

Generador de bots para whatsapp con python

**Desarrollo de una aplicación intuitiva para la
creación de bots automáticos para WhatsApp.**

Luis Fernando Moreno Gonzalez

Estudiante GS DAM

Índice

Contenido

I.	Resumen.....	4
II.	Índice de figuras	¡Error! Marcador no definido.
III.	Introducción	5
A.	Contexto	5
B.	Motivación	6
C.	Objetivos	6
D.	Tecnologías usadas.....	6
IV.	Análisis del sistema	7
1.	Sistema inicial.....	7
2.	Catálogo de requisitos del sistema	7
1.	Requisitos funcionales.....	7
a.	Compatibilidad con la última versión de WhatsApp :	7
b.	Creación de bots automáticos para múltiples números de teléfono:	8
c.	Personalización del comportamiento del bot	8
d.	Activación/Desactivación del bot de manera sencilla.....	8
e.	Interfaz intuitiva y fácil de usar	8
2.	Requisitos no funcionales	8
a.	Calidad del software.....	8
b.	Eficiencia en el uso de recursos	9
c.	Usabilidad y accesibilidad	9
d.	Seguridad de la información	9
V.	Diseño de la solución.....	10
A.	Arquitectura	10
B.	Diseño de la interfaz de usuario.....	12
C.	Diseño de la base de datos.....	21
D.	Desarrollo de la funcionalidad de la aplicación.....	24
E.	Opciones de personalización.....	26
VI.	Implementación	27
A.	Herramientas y tecnologías.....	27
B.	Desarrollo del proyecto.....	28

C.	Pruebas y depuración.....	29
VII.	Evaluación y resultados.....	30
A.	Evaluación de los requisitos	30
B.	Resultados obtenidos.....	31
C.	Análisis de los resultados	33
D.	Ampliaciones y mejoras	33
E.	Estimación del tiempo empleado.....	33
F.	Valoración personal.....	33
VIII.	Bibliografía:	34
	Documentación sobre Python.....	34
	Documentación sobre paquetes de Python.....	34
	Documentación sobre React Native.....	35
	Documentación sobre paquetes de Node.js.....	35
	Documentación sobre autenticación con Google	35
	Documentación sobre manejo de mensajes de Whatsapp con Twilio	36
	Tutoriales de Youtube	36
I.	Anexo I.....	37
A.	Código MySQL para la creación de la base de datos.....	37
B.	Elementos del BackEnd	48
a.	Clases usadas para el modelo	48
b.	Mapeo de rutas de la API	56
c.	Controladores.....	61
d.	Elementos auxiliares	79
e.	Script que ejecuta el bot y la comunicación por WhatsApp	81
f.	Código base que ejecuta el servidor con Flask.....	83
C.	Elementos del FrontEnd	84
a.	Pantalla base que inicia la app	84
b.	Pantallas de la app	89
c.	Componentes	115
d.	Funciones de rutas	120
II.	Anexo II: Justificación de las pruebas de funcionamiento	130
	Pruebas de inicio de sesión:	130
	Prueba con un primer usuario:	130
	Prueba con un segundo usuario:.....	131

Pruebas de navegación por la página:	132
Carga de Home screen:	132
Carga de bots del usuario:.....	133
Selección de un bot:	134
Selección de las respuestas automáticas del bot:.....	135
Selección de una respuesta automática:	136
Creación de una nueva respuesta automática:.....	137
Eliminación de una respuesta automática:.....	140
Edición de una respuesta automática:	141
Creación de nuevo bot:	143
Eliminación de bot:.....	144
Edición de bot:.....	146
Pruebas de funcionamiento del bot:.....	148
Enviando mensajes de WhatsApp al número configurado en Twilio:	149

I. Resumen

El presente proyecto consistió en el desarrollo de una aplicación móvil y un sistema backend para la creación y gestión de bots automáticos de WhatsApp. El objetivo principal fue proporcionar a los usuarios una plataforma intuitiva y fácil de usar que les permitiera configurar y personalizar el comportamiento de bots de acuerdo a sus necesidades.

La aplicación permite a los usuarios crear bots automáticos para WhatsApp. Los usuarios pueden personalizar el comportamiento de los bots mediante la definición de respuestas automáticas, establecimiento de horarios de actividad y definición de palabras clave para activar el bot.

Se priorizó la eficiencia en el uso de recursos, y durante las evaluaciones realizadas se confirmó que la aplicación cumplió con este objetivo, brindando tiempos de respuesta rápidos y una interfaz fluida. La usabilidad fue otro aspecto destacado, con una interfaz intuitiva y fácil de navegar, diseñada para que los usuarios, independientemente de sus conocimientos técnicos, pudieran utilizar la aplicación sin dificultades.

En resumen, el proyecto logró desarrollar una aplicación móvil y un sistema backend que cumplió con gran parte de los requisitos funcionales y no funcionales establecidos.

II. Introducción

Breve presentación del proyecto y sus objetivos:

El proyecto consiste en desarrollar una aplicación utilizando las tecnologías Python y React Native para el back end y el front end respectivamente, que permita a los usuarios crear bots automáticos para WhatsApp de manera sencilla y rápida. La interfaz de usuario debe ser intuitiva y fácil de usar, permitiendo a los usuarios configurar y personalizar el comportamiento del bot de acuerdo a sus necesidades.

La aplicación debe permitir a los usuarios introducir diferentes parámetros para personalizar el comportamiento del bot, como por ejemplo, las respuestas automáticas que debe enviar, el horario en que está activo, las palabras clave que deben activar el bot, la capacidad de programar mensajes automatizados para ser enviados en un momento determinado, entre otras cosas. También se debe incluir un sistema de seguimiento para que los usuarios puedan monitorear las interacciones del bot con los contactos de WhatsApp.

Además de esto, se debe incluir una interfaz de administrador para permitir a los usuarios gestionar varios bots al mismo tiempo. En resumen, el proyecto busca facilitar el desarrollo de bots para WhatsApp mediante una plataforma intuitiva, lo que permitirá a los usuarios automatizar tareas y mejorar su eficiencia en el manejo de sus interacciones con clientes o contactos en WhatsApp.

A. Contexto

En la actualidad, la comunicación instantánea a través de aplicaciones de mensajería es una herramienta esencial en la vida cotidiana de muchas personas. Una de las aplicaciones más utilizadas en todo el mundo es WhatsApp, que permite a los usuarios enviar y recibir mensajes, realizar llamadas y compartir archivos. Debido a la gran cantidad de usuarios activos, existe una necesidad creciente de automatizar procesos y tareas repetitivas para mejorar la eficiencia y la productividad.

El proyecto que se llevará a cabo consiste en el desarrollo de una aplicación para crear bots automáticos en WhatsApp. La aplicación permitirá a los usuarios personalizar el comportamiento de los bots para adaptarse a sus necesidades específicas, como responder automáticamente a mensajes entrantes, activarse en horarios específicos, etc.

El desarrollo de esta aplicación es de gran importancia en el contexto actual, ya que aporta soluciones innovadoras para las necesidades de automatización en la comunicación a través de WhatsApp, especialmente en el contexto de negocios que utilicen WhatsApp como medio de comunicación con sus clientes.

B. Motivación

Mis motivaciones para realizar este proyecto vienen dadas por varios factores, primero está el deseo de profundizar mis conocimientos en las tecnologías usadas en clase y aplicarlas en un proyecto real que además resulta algo innovador con respecto a lo ya aprendido. Por otro lado, conozco de primera mano personas que desarrollan una parte de su actividad económica en línea y una de las principales herramientas que utilizan para comunicarse con sus clientes es WhatsApp. Considero que esta aplicación puede llegar a ser una herramienta clave para su negocio y el de muchas otras personas en una situación similar o con modelos de negocio parecidos; por ello, tener experiencia desarrollando una aplicación relacionada con WhatsApp sería valioso. Con este proyecto, quiero combinar mis intereses y habilidades para crear algo útil y aplicable en la vida real.

C. Objetivos

Descripción detallada de los objetivos generales y específicos del proyecto. Hasta ahora se contemplan los siguientes objetivos:

1. Facilitar la creación de bots automáticos para WhatsApp a los usuarios sin conocimientos técnicos avanzados: El objetivo principal de este proyecto es brindar una herramienta fácil de usar para aquellos que deseen crear bots automáticos para WhatsApp, sin necesidad de tener conocimientos profundos de programación o desarrollo de software. Esto se logrará mediante una interfaz de usuario intuitiva y una serie de opciones de configuración preestablecidas que permitirán a los usuarios crear y personalizar sus bots de manera sencilla.
2. Proporcionar una herramienta intuitiva y fácil de usar para la creación de bots automáticos para WhatsApp. Para esto se necesita diseñar una interfaz de usuario amigable para que los usuarios puedan crear, configurar y personalizar sus bots con facilidad. Esto podría necesitar incluir una serie de tutoriales y ayudas en línea para guiar a los usuarios a través del proceso de creación de bots, sin embargo quizás el alcance de este proyecto sólo llegue hasta unos tutoriales sencillos.
3. Ofrecer una variedad de opciones de personalización para que los usuarios puedan adaptar el comportamiento del bot a sus necesidades. Esto incluirá la posibilidad de definir respuestas automáticas, programar mensajes automatizados, establecer horarios de actividad, definir palabras clave para activar el bot, entre otras opciones. De esta manera los usuarios podrán crear bots automáticos altamente personalizados y adaptados a sus necesidades específicas.

D. Tecnologías usadas

En este proyecto se han utilizado diversas tecnologías para el desarrollo de una aplicación de mensajería en línea. Para la implementación de la interfaz de usuario, se ha utilizado React Native. Para la gestión de la base de datos, se ha utilizado MySQL. Para el desarrollo de la funcionalidad de la aplicación, se ha utilizado Python.

React Native, MySQL y Python son tecnologías muy populares y compatibles entre sí, y se pueden utilizar juntas para desarrollar una aplicación móvil compleja y escalable. React Native es un marco de desarrollo de aplicaciones móviles que se basa en React, y permite crear aplicaciones nativas para iOS y Android con una única base de código. MySQL es un sistema de gestión de bases de datos relacionales ampliamente utilizado, que permite almacenar y gestionar grandes cantidades de datos de manera eficiente y segura. Python es un lenguaje de programación de alto nivel que permite un desarrollo rápido y eficiente de aplicaciones, y ofrece una amplia gama de bibliotecas y herramientas para trabajar con bases de datos.

En resumen, estas tres tecnologías son compatibles y se pueden usar juntas para desarrollar una aplicación móvil con una interfaz de usuario atractiva, una base de datos robusta y una funcionalidad de aplicación avanzada. Por lo tanto, se puede decir que esta combinación de tecnologías es adecuada para el desarrollo del proyecto.

III. Análisis del sistema

Este apartado tiene como objetivo identificar y documentar los requisitos y necesidades del proyecto, partiendo de una evaluación del sistema inicial desde el que se empieza a desarrollar el proyecto, y evaluando también requisitos tanto funcionales como no funcionales. Los requisitos funcionales se refieren a lo que el sistema debe hacer, mientras que los requisitos no funcionales hablan de las propiedades del sistema que deben cumplirse. También es necesario identificar los actores, es decir, las entidades que van a estar en contacto con la aplicación.

1. Sistema inicial

En este caso se parte de un sistema en el que no se ha creado ninguna parte en concreto y sólo consiste en una idea que se encuentra en fase de desarrollo y de creación de un plan de acción. Sin embargo, teniendo en cuenta los objetivos previamente establecidos, podemos decir que el sistema inicial tiene como objetivo desarrollar una aplicación móvil que permita la gestión de las comunicaciones a través de WhatsApp de manera más eficiente y organizada.

2. Catálogo de requisitos del sistema

1. Requisitos funcionales

Los requisitos funcionales son una lista de funcionalidades que el sistema debe cumplir para cumplir con las expectativas del usuario. En este caso, los requisitos funcionales incluyen:

a. Compatibilidad con la última versión de WhatsApp :

El sistema debe ser compatible con la última versión de WhatsApp y debe ser capaz de funcionar sin problemas en ella, para garantizar que la funcionalidad de los bots automáticos se mantenga. Esto incluiría la compatibilidad con cualquier nueva característica o actualización de seguridad de WhatsApp

b. Creación de bots automáticos para múltiples números de teléfono:

Debe ser posible crear bots automáticos para múltiples números de teléfono desde la misma aplicación.

Es decir que los usuarios deben tener la capacidad de crear bots automáticos para varios números de teléfono desde una sola instancia de la aplicación. Esto permitirá a los usuarios gestionar varios bots automáticos desde una sola interfaz.

También se puede incluir una función para duplicar un bot, o adaptarlo para que el mismo bot funcione para varios números de teléfono, en caso de necesidad.

c. Personalización del comportamiento del bot

El sistema debe permitir a los usuarios personalizar el comportamiento del bot. mediante la introducción de diferentes parámetros

La aplicación debe proporcionar una variedad de opciones de personalización para que los usuarios puedan adaptar el comportamiento del bot a sus necesidades específicas. Esto incluiría opciones mencionadas antes como definir respuestas automáticas, programar mensajes automatizados, establecer horarios de actividad, definir palabras clave para activar el bot, entre otras opciones.

d. Activación/Desactivación del bot de manera sencilla

La aplicación debe proporcionar una opción sencilla y fácil de usar para activar y desactivar el bot automáticamente, esto permitirá a los usuarios controlar en todo momento su bot.

De igual forma, se puede incluir una opción para programar el encendido o apagado automático de los bots.

e. Interfaz intuitiva y fácil de usar

La interfaz de usuario debe ser diseñada de manera intuitiva, de modo que cada usuario, independientemente de su nivel de experiencia en programación o desarrollo de software pueda usar fácilmente la aplicación, incluso si sus conocimientos sobre programación son casi nulos. Esto incluiría una navegación clara y lógica, iconos y etiquetas intuitivas, tutoriales, y una interfaz de usuario fácil de entender y utilizar.

En caso de un futuro desarrollo del proyecto, también hay que tener en cuenta la publicación de tutoriales y ayudas en línea.

2. Requisitos no funcionales

Los requisitos no funcionales tratan sobre las propiedades del sistema que deben cumplirse, independientemente de las funcionalidades específicas que proporcione. En este caso, los requisitos no funcionales incluyen:

a. Calidad del software

El software debe tener una calidad adecuada, que garantice que funcione correctamente y cumpla con las expectativas del usuario.

b. Eficiencia en el uso de recursos

El sistema debe ser eficiente en el uso de recursos en lo posible, lo que significa que debe consumir una cantidad mínima de recursos del sistema.

c. Usabilidad y accesibilidad

El sistema debe ser fácil de usar y accesible para todos los usuarios, independientemente de sus habilidades técnicas.

d. Seguridad de la información

El sistema debe garantizar la seguridad de la información de los usuarios, así como su privacidad. Para esto se deben implementar medidas de seguridad adecuadas para proteger los datos y prevenir accesos no autorizados.

La seguridad de la información es un aspecto crítico que afecta a la calidad del sistema, pero no es una función directa que realiza el sistema, es decir, no es el objetivo directo de la aplicación. En lugar de eso, es una característica general que debe cumplirse en el diseño y la implementación del sistema, y se refiere a cómo el sistema protege la información que maneja. Por lo tanto, se considera un requisito no funcional.

IV. Diseño de la solución

En este apartado se describen los detalles de cómo se implementará la solución propuesta para satisfacer los requisitos mencionados en el análisis previo. Este apartado incluye información sobre la arquitectura general, el diseño de la interfaz de usuario, la base de datos, el desarrollo de la funcionalidad de la aplicación y las opciones de personalización. Este apartado debe incluir suficiente información para que un tercero pueda comprender cómo se construirá y funcionará el sistema.

A. Arquitectura

La estructura general de la aplicación consta de varios componentes que se comunican entre sí para formar la solución completa. A continuación se describe cada uno de estos componentes y las relaciones entre ellos:

1. Componente de Frontend (React Native):

Pantallas: Se crean pantallas utilizando componentes de React Native para mostrar la interfaz de usuario al usuario final. Estas pantallas incluyen la creación de bots automáticos, la personalización del comportamiento del bot, la gestión de interacciones y cualquier otra funcionalidad requerida.

Componentes reutilizables: Se utilizan componentes reutilizables para elementos comunes de la interfaz de usuario, como listas.

Navegación: Se implementa la navegación entre las diferentes pantallas de la aplicación utilizando bibliotecas como React Navigation.

2. Componente de Backend (Python y Flask):

Controladores: Se implementan controladores en Python utilizando Flask para manejar las solicitudes HTTP del frontend. Estos controladores interactúan con los modelos de datos y realizan las operaciones necesarias en la base de datos.

Rutas: Se definen las rutas en Flask para asociar las URL con los controladores correspondientes.

Modelos de datos: Se definen modelos de datos utilizando SQLAlchemy para representar las entidades del sistema, como usuarios, bots, respuestas automáticas, plantillas, palabras clave, etc. Estos modelos de datos se utilizan para realizar operaciones en la base de datos, como consultas, inserciones, actualizaciones y eliminaciones.

Comunicación con la API de WhatsApp: Se establece una comunicación con la API de WhatsApp para enviar y recibir mensajes a través de los bots automáticos. Esto implica el uso de bibliotecas o soluciones de terceros compatibles con la API de WhatsApp.

3. Componente de Base de Datos (MySQL):

Base de datos: Se utiliza una base de datos relacional MySQL para almacenar la información relacionada con los usuarios, bots, respuestas automáticas, plantillas, palabras clave, etc. Se definen tablas y relaciones entre ellas para mantener la integridad y consistencia de los datos.

Interacción con el Backend: El componente de backend interactúa con la base de datos a través de consultas y operaciones definidas en los modelos de datos utilizando SQLAlchemy.

4. API de WhatsApp:

Comunicación con los Bots Automáticos: La API de WhatsApp permite la comunicación entre los bots automáticos y los contactos de WhatsApp. Para poder gestionar la comunicación con la API de WhatsApp, se utiliza el servicio de terceros Twilio. Los mensajes enviados por los contactos se reciben a través de la API y se envían al backend para su procesamiento. Del mismo modo, las respuestas automáticas generadas por los bots se envían a través de la API para ser entregadas a los contactos correspondientes.

Además de los componentes mencionados anteriormente, también se incluyen otros elementos de seguridad necesarios para autenticarse con los servicios de Twilio, que garantizan la privacidad y protección de los datos de los usuarios. Esto implica el uso de tokens de acceso y validación de usuarios.

En resumen, la aplicación se compone de un frontend desarrollado en React Native, un backend desarrollado en Python y Flask, una base de datos relacional y la comunicación con la API de WhatsApp. Estos componentes se comunican entre sí para permitir la creación, personalización y gestión de bots automáticos.

B. Diseño de la interfaz de usuario

El diseño gráfico y la experiencia de usuario (UX) de la aplicación se centran en proporcionar una interfaz intuitiva y fácil de usar para la gestión de bots automáticos de WhatsApp. El objetivo es permitir a los usuarios configurar y personalizar el comportamiento de sus bots de manera eficiente y sin complicaciones. A continuación se detallan las funciones y características clave de la interfaz, así como las diferentes vistas de navegación y la interacción con el usuario:

1. Pantalla de login

Consiste en una pantalla de autenticación con Google

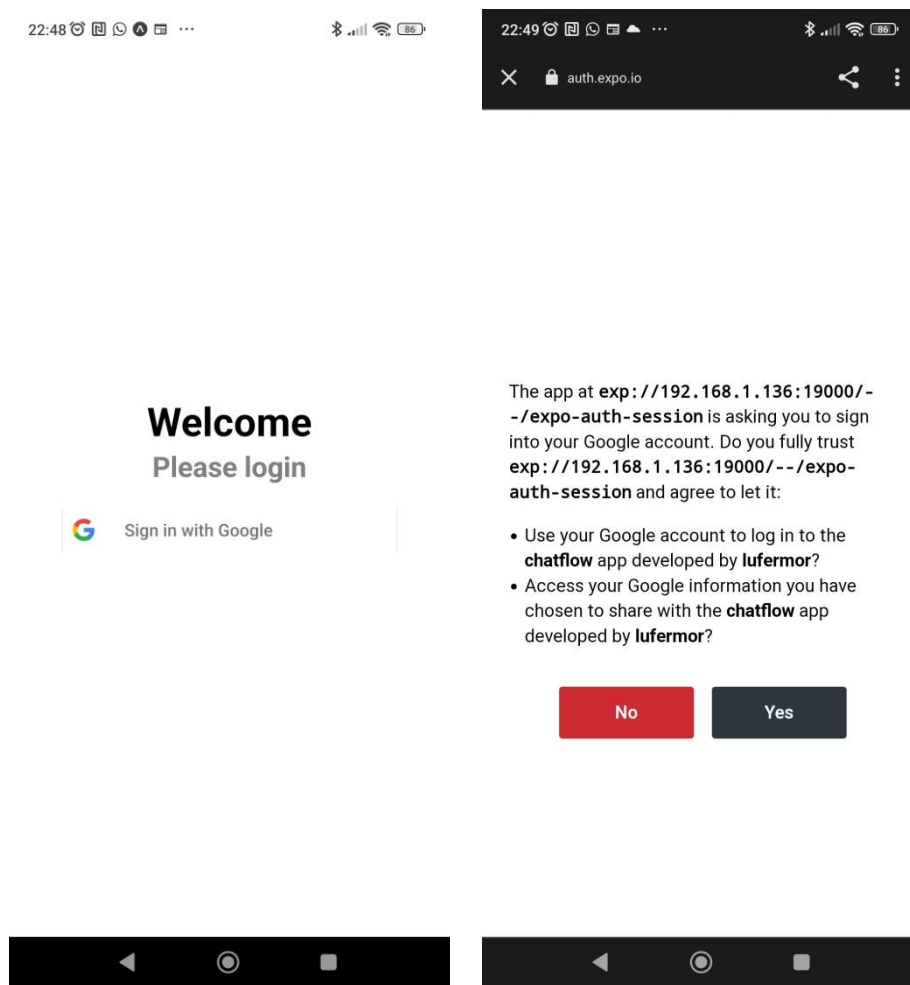


Ilustración 1: Pantalla Inicial

Ilustración 2: Pantalla información de autenticación

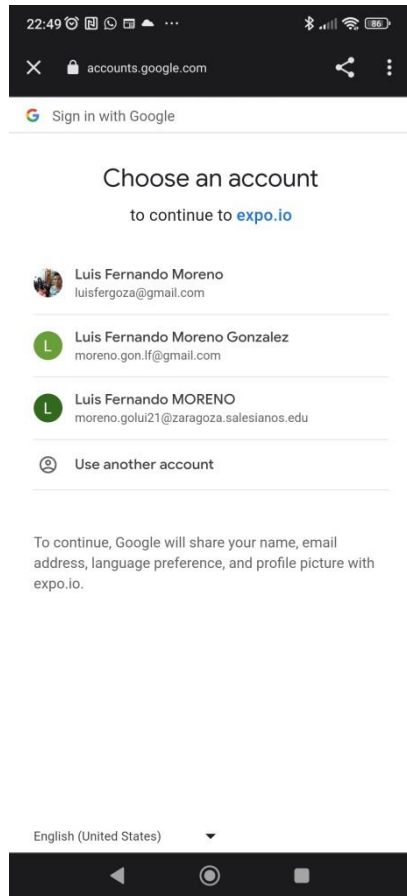


Ilustración 3 : Perfiles para autenticación con Google

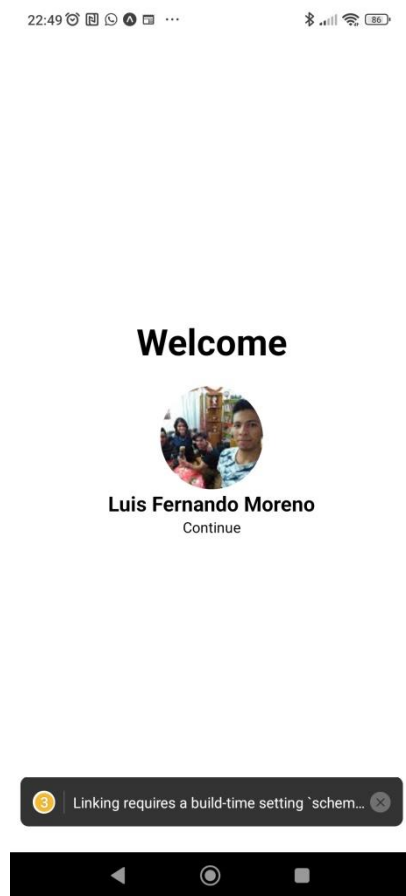


Ilustración 4: Pantalla de autenticación exitosa

2. Pantalla de Inicio:

Esta pantalla muestra al usuario las opciones básicas de la aplicación:

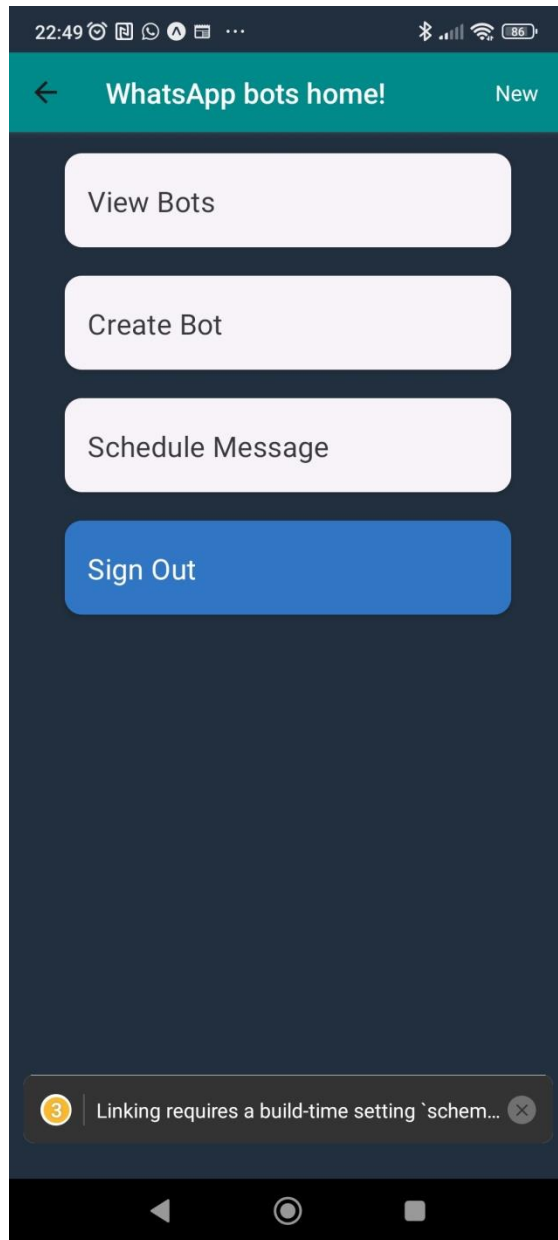


Ilustración 5 : Pantalla Home del usuario

3. Pantalla de listado de bots

Esta pantalla muestra una vista general de los bots automáticos existentes y proporciona opciones para crear un nuevo bot.

Los usuarios pueden ver una lista de bots y realizar acciones como crear un nuevo bot o seleccionar uno existente para editarlo o eliminarlo.

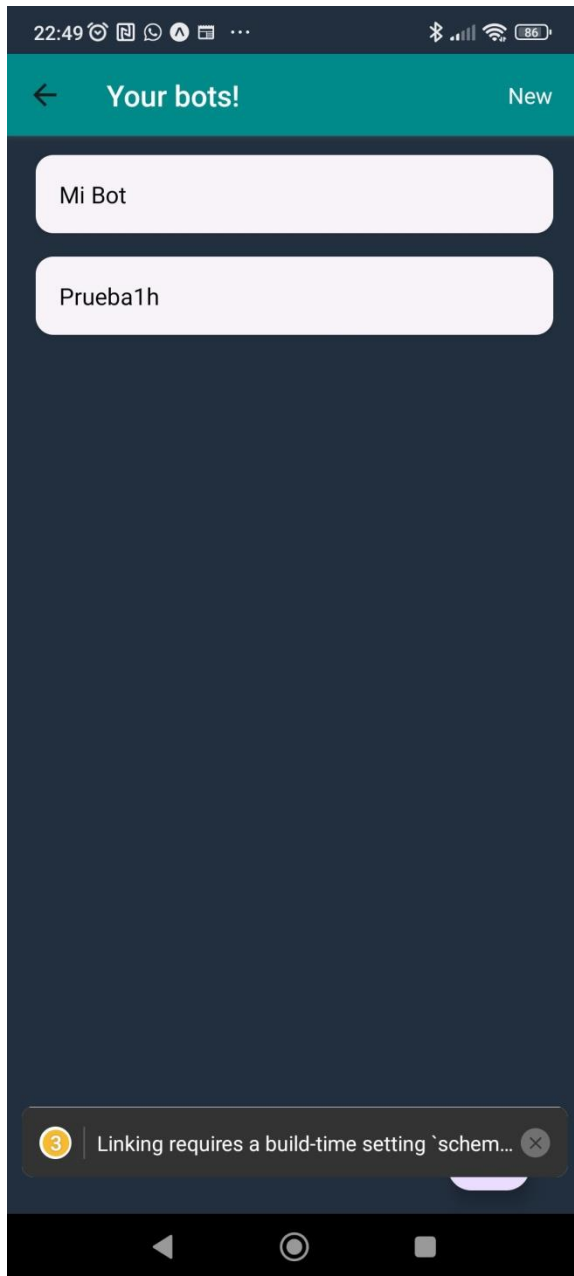


Ilustración 6: Listado de bots del usuario

2. Creación de Bots:

Los usuarios pueden acceder a la pantalla de creación de bots para configurar un nuevo bot automático.

