



SECRETARÍA
DE INNOVACIÓN

Escuela Superior de Innovación y Tecnología



ESCUELA SUPERIOR
de Innovación
y Tecnología

Técnico Superior Universitario Servicios en la Nube

Estancia Profesional

Entregable Fase 2

Arquitectura de Integración y Bitácora Técnica

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

Resumen ejecutivo.....	2
1. Infraestructura creada	3
1.1 Infraestructura en la nube	3
1.2 Almacenamiento	3
1.3 Entorno Local	4
2. Códigos y Scripts funcionales	5
2.1 Estructura del Proyecto en Visual Studio Code	5
2.2 Códigos funcionales	6
3. Configuraciones de Servicio	7
3.1 Configuración de la plataforma conversacional: Dialogflow	7
3.2 Configuración del backend en Python (Flask).....	8
3.3 Exposición del webhook mediante Ngrok	9
3.4 Configuración de servicios en la nube	9
3.5 Consideraciones de seguridad y buenas prácticas	10
4. Bitácora Técnica	10
4.1 Comandos utilizados	10
4.2 Procedimiento seguido	11
4.3 Funcionamiento Básico	11
4.4 Justificación de cambios realizados	12
4.5 Observaciones técnicas.....	13
Evidencias.....	14
1. Pruebas en POSTMAN	14
2. Pruebas en Telegram.....	18
Conclusiones	19
Anexos.....	20

Resumen ejecutivo

El objetivo de la Fase 2 fue implementar el chatbot diseñado en la fase anterior, integrando la lógica conversacional, la infraestructura necesaria y los servicios de almacenamiento, garantizando un funcionamiento básico y comprobable del sistema.

El chatbot fue implementado en Python utilizando el framework Flask para exponer un webhook que recibe las solicitudes desde la plataforma conversacional.

La lógica de los flujos conversacionales (requisitos de trámites y registro de solicitudes) se implementó mediante un manejador de estados que permite mantener el contexto de la conversación en interacciones de múltiples turnos.

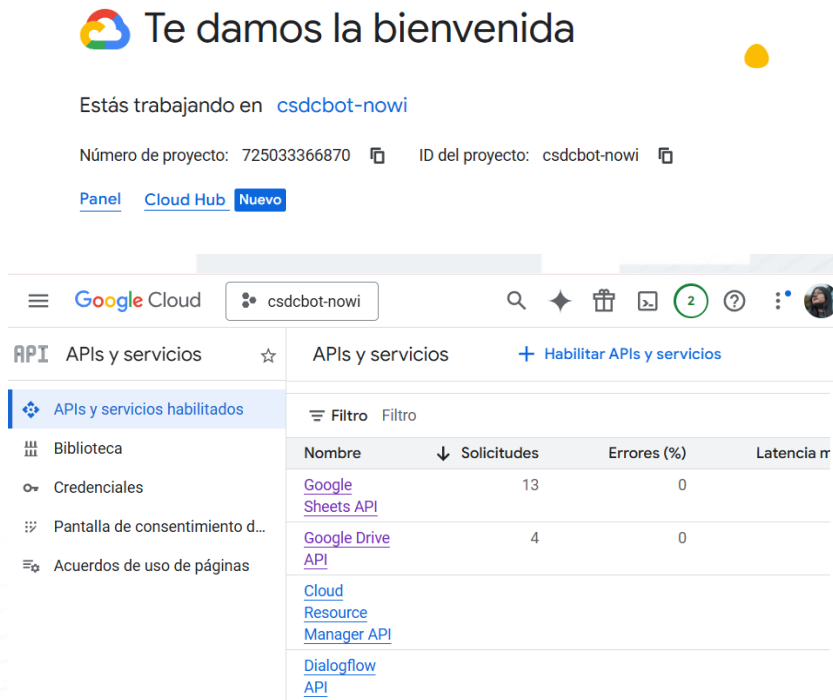
Asimismo, se desarrolló un módulo de integración con Google Sheets para almacenar las solicitudes ciudadanas registradas por el chatbot, utilizando credenciales de cuenta de servicio de Google Cloud.

La estructura modular del proyecto permite separar responsabilidades entre orquestación, lógica de negocio y persistencia de datos, facilitando futuras extensiones del sistema.

1. Infraestructura creada

1.1 Infraestructura en la nube

Infraestructura en Google Cloud utilizada para la integración del chatbot con Google Sheets mediante una cuenta de servicio.



Google Sheets API para guardar los datos de una solicitud.

Google Drive API para que se pueda acceder al documento en la nube.

1.2 Almacenamiento

Hoja de cálculo utilizada como almacenamiento de solicitudes registradas automáticamente por el chatbot.

Solicitudes Chatbot CSDC - AmiBot

Archivo Editar Ver Insertar Formato Datos Herramientas Gemini Extensiones Ayuda

Menús 100% Predet... 10 B I A

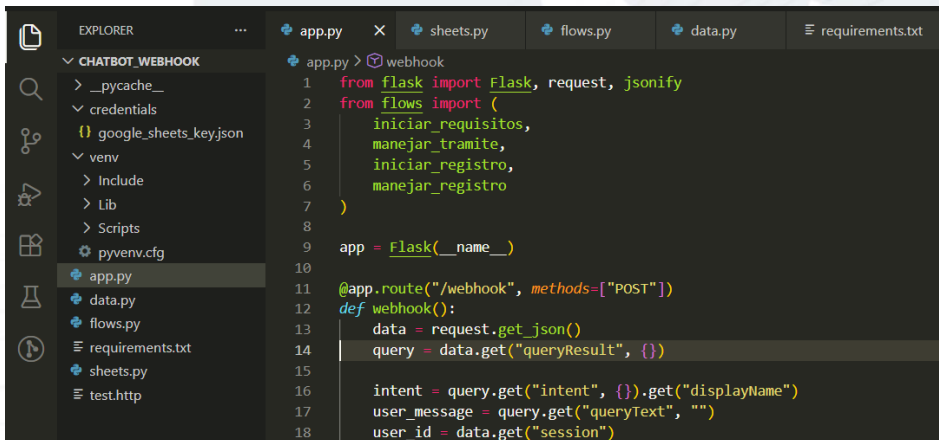
	A	B	C	D
1	Fecha	Nombre	Descripción	Contacto
2	2026-02-09 20:25:44	ana martínez	no encontré información clara sobre mi trámite	ana@email.com
3	2026-02-10 17:46:07	Karen	No encontré la información que necesitaba	698526544
4	2026-02-10 17:56:03	Daniel Carvajal	No encontré lo que buscaba	294858191
5	2026-02-10 19:18:24	Sarah Castro	No encuentro información sobre vacunación	sara@email.com
6				

Los datos que aparecen en la hoja fueron capturados a través de Postman, Dialogflow y Telegram durante las pruebas.

1.3 Entorno Local

Entorno local de desarrollo utilizado para ejecutar el backend del chatbot.

