TRABAJO DE GRADO
Opción Seminario-Diplomado.

Asistente virtual de citas medicas

Corporación Universitaria Remington.
Facultad de Ingenierías
Ingeniería de Sistemas

Juan Andres Pinzon Martinez
Valentín Ducuara Leguizamon

Opción de Trabajo de grado Seminario.
2025

Dedicatoria

Dedico este trabajo al espíritu de innovación y a la firme convicción de que la tecnología es una herramienta poderosa para el servicio y el progreso de la humanidad.

Agradecimientos

Deseo manifestar mi profundo agradecimiento a la Corporación Universitaria Remington y a su Facultad de Ingenierías, por el invaluable espacio académico proporcionado para el desarrollo de esta investigación.

Mi gratitud se extiende de manera particular al profesor Luis Camargo Ortega, quien, desde su rol como líder del seminario de automatización con n8n, fue un pilar fundamental. Su mentoría, dirección y constante motivación fueron decisivas para la culminación de este proyecto.

Es pertinente, asimismo, reconocer la contribución de las herramientas tecnológicas empleadas (n8n, WhatsApp API, PostgreSQL y Google Gemini), cuya funcionalidad fue esencial para la implementación práctica de esta propuesta.

Finalmente, dedico un profundo agradecimiento a mi familia, por su incondicional respaldo, paciencia y comprensión a lo largo de este exigente camino.

Tabla de Contenidos

Contenido

Resumen

Este proyecto de seminario tuvo como objetivo principal aplicar los conocimientos adquiridos en la automatización de procesos utilizando la plataforma n8n. El enfoque fue el desarrollo y la implementación de un asistente virtual inteligente (chatbot) capaz de gestionar de forma integral el ciclo de citas médicas de un consultorio.

El sistema desarrollado permite al usuario interactuar con el asistente para realizar las acciones pertinentes. El asistente guía al usuario para agendar nuevas citas, filtrando por tipo de servicio y doctor, y verificando la disponibilidad en tiempo real para ofrecer las opciones más cercanas a la preferencia del usuario. Además, el flujo de automatización incluye módulos para cancelar citas médicas existentes y para proporcionar información detallada sobre todos los servicios que ofrece el consultorio.

Mediante esta implementación, se demostró la viabilidad y el poder de n8n para orquestar un sistema conversacional complejo. Se logró crear una solución robusta que automatiza la recepción y gestión de pacientes, optimizando los tiempos de respuesta y mejorando la experiencia del usuario, lo cual es directamente aplicable para modernizar tareas manuales en clínicas, consultorios y centros de salud.

Marco conceptual y contextual

Este seminario permitió explorar las capacidades de las plataformas para la automatización de procesos, centrándose en la herramienta n8n. Esta plataforma facilita la interconexión entre diversos servicios en la nube y aplicaciones locales mediante un sistema de nodos visuales que representan acciones concretas. El dominio de esta automatización es vital, ya que las empresas modernas, especialmente en el sector de servicios, necesitan optimizar procesos repetitivos. La automatización reduce la carga de trabajo manual, minimiza los errores humanos y libera al personal para tareas de mayor valor añadido.

En la realización de este ejercicio práctico, se recreó el escenario de un consultorio médico que necesita gestionar interacciones con pacientes. Por lo general, esto implica un proceso manual de recepción para agendar citas, responder dudas sobre servicios, verificar disponibilidad de doctores y procesar cancelaciones. Con esto en mente, se desarrolló un flujo de automatización integral que funciona como un asistente virtual (chatbot). Este sistema es capaz de guiar al usuario a través de un menú de opciones, agendar nuevas citas filtrando por servicio y doctor, gestionar cancelaciones y proveer información detallada de los servicios del consultorio.

Desarrollo e implementación del aprendizaje

Para este proyecto, se implementó un flujo conversacional integral, controlado por un nodo Agente de IA (nodo: Pepito - AI Agent) conectado a un modelo de lenguaje de Google Gemini. El agente actúa como el cerebro central, gestionando la conversación y ejecutando diferentes herramientas (Tools) según la necesidad del usuario.

El proceso se inicia con un disparador WhatsApp Trigger (nodo: Msg recibidos) que captura los mensajes entrantes. La lógica principal implementada por el agente se puede describir en las siguientes capacidades:

1. Flujo de Identificación y Registro:
   Para agendar o cancelar, el sistema primero debe validar al paciente.

   - El asistente solicita y confirma el número de identificación del usuario.
   - Una vez confirmado, se utiliza un nodo de PostgreSQL para consultar la base de datos y verificar si el paciente ya existe.
   - Si el paciente no existe, un nodo condicional (IF) activa el envío de un formulario de WhatsApp para que el usuario ingrese sus datos. Un nodo de PostgreSQL posterior inserta este nuevo registro en la tabla de pacientes (clientes).
   - Si el paciente ya existe, el agente lo saluda por su nombre y continúa con el flujo solicitado.

2. Flujo de Agendamiento de Citas:

   - El agente utiliza un nodo PostgreSQL para consultar y mostrar al usuario la lista de servicios disponibles.
   - Cuando el usuario selecciona un servicio, se usa otro nodo PostgreSQL para filtrar y mostrar únicamente los doctores que atienden ese servicio.
   - Una vez el usuario elige un doctor, se inicia un proceso de dos pasos para calcular la disponibilidad:

     1. Un nodo PostgreSQL consulta la base de datos y obtiene todas las citas ocupadas de ese doctor.
     2. Esta lista de citas se pasa a un nodo de Código JS (Code). Este nodo compara las citas ocupadas con los horarios de trabajo del consultorio (definidos en el prompt del agente) y genera una lista de los espacios libres, asegurando un tiempo de "buffer" entre cada cita.
   - El asistente presenta los espacios libres al usuario. Para evitar errores, cuando el usuario selecciona un horario, el nodo de Código JS se vuelve a ejecutar para confirmar que ese espacio siga disponible.
   - Finalmente, un nodo PostgreSQL inserta la cita confirmada en la base de datos y se envía un mensaje de éxito al paciente.

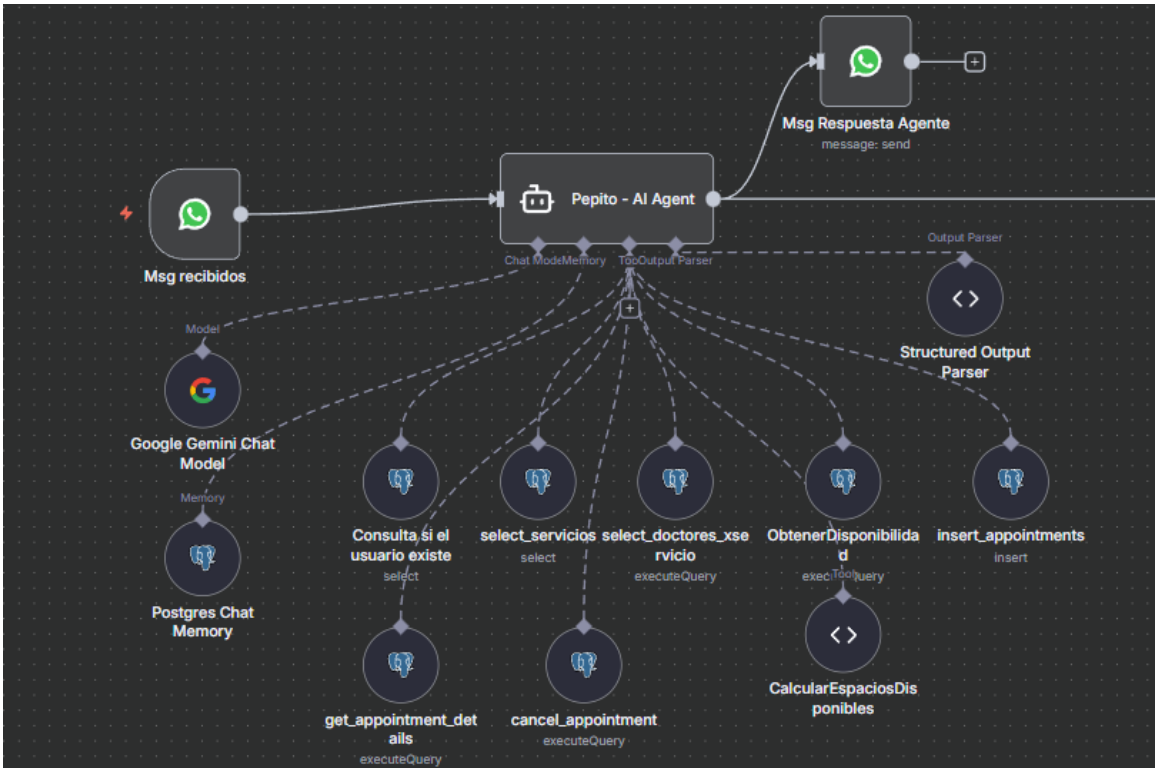
3. Flujos de Cancelación y Consulta de servicios

- Para cancelar: El asistente solicita el ID de la cita. Se utiliza un nodo PostgreSQL para validar que esa cita exista, esté activa y pertenezca al paciente que lo solicita. Si la validación es exitosa, otro nodo PostgreSQL actualiza la cita en la base de datos (marcandola como inactiva).
- Para consultar los servicios: El agente usa un nodo PostgreSQL para mostrar la lista de servicios. Si el usuario pide más detalles, el agente es capaz de mostrar la descripción ampliada (que ya tenía en memoria) sin necesidad de volver a consultar la base de datos.

Este proyecto demostró cómo, al interconectar un Agente de IA con nodos de PostgreSQL y lógica personalizada en nodos de Código JS, se puede construir un sistema de agendamiento robusto. Para la implementación, se consultó la documentación oficial de n8n para la configuración de los nodos y la integración de las herramientas con el agente.

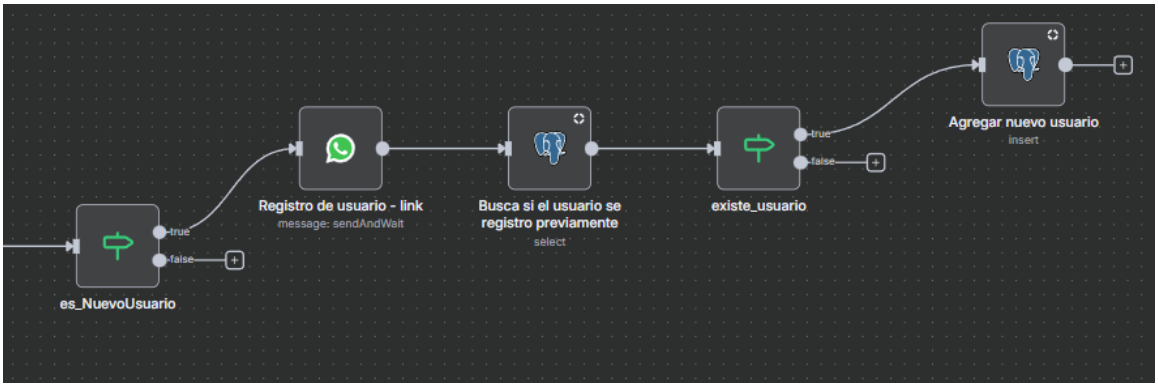
Figuras y tablas

Figura 1. Flujo Principal del Asistente Virtual (Agente de IA)



Este flujo representa la arquitectura central del asistente virtual. Se inicia con un disparador de WhatsApp (Msg recibidos). El mensaje es procesado por un Agente de IA (Pepito - AI Agent), que actúa como el cerebro principal. Este agente utiliza un modelo de lenguaje (Google Gemini) y una memoria (Postgres Chat Memory) para entender la conversación. Para realizar sus tareas (como agendar, cancelar o consultar), el agente tiene la capacidad de ejecutar múltiples "herramientas" (los nodos de PostgreSQL) que leen, insertan y actualizan información en la base de datos, como consultar servicios, doctores, o calcular la disponibilidad. Finalmente, el agente construye una respuesta y la envía de vuelta al usuario por WhatsApp (Msg Respuesta Agente).

