



SECRETARÍA
DE INNOVACIÓN

Escuela Superior de Innovación y Tecnología



Técnico Superior Universitario Servicios en la Nube

Estancia Profesional

Entregable fase 2

Diseño de flujos conversacionales y Arquitectura

Chatbot Asistente en la Nube para Atención Automatizada (Lite)

Docente: **Carlos Guillermo Rodriguez**

Integrantes Grupo SN20

Ada Lissette Gonzalez Castro

Karen Soraya Martínez Corbera

Fecha: **11 de febrero de 2026**



SECRETARÍA
DE INNOVACIÓN

ESIT - TSU EN SERVICIOS EN LA NUBE



Tabla de contenido

Introducción	2
1. Árbol de Conversación	2
1.1 Flujo General de Conversación	2
1.2 Flujo Bienvenida	3
1.3 Flujo Consultar Horario	3
1.5 Flujo Registrar Solicitud	4
1.6 Flujo Ayuda.....	5
1.7 Flujo Fallback.....	6
2. Arquitectura de Implementación	6
2.1 Modelo de Datos.....	7
2.2 Componentes de la arquitectura	7
2.3 Principios de Diseño de la Arquitectura.....	9
3. Puntos de Integración	9
3.1 Integración con el Canal de Comunicación	9
3.2 Integración con el Motor de Intenciones y Flujos	9
3.3 Integración con la Base de Datos / Sistema de Solicitudes	10
3.4 Importancia de los Puntos de Integración	10
Evidencias.....	10
URL de diagramas.....	10
Repositorio	11
Conclusiones	11
Anexos.....	12



Introducción

En esta fase se define el diseño lógico del funcionamiento del Chatbot Asistente del Centro de Servicios Digitales al Ciudadano (CSDC), estableciendo los flujos conversacionales que guían la interacción entre el ciudadano y el chatbot, así como la arquitectura general de integración tecnológica.

El documento describe los **árboles de conversación**, las **intenciones soportadas**, la **arquitectura de la solución (plataforma, canal y almacenamiento)** y los **puntos de integración**, con el objetivo de garantizar una experiencia clara, ordenada y eficiente para el usuario final.

1. Árbol de Conversación

El chatbot opera bajo un modelo basado en **intenciones**, donde cada mensaje del ciudadano es interpretado para dirigirlo a un flujo específico. Todos los flujos parten desde una intención inicial de bienvenida y pueden concluir con una despedida o una escalación mediante el registro de una solicitud.

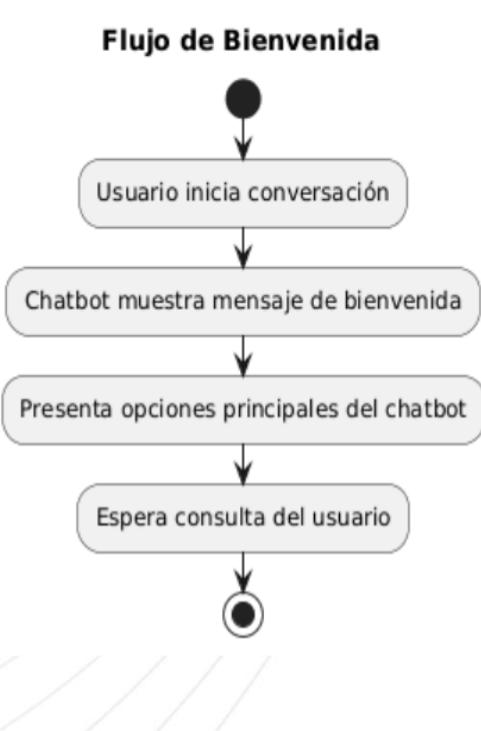
1.1 Flujo General de Conversación

Inicio → Bienvenida → Identificación de intención → Respuesta específica → (Opcional)
Registro de solicitud → Despedida





1.2 Flujo Bienvenida



Este diagrama representa el flujo general de interacción del chatbot desde el inicio de la conversación hasta el cierre, incluyendo la presentación de opciones al usuario, la identificación de la intención y la ejecución de los flujos correspondientes. Este flujo sirvió como referencia conceptual para la implementación del sistema durante la Fase 2.