Entorno en Visual Studio Code

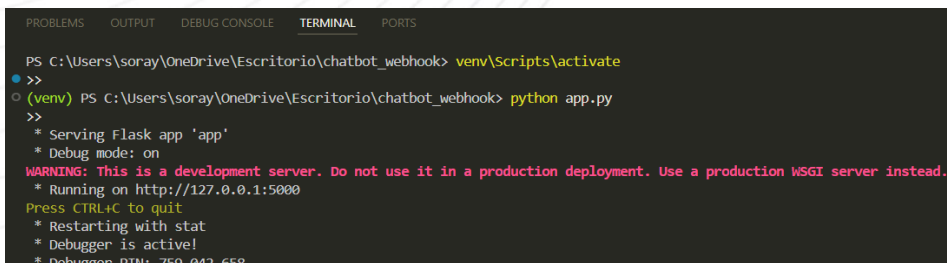


```

EXPLORER
  CHATBOT_WEBHOOK
    _pycache_
    credentials
    google_sheets_key.json
    venv
      Include
      Lib
      Scripts
    pyvenv.cfg
    app.py
    data.py
    flows.py
    requirements.txt
    sheets.py
    test.http

app.py
1 from flask import Flask, request, jsonify
2 from flows import (
3     iniciar_requisitos,
4     manejar_tramite,
5     iniciar_registro,
6     manejar_registro
7 )
8
9 app = Flask(__name__)
10
11 @app.route("/webhook", methods=["POST"])
12 def webhook():
13     data = request.get_json()
14     query = data.get("queryResult", {})
15
16     intent = query.get("intent", {}).get("displayName")
17     user_message = query.get("queryText", "")
18     user_id = data.get("session")
  
```

Terminal de app.py



```

PS C:\Users\soray\OneDrive\Escritorio\chatbot_webhook> venv\Scripts\activate
>>
(venv) PS C:\Users\soray\OneDrive\Escritorio\chatbot_webhook> python app.py
>>
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 759-042-658
  
```

Terminal con Ngrok activo (se utilizó para integrar el código con Dialogflow).



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

ngrok

Session Status      online
Account             sorayacorbera.k@gmail.com (Plan: Free)
Update             update available (version 3.36.0, Ctrl-U to update)
Version             3.36.0-msix-stable
Region             United States (us)
Latency             90ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://noncensurable-onita-wageless.ngrok-free.dev -> http://localhost:5000

Connections  ttl  opn  rt1  rt5  p50  p90
```

2. Códigos y Scripts funcionales

2.1 Estructura del Proyecto en Visual Studio Code

Lenguaje de programación: **Python**.

Carpeta principal: **chatbot_webhook**.

Archivos dentro de la carpeta principal:

app.py: Este archivo contiene la aplicación principal del backend del chatbot, implementada utilizando el framework Flask. Su función es exponer un endpoint web (/webhook) que recibe las solicitudes enviadas desde la plataforma conversacional (Telegram, Dialogflow o pruebas realizadas mediante Postman).

flows.py: permite gestionar interacciones de múltiples turnos, como la consulta de requisitos de trámites y el registro de solicitudes ciudadanas, mediante el uso de un diccionario de estados por usuario que mantiene el contexto de la conversación sin perderse.

data.py: Este archivo contiene los datos estáticos utilizados por el chatbot, como catálogos de trámites, sinónimos, y los mensajes predefinidos que el bot utiliza durante la conversación.

sheets.py: Configura la autenticación mediante una cuenta de servicio de Google Cloud y se definen las funciones encargadas de establecer la conexión con la hoja de cálculo, así como de insertar nuevas filas con los datos proporcionados por el usuario.

credentials/service_account.json: Este archivo contiene las credenciales de una cuenta de servicio de Google Cloud utilizadas para autenticar el acceso del chatbot a la API de Google Sheets.

requirements.txt: Este archivo especifica las librerías y dependencias necesarias para la correcta ejecución del chatbot, tales como Flask para la creación del webhook, así como las bibliotecas requeridas para la autenticación e integración con Google Sheets.

2.2 Códigos funcionales

Webhook (app.py)

El webhook implementado con Flask recibe las solicitudes desde la plataforma del chatbot y enruta la conversación según la intención detectada.

```
11 @app.route("/webhook", methods=["POST"])
12 def webhook():
13     data = request.get_json()
14     query = data.get("queryResult", {})
15
16     intent = query.get("intent", {}).get("displayName")
17     user_message = query.get("queryText", "")
18     user_id = data.get("session")
```

Manejo de flujos (flows.py)

La lógica conversacional se implementó mediante un manejador de estados que permite controlar flujos multi-turno como el registro de solicitudes.

```
81 def iniciar_registro(user_id):
82     user_states[user_id] = {
83         "flow": "registro",
84         "step": "nombre",
85         "data": {}
86     }
87
88     return {
89         "fulfillmentText": (
90             MENSAJES_REGISTRO["inicio"]
91             + "\n\n"
92             + MENSAJES_REGISTRO["nombre"]
93         )
94     }
```

Integración con Google Sheets (sheets.py)

Se implementó una función para persistir los datos capturados por el chatbot en una hoja de cálculo de Google Sheets utilizando credenciales de cuenta de servicio.

```
def guardar_solicitud(nombre, descripcion, contacto):  
    sheet = get_sheet()  
  
    fecha = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  
  
    sheet.append_row([  
        fecha,  
        nombre,  
        descripcion,  
        contacto  
    ])
```

3. Configuraciones de Servicio

3.1 Configuración de la plataforma conversacional: Dialogflow

Se configuró un agente en Dialogflow como interfaz conversacional principal (hasta ahora) del chatbot. En este agente se definieron las intenciones (intents) básicas que permiten al usuario interactuar con el sistema, como la consulta de horarios de atención, el inicio del flujo de registro de solicitudes y el inicio del flujo de la consulta de trámites.

La intención de bienvenida fue configurada mediante el uso del evento WELCOME, de modo que el mensaje inicial se muestre únicamente al iniciar la conversación con el chatbot, presentando al usuario las opciones disponibles. Asimismo, se configuró un intent de fallback para manejar entradas no reconocidas, redirigiendo al usuario hacia las opciones válidas del sistema.

Se habilitó el uso de Webhook para las intenciones que requieren lógica dinámica o flujos de múltiples pasos permitiendo que Dialogflow delegue el procesamiento de la conversación al backend desarrollado en Python.

• ayuda
• bienvenida
• consultar_horario
🔖 Default Fallback Intent
• despedida
• registrar_solicitud_inicio
• requisitos_tramite_inicio

• bienvenida

SAVE

Contexts ?

Events ?

Welcome

Add event

Fulfillment ?

☒ Enable webhook call for this intent

☐ Enable webhook call for slot filling

3.2 Configuración del backend en Python (Flask)

El backend del chatbot fue implementado utilizando el framework Flask, el cual expone un endpoint web que actúa como webhook para recibir las solicitudes provenientes de la plataforma conversacional.

Se configuró el entorno de ejecución mediante un entorno virtual de Python donde se instalaron las dependencias necesarias para el funcionamiento del servidor web y la integración con servicios externos.

