Universidad de Almería

Máster en Ingeniería Informática

Integración de Tecnologías y Servicios Informáticos

Práctica 7

Orquestación de Modelos de IA - Integración con Gemini

Autor: Johan Eduardo CAla Torra

Profesor: Manel Manu Vicente

Fecha: 14 de diciembre de 2025

Curso 2024-2025

Índice

1. Introducción

En esta práctica se integró la inteligencia artificial con la base de datos de tareas creada en la Práctica 6. El objetivo consistió en diseñar un flujo de trabajo capaz de:

- Leer tareas que aún no han sido analizadas desde PostgreSQL
- Procesarlas mediante Google Gemini para determinar su nivel de prioridad
- Tomar decisiones automatizadas basadas en la clasificación de la IA
- Implementar notificaciones y registros según la prioridad asignada

Esta práctica demuestra la orquestación de servicios de IA en flujos de trabajo empresariales, permitiendo la automatización inteligente de procesos de negocio mediante el uso de modelos de lenguaje natural.

2. Desarrollo del Flujo de Trabajo Guiado

2.1. Paso 1: Obtener Credenciales de Google AI

Para comenzar la integración con Google Gemini, fue necesario obtener las credenciales de acceso a la API.

2.1.1. Proceso de obtención

1. Acceder a `https://aistudio.google.com/app/apikey`
2. Seleccionar Çrear clave API"
3. Copiar la clave generada (cadena alfanumérica extensa)

La clave obtenida fue:

```
AIzaSyACsjKxw_zf7E0ABAfx2CUXttGz9YmXinY
```

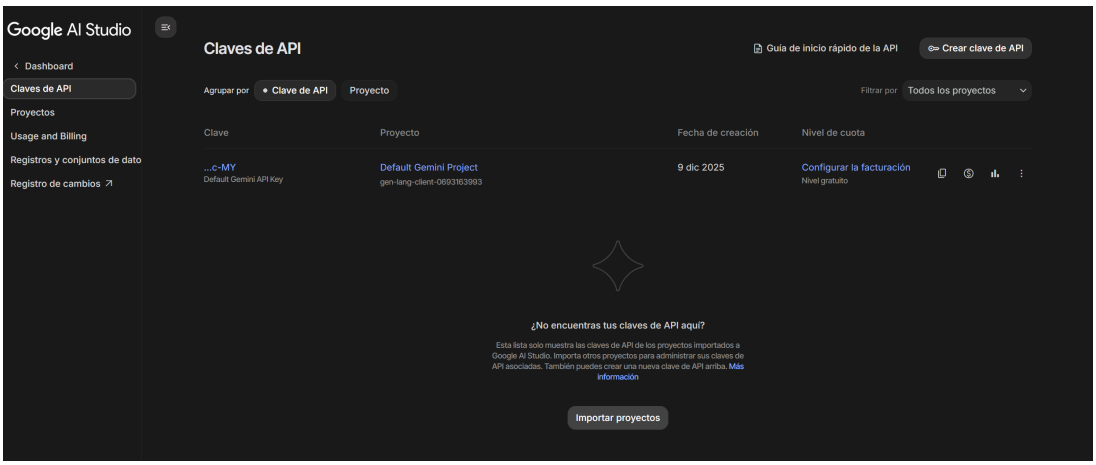

Figura 1: Interfaz de Google AI Studio para generar claves API

2.1.2. Configuración en n8n

Una vez obtenida la clave, se procedió a configurarla en n8n:

1. En n8n, acceder a **Credentials ¿Add credential**
2. Buscar "Google Gemini API"(o "Google PaLM API")
3. Asignar el nombre identificativo: `Gemini Key`
4. Pegar la clave API en el campo correspondiente
5. Guardar la credencial

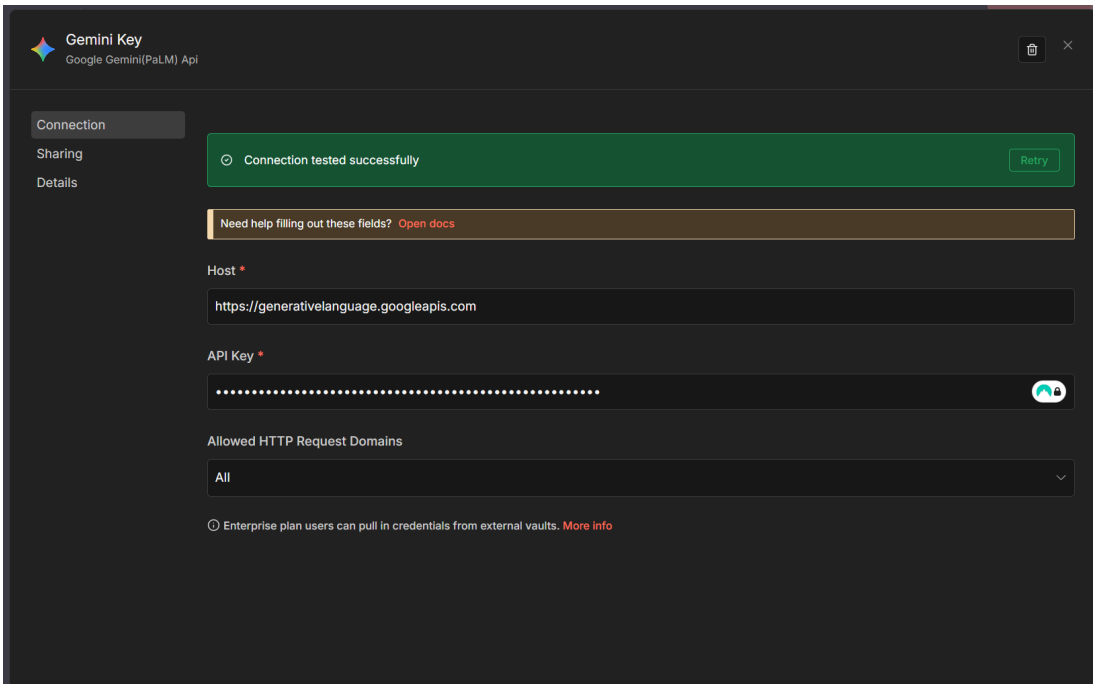

Figura 2: Configuración de credenciales de Gemini en n8n

2.2. Paso 2: Flujo Base - Leer Tareas de PostgreSQL

2.2.1. Configuración del flujo

Se creó un nuevo flujo de trabajo con los siguientes componentes:

1. **Manual Trigger**: Nodo de inicio manual del flujo
2. **PostgreSQL**: Nodo para consultar la base de datos

2.2.2. Configuración del nodo PostgreSQL

El nodo PostgreSQL se configuró con los siguientes parámetros:

- **Credential**: PostgreSQL (Tasks) - creada en la Práctica 6
- **Operation**: Execute Query
- **Query**:

```
SELECT * FROM tasks
WHERE description IS NOT NULL
  AND description != ''
LIMIT 5;
```

Nota: Se utilizó el límite de 5 registros para no consumir excesiva cuota de API durante las pruebas.