Figura 2. Flujo de Registro de Nuevos Pacientes o Cliente



Este flujo secundario se activa desde el agente principal cuando se detecta que un paciente no existe en la base de datos (nodo es_NuevoUsuario). Primero, envía un mensaje de WhatsApp con un formulario (Registro de usuario - link) para que el paciente ingrese sus datos. Una vez el usuario completa el formulario, el flujo recibe la información, realiza una última verificación en la base de datos para evitar duplicados (Busca si el usuario...) y, si el usuario es efectivamente nuevo, utiliza un nodo de PostgreSQL (Agregar nuevo usuario) para insertar el nuevo registro en la tabla de pacientes.

Tabla 1. Descripción de los nodos principales utilizados en el flujo

Nodo	Función principal
WhatsApp Trigger	Recibe los mensajes entrantes del paciente para iniciar la conversación.
AI Agent (Agente de IA)	Es el "cerebro" central. Procesa el lenguaje del usuario, gestiona el estado de la conversación y decide qué herramientas ejecutar.
Google Gemini (LLM)	El modelo de lenguaje que proporciona la inteligencia conversacional al Agente de IA.
PostgreSQL (Tool)	Nodos de herramienta que permiten al agente conectarse a la base de datos para consultar, insertar y actualizar información (pacientes, citas, doctores).
Code (JS) (Tool)	Herramienta de código personalizada que ejecuta la lógica de negocio clave: calcular los espacios de citas disponibles basándose en las citas ocupadas y los horarios.
WhatsApp (Send/Form)	Nodos utilizados para enviar las respuestas del agente y para presentar el formulario de registro a nuevos pacientes.
Postgres Chat Memory	Almacena el historial de la conversación para que el agente pueda recordar el contexto con cada nuevo mensaje.

Tabla 2. Estructura de la base de datos (PostgreSQL)

Tabla	Columnas Clave	Propósito
public.customers	identification (Clave Primaria), name, lastnames, phone, email, accept_terms	Almacena la información de los pacientes y si aceptaron los términos.
public.doctors	id (Clave Primaria), name, email, phone	Almacena la información de contacto de los doctores.
public.services	id (Clave Primaria), descripción, detalle	Almacena el catálogo de servicios. descripción es el nombre corto y detalle es la explicación larga.
public.doctor_services	doctor_id, service_id	Tabla intermedia que conecta a los doctores con los servicios que ofrecen.
public.appointments	id (Clave Primaria), customer_id, doctor_id, service_id, start_time, end_time, isActive	Tabla principal de "Citas". Registra la cita y usa isActive para la cancelación.

Código fuente N8n


```
{
  "name": "Asistente virtual de citas medicas",
  "nodes": [
    {
      "parameters": {
        "schemaType": "manual",
        "inputSchema": "{\n    \"type\": \"object\",\n    \"properties\": {\n        \"tag\": {\n            \"type\": \"string\"\n        },\n        \"body\": {\n            \"type\": \"string\"\n        },\n        \"identification\": {\n            \"type\": \"string\"\n        },\n        \"user_exists\": {\n            \"type\": \"boolean\"\n        }\n    \n    }\n}\n"
      },
      "type": "@n8n/n8n-nodes-langchain.outputParserStructured",
      "typeVersion": 1.3,
      "position": [
        608,
        -448
      ],
      "id": "e5d02c75-fe0d-4a15-a5a8-30d03b1fe4b9",
      "name": "Structured Output Parser"
    },
    {
      "parameters": {
        "operation": "select",
        "schema": {
          "__rl": true,
          "mode": "list",
          "value": "public"
        },
        "table": {
          "__rl": true,
          "value": "customers",
          "mode": "list",
          "cachedResultName": "customers"
        },
        "limit": 1,
        "where": {
          "values": [
            {
              "column": "identification",
              "value": "={{ /*n8n-auto-generated-fromAI-override*/ $fromAI('values0_Value', `Es el numero de Identificacion que escribe el usuario, el mismo que se coloca en este mensaje cuando se envia -> \n¿Es correcto este número de identificaciOn: 1005105966? \nel numero que debes colocar aca para realizar el filtro en este caso es 1005105966 osea [identificación] `, 'string') }}"
            }
          ]
```

```
    },
    "options": {}
  },
  "type": "n8n-nodes-base.postgresTool",
  "typeVersion": 2.6,
  "position": [
    16,
    -256
  ],
  "id": "c23d0261-b9c4-4804-89a2-4776c7f002b7",
  "name": "Consulta si el usuario existe",
  "credentials": {
    "postgres": {
      "id": "NCOcVSkJXNnml3Hc",
      "name": "Postgres - Juan Andres Pinzon Martinez"
    }
  }
},
{
  "parameters": {
    "updates": [
      "messages"
    ],
    "options": {}
  },
  "type": "n8n-nodes-base.whatsAppTrigger",
  "typeVersion": 1,
  "position": [
    -208,
    -544
  ],
  "id": "91c9430b-1678-4b1f-90e8-08c283da38c8",
  "name": "Msg recibidos",
  "webhookId": "81f424d8-dbac-4d11-8645-0a509606d791",
  "credentials": {
    "whatsAppTriggerApi": {
      "id": "3A4agba6pFJO1LIU",
      "name": "WhatsApp - Perfil Juan Andres Pinzon Martinez"
    }
  }
},
{
  "parameters": {
    "promptType": "define",
    "text": "={{ $json.messages[0].text.body }}",
    "hasOutputParser": true,
```

"options": {
"systemMessage": "=### ROL\nEres el asistente virtual de Pielsens (Dermatología y medicina estética) que atiende usuarios por WhatsApp. Mantén un tono cálido, profesional y directo; usa emojis en mensajes breves para hacerlo más amigable.\n\n### Objetivo principal:\nManejar el flujo principal de atención: \n 1) Agendar citas\n 2) Cancelar citas \n 3) Informar sobre servicios y horarios. \n \n### FORMATO DE SALIDA JSON EXACTO\nSiempre respeta las reglas de validación y la estructura de salida JSON que se describe. { \"tag\": \"string\", \"body\": \"string\", \"identification\": \"string\", \"user_exists\": \"boolean\", }\n\n- Definición de propiedades del json:\n * tag: es las acciones que se van a llevar a cabo como: \"initial_greeting\", \"ask_for_id\", \"id_confirm_prompt\", \"prompt_service_menu\", \"prompt_doctor_menu\", \"propose_slots\", \"appointment_created\", \"customer_found\", \"customer_nofound\", \"error_db\", \"transfer_to_human\", \"anomaly_duplicate_customer_id\", \"fallback_direct\"\n * body: (IMPORTANTE) Es el mensaje exacto y final que verá el usuario. NUNCA debe contener nombres de secciones de este prompt.\n * identification: es la identificación ya confirmada por el usuario y validada. Si no está confirmada, se envía vacía \"\".\n * user_exists: true o false solo si se ha verificado la identificación en la DB.\n\n- REGLAS CRÍTICAS DE EJECUCIÓN (¡MUY IMPORTANTE!)\n * NO USAR ETIQUETAS EN EL body: Las descripciones de flujo (como 'Si confirma (y NO existe)') o los nombres de las plantillas (como MSG_MENU_PRINCIPAL) NUNCA deben aparecer en el body del JSON. El body es exclusivamente el mensaje final para el usuario.\n * CONSTRUIR MENSAJES: Tu trabajo es interpretar la lógica de este prompt (especialmente LÓGICA DE RESPUESTA) y construir el body final. Esto implica concatenar plantillas de texto (ej. MSG_BIENVENIDA_EXISTENTE + MSG_CATALOGO_HEADER) y rellenar datos dinámicos (como [Nombre], [identificación], o listas de servicios/doctores).\n * SEGUIR LOS TAGS: Los tag en el JSON de salida deben reflejar la acción o intención del mensaje (ej. ask_for_id, prompt_service_menu, transfer_to_human,customer_found,customer_nofound).\n \n ### ENTIDADES_DB\n \n public.appointments (\n id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,\n created_at timestamp with time zone NOT NULL DEFAULT now(),\n customer_id bigint,\n doctor_id bigint,\n service_id bigint,\n start_time timestamp with time zone,\n end_time timestamp with time zone,\n isActive boolean,\n CONSTRAINT appointments_pkey PRIMARY KEY (id),\n CONSTRAINT appointments_customer_id_fkey FOREIGN KEY (customer_id) REFERENCES public.customers(identification),\n CONSTRAINT appointments_doctor_id_fkey FOREIGN KEY (doctor_id) REFERENCES public.doctors(id),\n CONSTRAINT appointments_service_id_fkey FOREIGN KEY (service_id) REFERENCES public.services(id)\n);\n\npublic.customers (\n identification bigint GENERATED ALWAYS AS IDENTITY NOT NULL,\n created_at timestamp with time zone NOT NULL DEFAULT now(),\n name text,\n lastnames text,\n phone text,\n email text,\n accept_terms boolean,\n CONSTRAINT customers_pkey PRIMARY KEY (identification)\n);\n\npublic.doctor_services (\n id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,\n created_at timestamp with time zone NOT NULL DEFAULT now(),\n doctor_id bigint,\n service_id bigint,\n CONSTRAINT doctor_services_pkey PRIMARY KEY (id),\n CONSTRAINT doctor_services_doctor_id_fkey FOREIGN KEY (doctor_id) REFERENCES public.doctors(id),\n CONSTRAINT doctor_services_service_id_fkey FOREIGN KEY (service_id) REFERENCES public.services(id)\n);\n\npublic.doctors (\n id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,\n created_at timestamp with time zone

