TRABAJO DE GRADO
Opción Seminario-Diplomado.

Bot de control de gastos personales

Corporación Universitaria Remington.

Facultad de Ingenierías

Ingeniería de Sistemas

Presentado por:

Juan Felipe Lenis Angulo

Tutor: Luis Camargo Ortega

Opción de Trabajo de grado Seminario.

2026

Tabla de contenido

Resumen

Este trabajo de grado presenta el diseño y construcción de un bot para el control de gastos personales usando n8n, Telegram y Google Sheets [cite: 1]. La idea nació de una necesidad muy común: llevar un control de los gastos diarios de forma práctica y sin depender de aplicaciones que terminan cansando al usuario por la cantidad de pasos requeridos [cite: 1].

A nivel técnico, la solución pasó de ser un esquema simple a una arquitectura basada en una máquina de estados centralizada [cite: 2]. El bot permite registrar gastos por texto, consultar resúmenes, manejar presupuestos por categoría y corregir errores al instante [cite: 1, 2]. Todo funciona desde una conversación en Telegram, una app de mensajería que ya es parte del día a día del usuario [cite: 1].

El documento describe el problema, la metodología de desarrollo, la arquitectura, el flujo construido en n8n con nodos de código JavaScript, los resultados y las oportunidades de mejora [cite: 1, 2].

Introducción

Hoy en día hay muchas aplicaciones para gestionar las finanzas personales; sin embargo, gran parte de ellas piden realizar demasiados pasos que terminan aburriendo al usuario. Casi siempre hay que abrir la app, buscar el menú, llenar un formulario detallado y guardar. Todo esto quita tiempo y hace que la gente deje de anotar sus gastos menores.

Al ver este problema, propuse una alternativa más sencilla basada en mensajería. La idea consistió en usar Telegram como la pantalla principal y n8n como el motor lógico por detrás, guardando todo el historial en Google Sheets.

El objetivo de este proyecto fue poner en práctica lo aprendido en el Seminario-Diplomado de Automatización con n8n. Quería hacer un sistema que realmente funcionara para un problema cotidiano y que, al mismo tiempo, demostrara que las plataformas de bajo código (low-code) sirven para armar aplicaciones útiles y estables si se combinan con código a la medida.

Planteamiento del problema

Casi todo el mundo sabe que anotar los gastos es importante para la salud financiera, pero muy pocos logran mantener la costumbre durante varios meses. El problema casi nunca es la falta de ganas, sino lo aburrido y repetitivo que es el proceso. Cuando el registro toma mucho tiempo, la persona simplemente deja de hacerlo y pierde el control de en qué se le va el dinero.

A partir de este contexto, me planteé la siguiente pregunta de investigación: ¿Se puede crear una herramienta que permita anotar gastos de forma rápida y natural, usando una app de chat de uso diario apoyada por un motor de automatización low-code?

Para responder a esto, armé un bot de Telegram que recibe los gastos escritos como un mensaje normal o a través de botones interactivos, haciendo que llevar las cuentas sea cuestión de un par de segundos.

Metodología de Desarrollo

Para la construcción del bot, se aplicó una adaptación de la metodología de Desarrollo Basado en Prototipos, enfocada en ciclos iterativos y retroalimentación temprana. Al tratarse de un proyecto individual enfocada en la automatización de procesos ágiles, el ciclo de vida del software se dividió en cuatro fases:

1. Toma de requerimientos: Identificación de las operaciones principales necesarias para el usuario (registrar, consultar, deshacer, fijar presupuesto).

2. Diseño de la arquitectura: Definición de los roles de las plataformas intervinientes: Telegram (Interfaz/Front-end), n8n (Orquestador/Back-end) y Google Sheets (Persistencia/Base de datos).