Se proporcionan campos para ingresar el nombre del bot y las horas de inicio y fin de su actividad.

También hay opciones para personalizar el comportamiento del bot, como definir respuestas automáticas, establecer palabras clave y programar mensajes automatizados.

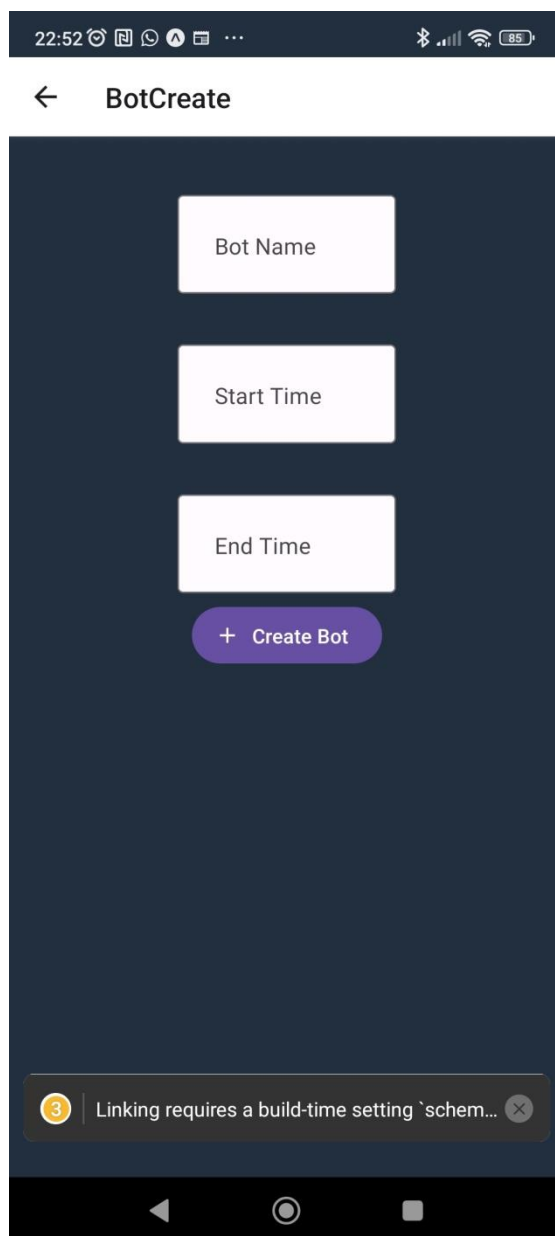


Ilustración 7: Pantalla creación de bots

3. Personalización de Bots:

Los usuarios pueden acceder a la pantalla de personalización de bots para editar las configuraciones y el comportamiento de un bot existente.

Pueden modificar las respuestas automáticas, las palabras clave, el horario de actividad y otros parámetros del bot.

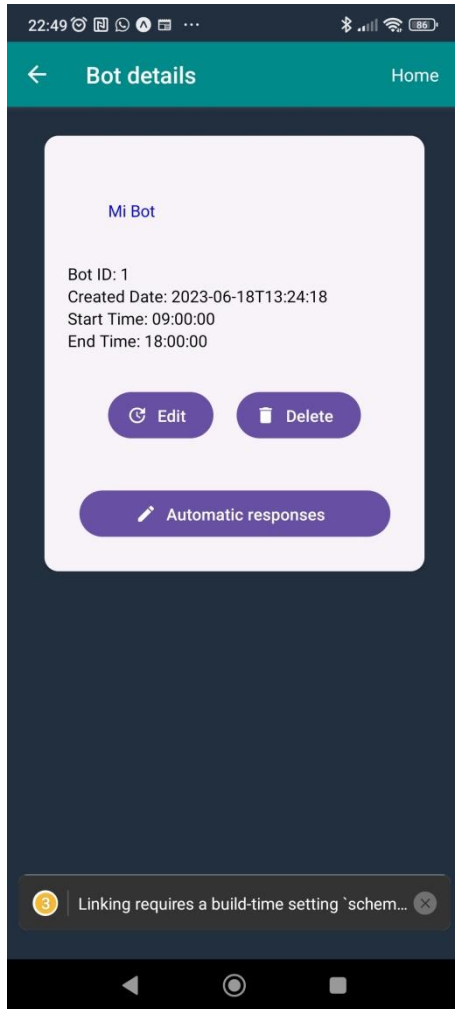


Ilustración 8: Pantalla de detalles del bot

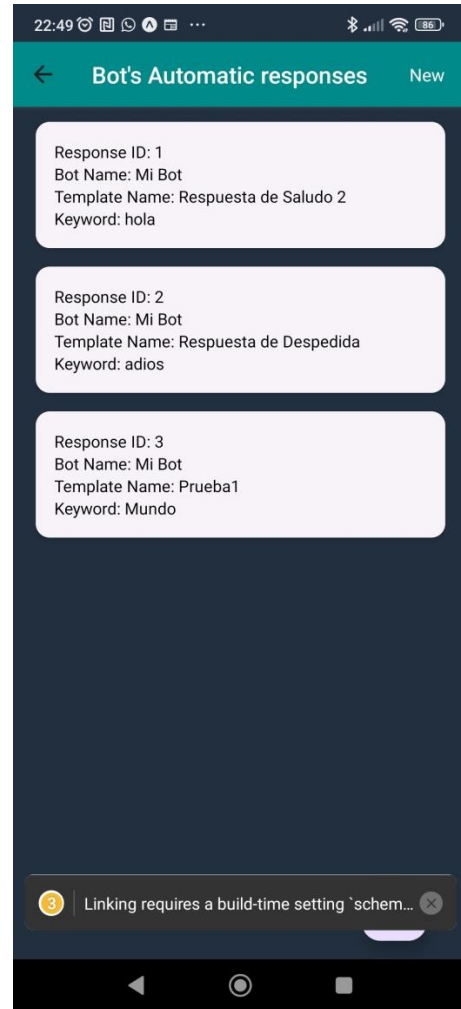


Ilustración 9: Pantalla respuestas automáticas

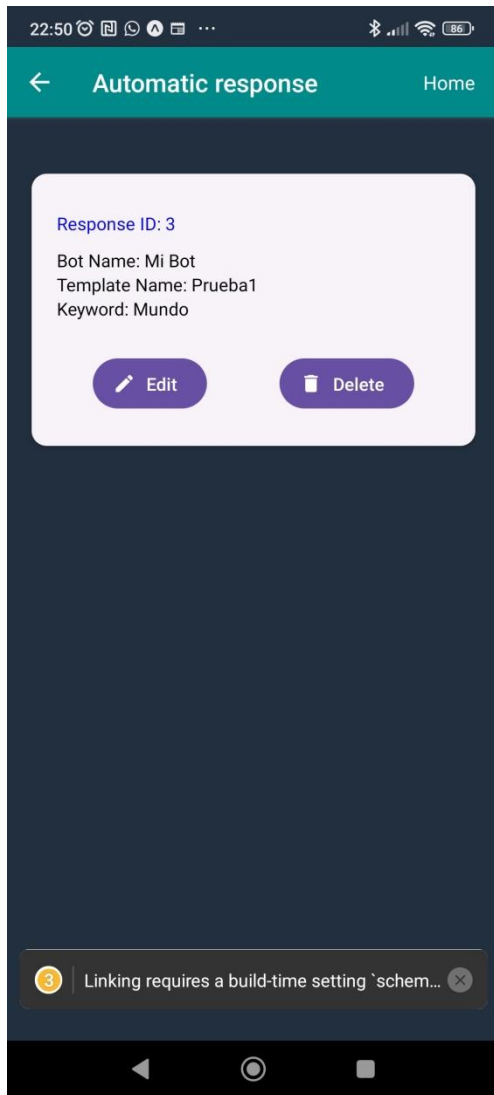


Ilustración 10 Pantalla respuesta details

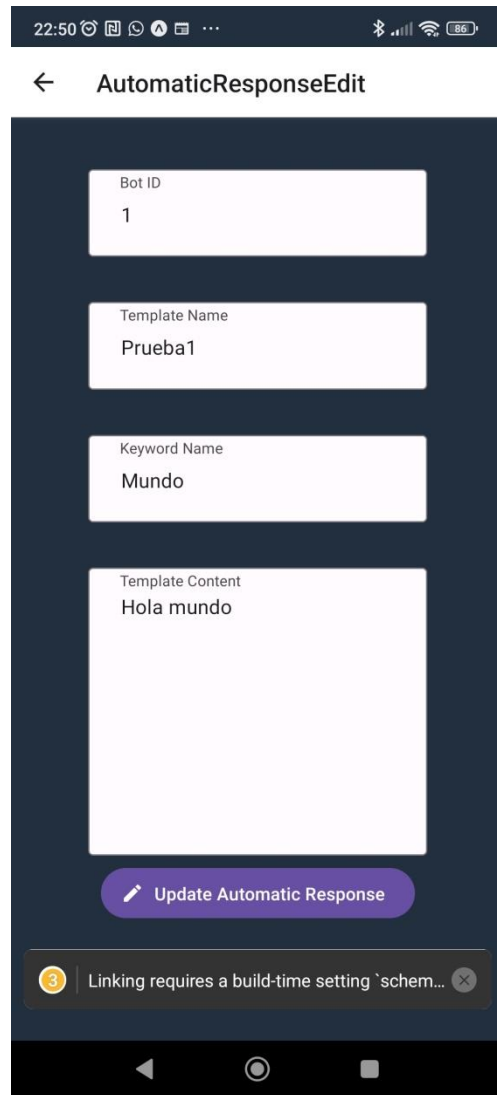


Ilustración 11 Pantalla actualizar respuesta

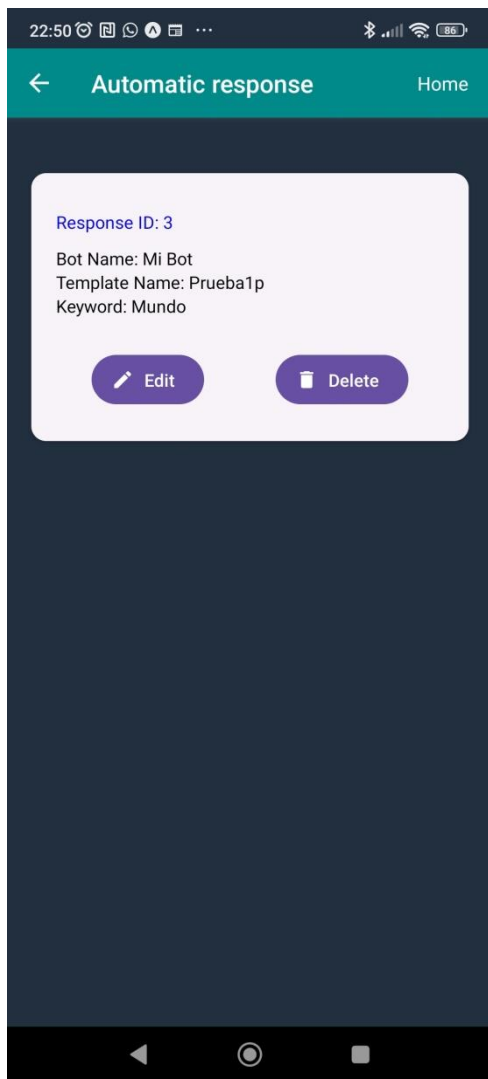


Ilustración 12 Listado respuestas

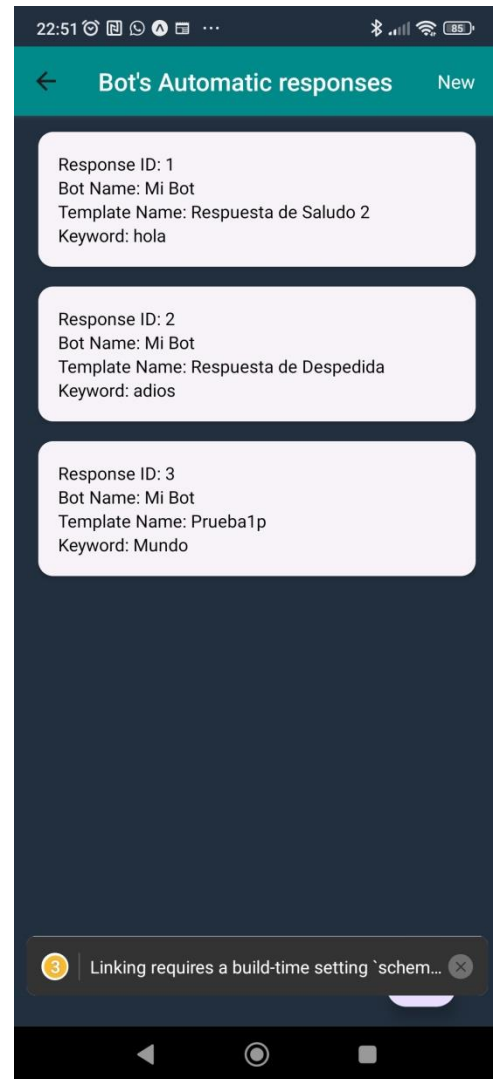


Ilustración 13 Pantalla respuesta editada

4. Gestión de Interacciones:

Como idea para un desarrollo futuro, se puede crear vista donde los usuarios puedan ver y gestionar las interacciones entre los bots automáticos y los contactos de WhatsApp. Por ejemplo, los usuarios podrán ver los mensajes recibidos y enviados, y realizar acciones como responder, reenviar o eliminar mensajes.

5. Otras vistas y características:

De igual manera, la aplicación puede incluir otras vistas y características, como la gestión de plantillas de mensajes, la programación de mensajes automatizados, la configuración de ajustes de notificación, entre otros. Son funciones que por razones de tiempo, no se han podido implementar en esta entrega.

La navegación entre las diferentes vistas se realiza mediante una barra de navegación inferior o mediante la barra superior. Se utiliza un diseño limpio y claro, con iconos intuitivos y etiquetas descriptivas para facilitar la comprensión y el uso de la aplicación. Se pueden aplicar principios de diseño de materiales o de diseño visual coherente con las directrices de la plataforma móvil (Android o iOS) para garantizar una experiencia de usuario consistente.

En términos de interacción con el usuario, se utilizan elementos interactivos como botones y campos de entrada de texto, en una próxima entrega se implementarán también casillas de verificación y listas desplegables para permitir al usuario ingresar y editar información de forma más fácil e intuitiva. También se pueden utilizar notificaciones y mensajes de confirmación para informar al usuario sobre el estado de las acciones realizadas, como la creación exitosa de un bot o la eliminación de un mensaje.

En general, el diseño gráfico y la experiencia de usuario de la aplicación se centran en proporcionar una interfaz amigable y fácil de usar que permita a los usuarios gestionar sus bots automáticos de WhatsApp de manera eficiente y personalizada, facilitando así la automatización de tareas y mejorando la eficiencia en las interacciones con los contactos.

C. Diseño de la base de datos

La base de datos utilizada consta de varias tablas que se utilizan para almacenar la información relacionada con la gestión de bots automáticos de WhatsApp. A continuación, se proporciona una descripción de la estructura de las tablas, las relaciones entre ellas y las reglas implementadas, como triggers y procedimientos almacenados:

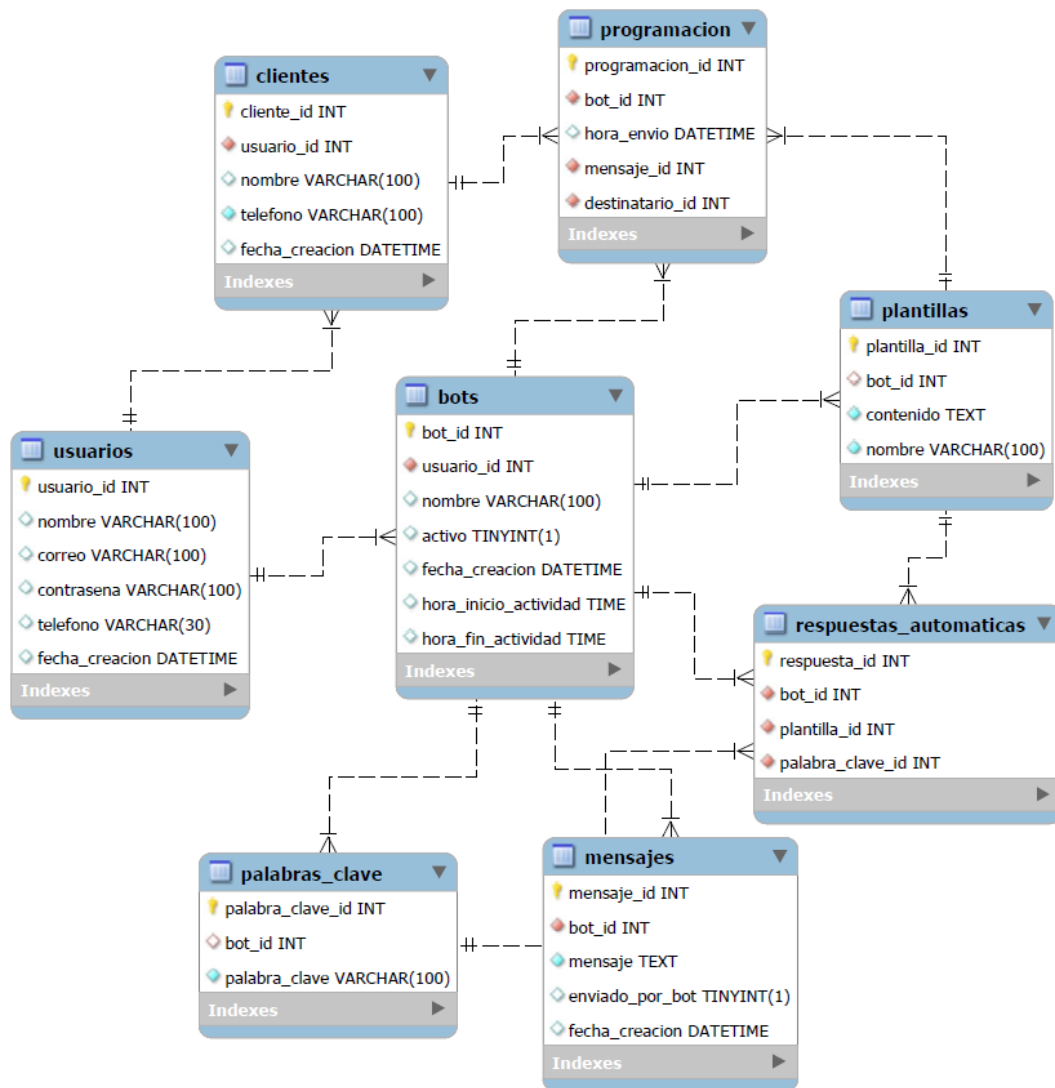


Ilustración 14 Esquema Relacional

1. Tabla "usuarios":

Almacena la información de los usuarios registrados en el sistema.

Campos: usuario_id, nombre, correo, contraseña, teléfono, fecha_creacion.

Restricciones: El correo y el teléfono deben ser únicos.

2. Tabla "bots":

Almacena la información de los bots automáticos de WhatsApp creados por los usuarios.

Campos: bot_id, usuario_id, nombre, activo, fecha_creacion, hora_inicio_actividad, hora_fin_actividad.

Restricciones: El bot_id es una clave primaria. Hay una relación de clave externa con la tabla "usuarios" a través del campo usuario_id.

Restricción única: Combinación única de usuario_id y nombre para evitar la creación de bots duplicados.

3. Tabla "clientes":

Almacena la información de los clientes de los bots.

Campos: cliente_id, usuario_id, nombre, teléfono, fecha_creacion.

Restricciones: El cliente_id es una clave primaria. Hay una relación de clave externa con la tabla "usuarios" a través del campo usuario_id.

Restricción única: Combinación única de usuario_id y teléfono para evitar la creación de clientes duplicados.

4. Tabla "mensajes":

Almacena los mensajes enviados por los bots automáticos.

Campos: mensaje_id, bot_id, mensaje, enviado_por_bot, fecha_creacion.

Restricciones: El mensaje_id es una clave primaria. Hay una relación de clave externa con la tabla "bots" a través del campo bot_id.

5. Tabla "palabras_clave":

Almacena las palabras clave asociadas a los bots automáticos.

Campos: palabra_clave_id, bot_id, palabra_clave.

Restricciones: La palabra_clave_id es una clave primaria. Hay una relación de clave externa con la tabla "bots" a través del campo bot_id.

Restricción única: Combinación única de bot_id y palabra_clave para evitar la creación de palabras clave duplicadas para un mismo bot.

6. Tabla "plantillas":

Almacena las plantillas de mensajes utilizadas por los bots automáticos.

Campos: plantilla_id, bot_id, contenido, nombre.

Restricciones: El plantilla_id es una clave primaria. Hay una relación de clave externa con la tabla "bots" a través del campo bot_id.

Restricción única: Combinación única de bot_id y nombre para evitar la creación de plantillas duplicadas.

7. Tabla "programacion":

Almacena la información de los mensajes programados para ser enviados por los bots automáticos.

Campos: programacion_id, bot_id, hora_envio, mensaje_id, destinatario_id.

Restricciones: El programacion_id es una clave primaria. Hay relaciones de clave externa con las tablas "bots", "mensajes" y "clientes" a través de los campos bot_id, mensaje_id y destinatario_id respectivamente.

8. Tabla "respuestas_automaticas":

Almacena la configuración de las respuestas automáticas de los bots automáticos.

Campos: respuesta_id, bot_id, plantilla_id, palabra_clave_id.

Restricciones: El respuesta_id es una clave primaria. Hay relaciones de clave externa con las tablas "bots", "plantillas" y "palabras_clave" a través de los campos bot_id, plantilla_id y palabra_clave_id respectivamente.

Además de las tablas, se han incluido los siguientes elementos adicionales en la base de datos:

Triggers:

"validar_hora_fin_actividad": Un trigger que se activa antes de insertar un registro en la tabla "bots" y valida que la hora de fin de actividad no sea menor que la hora de inicio de actividad.

Procedimientos almacenados:

"crear_bot": Un procedimiento almacenado utilizado para crear un nuevo bot automático asociado a un usuario.

"crear_usuario": Un procedimiento almacenado utilizado para crear un nuevo usuario en el sistema.

"programar_mensaje": Un procedimiento almacenado utilizado para programar el envío de un mensaje por un bot automático.

Estos triggers y procedimientos almacenados ayudan a mantener la integridad de los datos y proporcionan funcionalidades adicionales para la creación y gestión de bots automáticos.

En resumen, la base de datos está diseñada de manera que las tablas están relacionadas entre sí a través de claves primarias y claves externas. Las restricciones y reglas implementadas, como claves únicas, relaciones de clave externa y triggers, aseguran la coherencia y consistencia de los datos almacenados y facilitan la gestión de bots automáticos y la interacción con ellos.

D. Desarrollo de la funcionalidad de la aplicación

A continuación, se describen las funcionalidades y características principales de la aplicación:

1. Registro de usuarios:

Los usuarios pueden registrarse en la aplicación autenticándose con su cuenta de google. Con esto ya directamente se queda guardada en la aplicación su nombre y correo, y más adelante pueden añadir el teléfono.

Al registrar el correo electrónico y el número de teléfono, se verifica que no existan ya en la base de datos.

Se utiliza el procedimiento almacenado "crear_usuario" para crear un nuevo usuario en la base de datos.

2. Inicio de sesión:

El inicio de sesión en la aplicación también se gestiona con ayuda de la autenticación de Google

3. Creación de bots automáticos:

Los usuarios pueden crear bots automáticos asociados a su cuenta.

Se proporciona un formulario donde los usuarios pueden ingresar el nombre del bot y su horario de actividad. Más adelante también podrán establecer su estado de activo o inactivo.

Se utiliza el procedimiento almacenado "crear_bot" para crear un nuevo bot asociado al usuario en la base de datos.

4. Personalización del comportamiento del bot:

Los usuarios pueden personalizar el comportamiento del bot configurando diferentes parámetros, como respuestas automáticas, horarios de actividad y palabras clave.

Para las respuestas automáticas, los usuarios pueden seleccionar una plantilla de mensaje predefinida o crear su propio mensaje personalizado.

Los usuarios pueden establecer un horario de actividad para el bot, especificando la hora de inicio y la hora de fin.

Se pueden agregar palabras clave que, al ser detectadas en los mensajes recibidos, activarán una respuesta automática específica del bot.

5. Programación de mensajes automáticos:

Lamentablemente esta característica no se ha implementado aún, pero la idea es que los usuarios pueden programar mensajes automáticos para ser enviados en un momento determinado.

Pueden seleccionar una plantilla de mensaje predefinida o crear un mensaje personalizado.

Se utiliza el procedimiento almacenado "programar_mensaje" para programar el envío del mensaje por el bot automático.

6. Administración de clientes:

Los usuarios pueden gestionar la lista de clientes con los que interactúa el bot. Esta característica tampoco se ha implementado todavía

Pueden agregar nuevos clientes, editar la información existente y eliminar clientes.

La información de los clientes, como su nombre y número de teléfono, se almacena en la tabla "clientes" de la base de datos.

7. Seguimiento de interacciones:

Los usuarios pueden realizar un seguimiento de las interacciones del bot con los clientes.

Pueden ver los mensajes enviados y recibidos por el bot, así como la fecha y hora de cada interacción.

Esta información se almacena en la tabla "mensajes" de la base de datos.

Para lograr los requisitos especificados, se utilizan varios procesos y algoritmos en la aplicación:

8. Algoritmos de validación:

Se realizan validaciones en diferentes etapas, como al registrar usuarios, crear bots y programar mensajes, para garantizar que los datos ingresados cumplan con ciertas reglas y restricciones.

9. Algoritmos de búsqueda y filtrado:

Se implementan algoritmos para buscar y filtrar bots, plantillas de mensajes, palabras clave y clientes, con el fin de permitir a los usuarios acceder y administrar la información de manera eficiente.

E. Opciones de personalización

Hasta ahora, las opciones de personalización de la aplicación se limitan a la personalización del comportamiento de los bots. Las opciones de personalización de la apariencia de la aplicación en un principio no está dentro del alcance de este proyecto, aunque no se descarta para una futura versión.

V. Implementación

Para la implementación del proyecto, se utilizaron diversas herramientas y se siguieron ciertos procedimientos para desarrollar de manera efectiva la aplicación de bots automáticos para WhatsApp. A continuación, se detallan algunas de estas herramientas y procedimientos:

A. Herramientas y tecnologías

1. Lenguajes de programación y tecnologías:

Python: Se utilizó Python como lenguaje de programación principal para el desarrollo del backend de la aplicación. Python ofrece una sintaxis clara y legible, así como una amplia gama de bibliotecas y frameworks que facilitan el desarrollo.

React Native: Se optó por React Native como el framework de desarrollo de aplicaciones móviles para el frontend de la aplicación. React Native permite crear aplicaciones móviles multiplataforma utilizando JavaScript, lo que brinda la ventaja de desarrollar una sola vez y desplegar en diferentes sistemas operativos móviles.

2. Base de datos:

MySQL: Se eligió MySQL como el sistema de gestión de bases de datos relacional para almacenar y administrar los datos de la aplicación. MySQL es ampliamente utilizado, de código abierto y ofrece un rendimiento sólido y confiable.

3. Frameworks y bibliotecas:

Flask: Se utilizó Flask, un framework de Python, para el desarrollo del backend de la aplicación. Flask es minimalista y flexible, permitiendo construir aplicaciones web de manera rápida y eficiente.

SQLAlchemy: Se utilizó SQLAlchemy como una herramienta de mapeo objeto-relacional (ORM) para facilitar la interacción con la base de datos MySQL. SQLAlchemy simplifica la manipulación de los datos y permite realizar consultas de manera más intuitiva a través de objetos y sentencias SQL generadas automáticamente.

4. Control de versiones:

Git: Se utilizó Git como sistema de control de versiones para mantener un registro de los cambios realizados en el código fuente.

B. Desarrollo del proyecto

Metodología ágil:

Se adoptó un método de trabajo similar a una metodología ágil como Scrum o Kanban, para gestionar el desarrollo del proyecto. Esto implicó la división del trabajo en sprints, y la planificación de tareas en función del desarrollo de los sprints.

El proyecto se desarrolló en el siguiente orden:

1. Creación de la base de datos

Selección de una plataforma de base de datos. En este caso se escogió MySQL por ser muy conocida y versátil. Además de la experiencia previa del desarrollador con el lenguaje.

Diseño de la estructura de la base de datos. Para esto se diseñó un esquema entidad-relación que luego se transformó en un diagrama relacional. El diseño sufrió varios cambios durante el desarrollo.

Implementación de la conexión de la base de datos con la aplicación. Esto se realizó con ayuda de la librería SQLAlchemy.

2. Desarrollo de la funcionalidad de la aplicación

El siguiente paso fue la creación del modelo en el backend, el mapeo de las rutas de la API y la creación de los controladores necesarios para la gestión.

A continuación se desarrolló el funcionamiento del bot y su conexión a la API de WhatsApp por medio de Twilio.

Y finalmente, se realizó la implementación de las funciones básicas de la aplicación, como la creación de bots, la configuración de respuestas automáticas y horarios de actividad, entre otras.

Es posible la implementación de un sistema de seguimiento para monitorear las interacciones del bot con los contactos de WhatsApp. Para esto se almacenan todas las interacciones de los bots con los clientes. Sin embargo esta funcionalidad aún está a medio desarrollar.

3. Diseño de la interfaz de usuario

Una vez desarrollada la funcionalidad, se empezó a desarrollar el frontend con React Native.