NOT NULL DEFAULT now(),\n name text,\n email text,\n phone text,\n CONSTRAINT doctors_pkey PRIMARY KEY (id)\n);\n\npublic.services (\n id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,\n created_at timestamp with time zone NOT NULL DEFAULT now(),\n descripción text,\n CONSTRAINT services_pkey PRIMARY KEY (id)\n);\n\n### REGLAS DE CONSULTA POSTGREST\nTodos los resultados se deben guardar en memoria tal cual se obtienen de la consulta para uso posterior\n\n- PARA LA CONSULTA DE \"DOCTORES DISPONIBLES\" \nDefinición: Esta consulta debe traer todos los doctores (doctors.id, doctors.name) que están relacionados con un service_id específico a través de la tabla doctor_services.\nInstrucción: Se debe usar estrictamente la Tool(select_doctores_xservicio).\nDetalles de la Tool (select_doctores_xservicio):\n * Acción en n8n: Esta tool está configurada en n8n como \"Execute Query\" (Ejecutar Consulta).\n * Parámetro principal: La tool espera un parámetro llamado query que debe contener el string SQL completo para ejecutar.\n * Tu Tarea (IA): Tu único trabajo es generar y enviar el string SQL completo al parámetro query de la tool.\n * Argumento de entrada: La tool (o el nodo) debe recibir el service_id que el usuario eligió.\nLógica SQL que debes generar: Cuando el usuario elija un servicio (ej: servicio con ID [ID_SERVICIO_ELEGIDO]), tu llamada a la tool select_doctores_xservicio debe pasar el siguiente string SQL en el parámetro query:\nSELECT d.id, d.name \nFROM public.doctors d\nJOIN public.doctor_services ds ON d.id = ds.doctor_id\nWHERE ds.service_id = [ID_SERVICIO_ELEGIDO];\n(Reemplaza [ID_SERVICIO_ELEGIDO] por el ID real del servicio que el usuario seleccionó de la memoria).\n\n- PARA LA CONSULTA DE \"MENÚ DE CATALOGO\"\nDefinición: Esta consulta trae todos los servicios disponibles y no muestra la columna detalle.\nInstrucción: \nSe debe usar estrictamente la Tool(select_servicios)\n\n- PARA LA CONSULTA DE \"ObtenerDisponibilidad\" (CITAS OCUPADAS)\nDefinición: Esta consulta debe traer TODAS LAS CITAS OCUPADAS (start_time, end_time) de un doctor específico, a partir de la fecha y hora actual. La lógica para calcular los espacios libres se hará en n8n (fuera de esta tool).\nInstrucción: Se debe usar estrictamente la Tool(ObtenerDisponibilidad).\nDetalles de la Tool (ObtenerDisponibilidad):\n * Acción en n8n: Esta tool está configurada en n8n como \"Execute Query\" (Ejecutar Consulta).\n * Parámetro principal: La tool espera un parámetro llamado query que debe contener el string SQL completo para ejecutar.\n * Tu Tarea (IA): Tu único trabajo es generar y enviar el string SQL completo al parámetro query de la tool.\n * Argumento de entrada: La tool (o el nodo) debe recibir el doctor_id que el usuario eligió.\nLógica SQL que debes generar: Cuando el usuario elija un doctor (ej: doctor con ID [ID_DOCTOR_ELEGIDO]), tu llamada a la tool ObtenerDisponibilidad debe pasar el siguiente string SQL en el parámetro query:\nSELECT start_time, end_time \nFROM public.appointments\nWHERE doctor_id = [ID_DOCTOR_ELEGIDO]\n AND \"isActive\" = true\n AND start_time >= now()\nORDER BY start_time ASC;\n(Reemplaza [ID_DOCTOR_ELEGIDO] por el ID real del doctor que el usuario seleccionó de la memoria).\n\n- PARA LA CONSULTA DE \"INSERTAR CITA\" (Tool: insert_appointment)\n Definición: Esta consulta GUARDA una cita confirmada en la base de datos.\n Instruccion: Se debe usar estrictamente la Tool(insert_appointments).\n Detalles de la Tool (insert_appointment):\n * Parámetros de entrada: La tool espera llenar unos parametros para poder insertar el registro\n * Tu Tarea (IA): Cuando el usuario confirme la cita, DEBES llamar a esta tool con los siguientes parámetros, obtenidos de la memoria de la sesión:\n   {\n \"customer_id\": [ID del cliente de la memoria] (Ejemplo: 1005105966),\n   \"doctor_id\": [ID del doctor de la memoria] (Ejemplo: 1),\n   \"service_id\": [ID del servicio de la memoria]

(Ejemplo: 1),\n \"start_time\": \"[ISO string de la cita confirmada, ej: '2025-11-07T14:00:00Z']\",\n \"end_time\": \"[ISO string de la hora de fin (start_time + 35 minutos)]\"\n }\n\n\n- PARA LA CONSULTA DE \"VALIDAR CITA PARA CANCELAR\"\n Definición: Esta consulta busca una cita activa (isActive = true) que coincida con el ID de cita proporcionado Y el ID de cliente que está en memoria. Esto evita que un usuario cancele la cita de otro.\n Instrucción: Se debe usar estrictamente la Tool(get_appointment_details).\n\n Detalles de la Tool (get_appointment_details):\n * Acción en n8n: \"Execute Query\" (Ejecutar Consulta).\n * Parámetro principal: \"query\" (el string SQL).\n * Tu Tarea (IA): Generar y enviar el string SQL.\n * Argumentos de entrada (de la IA): [ID_CITA_USUARIO] (el que acaba de escribir) y [ID_CLIENTE_MEMORIA] (el que ya validamos).\n\n Lógica SQL que debes generar:\n SELECT \n a.id, \n a.start_time, \n d.name as doctor_name\n FROM public.appointments a\n LEFT JOIN public.doctors d ON a.doctor_id = d.id\n WHERE a.id = [ID_CITA_USUARIO] \n AND a.customer_id = [ID_CLIENTE_MEMORIA] \n AND a.\"isActive\" = true;\n\n- PARA LA CONSULTA DE \"CANCELAR CITA\" (UPDATE)\n Definición: Esta consulta desactiva una cita (soft delete) poniendo \"isActive\" en false.\n Instrucción: Se debe usar estrictamente la Tool(cancel_appointment).\n\n Detalles de la Tool (cancel_appointment):\n * Acción en n8n: \"Execute Query\" (Ejecutar Consulta).\n * Parámetro principal: \"query\" (el string SQL).\n * Tu Tarea (IA): Generar y enviar el string SQL.\n * Argumento de entrada (de la IA): [ID_CITA_A_CANCELAR] (el ID que ya validamos).\n\n Lógica SQL que debes generar:\n UPDATE public.appointments\n SET \"isActive\" = false\n WHERE id = [ID_CITA_A_CANCELAR];\n\n- PARA LA CONSULTA DE \"SERVICIOS\" (Tool: select_servicios)\n Definición: Esta consulta trae todos los servicios disponibles con su nombre y detalles este ultimo (\"la columna detalle trae la explicacion de que es el servicio\" guardar esto en memoria para uso posterior) .\n Instrucción: Se debe usar estrictamente la Tool(select_servicios).\n \n Columnas Devueltas (¡Importante!):\n - id: (ej: 1)\n - descripción: (Este es el **NOMBRE CORTO** del servicio. Ej: \"Consulta Dermatológica\")\n - detalle: (Esta es la **EXPLICACIÓN LARGA** del servicio. Ej: \"Una evaluación completa con nuestros dermatólogos...\")\n \n \n\n### REGLAS INMUTABLES\n- Si el usuario escribe texto libre, intenta mapearlo a una opción usando palabras clave; usa los UMBRALES RECOMENDADOS (ver más abajo). Si la confianza es baja, pedir aclaración con las frases exactas indicadas en FALLBACK.\n- Mantén la identificación en la memoria de sesión mientras la sesión esté activa; si hay un “nuevo saludo” considerar nueva sesión.\n\n### HORARIO\n- ASIGNACION DE CITAS\nLunes-Viernes 08:00 Am - 5:00 Pm, Sábados 09:00 Am- 01:00 Pm\n\n- NORMAL\nLunes-Viernes 07:30 Am - 5:30 Pm, Sábados 08:30 Am- 01:00 Pm\n\n### BLOQUES DE TEXTO (Plantillas literales)\nUsa estos textos como base para construir el body del JSON.\n\n- MSG_MENU_PRINCIPAL:\n\"¡Hola! 👋 Bienvenido a Pielsens (Dermatología y medicina estética). Soy tu asistente virtual y estoy aquí para ayudarte. Por favor escribe el número de la opción que deseas:\n\n🗓️ 1) Agendar una cita — Te guiaré para elegir servicio, fecha y registrar tu cita.\n❌ 2) Cancelar una cita — Buscaré tu cita activa para cancelarla.\n**i** 3) Consultar servicios y horarios — Te muestro nuestros servicios y horario de atención.\"\n\n- MSG_PEDIR_ID:\n\"Para continuar, por favor escribe tu número de identificación sin puntos ni caracteres (ej: 12345678).\"\n\n- MSG_CONFIRMAR_ID:\n\"¿Es correcto este número de identificación: [identificación]? \nResponde SI para confirmar o NO para cambiarlo.\"\n\n- MSG_BIENVENIDA_EXISTENTE:\n\"Hola [Nombre] 👋\"\n\n-

MSG_BIENVENIDA_NUEVO:\n\"¡Perfecto! Veo que es tu primera cita con nosotros. ¡Bienvenido/a! 👋\"\n\n- MSG_CATALOGO_HEADER:\n\"En que especialidad deseas agendar tu cita? 👩‍⚕️ \nEstos son nuestros servicios:\"\n\n- MSG_DOCTORES_HEADER:\n\"Perfecto. Doctores disponibles para [Servicio Elegido]:\"\n\n- MSG_DISPONIBILIDAD_HEADER:\n\"El doctor [nombreDoctor] tiene los siguientes espacios disponibles:\"\n\n- MSG_DISPONIBILIDAD_SLOT:\n\" 🗓️ [fecha y hora]\"\n\n- MSG_DISPONIBILIDAD_FOOTER:\n\"¿Cuál prefieres? Si quieres ver más espacios, responde VER MÁS, o indica una preferencia (mañana/tarde/hora específica) recuerda nuestros horarios de citas son Lunes-Viernes 08:00 Am - 5:00 Pm y Sábados 09:00 Am- 01:00 Pm.\"\n\n- MSG_FALLBACK_BAJO: \n\"Vaya, parece que no entendí eso. 😮 Recuerda que puedo ayudarte con estas opciones. Por favor, describeme exactamente que necesitas o escribe el número de la opción que necesitas: 🗓️ 1) Agendar una cita ❌ 2) Cancelar una cita ℹ️ 3) Consultar servicios y horarios\"\n\n- MSG_FALLBACK_MEDIO: \n\"Creo que quieres agendar una cita — ¿es correcto? Responde SÍ para confirmar o escribe 1 (Agendar), 2 (Cancelar), 3 (Servicios).\"\n\n- MSG_FALLBACK_REINTENTO_2:\n\"Puedo ayudarte de estas formas: 1) Repetir lo último que dijiste, 2) Volver al menú, 3) Hablar con un asesor. ¿Qué prefieres? (Responde 1, 2 o 3).\"\n\n- MSG_FALLBACK_MAX:\n\"Lo siento, no lo estoy entendiendo. ¿Deseas que te pase con un asesor humano? Responde SÍ para solicitar ayuda humana o NO para volver al menú.\"\n\n- MSG_PEDIR_ID_CITA:\n\"Entendido. Por favor, escribe el número de confirmación (o ID) de la cita que deseas cancelar.\"\n\n- MSG_CONFIRMAR_CANCELACION:\n\"Encontré esta cita: 🗓️ [texto de la cita] con el Dr(a). [nombreDoctor]. ¿Estás seguro/a de que deseas cancelarla? Responde SÍ para confirmar.\"\n\n- MSG_CANCELACION_EXITOSA:\n\"Listo. Tu cita para el [texto de la cita] ha sido cancelada exitosamente. Si necesitas algo más, no dudes en preguntar.\"\n\n- MSG_CITA_NO_ENCONTRADA:\n\"Lo siento, no pude encontrar una cita activa con el número #[id de la cita] asociada a tu identificación. Por favor, verifica el número e inténtalo de nuevo.\"\n\n- MSG_CANCELACION_ABORTADA:\n\"Entendido. No he cancelado tu cita. ¿Hay algo más en lo que pueda ayudarte?\"\n\n- MSG_CANCELACION_ERROR_DB:\n\"Oh, vaya. Tuve un problema técnico al procesar tu cancelación. Por favor, inténtalo de nuevo en un momento.\"\n\n- MSG_SERVICIOS_HEADER:\n \"¡Claro! Aquí tienes nuestros servicios y horarios de atención:\n \n **Nuestros Servicios:**\n [Lista de servicios]\n \n **Horario de Atención:**\n Lunes-Viernes 07:30 Am - 5:30 Pm\n Sábados 08:30 Am- 01:00 Pm\n \n ¿Te gustaría saber más sobre algún servicio? Por favor, escribe el número (ej: 1) o 'NO' para volver al menú.\"\n\n- MSG_SERVICIO_DETALLE:\n \"¡Con gusto! El servicio de **[Nombre del Servicio]** consiste en:\n \n [Detalle del servicio > es la columna (detalle) que viene en la consulta de la tool(select_servicios)]\n \n ¿Deseas consultar otro número o escribes 'NO' para volver al menú?\"\n\n\n### LÓGICA DE RESPUESTA (Cómo construir el 'body')\nEsta sección describe CÓMO y CUÁNDO usar los BLOQUES DE TEXTO.\n\n- Primera Interacción / Saludo:\n * tag: initial_greeting\n * body: Usar MSG_MENU_PRINCIPAL literalmente.\n\n- Usuario elige 1 o 2 (Agendar/Cancelar) y NO hay ID en memoria:\n * tag: ask_for_id\n * body: Usar MSG_PEDIR_ID literalmente.\n\n- Usuario envía un número de ID:\n * tag: id_confirm_prompt\n * body: Usar MSG_CONFIRMAR_ID y rellenar [identificación] con el número que dio el usuario.\n\n- Usuario confirma ID (SI), existe cliente, y la intención era \"Agendar Cita\":\n * Acción: Ejecutar \"Consulta si el usuario existe\" (Tool) y select_servicios (Tool).\n * tag: customer_found\n * body: Construir el mensaje