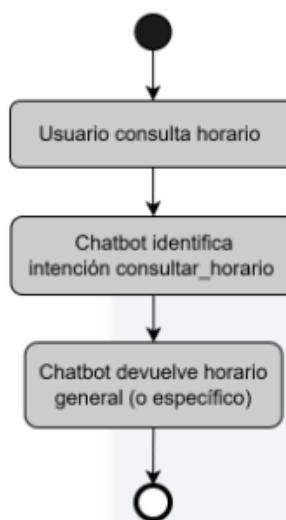
Objetivo: Iniciar la conversación y presentar las capacidades principales del chatbot.

Resultado esperado: El ciudadano comprende qué puede hacer el chatbot y continúa con su consulta. El flujo de bienvenida muestra el mensaje inicial presentado al usuario al iniciar la conversación con el chatbot. En la implementación, este mensaje se gestiona mediante la plataforma conversacional y presenta las opciones principales disponibles para el usuario.



1.3 Flujo Consultar Horario

Flujo Consultar Horario

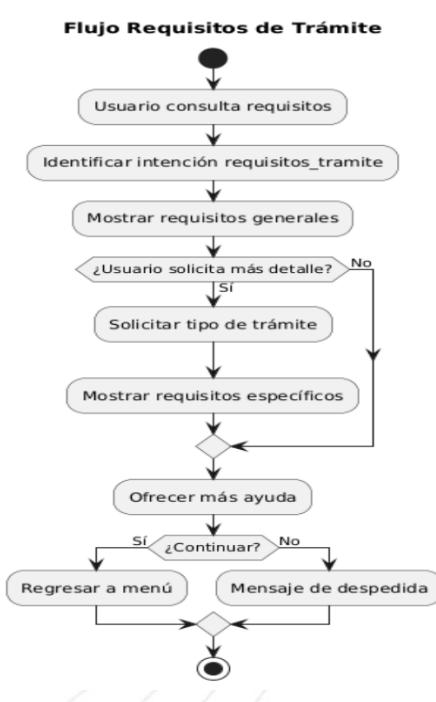


Objetivo: Brindar información sobre los horarios de atención de las clínicas comunales.

Este diagrama ilustra el flujo de interacción para la consulta de horarios de atención. En la implementación, este flujo se resolvió mediante respuestas directas, al tratarse de una consulta informativa que no requiere manejo de estado ni persistencia de datos.

Actualmente se maneja solo el horario general.

1.4 Flujo Requisitos Trámite

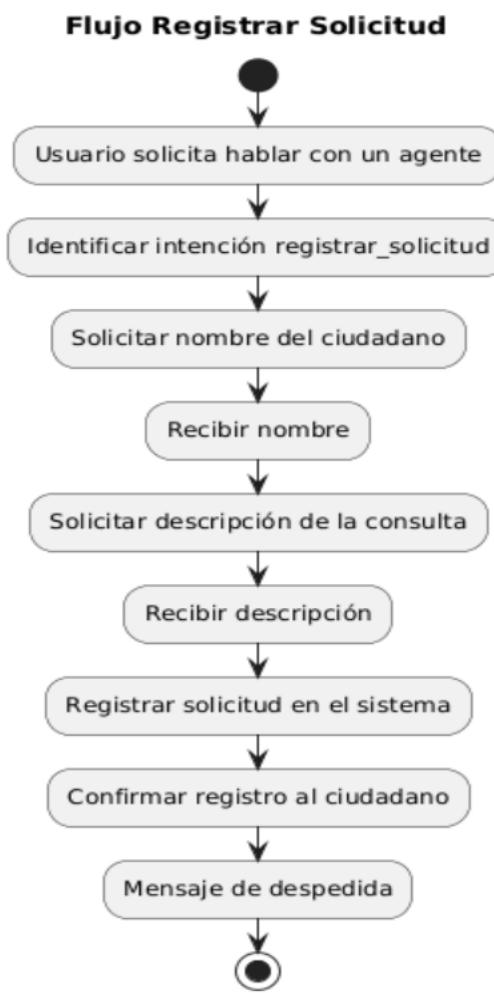


Objetivo: Informar sobre los documentos o requisitos necesarios para realizar trámites en las clínicas.

El diagrama presenta el flujo conceptual para la consulta de requisitos de trámites. Durante la implementación, este flujo fue ajustado para incorporar un menú estructurado con opciones numeradas y validación en el backend, con el fin de reducir la ambigüedad en las respuestas del usuario y mejorar la precisión en la entrega de la información.