```
PS C:\Users\soray\OneDrive\Escritorio\chatbot_webhook> venv\Scripts\activate
>>
(venv) PS C:\Users\soray\OneDrive\Escritorio\chatbot_webhook> python app.py
>>
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

3.3 Exposición del webhook mediante Ngrok

Para permitir que Dialogflow se comunice con el backend en Python durante la fase de desarrollo, se utilizó la herramienta Ngrok para exponer de forma temporal el servidor local a través de una URL pública accesible desde Internet.

Mediante esta configuración el endpoint del webhook local fue enlazado con la configuración de fulfillment en Dialogflow, permitiendo la comunicación bidireccional entre la plataforma conversacional y el backend. Esta solución resulta adecuada para entornos de desarrollo y pruebas, ya que evita la necesidad de desplegar el backend en un servidor remoto durante la fase de implementación inicial.

⚡ Fulfillment

Webhook

ENABLED ☒

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.

URL*

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

ngrok

Session Status      online
Account             sorayacorbera.k@gmail.com (Plan: Free)
Update              update available (version 3.36.0, Ctrl-U to update)
Version             3.36.0-msix-stable
Region              United States (us)
Latency             85ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://noncensurable-onita-wageless.ngrok-free.dev -> http://localhost:5000
```

3.4 Configuración de servicios en la nube

Se configuró un proyecto en Google Cloud Platform con el objetivo de habilitar el acceso programático a Google Sheets, que se utiliza como mecanismo de almacenamiento ligero para las solicitudes registradas por el chatbot.

Dentro del proyecto se habilitaron las APIs necesarias (Google Sheets API y Google Drive API) y se creó una cuenta de servicio para la autenticación segura del backend. Las credenciales generadas fueron almacenadas de forma local y utilizadas por el módulo de integración del chatbot para realizar operaciones de escritura en la hoja de cálculo.

Asimismo, se configuró una hoja de cálculo específica con encabezados definidos para estructurar la información registrada (nombre del ciudadano, descripción de la solicitud y medio de contacto opcional), permitiendo su posterior consulta y seguimiento por parte del personal correspondiente.

3.5 Consideraciones de seguridad y buenas prácticas

Durante la configuración de los servicios se tomaron en cuenta consideraciones básicas de seguridad, tales como el uso de cuentas de servicio para la autenticación en Google Cloud y la exclusión de credenciales sensibles en repositorios públicos.

Se optó por una arquitectura modular que permite desacoplar la lógica del chatbot de los servicios externos, facilitando futuras migraciones o sustituciones de componentes (por ejemplo, el uso de otro mecanismo de almacenamiento en fases posteriores del proyecto).

4. Bitácora Técnica

4.1 Comandos utilizados

Durante la implementación del chatbot se utilizaron los siguientes comandos para la configuración del entorno, instalación de dependencias y ejecución del sistema:

- `python -m venv venv`
- `venv\Scripts\activate`
- `pip install flask gspread google-auth`
- `python app.py`
- `ngrok http 5000`

Estos comandos permitieron crear un entorno virtual aislado, instalar las librerías necesarias para el funcionamiento del backend y exponer temporalmente el servicio local mediante un túnel seguro para su integración con la plataforma conversacional.

4.2 Procedimiento seguido

El desarrollo e implementación del chatbot se realizó siguiendo el siguiente procedimiento general:

1. Se creó un entorno virtual en Python para aislar las dependencias del proyecto.
2. Se implementó un servidor web utilizando Flask para exponer un webhook que recibe las solicitudes desde la plataforma conversacional.
3. Se desarrolló la lógica conversacional del chatbot, definiendo flujos para la consulta de requisitos de trámites y el registro de solicitudes ciudadanas.
4. Se estructuró el proyecto de manera modular, separando la orquestación del webhook, la lógica de negocio y los datos del chatbot.
5. Se configuró un proyecto en Google Cloud Platform, habilitando las APIs necesarias para la integración con Google Sheets.
6. Se creó una cuenta de servicio y se generaron credenciales para permitir la escritura segura de datos en la hoja de cálculo.