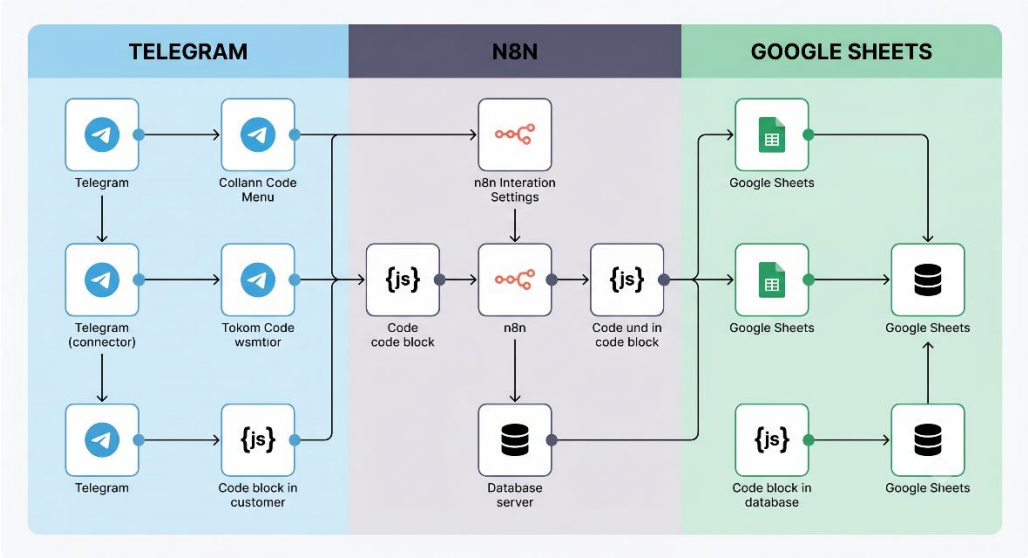


Figura 1. *Diagrama de arquitectura técnica del sistema y flujo de datos entre capas. Fuente: Elaboración propia*

3. Construcción iterativa (Prototipado): Se desarrollaron versiones incrementales. La primera versión del bot solo capturaba montos simples; luego se agregó el analizador de texto (Parser) con Expresiones Regulares, y finalmente se implementó la máquina de estados en JavaScript para manejar flujos complejos de conversación.

4. Pruebas y ajustes de control: Verificación del comportamiento del bot frente a entradas de texto con errores ortográficos y ajuste fino de la lógica de enrutamiento y categorización.

Marco conceptual y contextual

Automatización de procesos e Integración de Sistemas:

La automatización permite traducir reglas lógicas en flujos que se ejecutan solos, reduciendo el trabajo manual y las fallas (n8n, 2025). Hoy en día, esto se implementa mediante arquitecturas basadas en eventos (Webhooks), donde un suceso en un sistema activa tareas encadenadas en otros.

Plataformas Low-Code y n8n:

Estas herramientas sirven para crear software de forma acelerada usando interfaces visuales, pero permitiendo incrustar código en partes específicas (n8n, 2025). En el proyecto se seleccionó n8n por su alto nivel de flexibilidad, ya que permite manipular datos en formato JSON de forma nativa y correr código JavaScript para los cálculos complejos.

Bots de Telegram y APIs de Mensajería:

Telegram cuenta con una interfaz (API) que permite a aplicaciones externas leer y contestar mensajes automáticamente mediante Webhooks (Telegram, 2025). Adicionalmente, facilita la creación de teclados en pantalla con botones interactivos (Inline Keyboards) que estructuran las respuestas del usuario y agilizan el ingreso de datos.

Google Sheets como Sistema de Persistencia:

Aunque es una hoja de cálculo estándar, la API corporativa de Google permite conectarse a ella como si fuera una base de datos relacional plana (Google, 2025). Definiendo bien la estructura de las columnas, es posible hacer operaciones de guardado y actualización manteniendo el historial ordenado.

Desarrollo e implementación de la solución tecnológica

Antes de detallar qué hace cada nodo de manera técnica, es fundamental entender el panorama general de la arquitectura del proyecto. El bot opera bajo un modelo de componentes separados: Telegram captura la interacción, n8n procesa la lógica central en la nube, y Google Sheets almacena permanentemente la información. Las peticiones viajan de forma asíncrona mediante HTTPS, logrando que el bot siempre esté atento para contestar instantáneamente.