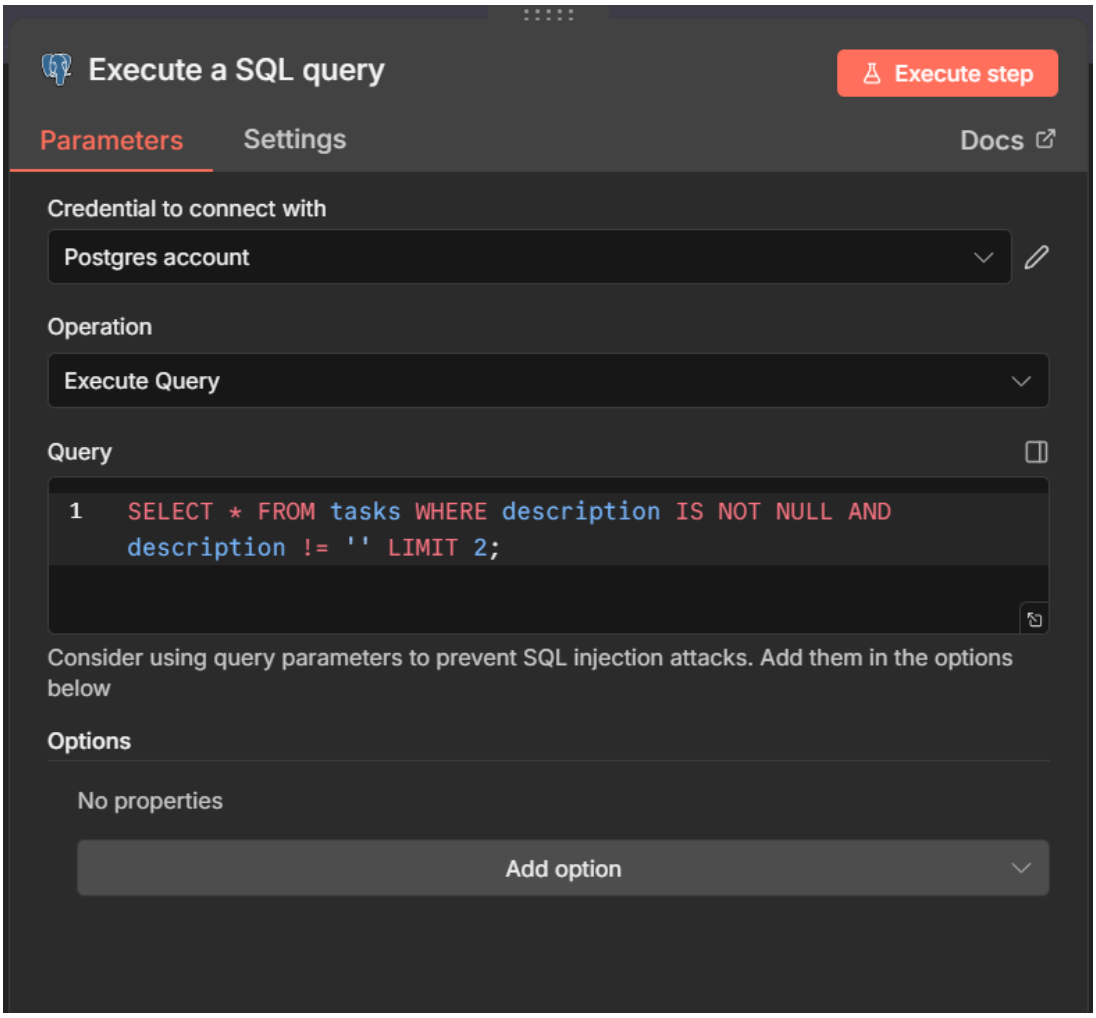

Figura 3: Configuración del nodo PostgreSQL para leer tareas

2.2.3. Verificación

Al ejecutar el nodo en modo "Test step", se verificó que las tareas se recibían correctamente desde la base de datos, confirmando la conexión exitosa.

2.3. Paso 3: Analizar Tareas con Gemini (Prompt Básico)

2.3.1. Configuración del nodo Google Gemini

Se conectó un nodo "Google Gemini.ªl nodo PostgreSQL con la siguiente configuración:

- **Credential**: Gemini Key (creada en el Paso 1)
- **Resource**: Text
- **Operation**: Message a Model
- **Model**: models/gemini-2.5-flash

2.3.2. Diseño del Prompt

El prompt se diseñó para obtener una clasificación clara y consistente:

```
Eres un jefe de proyecto experimentado. Basandote en la
siguiente descripcion de una tarea, determina su prioridad.
Responde UNICAMENTE con una de estas tres palabras: ALTA,
MEDIA o BAJA.

Descripcion de la tarea: "{{ $json.description }}"
```

Características clave del prompt:

- **Rol definido**: "jefe de proyecto experimentado.^{es}tablece el contexto
- **Instrucción clara**: Se especifica exactamente qué hacer
- **Formato de salida restringido**: Solo tres palabras posibles
- **Inyección de datos**: Usa expresiones de n8n para insertar la descripción