7. Se integró el backend del chatbot con Google Sheets para el almacenamiento de solicitudes.
8. Se configuró la plataforma conversacional para delegar al webhook el manejo de los flujos que requieren lógica dinámica.
9. Se realizaron pruebas de funcionamiento utilizando la herramienta Postman, verificando el correcto comportamiento del chatbot y la persistencia de datos.

4.3 Funcionamiento Básico

Para validar el correcto funcionamiento del sistema, se realizaron pruebas de extremo a extremo que permitieron comprobar:

- La activación correcta de los flujos conversacionales desde la plataforma conversacional.
- El manejo de conversaciones de múltiples turnos para el registro de solicitudes.
- El almacenamiento exitoso de la información capturada en una hoja de cálculo de Google Sheets.

Estas son las intenciones funcionales: 2 de ellas están pendientes de estabilizar en la integración con Dialogflow (requisitos_tramite y registrar_solicitud):

Intención	Plataforma que gestiona	Descripción
bienvenida	Dialogflow	El chatbot es capaz de saludar y presentar las opciones disponibles.
consultar_horario	Dialogflow	Cuando Dialogflow identifica la intención del usuario, responde con la información de los horarios correctamente.
requisitos_tramite	Python (Webhook)	El flujo lógico funciona en Python y en la integración a Dialogflow siempre y cuando el entorno esté activo.
registrar_solicitud	Python (Webhook)	Sigue un flujo conversacional y guarda la información en Google Sheets (siempre que el entorno esté activo).
ayuda	Dialogflow	Muestra al ciudadano las opciones disponibles y orienta sobre el uso del chatbot exitosamente.
fallback	Python (Webhook)	Efectivamente gestiona mensajes que el chatbot no logra interpretar o que no son parte del catálogo.
despedida	Dialogflow	Finaliza la conversación de forma cordial.

(Las evidencias visuales se encuentran en el apartado “Evidencias”).

4.4 Justificación de cambios realizados

Durante la fase de implementación se realizaron ajustes respecto al diseño conceptual inicial con el objetivo de mejorar la experiencia de usuario y la robustez técnica del sistema. Entre los principales cambios se encuentran:

- **Gestión de flujos desde el backend:** Se decidió manejar los flujos conversacionales directamente desde el backend en Python, en lugar de depender exclusivamente de la lógica de la plataforma conversacional, lo que permitió un mayor control del estado de la conversación y facilitó la implementación de interacciones de múltiples turnos.
- **Uso de menús estructurados:** Para la consulta de requisitos de trámites se incorporó un menú con opciones numeradas, reduciendo la ambigüedad en la interpretación de las respuestas del usuario y mejorando la usabilidad del chatbot.
- **Contacto opcional del usuario:** Se definió el medio de contacto como un campo opcional durante el registro de solicitudes, con el fin de respetar la privacidad del usuario y permitir flexibilidad en los mecanismos de seguimiento.
- **Separación del código en módulos (app, flows y data):** Durante la implementación se decidió separar el código general del chatbot en distintos módulos, lo que permitió mejorar la legibilidad del proyecto, facilitar el mantenimiento del código y hacer más sencilla la incorporación de nuevos flujos o funcionalidades en el futuro, siguiendo buenas prácticas de diseño de software y principios de separación de responsabilidades.

4.5 Observaciones técnicas

Durante la integración con la plataforma conversacional se identificaron conflictos en la detección de intenciones, particularmente con la intención de bienvenida, la cual interfería con los flujos activos del chatbot. Para resolver este problema, se ajustó la configuración de la intención de bienvenida utilizando un evento específico para el inicio de la conversación, evitando que se activara de forma incorrecta durante flujos en curso.

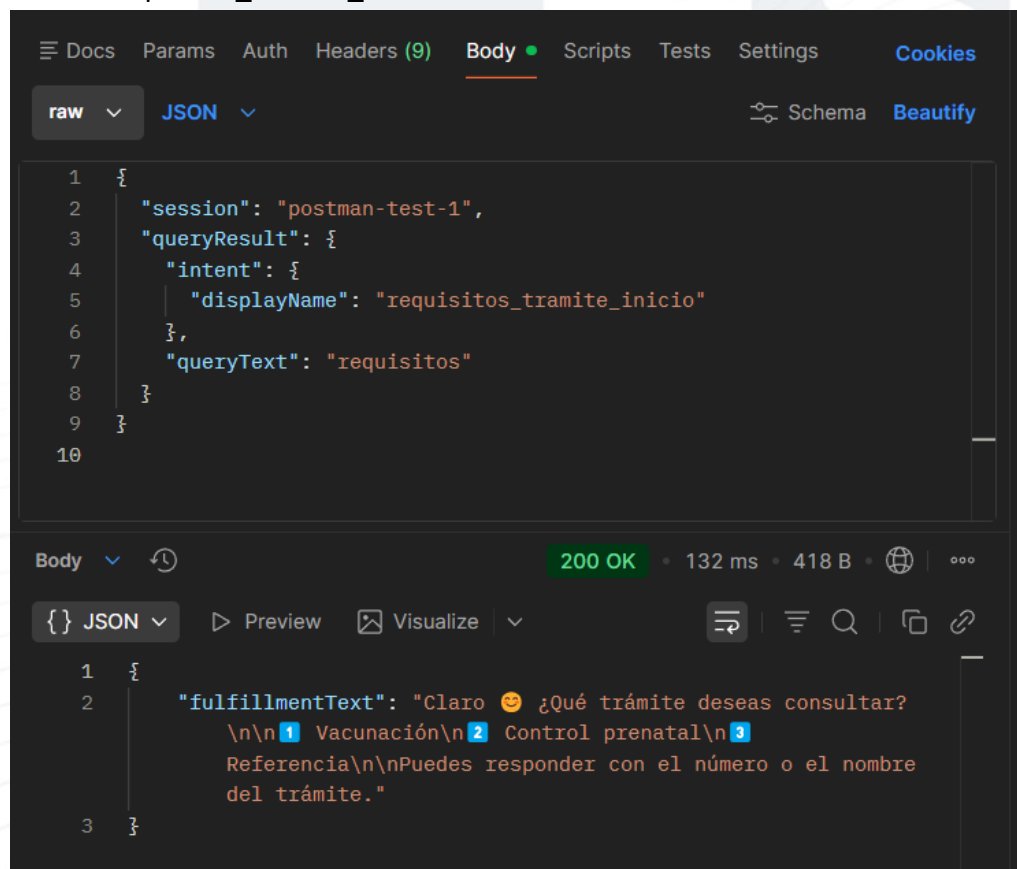
Asimismo, se configuró el uso selectivo del webhook únicamente en aquellas intenciones que requerían lógica dinámica, manteniendo respuestas estáticas en la plataforma conversacional para casos simples. Este ajuste permitió lograr una integración más estable entre la plataforma conversacional y el backend en Python.

Evidencias

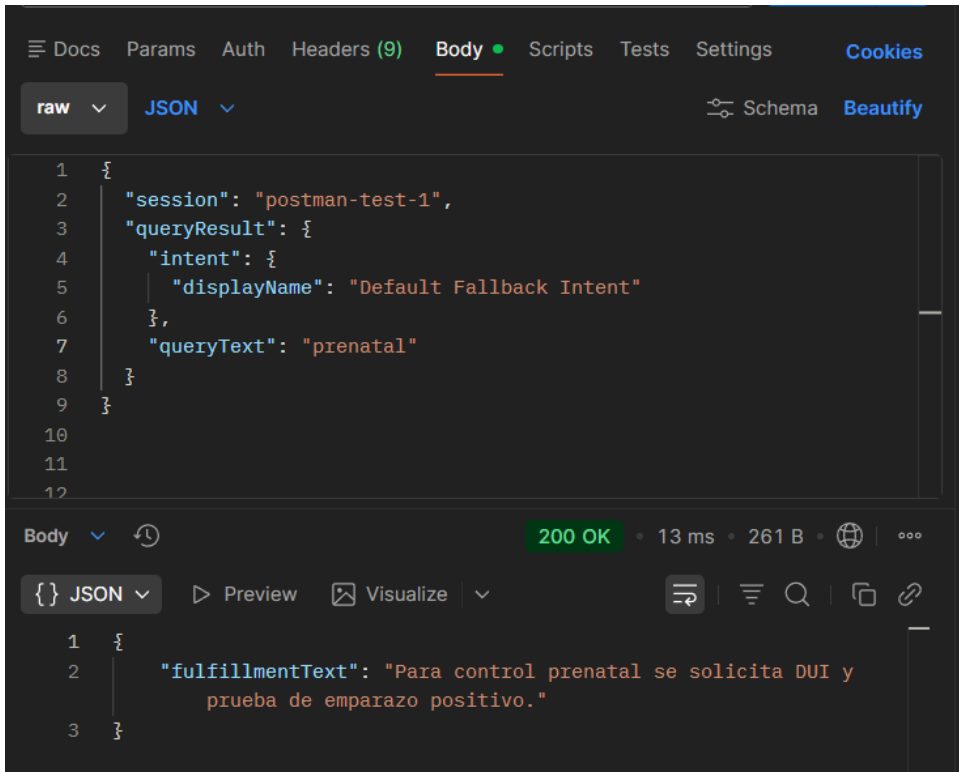
1. Pruebas en POSTMAN

Prueba de “requisitos_tramite”

Activar requisitos_tramite_inicio



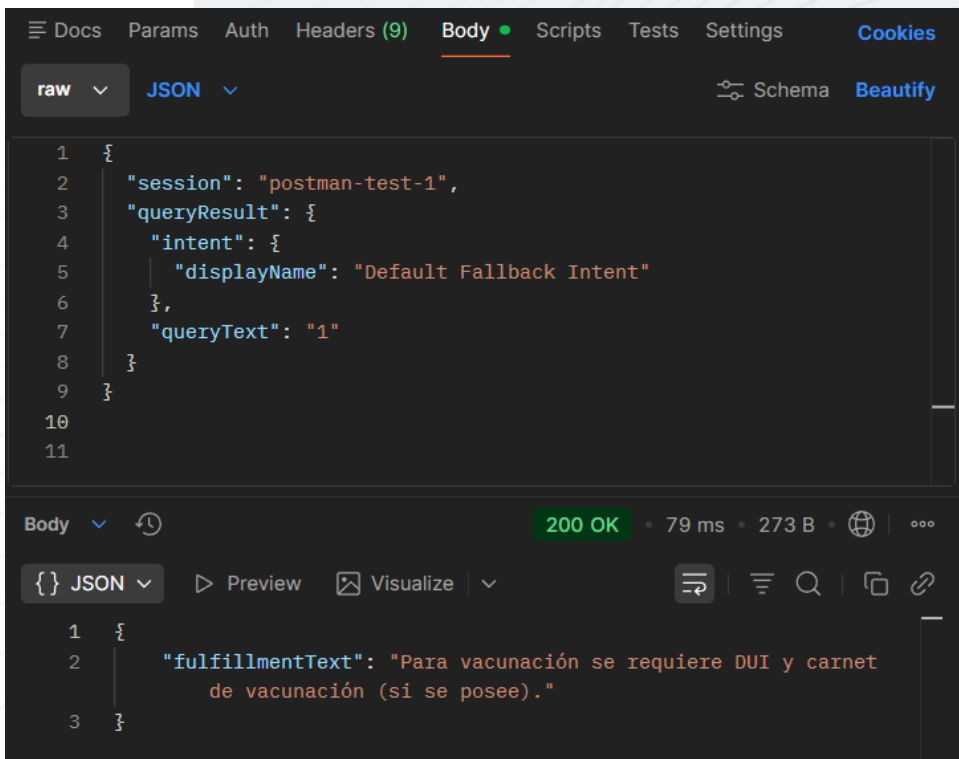
Indicar trámite y recibir respuesta específica:



