DISEÑO DE UN CHATBOT PARA LA OPTIMIZACIÓN DEL PROCESO DE COTIZACIONES

- **RUBY YULIANA PEÑARANDA HERNÁNDEZ**
- **DIEGO FERNANDO TANGARIFE RAMÍREZ**

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE TECNOLOGÍAS
PROGRAMA DE INGENIERÍA MECATRÓNICA

PEREIRA
2023

DISEÑO DE UN CHATBOT PARA LA OPTIMIZACIÓN DEL PROCESO DE COTIZACIONES

- **RUBY YULIANA PEÑARANDA HERNÁNDEZ**
- **DIEGO FERNANDO TANGARIFE RAMÍREZ**

TRABAJO DE GRADO PARA OPTAR AL TÍTULO DE TECNÓLOGO EN MECATRÓNICA

DIRECTOR: CARLOS ANDRÉS RODRÍGUEZ PÉREZ

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE TECNOLOGÍAS

PROGRAMA DE INGENIERÍA MECATRÓNICA

PEREIRA

2023

CONTENIDO

pág

INTRODUCCIÓN 6

1 MARCO DE REFERENCIA 7

2 INFORMACIÓN ACERCA DE LA LIBRERÍA 14

3 FUNCIONALIDAD DEL PROGRAMA 19

4 PRUEBAS DE FUNCIONALIDAD 21

5 PRESUPUESTO 26

6 CONCLUSIONES 27

7 BIBLIOGRAFÍA 28

8 ANEXOS 29

LISTA DE FIGURAS

Pág

Figura 1. Diagrama de flujo ……………...…………………………….………...10

Figura 2. Base de datos MongoDB ……………...……………………..…………...13

Figura 3. Código qr generado con qrcode-terminal………………...………………...15

Figura 4. Código para generar código qr……….……………………………………15

Figura 5. Código para capturar mensajes …………………………...…………..…16

Figura 6. Responder mensaje citando método message.reply……………………..…16

Figura 7. Responder mensaje método sendMessage………………….....………..…17

Figura 8. Código para conectar base de datos……………………....……….……18

Figura 9. Código recibir mensajes……………………………………….………19

Figura 10. Código responder mensajes……………………………………………19

Figura 11. Código para mantener sesión activa…………………..………...……20

Figura 12. Recolección de metadatos…………………………………....……20

Figura 13. Interacción con el bot parte 1……………………………………………21

Figura 14. Interacción con el bot parte 2……………………………………………22

Figura 15. Interacción con el bot parte 3……………………………………….……23

Figura 16. Interacción con el bot parte 4……………………………………….……24

Figura 17. Interacción con el bot parte 5…………………………………..…………25

LISTA DE ANEXOS

pág

Anexo A. Código para enviar y recibir mensajes ………………………..……………30

Anexo B. Código para guardar y borrar mensajes …………..……………………….36

Anexo C. Código para crear tabla de categoría en la base de datos………………..38

Anexo D. Código para crear tabla de comerciantes en la base de datos……………39

Anexo E. Código para crear tabla del mensaje recibido………………………………40

INTRODUCCIÓN

En una empresa de distribución de componentes eléctricos y electrónicos, se enfrenta una problemática que afecta la eficiencia diaria: la tediosa y lenta tarea de cotizar elementos con distintos proveedores. Ya sea separándolos por categoría, marca o elemento específico, enviar mensajes o correos para conocer el precio de los productos consume una gran cantidad de tiempo y energía. Para enviar un mensaje de broadcast por medio de WhatsApp existen inconvenientes como la cantidad de personas que reciben el mensaje. Al enviar un mismo mensaje a un gran grupo de personas, se puede reducir la velocidad del mensaje y ser confundido como un mensaje de spam, lo que puede generar que sea borrado por el destinatario. Este problema puede afectar de manera directa los procesos industriales en el área de compras de material, ya que al no ser eficiente el proceso de cotización, se puede perder rápidamente la oportunidad de obtener precios de manera inmediata.

La creación de un bot de WhatsApp para automatizar el proceso de cotización de componentes es una solución innovadora y pertinente para resolver este problema. La automatización de trabajos se ha convertido en una tendencia clave para mejorar la eficiencia de los procesos empresariales y reducir los tiempos de trabajo. Un bot de WhatsApp puede responder rápidamente y de manera precisa, ahorrando tiempo y recursos para la empresa y beneficiando a los clientes que realizan sus cotizaciones. Además, al implementar un bot de WhatsApp se liberará a los trabajadores de esta tarea monótona y repetitiva, permitiéndoles enfocarse en actividades más creativas y estratégicas dentro de la empresa.

El objetivo general de este proyecto es diseñar un Chatbot de WhatsApp para sintetizar la información y facilitar la toma de decisión en un proceso de cotización de componentes eléctricos. Para lograr este objetivo, se llevarán a cabo objetivos específicos como recopilar información acerca de la librería a utilizar, desarrollar y proporcionar el código de programación adecuado para el proyecto, y desarrollar pruebas que permitan validar la operatividad del chatbot.

La metodología que se utilizará para el desarrollo de este proyecto consiste en una investigación exhaustiva acerca de las librerías disponibles para la creación de un bot de WhatsApp. Posteriormente, se procederá con la programación del bot y se elegirá una base de datos adecuada para almacenar la información necesaria. Durante el proceso de programación se realizarán pruebas simultáneas para simular el comportamiento del cliente y asegurar que todo funciona adecuadamente. De esta manera, se garantiza que el bot de WhatsApp cumpla con los objetivos específicos y generales del proyecto.

1. MARCO DE REFERENCIA.

1.1 ESTADO ACTUAL

En los últimos años, los Chatbots han dado un paso gigantesco en un mundo que se hace cada vez más y más digital. Estos pequeños robots han estado evolucionando para interactuar mucho mejor con nosotros y darnos una experiencia inolvidable.

Ellos sin duda han llevado al comercio electrónico a un nuevo nivel y han logrado que las empresas ahorran cientos de miles de dólares en gastos de personal para el sector de atención al cliente.

Cada vez son más las empresas que crean sus bots personales y existen ciertas aplicaciones que están expandiendo su plataforma para ofrecerlos. Un ejemplo de esto es WhatsApp. [1]

Actualmente, en el ámbito del comercio electrónico, los bots ayudan a los clientes a encontrar y seleccionar productos, proporcionar información detallada sobre los productos, realizar productos y hacer seguimiento de los envíos, mejorando así la eficiencia y reduciendo los tiempos de respuesta.

En la atención al cliente, los bots pueden responder preguntar frecuentes y proporcionar información sobre políticas y procesos, también ayudar a resolver problemas a los clientes de manera más rápida y eficiente, de esta manera los bots reducen la carga de trabajo a los agentes de atención al cliente, permitiéndoles centrarse en tareas más complejas y ofreciendo una atención más personalizada a los clientes.