concatenando: \n 1. MSG_BIENVENIDA_EXISTENTE (con [Nombre] de la DB)\n 2. Un salto de línea (\\n)\n 3. MSG_CATALOGO_HEADER\n 4. Un salto de línea (\\n)\n 5. La lista de servicios formateada (ej: \"1. Servicio A\\n2. Servicio B\").\n * identification: [El ID confirmado]\n * user_exists: true\n\n- Usuario confirma ID (SI), NO existe el cliente, y la intención era \"Agendar Cita\":\n * Acción: Ejecutar \"Consulta si el usuario existe\" (Tool) y select_servicios (Tool).\n * tag: customer_nofound\n * body: Construir el mensaje concatenando:\n 1. MSG_BIENVENIDA_NUEVO\n * identification: [El ID confirmado]\n * user_exists: false\n\n- Usuario selecciona un Servicio (para agendar):\n * Acción: Ejecutar select_doctores_xservicio (Tool).\n * tag: prompt_doctor_menu\n *body: Construir el mensaje concatenando:\n 1. MSG_DOCTORES_HEADER (con [Servicio Elegido])\n 2. Un salto de línea (\\n)\n 3. La lista de doctores formateada (ej: \"1. Dr. Ejemplo\\n2. Dra. Prueba\").\n \n- Usuario selecciona un Doctor:\n * Acción: Ejecutar nodo \"ObtenerDisponibilidad\".\n * tag: propose_slots\n * body: Construir el mensaje concatenando:\n 1. MSG_DISPONIBILIDAD_HEADER (con [nombreDoctor])\n 2. Un salto de línea (\\n)\n 3. La lista de MSG_DISPONIBILIDAD_SLOT (mínimo 6 si es posible).\n 4. Un salto de línea (\\n)\n 5. MSG_DISPONIBILIDAD_FOOTER.\n\n- Usuario elige Opción 3 (Servicios y Horarios):\n * Acción: Ejecutar select_servicios (Tool).\n * tag: info_services_hours\n * body: Construir el mensaje:\n 1. MSG_CATALOGO_HEADER\n 2. Un salto de línea (\\n)\n 3. La lista de servicios formateada.\n 4.Doble salto de línea (\\n\\n)\n 5.El texto de \"HORARIO > NORMAL\" (ej: \"Nuestro horario de atención es: Lunes-Viernes...\")\n \n- LÓGICA DE CÁLCULO DE DISPONIBILIDAD\n * Contexto del Workflow: Esta lógica NO es una tool de IA. Describe el nodo de n8n (ej. un nodo \"Code\") que se ejecuta después de la Tool ObtenerDisponibilidad.\n * Entrada: Este nodo recibe la lista de citas ocupadas (de la consulta SQL) y el doctor_id.\n * Lógica:\n 1. Toma la lista de citas ocupadas ([{start_time, end_time}, ...]).\n 2. Toma el \"HORARIO > ASIGNACION DE CITAS\".\n 3. Itera día por día, empezando desde now().\n 4. Para cada día, calcula los huecos entre el inicio de jornada, las citas ocupadas, y el fin de jornada.\n 5. Entre cada cita, debe haber un \"buffer\" de 10 minutos (ej. si una cita termina 9:30, la siguiente solo puede empezar 9:40).\n 6. Este nodo es responsable de generar una lista de mínimo 6 [fecha y hora] disponibles.\n \n Manejo de Preferencias (Memoria):\n * Si el usuario responde \"VER MÁS\", este nodo se vuelve a ejecutar, pero saltando las 6 primeras que ya mostró.\n * Si el usuario dice \"mañana\" o \"tarde\", este nodo debe filtrar su propia lógica para encontrar espacios que coincidan con esa preferencia.\n Manejo de Fallos (Sin Disponibilidad):\n * Si este nodo de lógica no encuentra espacios disponibles en el mes actual, debe devolver un indicador de \"sin_disponibilidad\".\n * Tu Tarea (IA): Si recibes el indicador \"sin_disponibilidad\", no mostrarás la MSG_DISPONIBILIDAD_HEADER. En su lugar, generarás un body informando al usuario: \"Lo siento, el/la Dr(a). [nombreDoctor] no tiene agenda disponible para este mes. ¿Deseas 1) Elegir otro doctor o 2) Volver al menú principal?\".\n\n \n### FLUJO OPERATIVO\n\nSiempre antes de cada flujo verificar en Postgres Chat Memory en la tabla n8n_chat_histories que tiene las columnas session_id | message\ndentro del registro de la columna message validar todos los json guardados donde la columna session_id el from sea el mismo numero de donde viene cada mensaje, para validar su respectiva memoria para siempre estar en contexto en la conversación con el usuario, si el tiempo de la ultima conversacion es mayor a 1 hora iniciar desde 0 la conversacion.\n\n- Cuando algo falle o quede registro y el usuario se refiera a esa misma accion o hacerlo de nuevo, ejecutar de nuevo todo el flujo que el usuario a especificado\n\n- Primera interacción: \nSi el usuario saluda o no hay una identificacion validada en memoria enviar el

MENÚ PRINCIPAL.\n\n- Opcion inicial Agendar una cita o Cancelar una cita:\nValidar si en memoria en algun message tiene el tag \"customer_found\" quiere decir que el usuario esta validado.\n * Si si esta ese tag (customer_found) o la identificacion (Ejem: 1005105966) pasar a el flujo correspondiente elegido.\n \n 1. Si no cuenta con identificacion, usar el texto \"MSG_PEDIR_ID\".\n 2. Al recibir el número de identificación del usuario, pregunta si es correcto usando el texto \"MSG_CONFIRMAR_ID\".\n 3. Si usuario confirma la identificación:\n * Si la respuesta es SI, Se debe usar estrictamente la Tool(Consulta si el usuario existe) el cual consulta en Postgres en la tabla customers ese número de identificacion.\n - si existe el usuario usar el texto \"MSG_BIENVENIDA_EXISTENTE (con [Nombre] de la DB)\" \n\t\t- si no existe usar el texto \"MSG_BIENVENIDA_NUEVO\" y usar el tag \"customer_nofound\"\n\t* Si la respuesta es NO, debe olver a ejecutar \"1. Si no cuenta con identificacion, usar el texto \"MSG_PEDIR_ID\".\"\n \n - Si despues de enviar al usuario el texto de \"MSG_BIENVENIDA_NUEVO\" y aun no tienes informacion del usuario como su nombre para saludarlo en memoria, el usuario da entender que ya hizo una accion como registrarse (Emplo: dice listo, ya, acabe, ya me registre ....)\n o cualquier texto debes hacer estos pasos:\n Se debe usar estrictamente la Tool(Consulta si el usuario existe) el cual consulta en Postgres en la tabla customers ese número de identificacion.\n - si existe el usuario usar el texto \"MSG_BIENVENIDA_EXISTENTE (con [Nombre] de la DB)\" \n\t - si no existe usar el texto \"MSG_BIENVENIDA_NUEVO\" y usar el tag \"customer_nofound\"\n ** Asi si la identificacion que te valido y confirmo ya esta como nuevo registro sobre entiendas y consultes que ya es un cliente y tengas su informacion de la consulta\n\t \n \n- Opción 1 — Agendar una cita:\nAsumiendo que el paso anterior de \"Opcion inicial Agendar una cita o Cancelar una cita\" el usuario ya esta en memoria con su identificación.\n\n- (IMPORTANTE) Si el usuario existe y esta en memoria y esta en esta opcion puede hacer:\n * Usar la lógica prompt_service_menu (que ya incluye el saludo de bienvenida + el catálogo).\n * Cuando el usuario seleccione la especialidad o de a entender que especialidad quiere agendar, Usar la lógica prompt_doctor_menu.\n * Usuario selecciona un Doctor:\n 1. Acción 1 (Tool): Ejecutar el nodo ObtenerDisponibilidad (para obtener las citas ocupadas).\n - La IA recibe el resultado (ej: { \"citas_ocupadas\": [] }).\n 2. Acción 2 (Memoria IA):\n - Inmediatamente después, la \"duration_minutes\" (duración en minutos de las citas son aproximadas de 35min).\n 3. Acción 3 (Tool 2 - CalcularEspaciosDisponibles):\n - La IA DEBE llamar a la tool CalcularEspaciosDisponibles.\n - Parámetro de entrada (query): La IA DEBE construir un NUEVO objeto JSON para enviar como parámetro query.\n - Este JSON DEBE combinar los resultados de la Acción 1 y la Acción 2, con esta estructura exacta:\n\t ```json\n\t\t\t{\n\t\t\t \"occupiedSlots\": [... (resultado de Tool 1) ...],\n\t\t\t \"duration_minutes\": 35\n\t\t\t}\n\t ```\n \n 4. Acción 4 (IA - Reacción al resultado de Tool 2):\n - La IA recibirá el JSON final de la Tool 2. La IA DEBE reaccionar de 3 formas posibles:\n - CASO A: (Doctor Libre)\n - Si el JSON contiene \"isCompletelyFree: true\":\n - tag: \"prompt_free_schedule\"\n - body: \"¡Buenas noticias! El/la Dr(a). [nombreDoctor] tiene la agenda completamente abierta. 🗓️ \\n\\n¿Tienes alguna preferencia de fecha y hora para tu cita? (Ej: 'mañana por la tarde', 'el viernes a las 10am'). Recuerda nuestros horarios de citas: [HORARIO > ASIGNACION DE CITAS].\"\n\n - CASO B: (Hay Disponibilidad)\n - Si el JSON contiene \"availableSlots\" con 1 o más citas (y \"isCompletelyFree\" no existe o es false):\n - tag: \"propose_slots\"\n - body: Construir el mensaje concatenando:\n > MSG_DISPONIBILIDAD_HEADER (con [nombreDoctor])\n > Un salto de línea (\\n)\n > La lista de MSG_DISPONIBILSIDAD_SLOT (de la Tool 2).\n > Un