```
1 {
2   "session": "postman-test-1",
3   "queryResult": {
4     "intent": {
5       "displayName": "Default Fallback Intent"
6     },
7     "queryText": "prenatal"
8   }
9 }
10
11
12
```

Body 200 OK • 13 ms • 261 B

```
1 {
2   "fulfillmentText": "Para control prenatal se solicita DUI y
3     prueba de emparazo positivo."
4 }
```



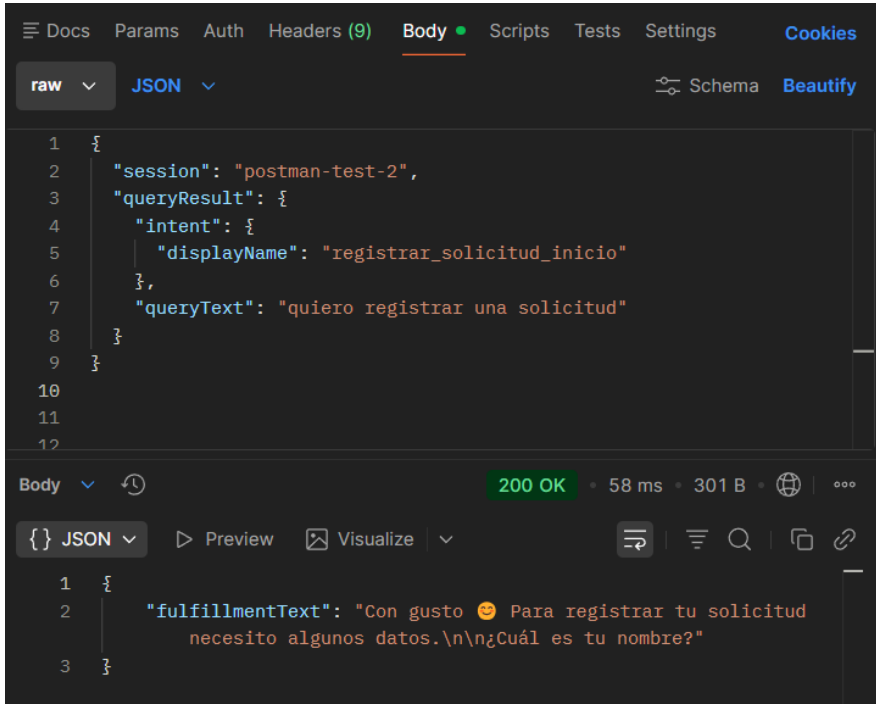
```
1 {
2   "session": "postman-test-1",
3   "queryResult": {
4     "intent": {
5       "displayName": "Default Fallback Intent"
6     },
7     "queryText": "1"
8   }
9 }
10
11
```

Body 200 OK • 79 ms • 273 B

```
1 {
2   "fulfillmentText": "Para vacunación se requiere DUI y carnet
3     de vacunación (si se posee)."
4 }
```

Prueba de registrar_solicitud

Activar la intención: registrar_solicitud_inicio



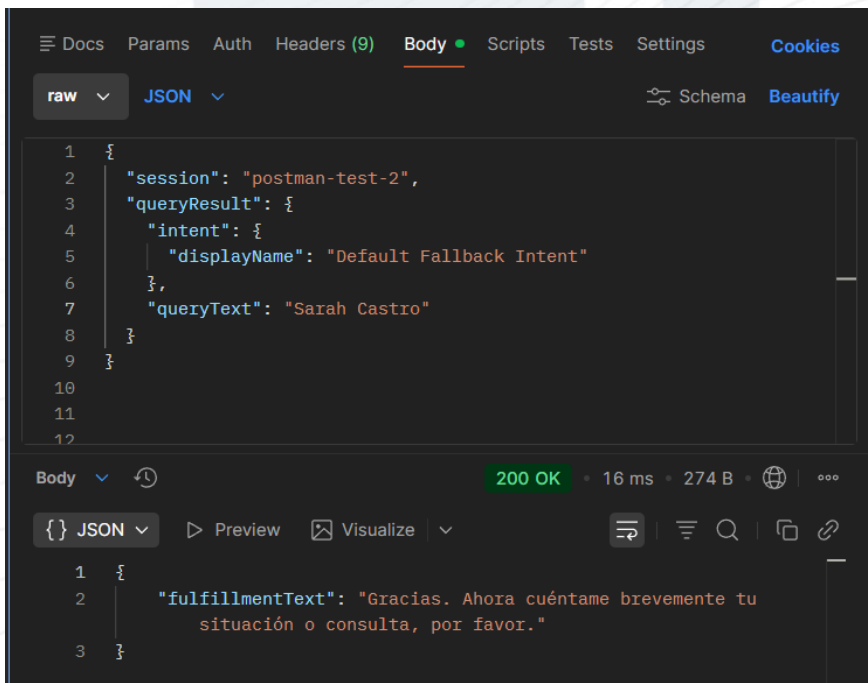
The image shows a Postman REST client interface. The top tab is 'Body', and the request body is a JSON object. The response is a 200 OK status with a JSON body containing a fulfillment text.