En el contexto del marketing, los bots de mensajes se han convertido en una herramienta esencial para interactuar con los actuales o futuros clientes, estos bots pueden enviar mensajes personalizados, ofrecer promociones y descuentos y recopilar información sobre los clientes, mejorando así la efectividad de las campañas de marketing.

Actualmente, hay varias empresas que ofrecen diferentes servicios relacionados con los chatbots, algunas de estas son:

- Brand embassy.
  Brand Embassy es la plataforma de atención al cliente en la nube mejor calificada que ofrece más de 30 canales integrados de redes sociales, mensajería instantánea, chat en vivo, correo electrónico y, lo que más nos interesa: un generador de chatbot potenciado con IA.
  Con esta herramienta podrás diseñar bots para canales populares como WhatsApp Business o Apple Business Chat.[2]

- Cliengo.
  Cliengo es una plataforma de marketing conversacional pensada para mejorar la experiencia de los usuarios. La inteligencia artificial de esta herramienta permite gestionar conversaciones personalizadas con respuestas rápidas y automáticas.
  Un factor diferencial es que el bot identifica cuando hay una oportunidad comercial y permite que la conversación sea intervenida por tu equipo comercial.[2]

- VirtualSpirits
  Esta herramienta de chatbot WhatsApp, es un desarrollo de origen belga, que realmente tiene precios de los más accesibles. Esto se debe a que es una solución íntegramente pensada en los chatbots.
  Puedes personalizar a tu agente de chat eligiendo su propio diseño, estilo, colores, imágenes, fotos de subida, añadir el logotipo, y mucho más.
  Puede elegir utilizar solo su Chatbot, Chat en Vivo o combinar ambos. Usted lo controla y elige lo más adecuado para su negocio.[2]

- Zendesk Chatbot
  Zendesk es un desarrollo que ofrece múltiples herramientas para atención al cliente. Entre estas opciones ofrece chat en vivo y mensajería para poder diseñar un bot en WhatsApp.
  Esta solución ofrece Interacción proactiva para que puedas comunicarte en tiempo real. Y funciones de análisis en vivo para conocer sobre la satisfacción del cliente y el rendimiento de los agentes.
  Con esta solución encontrarás que puedes integrar nuevas funciones vinculadas con la atención al cliente.[2]

- MessengerPeople
  Esta herramienta de chatbot para WhatsApp es una de las más fáciles de usar.
  Es sencillo construir flujos de conversación para hacer que la comunicación con los clientes sea más intuitiva.
  MessengerPeople también brinda la posibilidad de recolectar las características del usuario, una función que es realmente útil para la gestión de tus clientes.
  Las características de los clientes se archivan a medida que surjan conversaciones de chatbot. Esto permite generar un engagement más significativo con los clientes según sus necesidades y sobre cómo tu marca puede ayudarles.[2]

1.2 MARCO CONCEPTUAL

El sistema de bot de WhatsApp es un sistema automatizado que permite a las empresas interactuar con los usuarios a través de la aplicación de mensajería.

El sistema se ejecuta en una plataforma en la nube que permite la conexión entre la empresa y los usuarios a través de la API de WhatsApp Business, lo cual garantiza la integridad de los datos y la privacidad de los usuarios, cumpliendo con los estándares de seguridad de WhatsApp.

El sistema permite a las empresas enviar mensajes estandarizados a los usuarios, como por ejemplo la implementación de un menú inicial para hacer la transición más simple de segmentación de categorías. También se admite entrada de texto libre para colocar más especificaciones del producto a buscar, por lo tanto, el usuario tendrá una interacción real con el proveedor seleccionado.

El sistema está diseñado para responder automáticamente a las preguntas de los usuarios y realizar acciones, como enviar información adicional o redirigirlos a un agente humano si es necesario, brindando una experiencia personalizada y mejorando la satisfacción del cliente.

La automatización es una parte esencial del sistema de bot de WhatsApp. El control y la automatización se implementan en la solución.

A continuación, se presenta el diagrama de flujo que muestra cómo funciona el algoritmo del sistema de bot de WhatsApp. Este diagrama ilustra las diferentes etapas del proceso de interacción con los usuarios.