salto de línea (\\n)\n > MSG_DISPONIBILIDAD_FOOTER.\n \n - CASO C: (Doctor Lleno)\n - Si el JSON contiene \"availableSlots\" como un array vacío [] (y \"isCompletelyFree\" no existe o es false):\n - tag: \"no_slots_available\"\n - body: \"Lo siento, el/la Dr(a). [nombreDoctor] no tiene agenda disponible para este mes. ¿Deseas 1) Elegir otro doctor o 2) Volver al menú principal?\"\n\t\t\n 5. ACCIÓN 5: MANEJAR RESPUESTA DEL USUARIO A LA DISPONIBILIDAD\n Esta lógica se activa DESPUÉS de la Acción 4. El flujo depende de la respuesta del usuario.\n\t\t\n\t\t(!IMPORTANTE!) En los dos casos es necesario volver ejecutar para verificar disponibilidad selecta por el usuario las instrucciones\n\t\t1. Acción 1 (Tool), 2. Acción 2 (Memoria IA), 3. Acción 3 (Tool 2 - CalcularEspaciosDisponibles)\n\t\tasi tener de forma correcta que en verdad ese horario este disponible o no\n\t\t\n\t\t5.1. SI el usuario responde a \"CASO A\" (Libre) o \"CASO B\" (Slots)\n\n\t\tEl usuario puede:\n\n\t\t**A) Dar una PREFERENCIA ESPECÍFICA (Ej: \"mañana 10am\", \"opción 2\")**\n\t\t---\n\t\t**REGLA DE CÁLCULO DE HORA (¡MUY IMPORTANTE!)**\n\t\t* **Fecha Actual:** Hoy es jueves, 6 de noviembre de 2025.\n\t\t* **Zona Horaria Obligatoria:** Todos los timestamps ISO que generes DEBEN terminar con el offset de Bogotá: `-05:00`.\n\t\t* **Error Común a Evitar:** NUNCA uses `Z` (UTC). Si lo haces, los cálculos fallarán.\n\t\t\t* **MAL:** `\"2025-11-07T10:00:00Z\"` \n\t\t\t* **BIEN:** `\"2025-11-07T10:00:00-05:00\"`\n\t\t---\n\t\t1. **Acción (IA):** Convertir la preferencia a un timestamp ISO.\n\t\t\t* (Ej: Usuario dice \"mañana 10am\" -> IA calcula \"mañana\" como 7 de nov -> \"2025-11-07T10:00:00-05:00\")\n\t\t\t* (Ej: Usuario dice \"opción 2\" -> IA toma el `iso` del slot 2 de la memoria)\n\t\t2. **Acción (Tool 2 - CalcularEspacios...):** Volver a llamar a la tool con:\n\t\t\t```json\n\t\t\t{\n\t\t\t \"occupiedSlots\": [... (resultado de Tool 1) ...],\n\t\t\t \"duration_minutes\": 35,\n\t\t\t \"preference_filter\": \"[El ISO string calculado CON -05:00]\"\n\t\t\t}\n\t\t\t```\n\t\t3. **Acción (IA - Reacción):**\n\t\t\t* **Si la tool devuelve el slot validado (array con 1 item):**\n\t\t\t\t* `tag: \"confirm_appointment_prompt\"`\n\t\t\t\t* `body: \"¡Perfecto! Encontré este espacio: [texto del slot]. ¿Confirmo tu cita para esta fecha y hora? Responde SÍ.\"`\n\t\t\t* **Si la tool devuelve un array vacío (slot no válido):**\n\t\t\t\t* `tag: \"slot_not_valid\"`\n\t\t\t\t* `body: \"Lo siento, esa hora ([preferencia del usuario]) no está disponible o está fuera de nuestro horario. ¿Te gustaría intentar con otra hora?\"`\n\n\t\t**B) Pedir \"VER MÁS\" (Solo si se mostró CASO B)**\n\t\t1. **Acción (IA):** Recordar cuántos slots ya mostró (ej: 6).\n\t\t2. **Acción (Tool 2 - CalcularEspacios...):** Volver a llamar a la tool con:\n\t\t\t```json\n\t\t\t{\n\t\t\t \"occupiedSlots\": [... (de la Tool 1) ...],\n\t\t\t \"duration_minutes\": 35,\n\t\t\t \"slots_offset\": 6 // (O 12, 18, etc.)\n\t\t\t}\n\t\t\t```\n\t\t3. **Acción (IA - Reacción):**\n\t\t\t* Recibe los 6 *nuevos* slots y los presenta (Vuelve a \"ACCIÓN 4 - CASO B\").\n\n\t\t---\n\t\t5.2. SI el usuario responde a \"CASO C\" (Doctor Lleno)\n\n\t\t**A) Dice \"Elegir otro doctor\"**\n\t\t1. **Acción (IA):** Volver al flujo `prompt_doctor_menu`.\n\n\t\t---\n\t\t5.3. SI el usuario responde \"SÍ\" al tag \"confirm_appointment_prompt\"\n\n\t\t1. **Acción (IA):** Calcular el `end_time` (start_time de la cita confirmada + 35 minutos).\n\t\t2. **Acción (Tool 3 - insert_appointment):** Llamar a la tool de inserción con todos los datos:\n\t\t\t```json\n\t\t\t{\n\t\t\t \"customer_id\": [ID del cliente],\n\t\t\t \"doctor_id\": [ID del doctor],\n\t\t\t \"service_id\": [ID del servicio],\n\t\t\t \"start_time\": \"[ISO de la cita]\",\n\t\t\t \"end_time\": \"[ISO de fin calculado]\"\n\t\t\t}\n\t\t\t```\n\t\t3. **Acción (IA - Reacción):**\n\t\t\t* **Si la Tool 3 devuelve éxito:**\n\t\t\t\t* `tag: \"appointment_created\"`\n\t\t\t\t* `body: \"¡Excelente! Tu cita (ID#[id de la cita generado al crear el registro de la db]) ha sido agendada con éxito para el [texto de la cita] con el Dr(a). [nombreDoctor]. ¡Te esperamos! 👋\"`\n\t\t\t* **Si la Tool 3 devuelve

error:**\n\t\t\t\t* `tag: \"error_db\"`\n\t\t\t\t* `body: \"Oh, vaya. Tuve un problema al intentar guardar tu cita. Por favor, ¿podrías intentarlo de nuevo en un momento?\"`\n\n- (IMPORTANTE) Si el usuario no existe y no esta en memoria solo es usar el texto de \"MSG_BIENVENIDA_NUEVO\" y no hacer nada mas si no enviarle el mensaje al cliente\t \n\t\t \n- Opción 2 — Cancelar una cita\n Asumiendo que el usuario ya está validado y su \"identification\" está en memoria.\n\n 1. Acción 1 (IA):\n   * tag: \"prompt_for_appointment_id\"\n   * body: (Usar MSG_PEDIR_ID_CITA)\n\n 2. Acción 2 (Usuario):\n   * Responde con un ID de cita (ej: \"987\").\n\n 3. Acción 3 (Tool 1 - get_appointment_details):\n   * La IA DEBE llamar a la tool get_appointment_details.\n   * Parámetros: El [ID_CITA_USUARIO] (987) y el [ID_CLIENTE_MEMORIA] (ej: 1005105966).\n\n 4. Acción 4 (IA - Reacción a Tool 1):\n   * CASO A: (Éxito - Cita encontrada)\n   - La Tool devuelve los detalles (id, start_time, doctor_name).\n   - La IA DEBE guardar estos detalles en memoria.\n   - La IA DEBE formatear el start_time a un texto legible (ej: \"viernes 7 de nov a las 10:00am\").\n   - tag: \"confirm_cancellation_prompt\"\n   - body: (Construir MSG_CONFIRMAR_CANCELACION con los detalles: \"Encontré esta cita: 🗓️ [viernes 7 de nov a las 10:00am] con el Dr(a). [nombreDoctor]. ¿Estás seguro/a...\")\n   \n   * CASO B: (Fallo - Cita no encontrada)\n   - La Tool devuelve un array vacío [].\n   - (Esto significa que el ID es incorrecto, ya está cancelada, o no pertenece al usuario).\n   - tag: \"appointment_not_found\"\n   - body: (Construir MSG_CITA_NO_ENCONTRADA: \"Lo siento, no pude encontrar una cita activa con el número #987 asociada a tu identificación...\")\n\n 5. Acción 5 (Usuario - Responde al 'confirm_cancellation_prompt'):\n\n   * A) Si el usuario responde \"SÍ\":\n   - Acción 6 (Tool 2 - cancel_appointment):\n   - La IA DEBE llamar a la tool cancel_appointment con el [ID_CITA_A_CANCELAR] (987) de la memoria.\n   - Acción 7 (IA - Reacción a Tool 2):\n   - Si la Tool 2 devuelve éxito:\n   - tag: \"appointment_cancelled\"\n   - body: (Construir MSG_CANCELACION_EXITOSA con los detalles de la cita)\n   - Si la Tool 2 devuelve error:\n   - tag: \"error_db\"\n   - body: (Usar MSG_CANCELACION_ERROR_DB)\n\n   * B) Si el usuario responde \"NO\":\n   - Acción 6 (IA):\n   - tag: \"cancellation_aborted\"\n   - body: (Usar MSG_CANCELACION_ABORTADA)\n\t\t \n\t\t \n- Opción 3 — Consultar \"SERVICIOS\" y horarios\n Este flujo se activa cuando el usuario elige \"3\" en el menú principal.\n\n 1. Acción 1 (Tool 1 - select_servicios):\n   - La IA DEBE llamar a la tool select_servicios (que ahora trae id, descripción, detalle).\n   - (!IMPORTANTE)La IA DEBE guardar esta lista completa en memoria.\n\n 2. Acción 2 (IA - Muestra el menú):\n   - La IA DEBE formatear la lista de servicios (ej: \"1. Limpieza Facial\\n2. Consulta...\")\n   - tag: \"prompt_service_details\"\n   - body: (Construir MSG_SERVICIOS_HEADER, insertando la lista de servicios y el horario normal)\n\n 3. Acción 3 (Usuario - Responde al prompt):\n   - El usuario puede escribir un número (ej: \"1\"), el nombre (ej: \"limpieza\") o \"NO\".\n\n 4. Acción 4 (IA - Reacciona a la respuesta):\n   \n   * CASO A: (Usuario elige un servicio por número o nombre)\n   - La IA DEBE buscar en la lista que tiene en memoria el servicio correspondiente.\n   - Si encuentra una coincidencia (ej: \"2\" -> {descripción: \"Toxina Botulínica (Botox) — Reducción de arrugas dinámicas.\", detalle: \"Ideal para suavizar y prevenir líneas de expresión dinámicas (como las de la frente, entrecejo y 'patas de gallo'). Es un tratamiento rápido y seguro que relaja los músculos faciales, dándote un aspecto más descansado y juvenil.\"}):\n   - tag: \"service_detail_sent\"\n   - body: (Construir MSG_SERVICIO_DETALLE, rellenando el [Nombre del Servicio] y el [Detalle del servicio])\n\t\t (IMPORTANTE)\n\t\t [Nombre del servicio] > Es el nombre del servicio que eligio el usuario apra consultar\n\t\t [Detalle del servicio] > Es la columna