```
1 {
2   "session": "postman-test-2",
3   "queryResult": {
4     "intent": {
5       "displayName": "registrar_solicitud_inicio"
6     },
7     "queryText": "quiero registrar una solicitud"
8   }
9 }
10
11
12
```

Body 200 OK • 58 ms • 301 B

```
1 {
2   "fulfillmentText": "Con gusto 😊 Para registrar tu solicitud
3     necesito algunos datos.\n¿Cuál es tu nombre?"
4 }
```

Capturar nombre



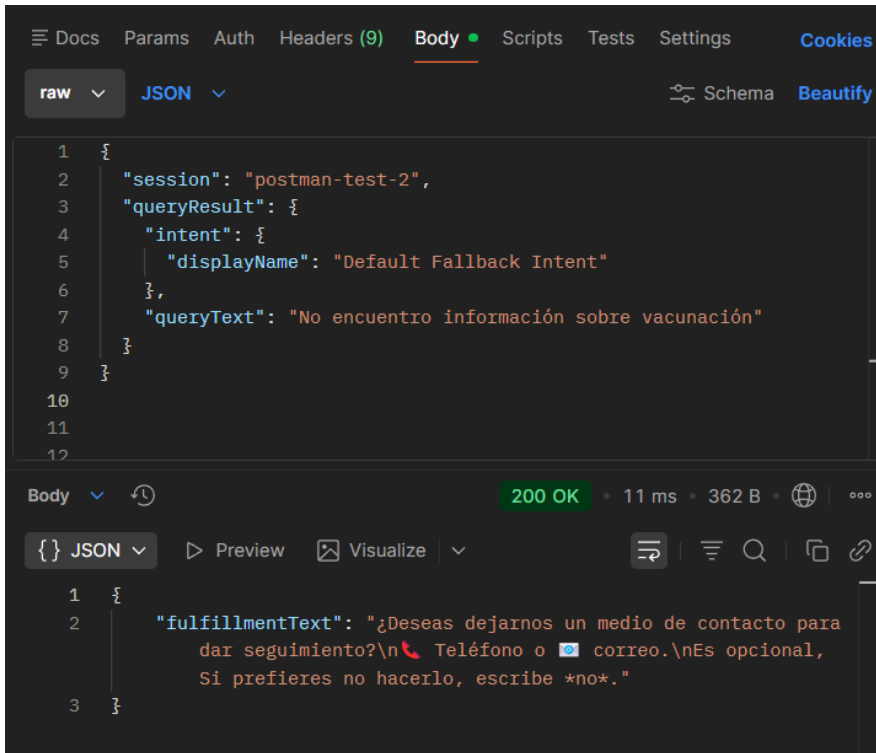
The image shows a Postman REST client interface. The top tab is 'Body', and the request body is a JSON object. The response is a 200 OK status with a JSON body containing a fulfillment text.

```
1 {
2   "session": "postman-test-2",
3   "queryResult": {
4     "intent": {
5       "displayName": "Default Fallback Intent"
6     },
7     "queryText": "Sarah Castro"
8   }
9 }
10
11
12
```

Body 200 OK • 16 ms • 274 B

```
1 {
2   "fulfillmentText": "Gracias. Ahora cuéntame brevemente tu
3     situación o consulta, por favor."
4 }
```

Captura descripción



```

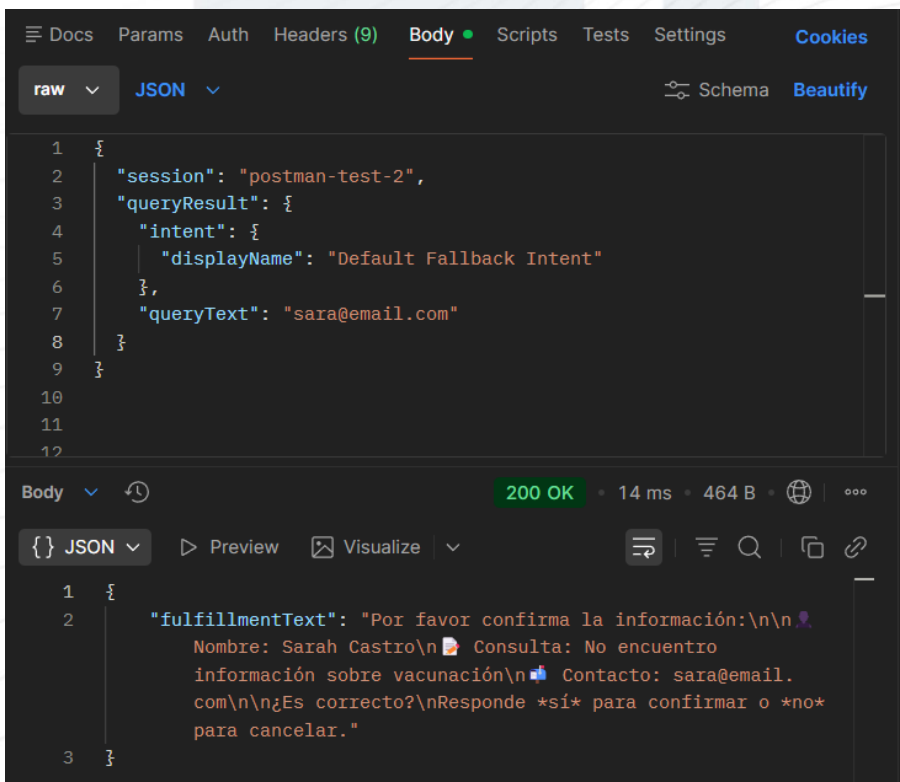
1 {
2   "session": "postman-test-2",
3   "queryResult": {
4     "intent": {
5       "displayName": "Default Fallback Intent"
6     },
7     "queryText": "No encuentro información sobre vacunación"
8   }
9 }
10
11
12
  
```

Body 200 OK • 11 ms • 362 B

```

1 {
2   "fulfillmentText": "¿Deseas dejarnos un medio de contacto para
3     dar seguimiento?\n☎ Teléfono o ✉ correo.\nEs opcional,
4     Si prefieres no hacerlo, escribe *no*."
5 }
  
```

Confirma datos guardados



```

1 {
2   "session": "postman-test-2",
3   "queryResult": {
4     "intent": {
5       "displayName": "Default Fallback Intent"
6     },
7     "queryText": "sara@email.com"
8   }
9 }
10
11
12
  
```

Body 200 OK • 14 ms • 464 B

```

1 {
2   "fulfillmentText": "Por favor confirma la información:\n\n👤
3     Nombre: Sarah Castro\n📄 Consulta: No encuentro
4     información sobre vacunación\n📧 Contacto: sara@email.
5     com\n¿Es correcto?\nResponde *sí* para confirmar o *no*
6     para cancelar."
7 }
  
```

Despues de la confirmación envía mensaje:

```

1  {
2    "session": "postman-test-2",
3    "queryResult": {
4      "intent": {
5        "displayName": "Default Fallback Intent"
6      },
7      "queryText": "sí"
8    }
9  }
10
11
12

```

Body 200 OK • 5.44 s • 277 B • Visualize Schema Beautify