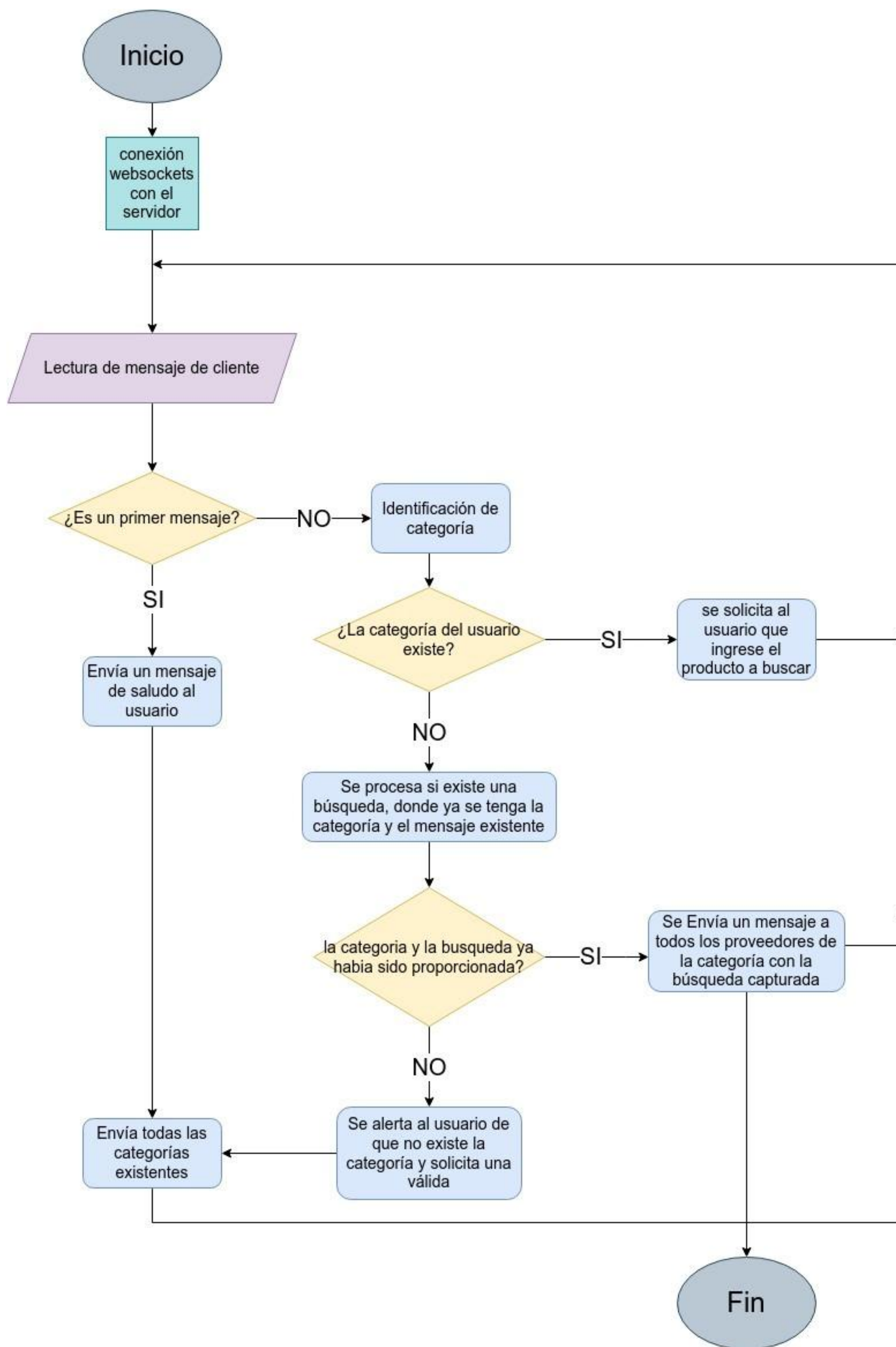

Figura 1: diagrama de flujo

1.2.1 Análisis de público

El análisis de público es un proceso importante para asegurar que un chatbot esté diseñado para satisfacer las necesidades y expectativas de los usuarios finales. Se tendrán en cuenta las siguientes características para realizar el proyecto:

- Identificación de los usuarios y comprensión de la necesidad

Los usuarios principales de este sistema automatizado son aquellos que necesiten un método de cotización rápido, que simplifique el tiempo de búsqueda, ya que muchas veces el cliente necesita un material o precio de manera urgente, por ejemplo por un daño en planta, lo cual puede parar una producción, pero también busca la economía por el hecho de que al tener un mercado tan amplio donde no se estandarizan los precios se puede encontrar con costos muy elevados o muy bajos, en conclusión los usuarios utilizarían este sistema para obtener respuestas rápidas, para esto se debe hacer investigaciones de mercado y seccionar las categorías por marcas y categorías.

- Definir los objetivos del chatbot

El objetivo del chatbot debe estar claro para que los usuarios entiendan la funcionalidad que este debe cumplir, por esta razón se hace una breve introducción al momento de utilizarlo, con esta interfaz se facilita el sistema de implementación.

- Identificar las preguntas y necesidades más frecuentes

Es importante identificar las preguntas y necesidades más frecuentes de los usuarios para asegurarse de que el chatbot esté diseñado para responder a ellas de manera efectiva.

- Definir el tono y la personalidad del chatbot

El tono y la personalidad del chatbot deben ser coherentes con la marca de la empresa y adaptados al público objetivo. Por ejemplo, si los públicos objetivos son jóvenes, el tono y la personalidad del chatbot pueden ser más relajados y divertidos.

- Evaluar la experiencia del usuario

Es importante evaluar la experiencia del usuario al interactuar con el chatbot, lo que se puede hacer a través de pruebas y evaluaciones de usuarios. La retroalimentación de los usuarios puede ayudar a mejorar la experiencia del chatbot y asegurarse de que cumple con las necesidades de los usuarios.

1.2.2 Qué es una librería

En el contexto de la programación, una librería son conjuntos de archivos de código que se utilizan para desarrollar software. Su objetivo es facilitar la programación al proporcionar funcionalidades comunes, tales como algoritmos, estructuras de datos y herramientas de interacción con el sistema operativo, que ya han sido resueltas previamente por otros programadores.

Estas librerías pueden ser utilizadas por los desarrolladores para crear aplicaciones más rápidamente, ya que no necesitan escribir código desde cero para cada funcionalidad que requiera. En cambio, pueden aprovechar las bibliotecas existentes para utilizar la funcionalidad necesaria y, de esta manera, acelerar el desarrollo del software.

Las librerías de programación también pueden mejorar la calidad del software, porque han sido desarrolladas y probadas por otros programadores y, por lo tanto, son menos propensas a contener errores. Además, al utilizar una biblioteca estándar, los desarrolladores pueden asegurarse de que su software sea compatible con una amplia variedad de plataformas y sistemas operativos.

1.2.3 Base de datos Mongodb

MongoDB es una base de datos NoSQL muy popular y flexible, y se utiliza ampliamente para la implementación de chatbots en JavaScript debido a su escalabilidad. Para utilizar MongoDB en un chatbot, es necesario instalar y configurar el controlador de MongoDB, que se puede instalar a través de Node.js (npm). Después de instalar el controlador, se puede acceder y actualizar las colecciones de MongoDB desde el código del chatbot, lo que permite almacenar y recuperar información relevante para las conversaciones con los usuarios, como información de usuarios, conversaciones, mensajes, entre otros.

MongoDB también ofrece herramientas avanzadas y funcionalidades para el procesamiento de datos y el análisis de la información almacenada en la base de datos, lo que puede ayudar a desarrollar chatbots más inteligentes y personalizados.

En este caso particular, se está utilizando MongoDB para almacenar y gestionar datos relevantes para el chatbot. La base de datos consta de cinco colecciones, cada una de las cuales contiene información específica que es relevante para el funcionamiento del chatbot. Por ejemplo, una colección contiene información de las categorías, otra colección contiene información sobre los comerciantes, y otra contiene información sobre los mensajes enviados y recibidos.