1.5 Flujo Registrar Solicitud



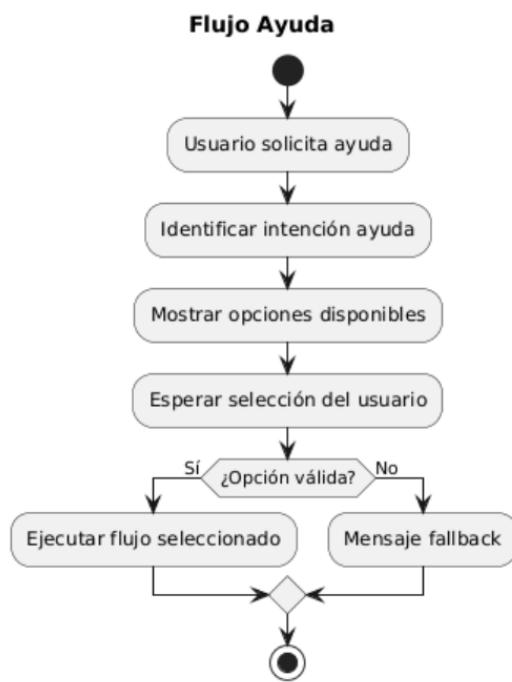
Objetivo: Registrar una solicitud ciudadana cuando la consulta no puede resolverse automáticamente.

Este diagrama describe el flujo de registro de solicitudes ciudadanas. En la implementación, el flujo fue ampliado para incluir la captura de un medio de contacto opcional y la persistencia de la información en una hoja de cálculo, permitiendo un seguimiento posterior por parte del equipo responsable.

Actualmente funciona con el código en Python, siempre y cuando en entorno esté activo y Ngrok también. Ya registra exitosamente los datos.



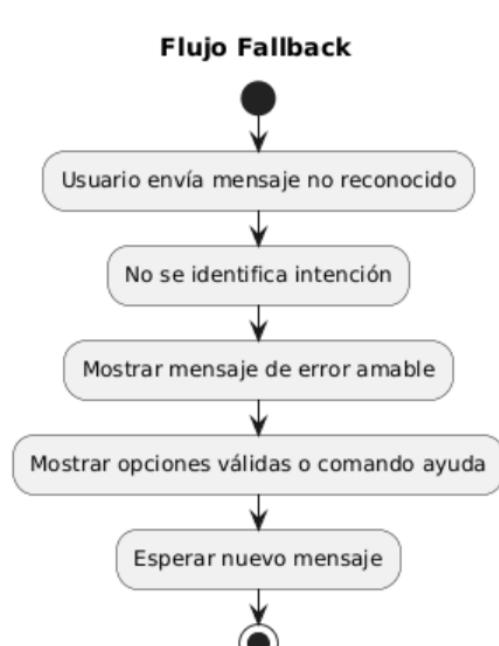
1.6 Flujo Ayuda



Objetivo: Orientar al ciudadano sobre las opciones disponibles dentro del chatbot.

El flujo de ayuda orienta al usuario sobre las funcionalidades disponibles del chatbot y las opciones de interacción. Este flujo permite mejorar la experiencia de usuario al brindar una guía rápida sobre el uso del sistema.