```

1  {
2    "fulfillmentText": "✅ Tu solicitud fue registrada correctamente. Gracias por comunicarte con CSDC."
3  }

```

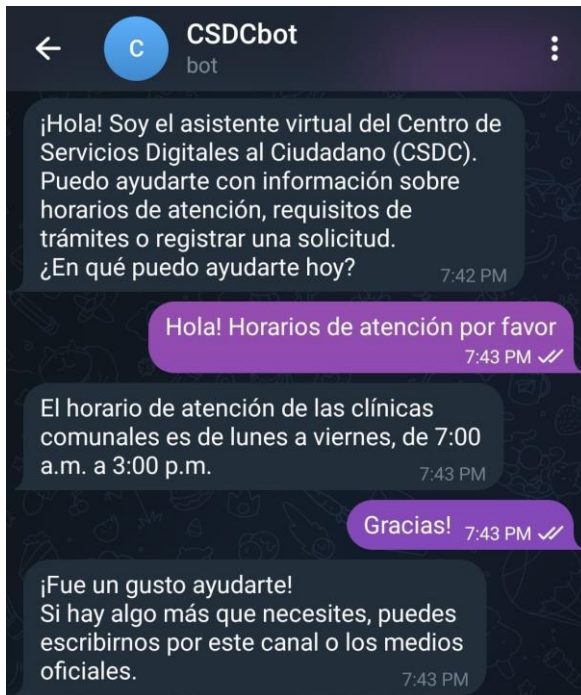
La solicitud queda guardada en Google Sheets

	A	B	C	D	E
1	Fecha	Nombre	Descripción	Contacto	
2	2026-02-09 20:25:44	ana martínez	no encontré información clara sobre mi trámite	ana@email.com	
3	2026-02-10 17:46:07	Karen	No encontré la información que necesitaba	698526544	
4	2026-02-10 17:56:03	Daniel Carvajal	No encontré lo que buscaba	294858191	
5	2026-02-10 19:18:24	Sarah Castro	No encuentro información sobre vacunación	sara@email.com	
6					

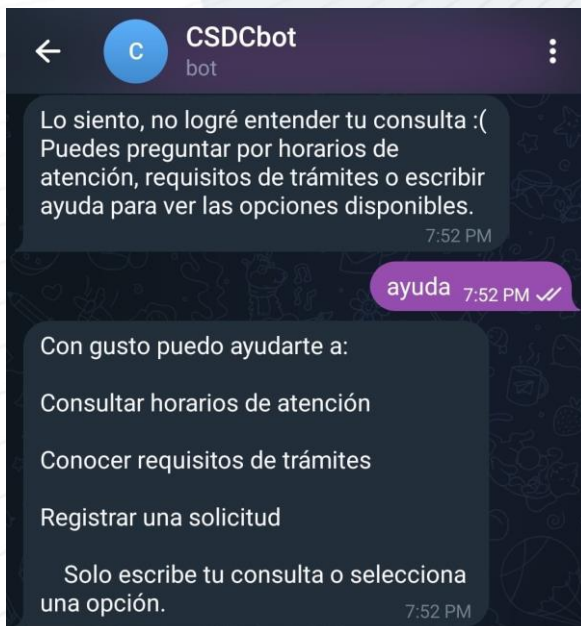
2. Pruebas en Telegram

Bot en Telegram: **@CSDC_AmiBot**

Intenciones: bienvenida, consultar horario y despedida



Intención: ayuda



Intención: registrar_solicitud

CSDCbot
bot

¡Hola! Soy el asistente virtual de CSDC. **Yesterday**
Puedo brindarte información sobre horarios de atención,
requisitos de trámites y ayudarte a registrar una solicitud.
Escribe tu consulta o selecciona una opción para continuar.
05:13 PM

Solicitud 05:55 PM ✓

Con gusto 😊 Para registrar tu solicitud necesito algunos
datos.
¿Cuál es tu nombre?
05:55 PM

Daniel Carvajal 05:55 PM ✓

Gracias. Ahora cuéntame brevemente tu situación o consulta,
por favor.
05:55 PM

No encontré lo que buscaba 05:55 PM ✓

¿Deseas dejarnos un medio de contacto para dar seguimiento?
📞 Teléfono o ✉ correo.
Es opcional, Si prefieres no hacerlo, escribe *no*.
05:55 PM

294858191 05:55 PM ✓

Por favor confirma la información:
👤 Nombre: Daniel Carvajal
📝 Consulta: No encontré lo que buscaba
📞 Contacto: 294858191
¿Es correcto?
Responde *sí* para confirmar o *no* para cancelar.
05:55 PM

Si 05:55 PM ✓

Solicitudes Chatbot CSDC - AmiBot ★ 📁 ☁

Archivo Editar Ver Insertar Formato Datos Herramientas Gemini Extensiones Ayuda

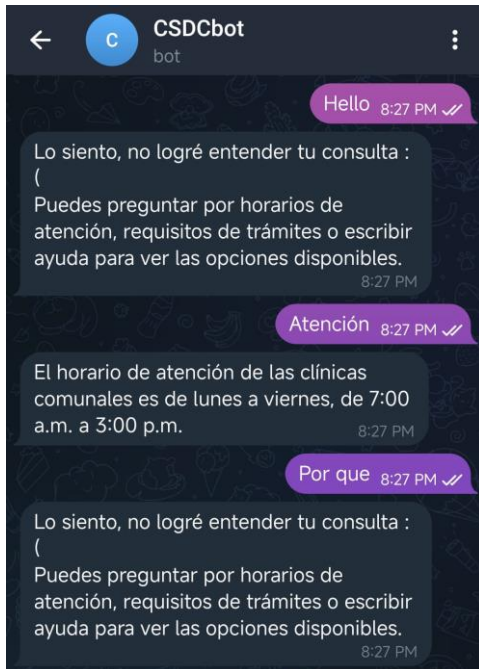
🔍 Menús ↶ ↷ 🖨 📄 100% | € % .0 .00 123 | Predet... | - 10 + | B I 🔗 A

A4:D4 | fx: '2026-02-10 17:56:03

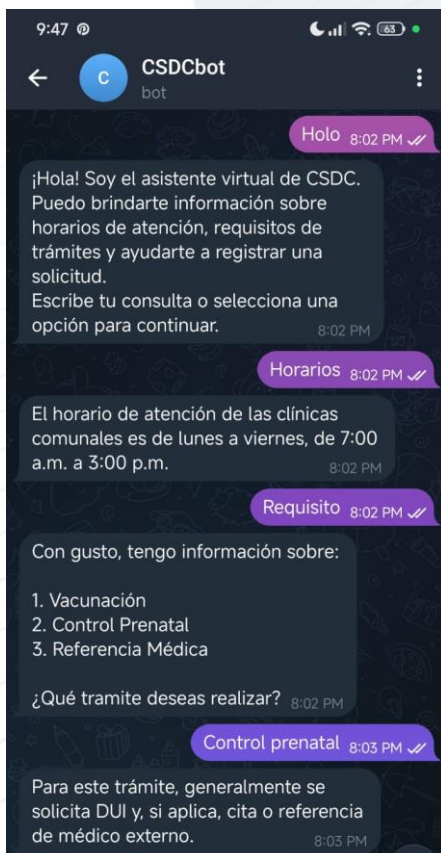
	A	B	C	D
1	Fecha	Nombre	Descripción	Contacto
2	2026-02-09 20:25:44	ana martinez	no encontré información clara sobre mi trámite	ana@email.com
3	2026-02-10 17:46:07	Karen	No encontré la información que necesitaba	698526544
4	2026-02-10 17:56:03	Daniel Carvajal	No encontré lo que buscaba	294858191



Intención: Fallback



Intenciones: Bienvenida, consultar_horarios, requisitos_tramites

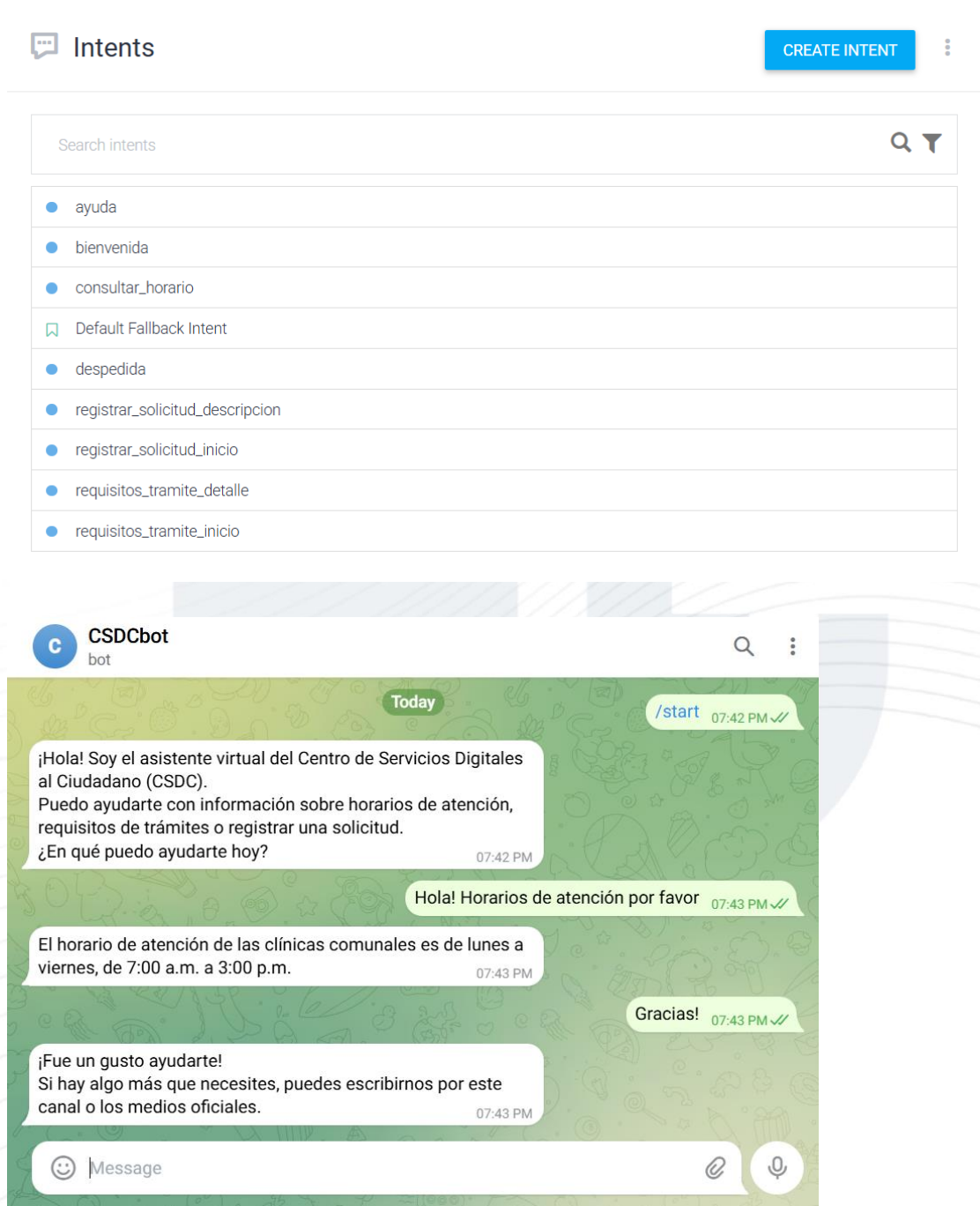


Conclusiones