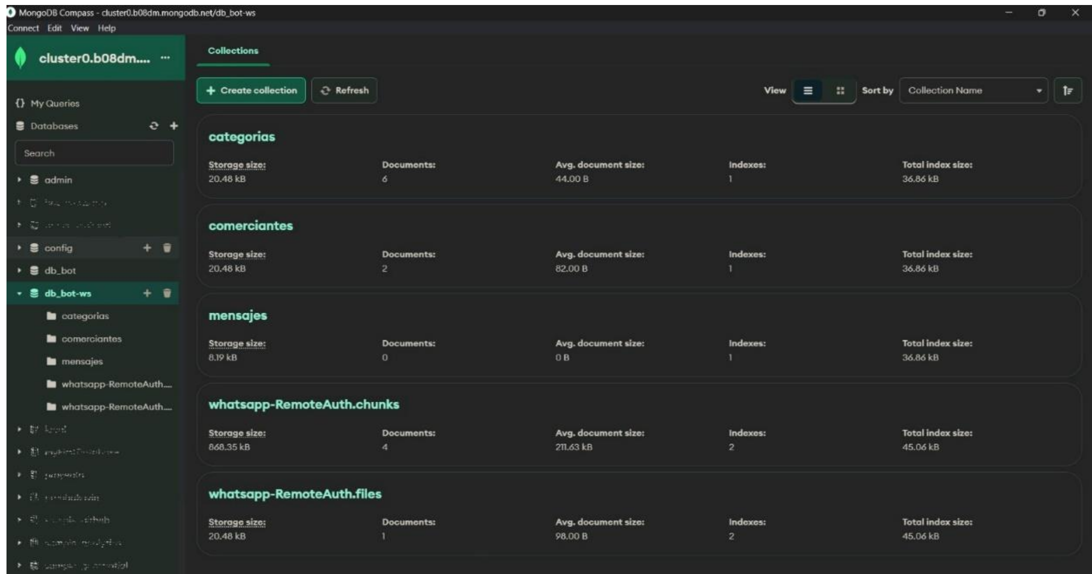

Figura 2: base de datos MongoDB

2. INFORMACION ACERCA DE LA LIBRERÍA.

2.1 LIBRERÍA WHATSAPP-WEB.JS

Whatsapp-web.js es una librería de JavaScript que permite interactuar con la interfaz web de WhatsApp, que se utiliza para acceder a la aplicación de mensajería instantánea WhatsApp en un navegador de escritorio. Esta librería ofrece una interfaz de programación de aplicaciones (API) que permite enviar y recibir mensajes, gestionar chats, crear grupos y realizar muchas otras tareas relacionadas con WhatsApp, todo ello desde un programa escrito en JavaScript.

La librería whatsapp-web.js se basa en la API no oficial de WhatsApp Web y funciona con un proceso de inicio de sesión similar al que se utiliza en el sitio web de WhatsApp.

La librería whatsapp-web.js es utilizada para crear aplicaciones personalizadas de WhatsApp que automatizan ciertas tareas, como enviar mensajes a varios destinatarios, programar mensajes, realizar tareas de seguimiento de los mensajes enviados y recibidos, envío de imágenes, audio, videos y documentos.

Cómo se mencionó anteriormente esta librería tiene un proceso de inicio similar al que se utiliza al momento de conectar whatsapp app al whatsapp web qué es el sitio de escritorio, por esta razón es necesario para empezar a trabajar con la librería poder generar un código qr que nos permita conectarnos de manera correcta y por esta razón se utiliza otra librería.

2.2 QRCODE TERMINAL

Qrcode terminal es una librería de JavaScript que permite generar códigos QR en la línea de comandos (terminal). Esta biblioteca utiliza la consola o terminal para generar el código QR y lo muestra en forma de caracteres ASCII en la pantalla.

Con "qrcode-terminal", los desarrolladores pueden crear aplicaciones de línea de comandos que generen códigos QR de manera dinámica, lo que puede ser útil en diversas situaciones, como la generación de códigos QR para compartir información de contacto, para realizar pagos, para autenticación de dos factores, y muchas otras aplicaciones en las que se necesite un código QR, en este caso se utilizará para generar el código qr y poder iniciar sesión, de esta manera el bot está listo para responder mensajes.

Figura 3: código qr generado con qrcode-terminal

2.3 MÉTODOS QUE OFRECE LA LIBRERÍA

2.3.1 QR code generation

La librería nos ofrece varios bloques de código bastante útiles como el QR code generation al ser un bot que funciona similar al whatsapp web tenemos que generar un código qr para dar autorización de autenticación.

```
const qrcode = require('qrcode-terminal');

const { Client } = require('whatsapp-web.js');
const client = new Client();

client.on('qr', qr => {
    qrcode.generate(qr, {small: true});
});

client.on('ready', () => {
    console.log('Client is ready!');
});

client.initialize();
```

Figura 4: código para generar código qr [1]

2.3.2 Listening for Messages

Después de conectarse correctamente ya podremos capturar los mensajes entrantes cada vez que llega un mensaje lo podemos obtener de la siguiente manera.

```
client.on('message', message => {
    console.log(message.body);
});
```

Figura 5: código para capturar mensajes [1]

2.3.3 replying to messages

Para responder mensajes tenemos dos alternativas, una de estas es responder citando al mensaje enviado.

Figura 6: responder mensaje citando método message.reply [1]

La otra manera de generar respuestas es enviando un mensaje normal sin citar ningún mensaje anterior. Esta forma es bastante útil porque tenemos a nuestra disposición el método sendMessage que internamente tiene el atributo message.from, este método es bastante útil porque de esta manera podremos recuperar el número del cual llega el mensaje y poder responder e incluso a múltiples cuentas.

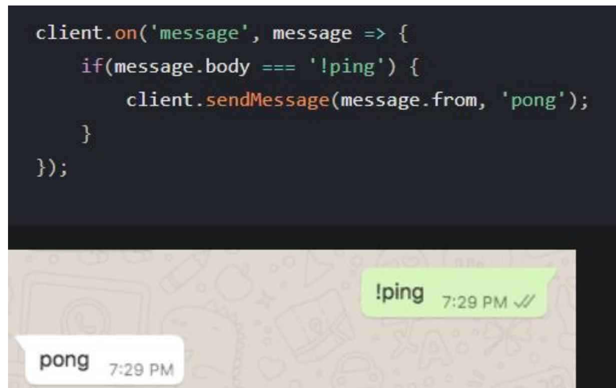

Figura 7: responder mensaje método sendMessage [1]

2.4 BASE DE DATOS

Implementar una base de datos es esencial para este proyecto porque permite almacenar y organizar información importante, como las categorías de los productos, los comerciantes asociados a esas categorías y los mensajes que se intercambian los usuarios con el bot. En este caso en particular se utilizará la base de datos MongoDB que permite almacenar datos semiestructurados y no estructurados, esto quiere decir que la información no tiene que ir estrictamente en tablas con filas y columnas, sino que se puede manejar a libertad, además para este proyecto en específico esta base de datos proporciona la interfaz para almacenar los contactos de los comerciantes asociados y también las categorías de los productos.

2.4.1 conexión base de datos y librería

Para conectar correctamente la base de datos primero necesitamos generar un link de conexión que nos proporciona mongoDB. Luego de tener ese link se guarda en una variable de entorno para hacer la conexión con la librería.

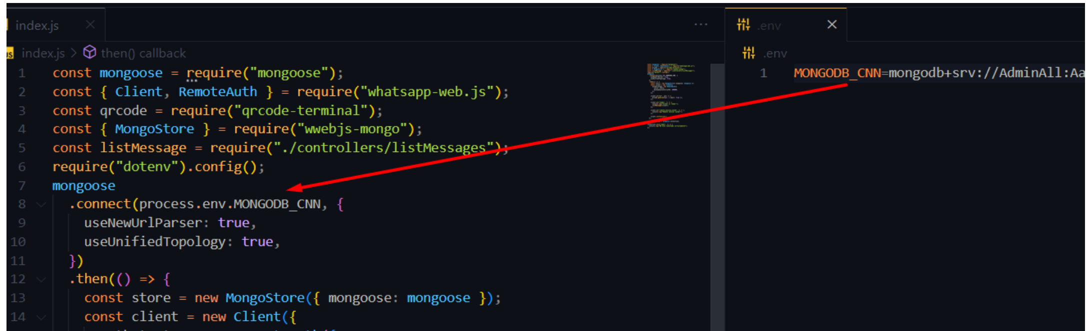

Figura 8: código para conectar base de datos

3. FUNCIONALIDAD DEL PROGRAMA.

3.1 MÉTODOS PRINCIPALES

- Con la función listMessage tenemos la posibilidad de recibir los mensajes como se puede observar en la línea siete en la variable “msg” es donde se guarda el mensaje que manda el usuario, también podemos recuperar el número de quien lo manda en la variable “forma".

```
2   const Comerciantes = require("../models/comerciantes");
3   const Categoria = require("../models/categoria");
4   const { saveMessage, deleteAllMessages } = require("./methodsBD");
5
6   const listMessage = (client) => {
7     client.on("message", async (msg) => {
8       let resultSave;
9       let resultDelete;
10      const { from, to, body } = msg;
11      try {
12        const messages = await Mensaje.find({ from });
13        const categoriasDisponibles = await Categoria.find();
```