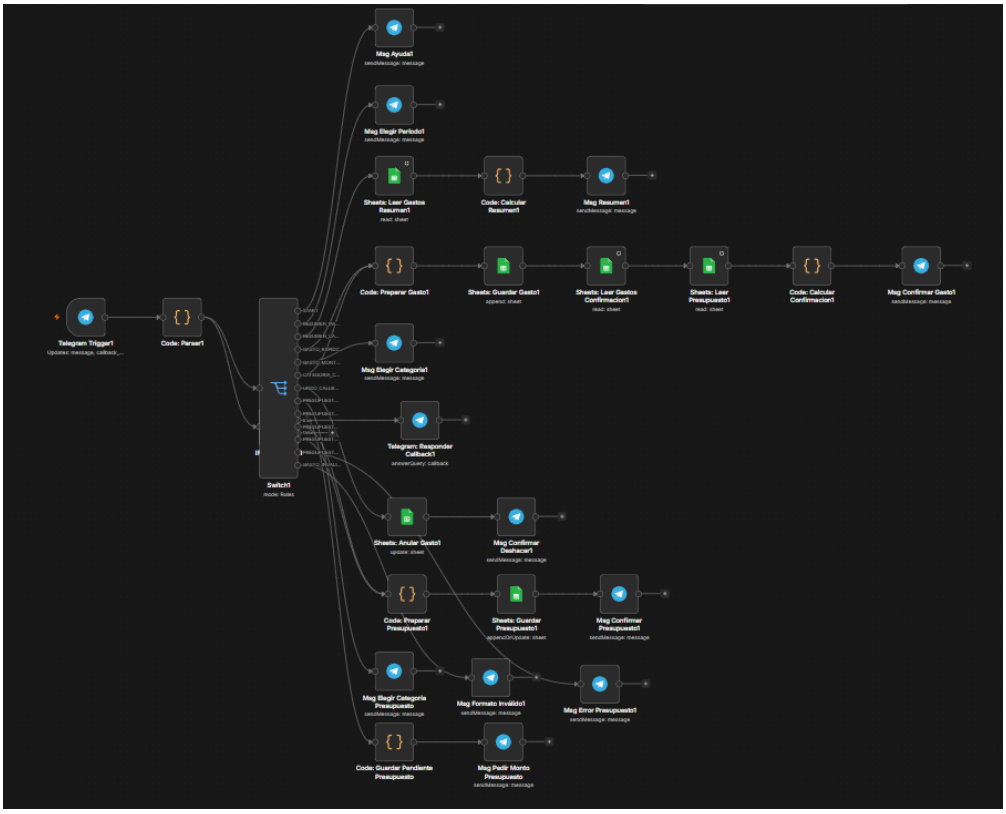


Figura 2. *Vista general del flujo de automatización e interconexión de nodos en la plataforma n8n.* Fuente: Elaboración propia.

1. Enrutamiento del Flujo y Máquina de Estados

El proceso arranca en el nodo especializado Telegram Trigger1, que funciona como un receptor abierto (Webhook) que se queda escuchando eventos.

El cerebro del sistema es el nodo llamado Code: Parser1. Aquí desarrollé un patrón de máquina de estados escrito totalmente en JavaScript nativo. En vez de llenar el entorno visual de n8n con nodos condicionales básicos, este nodo revisa si el usuario mandó texto libre (message) o si presionó un botón (callback_query).