1. Concluyendo, la Fase 2 permitió materializar el diseño conceptual del chatbot en una solución funcional, demostrando la importancia de contar con diagramas de flujos conversacionales y una arquitectura definida como base para la implementación técnica. La integración de una plataforma conversacional con un backend desarrollado en Python y servicios en la nube para el almacenamiento de información permitió gestionar flujos de múltiples turnos, validar la comunicación entre componentes y asegurar la persistencia de datos, sentando las bases para la escalabilidad y evolución futura del sistema.
2. La implementación del chatbot evidenció el potencial de las tecnologías conversacionales como un canal de apoyo para la atención ciudadana, facilitando el acceso a información básica y la canalización de solicitudes de forma estructurada. Además, el proceso permitió identificar mejoras en la experiencia de usuario y en la organización interna del sistema, lo que contribuye a una solución más usable, mantenible y alineada con las necesidades del proyecto.

Anexos

Estructura previa a Python, todo era gestionado por Dialogflow, sin embargo, los flujos no eran muy estables ya que ciertas palabras podían activar el intent no deseado.



The image shows two screenshots related to a chatbot interface.

The top screenshot displays the "Intents" section of a chatbot configuration tool. It features a search bar labeled "Search intents" and a list of intents:

- ayuda
- bienvenida
- consultar_horario
- Default Fallback Intent
- despedida
- registrar_solicitud_descripcion
- registrar_solicitud_inicio
- requisitos_tramite_detalle
- requisitos_tramite_inicio

A "CREATE INTENT" button is visible in the top right corner.

The bottom screenshot shows a chat conversation with a bot named "CSDCbot". The chat history includes:

- User: /start 07:42 PM ✓
- Bot: ¡Hola! Soy el asistente virtual del Centro de Servicios Digitales al Ciudadano (CSDC). Puedo ayudarte con información sobre horarios de atención, requisitos de trámites o registrar una solicitud. ¿En qué puedo ayudarte hoy? 07:42 PM
- User: Hola! Horarios de atención por favor 07:43 PM ✓
- Bot: El horario de atención de las clínicas comunales es de lunes a viernes, de 7:00 a.m. a 3:00 p.m. 07:43 PM
- User: Gracias! 07:43 PM ✓
- Bot: ¡Fue un gusto ayudarte! Si hay algo más que necesites, puedes escribirnos por este canal o los medios oficiales. 07:43 PM

The chat interface includes a search bar, a "Message" input field, and a microphone icon for voice input.