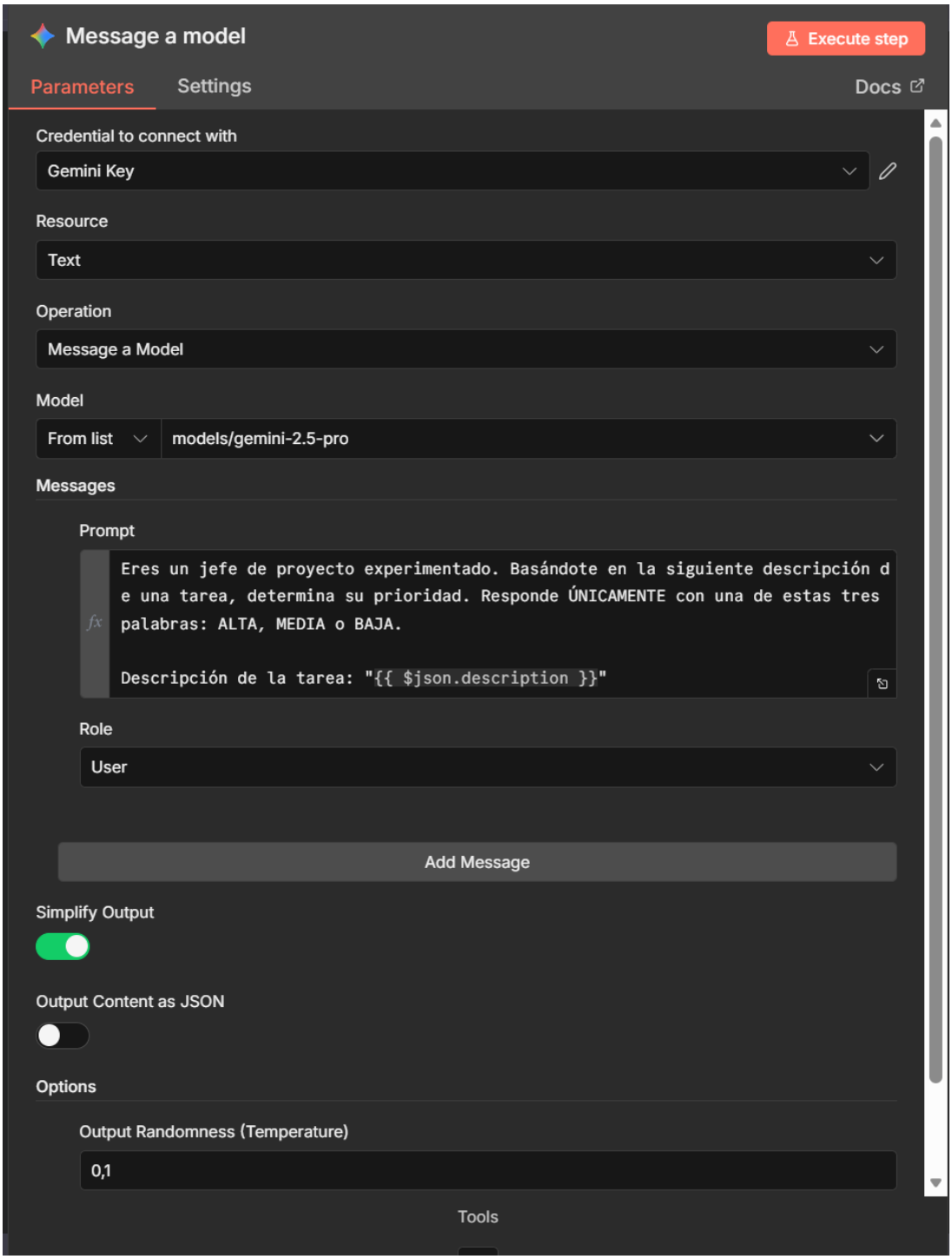

Figura 4: Configuración del nodo Google Gemini con el prompt de clasificación

2.3.3. Resultado

De esta manera, cada tarea enviada al modelo obtiene automáticamente una clasificación de prioridad (ALTA, MEDIA o BAJA) basada en su descripción.

2.4. Paso 4: Tomar Decisiones Basadas en la IA

2.4.1. Implementación del nodo Switch

Una vez que Gemini ha etiquetado las tareas, se implementó la lógica de decisión mediante un nodo "Switch".

2.4.2. Configuración del Switch

- **Mode**: Rules
- **Routing Field**: {{ `$json.content.parts[0].text` }}

Reglas de enrutamiento:

1. **Output 0 (ALTA)**:
   - Condición: `content.parts[0].text` equals .ªLTA"
2. **Output 1 (MEDIA)**:
   - Condición: `content.parts[0].text` equals "MEDIA"
3. **Output 2 (Default - BAJA)**:
   - Captura: "BAJA.º cualquier respuesta inesperada

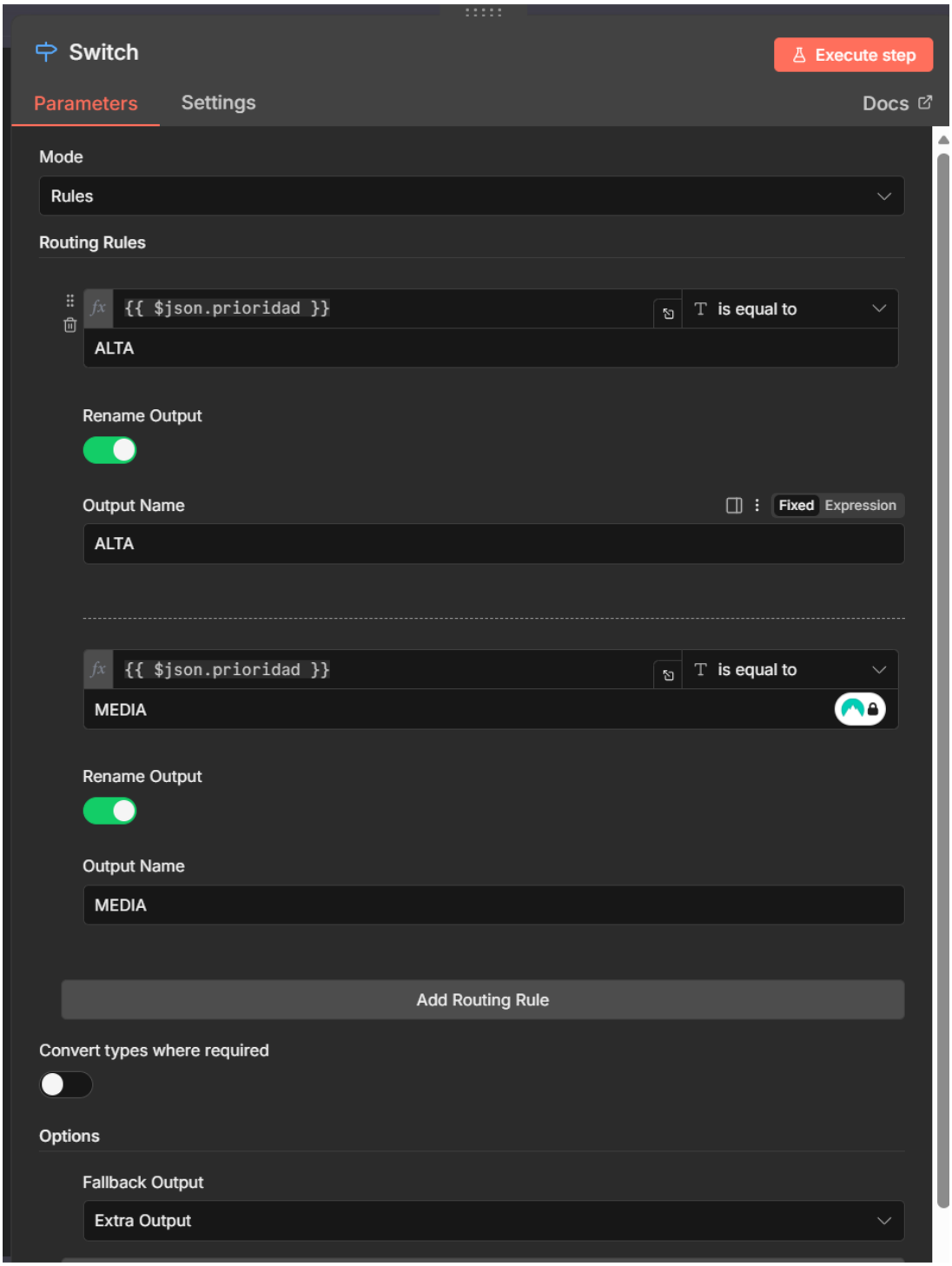

Figura 5: Configuración del nodo Switch con las reglas de enrutamiento

2.5. Paso 5: Notificar y Registrar (Simulación)

2.5.1. Implementación de notificaciones

Para la rama de prioridad ALTA (Output 0), se conectó un nodo "Send Emailçon la siguiente configuración:

- **From Email**: pm161@inlumine.ual.es

- **To Email**: chrisabeonline@gmail.com
- **Subject**: ¡Alerta de Tarea Prioridad ALTA!
- **Email Format**: Text

2.5.2. Contenido del mensaje

El cuerpo del correo utiliza expresiones de n8n para personalizar el mensaje:

```
La tarea "{{ $('Execute a SQL query').item.json.title }}"
(ID: {{ $('Execute a SQL query').item.json.id }})
ha sido clasificada como ALTA prioridad por la IA.
```