Se creó el diseño y una arquitectura de navegación para la aplicación.

Se realizó el diseño de cada pantalla de la aplicación.

4. Implementación de las opciones de personalización

Se implementaron algunas opciones de personalización en la aplicación, como definir palabras clave para activar el bot.

C. Pruebas y depuración

Se realizaron pruebas para asegurar el correcto funcionamiento de la aplicación. Tales como pruebas de las peticiones a la API usando Postman. También pruebas de la capacidad de enviar y recibir mensajes usando Twilio.

Para poder ver el seguimiento de las pruebas realizadas, ver el Anexo II: Justificación de las pruebas de funcionamiento

Se encontraron inconvenientes a la hora de gestionar los valores 'time' en la base de datos. Aún está por terminar de solucionarse.

Para poder comunicar la aplicación con la API de WhatsApp, era necesario que la aplicación se ejecutara en un host accesible desde internet. Para conseguir esto se utilizó la biblioteca ngrok, que permite que un host local sea accesible desde internet.

Para seguir garantizando el correcto funcionamiento de la aplicación, es necesario seguir haciendo pruebas y mantenerse en la búsqueda de errores que se puedan producir, especialmente al momento de desarrollar más funcionalidades de la aplicación.

VI. Evaluación y resultados

A. Evaluación de los requisitos

A continuación se evalúa el cumplimiento de requisitos funcionales y no funcionales:

Requisitos Funcionales	Cumple
Compatibilidad con la última versión de WhatsApp	Sí
Creación de bots automáticos para múltiples números de teléfono	No
Personalización del comportamiento del bot	Sí
Activación/Desactivación del bot de manera sencilla	No
Interfaz intuitiva y fácil de usar	Sí

Tabla 1: Evaluación de requisitos funcionales

Requisitos No Funcionales	Cumple
Calidad del software	Sí

Requisitos No Funcionales	Cumple
Eficiencia en el uso de recursos	Sí
Usabilidad y accesibilidad	No
Seguridad de la información	No

Tabla 2: Evaluación de requisitos no funcionales

Comentarios sobre los requisitos cumplidos y no cumplidos:

- Se consigue la compatibilidad con la última versión de WhatsApp ya que se usa la API oficial
- No se han podido hacer pruebas de implementación de bots para múltiples números de teléfono, y por tanto es posible que surjan errores cuando se quiera implementar.
- Se ha conseguido una personalización básica del comportamiento de los bots ya que se les puede especificar palabras claves y respuestas, así como horarios de actividad
- Activación o desactivación de los bots:
- Actualmente la única manera de desactivar un bot es activando otro, es necesario implementar un control en la interfaz para poder gestionar la actividad de los bots.
- La interfaz es sencilla y fácil de entender, además tiene palabras descriptivas para cada parte de esta
- El software está bien estructurado y modularizado, facilitando la reutilización de código y aumentando la eficiencia
- Las prácticas de accesibilidad no se han implementado aún en la aplicación, aunque sigue siendo muy fácil de usar.
- La aplicación utiliza tokens de autenticación tanto para la autenticación con google como para el manejo de mensajes de WhatsApp con Twilio, pero aparte de eso no implementa protocolos estrictos de seguridad.

B. Resultados obtenidos

Durante la implementación de la aplicación, se llevaron a cabo diversas evaluaciones para medir los resultados concretos en términos de eficiencia, usabilidad y tiempo de respuesta. A continuación, se presentan los principales resultados obtenidos:

1. Eficiencia en el uso de recursos:

Al momento de realizar pruebas con la aplicación, se observa que los tiempos de respuesta para realizar sus funciones son bastante cortos, lo que da testimonio de su eficiencia, además la interfaz se siente fluida y sin problemas. En la siguiente imagen se puede ver cómo los logs nos muestran que se pueden hacer varias peticiones por minuto a la aplicación, y haciendo esto no se observan inconvenientes en la interfaz.

```
192.168.1.143 - - [18/Jun/2023 22:51:12] "GET /bots/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:13] "GET /bots/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:15] "GET /respuestas_automaticas/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:15] "GET /bots/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:16] "GET /plantillas/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:16] "GET /bots/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:16] "GET /palabras_clave/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:16] "GET /plantillas/2 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:16] "GET /palabras_clave/2 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:16] "GET /palabras_clave/3 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:16] "GET /bots/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:16] "GET /plantillas/3 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:40] "GET /plantillas/3 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:40] "GET /palabras_clave/3 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:40] "GET /bots/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:51:42] "DELETE /respuestas_automaticas/3 HTTP/1.1" 200 -
<Response 163 bytes [200 OK]>
192.168.1.143 - - [18/Jun/2023 22:52:20] "GET /users/email/luisfergoza@gmail.com HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:52:20] "POST /users HTTP/1.1" 200 -
{'usuario_id': 1, 'nombre': 'Uno', 'hora_inicio_actividad': '12:00', 'hora_fin_actividad': '13:00'}
192.168.1.143 - - [18/Jun/2023 22:52:41] "POST /bots HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:52:41] "GET /bots/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:52:45] "GET /bots/1 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:52:53] "GET /respuestas_automaticas/bot/3 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:53:24] "POST /plantillas HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:53:24] "POST /palabras_clave HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:53:24] "POST /respuestas_automaticas HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:53:30] "GET /respuestas_automaticas/bot/3 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:53:31] "GET /plantillas/4 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:53:31] "GET /bots/bot/3 HTTP/1.1" 200 -
192.168.1.143 - - [18/Jun/2023 22:53:31] "GET /palabras_clave/4 HTTP/1.1" 200 -
127.0.0.1 - - [18/Jun/2023 22:58:59] "POST /bot HTTP/1.1" 200 -
127.0.0.1 - - [18/Jun/2023 22:59:24] "POST /bot HTTP/1.1" 200 -
127.0.0.1 - - [18/Jun/2023 22:59:29] "POST /bot HTTP/1.1" 200 -
```

2. Usabilidad:

La usabilidad de la aplicación fue evaluada por el desarrollador, el resultado de la evaluación es que la app es fácil de usar pero necesita implementar reglas de accesibilidad. Se tuvo en cuenta la facilidad de navegación, la claridad de la interfaz, la intuitividad de las acciones y la comprensión de las funcionalidades disponibles. Los resultados demostraron que la aplicación cumplió con éxito con los requisitos de usabilidad.

3. Cumplimiento de requisitos:

Se verificó que la aplicación cumpliera con los requisitos funcionales y no funcionales establecidos durante la etapa de análisis. Los resultados mostraron que la aplicación cumplió satisfactoriamente con al menos la mitad de los requisitos.

En resumen, los resultados obtenidos de la implementación de la aplicación fueron satisfactorios. Se logró una alta eficiencia en el uso de recursos y buena usabilidad. Estos resultados indican que la aplicación todavía necesita trabajo pero que tiene potencial.

C. Análisis de los resultados

En general el trabajo realizado ha sido completo y buscando seguir las buenas prácticas. La aplicación no está completa pero eso sólo indica que hay mucho espacio para mejorar. Se ha aprendido mucho en el proceso, sobre todos de los lenguajes usados y varias herramientas que el desarrollador no conocía.

D. Ampliaciones y mejoras

- Mejorar la interfaz para tener un mejor control sobre los bots
- Implementar protocolos de seguridad.
- Aumentar las funcionalidades de la aplicación, así como las herramientas que faciliten su uso, especialmente componentes que no se han llegado a usar.
- Implementación de una pantalla donde los usuarios puedan ver y gestionar las interacciones entre los bots automáticos y los contactos de WhatsApp. Por ejemplo, los usuarios podrán ver los mensajes recibidos y enviados, y realizar acciones como responder, reenviar o eliminar mensajes.

E. Estimación del tiempo empleado

De un cálculo arbitrario se obtiene que se han empleado alrededor de 200 horas en el desarrollo del proyecto

F. Valoración personal

En general el proyecto me ha gustado mucho y ha sido muy satisfactorio sacarlo adelante, a pesar de los dolores de cabeza que ocasionan esos errores que son difíciles de sacar pero al final se aprende mucho.

VII. Bibliografía:

Documentación sobre Python

Python Software Foundation. (s.f.). Módulos de Python. Recuperado de <https://docs.python.org/es/3/tutorial/modules.html>

El libro de Python. (s.f.). Guía de estilo Python PEP8. Recuperado de <https://ellibrodepython.com/python-pep8>

Recursos Python. (s.f.). PEP 8 en español. Recuperado de <https://recursospython.com/pep8es.pdf>

Documentación sobre paquetes de Python

Pallets. (s.f.). Flask Installation. Recuperado de <https://flask.palletsprojects.com/en/2.3.x/installation/>

Pallets. (s.f.). Flask Quickstart. Recuperado de <https://flask.palletsprojects.com/en/2.3.x/quickstart/>

Pallets. (s.f.). Flask-SQLAlchemy Quickstart. Recuperado de <https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/quickstart/>

Python Package Index. (s.f.). Flask-Marshmallow. Recuperado de <https://pypi.org/project/flask-marshmallow/>

Python Package Index. (s.f.). Marshmallow-SQLAlchemy. Recuperado de <https://pypi.org/project/marshmallow-sqlalchemy/>

J2Logo. (s.f.). Tutorial de Python SQLAlchemy. Recuperado de <https://j2logo.com/python/sqlalchemy-tutorial-de-python-sqlalchemy-guia-de-inicio/>

J2Logo. (s.f.). Tutorial Flask: Base de datos con Flask-SQLAlchemy. Recuperado de <https://j2logo.com/tutorial-flask-leccion-5-base-de-datos-con-flask-sqlalchemy/>

Flask-CORS. (s.f.). Documentación Flask-CORS. Recuperado de <https://flask-cors.readthedocs.io/en/latest/>

Python Package Index. (s.f.). mysqlclient. Recuperado de <https://pypi.org/project/mysqlclient/>

Python Package Index. (s.f.). Flask-DotEnv. Recuperado de <https://pypi.org/project/Flask-DotEnv/>

Documentación sobre React Native

React Native. (s.f.). Environment Setup. Recuperado de <https://reactnative.dev/docs/environment-setup>

Create React App. (s.f.). Homepage. Recuperado de <https://create-react-app.dev/>

React Native. (s.f.). Environment Setup - Quickstart. Recuperado de <https://reactnative.dev/docs/environment-setup?guide=quickstart>

npm, Inc. (s.f.). react-router-dom. Recuperado de <https://www.npmjs.com/package/react-router-dom>

Callstack. (s.f.). React Native Paper. Recuperado de <https://callstack.github.io/react-native-paper/>

React Navigation. (s.f.). Getting Started. Recuperado de <https://reactnavigation.org/docs/getting-started/>

Documentación sobre paquetes de Node.js

Axios. (s.f.). Axios. Recuperado de <https://github.com/axios/axios>

npm, Inc. (s.f.). axios. Recuperado de <https://www.npmjs.com/package/axios>

Documentación sobre autenticación con Google

Google Developers. (s.f.). Implementar OAuth. Recuperado de <https://developers.google.com/my-business/content/implement-oauth?hl=es>

Documentación sobre manejo de mensajes de Whatsapp con Twilio

Twilio. (s.f.). Enviar y recibir mensajes multimedia con WhatsApp en Python. Recuperado de <https://www.twilio.com/es-mx/docs/whatsapp/tutorial/send-and-receive-media-messages-whatsapp-python#genera-un-twiml-en-tu-aplicacio%CC%81n>

Tutoriales de Youtube

Cairocoders. (17 abr 2023). React-JS and Python Flask CRUD Create, Read, Update and Delete MySQL-Database [Video]. Youtube.

https://www.youtube.com/watch?v=70x6BEvZ1TI&ab_channel=Cairocoders

Beto Moedano. (27 jun 2022). Login con Google | Tutorial React Native Expo 2022 [Video].

Youtube. https://www.youtube.com/watch?v=DN9dQ_6ezvA&ab_channel=BetoMoedano

Fazt Code. (14 jun 2021). React Native, Node & MySQL - Aplicacion de Tareas (usando Tabnine) [Video]. Youtube.

https://www.youtube.com/watch?v=HMKVnwlhJO0&t=385s&ab_channel=FaztCode

Parwiz Forogh. (15 jun 2021). Build Mobile Apps with Python Backend & React Native [Video].

Youtube. https://www.youtube.com/watch?v=mEUSNId1Hfc&ab_channel=ParwizForogh

Code With Prince. (21 abr 2021). Flask REST API Python series: How to create a swagger UI for flask REST API | flask_swagger_ui [Video]. Youtube.

https://www.youtube.com/watch?v=AyyX9yM_OZk&ab_channel=CodeWithPrince

RealHackRWAM. (4 sept 2021). Como hacer un BOT para WHATSAPP usando lenguaje PYTHON [Video]. Youtube.

https://www.youtube.com/watch?v=fNesr7MrqGM&ab_channel=RealHackRWAM

I. Anexo I

A. Código MySQL para la creación de la base de datos

```
CREATE DATABASE IF NOT EXISTS `tfg_dam_app6` /*!40100 DEFAULT CHARACTER
SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N'
*/;
USE `tfg_dam_app6`;
-- MySQL dump 10.13 Distrib 8.0.28, for Win64 (x86_64)
--
-- Host: localhost Database: tfg_dam_app6
-- -----
-- Server version 8.0.28

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO'
*/;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `articles`
--

DROP TABLE IF EXISTS `articles`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `articles` (
  `id` int NOT NULL AUTO_INCREMENT,
  `title` varchar(100) NOT NULL,
  `body` text NOT NULL,
  `date` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=67 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `articles`
--
```

```

LOCK TABLES `articles` WRITE;
/*!40000 ALTER TABLE `articles` DISABLE KEYS */;
INSERT INTO `articles` VALUES (2,'artículo 2 updated','Cuerpo del
artículo 2 updatedjhgjbh','2023-06-10 18:31:40'),(5,'updated desde
articles.http','el cuerpo updated','2023-06-10 23:28:39'),(6,'artículo 6
updated a','Cuerpo del artículo 6 updated','2023-06-12
21:36:28'),(8,'artículo 8 ueiy','Cuerpo del artículo 8','2023-06-13
09:22:10'),(9,'Primer insert desde app','Body del primer insert','2023-
06-13 09:38:46'),(11,'Body3 updated','Cuerpo del artículo 1 updated
updated updated a','2023-06-13 09:40:46'),(13,'App insert 4
updated','Body 4 updated','2023-06-13 09:47:36'),(17,'Svkkj','F','2023-
06-13 10:31:16'),(18,'18 u','18','2023-06-13
10:32:20'),(22,'21','21','2023-06-13 10:54:20'),(24,'24','24','2023-06-13
10:56:58'),(26,'26','26667','2023-06-13 10:59:57'),(28,'28','28','2023-
06-13 11:05:26'),(29,'30','30','2023-06-13
11:10:27'),(30,'30','30','2023-06-13 11:12:55'),(31,'32','32','2023-06-13
11:14:43'),(33,'34','34','2023-06-13 11:16:23'),(34,'35','35','2023-06-13
11:17:23'),(36,'37','37','2023-06-13 11:18:51'),(37,'38','38','2023-06-13
11:34:00'),(38,'Clase article y db externalizadas','Cuerpo del artículo
8','2023-06-14 07:11:15'),(39,'Clase article, controladores y db
externalizadas','Clase article, controladores y db externalizadas','2023-
06-14 08:13:08'),(40,'Clase article, controladores y db
externalizadas','Clase article, controladores, rutas y config y db
externalizadas','2023-06-14 08:41:45'),(41,'desde articles.http','el
cuerpo','2023-06-14 17:33:38'),(43,'43?','asdfadsfa 43','2023-06-15
16:24:55'),(44,'desde articles.http','el cuerpo','2023-06-15
16:57:48'),(45,'Viernes ','Akeja','2023-06-16
07:02:56'),(46,'Hay','CSS','2023-06-16 07:06:20'),(48,'Hay','CSS','2023-
06-16 07:07:27'),(49,'Hehe','Add','2023-06-16
07:10:17'),(50,'Ojalá','Add','2023-06-16
07:11:04'),(51,'Review','Sigo','2023-06-16
07:18:54'),(52,'Review','Sigo','2023-06-16
07:19:07'),(53,'Review','Sigo','2023-06-16
07:20:23'),(54,'Look','Loli','2023-06-16
07:21:37'),(55,'Look','Loli','2023-06-16
07:22:16'),(56,'Look','Loli','2023-06-16 07:23:00'),(57,'Juguetes','Mi
mamá','2023-06-16 07:24:17'),(58,'Juguetes','Mi mamá','2023-06-16
07:27:23'),(59,'Juguetes','Mi mamá','2023-06-16
07:27:27'),(60,'41?','Clase article, controladores, rutas y config y db
externalizadas','2023-06-16 07:32:12'),(62,'Nerea','','2023-06-16
07:36:49'),(64,'Probando','Algo','2023-06-16
09:12:38'),(65,'Wtf','No','2023-06-16 09:13:36'),(66,'Pues ya','Ya
h','2023-06-16 09:15:26');
/*!40000 ALTER TABLE `articles` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `bots`

```

```

--
DROP TABLE IF EXISTS `bots`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `bots` (
  `bot_id` int NOT NULL AUTO_INCREMENT,
  `usuario_id` int NOT NULL,
  `nombre` varchar(100) DEFAULT NULL,
  `activo` tinyint(1) DEFAULT '1',
  `fecha_creacion` datetime DEFAULT CURRENT_TIMESTAMP,
  `hora_inicio_actividad` time DEFAULT NULL,
  `hora_fin_actividad` time DEFAULT NULL,
  PRIMARY KEY (`bot_id`),
  UNIQUE KEY `unique_user_nombre_bot` (`usuario_id`,`nombre`),
  CONSTRAINT `bots_ibfk_1` FOREIGN KEY (`usuario_id`) REFERENCES
`usuarios` (`usuario_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `bots`
--

LOCK TABLES `bots` WRITE;
/*!40000 ALTER TABLE `bots` DISABLE KEYS */;
INSERT INTO `bots` VALUES (1,1,'Mi Bot',1,'2023-06-18
13:24:18','09:00:00','18:00:00');
/*!40000 ALTER TABLE `bots` ENABLE KEYS */;
UNLOCK TABLES;
/*!50003 SET @saved_cs_client      = @@character_set_client */ ;
/*!50003 SET @saved_cs_results     = @@character_set_results */ ;
/*!50003 SET @saved_col_connection = @@collation_connection */ ;
/*!50003 SET character_set_client  = utf8mb4 */ ;
/*!50003 SET character_set_results = utf8mb4 */ ;
/*!50003 SET collation_connection  = utf8mb4_0900_ai_ci */ ;
/*!50003 SET @saved_sql_mode       = @@sql_mode */ ;
/*!50003 SET sql_mode              =
'STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION' */ ;
DELIMITER ;;
/*!50003 CREATE*/ /*!50017 DEFINER=`root`@`localhost`*/ /*!50003 TRIGGER
`validar_hora_fin_actividad` BEFORE INSERT ON `bots` FOR EACH ROW BEGIN
  IF NEW.hora_fin_actividad < NEW.hora_inicio_actividad THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La hora de fin de
actividad no puede ser menor que la hora de inicio de actividad.';
  END IF;
END */;;
DELIMITER ;

```



```

/*!50003 SET sql_mode            = @saved_sql_mode */ ;
/*!50003 SET character_set_client = @saved_cs_client */ ;
/*!50003 SET character_set_results = @saved_cs_results */ ;
/*!50003 SET collation_connection = @saved_col_connection */ ;

--
-- Table structure for table `clientes`
--

DROP TABLE IF EXISTS `clientes`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `clientes` (
  `cliente_id` int NOT NULL AUTO_INCREMENT,
  `usuario_id` int NOT NULL,
  `nombre` varchar(100) DEFAULT NULL,
  `telefono` varchar(100) NOT NULL,
  `fecha_creacion` datetime DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`cliente_id`),
  UNIQUE KEY `unique_usuario_telefono` (`usuario_id`,`telefono`),
  CONSTRAINT `fk_clientes_usuarios` FOREIGN KEY (`usuario_id`) REFERENCES
`usuarios` (`usuario_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `clientes`
--

LOCK TABLES `clientes` WRITE;
/*!40000 ALTER TABLE `clientes` DISABLE KEYS */;
INSERT INTO `clientes` VALUES (1,1,'Fernando Moreno','34680683625','2023-
06-18 13:56:28');
/*!40000 ALTER TABLE `clientes` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `mensajes`
--

DROP TABLE IF EXISTS `mensajes`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `mensajes` (
  `mensaje_id` int NOT NULL AUTO_INCREMENT,
  `bot_id` int NOT NULL,
  `mensaje` text NOT NULL,
  `enviado_por_bot` tinyint(1) DEFAULT '1',

```

```

    `fecha_creacion` datetime DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (`mensaje_id`),
    KEY `bot_id` (`bot_id`),
    CONSTRAINT `mensajes_ibfk_1` FOREIGN KEY (`bot_id`) REFERENCES `bots`
    (`bot_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `mensajes`
--

LOCK TABLES `mensajes` WRITE;
/*!40000 ALTER TABLE `mensajes` DISABLE KEYS */;
INSERT INTO `mensajes` VALUES (1,1,'hola',0,'2023-06-18
13:56:28'),(2,1,'hola',0,'2023-06-18 14:00:27'),(3,1,'¡Hasta
luego!',1,'2023-06-18 14:00:27'),(4,1,'adiós',0,'2023-06-18
14:00:40'),(5,1,'¡Hasta luego!',1,'2023-06-18
14:00:40'),(6,1,'j',0,'2023-06-18 14:00:45'),(7,1,'¡Hasta
luego!',1,'2023-06-18 14:00:45'),(8,1,'adios',0,'2023-06-18
14:01:47'),(9,1,'¡Hasta luego!',1,'2023-06-18
14:01:48'),(10,1,'hola',0,'2023-06-18 14:01:58'),(11,1,'¡Hasta
luego!',1,'2023-06-18 14:01:58'),(12,1,'hola',0,'2023-06-18
14:06:47'),(13,1,'¡Hasta luego!',1,'2023-06-18 14:06:48');
/*!40000 ALTER TABLE `mensajes` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `palabras_clave`
--

DROP TABLE IF EXISTS `palabras_clave`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `palabras_clave` (
  `palabra_clave_id` int NOT NULL AUTO_INCREMENT,
  `bot_id` int DEFAULT NULL,
  `palabra_clave` varchar(100) NOT NULL,
  PRIMARY KEY (`palabra_clave_id`),
  UNIQUE KEY `unique_bot_keyword` (`bot_id`,`palabra_clave`),
  CONSTRAINT `palabras_clave_ibfk_1` FOREIGN KEY (`bot_id`) REFERENCES
`bots` (`bot_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `palabras_clave`

```

```

--

LOCK TABLES `palabras_clave` WRITE;
/*!40000 ALTER TABLE `palabras_clave` DISABLE KEYS */;
INSERT INTO `palabras_clave` VALUES (2,1,'adios'),(1,1,'hola');
/*!40000 ALTER TABLE `palabras_clave` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `plantillas`
--

DROP TABLE IF EXISTS `plantillas`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `plantillas` (
  `plantilla_id` int NOT NULL AUTO_INCREMENT,
  `bot_id` int DEFAULT NULL,
  `contenido` text NOT NULL,
  `nombre` varchar(100) NOT NULL,
  PRIMARY KEY (`plantilla_id`),
  UNIQUE KEY `unique_bot_nombre_plantilla` (`bot_id`,`nombre`),
  CONSTRAINT `plantillas_ibfk_1` FOREIGN KEY (`bot_id`) REFERENCES `bots`
(`bot_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `plantillas`
--

LOCK TABLES `plantillas` WRITE;
/*!40000 ALTER TABLE `plantillas` DISABLE KEYS */;
INSERT INTO `plantillas` VALUES (1,1,'¡Hola! ¿En qué puedo
ayudarte?','Respuesta de Saludo'),(2,1,'¡Hasta luego!','Respuesta de
Despedida');
/*!40000 ALTER TABLE `plantillas` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `programacion`
--

DROP TABLE IF EXISTS `programacion`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `programacion` (
  `programacion_id` int NOT NULL AUTO_INCREMENT,

```

```

`bot_id` int NOT NULL,
`hora_envio` datetime DEFAULT CURRENT_TIMESTAMP,
`mensaje_id` int NOT NULL,
`destinatario_id` int NOT NULL,
PRIMARY KEY (`programacion_id`),
KEY `bot_id` (`bot_id`),
KEY `mensaje_id` (`mensaje_id`),
KEY `destinatario_id` (`destinatario_id`),
CONSTRAINT `programacion_ibfk_1` FOREIGN KEY (`bot_id`) REFERENCES
`bots` (`bot_id`) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT `programacion_ibfk_2` FOREIGN KEY (`mensaje_id`) REFERENCES
`plantillas` (`plantilla_id`) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT `programacion_ibfk_3` FOREIGN KEY (`destinatario_id`)
REFERENCES `clientes` (`cliente_id`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `programacion`
--

LOCK TABLES `programacion` WRITE;
/*!40000 ALTER TABLE `programacion` DISABLE KEYS */;
/*!40000 ALTER TABLE `programacion` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `respuestas_automaticas`
--

DROP TABLE IF EXISTS `respuestas_automaticas`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `respuestas_automaticas` (
  `respuesta_id` int NOT NULL AUTO_INCREMENT,
  `bot_id` int NOT NULL,
  `plantilla_id` int NOT NULL,
  `palabra_clave_id` int NOT NULL,
  PRIMARY KEY (`respuesta_id`),
  KEY `bot_id` (`bot_id`),
  KEY `plantilla_id` (`plantilla_id`),
  KEY `palabra_clave_id` (`palabra_clave_id`),
  CONSTRAINT `respuestas_automaticas_ibfk_1` FOREIGN KEY (`bot_id`)
REFERENCES `bots` (`bot_id`) ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT `respuestas_automaticas_ibfk_2` FOREIGN KEY (`plantilla_id`)
REFERENCES `plantillas` (`plantilla_id`) ON DELETE CASCADE ON UPDATE
CASCADE,

```

```

    CONSTRAINT `respuestas_automaticas_ibfk_3` FOREIGN KEY
(`palabra_clave_id`) REFERENCES `palabras_clave` (`palabra_clave_id`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `respuestas_automaticas`
--

LOCK TABLES `respuestas_automaticas` WRITE;
/*!40000 ALTER TABLE `respuestas_automaticas` DISABLE KEYS */;
INSERT INTO `respuestas_automaticas` VALUES (1,1,1,1),(2,1,2,2);
/*!40000 ALTER TABLE `respuestas_automaticas` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `usuarios`
--

DROP TABLE IF EXISTS `usuarios`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `usuarios` (
  `usuario_id` int NOT NULL AUTO_INCREMENT,
  `nombre` varchar(100) DEFAULT NULL,
  `correo` varchar(100) DEFAULT NULL,
  `contrasena` varchar(100) DEFAULT NULL,
  `telefono` varchar(30) DEFAULT NULL,
  `fecha_creacion` datetime DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`usuario_id`),
  UNIQUE KEY `correo` (`correo`),
  UNIQUE KEY `telefono_UNIQUE` (`telefono`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `usuarios`
--

LOCK TABLES `usuarios` WRITE;
/*!40000 ALTER TABLE `usuarios` DISABLE KEYS */;
INSERT INTO `usuarios` VALUES (1,'Luis Fernando
Moreno','luisfergoza@gmail.com',NULL,'1415523886','2023-06-18
05:18:30');
/*!40000 ALTER TABLE `usuarios` ENABLE KEYS */;
UNLOCK TABLES;

```

```

--
-- Dumping events for database 'tfg_dam_app6'
--
--
-- Dumping routines for database 'tfg_dam_app6'
--
/*!50003 DROP PROCEDURE IF EXISTS `crear_bot` */;
/*!50003 SET @saved_cs_client      = @@character_set_client */ ;
/*!50003 SET @saved_cs_results    = @@character_set_results */ ;
/*!50003 SET @saved_col_connection = @@collation_connection */ ;
/*!50003 SET character_set_client  = utf8mb4 */ ;
/*!50003 SET character_set_results = utf8mb4 */ ;
/*!50003 SET collation_connection  = utf8mb4_0900_ai_ci */ ;
/*!50003 SET @saved_sql_mode      = @@sql_mode */ ;
/*!50003 SET sql_mode              =
'STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION' */ ;
DELIMITER ;;
CREATE DEFINER=`root`@`localhost` PROCEDURE `crear_bot`(IN p_usuario_id
INT, IN p_nombre VARCHAR(100), IN p_activo BOOLEAN)
BEGIN
    IF EXISTS (SELECT 1 FROM usuarios WHERE usuario_id = p_usuario_id)
    THEN
        INSERT INTO bots(usuario_id, nombre, activo, fecha_creacion)
        VALUES (p_usuario_id, p_nombre, p_activo, NOW());
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Usuario no
encontrado';
    END IF;
END ;;
DELIMITER ;
/*!50003 SET sql_mode              = @saved_sql_mode */ ;
/*!50003 SET character_set_client  = @saved_cs_client */ ;
/*!50003 SET character_set_results = @saved_cs_results */ ;
/*!50003 SET collation_connection = @saved_col_connection */ ;
/*!50003 DROP PROCEDURE IF EXISTS `crear_usuario` */;
/*!50003 SET @saved_cs_client      = @@character_set_client */ ;
/*!50003 SET @saved_cs_results    = @@character_set_results */ ;
/*!50003 SET @saved_col_connection = @@collation_connection */ ;
/*!50003 SET character_set_client  = utf8mb4 */ ;
/*!50003 SET character_set_results = utf8mb4 */ ;
/*!50003 SET collation_connection  = utf8mb4_0900_ai_ci */ ;
/*!50003 SET @saved_sql_mode      = @@sql_mode */ ;
/*!50003 SET sql_mode              =
'STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION' */ ;
DELIMITER ;;