1.7 Flujo Fallback

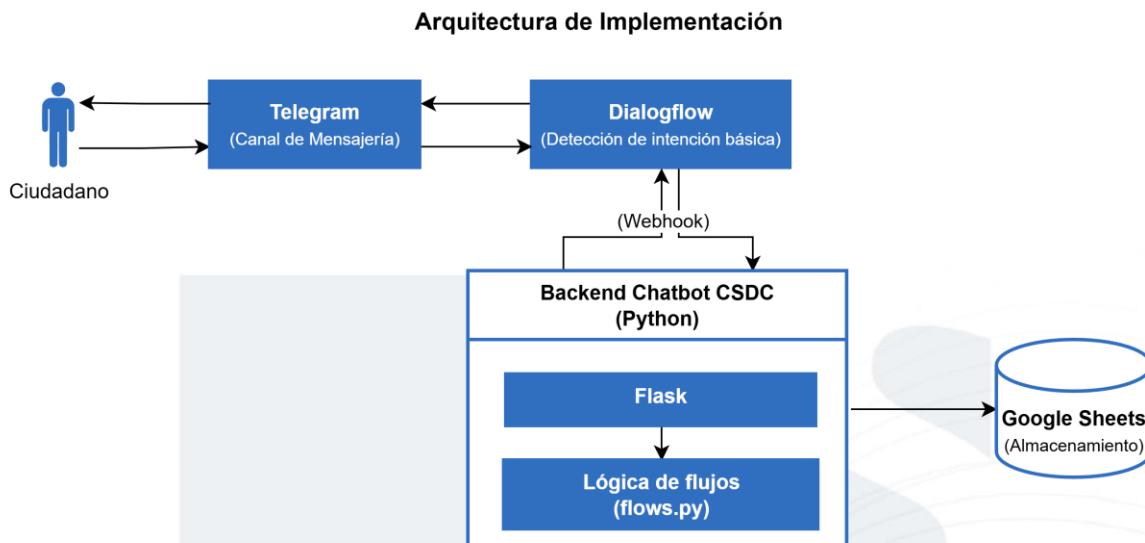


Objetivo: Gestionar mensajes que no coinciden con ninguna intención definida.

Este diagrama representa el comportamiento del chatbot cuando el mensaje del usuario no puede ser interpretado correctamente. En la implementación, el flujo de fallback se utiliza para redirigir al usuario hacia opciones válidas y evitar la ruptura de la conversación.



2. Arquitectura de Implementación



Este diagrama muestra la arquitectura técnica de la solución implementada, en la cual la plataforma conversacional se integra con un backend desarrollado en Python mediante un webhook, y este a su vez se conecta con servicios en la nube para el almacenamiento de la información.

2.1 Modelo de Datos

Campos mínimos de la solicitud registrada:

- Nombre del ciudadano
- Descripción de la solicitud
- Medio de contacto (opcional)
- Fecha y hora

Este chatbot no debe utilizarse para ingresar datos personales sensibles (DUI, dirección exacta, información médica privada, etc.)



2.2 Componentes de la arquitectura

Componente	Responsabilidad	Riesgo	Mitigación
Telegram Bot	Canal de interacción con el ciudadano. Recibe los mensajes del usuario y muestra las respuestas del chatbot.	Caídas del servicio de Telegram o problemas de conectividad del usuario.	Informar al usuario de posibles fallos del canal y documentar canales alternos de atención (línea humana del CSDC).
Dialogflow	Identificar la intención del usuario y enrutar la conversación hacia el webhook cuando se requiere lógica dinámica.	Baja precisión en la detección de intenciones o activación incorrecta del fallback.	Entrenar las intenciones con frases variadas, revisar periódicamente los intents y usar fallback con mensajes orientadores.
Webhook (Flask)	Procesar la lógica del chatbot, manejar flujos multi-turn y orquestar la integración con servicios externos.	Caídas del servidor local, errores en el código o pérdida de contexto en conversaciones multi-turn.	Manejo de errores en el código, validación de entradas y pruebas funcionales básicas por intención.
Ngrok	Exponer temporalmente el webhook local para permitir la integración con Dialogflow durante la fase de desarrollo.	La URL cambia al reiniciar Ngrok, interrumpiendo la comunicación con Dialogflow.	Documentar el procedimiento de actualización de la URL en Dialogflow y considerar despliegue en la nube en fases futuras.
Google Sheets	Almacenar las solicitudes ciudadanas registradas por el chatbot de forma estructurada.	Pérdida de acceso a la hoja, cambios accidentales en el formato o eliminación de datos.	Definir encabezados fijos, control de permisos de acceso y respaldos periódicos del archivo.
Google Cloud (APIs)	Proveer autenticación y acceso a las APIs de Google Sheets y Google Drive mediante cuentas de servicio.	Exposición accidental de credenciales o mala configuración de permisos.	No subir credenciales al repositorio, uso de variables de entorno y control de accesos mínimos necesarios.
Repositorio GitHub	Almacenar el código fuente, documentación	Exposición de información	Uso de .gitignore, revisión antes de



técnica y evidencias del sensible (tokens, subir cambios y credenciales) o buenas prácticas de desorganización del versionamiento y repositorio.

La tabla anterior permite identificar los principales puntos de riesgo de la arquitectura propuesta y las acciones básicas de mitigación aplicadas en la Fase 2, contribuyendo a una implementación más controlada y alineada a buenas prácticas de desarrollo en la nube.