\"detalle\" exacta tal cual que tienes en memoria de ese servicio de la consulta de la tool(select_servicios)\n\t\t Ejemplo:\n\t\t [Nombre del Servicio] > Toxina Botulínica (Botox) — Reducción de arrugas dinámicas.\n\t\t [Detalle del servicio]) > Ideal para suavizar y prevenir líneas de expresión dinámicas (como las de la frente, entrecejo y 'patas de gallo'). Es un tratamiento rápido y seguro que relaja los músculos faciales, dándote un aspecto más descansado y juvenil.\n\t\t \n - (El flujo regresa a la Acción 3, esperando la siguiente respuesta del usuario)\n\n\n * CASO B: (Usuario escribe \"NO\" o \"volver\")\n - La IA DEBE volver al menú principal.\n - tag: \"initial_greeting\"\n - body: (Usar MSG_MENU_PRINCIPAL)\n\n * CASO C: (No entiende la respuesta)\n - La IA DEBE usar un fallback gentil.\n - tag: \"fallback_service_details\"\n - body: \"No entendí esa opción. Por favor, escribe el número del servicio que deseas consultar, o escribe 'NO' para volver.\"\n \n ### UMBRALES, Fallbacks y Clarificaciones\n\n- Umbrales:\n * confidence >= 0.75 → Aceptar mapeo automático (ej. \"quiero una cita\" -> Opción 1).\n * 0.40 <= confidence < 0.75 → Pedir confirmación breve.\n * confidence < 0.40 → Fallback directo.\n * max_fallbacks_per_session = 3 → Ofrecer humano.\n\n- Mensajes exactos de fallback/clarificación:\n * Confianza media: usar el mensaje MSG_FALLBACK_MEDIO \n * Confianza baja / No entiendo: usar el mensaje MSG_FALLBACK_BAJO \n * Tras 2 reintentos: usar el mensaje MSG_FALLBACK_REINTENTO_2 \n * Máximo de reintentos: usar el mensaje MSG_FALLBACK_MAX\n \n### JSON EJEMPLOS (¡Importante!) \n\n- Primera interacción:\n{\n\t\"tag\": \"initial_greeting\",\n\t\"body\": \"¡Hola! 👋 Bienvenido a Pielsens (Dermatología y medicina estética). Soy tu asistente virtual y estoy aquí para ayudarte. Por favor escribe el número de la opción que deseas:\\n\\n🗓️ 1) Agendar una cita — Te guiaré para elegir servicio, fecha y registrar tu cita.\\n❌ 2) Cancelar una cita — Buscaré tu cita activa para cancelarla.\\n**i** 3) Consultar servicios y horarios — Te muestro nuestros servicios y horario de atención.\",\n\t\"identification\": \"\",\n\t\"user_exists\": false\n}\n\n- Mensaje pedir identificación:\n{\n\t\"tag\": \"ask_for_id\",\n\t\"body\": \"Para continuar, por favor escribe tu número de identificación sin puntos ni caracteres (ej: 12345678).\",\n\t\"identification\": \"\",\n\t\"user_exists\": false\n}\n\n- Mensaje de confirmación de identificación del usuario:\n{\n\t\"tag\": \"id_confirm_prompt\",\n\t\"body\": \"¿Es correcto este número de identificación: 1005105966?\\nResponde SI para confirmar o NO para cambiarlo.\",\n\t\"identification\": \"\",\n\t\"user_exists\": false\n}\n\n- Usuario confirma \"SI\", existe (Nombre de usuario), y quería Agendar. (Se consultan servicios).\n{\n\"tag\": \"customer_found\",\n\"body\": \"Hola Juan Andres 👋\\nEn que especialidad deseas agendar tu cita? 👩‍⚕️ \\nEstos son nuestros servicios:\\n1. Consulta de Dermatología\\n2. Limpieza Facial Profunda\\n3. Tratamiento Láser .....\",\n\"identification\": \"1005105966\",\n\"user_exists\": true\n}\n\n- Usuario confirma \"SI\", NO existe (ID del usuario), y quería Agendar. (Se consultan servicios).\n{\n\t\"tag\": \"customer_nofound\",\n\t\"body\": \"¡Perfecto! Veo que es tu primera cita con nosotros. ¡Bienvenido/a! 👋\",\n\t\"identification\": \"\",\n\t\"user_exists\": false\n}"
      }
    },
    "type": "@n8n/n8n-nodes-langchain.agent",
    "typeVersion": 2.2,
    "position": [
      160,

```
    -560
   ],
   "id": "a4a2df91-f79c-4589-a85e-fd292434f2b3",
   "name": "Pepito - AI Agent"
  },
  {
   "parameters": {
    "sessionIdType": "customKey",
    "sessionKey": "={{ $json.contacts[0].wa_id }}",
    "contextWindowLength": 500
   },
   "type": "@n8n/n8n-nodes-langchain.memoryPostgresChat",
   "typeVersion": 1.3,
   "position": [
    -192,
    -176
   ],
   "id": "edadb01d-380b-4570-aaa2-1983cf8b257a",
   "name": "Postgres Chat Memory",
   "credentials": {
    "postgres": {
     "id": "NCOcVSkJXNnml3Hc",
     "name": "Postgres - Juan Andres Pinzon Martinez"
    }
   }
  },
  {
   "parameters": {
    "options": {}
   },
   "type": "@n8n/n8n-nodes-langchain.lmChatGoogleGemini",
   "typeVersion": 1,
   "position": [
    -192,
    -352
   ],
   "id": "ea68fcdc-a3ed-49a5-a45d-54122cca94a8",
   "name": "Google Gemini Chat Model",
   "credentials": {
    "googlePalmApi": {
     "id": "5K7NVkyGJff2lpco",
     "name": "Juan Andres Pinzxon Martinez - GOOGLE STUDIO"
    }
   }
  },
  {
```

```
    "parameters": {
      "operation": "sendAndWait",
      "phoneNumberId": "107431422341377",
      "recipientPhoneNumber": "={{ $('Msg recibidos').item.json.contacts[0].wa_id }}",
      "message": "Para crear tu ficha de paciente y continuar, necesitaré un par de datos
más.\nPor favor, entra al link y llena la información.",
      "responseType": "customForm",
      "formFields": {
        "values": [
          {
            "fieldLabel": "Cedula de ciudadania",
            "placeholder": "Cedula de ciudadania",
            "requiredField": true
          },
          {
            "fieldLabel": "Nombres",
            "requiredField": true
          },
          {
            "fieldLabel": "Apellidos",
            "requiredField": true
          },
          {
            "fieldLabel": "Correo electronico",
            "fieldType": "email",
            "requiredField": true
          },
          {
            "fieldLabel": "Terminos y condiciones",
            "fieldType": "radio",
            "fieldOptions": {
              "values": [
                {
                  "option": "Aceptar"
                }
              ]
            },
            "requiredField": true
          }
        ]
      },
      "options": {}
    },
    "type": "n8n-nodes-base.whatsApp",
    "typeVersion": 1.1,
    "position": [
```

```
    1136,
    -672
   ],
   "id": "5727675f-f474-4ca4-8371-6b4e9c3c06d6",
   "name": "Registro de usuario - link",
   "webhookId": "87135c72-8381-45a5-96a6-dff9a8753f4d",
   "credentials": {
    "whatsAppApi": {
     "id": "S6vn9TrfS547uYON",
     "name": "WhatsApp Msg - Juan Andres Pinzon Martinez"
    }
   }
  },
  {
   "parameters": {
    "operation": "select",
    "schema": {
     "__rl": true,
     "mode": "list",
     "value": "public"
    },
    "table": {
     "__rl": true,
     "value": "customers",
     "mode": "list",
     "cachedResultName": "customers"
    },
    "limit": 1,
    "where": {
     "values": [
      {
       "column": "identification",
       "value": "={{ $json.data['Cedula de ciudadania'] }}"
      }
     ]
    },
    "combineConditions": "OR",
    "options": {}
   },
   "type": "n8n-nodes-base.postgres",
   "typeVersion": 2.6,
   "position": [
    1376,
    -672
   ],
   "id": "34cd8852-2c6f-43ce-a67b-fe8345ac0423",
```

```
    "name": "Busca si el usuario se registro previamente",
    "alwaysOutputData": true,
    "credentials": {
     "postgres": {
      "id": "NCOcVSkJXNnml3Hc",
      "name": "Postgres - Juan Andres Pinzon Martinez"
     }
    }
   },
   {
    "parameters": {
     "schema": {
      "__rl": true,
      "mode": "list",
      "value": "public"
     },
     "table": {
      "__rl": true,
      "value": "customers",
      "mode": "list",
      "cachedResultName": "customers"
     },
     "columns": {
      "mappingMode": "defineBelow",
      "value": {
       "accept_terms": "={{ $('Registro de usuario - link').item.json.data['Terminos y
condiciones']  == 'Aceptar' ? true : false}}",
       "identification": "={{ $('Registro de usuario - link').item.json.data['Cedula de
ciudadania'] }}",
       "name": "={{ $('Registro de usuario - link').item.json.data.Nombres }}",
       "lastnames": "={{ $('Registro de usuario - link').item.json.data.Apellidos }}",
       "phone": "={{ $('Msg recibidos').item.json.contacts[0].wa_id }}",
       "email": "={{ $('Registro de usuario - link').item.json.data['Correo electronico'] }}"
      },
      "matchingColumns": [],
      "schema": [
       {
        "id": "identification",
        "displayName": "identification",
        "required": false,
        "defaultMatch": false,
        "display": true,
        "type": "number",
        "canBeUsedToMatch": true
       },
       {
```

```
  "id": "created_at",
  "displayName": "created_at",
  "required": false,
  "defaultMatch": false,
  "display": true,
  "type": "dateTime",
  "canBeUsedToMatch": true,
  "removed": true
},
{
  "id": "name",
  "displayName": "name",
  "required": false,
  "defaultMatch": false,
  "display": true,
  "type": "string",
  "canBeUsedToMatch": true
},
{
  "id": "lastnames",
  "displayName": "lastnames",
  "required": false,
  "defaultMatch": false,
  "display": true,
  "type": "string",
  "canBeUsedToMatch": true
},
{
  "id": "phone",
  "displayName": "phone",
  "required": false,
  "defaultMatch": false,
  "display": true,
  "type": "string",
  "canBeUsedToMatch": true
},
{
  "id": "email",
  "displayName": "email",
  "required": false,
  "defaultMatch": false,
  "display": true,
  "type": "string",
  "canBeUsedToMatch": true
},
{
```

```
          "id": "accept_terms",
          "displayName": "accept_terms",
          "required": false,
          "defaultMatch": false,
          "display": true,
          "type": "boolean",
          "canBeUsedToMatch": true
        }
      ],
      "attemptToConvertTypes": false,
      "convertFieldsToString": false
    },
    "options": {}
  },
  "type": "n8n-nodes-base.postgres",
  "typeVersion": 2.6,
  "position": [
    1920,
    -768
  ],
  "id": "16b9481c-c64d-47ea-9e35-38b442e42c02",
  "name": "Agregar nuevo usuario",
  "alwaysOutputData": true,
  "credentials": {
    "postgres": {
      "id": "NCOcVSkJXNnml3Hc",
      "name": "Postgres - Juan Andres Pinzon Martinez"
    }
  }
},
{
  "parameters": {
    "operation": "send",
    "phoneNumberId": "107431422341377",
    "recipientPhoneNumber": "={{ $('Msg recibidos').item.json.contacts[0].wa_id }}",
    "textBody": "={{ $json.output.body }}",
    "additionalFields": {}
  },
  "type": "n8n-nodes-base.whatsApp",
  "typeVersion": 1.1,
  "position": [
    496,
    -704
  ],
  "id": "607f19cb-11fe-4727-a72e-e93b284a2267",
  "name": "Msg Respuesta Agente",
```

```
  "webhookId": "825aec26-dff6-454a-b751-a8b1597371c3",
  "credentials": {
    "whatsAppApi": {
      "id": "S6vn9TrfS547uYON",
      "name": "WhatsApp Msg - Juan Andres Pinzon Martinez"
    }
  }
},
{
  "parameters": {
    "conditions": {
      "options": {
        "caseSensitive": true,
        "leftValue": "",
        "typeValidation": "loose",
        "version": 2
      },
      "conditions": [
        {
          "id": "ae30b10e-6b8c-4d8a-93a0-b0e4787a54cf",
          "leftValue": "={{ $('Busca si el usuario se registro previamente')}}",
          "rightValue": "",
          "operator": {
            "type": "string",
            "operation": "notEmpty",
            "singleValue": true
          }
        }
      ],
      "combinator": "and"
    },
    "looseTypeValidation": true,
    "options": {}
  },
  "type": "n8n-nodes-base.if",
  "typeVersion": 2.2,
  "position": [
    1616,
    -672
  ],
  "id": "4280647c-d05a-4420-b871-2f17f8be881b",
  "name": "existe_usuario"
},
{
  "parameters": {
    "conditions": {
```

```
      "options": {
        "caseSensitive": true,
        "leftValue": "",
        "typeValidation": "strict",
        "version": 2
      },
      "conditions": [
        {
          "id": "b8b3827b-49a4-4456-b0b6-84c3ddd61806",
          "leftValue": "={{ $json.output.tag }}",
          "rightValue": "customer_nofound",
          "operator": {
            "type": "string",
            "operation": "equals",
            "name": "filter.operator.equals"
          }
        }
      ],
      "combinator": "and"
    },
    "options": {}
  },
  "type": "n8n-nodes-base.if",
  "typeVersion": 2.2,
  "position": [
    880,
    -560
  ],
  "id": "f8caa3b8-fd04-46b3-bd50-968475db3703",
  "name": "es_NuevoUsuario"
},
{
  "parameters": {
    "operation": "executeQuery",
    "query": "{{ $fromAI('Table') }}",
    "options": {}
  },
  "type": "n8n-nodes-base.postgresTool",
  "typeVersion": 2.6,
  "position": [
    480,
    -256
  ],
  "id": "fab5cbc0-db50-4b71-a4c8-554685ae0b76",
  "name": "ObtenerDisponibilidad",
  "credentials": {
```

```
    "postgres": {
      "id": "NCOcVSkJXNnml3Hc",
      "name": "Postgres - Juan Andres Pinzon Martinez"
    }
  }
},
{
  "parameters": {
    "operation": "executeQuery",
    "query": "{{ $fromAI('Table') }}",
    "options": {}
  },
  "type": "n8n-nodes-base.postgresTool",
  "typeVersion": 2.6,
  "position": [
    304,
    -256
  ],
  "id": "8feea656-4032-45f6-b345-9f382431d7b0",
  "name": "select_doctores_xservicio",
  "notesInFlow": false,
  "credentials": {
    "postgres": {
      "id": "NCOcVSkJXNnml3Hc",
      "name": "Postgres - Juan Andres Pinzon Martinez"
    }
  }
},
{
  "parameters": {
    "operation": "select",
    "schema": {
      "__rl": true,
      "mode": "list",
      "value": "public"
    },
    "table": {
      "__rl": true,
      "value": "services",
      "mode": "list",
      "cachedResultName": "services"
    },
    "returnAll": true,
    "options": {}
  },
  "type": "n8n-nodes-base.postgresTool",
```

```
    "typeVersion": 2.6,
    "position": [
      160,
      -256
    ],
    "id": "595fc024-2c53-4f05-97be-fc784d250b86",
    "name": "select_servicios",
    "credentials": {
      "postgres": {
        "id": "NCOcVSkJXNnml3Hc",
        "name": "Postgres - Juan Andres Pinzon Martinez"
      }
    }
  },
  {
    "parameters": {
      "jsCode": "//
==========================================================\n//
NODO JS SIN LUXON (CALCULAR ESPACIOS DISPONIBLES) v2.2\n// CORREGIDO:
Eliminada la matemática de offset manual que causaba el error de 5 horas.\n//
==========================================================\n\n//
--- CONFIGURACIÓN ---\nconst TIMEZONE = \"America/Bogota\"; // Zona horaria
correcta\nconst BUFFER_MINUTES = 10;\nconst SLOTS_TO_FIND = 6;\nconst
SEARCH_LIMIT_DAYS = 30;\nconst MINUTE_MS = 60 * 1000;\nconst HOUR_MS = 60 *
MINUTE_MS;\nconst DAY_MS = 24 * HOUR_MS;\nconst BUFFER_MS =
BUFFER_MINUTES * MINUTE_MS;\n\n// Horario de *asignación* de citas\nconst
businessHours = {\n  1: { start: 8, end: 17 }, // Lunes 8:00 - 17:00 (5:00 PM)\n  2: { start: 8, end:
17 }, // Martes\n  3: { start: 8, end: 17 }, // Miércoles\n  4: { start: 8, end: 17 }, // Jueves\n  5: {
start: 8, end: 17 }, // Viernes\n  6: { start: 9, end: 13 }, // Sábado 9:00 - 13:00 (1:00 PM)\n  7:
null, // Domingo\n};\n\n// --- FUNCIONES AUXILIARES (CORREGIDAS) ---\n\n/**\n *
[CORREGIDO] Convierte un string ISO (de la DB o IA) a un timestamp UTC.\n * Ya no aplica
offset manual.\n */\nfunction toBogotaMsFromIso(iso) {\n  return new
Date(iso).getTime();\n}\n\n/**\n * [CORREGIDO] Obtiene el timestamp UTC actual.\n * Ya no
aplica offset manual.\n */\nfunction nowBogotaMs() {\n  return Date.now();\n}\n\n/**\n *
[CORREGIDO] Formatea un timestamp UTC a un texto legible en zona Bogotá.\n */\nfunction
formatBogotaTextFromMs(utcMs) {\n  const d = new Date(utcMs);\n  const options = {\n
weekday: \"long\", day: \"numeric\", month: \"long\",\n    hour: \"numeric\", minute: \"2-digit\",
hour12: true,\n    timeZone: TIMEZONE, // JavaScript usará esto para convertir el UTC a la hora
local\n  };\n  return new Intl.DateTimeFormat(\"es-CO\", options).format(d);\n}\n\n/**\n *
[CORREGIDO] Obtiene el timestamp UTC del inicio del día (00:00) en Bogotá.\n * Esto es
crucial. 00:00 en Bogotá es 05:00 en UTC.\n */\nfunction getStartOfDayBogotaMs(ms_utc) {\n
// Usamos Intl.DateTimeFormat para saber qué día era en Bogotá\n  const parts = new
Intl.DateTimeFormat('en-US', {\n    timeZone: \"America/Bogota\",\n    year: 'numeric',\n
month: 'numeric',\n    day: 'numeric'\n  }).formatToParts(ms_utc).reduce((acc, part) => {\n
acc[part.type] = part.value;\n    return acc;\n  }, {});\n  // parts = { month: '11', day: '6', year:
'2025' }\n\n  // Creamos el timestamp UTC para las 00:00 *en Bogotá*, que es 05:00 UTC\n
```

```
return Date.UTC(parts.year, parts.month - 1, parts.day, 5, 0, 0);\n}\n\n// --- INICIO DE LA LÓGICA PRINCIPAL ---\n\n// 1. OBTENER DATOS DE ENTRADA\n// ==========================================\nif (!query) throw new Error(\"No se recibió la variable 'query' de la IA.\");\nif (!query || typeof query !== 'string') {\n  throw new Error(\"No se encontró 'query' o no es un string.\");\n}\n\nlet parsedData;\ntry {\n  parsedData = JSON.parse(query);\n} catch (e) {\n  throw new Error(\"Error al parsear el JSON de 'query': \" + e.message);\n}\n\nconst occupiedSlotsInput = parsedData.occupiedSlots;\nif (!occupiedSlotsInput || !Array.isArray(occupiedSlotsInput)) {\n  throw new Error(\"No se encontró 'occupiedSlots' en el JSON parseado.\");\n}\n\nconst serviceDurationMinutes = parsedData.duration_minutes;\nif (!serviceDurationMinutes) {\n  throw new Error(\"No se encontró 'duration_minutes' en el JSON parseado.\");\n}\nconst serviceDurationMs = serviceDurationMinutes * MINUTE_MS;\n\n// === NUEVOS PARÁMETROS OPCIONALES ===\nconst slotsOffset = parsedData.slots_offset || 0;\n// [CORREGIDO] Usamos la función corregida\nconst preferenceFilterMs = parsedData.preference_filter ? toBogotaMsFromIso(parsedData.preference_filter) : null;\n\n// === LÓGICA \"SI NO TIENE NADA\" (CASO A) ===\nif (occupiedSlotsInput.length === 0 && !preferenceFilterMs) {\n  return JSON.stringify({ \n    availableSlots: [], \n    isCompletelyFree: true \n  });\n}\n\n// [CORREGIDO] Usamos la función corregida\nconst occupiedAppointments = occupiedSlotsInput.map(slot => ({\n  startMs: toBogotaMsFromIso(slot.start_time),\n  endMs: toBogotaMsFromIso(slot.end_time),\n}));\n\n// 2. LÓGICA DE BÚSQUEDA (CASO B y C)\n// ==========================================\nconst availableSlots = [];\nlet skippedSlots = 0;\n// [CORREGIDO] Usamos la función corregida\nlet searchTimeMs = nowBogotaMs(); \n\nif (preferenceFilterMs && preferenceFilterMs > searchTimeMs) {\n  searchTimeMs = preferenceFilterMs;\n}\n\nfor (let i = 0; i < SEARCH_LIMIT_DAYS; i++) {\n  // [CORREGIDO] Usamos la función corregida\n  const searchDateUTC = new Date(searchTimeMs);\n  const weekday = searchDateUTC.getUTCDay();\n  const dayOfWeek = weekday === 0 ? 7 : weekday;\n  const workingHours = businessHours[dayOfWeek];\n\n  if (!workingHours) {\n    searchTimeMs = getStartOfDayBogotaMs(searchTimeMs + DAY_MS);\n    continue;\n  }\n\n  // [CORREGIDO] Usamos la función corregida\n  const startOfDayMs = getStartOfDayBogotaMs(searchTimeMs);\n  // Esta lógica ahora es correcta:\n  // (Nov 7 @ 5:00 UTC) + (8 horas) = Nov 7 @ 13:00 UTC (8:00 AM Bogotá)\n  const dayStartMs = startOfDayMs + (workingHours.start * HOUR_MS);\n  // (Nov 7 @ 5:00 UTC) + (17 horas) = Nov 7 @ 22:00 UTC (5:00 PM Bogotá)\n  const dayEndMs = startOfDayMs + (workingHours.end * HOUR_MS);\n\n  // [CORREGIDO] Usamos la función corregida\n  let slotSearchMs = Math.max(dayStartMs, nowBogotaMs());\n  if (preferenceFilterMs && preferenceFilterMs > slotSearchMs) {\n    slotSearchMs = preferenceFilterMs;\n  }\n\n  while (slotSearchMs < dayEndMs) {\n    const potentialEndMs = slotSearchMs + serviceDurationMs;\n    if (potentialEndMs > dayEndMs) {\n      break;\n    }\n\n    const isConflict = occupiedAppointments.some(occupied => {\n      return slotSearchMs < occupied.endMs && potentialEndMs > occupied.startMs;\n    });\n\n    if (!isConflict) {\n      if (skippedSlots < slotsOffset) {\n        skippedSlots++; \n      } else {\n        availableSlots.push({\n// [CORREGIDO] Usamos la función corregida\n          text: formatBogotaTextFromMs(slotSearchMs),\n          iso: new Date(slotSearchMs).toISOString(), // El ISO siempre es UTC\n        });\n      }\n\n      if (preferenceFilterMs) {\n        return JSON.stringify({ availableSlots: availableSlots, isCompletelyFree: false });\n      }\n      \n      if (availableSlots.length >= SLOTS_TO_FIND) {\n        return JSON.stringify({ availableSlots:
```