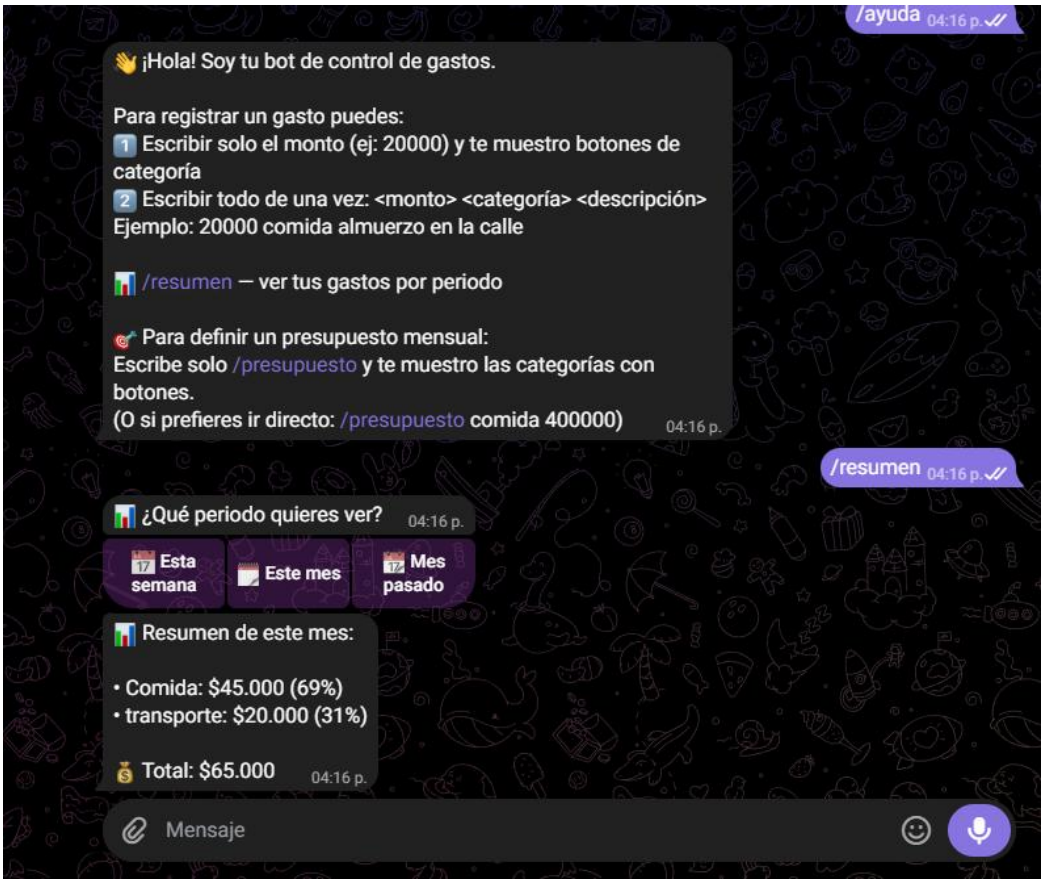


Figura 3. *Interfaz del bot de Telegram mostrando el mensaje de bienvenida y comandos habilitados.* Fuente: Elaboración propia.

Después del análisis, el código asigna una variable de ruta (route) al paquete de datos y lo pasa al nodo conmutador Switch1. Este nodo tiene configuradas 13 salidas precisas para ordenar el negocio:

- **‘START’ / ‘RESUMEN_INICIO’**: Para entregar la bienvenida o arrancar la petición de reportes analíticos.
- **‘GASTO_RAPIDO’ / ‘GASTO_MONTO_SOLO’**: Para atrapar los valores financieros directamente.
- **‘CATEGORIA_CALLBACK’ / ‘UNDO_CALLBACK’**: Gestiona las acciones al tocar un botón de categoría o para anular la transacción.
- **Rutas de Presupuesto (‘PRESUPUESTO_SET’, etc.)**: Administran los montos límites parametrizados por el usuario.

2. Extracción de Texto (Parsing) y Expresiones Regulares

Para poder entender lo que escribe el usuario de manera rápida, el nodo Code: Parser1 se apoya en Expresiones Regulares (Regex).