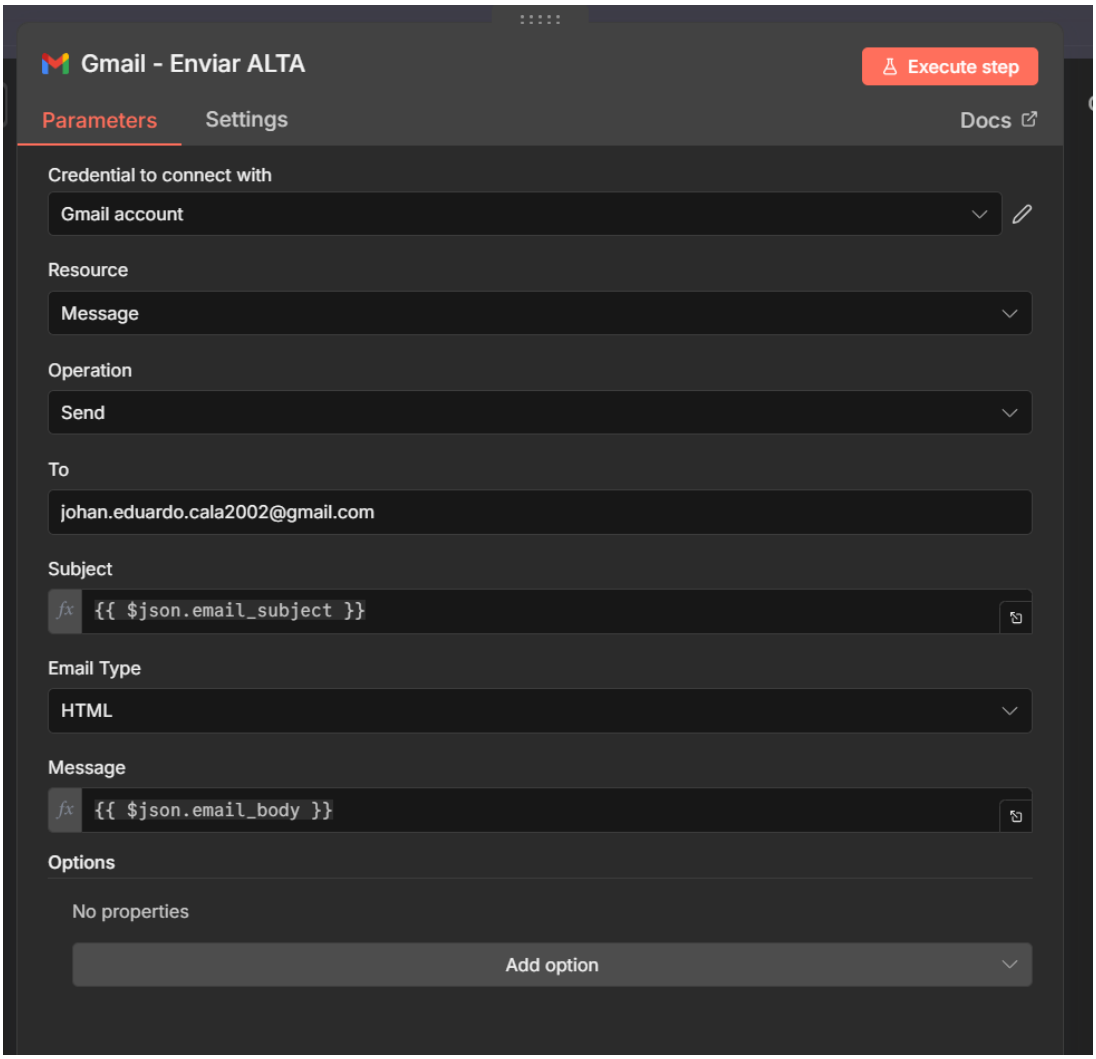

Figura 6: Configuración del nodo Send Email para alertas de alta prioridad

2.5.3. Verificación de entrega

Se verificó que el mensaje llegó exitosamente al correo electrónico configurado.

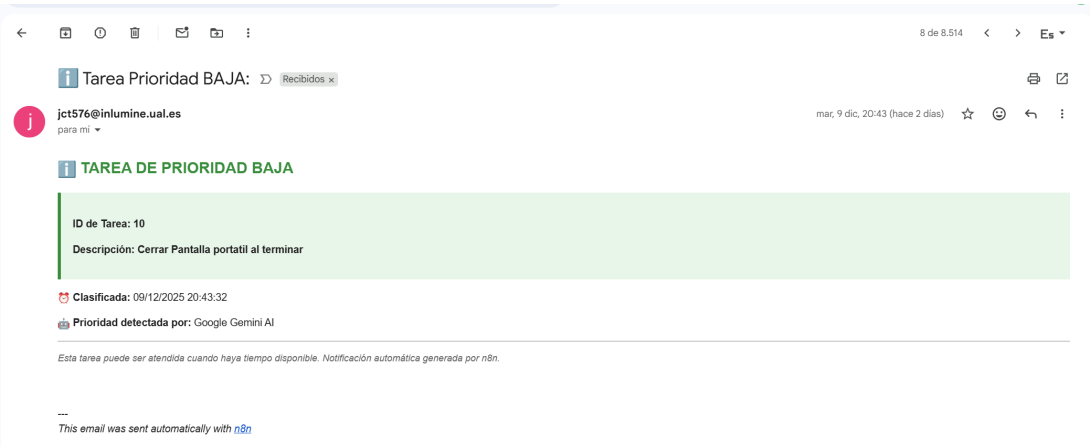

Figura 7: Email recibido confirmando la alerta de tarea de baja prioridad

2.5.4. Diagrama del flujo completo

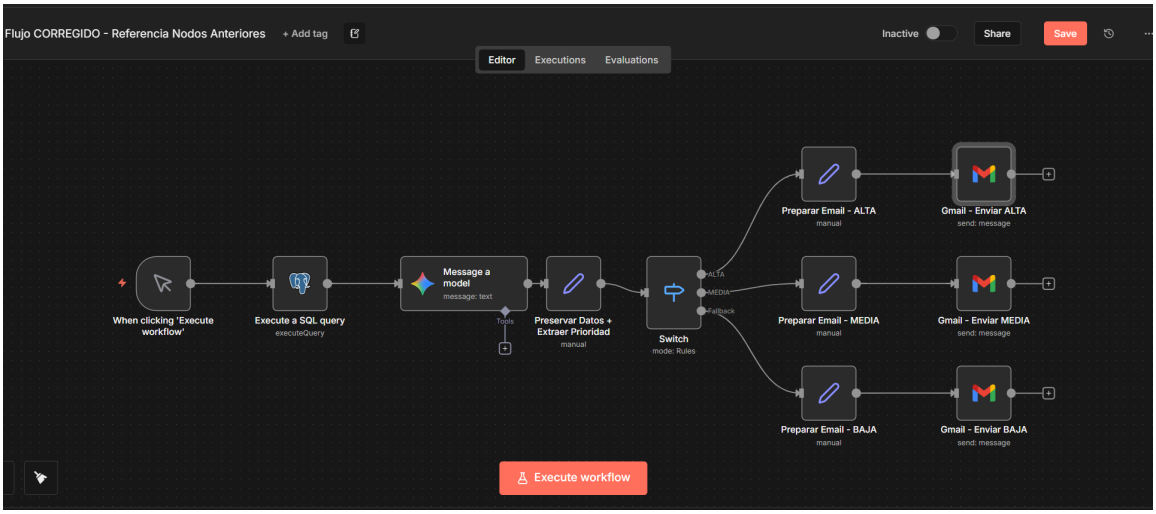

Figura 8: Flujo de trabajo completo del análisis de prioridad con Gemini

3. Ejercicios Propuestos

3.1. Ejercicio 1: Generador de Resúmenes (Dificultad: Baja)

3.1.1. Objetivo

Adaptar el flujo de trabajo guiado para que, en lugar de clasificar la prioridad, genere un resumen conciso de cada tarea.