2.3 Principios de Diseño de la Arquitectura

La arquitectura del chatbot se fundamenta en los siguientes principios:

- **Modularidad:** cada componente cumple una función específica.
- **Escalabilidad:** permite incorporar nuevas intenciones y canales.
- **Mantenibilidad:** facilita cambios sin afectar todo el sistema.
- **Disponibilidad:** el servicio puede mantenerse activo de forma continua.
- **Orientación al ciudadano:** prioriza claridad y facilidad de uso.

3. Puntos de Integración

3.1 Integración con el Canal de Comunicación

El primer punto de integración se establece entre la **plataforma del chatbot y el canal de comunicación**.

Objetivo de la integración: Permitir el intercambio de mensajes en tiempo real entre el ciudadano y el chatbot.

Funciones integradas:

- Recepción de mensajes del usuario
- Envío de respuestas generadas por el chatbot
- Manejo de sesiones conversacionales

Esta integración asegura una experiencia fluida y continua para el ciudadano.



3.2 Integración con el Motor de Intenciones y Flujos

La plataforma del chatbot se integra internamente con el **motor de intenciones**, el cual:

- Analiza el texto del mensaje
- Identifica la intención correspondiente
- Activa el flujo conversacional adecuado

Este punto de integración es crítico para garantizar que cada consulta sea atendida correctamente según el diseño establecido.

3.3 Integración con la Base de Datos / Sistema de Solicitudes

Cuando una consulta no puede resolverse automáticamente, el chatbot activa el flujo de **registro de solicitud**, integrándose con la base de datos.

Funciones de esta integración:

- Registrar solicitudes ciudadanas
- Almacenar información básica del caso
- Permitir el seguimiento posterior por personal del CSDC

Este mecanismo garantiza la continuidad del servicio, aun cuando la atención automatizada no sea suficiente.

3.4 Importancia de los Puntos de Integración

Los puntos de integración permiten:

- Unificar la atención ciudadana
- Reducir la carga de trabajo humano
- Aumentar la eficiencia operativa
- Garantizar trazabilidad de las solicitudes



Evidencias

URL de diagramas

1.1 Flujo General de Conversación: <https://goo.su/0BNgaP>

1.2 Flujo: Bienvenida: <https://goo.su/fThN>

1.3 Flujo: Consultar Horario:

https://viewer.diagrams.net/?tags=%7B%7D&lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1&title=Diagrama%20Consultar%20horario.drawio&dark=auto#Uhttps%3A%2F%2Fraw.githubusercontent.com%2Fcorberaks%2Fgrupo20_Chatbot%2Fmain%2Fflujos%2FDiagrama%2520Consultar%2520horario.drawio

1.4 Flujo: Requisitos de Trámite: <https://goo.su/vuaq>

1.5 Flujo: Registrar Solicitud (Escalación): <https://goo.su/OLS0za>

1.6 Flujo: Ayuda: <https://goo.su/tOBVcBj>

1.7 Flujo: Fallback: <https://goo.su/SeeBnmu>

2. Arquitectura de Implementación:

https://viewer.diagrams.net/?tags=%7B%7D&lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1&title=Arquitectura%20Implementacion.drawio&dark=auto#Uhttps%3A%2F%2Fraw.githubusercontent.com%2Fcorberaks%2Fgrupo20_Chatbot%2Fmain%2Fflujos%2FArquitectura%2520Implementacion.drawio

Repositorio

Enlace a GitHub: https://github.com/corberaks/grupo20_Chatbot



Conclusiones

1. La arquitectura de integración desarrollada permitió validar el funcionamiento básico del chatbot asistente en la nube, logrando una comunicación efectiva entre el canal de atención, la plataforma conversacional, el backend y el almacenamiento de solicitudes. Esta implementación confirma la viabilidad técnica del enfoque propuesto para la versión Lite del proyecto y deja una base estructurada para su evolución en fases posteriores, tanto a nivel funcional como de infraestructura.
2. Podemos concluir que los flujos de integración creados fueron esenciales para el desarrollo real del proyecto, ya que guiaron la implementación práctica del chatbot y fueron ajustados de acuerdo con las herramientas seleccionadas y las capacidades técnicas del equipo, permitiendo validar la viabilidad de la solución en su versión Lite.