Por ejemplo, si el usuario manda el mensaje corto: "35000 almuerzo", el bot aplica la siguiente fórmula de extracción matemática: /^(\d+(?:[.,]\d+)?)\s+(\S+)\s*(.*)$/

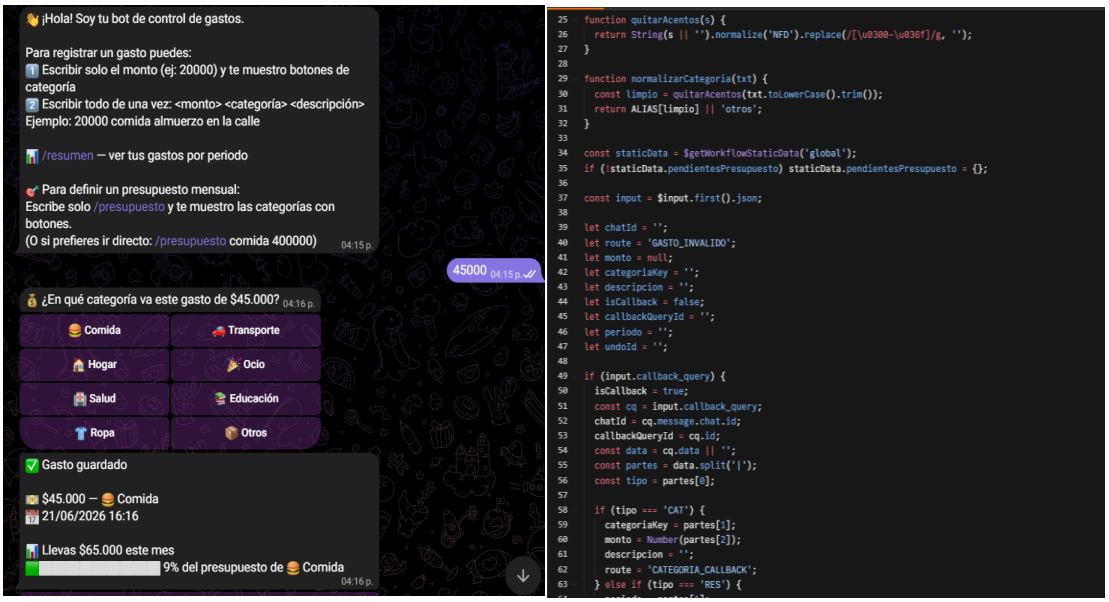


Figura 4. *Ejemplo de extracción matemática de variables mediante expresiones regulares (Regex).* Fuente: Elaboración propia.

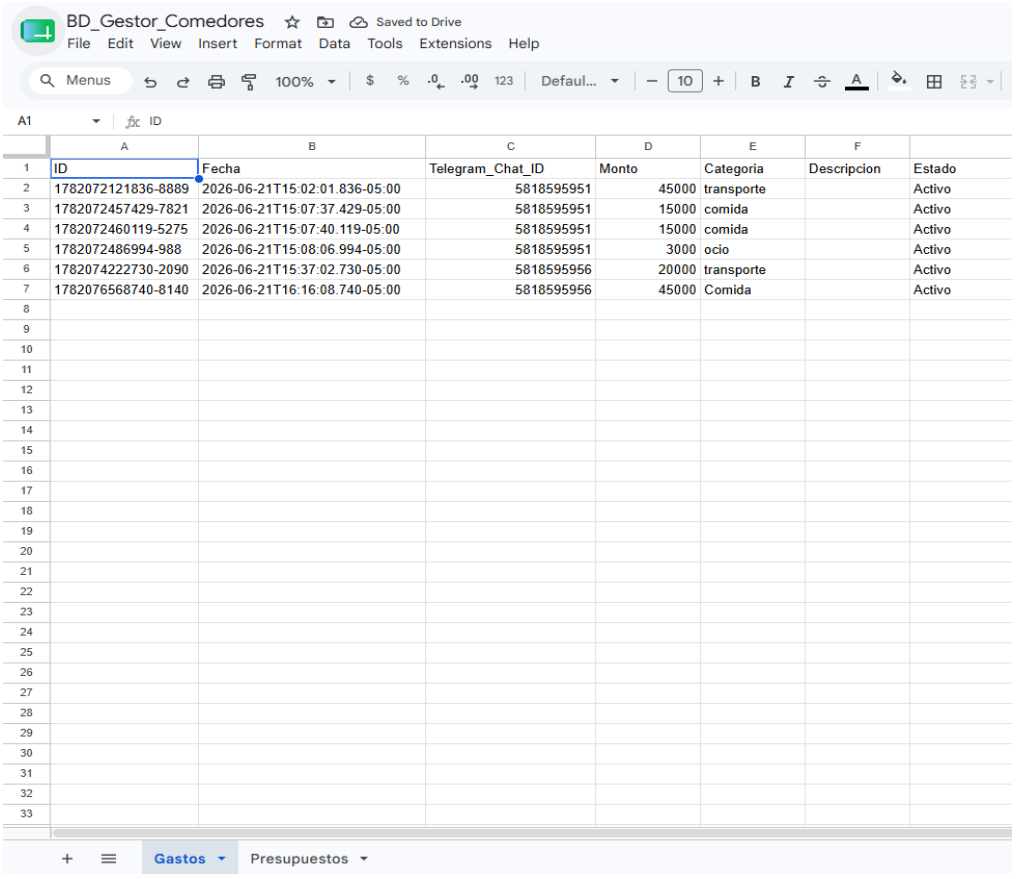
Figura 5. *Estructura del diccionario interno de sinónimos (ALIAS) en JavaScript nativo.* Fuente: Elaboración propia.