3.1.2. Modificaciones realizadas

1. Modificación del Prompt de Gemini Se cambió el prompt del nodo "Google Gemini"para solicitar un resumen en lugar de clasificación:

```
Genera un resumen conciso de una sola frase
(maximo 10 palabras) para la siguiente tarea:
"{{ $json.description }}"
```

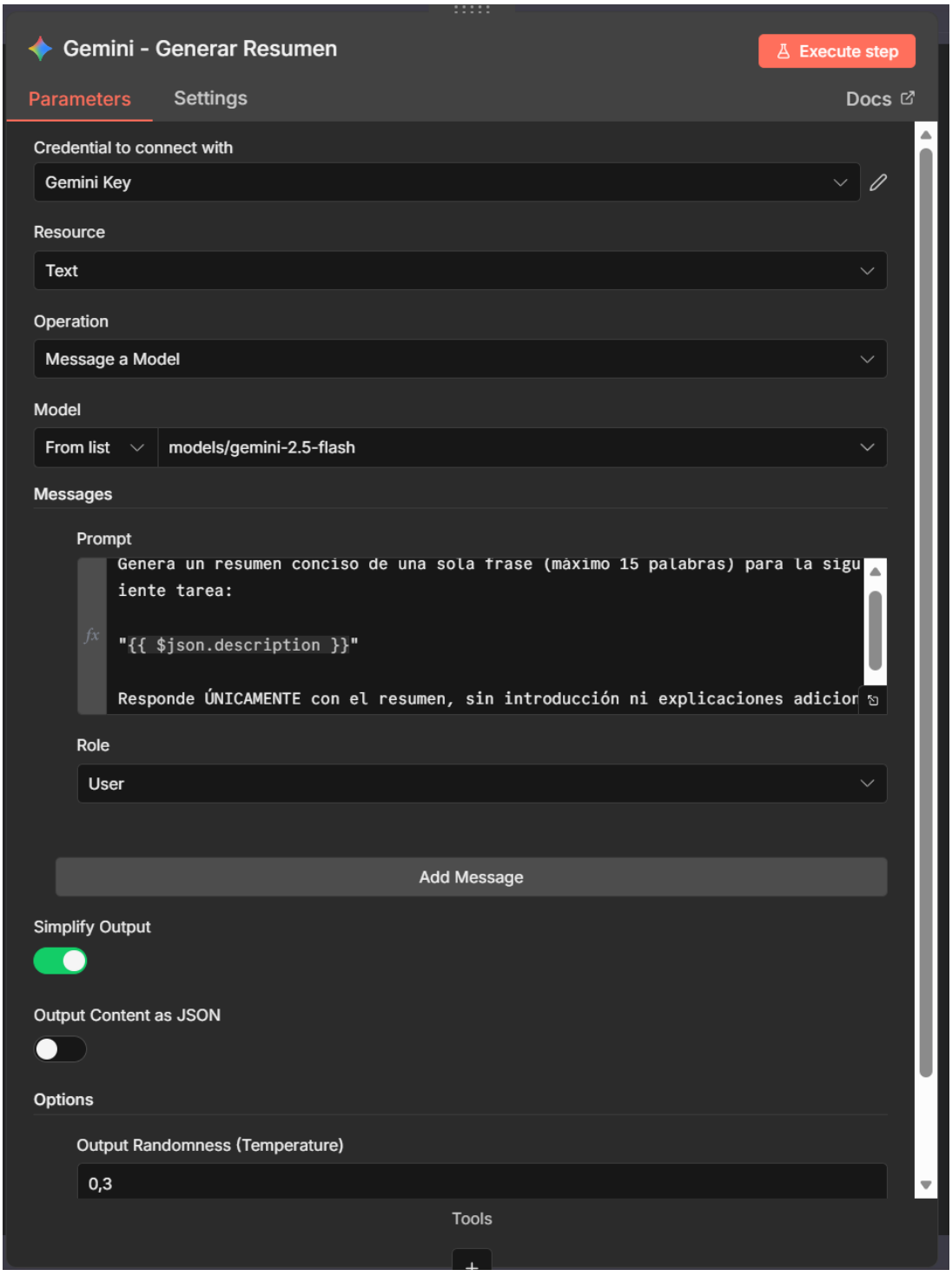

Figura 9: Prompt modificado para generación de resúmenes

2. Eliminación del Switch Se eliminó el nodo "Switchza que no se necesita enrutamiento condicional para esta funcionalidad.

3. Almacenamiento del resumen Se conectó un nodo .Edit Fields (Set)"después de Gemini con la siguiente configuración:

- **Field to Set**: resumen

- **Value**: {{ `$json.content.parts[0].text` }}

Esto almacena la salida de Gemini en un nuevo campo llamado `resumen` dentro de los datos de la tarea.

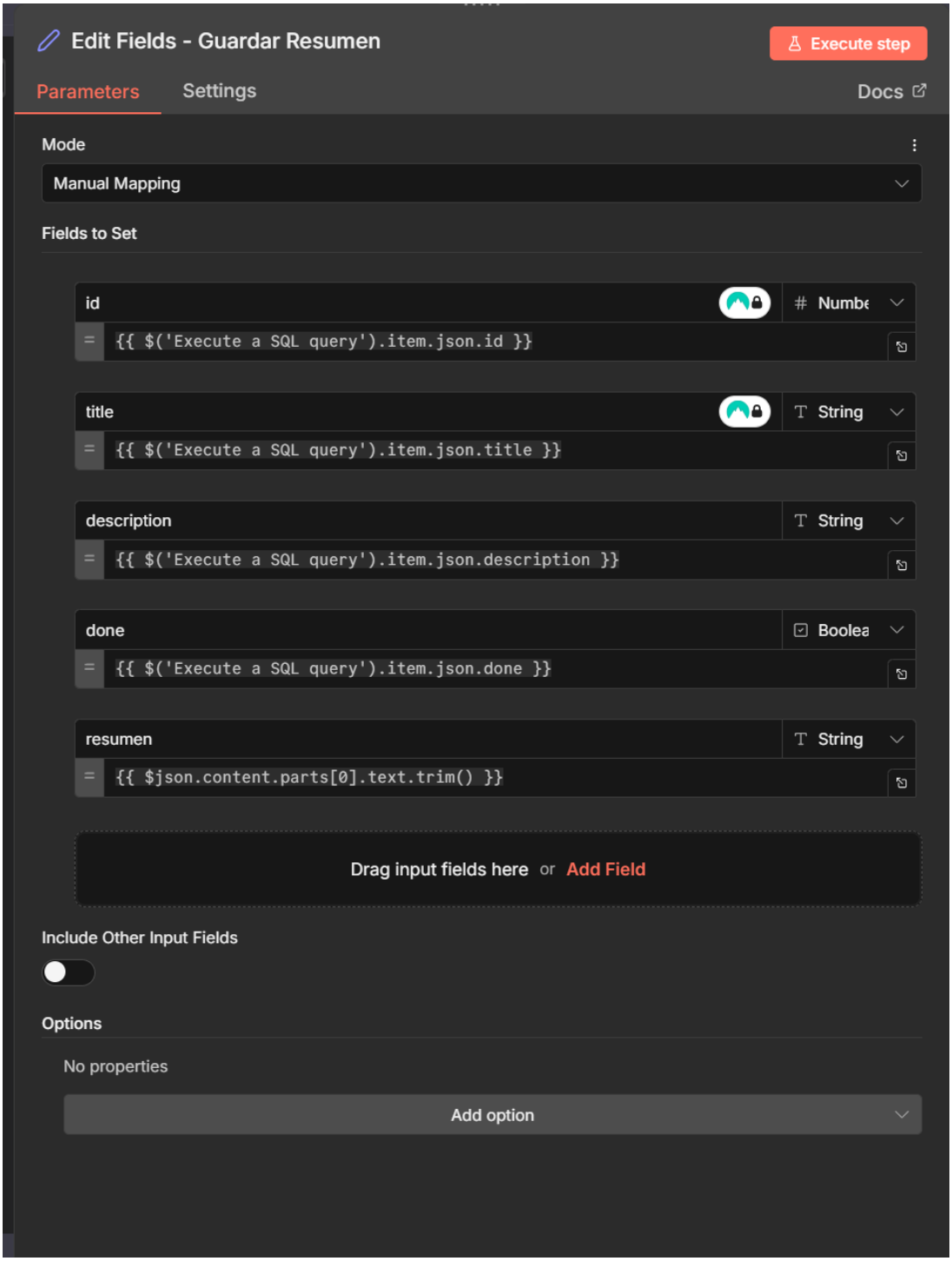

Figura 10: Configuración del nodo Edit Fields para almacenar el resumen

3.1.3. Resultado

Cada tarea ahora incluye un campo adicional `resumen` con una descripción concisa generada por IA.