Figura 9: código recibir mensajes

- Para responder el mensaje al usuario utilizamos la función sendMessage que nos permite enviar mensajes al mismo número que nos escribe, de esta manera después de saludar le mandamos también la lista de categorías existentes.

```
26        if (messagesOne.length === 0) {
27          saveMessage(from, 1);
28          sendMessage(
29            client,
30            from,
31            "buen dia, por favor escribe una de las siguientes opciones"
32          );
33          setTimeout(() => {
34            categoriasDisponibles.forEach(function (objeto) {
35              sendMessage(client, from, objeto.categoria);
36            });
37          }, 4000);
38          return;
39        }
40        if (messagesTwo.length === 0) {
41          if (categorias.includes(body.toLowerCase())) {
42            resultSave = saveMessage(
43              from,
44              2,
45              categoriasDisponibles[categorias.indexOf(body.toLowerCase())]._id
46            );
```

Figura 10: código responder mensajes

3.2 MANTENER SESIÓN ACTIVA

Es importante mantener la sesión activa para que cada vez que queramos inicializar el bot no haya necesidad de autenticación y de esta manera no generar múltiples cuentas.
En el siguiente bloque de código primero se conecta a la base de datos, luego comprobamos si existen credenciales de autenticación, en caso de existir se conecta directamente y si no existe genera las credenciales por primera vez.

```
13    const store = new MongoStore({ mongoose: mongoose });
14    const client = new Client({
15      authStrategy: new RemoteAuth({
16        store: store,
17        backupSyncIntervalMs: 300000,
18      }),
19    });
```

Figura 11: código para mantener sesión activa

3.3 METADATOS DE MENSAJES

En la variable “msg” tenemos todos los metadatos disponibles del usuario, en este caso estamos obteniendo el número del cliente, número del bot y el mensaje que son los datos que necesitamos para el funcionamiento del bot, pero este método también nos puede ofrecer el prefijo del país del cliente esto podría ser útil en caso de que ofrezcamos un servicio internacional y así estos metadatos puedan ser utilizada de alguna manera.

```
6   const listMessage = (client) => {
7     client.on("message", async (msg) => {
8       let resultSave;
9       let resultDelete;
10      const { from, to, body } = msg;
11      try {
12        const messages = await Mensaje.find({ from });
13        const categoriasDisponibles = await Categoria.finc
14        const categorias = categoriasDisponibles.map((cate
15          return categoria.categoria;
16        });
17        const messagesOne = messages.filter(
18          (message) => message.numberMessage === 1
```

Figura 12: recolección de metadatos

4. PRUEBAS DE FUNCIONALIDAD.

La interacción comienza con el usuario saludando al bot, quien responde con instrucciones e indica una lista de categorías disponibles en la base de datos. El usuario selecciona una categoría y, a continuación, introduce el nombre de un producto específico. El bot solicita una confirmación, como se puede apreciar en la siguiente imagen.

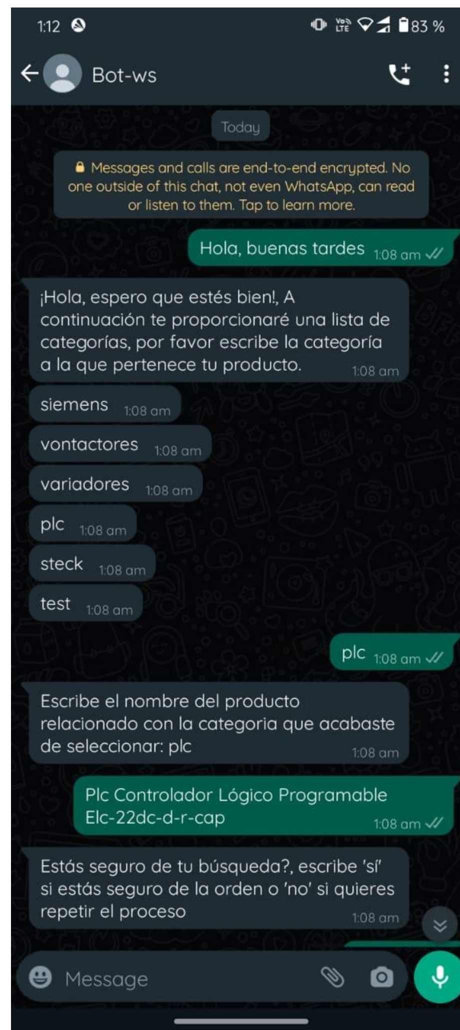

Figura 13: interacción con el bot parte 1

Una vez que el usuario confirma la solicitud, el bot responde con un mensaje indicando que la petición ha sido realizada exitosamente.