```

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `crear_usuario`(IN p_nombre
VARCHAR(100), IN p_correo VARCHAR(100), IN p_contrasena VARCHAR(100), in
p_telefono varchar(30))
BEGIN
    IF NOT EXISTS (SELECT 1 FROM usuarios WHERE correo = p_correo) THEN
        INSERT INTO usuarios(nombre, correo, contrasena, telefono,
fecha_creacion)
            VALUES (p_nombre, p_correo, p_contrasena, p_telefono, NOW());
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Este correo ya ha
sido registrado';
    END IF;
END ;;
DELIMITER ;
/*!50003 SET sql_mode            = @saved_sql_mode */ ;
/*!50003 SET character_set_client = @saved_cs_client */ ;
/*!50003 SET character_set_results = @saved_cs_results */ ;
/*!50003 SET collation_connection = @saved_col_connection */ ;
/*!50003 DROP PROCEDURE IF EXISTS `programar_mensaje` */;
/*!50003 SET @saved_cs_client      = @@character_set_client */ ;
/*!50003 SET @saved_cs_results     = @@character_set_results */ ;
/*!50003 SET @saved_col_connection = @@collation_connection */ ;
/*!50003 SET character_set_client  = utf8mb4 */ ;
/*!50003 SET character_set_results = utf8mb4 */ ;
/*!50003 SET collation_connection  = utf8mb4_0900_ai_ci */ ;
/*!50003 SET @saved_sql_mode       = @@sql_mode */ ;
/*!50003 SET sql_mode              =
'STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION' */ ;
DELIMITER ;;
CREATE DEFINER=`root`@`localhost` PROCEDURE `programar_mensaje`(IN
p_bot_id INT, IN p_mensaje_id INT, IN p_hora_envio DATETIME, in
p_destinatario_id int)
BEGIN
    IF EXISTS (SELECT 1 FROM bots WHERE bot_id = p_bot_id) AND
        EXISTS (SELECT 1 FROM mensajes WHERE mensaje_id = p_mensaje_id)
    AND
        EXISTS (SELECT 1 FROM clientes WHERE cliente_id =
p_destinatario_id) AND
        p_hora_envio > NOW() THEN
        INSERT INTO programacion(bot_id, mensaje_id, hora_envio,
destinatario_id)
            VALUES (p_bot_id, p_mensaje_id, p_hora_envio, p_destinatario_id);
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Datos no válidos para
programar un mensaje';
    END IF;
END ;;
DELIMITER ;
/*!50003 SET sql_mode            = @saved_sql_mode */ ;

```

```
/*!50003 SET character_set_client = @saved_cs_client */ ;
/*!50003 SET character_set_results = @saved_cs_results */ ;
/*!50003 SET collation_connection = @saved_col_connection */ ;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2023-06-18 14:07:41
```


B. Elementos del BackEnd

a. Clases usadas para el modelo

```
from dao.db import db, ma
import datetime

class Bot(db.Model):
    """Clase que representa a un bot y lo vincula con una tabla en la
    BBDD"""
    __tablename__ = "bots"
    bot_id = db.Column(db.Integer, primary_key=True)
    usuario_id = db.Column(db.Integer,
db.ForeignKey('usuarios.usuario_id'), nullable=False)
    nombre = db.Column(db.String(100))
    activo = db.Column(db.Boolean, default=True)
    fecha_creacion = db.Column(db.DateTime,
default=datetime.datetime.now)
    hora_inicio_actividad = db.Column(db.Time)
    hora_fin_actividad = db.Column(db.Time)

    def __init__(self, usuario_id, nombre, hora_inicio_actividad,
hora_fin_actividad):
        """Crea una nueva instancia de la clase Bot con los atributos
        indicados"""
        self.usuario_id = usuario_id
        self.nombre = nombre
        self.hora_inicio_actividad = hora_inicio_actividad
        self.hora_fin_actividad = hora_fin_actividad

class BotSchema(ma.Schema):
    """Clase que define el esquema de salida de los objetos Bot con
    Marshmallow"""
    class Meta:
        """Atributos de la entidad Bot que se mostrarán (serializarán)"""
        fields = ("bot_id", "usuario_id", "nombre", "activo",
"fecha_creacion", "hora_inicio_actividad", "hora_fin_actividad")
```

```

from dao.db import db, ma
import datetime

class Cliente(db.Model):
    """Clase que representa a un cliente y lo vincula con una tabla en la
    BBDD"""
    __tablename__ = "clientes"
    cliente_id = db.Column(db.Integer, primary_key=True)
    usuario_id = db.Column(db.Integer,
db.ForeignKey('usuarios.usuario_id'), nullable=False)
    nombre = db.Column(db.String(100))
    telefono = db.Column(db.String(100), nullable=False)
    fecha_creacion = db.Column(db.DateTime,
default=datetime.datetime.now)

    def __init__(self, usuario_id, nombre, telefono):
        """Crea una nueva instancia de la clase Cliente con los atributos
        indicados"""
        self.usuario_id = usuario_id
        self.nombre = nombre
        self.telefono = telefono

class ClienteSchema(ma.Schema):
    """Clase que define el esquema de salida de los objetos Cliente con
    Marshmallow"""
    class Meta:
        """Atributos de la entidad Cliente que se mostrarán
        (serializarán)"""
        fields = ("cliente_id", "usuario_id", "nombre", "telefono",
"fecha_creacion")

```

```

from dao.db import db, ma
import datetime

class Mensaje(db.Model):
    """Clase que representa a un mensaje y lo vincula con una tabla en la
    BBDD"""
    __tablename__ = "mensajes"
    mensaje_id = db.Column(db.Integer, primary_key=True)
    bot_id = db.Column(db.Integer, db.ForeignKey('bots.bot_id'),
    nullable=False)
    mensaje = db.Column(db.Text, nullable=False)
    enviado_por_bot = db.Column(db.Boolean, default=True)
    fecha_creacion = db.Column(db.DateTime,
    default=datetime.datetime.now)

    def __init__(self, bot_id, mensaje, enviado_por_bot):
        """Crea una nueva instancia de la clase Mensaje con los atributos
        indicados"""
        self.bot_id = bot_id
        self.mensaje = mensaje
        self.enviado_por_bot = enviado_por_bot

class MensajeSchema(ma.Schema):
    """Clase que define el esquema de salida de los objetos Mensaje con
    Marshmallow"""
    class Meta:
        """Atributos de la entidad Mensaje que se mostrarán
        (serializarán)"""
        fields = ("mensaje_id", "bot_id", "mensaje", "enviado_por_bot",
        "fecha_creacion")

```

```

from dao.db import db, ma

class PalabraClave(db.Model):
    """Clase que representa una palabra clave y la vincula con una tabla
    en la BBDD"""
    __tablename__ = "palabras_clave"
    palabra_clave_id = db.Column(db.Integer, primary_key=True)
    bot_id = db.Column(db.Integer, db.ForeignKey('bots.bot_id'),
    nullable=False)
    palabra_clave = db.Column(db.String(100), nullable=False)

    def __init__(self, bot_id, palabra_clave):
        """Crea una nueva instancia de la clase PalabraClave con los
        atributos indicados"""
        self.bot_id = bot_id
        self.palabra_clave = palabra_clave

class PalabraClaveSchema(ma.Schema):
    """Clase que define el esquema de salida de los objetos PalabraClave
    con Marshmallow"""
    class Meta:
        """Atributos de la entidad PalabraClave que se mostrarán
        (serializarán)"""
        fields = ("palabra_clave_id", "bot_id", "palabra_clave")

```

```

from dao.db import db, ma

class Plantilla(db.Model):
    """Clase que representa a una plantilla y la vincula con una tabla en
    la BBDD"""
    __tablename__ = "plantillas"
    plantilla_id = db.Column(db.Integer, primary_key=True)
    bot_id = db.Column(db.Integer, db.ForeignKey('bots.bot_id'),
    nullable=False)
    contenido = db.Column(db.Text, nullable=False)
    nombre = db.Column(db.String(100), nullable=False)

    def __init__(self, bot_id, contenido, nombre):
        """Crea una nueva instancia de la clase Plantilla con los
        atributos indicados"""
        self.bot_id = bot_id
        self.contenido = contenido
        self.nombre = nombre

class PlantillaSchema(ma.Schema):
    """Clase que define el esquema de salida de los objetos Plantilla con
    Marshmallow"""
    class Meta:
        """Atributos de la entidad Plantilla que se mostrarán
        (serializarán)"""
        fields = ("plantilla_id", "bot_id", "contenido", "nombre")

```

```

from dao.db import db, ma
import datetime

class Programacion(db.Model):
    """Clase que representa una programación de mensajes y la vincula con
    una tabla en la BBDD"""
    __tablename__ = "programacion"
    programacion_id = db.Column(db.Integer, primary_key=True)
    bot_id = db.Column(db.Integer, db.ForeignKey('bots.bot_id'),
    nullable=False)
    hora_envio = db.Column(db.DateTime, default=datetime.datetime.now)
    mensaje_id = db.Column(db.Integer,
    db.ForeignKey('plantillas.plantilla_id'), nullable=False)
    destinatario_id = db.Column(db.Integer,
    db.ForeignKey('clientes.cliente_id'), nullable=False)

    def __init__(self, bot_id, hora_envio, mensaje_id, destinatario_id):
        """Crea una nueva instancia de la clase Programacion con los
        atributos indicados"""
        self.bot_id = bot_id
        self.hora_envio = hora_envio
        self.mensaje_id = mensaje_id
        self.destinatario_id = destinatario_id

class ProgramacionSchema(ma.Schema):
    """Clase que define el esquema de salida de los objetos Programacion
    con Marshmallow"""
    class Meta:
        """Atributos de la entidad Programacion que se mostrarán
        (serializarán)"""
        fields = ("programacion_id", "bot_id", "hora_envio",
        "mensaje_id", "destinatario_id")

```

```

from dao.db import db, ma

class RespuestaAutomatica(db.Model):
    """Clase que representa una respuesta automática y la vincula con una
    tabla en la BBDD"""
    __tablename__ = "respuestas_automaticas"
    respuesta_id = db.Column(db.Integer, primary_key=True)
    bot_id = db.Column(db.Integer, db.ForeignKey('bots.bot_id'),
nullable=False)
    plantilla_id = db.Column(db.Integer,
db.ForeignKey('plantillas.plantilla_id'), nullable=False)
    palabra_clave_id = db.Column(db.Integer,
db.ForeignKey('palabras_clave.palabra_clave_id'), nullable=False)

    def __init__(self, bot_id, plantilla_id, palabra_clave_id):
        """Crea una nueva instancia de la clase RespuestaAutomatica con
        los atributos indicados"""
        self.bot_id = bot_id
        self.plantilla_id = plantilla_id
        self.palabra_clave_id = palabra_clave_id

class RespuestaAutomaticaSchema(ma.Schema):
    """Clase que define el esquema de salida de los objetos
    RespuestaAutomatica con Marshmallow"""
    class Meta:
        """Atributos de la entidad RespuestaAutomatica que se mostrarán
        (serializarán)"""
        fields = ("respuesta_id", "bot_id", "plantilla_id",
"palabra_clave_id")

```

```

from dao.db import db, ma
import datetime

class User(db.Model):
    """Clase que representa a un usuario y lo vincula con una tabla en la
    BBDD"""
    __tablename__ = "usuarios"
    usuario_id = db.Column(db.Integer, primary_key=True)
    nombre = db.Column(db.String(100))
    correo = db.Column(db.String(100), unique=True)
    contrasena = db.Column(db.String(100))
    telefono = db.Column(db.String(30))
    fecha_creacion = db.Column(db.DateTime,
    default=datetime.datetime.now)

    def __init__(self, nombre, correo, telefono):
        """Crea una nueva instancia de la clase User con los atributos
        indicados"""
        self.nombre = nombre
        self.correo = correo
        self.telefono = telefono

class UserSchema(ma.Schema):
    """Clase que define el esquema de salida de los objetos User con
    Marshmallow"""
    class Meta:
        """Atributos de la entidad User que se mostrarán
        (serializarán)"""
        fields = ("usuario_id", "nombre", "correo", "telefono",
        "fecha_creacion")

```


b. Mapeo de rutas de la API

```
from scripts.Envia Whatsapps.bot de respuestas 2 import bot

def active_bot_routes(app):
    """Define las rutas registradas para los bots activos."""
    # Rutas de bots activos
    app.route("/bot", methods=['POST'])(bot)
```

```
from controllers.bot controller import Bot controller

def bots_register_routes(app):
    """Define las rutas registradas para los bots"""
    # Rutas de bots
    app.route("/bots/<user_id>",
methods=['GET'])(Bot controller.get_bots)
    app.route("/bots", methods=['POST'])(Bot controller.add_bot)
    app.route("/bots/bot/<bot_id>",
methods=['GET'])(Bot controller.get_bot)
    app.route("/bots/<bot_id>",
methods=['PUT'])(Bot controller.update_bot)
    app.route("/bots/<bot_id>",
methods=['DELETE'])(Bot controller.delete_bot)
```

```
from controllers.cliente controller import Cliente controller

def clientes_register_routes(app):
    """Define las rutas registradas para los clientes"""
    # Rutas de clientes
    app.route("/clientes/user/<user_id>",
methods=['GET'])(Cliente controller.get_clientes_by_user)
    app.route("/clientes",
methods=['POST'])(Cliente controller.add_cliente)
    app.route("/clientes/<cliente_id>",
methods=['GET'])(Cliente controller.get_cliente)
    app.route("/clientes/<cliente_id>",
methods=['PUT'])(Cliente controller.update_cliente)
    app.route("/clientes/<cliente_id>",
methods=['DELETE'])(Cliente controller.delete_cliente)
```

```

from controllers.mensaje_controller import Mensaje_controller

def mensajes_register_routes(app):
    """Define las rutas registradas para los mensajes"""
    # Rutas de mensajes
    app.route("/mensajes/bot/<bot_id>",
methods=['GET'])(Mensaje_controller.get_mensajes_by_bot)
    app.route("/mensajes",
methods=['POST'])(Mensaje_controller.add_mensaje)
    app.route("/mensajes/<mensaje_id>",
methods=['GET'])(Mensaje_controller.get_mensaje)
    app.route("/mensajes/<mensaje_id>",
methods=['PUT'])(Mensaje_controller.update_mensaje)
    app.route("/mensajes/<mensaje_id>",
methods=['DELETE'])(Mensaje_controller.delete_mensaje)

```

```

from controllers.palabra_clave_controller import PalabraClave_controller

def palabras_clave_register_routes(app):
    """Define las rutas registradas para las palabras clave"""
    # Rutas de palabras clave
    app.route("/palabras_clave/bot/<bot_id>",
methods=['GET'])(PalabraClave_controller.get_palabras_clave_by_bot)
    app.route("/palabras_clave",
methods=['POST'])(PalabraClave_controller.add_palabra_clave)
    app.route("/palabras_clave/<palabra_clave_id>",
methods=['GET'])(PalabraClave_controller.get_palabra_clave)
    app.route("/palabras_clave/<palabra_clave_id>",
methods=['PUT'])(PalabraClave_controller.update_palabra_clave)
    app.route("/palabras_clave/<palabra_clave_id>",
methods=['DELETE'])(PalabraClave_controller.delete_palabra_clave)

```

```

from controllers.plantilla_controller import Plantilla_controller

def plantillas_register_routes(app):
    """Define las rutas registradas para las plantillas"""
    # Rutas de plantillas
    app.route("/plantillas/bot/<bot_id>",
methods=['GET'])(Plantilla_controller.get_plantillas_by_bot)
    app.route("/plantillas",
methods=['POST'])(Plantilla_controller.add_plantilla)
    app.route("/plantillas/<plantilla_id>",
methods=['GET'])(Plantilla_controller.get_plantilla)
    app.route("/plantillas/<plantilla_id>",
methods=['PUT'])(Plantilla_controller.update_plantilla)
    app.route("/plantillas/<plantilla_id>",
methods=['DELETE'])(Plantilla_controller.delete_plantilla)

```

```

from controllers.programacion_controller import Programacion_controller

def programacion_register_routes(app):
    """Define las rutas registradas para las programaciones"""
    # Rutas de programaciones
    app.route("/programaciones/bot/<bot_id>",
methods=['GET'])(Programacion_controller.get_programaciones_by_bot)
    app.route("/programaciones",
methods=['POST'])(Programacion_controller.add_programacion)
    app.route("/programaciones/<programacion_id>",
methods=['GET'])(Programacion_controller.get_programacion)
    app.route("/programaciones/<programacion_id>",
methods=['PUT'])(Programacion_controller.update_programacion)
    app.route("/programaciones/<programacion_id>",
methods=['DELETE'])(Programacion_controller.delete_programacion)

```

```

from controllers.respuesta_automatica_controller import
RespuestaAutomatica_controller

def respuestas_automaticas_register_routes(app):
    """Define las rutas registradas para las respuestas automáticas"""
    # Rutas de respuestas automáticas
    app.route("/respuestas_automaticas/bot/<bot_id>",
methods=['GET'])(RespuestaAutomatica_controller.get_respuestas_automatica
s_by_bot)
    app.route("/respuestas_automaticas",
methods=['POST'])(RespuestaAutomatica_controller.add_respuesta_automatica
)
    app.route("/respuestas_automaticas/<respuesta_id>",
methods=['GET'])(RespuestaAutomatica_controller.get_respuesta_automatica)
    app.route("/respuestas_automaticas/<respuesta_id>",
methods=['PUT'])(RespuestaAutomatica_controller.update_respuesta_automati
ca)
    app.route("/respuestas_automaticas/<respuesta_id>",
methods=['DELETE'])(RespuestaAutomatica_controller.delete_respuesta_autom
atica)

```

```

from controllers.spec_controller import specs;

def specs_register_routes(app):
    """Define las rutas registradas para las especificaciones"""
    # Especificaciones:
    app.route("/api/docs", methods=['GET'])(Lambda: specs(app))
    #def spec(): return jsonify(swagger(app))

```

```

from controllers.user_controller import User_controller

def users_register_routes(app):
    """Define las rutas registradas para los usuarios"""
    # Rutas de usuarios
    app.route("/users", methods=['GET'])(User_controller.get_users)
    app.route("/users", methods=['POST'])(User_controller.add_user)
    app.route("/users/<user_id>",
methods=['GET'])(User_controller.get_user)
    app.route("/users/<user_id>",
methods=['PUT'])(User_controller.update_user)
    app.route("/users/<user_id>",
methods=['DELETE'])(User_controller.delete_user)
    # Ruta adicional para buscar a un usuario por su correo
    app.route("/users/email/<user_email>",
methods=['GET'])(User_controller.get_user_by_email)

```

c. Controladores

```
from flask import jsonify, request
import re

from models.bot import Bot, BotSchema
from dao.db import db

bot_schema = BotSchema()
bots_schema = BotSchema(many=True)

def validate_time_format(time_str):
    """valida que la hora que le llega cumple con un formato"""
    pattern = r'^([01]\d|2[0-3]):([0-5]\d)$'
    if re.match(pattern, time_str):
        return True
    else:
        return False

class Bot_controller:
    """Clase que almacena los métodos controladores para las peticiones
    relacionadas con la entidad Bot"""

    def get_bots(user_id):
        """Obtiene los bots del usuario activo"""
        all_bots = Bot.query.filter_by(usuario_id=user_id).all()
        results = bots_schema.dump(all_bots)
        return jsonify(results)

    def add_bot():
        """Añade un bot a la BBDD con los atributos que llegan en la
        petición POST."""
        try:
            usuario_id = request.json['usuario_id']
            nombre = request.json['nombre']
            hora_inicio_actividad = request.json['hora_inicio_actividad']
            hora_fin_actividad = request.json['hora_fin_actividad']
            print(request.json)

            # Verificar si ya existe un bot con el mismo usuario_id y
            nombre
            existente = Bot.query.filter_by(usuario_id=usuario_id,
            nombre=nombre).first()
            if existente:
                return jsonify({"Error": "Ya existe un bot con el mismo
            usuario_id y nombre"})
```

```

        # Verificar el formato de las horas (HH:MM)
        hora_inicio_valida =
validate_time_format(hora_inicio_actividad)
        hora_fin_valida = validate_time_format(hora_fin_actividad)
        # if not hora_inicio_valida or not hora_fin_valida:
        #     return jsonify({"Error": "El formato de las horas no es
válido. Debe ser HH:MM:SS"})

        # Verificar el orden de las horas
        # if hora_inicio_actividad >= hora_fin_actividad:
        #     print("La hora de inicio de actividad debe ser menor
que la hora de fin de actividad")
        #     return jsonify({"Error": "La hora de inicio de
actividad debe ser menor que la hora de fin de actividad"})

        # Cambiar el valor 'activo' a los demás bots del mismo
usuario
        otros_bots = Bot.query.filter_by(usuario_id=usuario_id).all()
        for bot in otros_bots:
            bot.activo = False

        bot = Bot(usuario_id, nombre, hora_inicio_actividad,
hora_fin_actividad)
        db.session.add(bot)
        db.session.commit()

        return bot_schema.jsonify(bot)
    except:
        return jsonify({"Error": "Could not create bot"})

def get_bot(bot_id):
    """Devuelve un bot dado su bot_id"""
    bot = Bot.query.get(bot_id)
    return bot_schema.jsonify(bot)

def delete_bot(bot_id):
    """Elimina un bot de la base de datos si existe"""
    bot = Bot.query.get(bot_id)

    if bot is not None:
        db.session.delete(bot)
        db.session.commit()

    return bot_schema.jsonify(bot)

def update_bot(bot_id):
    """Modifica un bot con los atributos que llegan en la petición"""
    bot = Bot.query.get(bot_id)

```

```

print(bot_id)
print(bot_schema.jsonify(bot))

if bot is not None:
    bot.nombre = request.json['nombre']
    hora_inicio_actividad = request.json['hora_inicio_actividad']
    hora_fin_actividad = request.json['hora_fin_actividad']
    print(request.json)

    # Verificar si ya existe otro bot con el mismo usuario_id y
nombre
    existente = Bot.query.filter(Bot.bot_id != bot_id,
Bot.usuario_id == bot.usuario_id, Bot.nombre == bot.nombre).first()
    if existente:
        print("Ya existe otro bot con el mismo usuario_id y
nombre")
        return jsonify({"Error": "Ya existe otro bot con el mismo
usuario_id y nombre"})

    # Verificar el formato de las horas de inicio y fin
    # if not validate_time_format(hora_inicio_actividad) or not
validate_time_format(hora_fin_actividad):
    #     print("El formato de las horas de inicio o fin no es
válido")
    #     return jsonify({"Error": "El formato de las horas de
inicio o fin no es válido"})

    # Verificar si la hora de inicio es menor que la hora de fin
    if hora_inicio_actividad >= hora_fin_actividad:
        print("La hora de inicio debe ser menor que la hora de
fin")
        return jsonify({"Error": "La hora de inicio debe ser
menor que la hora de fin"})

    # Cambiar el valor 'activo' a los demás bots del mismo
usuario si se está activando este bot
    if bot.activo:
        otros_bots = Bot.query.filter(Bot.bot_id != bot_id,
Bot.usuario_id == bot.usuario_id).all()
        for otro_bot in otros_bots:
            otro_bot.activo = False

    bot.hora_inicio_actividad = hora_inicio_actividad
    bot.hora_fin_actividad = hora_fin_actividad
    print(bot_schema.jsonify(bot))

try:
    db.session.commit()
    print("En el try")

```



```
        except Exception as e:
            print(e)
    else:
        print("bot is none")

    print(bot_schema jsonify(bot))
    db.session.commit()
    return bot_schema jsonify(bot)
```

```

from flask import jsonify, request
from models.cliente import Cliente, ClienteSchema
from dao.db import db

cliente_schema = ClienteSchema()
clientes_schema = ClienteSchema(many=True)

class Cliente_controller:
    """Clase que almacena los métodos controladores para las peticiones
    relacionadas con la entidad Cliente"""

    def get_clientes_by_user(user_id):
        """Obtiene los clientes asociados a un usuario específico"""
        clientes = Cliente.query.filter_by(usuario_id=user_id).all()
        resultados = clientes_schema.dump(clientes)
        return jsonify(resultados)

    def add_cliente():
        """Añade un cliente a la BBDD con los atributos que llegan en la
        petición POST."""
        try:
            usuario_id = request.json['usuario_id']
            nombre = request.json['nombre']
            telefono = request.json['telefono']

            nuevo_cliente = Cliente(usuario_id, nombre, telefono)
            db.session.add(nuevo_cliente)
            db.session.commit()

            return cliente_schema.jsonify(nuevo_cliente)
        except:
            return jsonify({"Error": "Could not create cliente"})

    def get_cliente(cliente_id):
        """Devuelve un cliente dado su cliente_id"""
        cliente = Cliente.query.get(cliente_id)
        return cliente_schema.jsonify(cliente)

    def update_cliente(cliente_id):
        """Modifica un cliente con los atributos que llegan en la
        petición"""
        cliente = Cliente.query.get(cliente_id)

        if cliente is not None:
            cliente.nombre = request.json['nombre']
            cliente.telefono = request.json['telefono']

```

```
        db.session.commit()

    return cliente_schema.jsonify(cliente)

def delete_cliente(cliente_id):
    """Elimina un cliente de la base de datos si existe"""
    cliente = Cliente.query.get(cliente_id)

    if cliente is not None:
        db.session.delete(cliente)
        db.session.commit()

    return cliente_schema.jsonify(cliente)
```

```

from flask import jsonify, request
from models.mensaje import Mensaje, MensajeSchema
from dao.db import db

mensaje_schema = MensajeSchema()
mensajes_schema = MensajeSchema(many=True)

class Mensaje_controller:
    """Clase que almacena los métodos controladores para las peticiones
    relacionadas con la entidad Mensaje"""

    def get_mensajes_by_bot(bot_id):
        """Obtiene los mensajes asociados a un bot específico"""
        mensajes = Mensaje.query.filter_by(bot_id=bot_id).all()
        resultados = mensajes_schema.dump(mensajes)
        return jsonify(resultados)

    def add_mensaje():
        """Añade un mensaje a la BBDD con los atributos que llegan en la
        petición POST."""
        try:
            bot_id = request.json['bot_id']
            mensaje = request.json['mensaje']
            enviado_por_bot = request.json['enviado_por_bot']

            nuevo_mensaje = Mensaje(bot_id, mensaje, enviado_por_bot)
            db.session.add(nuevo_mensaje)
            db.session.commit()

            return mensaje_schema.jsonify(nuevo_mensaje)
        except:
            return jsonify({"Error": "Could not create mensaje"})

    def get_mensaje(mensaje_id):
        """Devuelve un mensaje dado su mensaje_id"""
        mensaje = Mensaje.query.get(mensaje_id)
        return mensaje_schema.jsonify(mensaje)

    def update_mensaje(mensaje_id):
        """Modifica un mensaje con los atributos que llegan en la
        petición"""
        mensaje = Mensaje.query.get(mensaje_id)

        if mensaje is not None:
            mensaje.mensaje = request.json['mensaje']
            mensaje.enviado_por_bot = request.json['enviado_por_bot']

```

```
        db.session.commit()

    return mensaje_schema.jsonify(mensaje)

def delete_mensaje(mensaje_id):
    """Elimina un mensaje de la base de datos si existe"""
    mensaje = Mensaje.query.get(mensaje_id)

    if mensaje is not None:
        db.session.delete(mensaje)
        db.session.commit()

    return mensaje_schema.jsonify(mensaje)
```

```

from flask import jsonify, request
from models.palabra_clave import PalabraClave, PalabraClaveSchema
from dao.db import db

palabra_clave_schema = PalabraClaveSchema()
palabras_clave_schema = PalabraClaveSchema(many=True)

class PalabraClave_controller:
    """Clase que almacena los métodos controladores para las peticiones
    relacionadas con la entidad PalabraClave"""

    def get_palabras_clave_by_bot(bot_id):
        """Obtiene las palabras clave asociadas a un bot específico"""
        palabras_clave =
PalabraClave.query.filter_by(bot_id=bot_id).all()
        resultados = palabras_clave_schema.dump(palabras_clave)
        return jsonify(resultados)

    def add_palabra_clave():
        """Añade una palabra clave a la BBDD con los atributos que llegan
        en la petición POST."""
        try:
            bot_id = request.json['bot_id']
            palabra_clave = request.json['palabra_clave']

            # Verificar si ya existe una palabra clave con el mismo
            bot_id y palabra_clave
            existente = PalabraClave.query.filter_by(bot_id=bot_id,
palabra_clave=palabra_clave).first()
            if existente:
                return jsonify({"Error": "This keyword already exists for
this bot, choose a different keyword"})

            nueva_palabra_clave = PalabraClave(bot_id, palabra_clave)
            db.session.add(nueva_palabra_clave)
            db.session.commit()

            return palabra_clave_schema.jsonify(nueva_palabra_clave)
        except:
            return jsonify({"Error": "Could not create palabra clave"})

    def get_palabra_clave(palabra_clave_id):
        """Devuelve una palabra clave dado su palabra_clave_id"""
        palabra_clave = PalabraClave.query.get(palabra_clave_id)
        return palabra_clave_schema.jsonify(palabra_clave)

```