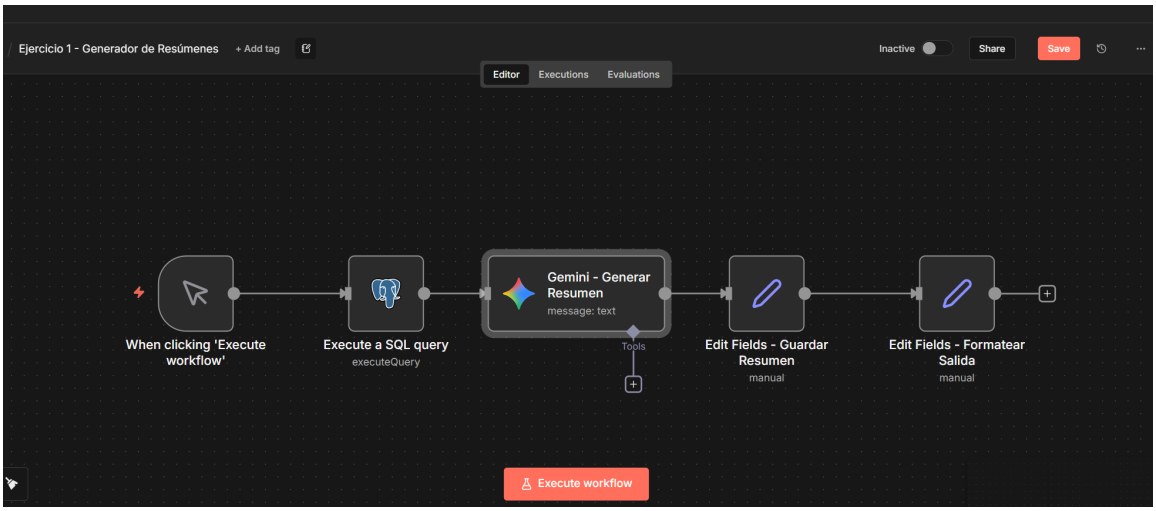

Figura 11: Flujo completo del Ejercicio 1

3.2. Ejercicio 2: Extracción de Entidades y Formato JSON (Dificultad: Media)

3.2.1. Objetivo

Demostrar el patrón de "Prompt Fuerte"mediante la extracción de entidades estructuradas desde texto no estructurado, obteniendo una respuesta en formato JSON válido.

3.2.2. Implementación

1. Webhook Trigger Se inició un nuevo flujo con un nodo "Webhook"para recibir datos desde una petición HTTP:

- **Path**: /webhook-test/extraccion-json
- **HTTP Method**: POST

2. Simulación de entrada con curl Se simuló la entrada de datos mediante el comando:

```
curl -X POST http://localhost:5678/webhook-test/extraccion-json \
  -H "Content-Type: application/json" \
  -d '{"texto": "El cliente manel.mena@ual.es informa de un error
     grave (ID: 500) en la pagina de login."}'
```

Figura 12: Configuración del Webhook y prueba con curl

3. Prompt de extracción estructurada Se configuró el nodo "Google Geminiçon un prompt diseñado para extraer entidades específicas:

```
Extrae las siguientes entidades de este texto.
Responde UNICAMENTE con un objeto JSON valido con
las claves "email_cliente", "tipo_problema" y
"id_incidencia". Si un valor no se encuentra,
retorna las siguientes utilidades del texto. NO
incluyas ningun texto extra, NI las triples
comillas de markdown.

Texto: "{{ $json.body.texto }}"
```

Características del prompt:

- Define claramente el formato de salida (JSON)
- Especifica las claves exactas requeridas
- Maneja casos donde falten valores
- Prohíbe texto adicional o formato markdown