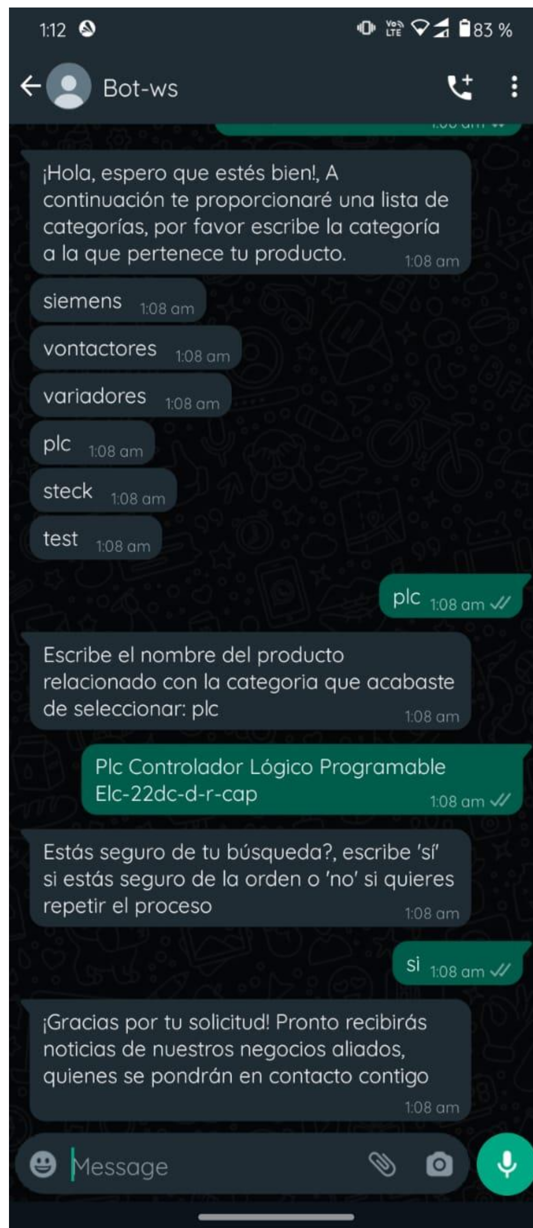

Figura 14: interacción con el bot parte 2

Continuando con el proceso, en este momento el bot envía un mensaje a los comerciantes para informarles sobre la solicitud del usuario. En este mensaje se incluye información sobre el producto que necesita el usuario, así como un contacto para comunicarse directamente con él.

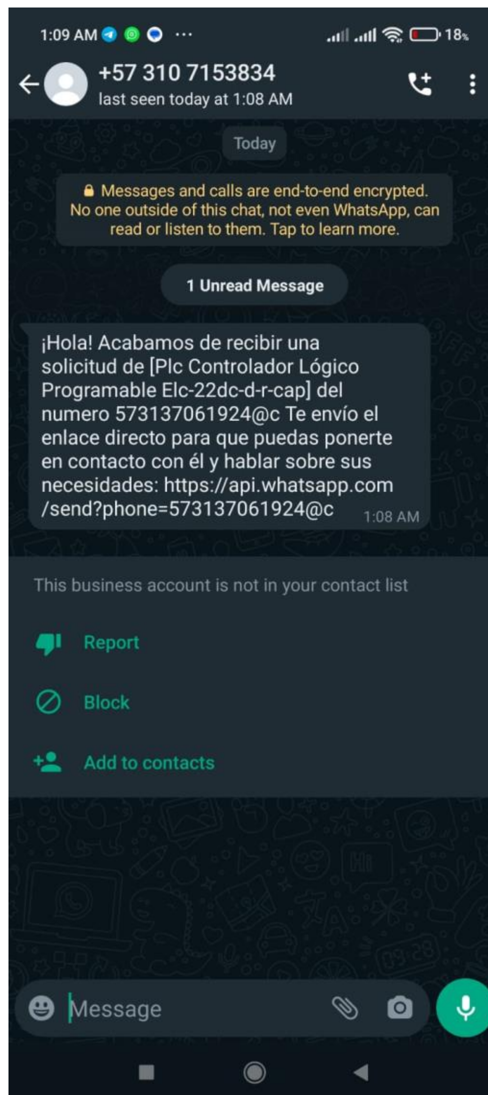

Figura 15: interacción con el bot parte 3

En ocasiones, el usuario puede no proporcionar las instrucciones de forma correcta al interactuar con el bot. En estos casos, el bot debe ser capaz de identificar el error y proporcionar una respuesta adecuada para guiar al usuario a continuar con el proceso.

Por ejemplo, a continuación, se simula un error en el que el usuario proporciona una categoría que no está disponible. En este caso, el bot detecta el error y proporciona una respuesta que ayuda al usuario a identificar el problema y corregirlo para continuar con el flujo de funcionamiento correcto.

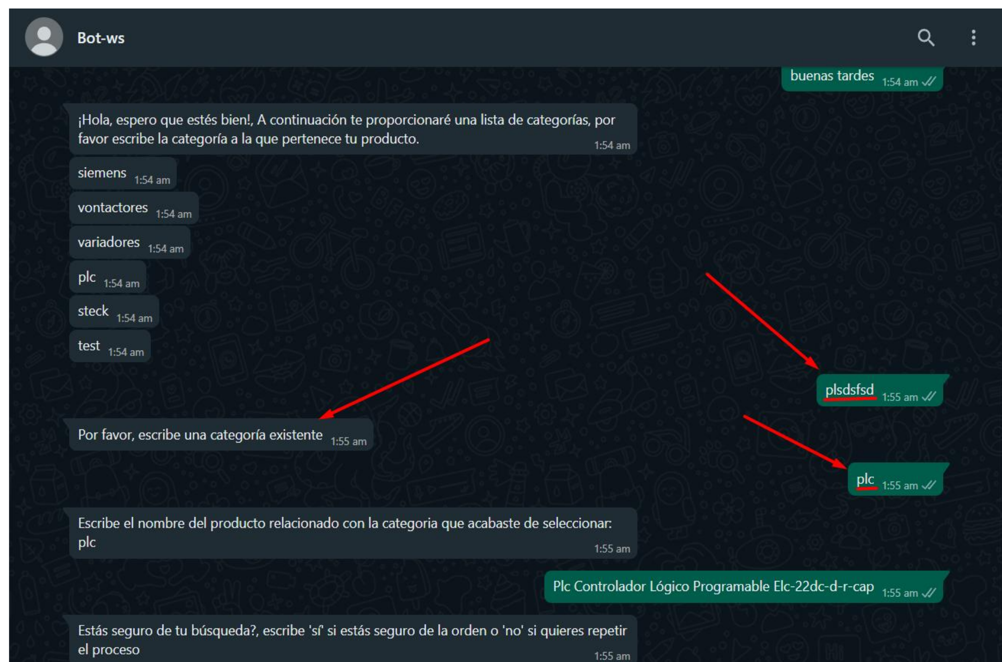

Figura 16: interacción con el bot parte 4

Otro error que puede ocurrir es cuando el bot solicita la confirmación del usuario y este responde con una opción diferente a "sí" o "no". En este caso, el bot informa al usuario que debe proporcionar una respuesta válida para poder continuar con el proceso.