Anexos

Código completo de flows.py (El archivo que maneja los flujos conversacionales)

```

import unicodedata
import re
from data import REQUISITOS
from data import MENSAJES_REGISTRO
from sheets import guardar_solicitud

#todo el texto en minúsculas, sin tildes ni símbolos
def normalizar_texto(texto):
    texto = texto.lower()
    texto = unicodedata.normalize("NFD", texto)
    texto = "".join(c for c in texto if unicodedata.category(c) != "Mn")
    texto = re.sub(r"^[a-z0-9\s]", "", texto)
    return texto.strip()

user_states = {}

##INTENCION 3: requisitos_tramite##
#Menu de trámites disponibles
def iniciar_requisitos(user_id):
    user_states[user_id] = {
        "flow": "requisitos",
        "step": "esperando_tramite"
    }
    return {
        "fulfillmentText": (
            "Claro ☺ ¿Qué trámite deseas consultar?\n\n"
            "[1] Vacunación\n"
            "[2] Control prenatal\n"
            "[3] Referencia\n\n"
            "Puedes responder con el número o el nombre del trámite."
        )
    }

#Entender el trámite
def resolver_tramite(user_input):
    texto = normalizar_texto(user_input)

```



```

# Si escribió un número
if texto in REQUISITOS:
    return REQUISITOS[texto]["respuesta"]

#Buscar en sinónimos
for trámite in REQUISITOS.values():
    sinónimos = trámite["sinónimos"]
    if any(alias in texto for alias in sinónimos):
        return trámite["respuesta"]

return None

#Devolver una respuesta al trámite
def manejar_trámite(user_id, user_message):
    state = user_states.get(user_id)

    if not state:
        return None

    if state.get("flow") != "requisitos":
        return None

    if state.get("step") != "esperando_trámite":
        return None
    respuesta = resolver_trámite(user_message)

    if respuesta:
        del user_states[user_id]
        return {"fulfillmentText": respuesta}

    return {
        "fulfillmentText": (
            "No logré identificar el trámite :(\n"
            "Responde con el número o el nombre:\n"
            "[1] Vacunación\n"
            "[2] Control prenatal\n"
            "[3] Referencia"
        )
    }

##INTENCIÓN 4: registrar_solicitud##

```



```

def iniciar_registro(user_id):
    user_states[user_id] = {
        "flow": "registro",
        "step": "nombre",
        "data": {}
    }

    return {
        "fulfillmentText": (
            MENSAJES_REGISTRO["inicio"]
            + "\n\n"
            + MENSAJES_REGISTRO["nombre"]
        )
    }
}

def manejar_registro(user_id, user_message):
    state = user_states.get(user_id)

    if not state or state.get("flow") != "registro":
        return None

    step = state["step"]
    data = state["data"]

    # PASO 1 – Nombre
    if step == "nombre":
        data["nombre"] = user_message
        state["step"] = "descripcion"

    return {"fulfillmentText": MENSAJES_REGISTRO["descripcion"]}

    # PASO 2 – Descripción
    if step == "descripcion":
        data["descripcion"] = user_message
        state["step"] = "contacto"

    return {"fulfillmentText": MENSAJES_REGISTRO["contacto"]}

    # PASO 3 – Contacto opcional
    if step == "contacto":
        if user_message.lower() in ["no", "ninguno", "prefiero no"]:

```



```
        data["contacto"] = "No proporcionado"
    else:
        data["contacto"] = user_message

    state["step"] = "confirmacion"

    return {
        "fulfillmentText": MENSAJES_REGISTRO["confirmacion"].format(
            nombre=data["nombre"],
            descripcion=data["descripcion"],
            contacto=data["contacto"]
        )
    }

# PASO 4 - Confirmación
if step == "confirmacion":
    if user_message.lower() in ["si", "sí"]:
        guardar_solicitud(
            nombre=data["nombre"],
            descripcion=data["descripcion"],
            contacto=data["contacto"]
        )

        del user_states[user_id]

    return {"fulfillmentText": MENSAJES_REGISTRO["exito"]}

    if user_message.lower() == "no":
        del user_states[user_id]
        return {"fulfillmentText": MENSAJES_REGISTRO["cancelado"]}

return {"fulfillmentText": "Por favor responde *sí* o *no*. "}
```