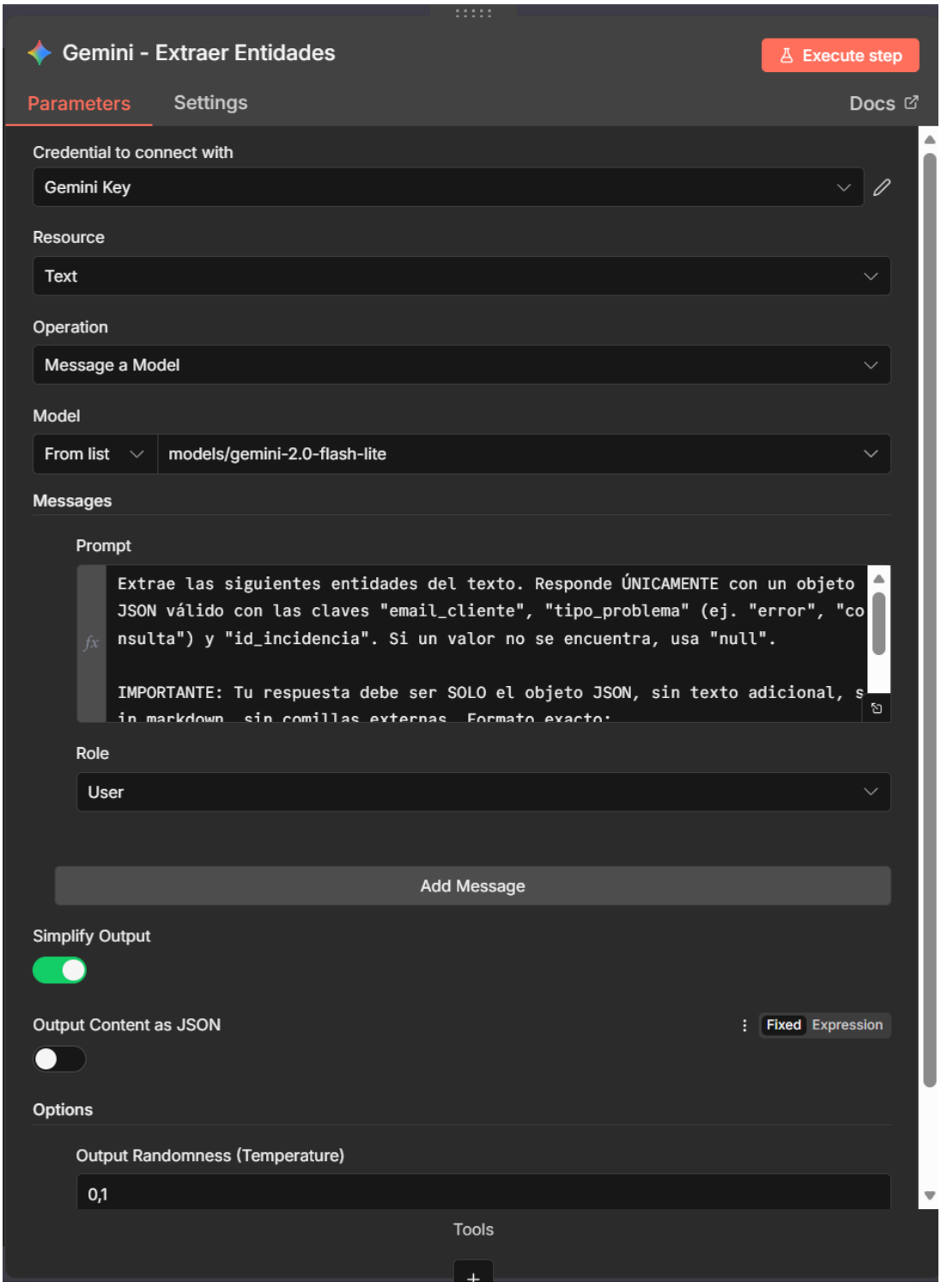

Figura 13: Prompt de extracción estructurada para Gemini

4. Parseo del JSON Se conectó un nodo "Edit Fields (Set)"para analizar el texto JSON devuelto por Gemini y crear campos separados:

- **email**:

```
{{ JSON.parse($json.content.parts[0].text).email_cliente }}
```

- **incidencia**:

```
{{ JSON.parse($json.content.parts[0].text).id_incidencia }}
```

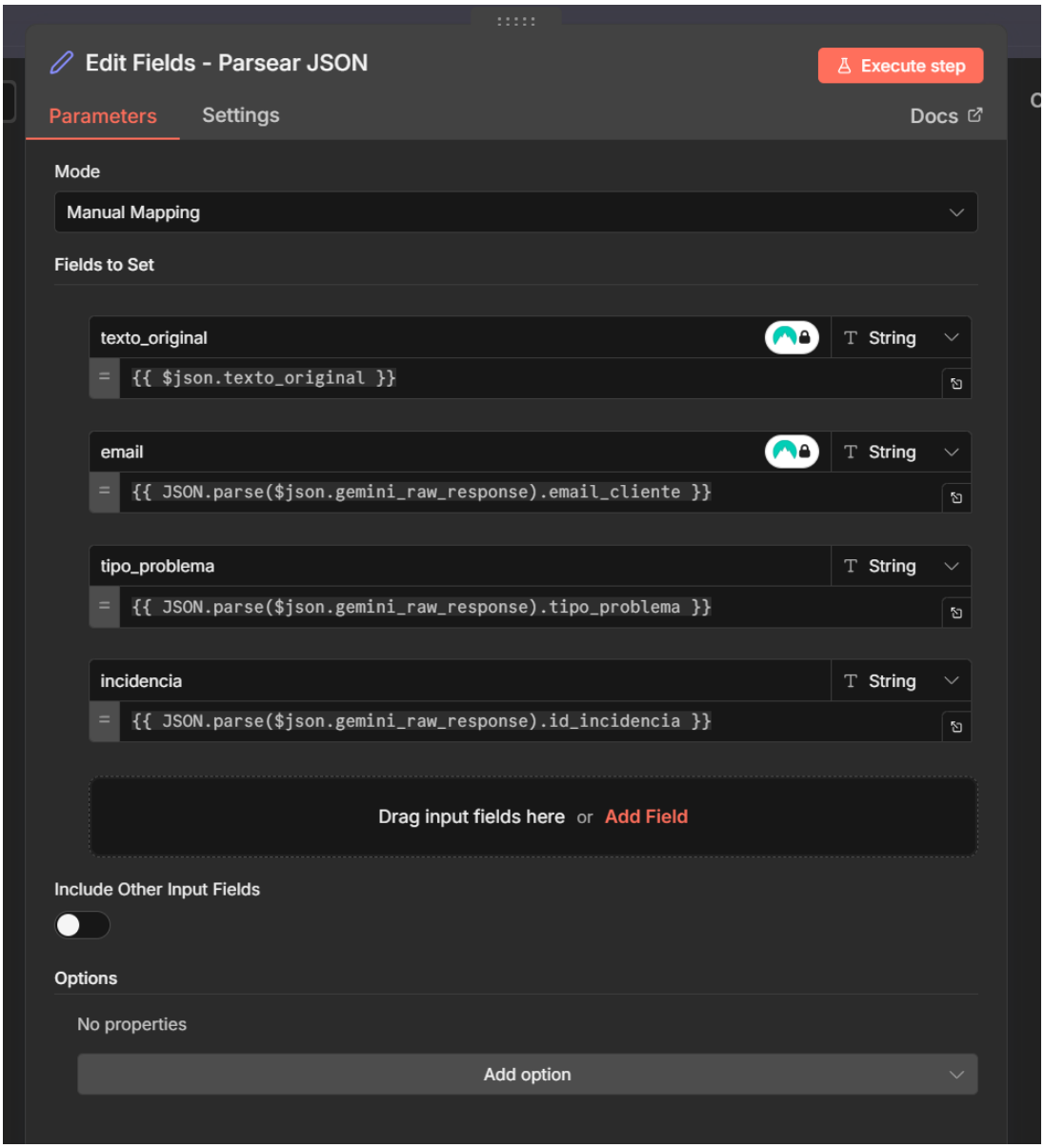

Figura 14: Parseo del JSON y extracción de campos individuales

5. Enrutamiento condicional Se añadió un nodo "Switch" para enrutar el flujo basándose en el dominio del email:

- **Routing Field**: {{ `$json.email` }}
- **Rule**: Contains "@ual.es"