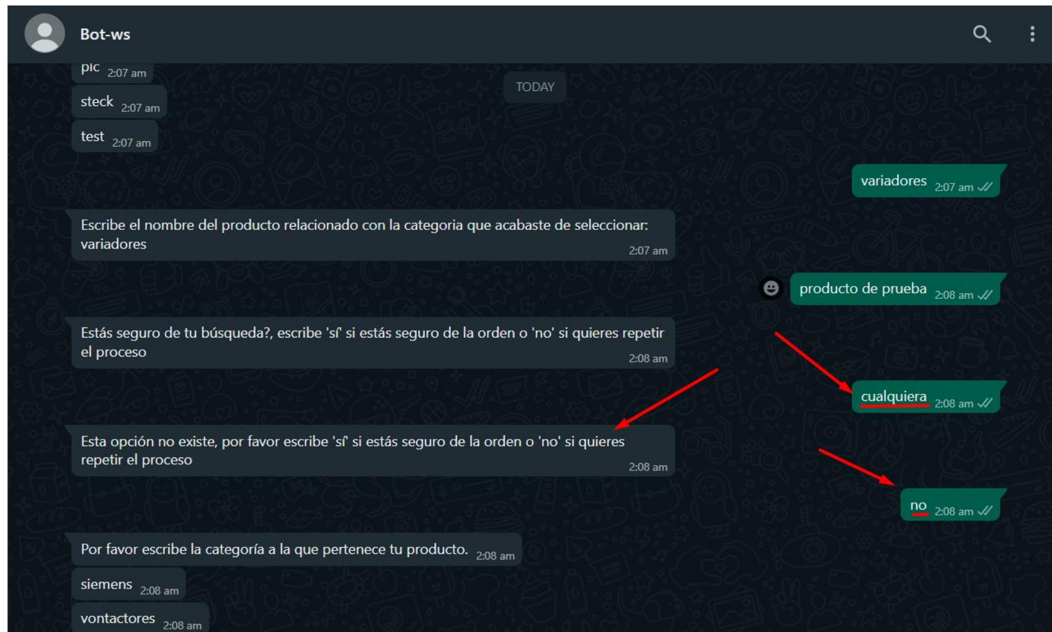

Figura 17: interacción con el bot parte 5

5. PRESUPUESTO.

Descripción	Rubro	Tiempo de ejecución	Valor unitario	subtotal
Computador	adquisición de material	1 día	$ 4.699.000	$ 4.699.000
Energía	Servicio	2 meses	$ 100.000	$ 100.000
Internet	Servicio	2 meses	$ 300.000	$ 300.000
Pruebas de funcionamiento	Transporte	2 meses	$ 5.000	$ 160.000
Servidores	adquisición de software	3 días	$ 300.000	$ 300.000
Base de datos	adquisición de software	3 días	$ 300.000	$ 300.000
Celular	adquisición de material	1 día	$ 659.900	$ 659.900
Total				$ 6.518.900

6. CONCLUSIONES.

En conclusión, el proyecto del Chatbot de WhatsApp para la automatización del proceso de cotización de componentes eléctricos y electrónicos en una empresa ha sido completado exitosamente. Se han alcanzado todos los objetivos planteados,

En primer lugar, se realizó una investigación exhaustiva para recopilar información acerca de las librerías disponibles para la creación del bot de WhatsApp, lo que permitió tomar la mejor decisión en cuanto a la tecnología a utilizar. Luego, se desarrolló el código de programación adecuado para el proyecto, garantizando su eficiencia y funcionalidad. Además, se llevaron a cabo pruebas exhaustivas para validar la operatividad del chatbot, asegurando que cumpla con los requisitos y necesidades de la empresa.

Otro aspecto relevante de este proyecto ha sido el aprendizaje obtenido a lo largo del proceso de diseño y desarrollo del Chatbot de WhatsApp. La investigación, programación y pruebas realizadas han permitido adquirir nuevos conocimientos en cuanto al uso de tecnologías de automatización, Además, se ha podido evidenciar el impacto positivo de la automatización en los procesos empresariales y la importancia de la innovación tecnológica en la mejora de la eficiencia y competitividad de una empresa.

7. BIBLIOGRAFIA.

[1] Chatcomuse, “WhatsApp, cómo funciona y qué es,” 2023. [En línea]. Disponible en: https://www.chatcomuse.com/whatsapp.html. [Consultado: 17-enero-2023].

[2]A. Vergara, “Las mejores aplicaciones de chatbot para WhatsApp en 2023,” 2023. [En línea]. Disponible en: https://blog.comparasoftware.com/aplicaciones-de-chatbot-whatsapp/. [Consultado: 20-febrero-2023].

[3] Ministerio de Ambiente y Desarrollo Sostenible, “Política de Protección de Datos Personales,” 2017. [En línea]. Disponible en: https://www.minambiente.gov.co/politica-de-proteccion-de-datos-personales/. [Consultado: 30-febrero-2023].

[4] Alcaldía Mayor de Bogotá, “Ley 527 de 1999,” 1999. [En línea]. Disponible en: https://www.habitatbogota.gov.co/transparencia/normatividad/leyes/ley-527-1999. [Consultado: 6-marzo-2023].

[5] B. M. Bogdan y C. E. Mocanu, "Whatsapp chatbot con soporte de inteligencia artificial", en Actas de la 19a Conferencia Internacional sobre Comunicaciones (COMM), Bucarest, Rumania, 2021. Enlace: https://ieeexplore.ieee.org/document/9511347

[6] A. E. M. M. Omar, M. H. Zaki y M. A. R. Alsewari, "Smart WhatsApp ChatBot: un innovador agente de conversación inteligente basado en aprendizaje profundo y procesamiento del lenguaje natural", Symmetry, vol. 13, no. 5, 2021. Enlace: https://www.mdpi.com/2073-8994/13/5/829

[7] G. Pratama, D. Darmawan y B. Nugroho, "Chatbot como servicio al cliente virtual utilizando WhatsApp Business API", en Actas de la 5a Conferencia Internacional sobre Informática y Computación (ICIC), Yakarta, Indonesia, 2021, pp. 1-4. Enlace: https://ieeexplore.ieee.org/document/9469693

