejo de errores robusto para anticiparse y responder adecuadamente a posibles problemas.
4. **Pruebas insuficientes:** Cada vez que agregas una sección de código nuevo, es esencial probar todas las formas posibles en las que este código puede fallar. Muchos desarrolladores avanzan demasiado rápido sin dedicar tiempo suficiente a probar adecuadamente, lo cual puede resultar en errores no detectados que afectan a los usuarios finales.

### ¿Cómo puedo evitar errores?

Para minimizar y manejar efectivamente los errores en el desarrollo de tu chatbot, considera las siguientes prácticas:

- **Uso de Bloques Try/Catch:** Una técnica esencial en cualquier aplicación de NodeJS es el uso de bloques `try/catch` para capturar errores en tiempo de ejecución. Esto no solo previene que tu aplicación se rompa, sino que también permite responder de manera controlada. Por ejemplo:

```
try {
  // Código que puede fallar
  let response = await flowDynamic("¿Cómo puedo ayudarte?");
} catch (error) {
  console.error("Error enviando mensaje:", error);
  // Opcional: enviar mensaje de error al usuario
  await flowDynamic("Lo siento, ocurrió un error. Intenta nuevamente.");
}
```

En este ejemplo, si el método `sendMessage` falla por alguna razón, el error es capturado y manejado sin detener la ejecución de la aplicación, y se informa al usuario adecuadamente.


## Links y referencias

ai.paths23@gmail.com

Curso completo:

### Vende Tu Chatbot de Whatsapp

Este curso intensivo está diseñado para transformar a desarrolladores sin experiencia o con conocimientos básicos en expertos capaces de diseñar, implementar y vender su primer chatbot de WhatsApp. Este curso está diseñado con un enfoque práctico y estructurado, vas a aprender los fundamentos de NodeJS, las bases de los chatbots, manejo de bases de

 <https://dub.sh/bot-wpp>

### AIPaths

Si eres un entusiasta de la programación, un desarrollador en ciernes o simplemente alguien con curiosidad por la tecnología, este es tu lugar. Aquí encontrarás tutoriales de programación relacionados a Inteligencia artificial y

 [https://www.youtube.com/channel/UCkk1guGQ6C6I4\\_XJ2Pa3SiA](https://www.youtube.com/channel/UCkk1guGQ6C6I4_XJ2Pa3SiA)

### AI Paths (@Ai\_paths23) on X

Estamos para ayudar a nuestra comunidad a crecer y entender mas sobre la Inteligencia Artificial

 [https://twitter.com/Ai\\_paths23](https://twitter.com/Ai_paths23)

### GonzaSab - Overview

GonzaSab has 9 repositories available. Follow their code on GitHub.

 <https://github.com/GonzaSab>