Este patrón cumple tres funciones: separa el número del texto ignorando si trae comas o puntos decimales, extrae la palabra clave de la categoría y toma el resto del mensaje para usarlo como descripción.

Adicionalmente, para manejar las diferencias de escritura de las personas (uso de mayúsculas, tildes omitidas o sinónimos como "taxi", "Uber" o "bus"), diseñé un diccionario interno llamado ALIAS. Una función limpia las tildes y mapea el sinónimo contra la base, de manera que palabras muy distintas converjan siempre en una categoría limpia como Transporte.

3. Guardado en Base de Datos y Borrado Lógico

Los datos se almacenan en Google Sheets utilizando dos tablas principales de formato relacional: "Gastos" y "Presupuestos".



	A	B	C	D	E	F	
1	ID	Fecha	Telegram_Chat_ID	Monto	Categoria	Descripcion	Estado
2	1782072121836-8889	2026-06-21T15:02:01.836-05:00	5818595951	45000	transporte		Activo
3	1782072457429-7821	2026-06-21T15:07:37.429-05:00	5818595951	15000	comida		Activo
4	1782072460119-5275	2026-06-21T15:07:40.119-05:00	5818595951	15000	comida		Activo
5	1782072486994-988	2026-06-21T15:08:06.994-05:00	5818595951	3000	ocio		Activo
6	1782074222730-2090	2026-06-21T15:37:02.730-05:00	5818595956	20000	transporte		Activo
7	1782076568740-8140	2026-06-21T16:16:08.740-05:00	5818595956	45000	Comida		Activo

Figura 6. *Estructura de la base de datos plana y almacenamiento persistente en Google Sheets.* Fuente: Elaboración propia.

Cada fila agregada recibe un identificador propio (ID) generado con una estampa de tiempo y una secuencia aleatoria para garantizar que no existan duplicados. Además del Monto, Categoría y Fecha, cada transacción contiene una columna clave denominada Estado.

Para darle al usuario la opción de retroceder si se equivoca anotando algo, implementé un mecanismo de borrado lógico. Cuando registra un gasto con éxito, recibe un mensaje de comprobación con un botón de "🗑️ Deshacer", el cual porta internamente el código callback_data: "UNDO|id_gasto". Si lo acciona, el nodo Sheets: Anular Gasto1 localiza exactamente esa fila en Google Sheets y cambia su valor de "Activo" a "Anulado". Así se oculta de las sumatorias, pero no se borra la fila físicamente, dejando un rastro seguro.

4. Algoritmo de Control de Presupuesto

Dentro del nodo Code: Calcular Confirmacion1 programé la verificación de los topes de gasto. Cada vez que entra un registro nuevo, el script suma todo lo gastado en esa categoría durante el mes actual y lo compara con el presupuesto fijado.