8. ANEXOS.

Anexo A: código para enviar y recibir mensajes

```
const Mensaje = require("../models/messages");
const Comerciantes = require("../models/comerciantes");
const Categoria = require("../models/categoria");
const { saveMessage, deleteAllMessages } = require("./methodsBD");
const listMessage = (client) => {
  client.on("message", async (msg) => {
    let resultSave;
    let resultDelete;
    const { from, to, body } = msg;
    try {
      const messages = await Mensaje.find({ from });
      const categoriasDisponibles = await Categoria.find();
      const categorias = categoriasDisponibles.map((categoria) => {
        return categoria.categoria;
      });
      const messagesOne = messages.filter(
        (message) => message.numberMessage === 1
      );
      const messagesTwo = messages.filter(
        (message) => message.numberMessage === 2
      );
      const messagesThree = messages.filter(
        (message) => message.numberMessage === 3
      );
      if (messagesOne.length === 0) {
        saveMessage(from, 1);
        sendMessage(
          client,
          from,
```

```
        "¡Hola, espero que estés bien!, A continuación te proporcionaré una lista de categorías, por favor escribe la categoría a la que pertenece tu producto."
      );
      setTimeout(() => {
        categoriasDisponibles.forEach(function (objeto) {
          sendMessage(client, from, objeto.categoria);
        });
      }, 4000);
      return;
    }
    if (messagesTwo.length === 0) {
      console.log(categorias,body)
      if (categorias.includes(body.toLowerCase())) {
        resultSave = saveMessage(
          from,
          2,
          categoriasDisponibles[categorias.indexOf(body.toLowerCase())]._id
        );
        resultSave
          ? sendMessage(
              client,
              from,
              `Escribe el nombre del producto relacionado con la categoria que acabaste de seleccionar: ${body.toLowerCase()}`
            )
          : sendMessage(
              client,
              from,
              "Ocurrió un problema al guardar la opción, por favor vuelve a seleccionar"
            );
      } else {
        sendMessage(
```

```
          client,
          from,
          "Por favor, escribe una categoría existente"
        );
      }
      return;
    }
    if (messagesThree.length === 0) {
      resultSave = saveMessage(from, 3, false, body);
      resultSave
        ? sendMessage(
            client,
            from,
            "Estás seguro de tu búsqueda?, escribe 'sí' si estás seguro de la orden o
'no' si quieres repetir el proceso"
          )
        : sendMessage(
            client,
            from,
            "Ocurrió un problema al guardar la búsqueda, por favor vuelve a escribirla."
          );
      return;
    }
    switch (body.toLowerCase()) {
      case "si":
        const messageCategoria = await Mensaje.find({
          from,
          numberMessage: 2,
        });
        const messageBusqueda = await Mensaje.find({
          from,
```

```
  numberMessage: 3,
});
console.log(messageBusqueda, messageCategoria);
const comerciantes = await Comerciantes.find({
  categoria: messageCategoria[0].categoria,
});
resultDelete = await deleteAllMessages(from);
console.log(comerciantes, resultDelete)
if (resultDelete) {
  comerciantes.forEach(function (objeto) {
    const indiceArroba = from.indexOf("@");
    const numeroSinPrefijo = from
      .replace("57", "")
      .substring(0, indiceArroba);
    sendMessage(
      client,
      objeto.number,
      `¡Hola! Acabamos de recibir una solicitud de [${messageBusqueda[0].busqueda}] del numero 57${numeroSinPrefijo} Te envío el enlace directo para que puedas ponerte en contacto con él y hablar sobre sus necesidades: https://api.whatsapp.com/send?phone=57${numeroSinPrefijo}`
    );
  });
  sendMessage(
    client,
    from,
    "¡Gracias por tu solicitud! Pronto recibirás noticias de nuestros negocios aliados, quienes se pondrán en contacto contigo"
  );
} else {
  sendMessage(
    client,
```

```
            from,
            "Ocurrió un error, por favor escribe de nuevo 'sí' si estás seguro de la orden o 'no' si quieres repetir el proceso"
          );
        }
        break;
      case "no":
        resultDelete = deleteAllMessages(from);
        if (resultDelete) {
          const categoriasDisponibles = await Categoria.find();
          saveMessage(from, 1);
          sendMessage(
            client,
            from,
            "Por favor escribe la categoría a la que pertenece tu producto."
          );
          setTimeout(() => {
            categoriasDisponibles.forEach(function (objeto) {
              sendMessage(client, from, objeto.categoria);
            });
          }, 4000);
        }
        break;

      default:
        sendMessage(
          client,
          from,
          "Esta opción no existe, por favor escribe 'sí' si estás seguro de la orden o 'no' si quieres repetir el proceso"
        );
        break;
```

```
      }
    } catch (error) {
      console.log(error);
    }
  });
};
const sendMessage = (client, to, msg) => {
  try {
    client.sendMessage(to, msg);
    return true;
  } catch (error) {
    return false;
  }
};
module.exports = listMessage;
```

Anexo B: código para guardar y borrar mensajes

```
const Mensaje = require("../models/messages");
const saveMessage = async (from, number, categoria, busqueda) => {
  let newMessage;
  try {
    if (categoria) {
      newMessage = new Mensaje({
        from,
        numberMessage: number,
        categoria,
      });
    } else if (busqueda) {
      newMessage = new Mensaje({
        from,
        numberMessage: number,
        busqueda,
      });
    } else {
      newMessage = new Mensaje({
        from,
        numberMessage: number,
      });
    }
    await newMessage.save();
    return true;
  } catch (error) {
    console.log(error);
    return false;
  }
};

const deleteAllMessages = async (from) => {
  try {
    await Mensaje.deleteMany({ from: from });
    return true;
  } catch (error) {
    console.log(error);
    return false;
  }
};
```

```
module.exports = {
  saveMessage,
  deleteAllMessages,
};
```

Anexo C: código para crear tabla de categoría en la base de datos

```
const {Schema,model} = require('mongoose');
const categoriaSchema = new Schema({
    categoria:  {
        type: String,
        required: true
    },
}
);
categoriaSchema.method('toJSON', function(){
    const { __V, _id, ...object} = this.toObject();
    object.ctg = _id;
    return object;
})
module.exports = model('Categoria', categoriaSchema);
```

Anexo D: código para crear tabla de comerciantes en la base de datos

```
const {Schema,model} = require('mongoose');
const comercieanteSchema = new Schema({
    number:  {
        type: String,
        required : true,
    },
    categoria:  {
        type: Schema.Types.ObjectId,
        ref: 'Categoria',
        required: true
    },
    pago: {
        type: Boolean,
        required : true,
    }
}
);
comercieanteSchema.method('toJSON', function(){
    const { __V, _id, ...object} = this.toObject();
    object.cmr = _id;
    return object;
})
module.exports = model('Comerciante', comercieanteSchema);
```

Anexo E: código para crear tabla del mensaje recibido en la base de datos

```
const {Schema,model} = require('mongoose');
const Mensaje = new Schema({
   from:  {
      type: String,
      required : true,
   },
   numberMessage:{
      type: Number,
      required : true,
   },
   categoria:  {
      type: Schema.Types.ObjectId,
      ref: 'Categoria',
   },
   busqueda:  {
      type: String
   },
},
{
   timestamps : true
});
Mensaje.method('toJSON', function(){
   const { __V, ...object} = this.toObject();
   return object;
})
module.exports = model('Mensaje', Mensaje);
```

Anexo e: código para inicializar el programa

```
const mongoose = require("mongoose");
const { Client, RemoteAuth } = require("whatsapp-web.js");
```

```
const qrcode = require("qrcode-terminal");
const { MongoStore } = require("wwebjs-mongo");
const listMessage = require("./controllers/listMessages");
require("dotenv").config();
mongoose
  .connect(process.env.MONGODB_CNN, {    // se conecta con las varibles de entorno
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then(() => {
    const store = new MongoStore({ mongoose: mongoose });  // capturar la sesion
    const client = new Client({
      authStrategy: new RemoteAuth({
        store: store,                  // mandando la utenticacion para whatsap web
        backupSyncIntervalMs: 300000,
      }),
    });
    // qr
    client.on("qr", (qr) => {
      qrcode.generate(qr, { small: true });
    });

    client.on("ready", () => {
      console.log("Client is ready!");
      listMessage(client);
    });

    client.on("remote_session_saved", () => {
      console.log("Remote session saved!");
    });
```

```
    client.initialize();   // inicializa internamente el bot despues del cliente is ready
  });

const connection = mongoose.connection;

connection.once("open", () => {
  console.log("DB esta conectada correctamente");
});
```