```

def update_palabra_clave(palabra_clave_id):
    """Modifica una palabra clave con los atributos que llegan en la
    petición"""
    palabra_clave = PalabraClave.query.get(palabra_clave_id)

    if palabra_clave is not None:
        nueva_palabra_clave = request.json['palabra_clave']

        # Verificar si ya existe una palabra clave con el mismo
        bot_id y palabra_clave
        existente =
        PalabraClave.query.filter_by(bot_id=palabra_clave.bot_id,
        palabra_clave=nueva_palabra_clave).first()
        if existente:
            return jsonify({"Error": "This keyword already exists for
            this bot, choose a different keyword"})

        palabra_clave.palabra_clave = nueva_palabra_clave

        db.session.commit()

    return palabra_clave_schema.jsonify(palabra_clave)

def delete_palabra_clave(palabra_clave_id):
    """Elimina una palabra clave de la base de datos si existe"""
    palabra_clave = PalabraClave.query.get(palabra_clave_id)

    if palabra_clave is not None:
        db.session.delete(palabra_clave)
        db.session.commit()

    return palabra_clave_schema.jsonify(palabra_clave)

```

```

from flask import jsonify, request
from models.plantilla import Plantilla, PlantillaSchema
from dao.db import db

plantilla_schema = PlantillaSchema()
plantillas_schema = PlantillaSchema(many=True)

class Plantilla_controller:
    """Clase que almacena los métodos controladores para las peticiones
    relacionadas con la entidad Plantilla"""

    def get_plantillas_by_bot(bot_id):
        """Obtiene las plantillas asociadas a un bot específico"""
        plantillas = Plantilla.query.filter_by(bot_id=bot_id).all()
        resultados = plantillas_schema.dump(plantillas)
        return jsonify(resultados)

    def add_plantilla():
        """Añade una plantilla a la BBDD con los atributos que llegan en
        la petición POST."""
        try:
            bot_id = request.json['bot_id']
            contenido = request.json['contenido']
            nombre = request.json['nombre']

            # Verificar si ya existe una plantilla con el mismo bot_id y
            nombre
            existente = Plantilla.query.filter_by(bot_id=bot_id,
            nombre=nombre).first()
            if existente:
                return jsonify({"Error": "This template name already
                exists for this bot, choose a different template name"})

            nueva_plantilla = Plantilla(bot_id, contenido, nombre)
            db.session.add(nueva_plantilla)
            db.session.commit()

            return plantilla_schema.jsonify(nueva_plantilla)
        except:
            return jsonify({"Error": "Could not create plantilla"})

    def get_plantilla(plantilla_id):
        """Devuelve una plantilla dado su plantilla_id"""
        plantilla = Plantilla.query.get(plantilla_id)
        return plantilla_schema.jsonify(plantilla)

```



```

def update_plantilla(plantilla_id):
    """Modifica una plantilla con los atributos que llegan en la
    petición"""
    plantilla = Plantilla.query.get(plantilla_id)

    if plantilla is not None:
        nuevo_contenido = request.json['contenido']
        nuevo_nombre = request.json['nombre']

        # Verificar si ya existe una plantilla con el mismo bot_id y
        nombre
        existente = Plantilla.query.filter(Plantilla.plantilla_id !=
        plantilla_id, Plantilla.bot_id == plantilla.bot_id, Plantilla.nombre ==
        nuevo_nombre).first()
        if existente:
            return jsonify({"Error": "This template name already
            exists for this bot, choose a different template name"})

        plantilla.contenido = nuevo_contenido
        plantilla.nombre = nuevo_nombre

        db.session.commit()

    return plantilla_schema.jsonify(plantilla)

def delete_plantilla(plantilla_id):
    """Elimina una plantilla de la base de datos si existe"""
    plantilla = Plantilla.query.get(plantilla_id)

    if plantilla is not None:
        db.session.delete(plantilla)
        db.session.commit()

    return plantilla_schema.jsonify(plantilla)

```

```

from flask import jsonify, request
from models.programacion import Programacion, ProgramacionSchema
from dao.db import db

# Creación del objeto esquema. Este objeto se usa cada vez que queremos
serializar una programación
programacion_schema = ProgramacionSchema()
# Creación del objeto esquema para varias programaciones (por ejemplo, al
listar todas las programaciones)
programaciones_schema = ProgramacionSchema(many=True)

class Programacion_controller:
    """Clase que almacena los métodos controladores para las
programaciones"""

    def get_programaciones_by_bot(bot_id):
        """Obtiene todas las programaciones de la base de datos y las
devuelve en formato JSON"""
        all_programaciones =
Programacion.query.filter_by(bot_id=bot_id).all()
        results = programaciones_schema.dump(all_programaciones)
        return jsonify(results)

    def add_programacion():
        """Añade una programación a la base de datos con los atributos
que llegan en la petición POST"""
        try:
            bot_id = request.json['bot_id']
            hora_envio = request.json['hora_envio']
            mensaje_id = request.json['mensaje_id']
            destinatario_id = request.json['destinatario_id']

            programacion = Programacion(bot_id, hora_envio, mensaje_id,
destinatario_id)

            db.session.add(programacion)
            db.session.commit()

            return programacion_schema.jsonify(programacion)
        except:
            return jsonify({"Error": "Could not create programacion"})

    def get_programacion(programacion_id):
        """Obtiene una programación según su ID y la devuelve en formato
JSON"""
        programacion = Programacion.query.get(programacion_id)
        return programacion_schema.jsonify(programacion)

```

```

def update_programacion(programacion_id):
    """Modifica una programación con los atributos que llegan en la
    petición PUT"""
    programacion = Programacion.query.get(programacion_id)

    if programacion is not None:
        bot_id = request.json['bot_id']
        hora_envio = request.json['hora_envio']
        mensaje_id = request.json['mensaje_id']
        destinatario_id = request.json['destinatario_id']

        programacion.bot_id = bot_id
        programacion.hora_envio = hora_envio
        programacion.mensaje_id = mensaje_id
        programacion.destinatario_id = destinatario_id

        db.session.commit()
    return programacion_schema.jsonify(programacion)

def delete_programacion(programacion_id):
    """Elimina una programación de la base de datos si existe"""
    programacion = Programacion.query.get(programacion_id)

    if programacion is not None:
        db.session.delete(programacion)
        db.session.commit()

    return programacion_schema.jsonify(programacion)

```

```

from flask import jsonify, request
from models.respuesta_automatica import RespuestaAutomatica,
RespuestaAutomaticaSchema
from dao.db import db

respuesta_automatica_schema = RespuestaAutomaticaSchema()
respuestas_automaticas_schema = RespuestaAutomaticaSchema(many=True)

class RespuestaAutomatica_controller:
    """Clase que almacena los métodos controladores para las peticiones
    relacionadas con la entidad RespuestaAutomatica"""

    def get_respuestas_automaticas_by_bot(bot_id):
        """Obtiene las respuestas automáticas asociadas a un bot
        específico"""
        respuestas_automaticas =
RespuestaAutomatica.query.filter_by(bot_id=bot_id).all()
        resultados =
respuestas_automaticas_schema.dump(respuestas_automaticas)
        return jsonify(resultados)

    def add_respuesta_automatica():
        """Añade una respuesta automática a la BBDD con los atributos que
        llegan en la petición POST."""
        try:
            bot_id = request.json['bot_id']
            plantilla_id = request.json['plantilla_id']
            palabra_clave_id = request.json['palabra_clave_id']

            nueva_respuesta_automatica = RespuestaAutomatica(bot_id,
plantilla_id, palabra_clave_id)
            db.session.add(nueva_respuesta_automatica)
            db.session.commit()

            return
respuesta_automatica_schema.jsonify(nueva_respuesta_automatica)
        except:
            return jsonify({"Error": "Could not create respuesta
automática"})

    def get_respuesta_automatica(respuesta_id):
        """Devuelve una respuesta automática dado su respuesta_id"""
        respuesta_automatica =
RespuestaAutomatica.query.get(respuesta_id)
        return respuesta_automatica_schema.jsonify(respuesta_automatica)

    def update_respuesta_automatica(respuesta_id):

```

```

        """Modifica una respuesta automática con los atributos que llegan
        en la petición"""
        respuesta_automatica =
RespuestaAutomatica.query.get(respuesta_id)

        if respuesta_automatica is not None:
            respuesta_automatica.plantilla_id =
request.json['plantilla_id']
            respuesta_automatica.palabra_clave_id =
request.json['palabra_clave_id']

            db.session.commit()

        return respuesta_automatica_schema.jsonify(respuesta_automatica)

    def delete_respuesta_automatica(respuesta_id):
        """Elimina una respuesta automática de la base de datos si
        existe"""
        respuesta_automatica =
RespuestaAutomatica.query.get(respuesta_id)

        if respuesta_automatica is not None:
            db.session.delete(respuesta_automatica)
            db.session.commit()

        return respuesta_automatica_schema.jsonify(respuesta_automatica)

```

```

from flask import jsonify;
from flask_swagger import swagger

from dao.db import db;

def specs(app):
    """Establece la documentación con ayuda de swagger"""
    swag = swagger(app)
    swag['info']['version'] = "1.0"
    swag['info']['title'] = "ChatFlow API Documentation"
    return jsonify(swag)

```

```

from flask import jsonify, request
from models.user import User, UserSchema
from dao.db import db

# Creación del objeto esquema. Este objeto se usa cada vez que queremos
# serializar un usuario
user_schema = UserSchema()
# Creación del objeto esquema para varios usuarios (por ejemplo, cuando
# queremos listar todos los usuarios)
users_schema = UserSchema(many=True)

class User_controller:
    """Clase que almacena los métodos controladores para las peticiones
    relacionadas con la entidad User"""

    def get_users():
        """Lista todos los usuarios de la base de datos y los devuelve
        formateados con ayuda de la clase UserSchema"""
        all_users = User.query.all()
        results = users_schema.dump(all_users)
        return jsonify(results)

    def add_user():
        """Añade un usuario a la BBDD con los atributos que llegan en la
        petición POST."""
        try:
            email = request.json["email"]
            nombre = request.json["name"]
            telefono = request.json["id"]
            user = User(nombre, email, telefono)
            print(user_schema.jsonify(user))

            db.session.add(user)
            db.session.commit()

            return user_schema.jsonify(user)

        except:
            return jsonify({"Error": "Could not create user"})

    def get_user(user_id):
        """Devuelve un usuario dado su usuario_id"""
        user = User.query.get(user_id)
        return user_schema.jsonify(user)

    def update_user(user_id):

```

```

        """Modifica un usuario con los atributos que llegan en la
petición"""
        user = User.query.get(user_id)

        if user is not None:
            user.nombre = request.json["nombre"]
            user.correo = request.json["correo"]
            user.contrasena = request.json["contrasena"]
            user.telefono = request.json["telefono"]

            db.session.commit()

        return user_schema.jsonify(user)

def delete_user(user_id):
    """Elimina un usuario de la base de datos si existe"""
    user = User.query.get(user_id)

    if user is not None:
        db.session.delete(user)
        db.session.commit()

    return user_schema.jsonify(user)

def get_user_by_email(user_email):
    """Devuelve un usuario dado su correo electrónico"""
    user = User.query.filter_by(correo=user_email).first()
    return user_schema.jsonify(user)

```

d. Elementos auxiliares

```
from flask sqlalchemy import SQLAlchemy
from flask marshmallow import Marshmallow

# Create an instance of marshmallow and initialize it with the app
ma = Marshmallow()

# Inicialización de la app con la extensión
db = SQLAlchemy()
```

```
from flask swagger ui import get_swaggerui_blueprint

import os

from api.articles_routes import articles_register_routes;
from api.specs_routes import specs_register_routes;
from api.users_routes import users_register_routes;
from api.bots_routes import bots_register_routes;
from api.messages_routes import mensajes_register_routes;
from api.cliente_routes import clientes_register_routes;
from api.palabra clave_routes import palabras_clave_register_routes;
from api.plantillas_routes import plantillas_register_routes;
from api.programacion_routes import programacion_register_routes;
from api.respuestas automaticas_routes import
respuestas_automaticas_register_routes;
from api.active_bot_routes import active_bot_routes;

class Config:
    # Database
    configuration = username:password@hostname/database
    SQLALCHEMY_DATABASE_URI = os.getenv("SQLALCHEMY_DATABASE_URI")
    # Esta línea no es obligatoria pero si no la ponemos salta un
    warning, se recomienda así en la doc oficial
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    HOST = os.getenv("HOST") # Host Piso
    # HOST = os.getenv("NICKNAME") # Host bt3
    PORT = os.getenv("PORT")
    DEBUG = True

    # Swagger config
    SWAGGER_URL = '/api/docs'
    # Call factory function to create our blueprint
    SWAGGERUI_BLUEPRINT = get_swaggerui_blueprint(
```



```

        SWAGGER_URL, # Swagger UI static files will be mapped to
        '{SWAGGER_URL}/dist/'
        '/static/swagger.json', # '' + HOST + ':' + str(PORT),
        config={ # Swagger UI config overrides
            'app_name': "ChatFlow API",
        },
    )

directorios_rutas = [articles_register_routes, specs_register_routes,
                    users_register_routes, bots_register_routes,
                    mensajes_register_routes, clientes_register_routes,
                    palabras_clave_register_routes,
                    programacion_register_routes,
                    plantillas_register_routes, respuestas_automaticas_re
                    gister_routes,
                    active_bot_routes
                    ]

```

e. Script que ejecuta el bot y la comunicación por WhatsApp

```
from flask import request
from twilio.twiml.messaging_response import MessagingResponse

from models.user import User
from models.bot import Bot
from models.cliente import Cliente
from models.mensaje import Mensaje
from models.palabra clave import PalabraClave
from models.respuesta automatica import RespuestaAutomatica
from models.plantilla import Plantilla

from dao.db import db

def bot():
    """Recibe un mensaje desde Twilio, comprueba la palabra clave y envía
    un mensaje de respuesta"""
    incoming_msg = request.values.get('Body', '').lower()
    from_phone_number = request.values.get('From').split('+')[1]
    to_phone_number = request.values.get('To').split('+')[1]
    bot_activo = ""
    plantilla = ""

    resp = MessagingResponse()
    msg = resp.message()
    responded = False

    # Buscar al usuario por el número de teléfono
    usuario = User.query.filter_by(telefono=to_phone_number).first()

    if usuario:
        # Buscar un bot activo para el usuario
        bot_activo = Bot.query.filter_by(usuario_id=usuario.usuario_id,
        activo=True).first()

        if bot_activo:
            # Guardar el cliente si no existe
            cliente =
            Cliente.query.filter_by(usuario_id=usuario.usuario_id,
            telefono=from_phone_number).first()
            if not cliente:
                cliente = Cliente(usuario_id=usuario.usuario_id,
                nombre=request.values.get('ProfileName'), telefono=from_phone_number)
                db.session.add(cliente)
                db.session.commit()
```

```

        # Guardar el mensaje
        mensaje = Mensaje(bot_id=bot_activo.bot_id,
mensaje=incoming_msg, enviado_por_bot=False)
        db.session.add(mensaje)
        db.session.commit()

        # Buscar palabras clave y respuestas automáticas
        palabras_clave =
PalabraClave.query.filter_by(bot_id=bot_activo.bot_id).all()

        for palabra_clave in palabras_clave:
            if palabra_clave.palabra_clave.lower() in incoming_msg:
                respuesta_automatica =
RespuestaAutomatica.query.filter_by(
                    bot_id=bot_activo.bot_id,
                    palabra_clave_id=palabra_clave.palabra_clave_id
                ).first()

                if respuesta_automatica:
                    plantilla =
Plantilla.query.get(respuesta_automatica.plantilla_id)
                    msg.body(plantilla.contenido)
                    responded = True
                    mensaje = Mensaje(bot_id=bot_activo.bot_id,
mensaje=plantilla.contenido, enviado_por_bot=True)
                    db.session.add(mensaje)
                    db.session.commit()
                    break

        # Verificar si se ha respondido
        if not responded and (not usuario or not bot_activo):
            # No se encontró usuario o no hay bot activo, no enviar respuesta
            return ''
        elif not responded:
            # No se encontró una respuesta automática, enviar mensaje
predeterminado
            msg.body('Lo siento, no entiendo tu mensaje.')
            mensaje = Mensaje(bot_id=bot_activo.bot_id, mensaje='Lo siento,
no entiendo tu mensaje.', enviado_por_bot=True)
            db.session.add(mensaje)
            db.session.commit()

    return str(resp)

```

f. Código base que ejecuta el servidor con Flask

```
from flask import Flask;

from dotenv import load_dotenv
load_dotenv()

from dao.db import db, ma;
from config import Config, directorios_rutas;

# Create an instance of Flask with argument being the name of the
application's module (__name__)
app = Flask(__name__)
app.config.from_object(Config)

# API Routes config
for directorio in directorios_rutas:
    directorio(app)

# Swagger config
app.register_blueprint(Config.SWAGGERUI_BLUEPRINT,
url_prefix=Config.SWAGGER_URL)

# App initialization with debug mode, this mode is only for development
level
if __name__ == '__main__':
    with app.app_context():
        db.init_app(app)
        ma.init_app(app)
        db.create_all()
        # app.run(debug=True)
        # Tenemos que poner como host nuestra ipv4 porque sino recibiremos
network request failed
        # app.run(host = Config.HOST, port= Config.PORT, debug=Config.DEBUG)
        app.run(host = '0.0.0.0', port= 3000, debug=True)
```

C. Elementos del FrontEnd

a. Pantalla base que inicia la app

```
import 'react-native-gesture-handler';
import * as React from 'react';
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Button, Text, View, Image, TouchableOpacity } from
'react-native';
import * as WebBrowser from 'expo-web-browser';
import * as Google from 'expo-auth-session/providers/google';

import ArticlesHome from './src/screens/ArticlesHome';
import ArticlesCreate from './src/screens/ArticlesCreate';
import ArticleDetails from './src/screens/ArticleDetails';
import ArticleEdit from './src/screens/ArticleEdit';

import LoginScreen from './src/screens/LoginScreen';
import HomeScreen from './src/screens/HomeScreen';

import BotsHome from './src/screens/BotsHome';
import BotDetails from './src/screens/BotDetails';
import BotCreate from './src/screens/BotCreate';
import BotEdit from './src/screens/BotEdit';

import AutomaticResponsesHome from
'./src/screens/AutomaticResponsesHome';
import AutomaticResponseDetails from
'./src/screens/AutomaticResponsesDetails';
import AutomaticResponseEdit from './src/screens/AutomaticResponseEdit';
import AutomaticResponseCreate from
'./src/screens/AutomaticResponseCreate';

import { UserProvider } from './src/UserContext';

import Contants from 'expo-constants';

import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

const Stack = createStackNavigator();

// Function containing the navigation screens.
function App() {
  return (
```

```

<UserProvider>
  <Stack.Navigator initialRouteName="LoginScreen">
    <Stack.Screen
      name="LoginScreen"
      component={LoginScreen}
      options={{ headerShown: false }}
      initialParams={{
        onLoginSuccess: () => navigation.navigate("HomeScreen"),
      }}
    />
    <Stack.Screen
      name="HomeScreen"
      component={HomeScreen}
      options={({ navigation }) => ({
        title: "WhatsApp bots home!",
        headerStyle: {
          backgroundColor: "#008B8B",
        },
        headerTitleStyle: {
          color: "#ffffff",
        },
        headerRight: () => (
          <TouchableOpacity
            onPress={() => navigation.navigate("ArticlesCreate")}
          >
            <Text style={{ color: "#fff", marginRight: 20, fontSize:
15 }}}>
              New
            </Text>
          </TouchableOpacity>
        ),
      })} />
    <Stack.Screen
      name="BotsHome"
      component={BotsHome}
      options={({ navigation }) => ({
        title: "Your bots!",
        headerStyle: {
          backgroundColor: "#008B8B",
        },
        headerTitleStyle: {
          color: "#ffffff",
        },
        headerRight: () => (
          <TouchableOpacity
            onPress={() => navigation.navigate("ArticlesCreate")}
          >
            <Text style={{ color: "#fff", marginRight: 20, fontSize:
15 }}}>

```

```

        New
        </Text>
      </TouchableOpacity>
    ),
  )} />
<Stack.Screen
  name="BotDetails"
  component={BotDetails}
  options={({ navigation }) => ({
    title: "Bot details",
    headerStyle: {
      backgroundColor: "#008B8B",
    },
    headerTitleStyle: {
      color: "#ffffff",
    },
    headerTintColor: "#fff",
    headerRight: () => (
      <TouchableOpacity
        onPress={() => navigation.navigate("BotsHome")}
      >
        <Text style={{ color: "#fff", marginRight: 20, fontSize:
15 }}>

          Home
        </Text>
      </TouchableOpacity>
    ),
  )} />
<Stack.Screen name="BotCreate" component={BotCreate} />
<Stack.Screen name="BotEdit" component={BotEdit} />
<Stack.Screen
  name="AutomaticResponsesHome"
  component={AutomaticResponsesHome}
  options={({ navigation }) => ({
    title: "Bot's Automatic responses",
    headerStyle: {
      backgroundColor: "#008B8B",
    },
    headerTitleStyle: {
      color: "#ffffff",
    },
    headerRight: () => (
      <TouchableOpacity
        onPress={() =>
navigation.navigate("AutomaticResponseCreate")}
      >
        <Text style={{ color: "#fff", marginRight: 20, fontSize:
15 }}>

          New

```

```

        </Text>
      </TouchableOpacity>
    ),
  }}} />
<Stack.Screen
  name="AutomaticResponseDetails"
  component={AutomaticResponseDetails}
  options={({ navigation }) => ({
    title: "Automatic response",
    headerStyle: {
      backgroundColor: "#008B8B",
    },
    headerTitleStyle: {
      color: "#ffffff",
    },
    headerTintColor: "#fff",
    headerRight: () => (
      <TouchableOpacity
        onPress={() =>
navigation.navigate("AutomaticResponsesHome")}
      >
        <Text style={{ color: "#fff", marginRight: 20, fontSize:
15 }}>
          Home
        </Text>
      </TouchableOpacity>
    ),
  }}} />
<Stack.Screen name="AutomaticResponseEdit"
component={AutomaticResponseEdit} />
<Stack.Screen name="AutomaticResponseCreate"
component={AutomaticResponseCreate} />
<Stack.Screen
  name="ArticlesHome"
  component={ArticlesHome}
  options={({ navigation }) => ({
    title: "Articles App",
    headerStyle: {
      backgroundColor: "#008B8B",
    },
    headerTitleStyle: {
      color: "#ffffff",
    },
    headerRight: () => (
      <TouchableOpacity
        onPress={() => navigation.navigate("ArticlesCreate")}
      >
        <Text style={{ color: "#fff", marginRight: 20, fontSize:
15 }}>

```



```

        New
        </Text>
      </TouchableOpacity>
    ),
  }} </>

  <Stack.Screen name="ArticlesCreate" component={ArticlesCreate} />
  <Stack.Screen
    name="ArticleDetails"
    component={ArticleDetails}
    options={({ navigation }) => ({
      title: "Article details",
      headerStyle: {
        backgroundColor: "#008B8B",
      },
      headerTitleStyle: {
        color: "#ffffff",
      },
      headerTintColor: "#fff",
      headerRight: () => (
        <TouchableOpacity
          onPress={() => navigation.navigate("ArticlesHome")}
        >
          <Text style={{ color: "#fff", marginRight: 20, fontSize:
15 }}>
            Home
          </Text>
        </TouchableOpacity>
      ),
    })} />
    <Stack.Screen name="ArticleEdit" component={ArticleEdit} />
  </Stack.Navigator>
</UserProvider>
);
}

// This is the main navigation function
export default () => {
  return (
    <NavigationContainer>
      <App />
    </NavigationContainer>
  )
}

```

b. Pantallas de la app

```
import React, { useState } from 'react';
import { StyleSheet, View } from 'react-native';
import { TextInput, Button } from 'react-native-paper';

import Layout from '../components/Layout';
import { saveRespuestaAutomatica } from
'../api/RespuestaAutomaticaRoutes';
import { savePalabraClave } from '../api/PalabraClaveRoutes';
import { savePlantilla } from '../api/PlantillaRoutes';

function AutomaticResponseCreate(props) {
  const botData = props.route.params.botData;

  const [botId, setBotId] = useState(botData.bot_id);
  const [plantillaNombre, setPlantillaNombre] = useState('');
  const [palabraClaveNombre, setPalabraClaveNombre] = useState('');
  const [contenidoPlantilla, setContenidoPlantilla] = useState('');

  const createResponse = async () => {
    try {
      const newPlantilla = {
        bot_id: botId,
        contenido: contenidoPlantilla,
        nombre: plantillaNombre,
      };
      const plantilla = await savePlantilla(newPlantilla);

      const newPalabraClave = {
        bot_id: botId,
        palabra_clave: palabraClaveNombre,
      };
      const palabraClave = await savePalabraClave(newPalabraClave);

      const newRespuestaAutomatica = {
        bot_id: botId,
        plantilla_id: plantilla.plantilla_id,
        palabra_clave_id: palabraClave.palabra_clave_id,
      };
      await saveRespuestaAutomatica(newRespuestaAutomatica);

      props.navigation.navigate('AutomaticResponsesHome', { data: botData
    });
    } catch (error) {
      console.log(error);
    }
  }
}
```

```

    }
};

return (
  <Layout>
    <View>
      <TextInput
        style={styles.inputStyle}
        label="Bot ID"
        value={botId.toString()}
        mode="outlined"
        disabled
      />
      <TextInput
        style={styles.inputStyle}
        label="Template Name"
        value={plantillaNombre}
        mode="outlined"
        onChangeText={({text}) => setPlantillaNombre(text)}
      />
      <TextInput
        style={styles.inputStyle}
        label="Keyword Name"
        value={palabraClaveNombre}
        mode="outlined"
        onChangeText={({text}) => setPalabraClaveNombre(text)}
      />
      <TextInput
        style={styles.inputStyle}
        label="Template Content"
        value={contenidoPlantilla}
        multiline
        numberOfLines={10}
        mode="outlined"
        onChangeText={({text}) => setContenidoPlantilla(text)}
      />
      <Button
        style={{ margin: 10 }}
        icon="plus"
        mode="contained"
        onPress={createResponse}
      >
        Create Automatic Response
      </Button>
    </View>
  </Layout>
);
}

```

```
const styles = StyleSheet.create({  
  inputStyle: {  
    marginTop: 30,  
    padding: 10,  
  },  
});  
  
export default AutomaticResponseCreate;
```

```

import React, { useState, useEffect } from 'react';
import { StyleSheet, View } from 'react-native';
import { TextInput, Button } from 'react-native-paper';

import Layout from '../components/Layout';
import { updateRespuestaAutomatica } from
'../api/RespuestaAutomaticaRoutes';
import { getBot } from '../api/BotRoutes';
import { getPlantilla, updatePlantilla } from '../api/PlantillaRoutes';
import { getPalabraClave, updatePalabraClave } from
'../api/PalabraClaveRoutes';

function AutomaticResponseEdit(props) {
  const data = props.route.params.data;

  const [botId, setBotId] = useState(data.bot_id);
  const [plantillaId, setPlantillaId] = useState('');
  const [palabraClaveId, setPalabraClaveId] = useState('');
  const [plantillaNombre, setPlantillaNombre] = useState('');
  const [palabraClaveNombre, setPalabraClaveNombre] = useState('');
  const [contenidoPlantilla, setContenidoPlantilla] = useState('');

  useEffect(() => {
    fetchData();
  }, []);

  const fetchData = async () => {
    try {
      const bot = await getBot(data.bot_id);
      const plantilla = await getPlantilla(data.plantilla_id);
      const palabraClave = await getPalabraClave(data.palabra_clave_id);

      setBotId(bot.bot_id);
      setPlantillaId(plantilla.plantilla_id);
      setPlantillaNombre(plantilla.nombre);
      setPalabraClaveId(palabraClave.palabra_clave_id);
      setPalabraClaveNombre(palabraClave.palabra_clave);
      setContenidoPlantilla(plantilla.contenido);
    } catch (error) {
      console.log(error);
    }
  };

  const updateData = async () => {
    try {
      await updateRespuestaAutomatica(data.respuesta_id, botId,
plantillaId, palabraClaveId);

```

```

    await updatePlantilla(plantillaId, {
      bot_id: botId,
      contenido: contenidoPlantilla,
      nombre: plantillaNombre,
    });

    await updatePalabraClave(palabraClaveId, {
      bot_id: botId,
      palabra_clave: palabraClaveNombre,
    });

    props.navigation.navigate('AutomaticResponseDetails', { data: data
  });
} catch (error) {
  console.log(error);
}
};

return (
  <Layout>
    <View>
      <TextInput
        style={styles.inputStyle}
        label="Bot ID"
        value={botId.toString()}
        mode="outlined"
        onChangeText={text => setBotId(Number(text))}
      />
      <TextInput
        style={styles.inputStyle}
        label="Template Name"
        value={plantillaNombre}
        mode="outlined"
        onChangeText={text => setPlantillaNombre(text)}
      />
      <TextInput
        style={styles.inputStyle}
        label="Keyword Name"
        value={palabraClaveNombre}
        mode="outlined"
        onChangeText={text => setPalabraClaveNombre(text)}
      />
      <TextInput
        style={styles.inputStyle}
        label="Template Content"
        value={contenidoPlantilla}
        multiline
        numberOfLines={10}

```