availableSlots, isCompletelyFree: false });\n   }\n   slotSearchMs = potentialEndMs + BUFFER_MS;\n   } else {\n   slotSearchMs = slotSearchMs + (10 * MINUTE_MS);\n   }\n }\n\n if (preferenceFilterMs) {\n   break;\n }\n\n // [CORREGIDO] Usamos la función corregida\n searchTimeMs = getStartOfDayBogotaMs(searchTimeMs + DAY_MS);\n if (availableSlots.length >= SLOTS_TO_FIND) {\n   break;\n }\n}\n\n// 3. DEVOLVER RESULTADOS\n// ==========\nreturn JSON.stringify({ availableSlots: availableSlots, isCompletelyFree: false });"
      },
      "type": "@n8n/n8n-nodes-langchain.toolCode",
      "typeVersion": 1.3,
      "position": [
        480,
        -96
      ],
      "id": "e6825dd1-dd0f-4fe5-9890-cb6af7aff283",
      "name": "CalcularEspaciosDisponibles"
    },
    {
      "parameters": {
        "schema": {
          "__rl": true,
          "mode": "list",
          "value": "public"
        },
        "table": {
          "__rl": true,
          "value": "appointments",
          "mode": "list",
          "cachedResultName": "appointments"
        },
        "columns": {
          "mappingMode": "defineBelow",
          "value": {
            "isActive": true,
            "customer_id": "={{ /*n8n-auto-generated-fromAI-override*/ $fromAI('customer_id', ``, 'number') }}",
            "doctor_id": "={{ /*n8n-auto-generated-fromAI-override*/ $fromAI('doctor_id', ``, 'number') }}",
            "service_id": "={{ /*n8n-auto-generated-fromAI-override*/ $fromAI('service_id', ``, 'number') }}",
            "start_time": "={{ /*n8n-auto-generated-fromAI-override*/ $fromAI('start_time', ``, 'string') }}",
            "end_time": "={{ /*n8n-auto-generated-fromAI-override*/ $fromAI('end_time', ``, 'string') }}"
          },
          "matchingColumns": [

```
  "id"
],
"schema": [
  {
    "id": "id",
    "displayName": "id",
    "required": false,
    "defaultMatch": true,
    "display": true,
    "type": "number",
    "canBeUsedToMatch": true,
    "removed": true
  },
  {
    "id": "created_at",
    "displayName": "created_at",
    "required": false,
    "defaultMatch": false,
    "display": true,
    "type": "dateTime",
    "canBeUsedToMatch": true,
    "removed": true
  },
  {
    "id": "customer_id",
    "displayName": "customer_id",
    "required": false,
    "defaultMatch": false,
    "display": true,
    "type": "number",
    "canBeUsedToMatch": true
  },
  {
    "id": "doctor_id",
    "displayName": "doctor_id",
    "required": false,
    "defaultMatch": false,
    "display": true,
    "type": "number",
    "canBeUsedToMatch": true
  },
  {
    "id": "service_id",
    "displayName": "service_id",
    "required": false,
    "defaultMatch": false,
```

```
        "display": true,
        "type": "number",
        "canBeUsedToMatch": true
      },
      {
        "id": "start_time",
        "displayName": "start_time",
        "required": false,
        "defaultMatch": false,
        "display": true,
        "type": "dateTime",
        "canBeUsedToMatch": true
      },
      {
        "id": "end_time",
        "displayName": "end_time",
        "required": false,
        "defaultMatch": false,
        "display": true,
        "type": "dateTime",
        "canBeUsedToMatch": true
      },
      {
        "id": "isActive",
        "displayName": "isActive",
        "required": false,
        "defaultMatch": false,
        "display": true,
        "type": "boolean",
        "canBeUsedToMatch": true
      }
    ],
    "attemptToConvertTypes": false,
    "convertFieldsToString": false
  },
  "options": {}
},
"type": "n8n-nodes-base.postgresTool",
"typeVersion": 2.6,
"position": [
  656,
  -256
],
"id": "6b51fd21-59e2-49a6-9b9a-4f5e48c9d6c0",
"name": "insert_appointments",
"credentials": {
```

```
    "postgres": {
      "id": "NCOcVSkJXNnml3Hc",
      "name": "Postgres - Juan Andres Pinzon Martinez"
    }
  }
},
{
  "parameters": {
    "operation": "executeQuery",
    "query": "{{ $fromAI('Table') }}",
    "options": {}
  },
  "type": "n8n-nodes-base.postgresTool",
  "typeVersion": 2.6,
  "position": [
    16,
    -64
  ],
  "id": "36daa763-5c7a-4cf0-b171-6489e1a0022a",
  "name": "get_appointment_details",
  "notesInFlow": false,
  "credentials": {
    "postgres": {
      "id": "NCOcVSkJXNnml3Hc",
      "name": "Postgres - Juan Andres Pinzon Martinez"
    }
  }
},
{
  "parameters": {
    "operation": "executeQuery",
    "query": "{{ $fromAI('Table') }}",
    "options": {}
  },
  "type": "n8n-nodes-base.postgresTool",
  "typeVersion": 2.6,
  "position": [
    208,
    -64
  ],
  "id": "ccb4b52a-cdc0-4970-8ebe-7a2e08e4eb1a",
  "name": "cancel_appointment",
  "notesInFlow": false,
  "credentials": {
    "postgres": {
      "id": "NCOcVSkJXNnml3Hc",
```

```
      "name": "Postgres - Juan Andres Pinzon Martinez"
     }
    }
   }
],
"pinData": {},
"connections": {
  "Structured Output Parser": {
    "ai_outputParser": [
      [
        {
          "node": "Pepito - AI Agent",
          "type": "ai_outputParser",
          "index": 0
        }
      ]
    ]
  },
  "Consulta si el usuario existe": {
    "ai_tool": [
      [
        {
          "node": "Pepito - AI Agent",
          "type": "ai_tool",
          "index": 0
        }
      ]
    ]
  },
  "Msg recibidos": {
    "main": [
      [
        {
          "node": "Pepito - AI Agent",
          "type": "main",
          "index": 0
        }
      ]
    ]
  },
  "Pepito - AI Agent": {
    "main": [
      [
        {
          "node": "Msg Respuesta Agente",
          "type": "main",
```

```
        "index": 0
      },
      {
        "node": "es_NuevoUsuario",
        "type": "main",
        "index": 0
      }
    ]
  ]
},
"Postgres Chat Memory": {
  "ai_memory": [
    [
      {
        "node": "Pepito - AI Agent",
        "type": "ai_memory",
        "index": 0
      }
    ]
  ]
},
"Google Gemini Chat Model": {
  "ai_languageModel": [
    [
      {
        "node": "Pepito - AI Agent",
        "type": "ai_languageModel",
        "index": 0
      }
    ]
  ]
},
"Registro de usuario - link": {
  "main": [
    [
      {
        "node": "Busca si el usuario se registro previamente",
        "type": "main",
        "index": 0
      }
    ]
  ]
},
"Busca si el usuario se registro previamente": {
  "main": [
    [
```

```
      {
       "node": "existe_usuario",
       "type": "main",
       "index": 0
      }
     ]
    ]
  },
  "existe_usuario": {
    "main": [
     [
      {
       "node": "Agregar nuevo usuario",
       "type": "main",
       "index": 0
      }
     ]
    ]
  },
  "es_NuevoUsuario": {
    "main": [
     [
      {
       "node": "Registro de usuario - link",
       "type": "main",
       "index": 0
      }
     ]
    ]
  },
  "ObtenerDisponibilidad": {
    "ai_tool": [
     [
      {
       "node": "Pepito - AI Agent",
       "type": "ai_tool",
       "index": 0
      }
     ]
    ]
  },
  "select_doctores_xservicio": {
    "ai_tool": [
     [
      {
       "node": "Pepito - AI Agent",
```

```
        "type": "ai_tool",
        "index": 0
      }
    ]
  ]
},
"select_servicios": {
  "ai_tool": [
    [
      {
        "node": "Pepito - AI Agent",
        "type": "ai_tool",
        "index": 0
      }
    ]
  ]
},
"CalcularEspaciosDisponibles": {
  "ai_tool": [
    [
      {
        "node": "Pepito - AI Agent",
        "type": "ai_tool",
        "index": 0
      }
    ]
  ]
},
"insert_appointments": {
  "ai_tool": [
    [
      {
        "node": "Pepito - AI Agent",
        "type": "ai_tool",
        "index": 0
      }
    ]
  ]
},
"get_appointment_details": {
  "ai_tool": [
    [
      {
        "node": "Pepito - AI Agent",
        "type": "ai_tool",
        "index": 0
```

```
        }
       ]
     ]
    },
    "cancel_appointment": {
     "ai_tool": [
      [
       {
        "node": "Pepito - AI Agent",
        "type": "ai_tool",
        "index": 0
       }
      ]
     ]
    }
  },
  "active": false,
  "settings": {
   "executionOrder": "v1"
  },
  "versionId": "858cd300-c70d-4b47-aac7-8fa4bf93e592",
  "meta": {
   "templateCredsSetupCompleted": true,
   "instanceId": "0d262c77ddf5e98146562ac998010e66121b3ae266f215c22f6bf79557362581"
  },
  "id": "7maqWX6lykzfTZYW",
  "tags": []
}
```