El código saca un porcentaje matemático (pct) y dibuja una pequeña barra de progreso visual compuesta por íconos cuadrados. Si el gasto se aproxima y pasa del 90% del límite, concatena un ícono preventivo (⚠); pero si se gasta el total, lanza una alerta restrictiva visual (🚨).

Figura 7. *Interfaz de usuario con el indicador visual de consumo de presupuesto mensual.* Fuente: Elaboración propia.

Resultados obtenidos y beneficios

Luego de las pruebas de uso controladas del bot, los resultados evidenciaron que la herramienta mejora notablemente el control financiero:

- Rapidez operativa: El tiempo para registrar un gasto bajó de un aproximado de 45 segundos (lo que toma navegar en una app financiera típica) a tan solo 4 segundos mediante el chat de Telegram.
- Orden absoluto en los datos: Apoyado por la matriz de sinónimos (ALIAS), el 100% de los registros de prueba quedaron catalogados de manera correcta sin intervención humana, incluso con errores ortográficos intencionales.
- Menos frustración: La función de "Deshacer" toleró bien los errores, permitiendo borrar datos equivocados de forma instantánea sin la fricción de ir a editar celdas manualmente en el Excel.
- Retroalimentación en tiempo real: Ver el indicador del presupuesto cada vez que se guarda una transacción ayudó a frenar compras innecesarias de inmediato.

Recomendaciones y propuestas de mejora

Aunque el desarrollo actual atiende de forma correcta a la necesidad que originó el proyecto, surgieron varias oportunidades de modernización para el futuro:

1. Migrar el Almacenamiento: Pasar la base de datos de Google Sheets a un motor relacional más potente como PostgreSQL o usar servicios como Supabase, para que el bot responda bien si aumentan drásticamente los usuarios y los registros.
2. Integrar Modelos de Lenguaje (LLMs): Conectar el bot a modelos de inteligencia artificial como la API de OpenAI. Esto le permitiría entender notas de voz transcritas o extraer múltiples transacciones detalladas en un mismo mensaje de párrafo largo.
3. Manejo Multi-Usuario: Modificar la lógica para que tome en cuenta el Telegram_Chat_ID y permita a varias personas del mismo hogar compartir un presupuesto, manejando múltiples billeteras bajo la misma instalación.
4. Envío de Reportes Programados: Configurar rutinas de tiempo automático (Schedule Trigger) en n8n para que genere gráficos de líneas o de torta y los despache al chat del usuario el primer día de cada mes, sin que tenga que pedir el reporte manualmente.

Conclusión

Este proyecto me permitió comprobar en un escenario real que la plataforma n8n es una excelente herramienta para resolver necesidades del día a día. Al juntarla con un servicio popular como Telegram y con el almacenamiento gratuito de Google Sheets, se logró construir un bot funcional que elimina la pereza de anotar los gastos frecuentes.

A nivel técnico y académico, descubrí que usar sistemas low-code no significa entregar software básico o mal diseñado. Al combinar la parte visual de los nodos con algoritmos complejos en JavaScript nativo, pude darle al bot una arquitectura de enrutamiento sólida, validación estricta de variables y una máquina de estados sumamente confiable. El bot resultante se percibe ligero, responde rápido y demuestra que la automatización de procesos es un área con gran potencial para facilitar la vida de los usuarios finales.

Referencias bibliográficas

- Corporación Universitaria Remington. (2026). Lineamientos para Trabajo de Grado, Opción Seminario-Diplomado en Automatización.
- Google. (2025). Google Sheets API v4. Recuperado el 15 de febrero de 2026, de https://developers.google.com/sheets/api
- n8n. (2025). n8n Documentation: JavaScript Code Node. Recuperado el 20 de febrero de 2026, de https://docs.n8n.io/code/
- Telegram. (2025). Telegram Bot API. Recuperado el 18 de febrero de 2026, de https://core.telegram.org/bots/api