```

        mode="outlined"
        onChangeText={text => setContenidoPlantilla(text)}
      />
      <Button
        style={{ margin: 10 }}
        icon="pencil"
        mode="contained"
        onPress={updateData}
      >
        Update Automatic Response
      </Button>
    </View>
  </Layout>
);
}

const styles = StyleSheet.create({
  inputStyle: {
    marginTop: 30,
    padding: 10,
  },
});

export default AutomaticResponseEdit;

```

```

import React, { useEffect, useState } from 'react';
import { View, Text, StyleSheet, Dimensions, RefreshControl } from
'react-native';
import { Card, Button } from 'react-native-paper';
import { useIsFocused, useNavigation } from '@react-navigation/native';

import Layout from '../components/Layout';
import { deleteRespuestaAutomatica } from
'../api/RespuestaAutomaticaRoutes';
import { getBot } from '../api/BotRoutes';
import { getPlantilla } from '../api/PlantillaRoutes';
import { getPalabraClave } from '../api/PalabraClaveRoutes';

const AutomaticResponseDetails = (props) => {
  const navigation = useNavigation();
  const response = props.route.params.data;
  const [botName, setBotName] = useState('');
  const [templateName, setTemplateName] = useState('');
  const [keyword, setKeyword] = useState('');
  const isFocused = useIsFocused();
  const [refreshing, setRefreshing] = useState(false);

  useEffect(() => {
    fetchBotName();
    fetchTemplateName();
    fetchKeyword();
  }, [isFocused]);

  const fetchBotName = async () => {
    try {
      const bot = await getBot(response.bot_id);
      setBotName(bot.nombre);
    } catch (error) {
      console.log(error);
    }
  };

  const onRefresh = React.useCallback(async () => {
    setRefreshing(true);
    await loadData();
    setRefreshing(false);
  }, []);

  const fetchTemplateName = async () => {
    try {
      const plantilla = await getPlantilla(response.plantilla_id);
      setTemplateName(plantilla.nombre);
    }
  };

```



```

    } catch (error) {
      console.log(error);
    }
  };

  const fetchKeyword = async () => {
    try {
      const palabraClave = await
getPalabraClave(response.palabra_clave_id);
      setKeyword(palabraClave.palabra_clave);
    } catch (error) {
      console.log(error);
    }
  };

  const handleDelete = async () => {
    await deleteRespuestaAutomatica(response.respuesta_id)
      .then(() => navigation.navigate('AutomaticResponsesHome'))
      .catch((error) => console.log(error));
  };

  return (
    <Layout>
      <View style={styles.container}>
        refreshControl={
          <RefreshControl
            refreshing={refreshing}
            onRefresh={onRefresh}
            colors={['#78e08f']}
            progressBackgroundColor="#0a3d62"
          />
        }>
        <Card style={styles.card}>
          <Text style={styles.text}>Response ID:
{response.respuesta_id}</Text>
          <Text>Bot Name: {botName}</Text>
          <Text>Template Name: {templateName}</Text>
          <Text>Keyword: {keyword}</Text>
          <View style={styles.buttonContainer}>
            <Button
              style={styles.button}
              icon="pencil"
              mode="contained"
              onPress={() => navigation.navigate('AutomaticResponseEdit',
{ data: response })}>
              Edit
            </Button>
            <Button

```

```

        style={styles.button}
        icon="delete"
        mode="contained"
        onPress={handleDelete}
      >
        Delete
      </Button>
    </View>
  </Card>
</View>
</Layout>
);
};

const windowWidth = Dimensions.get('window').width;

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'flex-start',
    alignItems: 'center',
    marginTop: 40
  },
  card: {
    width: windowWidth * 0.9,
    padding: 20,
  },
  text: {
    color: 'blue',
    paddingVertical: 10,
  },
  buttonContainer: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    marginTop: 20,
  },
  button: {
    marginVertical: 10,
  },
});

export default AutomaticResponseDetails;

```

```

import React, { useEffect, useState } from 'react';
import { View, StyleSheet, FlatList, SafeAreaView, RefreshControl,
Dimensions } from 'react-native';
import { useNavigation } from '@react-navigation/native';
import { FAB } from 'react-native-paper';

import AutomaticResponseItem from '../components/AutomaticResponsesItem';
import Layout from '../components/Layout';
import { getRespuestasAutomaticasByBot } from
'../api/RespuestaAutomaticaRoutes.js';

function AutomaticResponsesHome(props) {
  const navigation = useNavigation();
  const botData = props.route.params.data;
  console.log(props.route.params)

  const [automaticResponses, setAutomaticResponses] = useState([]);
  const [refreshing, setRefreshing] = useState(false);

  const loadData = async () => {
    try {
      const responses = await
getRespuestasAutomaticasByBot(botData.bot_id);
      setAutomaticResponses(responses);
    } catch (error) {
      console.log(error);
    }
  };

  const onRefresh = React.useCallback(async () => {
    setRefreshing(true);
    await loadData();
    setRefreshing(false);
  }, []);

  useEffect(() => {
    loadData();
  }, []);

  const renderResponseItem = ({ item }) => {
    return <AutomaticResponseItem response={item} />;
  };

  const handleCreateResponse = () => {
    navigation.navigate('AutomaticResponseCreate', { botData: botData });
    //console.log("handleCreateResponse")
  };
};

```

```

return (
  <Layout>
    <SafeAreaView style={styles.container}>
      <FlatList
        style={{ width: '100%' }}
        data={automaticResponses}
        keyExtractor={({item}) => `${item.respuesta_id}`}
        renderItem={renderResponseItem}
        refreshControl={
          <RefreshControl refreshing={refreshing} onRefresh={onRefresh}
colors={['#78e08f']} progressBackgroundColor="#0a3d62" />
        />
      </SafeAreaView>
    </Layout>
  );
};

const windowHeight = Dimensions.get('window').width;

const styles = StyleSheet.create({
  container: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
  },
  fab: {
    position: 'absolute',
    margin: 16,
    right: 0,
    bottom: 0,
  },
  card: {
    width: windowHeight * 0.9,
    padding: 20,
  },
});

export default AutomaticResponsesHome;

```

```

import React, { useContext, useState } from 'react';
import { StyleSheet, View } from 'react-native';
import { TextInput, Button } from 'react-native-paper';

import Layout from '../components/Layout';
import { saveBot } from '../api/BotRoutes';
import { UserContext } from '../UserContext';

function BotCreate(props) {
  const [botName, setBotName] = useState('');
  const [startTime, setStartTime] = useState('');
  const [endTime, setEndTime] = useState('');
  const { userId } = useContext(UserContext);

  const createBot = async () => {
    try {
      const newBot = {
        usuario_id: userId,
        nombre: botName,
        hora_inicio_actividad: startTime,
        hora_fin_actividad: endTime,
      };
      await saveBot(newBot);

      props.navigation.navigate('BotsHome');
    } catch (error) {
      console.log(error);
    }
  };

  return (
    <Layout>
      <View>
        <TextInput
          style={styles.inputStyle}
          label="Bot Name"
          value={botName}
          mode="outlined"
          onChangeText={(text) => setBotName(text)}
        />
        <TextInput
          style={styles.inputStyle}
          label="Start Time"
          value={startTime}
          mode="outlined"
          onChangeText={(text) => setStartTime(text)}
        />

```

```

    <TextInput
      style={styles.inputStyle}
      label="End Time"
      value={endTime}
      mode="outlined"
      onChangeText={({text}) => setEndTime(text)}
    />
    <Button
      style={{ margin: 10 }}
      icon="plus"
      mode="contained"
      onPress={createBot}
    >
      Create Bot
    </Button>
  </View>
</Layout>
);
}

const styles = StyleSheet.create({
  inputStyle: {
    marginTop: 30,
    padding: 10,
  },
});

export default BotCreate;

```

```

import React from 'react';
import { View, Text, ScrollView, StyleSheet, Alert } from 'react-native';
import { Card, FAB, Button } from 'react-native-paper';
import { useNavigation } from "@react-navigation/native";

import Layout from '../components/Layout';
import { deleteBot } from '../api/BotRoutes';

function BotDetails(props) {
  const navigation = useNavigation();
  const data = props.route.params.data;

  const handleDelete = () => {
    Alert.alert("Delete Bot", "Are you sure you want to delete the bot?",
    [
      {
        text: "Cancel",
        style: "cancel",
      },
      {
        text: "Ok",
        onPress: async () => {
          await deleteBot(data.bot_id)
            .then(() => navigation.navigate('BotsHome'))
            .catch(error => console.log(error));
        },
      },
    ],
  );
};

return (
  <Layout>
    <ScrollView>
      <Card style={styles.cardStyle}>
        <Text style={styles.textStyle1}>{data.nombre}</Text>
        <Text>Bot ID: {data.bot_id}</Text>
        <Text>Created Date: {data.fecha_creacion}</Text>
        <Text>Start Time: {data.hora_inicio_actividad}</Text>
        <Text>End Time: {data.hora_fin_actividad}</Text>
        <View style={styles.btnStyle}>
          <Button
            style={{ margin: 10 }}
            icon="update"
            mode="contained"
            onPress={() => navigation.navigate('BotEdit', { data: data
    >
  >

```

```

        Edit
      </Button>
      <Button
        style={{ margin: 10 }}
        icon="delete"
        mode="contained"
        onPress={() => handleDelete()}
      >
        Delete
      </Button>
    </View>
    <Button
      style={{ margin: 10 }}
      icon="pencil"
      mode="contained"
      onPress={() =>
navigation.navigate('AutomaticResponsesHome', { data: data })}
    >
      Automatic responses
    </Button>
  </Card>
</ScrollView>
</Layout>
);
}

const styles = StyleSheet.create({
  cardStyle: {
    marginTop: 20,
    padding: 20,
  },
  textStyle1: {
    color: 'blue',
    padding: 20,
    margin: 15,
  },
  btnStyle: {
    flexDirection: "row",
    justifyContent: "space-around",
    margin: 15,
    padding: 10,
  },
});

export default BotDetails;

```



```

import React, { useContext, useState, useEffect } from 'react';
import { StyleSheet, View, Text } from 'react-native';
import { TextInput, Button } from 'react-native-paper';

import Layout from '../components/Layout';
import { updateBot, getBot } from '../api/BotRoutes';
import { UserContext } from '../UserContext';

function BotEdit(props) {
  const botData = props.route.params.data;

  const [botName, setBotName] = useState('');
  const [startTime, setStartTime] = useState('');
  const [endTime, setEndTime] = useState('');
  const { userId } = useContext(UserContext);

  useEffect(() => {
    fetchData();
  }, []);

  const fetchData = async () => {
    try {
      const bot = await getBot(botData.bot_id);

      setBotName(bot.nombre);
      setStartTime(bot.hora_inicio_actividad);
      setEndTime(bot.hora_fin_actividad);
    } catch (error) {
      console.log(error);
    }
  };

  const updateData = async () => {
    try {
      const updatedBot = {
        usuario_id: parseInt(userId),
        nombre: botName,
        hora_inicio_actividad: startTime,
        hora_fin_actividad: endTime,
      };
      await updateBot(botData.bot_id, updatedBot);

      props.navigation.navigate('BotDetails', { data: updatedBot });
    } catch (error) {
      console.log(error);
    }
  };
};

```

```

return (
  <Layout>
    <View>
      <Text
        style={styles.inputStyle}
        label="User ID"
        value={userId}
        mode="outlined"
      />
      <TextInput
        style={styles.inputStyle}
        label="Bot Name"
        value={botName}
        mode="outlined"
        onChangeText={({text}) => setBotName(text)}
      />
      <TextInput
        style={styles.inputStyle}
        label="Start Time"
        value={startTime}
        mode="outlined"
        onChangeText={({text}) => setStartTime(text)}
      />
      <TextInput
        style={styles.inputStyle}
        label="End Time"
        value={endTime}
        mode="outlined"
        onChangeText={({text}) => setEndTime(text)}
      />
      <Button
        style={{ margin: 10 }}
        icon="pencil"
        mode="contained"
        onPress={updateData}
      >
        Update Bot
      </Button>
    </View>
  </Layout>
);
}

const styles = StyleSheet.create({
  inputStyle: {
    marginTop: 30,
    padding: 10,
  },
});

```

```
});  
  
export default BotEdit;
```

```

import React, { useEffect, useState, useContext } from 'react';
import { StyleSheet, FlatList, SafeAreaView, RefreshControl } from
"react-native";
import { useIsFocused, useNavigation } from "@react-navigation/native";
import { FAB } from 'react-native-paper';

import BotItem from "../components/BotItem";
import Layout from "../components/Layout";
import { getBots } from "../api/BotRoutes";

import { UserContext } from '../UserContext';

const BotsHome = () => {
  const navigation = useNavigation();
  const [bots, setBots] = useState([]);
  const [refreshing, setRefreshing] = useState(false);
  const isFocused = useIsFocused();
  const { userId } = useContext(UserContext);

  const loadData = async () => {
    try {
      const bots = await getBots(userId);
      setBots(bots);
    } catch (error) {
      console.log(error);
    }
  };

  const onRefresh = React.useCallback(async () => {
    setRefreshing(true);
    await loadData();
    setRefreshing(false);
  }, []);

  useEffect(() => {
    loadData();
  }, [isFocused]);

  const renderItem = (item) => {
    return <BotItem bot={item} />;
  };

  const handleCreateBot = () => {
    navigation.navigate('BotCreate');
  };

```

```

return (
  <Layout>
    <SafeAreaView style={styles.container}>
      <FlatList
        style={styles.flatList}
        data={bots}
        keyExtractor={({item}) => `${item.bot_id}`}
        renderItem={({ item }) => renderItem(item)}
        refreshControl={
          <RefreshControl
            refreshing={refreshing}
            onRefresh={onRefresh}
            colors={['#78e08f']}
            progressBackgroundColor="#0a3d62"
          />
        }
      />
    <FAB
      style={styles.fab}
      icon="plus"
      onPress={handleCreateBot}
    />
  </SafeAreaView>
</Layout>
);
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  flatList: {
    width: "100%",
  },
  fab: {
    position: "absolute",
    margin: 16,
    right: 0,
    bottom: 0,
  },
});

export default BotsHome;

```

```

import React from 'react';
import { View, StyleSheet } from 'react-native';
import { Card, Title } from 'react-native-paper';
import { useNavigation } from '@react-navigation/native';

// import { signOut } from '../api/AuthRoutes';
import Layout from '../components/Layout';

export default function HomeScreen() {
  const navigation = useNavigation();

  const handleViewBots = () => {
    navigation.navigate('BotsHome');
  };

  const handleCreateBot = () => {
    navigation.navigate('BotCreate');
  };

  const handleScheduleMessage = () => {
    navigation.navigate('ScheduleMessage');
  };

  const handleSignOut = async () => {
    console.log('handleSignOut');
    // await signOut();
    // navigation.reset({
    //   index: 0,
    //   routes: [{ name: 'Login' }],
    // });
  };

  return (
    <Layout>
      <Card onPress={handleViewBots} style={styles.card}>
        <Card.Content>
          <Title>View Bots</Title>
        </Card.Content>
      </Card>

      <Card onPress={handleCreateBot} style={styles.card}>
        <Card.Content>
          <Title>Create Bot</Title>
        </Card.Content>
      </Card>

      <Card onPress={handleScheduleMessage} style={styles.card}>

```

```

        <Card.Content>
          <Title>Schedule Message</Title>
        </Card.Content>
      </Card>

      <Card
        onPress={handleSignOut}
        style={[styles.card, styles.signOutCard]} // Estilo adicional
para Sign Out
      >
        <Card.Content>
          <Title style={styles.signOutText}>Sign Out</Title>
        </Card.Content>
      </Card>
    </Layout>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'flex-start', // Alineación en la parte superior
    alignItems: 'center',
    paddingTop: 40, // Espacio superior para separar del borde de la
pantalla
  },
  card: {
    width: '80%', // Ancho de la tarjeta
    marginVertical: 10, // Espaciado vertical entre las tarjetas
  },
  signOutCard: {
    backgroundColor: '#3076C4', // Color de fondo para Sign Out
  },
  signOutText: {
    color: 'white', // Color de texto para Sign Out
  },
});

```

```

import * as React from 'react';
import { useContext } from 'react';
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Button, Text, View, Image, TouchableOpacity } from
'react-native';
import * as WebBrowser from 'expo-web-browser';
import * as Google from 'expo-auth-session/providers/google';
import { useNavigation } from '@react-navigation/native';

import { saveUser, getUserByEmail } from '../api/UserRoutes';
import { WEB_CLIENT_ID, IOS_CLIENT_ID, ANDROID_CLIENT_ID } from
'../../Config';
import { UserContext } from '../UserContext';

// Ventana que nos va a permitir abrir la modal al momento de la
autenticación
WebBrowser.maybeCompleteAuthSession();

export default function LoginScreen({ onLoginSuccess }) {

  // Esta constante la usaremos para poder guardar y acceder a los datos
del usuario desde otras pantallas
  const { updateUserEmail, updateUserId } = useContext(UserContext);

  // Este accessToken nos ayuda a autenticarnos para obtener la
información de google del usuario que está autenticado
  const [accessToken, setAccessToken] = React.useState(null); //Lo
inicializamos a null ya que en principio no hay sesión iniciada
  // La siguiente variable user se usa para guardar la información del
usuario
  const [user, setUser] = React.useState(null); //Lo inicializamos a null
ya que en principio no tenemos usuario
  const [request, response, promptAsync] = Google.useAuthRequest({
    clientId: WEB_CLIENT_ID,
    iosClientId: IOS_CLIENT_ID,
    androidClientId: ANDROID_CLIENT_ID
  });

  // Obtenemos el objeto de navegación:
  const navigation = useNavigation();

  // Este use effect nos va a ayudar para que cuando la aplicación se
cargue podamos ver si el usuario tiene una sesión o no, o quiere iniciar
una sesión
  React.useEffect(() => {
    // En caso de que tengamos una respuesta, comprobamos si es success
    if(response?.type === "success"){

```



```

    //Se guarda la información del usuario en el accessToken:
    setAccessToken(response.authentication.accessToken);
    // De esta manera nos aseguramos de que fetchUserInfo se ejecute sí
y solo sí tenemos el accessToken (accessToken != null):
    accessToken && fetchUserInfo();
    onLoginSuccess && onLoginSuccess();
  }
}, [response, accessToken]) //Cada que cambie la response o el
accessToken se hace fetch de los datos del usuario (Esto es, cada que
haya cambio de login)

// Esta función obtiene un usuario desde el backend a partir de su
email y guarda su id con updateUserId
const fetchUserInfoByEmail = async (email) => {
  try {
    const userFromBack = await getUserByEmail(email);
    // Actualiza el ID de usuario en el contexto
    updateUserId(userFromBack.usuario_id);
  } catch (error) {
    console.log(error);
  }
};

//Con la siguiente función, obtenemos la información del usuario que se
loguea
//Es una función asíncrona porque se hace una request y tiene que
esperar
async function fetchUserInfo() {
  let response = await
fetch("https://www.googleapis.com/userinfo/v2/me", { //Esta dirección es
un endpoint por el que accedemos a la información del usuario
  // Como segundo parámetro necesitamos pasarle a la función la
autenticación del usuario (el token) en el objeto Authorization:
  headers: { Authorization: `Bearer ${accessToken}` }
});
  // La respuesta que nos arroja es un Json, así que lo parseamos:
  const userInfo = await response.json();
  //Guardamos la información en nuestra variable del usuario:
  setUser(userInfo);
  updateUserEmail(userInfo.email);
  fetchUserInfoByEmail(userInfo.email);
  sendUser(userInfo);
}

// Esta función nos ayudará a meter usuarios en la base de datos
const sendUser = async (user) => {
  try {
    await saveUser(user)
    .then(data => {

```

```

        console.log(data)
      })
      .catch(error => console.log(error))
      // const response = await axios.post('http://' + ip +
      ':3000/insertUsuario', user);
      // console.log(response.data);
    } catch (error) {
      console.error(error);
    }
  };

  // Este componente nos va a permitir renderizar la información del
  usuario si es que está logueado
  const ShowUserInfo = () => {
    if (user) { //Si no tenemos user, no renderizamos nada
      return ( //Aquí devolvemos una vista muy sencilla con un texto y
        una imagen: (user.name y user.profile)
        <View style={{ flex: 1, alignItems: 'center', justifyContent:
        'center' }}>
          <Text style={{ fontSize: 35, fontWeight: 'bold', marginBottom:
          20 }}>Welcome</Text>
          <Image source={{ uri: user.picture }} style={{ width: 100,
          height: 100, borderRadius: 50 }} />
          <Text style={{ fontSize: 20, fontWeight: 'bold'
          }}>{user.name}</Text>
          { /* Quizás me plantee cambiar este TouchableOpacity por un
          button */}
          <TouchableOpacity style={styles.button} onPress={() =>
          navigation.navigate('HomeScreen')}>
            <Text style={styles.buttonText}>Continue</Text>
          </TouchableOpacity>
        </View>
      )
    }
  }

  return (
    //Solo se llama a ShowUserInfo si tenemos un usuario logeado y el
    mostrar este componente también depende de que haya usuario (true)
    <View style={styles.container}>
      {user && <ShowUserInfo />}
      {user === null && //Si no tenemos un usuario, cargamos este
      componente para dar opción de login
      <>
        <Text style={{ fontSize: 35, fontWeight: 'bold'
        }}>Welcome</Text>
        <Text style={{ fontSize: 25, fontWeight: 'bold', marginBottom:
        20, color: 'gray' }}>Please login</Text>
      </>
    </View>
  )
}

```

```

        <TouchableOpacity //Esto es un componente que captura si el
usuario toca en esta parte
        disabled={!request}
        onPress={() => {
            promptAsync();
        }}
    >
        <Image source={require("../assets/images/login/btn.jpeg")}
style={{ width: 300, height: 40 }} />
    </TouchableOpacity>
</>
}
</View>
);
}

const styles = StyleSheet.create({
    container: {
        flex: 1,
        backgroundColor: '#fff',
        alignItems: 'center',
        justifyContent: 'center',
    },
});

```

c. Componentes

```
import React, { useEffect, useState } from 'react';
import { TouchableOpacity, View, Text, StyleSheet, Dimensions } from
'react-native';
import { useNavigation } from '@react-navigation/native';
import { Card } from 'react-native-paper';

import { getBot } from '../api/BotRoutes';
import { getPlantilla } from '../api/PlantillaRoutes';
import { getPalabraClave } from '../api/PalabraClaveRoutes';

const AutomaticResponseItem = ({ response }) => {
  const navigation = useNavigation();
  const [botName, setBotName] = useState('');
  const [templateName, setTemplateName] = useState('');
  const [keyword, setKeyword] = useState('');

  useEffect(() => {
    fetchBotName();
    fetchTemplateName();
    fetchKeyword();
  }, []);

  const fetchBotName = async () => {
    try {
      const bot = await getBot(response.bot_id);
      setBotName(bot.nombre);
    } catch (error) {
      console.log(error);
    }
  };

  const fetchTemplateName = async () => {
    try {
      const plantilla = await getPlantilla(response.plantilla_id);
      setTemplateName(plantilla.nombre);
    } catch (error) {
      console.log(error);
    }
  };

  const fetchKeyword = async () => {
    try {
      const palabraClave = await
getPalabraClave(response.palabra_clave_id);
```

```

        setKeyword(palabraClave.palabra_clave);
    } catch (error) {
        console.log(error);
    }
};

return (
    <TouchableOpacity onPress={() =>
navigation.navigate('AutomaticResponseDetails', { data: response })}>
        <Card style={[styles.cardStyle, { width: cardWidth }]}>
            <Text style={styles.textStyle1}>Response ID:
{response.respuesta_id}</Text>
            <Text>Bot Name: {botName}</Text>
            <Text>Template Name: {templateName}</Text>
            <Text>Keyword: {keyword}</Text>
        </Card>
    </TouchableOpacity>
);
};

const cardWidth = Dimensions.get('window').width * 0.9;

const styles = StyleSheet.create({
    cardStyle: {
        marginVertical: 8,
        padding: 16,
    },
    textStyle: {
        fontSize: 16,
    },
});

export default AutomaticResponseItem;

```

```

import React from "react";
import { TouchableOpacity, View, Text, StyleSheet, Dimensions } from
"react-native";
import { useNavigation } from "@react-navigation/native";
import { Card } from "react-native-paper";

const BotItem = ({ bot }) => {
  const navigation = useNavigation();

  const handleBotPress = () => {
    navigation.navigate("BotDetails", { data: bot });
  };

  const cardWidth = Dimensions.get("window").width * 0.9; // Calcula el
ancho de la tarjeta

  return (
    <TouchableOpacity onPress={handleBotPress}>
      <Card style={[styles.cardStyle, { width: cardWidth }]}>
        <Text style={styles.textStyle}>{bot.nombre}</Text>
      </Card>
    </TouchableOpacity>
  );
};

const styles = StyleSheet.create({
  cardStyle: {
    marginVertical: 8,
    padding: 16,
  },
  textStyle: {
    fontSize: 16,
  },
});

export default BotItem;

```

```

import React from "react";
import { View, StatusBar, StyleSheet } from "react-native";

const Layout = ({ children }) => {
  return (
    <View style={styles.container}>
      <StatusBar backgroundColor="#222f3e" />
      {children}
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    paddingTop: 5,
    backgroundColor: "#222f3e",
    flex: 1,
    alignItems: "center",
  },
  title: {
    color: "ffffff",
    fontSize: 20,
    textAlign: "center",
    marginTop: 10,
  },
});

export default Layout;

```

```
import React, { createContext, useState } from 'react';

export const UserContext = createContext();

export const UserProvider = ({ children }) => {
  const [userEmail, setUserEmail] = useState('');
  const [userId, setUserId] = useState('');

  const updateUserEmail = (email) => {
    setUserEmail(email);
  };

  const updateUserId = (id) => {
    setUserId(id);
  };

  return (
    <UserContext.Provider value={{ userEmail, updateUserEmail, userId,
updateUserId }}>
      {children}
    </UserContext.Provider>
  );
};
```


d. Funciones de rutas

```
import { api_ip } from "../../Config";
const API = api_ip + '/bots'; // Actualiza la ruta base para los bots

export const deleteBot = async (botId) => {
  await fetch(`${API}/${botId}`, {
    method: "DELETE",
  });
};

export const getBots = async (userId) => {
  const res = await fetch(`${API}/${userId}`, {
    method: "GET",
  });

  return await res.json();
};

export const saveBot = async (newBot) => {
  const res = await fetch(`${API}`, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newBot),
  });
  return await res.json();
};

export const getBot = async (botId) => {
  const res = await fetch(`${API}/bot/${botId}`);
  const bot = await res.json();
  return bot;
};

export const updateBot = async (botId, updatedBot) => {
  const res = await fetch(`${API}/${botId}`, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedBot),
  });
};
```

```
    return res;  
};
```

```

import { api_ip } from "../../Config";
const API = api_ip + '/clientes'; // Actualiza la ruta base para los
clientes

export const deleteCliente = async (clienteId) => {
  await fetch(`${API}/${clienteId}`, {
    method: "DELETE",
  });
};

export const getClientesByUser = async (userId) => {
  const res = await fetch(`${API}/user/${userId}`, {
    method: "GET",
  });

  return await res.json();
};

export const saveCliente = async (newCliente) => {
  const res = await fetch(`${API}`, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newCliente),
  });
  return await res.json();
};

export const getCliente = async (clienteId) => {
  const res = await fetch(`${API}/${clienteId}`);
  const cliente = await res.json();
  return cliente;
};

export const updateCliente = async (clienteId, updatedCliente) => {
  const res = await fetch(`${API}/${clienteId}`, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedCliente),
  });
  return res;
};

```

```

import { api_ip } from "../../Config";
const API = api_ip + '/mensajes'; // Actualiza la ruta base para los mensajes

export const deleteMensaje = async (mensajeId) => {
  await fetch(`${API}/${mensajeId}`, {
    method: "DELETE",
  });
};

export const getMensajesByBot = async (botId) => {
  const res = await fetch(`${API}/bot/${botId}`, {
    method: "GET",
  });

  return await res.json();
};

export const saveMensaje = async (newMensaje) => {
  const res = await fetch(`${API}`, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newMensaje),
  });
  return await res.json();
};

export const getMensaje = async (mensajeId) => {
  const res = await fetch(`${API}/${mensajeId}`);
  const mensaje = await res.json();
  return mensaje;
};

export const updateMensaje = async (mensajeId, updatedMensaje) => {
  const res = await fetch(`${API}/${mensajeId}`, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedMensaje),
  });
  return res;
};

```

```

import { api_ip } from "../../Config";
const API = api_ip + '/palabras_clave'; // Actualiza la ruta base para
las palabras clave

export const deletePalabraClave = async (palabraClaveId) => {
  await fetch(`${API}/${palabraClaveId}`, {
    method: "DELETE",
  });
};

export const getPalabrasClaveByBot = async (botId) => {
  const res = await fetch(`${API}/bot/${botId}`, {
    method: "GET",
  });

  return await res.json();
};

export const savePalabraClave = async (newPalabraClave) => {
  const res = await fetch(`${API}`, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newPalabraClave),
  });
  return await res.json();
};

export const getPalabraClave = async (palabraClaveId) => {
  const res = await fetch(`${API}/${palabraClaveId}`);
  const palabraClave = await res.json();
  return palabraClave;
};

export const updatePalabraClave = async (palabraClaveId,
updatedPalabraClave) => {
  const res = await fetch(`${API}/${palabraClaveId}`, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedPalabraClave),
  });
  return res;
};

```