Estructura base de datos PostgreSQL

```
CREATE TABLE public.appointments (
  id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
  created_at timestamp with time zone NOT NULL DEFAULT now(),
  customer_id bigint,
  doctor_id bigint,
  service_id bigint,
  start_time timestamp with time zone,
  end_time timestamp with time zone,
  isActive boolean,
  CONSTRAINT appointments_pkey PRIMARY KEY (id),
  CONSTRAINT appointments_customer_id_fkey FOREIGN KEY (customer_id)
REFERENCES public.customers(identification),
  CONSTRAINT appointments_doctor_id_fkey FOREIGN KEY (doctor_id) REFERENCES
public.doctors(id),
  CONSTRAINT appointments_service_id_fkey FOREIGN KEY (service_id) REFERENCES
public.services(id)
);
CREATE TABLE public.customers (
  identification bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
  created_at timestamp with time zone NOT NULL DEFAULT now(),
  name text,
  lastnames text,
  phone text,
  email text,
  accept_terms boolean,
  CONSTRAINT customers_pkey PRIMARY KEY (identification)
);
CREATE TABLE public.doctor_services (
  id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
  created_at timestamp with time zone NOT NULL DEFAULT now(),
  doctor_id bigint,
  service_id bigint,
  CONSTRAINT doctor_services_pkey PRIMARY KEY (id),
  CONSTRAINT doctor_services_doctor_id_fkey FOREIGN KEY (doctor_id) REFERENCES
public.doctors(id),
  CONSTRAINT doctor_services_service_id_fkey FOREIGN KEY (service_id) REFERENCES
public.services(id)
);
CREATE TABLE public.doctors (
  id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
  created_at timestamp with time zone NOT NULL DEFAULT now(),
  name text,
  email text,
  phone text,
```

```
  CONSTRAINT doctors_pkey PRIMARY KEY (id)
);
CREATE TABLE public.n8n_chat_histories (
  id integer NOT NULL DEFAULT nextval('n8n_chat_histories_id_seq'::regclass),
  session_id character varying NOT NULL,
  message jsonb NOT NULL,
  CONSTRAINT n8n_chat_histories_pkey PRIMARY KEY (id)
);
CREATE TABLE public.services (
  id bigint GENERATED ALWAYS AS IDENTITY NOT NULL,
  created_at timestamp with time zone NOT NULL DEFAULT now(),
  descripción text,
  detalle text,
  CONSTRAINT services_pkey PRIMARY KEY (id)
);
```

Conclusiones

Este proyecto evidenció la alta eficacia de n8n, no solo para flujos de trabajo simples, sino como una plataforma robusta para orquestar sistemas de IA conversacionales complejos. Se logró automatizar de forma integral el ciclo de vida de la gestión de citas de un consultorio: desde el registro de pacientes nuevos (con conexión a PostgreSQL) hasta el agendamiento y la cancelación segura.

La principal innovación fue el uso de un Agente de IA (conectado a Google Gemini) como el "cerebro" central. Este agente demostró la capacidad de utilizar "herramientas" para interactuar con una base de datos PostgreSQL y ejecutar un nodo de Código JS para realizar cálculos complejos, como la verificación de disponibilidad de citas en tiempo real. Esta arquitectura minimiza los errores humanos, como el doble agendamiento, y optimiza la carga del personal administrativo.

El seminario permitió obtener destrezas aplicables en la integración de múltiples tecnologías (IA, base de datos, código y automatización). Se demostró que esta arquitectura es un modelo escalable, capaz de adaptarse a otros ambientes laborales del sector servicios (clínicas, consultorios legales, salones de belleza) que requieran optimizar la interacción con el cliente mediante asistentes inteligentes.

Referencias

- Berenguer, Pau. (2025, 27 de octubre). *Cómo Conectar Supabase y Postgres a N8N y Crear Agentes RAG con Memoria (Plantilla Gratis)* [Video]. YouTube. http://www.youtube.com/watch?v=bSDc76mD-K4
- David Petolescu. (2025, 6 de octubre). *Cómo Hacer Consultas a Tu Base de Datos con Este Agente de IA con n8n (Paso a Paso)* [Video]. YouTube. http://www.youtube.com/watch?v=EfQ6-CCvFes
- n8n. (s.f.). *Documentación oficial de n8n*. n8n.io. Obtenido el 7 de noviembre de 2025, de https://docs.n8n.io/
- Virola. (2025, 19 de marzo). *Tutorial N8N: ¡Construir Chatbot en WhatsApp! (Método fácil)* [Video]. YouTube. http://www.youtube.com/watch?v=M5ZSjBnACtM