```

import { api_ip } from "../../Config";
const API = api_ip + '/plantillas'; // Actualiza la ruta base para las
plantillas

export const deletePlantilla = async (plantillaId) => {
  await fetch(`${API}/${plantillaId}`, {
    method: "DELETE",
  });
};

export const getPlantillasByBot = async (botId) => {
  const res = await fetch(`${API}/bot/${botId}`, {
    method: "GET",
  });

  return await res.json();
};

export const savePlantilla = async (newPlantilla) => {
  const res = await fetch(`${API}`, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newPlantilla),
  });
  return await res.json();
};

export const getPlantilla = async (plantillaId) => {
  const res = await fetch(`${API}/${plantillaId}`);
  const plantilla = await res.json();
  return plantilla;
};

export const updatePlantilla = async (plantillaId, updatedPlantilla) => {
  const res = await fetch(`${API}/${plantillaId}`, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedPlantilla),
  });
  return res;
};

```

```

import { api_ip } from "../../Config";
const API = api_ip + '/programaciones'; // Actualiza la ruta base para
las programaciones

export const deleteProgramacion = async (programacionId) => {
  await fetch(`${API}/${programacionId}`, {
    method: "DELETE",
  });
};

export const getProgramacionesByBot = async (botId) => {
  const res = await fetch(`${API}/bot/${botId}`, {
    method: "GET",
  });

  return await res.json();
};

export const saveProgramacion = async (newProgramacion) => {
  const res = await fetch(`${API}`, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newProgramacion),
  });
  return await res.json();
};

export const getProgramacion = async (programacionId) => {
  const res = await fetch(`${API}/${programacionId}`);
  const programacion = await res.json();
  return programacion;
};

export const updateProgramacion = async (programacionId,
updatedProgramacion) => {
  const res = await fetch(`${API}/${programacionId}`, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedProgramacion),
  });
  return res;
};

```

```

import { api_ip } from "../../Config";
const API = api_ip + '/respuestas_automaticas'; // Actualiza la ruta base
para las respuestas automáticas

export const deleteRespuestaAutomatica = async (respuestaId) => {
  await fetch(`${API}/${respuestaId}`, {
    method: "DELETE",
  });
};

export const getRespuestasAutomaticasByBot = async (botId) => {
  const res = await fetch(`${API}/bot/${botId}`, {
    method: "GET",
  });

  return await res.json();
};

export const saveRespuestaAutomatica = async (newRespuestaAutomatica) =>
{
  const res = await fetch(`${API}`, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newRespuestaAutomatica),
  });
  return await res.json();
};

export const getRespuestaAutomatica = async (respuestaId) => {
  const res = await fetch(`${API}/${respuestaId}`);
  const respuestaAutomatica = await res.json();
  return respuestaAutomatica;
};

export const updateRespuestaAutomatica = async (respuestaId,
updatedRespuestaAutomatica) => {
  const res = await fetch(`${API}/${respuestaId}`, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedRespuestaAutomatica),
  });
  return res;
};

```



```

//import { ip } from './Config';
//const API = 'http://192.168.10.22:3000/tasks';
//const API = 'http://' + ip + ':3000/tasks'
import { api_ip } from "../Config";
const API = api_ip + '/users'; // Actualiza la ruta base para los
usuarios

export const deleteUser = async (userId) => {
  await fetch(`${API}/${userId}`, {
    method: "DELETE",
  });
};

export const getUsers = async () => {
  const res = await fetch(API, {
    method: "GET",
  });

  return await res.json();
};

export const saveUser = async (newUser) => {
  const res = await fetch(API, {
    method: "POST",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(newUser),
  });
  return await res.json();
};

export const getUser = async (userId) => {
  const res = await fetch(`${API}/${userId}`);
  const user = await res.json();
  return user;
};

export const updateUser = async (userId, updatedUser) => {
  const res = await fetch(`${API}/${userId}`, {
    method: "PUT",
    headers: {
      Accept: "application/json",
      "Content-Type": "application/json",
    },
    body: JSON.stringify(updatedUser),
  });
};

```

```
});  
return res;  
};  
  
export const getUserByEmail = async (email) => {  
  const res = await fetch(`${API}/email/${email}`);  
  const user = await res.json();  
  return user;  
};
```

II. Anexo II: Justificación de las pruebas de funcionamiento

Pruebas de inicio de sesión:

Prueba con un primer usuario:

10:25

68

Welcome



Luis Fernando Moreno

Continue



Ilustración 15: Inicio de sesión

Logs generados:

<Response 163 bytes [200 OK]>

192.168.1.143 - - [19/Jun/2023 10:24:50] "GET /users/email/luisfergoza@gmail.com HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 10:24:50] "POST /users HTTP/1.1" 200 -

Prueba con un segundo usuario:

10:29

Bluetooth, cellular signal, Wi-Fi, and battery status icons.

Welcome



Luis Fernando Moreno Gonzalez

Continue



Ilustración 16: Inicio de sesión

Logs generados:

<Response 174 bytes [200 OK]>

192.168.1.143 - - [19/Jun/2023 10:29:20] "GET /users/email/moreno.gon.lf@gmail.com HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 10:29:20] "POST /users HTTP/1.1" 200 -

Pruebas de navegación por la página:

Carga de Home screen:

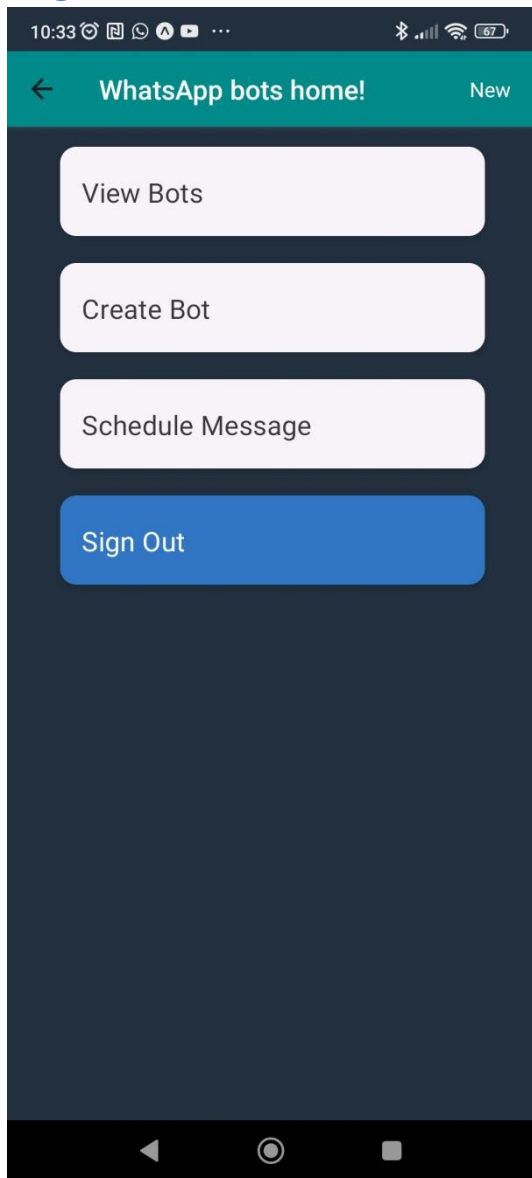


Ilustración 17: Home Screen

No se producen logs

Carga de bots del usuario:

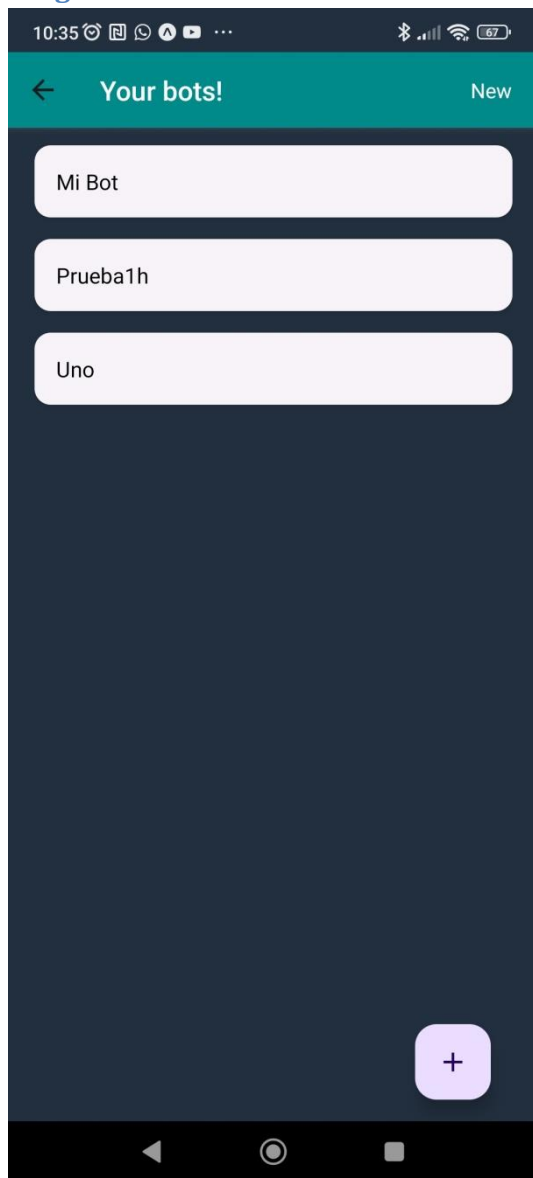


Ilustración 18: Bots del usuario

Log generado:

192.168.1.143 - - [19/Jun/2023 10:35:13] "GET /bots/1 HTTP/1.1" 200 -

Selección de un bot:

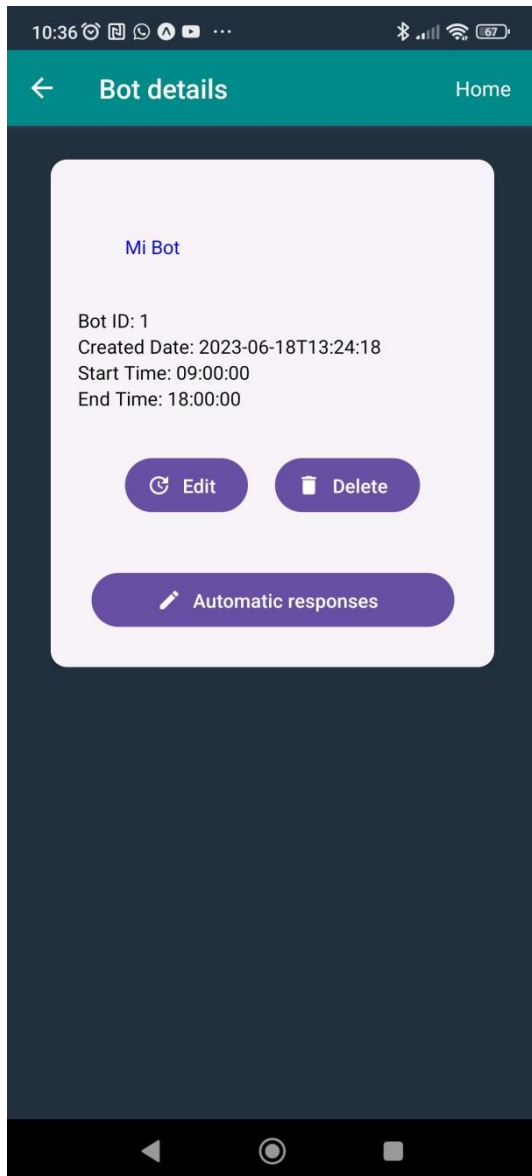


Ilustración 19: Detalles del bot

Log generado:

192.168.1.143 - - [19/Jun/2023 10:36:36] "GET /bots/1 HTTP/1.1" 200 -

Selección de las respuestas automáticas del bot:

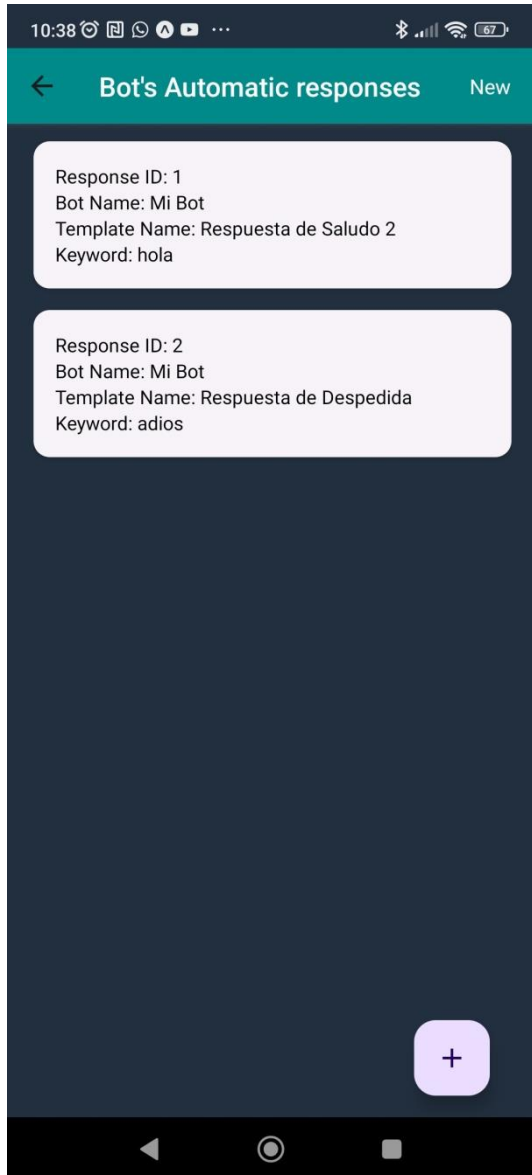


Ilustración 20: Respuestas de un bot

Logs generados:

```
192.168.1.143 - - [19/Jun/2023 10:38:46] "GET /respuestas_automaticas/bot/1 HTTP/1.1" 200
192.168.1.143 - - [19/Jun/2023 10:38:46] "GET /bots/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 10:38:46] "GET /bots/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 10:38:47] "GET /plantillas/1 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 10:38:47] "GET /plantillas/2 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 10:38:47] "GET /palabras_clave/2 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 10:38:47] "GET /palabras_clave/1 HTTP/1.1" 200 -
```


Selección de una respuesta automática:

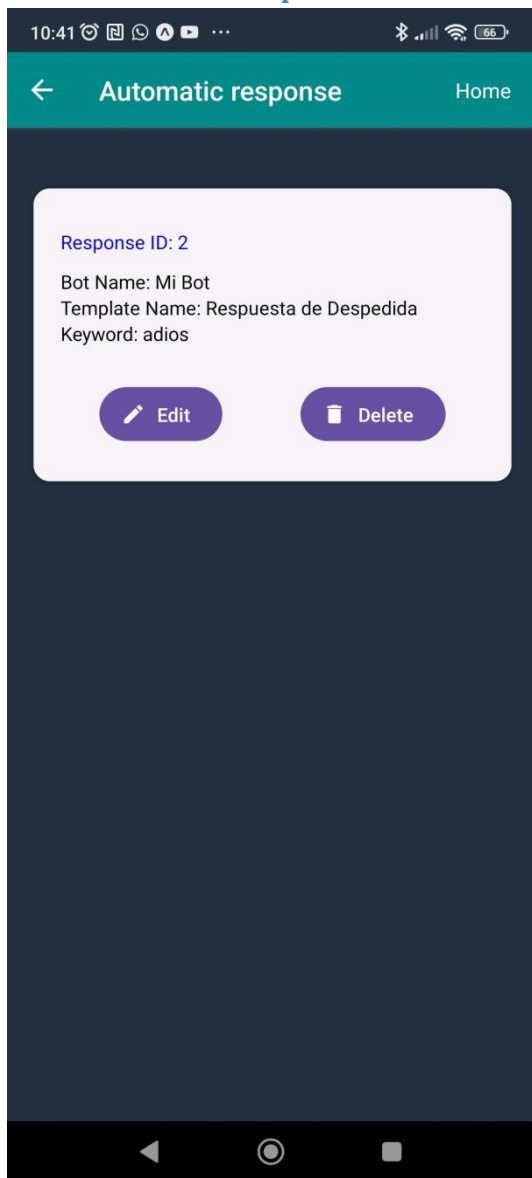


Ilustración 21: Detalles de una respuesta

Logs generados:

192.168.1.143 - - [19/Jun/2023 10:41:16] "GET /bots/bot/1 HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 10:41:16] "GET /palabras_clave/2 HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 10:41:16] "GET /plantillas/2 HTTP/1.1" 200 -

Creación de una nueva respuesta automática:

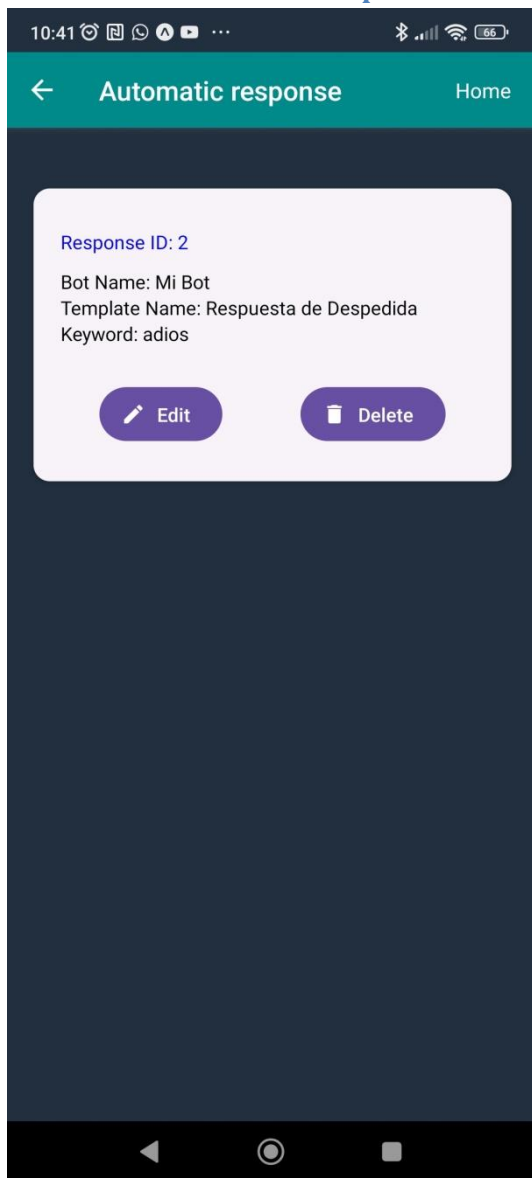


Ilustración 22: Detalles de una respuesta

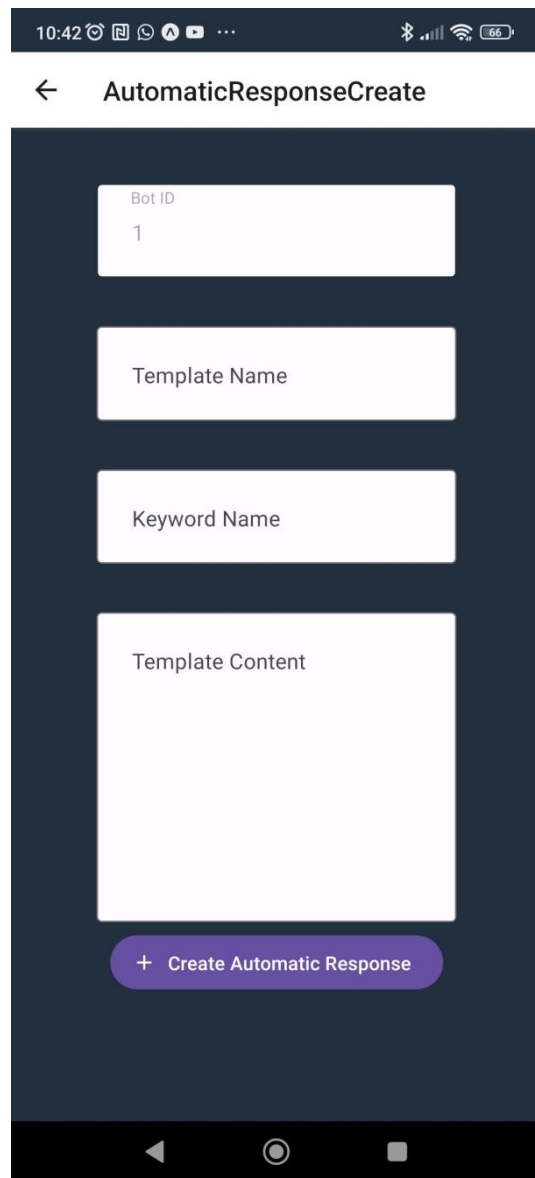


Ilustración 23: Creación de nueva respuesta

No se generan logs

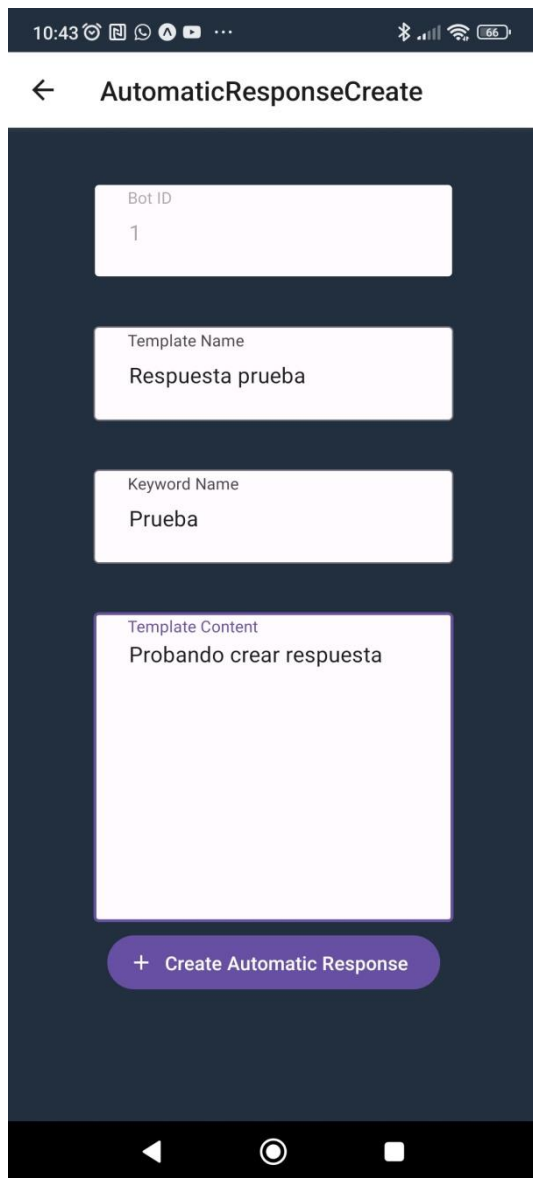


Ilustración 24: Creación de nueva respuesta

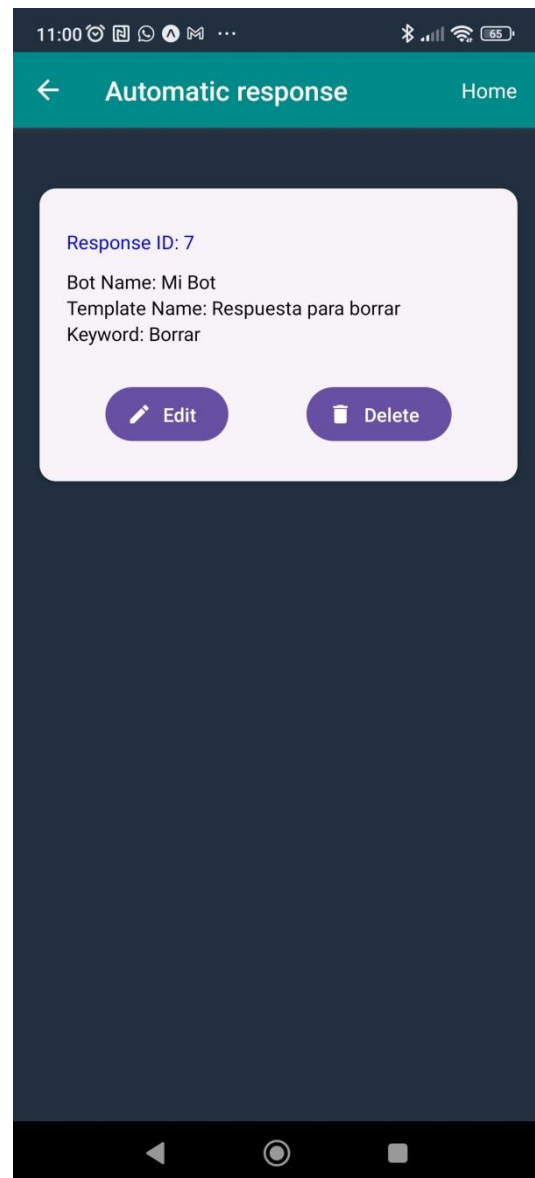


Ilustración 25 Nueva respuesta creada

Logs generados:

192.168.1.143 - - [19/Jun/2023 10:43:53] "POST /plantillas HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 10:43:53] "POST /palabras_clave HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 10:43:54] "POST /respuestas_automaticas HTTP/1.1" 200 -

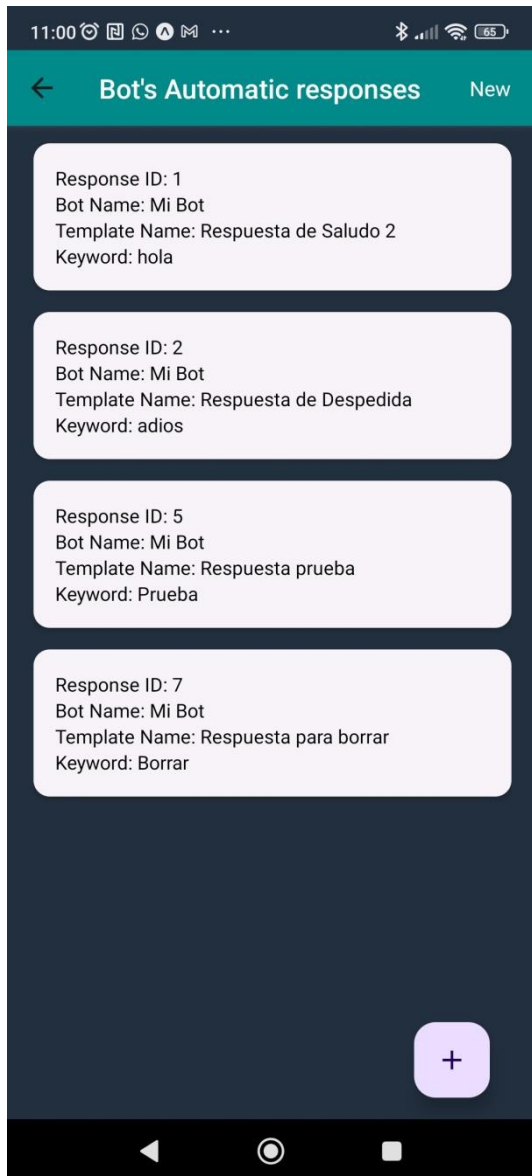


Ilustración 26: Nueva respuesta listada

Logs generados:

```
192.168.1.143 - - [19/Jun/2023 10:44:26] "GET /respuestas_automaticas/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 10:44:26] "GET /plantillas/5 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 10:44:26] "GET /bots/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 10:44:26] "GET /palabras_clave/5 HTTP/1.1" 200 -
```

Eliminación de una respuesta automática:

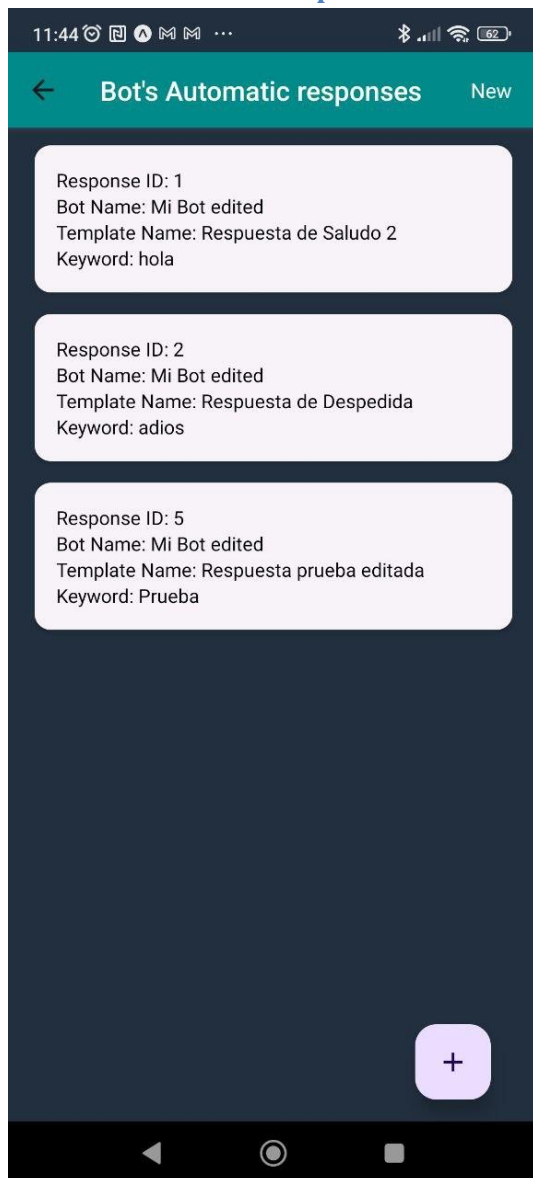


Ilustración 27: Nueva respuesta eliminada

Logs generados:

```
192.168.1.143 - - [19/Jun/2023 11:00:22] "DELETE /respuestas_automaticas/7 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 11:00:22] "GET /plantillas/7 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 11:00:22] "GET /palabras_clave/7 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 11:00:22] "GET /bots/bot/1 HTTP/1.1" 200 -
192.168.1.143 - - [19/Jun/2023 11:00:26] "GET /respuestas_automaticas/bot/1 HTTP/1.1" 200 -
```

Edición de una respuesta automática:

11:02 [status icons] 65%

← AutomaticResponseEdit

Bot ID
1

Template Name
Respuesta prueba editada

Keyword Name
Prueba

Template Content
Probando crear respuesta

Update Automatic Response

[navigation bar]

Ilustración 28: Editando respuesta

11:02 [status icons] 65%

← AutomaticResponseEdit

Bot ID
1

Template Name
Respuesta prueba

Keyword Name
Prueba

Template Content
Probando crear respuesta

Update Automatic Response

[navigation bar]

Ilustración 29: Editando respuesta

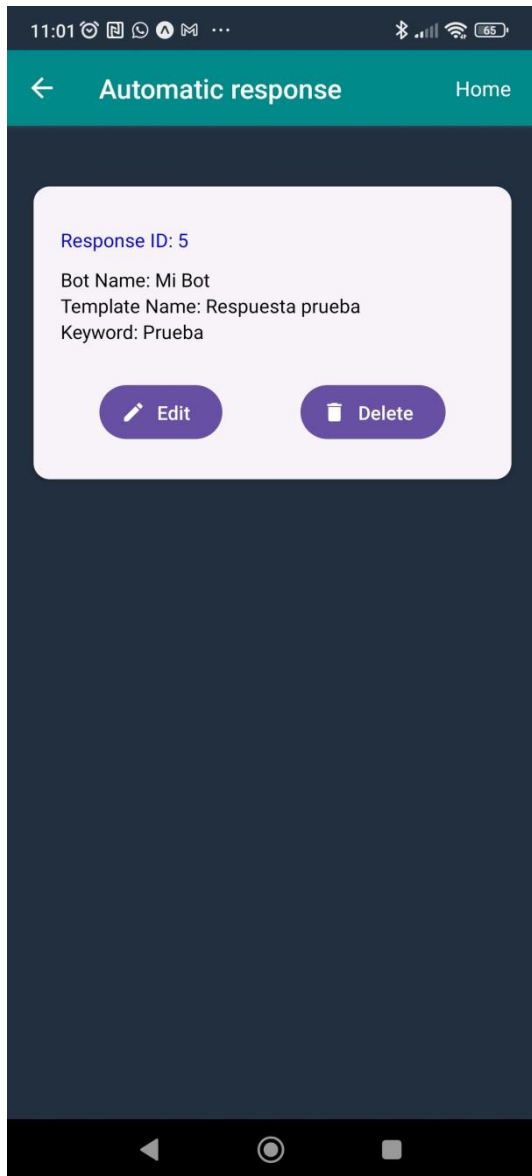


Ilustración 30: Respuesta editada

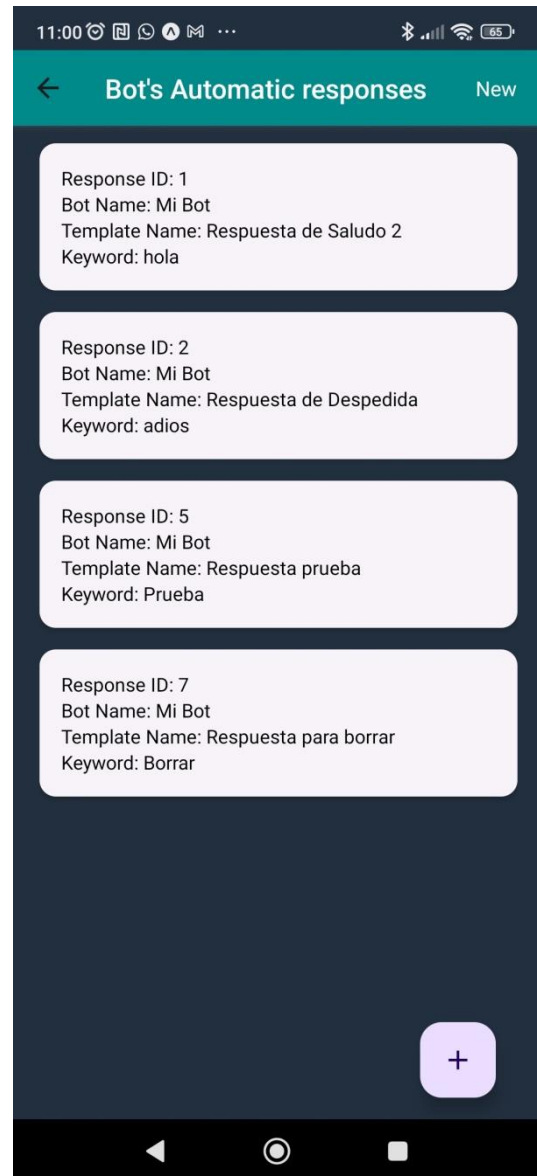


Ilustración 31: Respuesta editada listada

Logs generados:

192.168.1.143 -- [19/Jun/2023 11:02:18] "PUT /plantillas/5 HTTP/1.1" 200 -

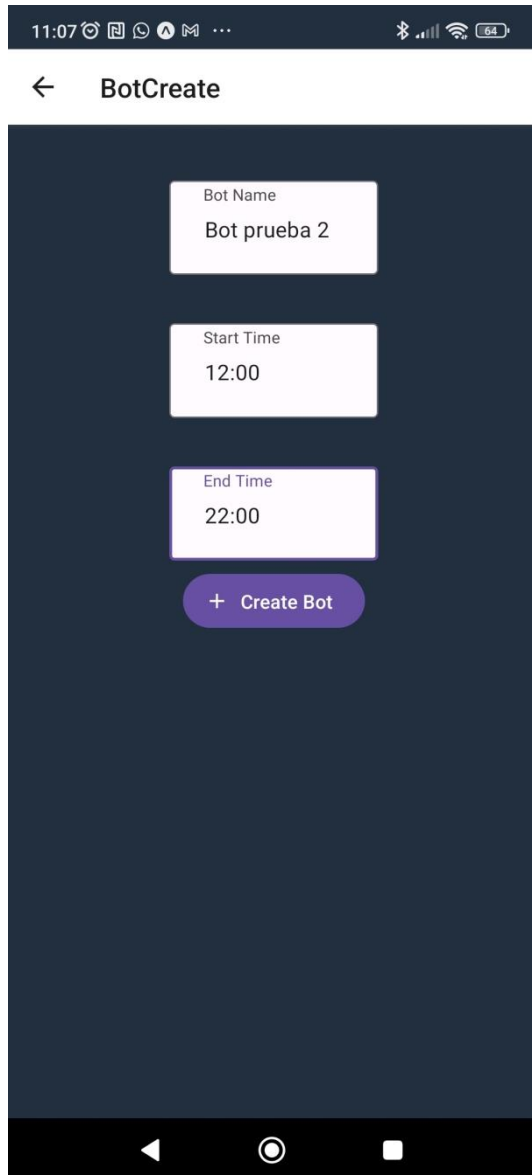
192.168.1.143 -- [19/Jun/2023 11:02:18] "PUT /palabras_clave/5 HTTP/1.1" 200 -

192.168.1.143 -- [19/Jun/2023 11:02:18] "GET /plantillas/5 HTTP/1.1" 200 -

192.168.1.143 -- [19/Jun/2023 11:02:18] "GET /bots/bot/1 HTTP/1.1" 200 -

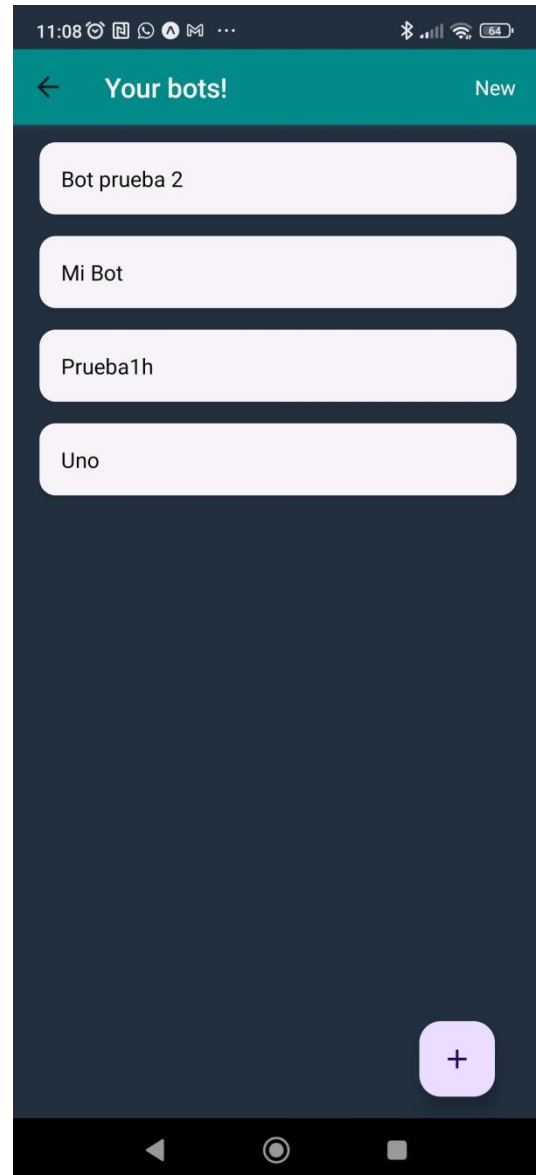
192.168.1.143 -- [19/Jun/2023 11:02:18] "GET /palabras_clave/5 HTTP/1.1" 200 -

Creación de nuevo bot:



The screenshot shows a mobile app interface for creating a new bot. At the top, there's a status bar with the time 11:07 and various icons. Below it, a header bar with a back arrow and the text 'BotCreate'. The main area has three input fields: 'Bot Name' with the value 'Bot prueba 2', 'Start Time' with the value '12:00', and 'End Time' with the value '22:00'. At the bottom, there's a purple button with a plus sign and the text '+ Create Bot'.

Ilustración 32: Creación de nuevo Bot



The screenshot shows a mobile app interface for viewing created bots. At the top, there's a status bar with the time 11:08 and various icons. Below it, a header bar with a back arrow, the text 'Your bots!', and a 'New' button. The main area displays a list of four bots in light purple rounded rectangles: 'Bot prueba 2', 'Mi Bot', 'Prueba1h', and 'Uno'. At the bottom right, there's a purple button with a plus sign.

Ilustración 33: Nuevo bot creado

Logs generados:

192.168.1.143 - - [19/Jun/2023 11:07:20] "GET /bots/1 HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 11:07:25] "GET /bots/1 HTTP/1.1" 200 -

{'usuario_id': 1, 'nombre': 'Bot prueba 2', 'hora_inicio_actividad': '12:00', 'hora_fin_actividad': '22:00'}

192.168.1.143 - - [19/Jun/2023 11:08:00] "POST /bots HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 11:08:00] "GET /bots/1 HTTP/1.1" 200 -

Eliminación de bot:

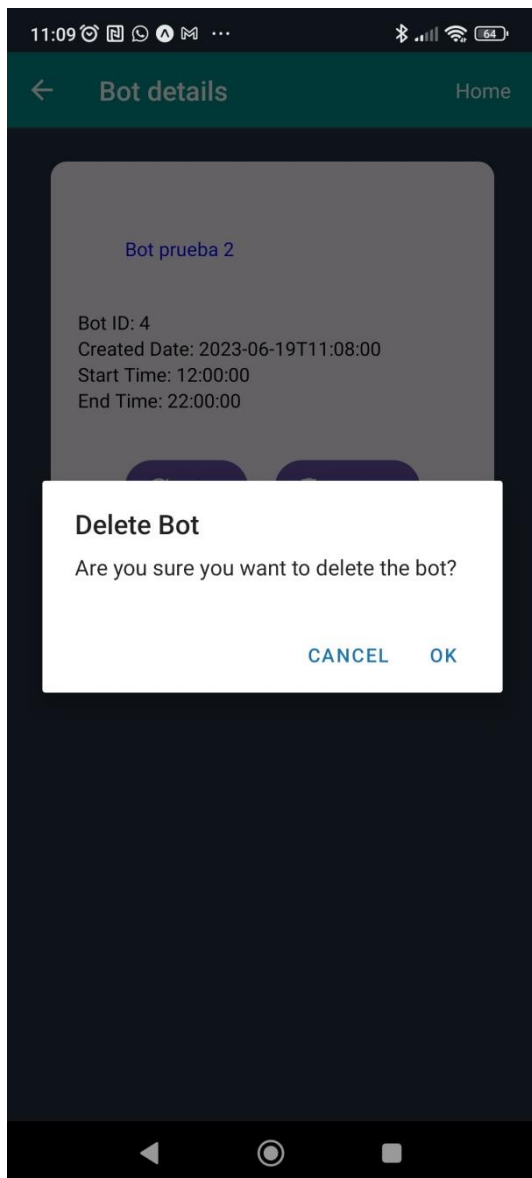


Ilustración 34: Eliminando bot

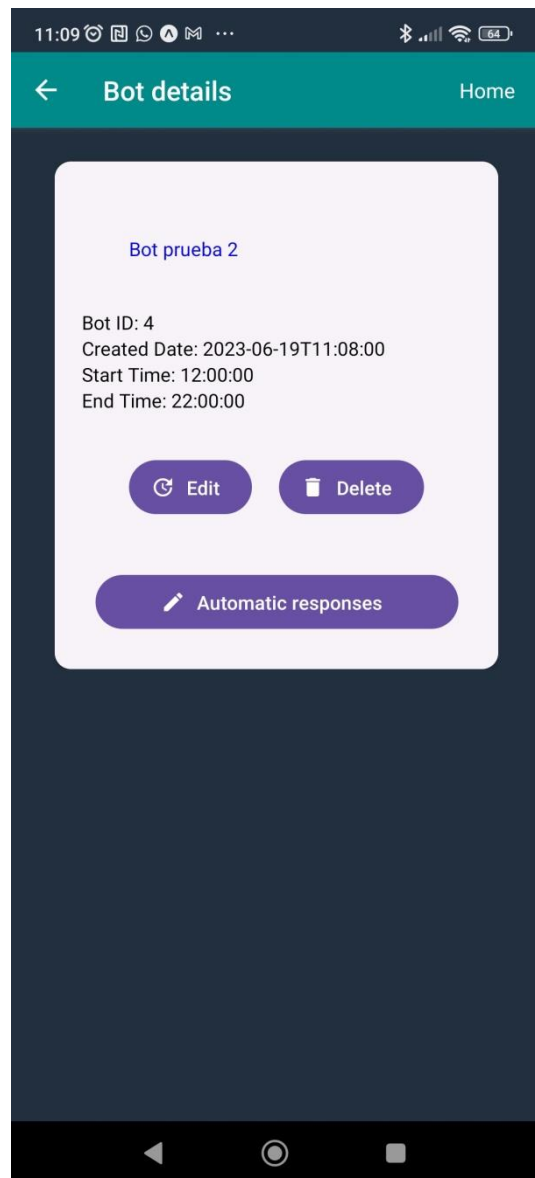


Ilustración 35: Detalles de bot eliminado

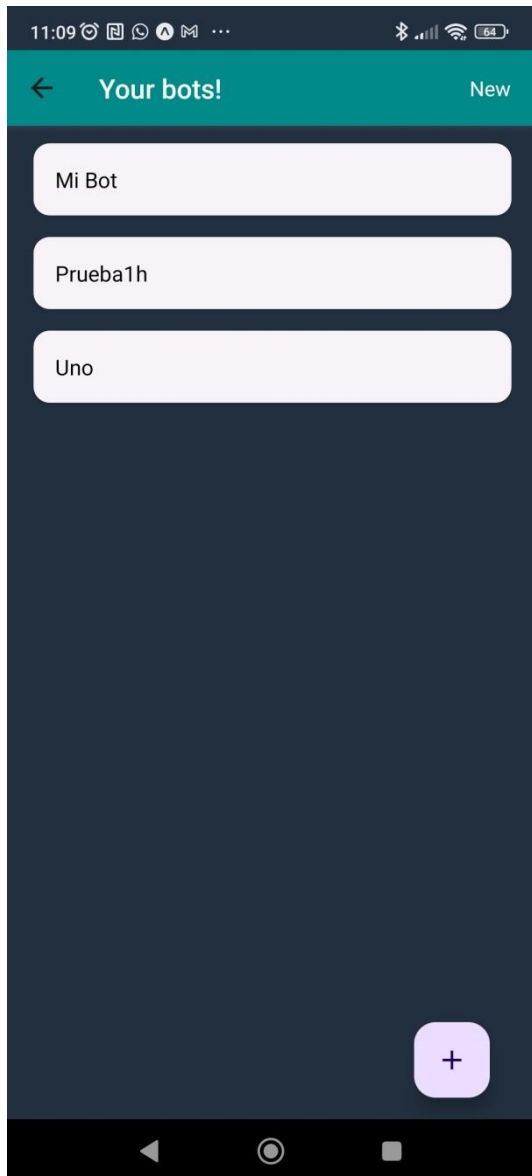


Ilustración 36: Nuevo listado de bots

Logs generados:

192.168.1.143 - - [19/Jun/2023 11:09:25] "GET /bots/1 HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 11:09:35] "DELETE /bots/4 HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 11:09:35] "GET /bots/1 HTTP/1.1" 200 -

Edición de bot:

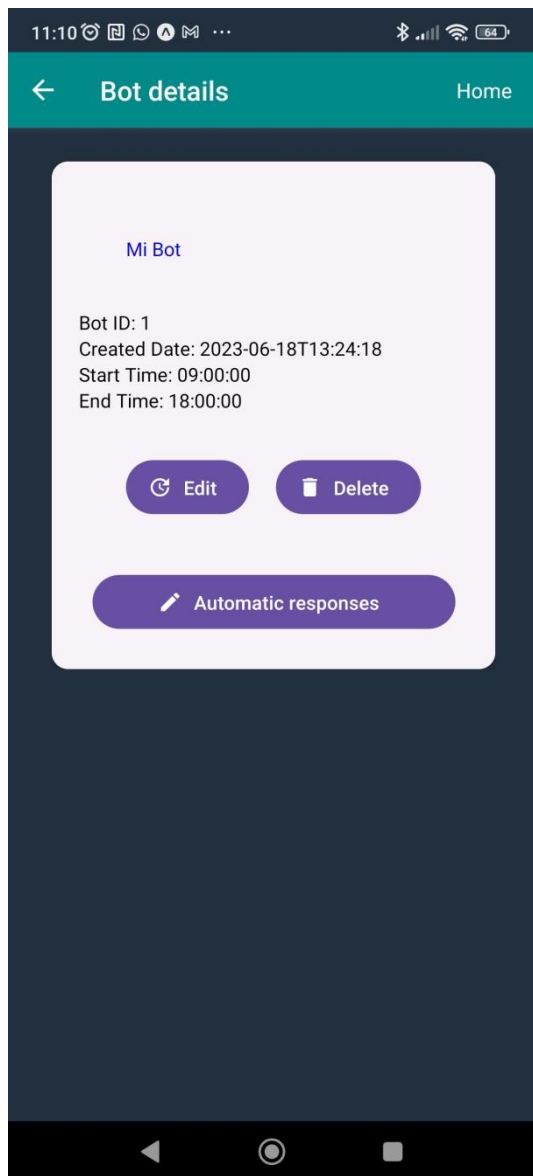


Ilustración 37: Bot a editar

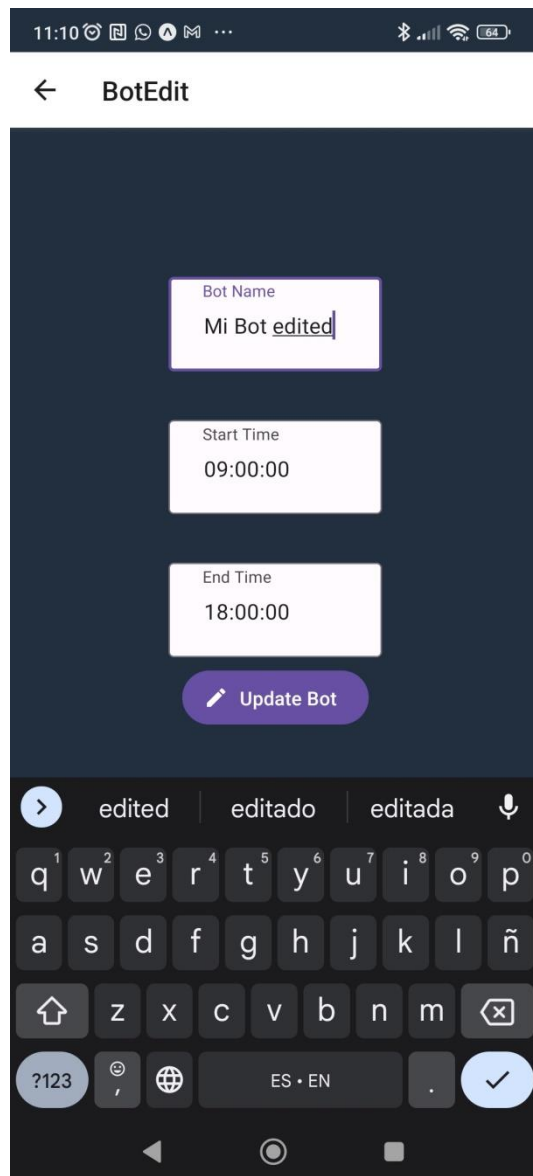


Ilustración 38: Editando bot

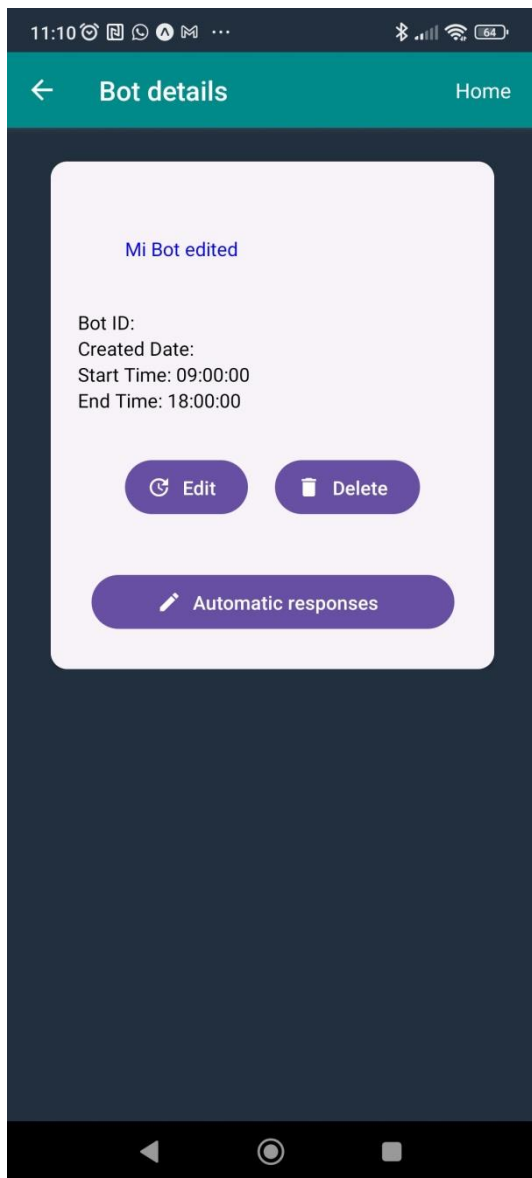


Ilustración 39: Detalles de bot editado

Logs generados:

192.168.1.143 - - [19/Jun/2023 11:10:25] "GET /bots/1 HTTP/1.1" 200 -

192.168.1.143 - - [19/Jun/2023 11:10:31] "GET /bots/bot/1 HTTP/1.1" 200 -

<Response 196 bytes [200 OK]>

```
{'usuario_id': 1, 'nombre': 'Mi Bot edited', 'hora_inicio_actividad': '09:00:00',
'hora_fin_actividad': '18:00:00'}
```

<Response 203 bytes [200 OK]>

<Response 203 bytes [200 OK]>

192.168.1.143 - - [19/Jun/2023 11:10:43] "PUT /bots/1 HTTP/1.1" 200 -

Pruebas de funcionamiento del bot:

Las respuestas que tiene configuradas el bot son:

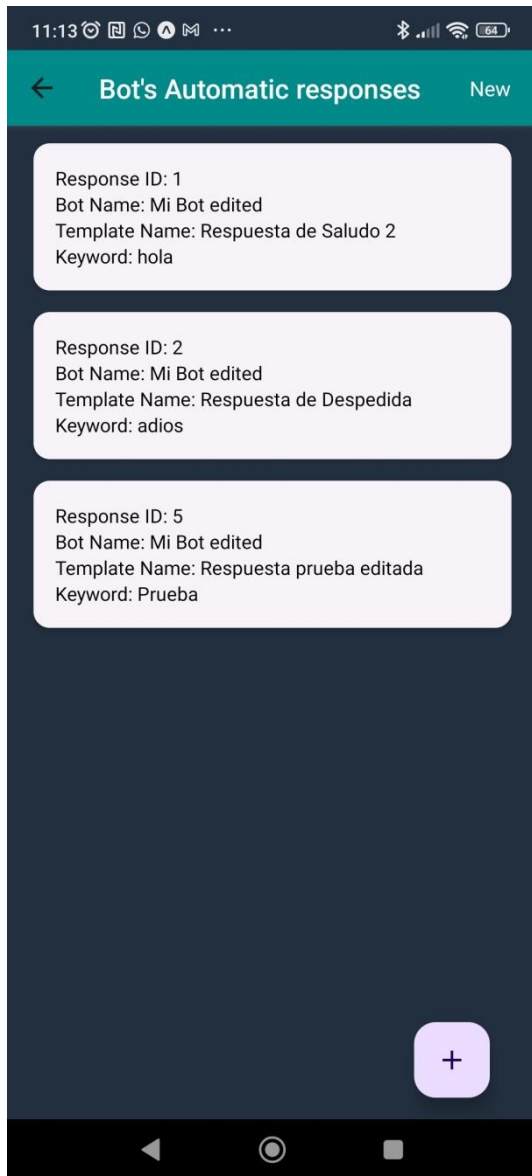


Ilustración 40: Respuestas configuradas del bot

Hay que tener en cuenta que en la imagen aparece el nombre de las respuestas, mas no el contenido de estas.

Enviando mensajes de WhatsApp al número configurado en Twilio:

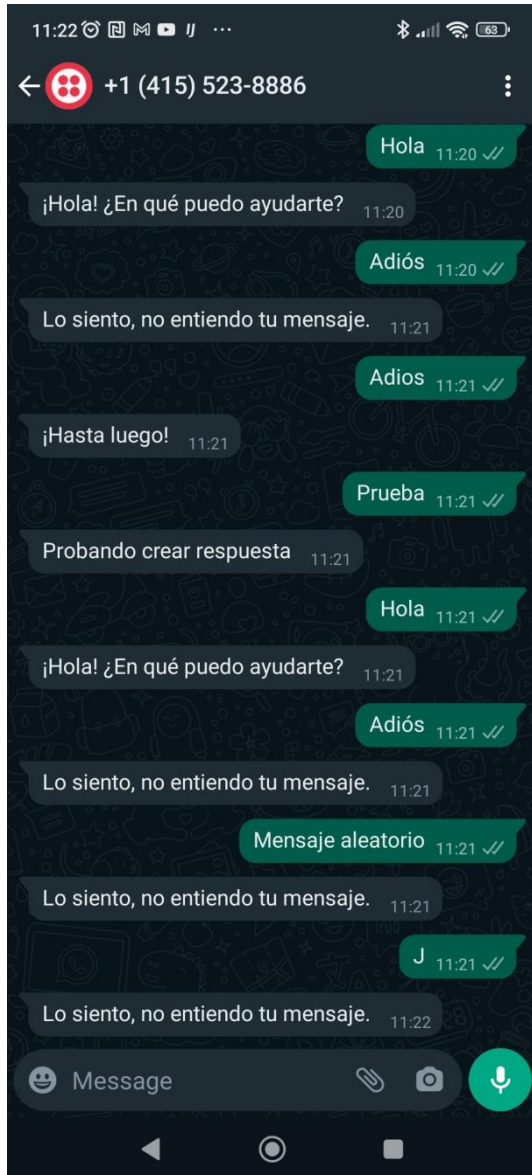


Ilustración 41: Mensajes de WhatsApp gestionados por el bot

Logs generados:

```
127.0.0.1 -- [19/Jun/2023 11:17:06] "POST /bot HTTP/1.1" 200 -
127.0.0.1 -- [19/Jun/2023 11:17:30] "POST /bot HTTP/1.1" 200 -
127.0.0.1 -- [19/Jun/2023 11:17:44] "POST /bot HTTP/1.1" 200 -
127.0.0.1 -- [19/Jun/2023 11:19:28] "POST /bot HTTP/1.1" 200 -
127.0.0.1 -- [19/Jun/2023 11:20:16] "POST /bot HTTP/1.1" 200 -
127.0.0.1 -- [19/Jun/2023 11:20:58] "POST /bot HTTP/1.1" 200 -
```

127.0.0.1 - - [19/Jun/2023 11:21:05] "POST /bot HTTP/1.1" 200 -
127.0.0.1 - - [19/Jun/2023 11:21:15] "POST /bot HTTP/1.1" 200 -
127.0.0.1 - - [19/Jun/2023 11:21:22] "POST /bot HTTP/1.1" 200 -
127.0.0.1 - - [19/Jun/2023 11:21:27] "POST /bot HTTP/1.1" 200 -
127.0.0.1 - - [19/Jun/2023 11:21:51] "POST /bot HTTP/1.1" 200 -
127.0.0.1 - - [19/Jun/2023 11:21:58] "POST /bot HTTP/1.1" 200 -