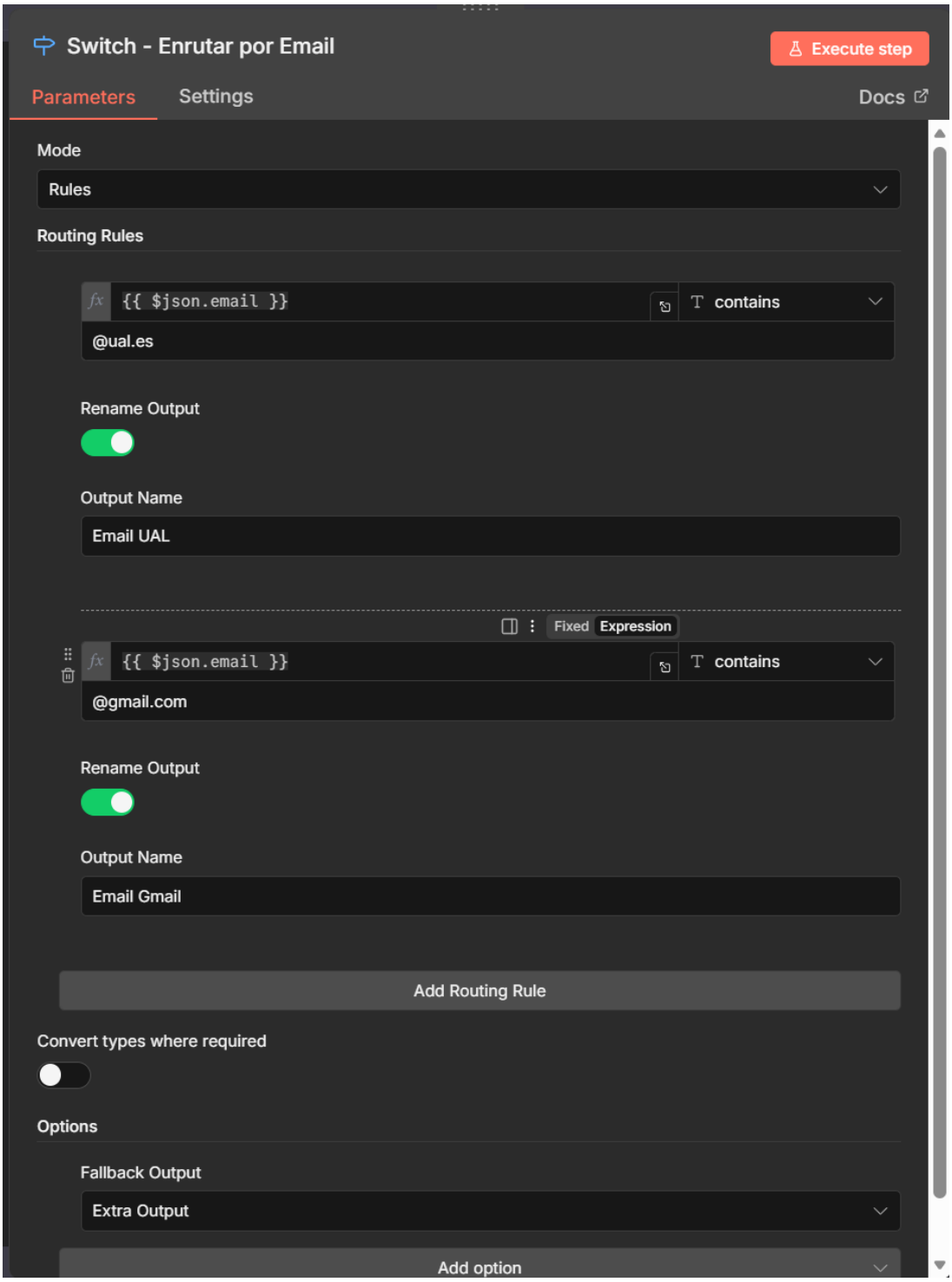

Figura 15: Switch para enrutamiento según dominio del email

3.2.3. Resultado

El flujo procesa exitosamente texto no estructurado y extrae información en formato JSON, demostrando el poder de los "prompts fuertes"para obtener salidas estructuradas de modelos de lenguaje.

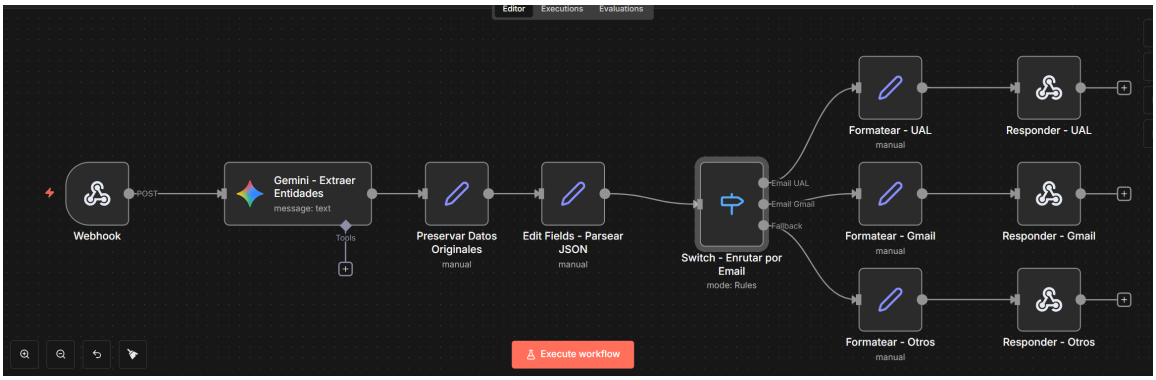

Figura 16: Flujo completo del Ejercicio 2

3.3. Ejercicio 3: Cadena de IA (AI Chain) (Dificultad: Alta)

3.3.1. Objetivo

Implementar un flujo donde la salida de una primera llamada a la IA se utiliza como entrada para una segunda llamada, demostrando el patrón de .[en]cadenamiento de IA".

3.3.2. Caso de uso

Generar múltiples títulos para un artículo de blog sobre un tema dado, y luego crear párrafos de introducción para cada uno de esos títulos.

3.3.3. Implementación

1. Inicialización del tema Se inició el flujo con:

- **Manual Trigger**
- **Edit Fields**: Define el tema base

```
{
  "tema": "Integracion de RabbitMQ en n8n"
}
```

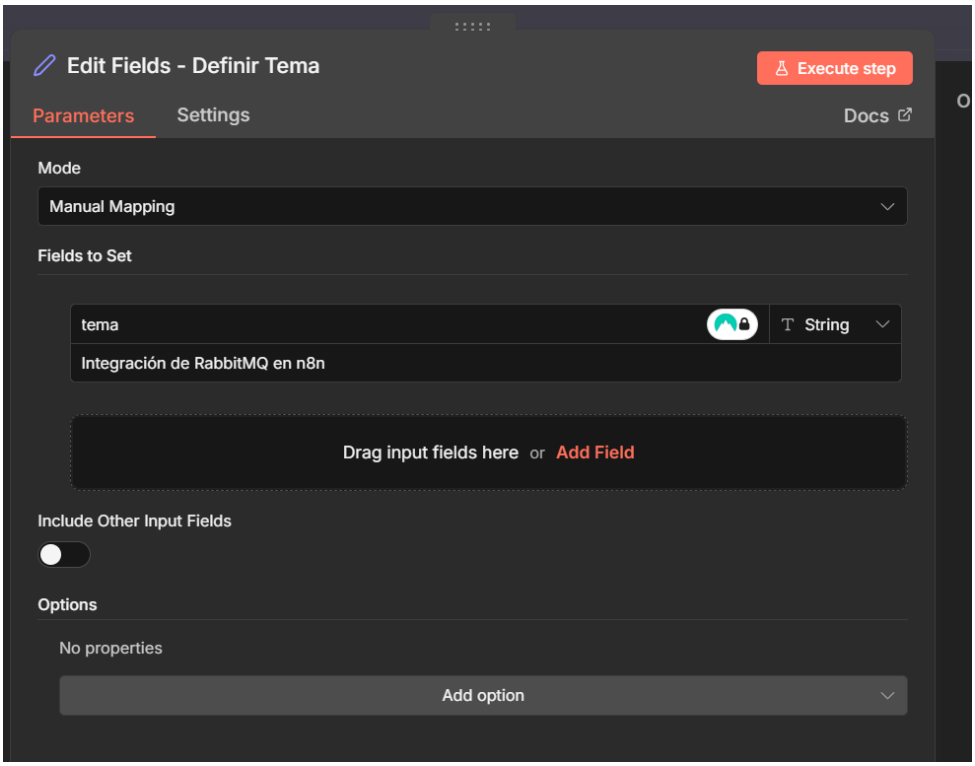

Figura 17: Inicialización del tema

2. Primera llamada a Gemini (Brainstorm) El primer nodo Gemini genera ideas de títulos:

```
Genera 3 ideas para un titulo de un articulo de blog
sobre el tema: '{{ $json.tema }}'.
Responde solo con una lista separada por el caracter '|'
```

Ejemplo de salida:

```
Integra RabbitMQ en n8n: Tu Guia Completa de
Automatizacion|Desbloquea el Poder de los Workflows:
Integrando n8n y RabbitMQ|RabbitMQ y n8n:
La Fusion Perfecta para una Automatizacion Eficiente
```

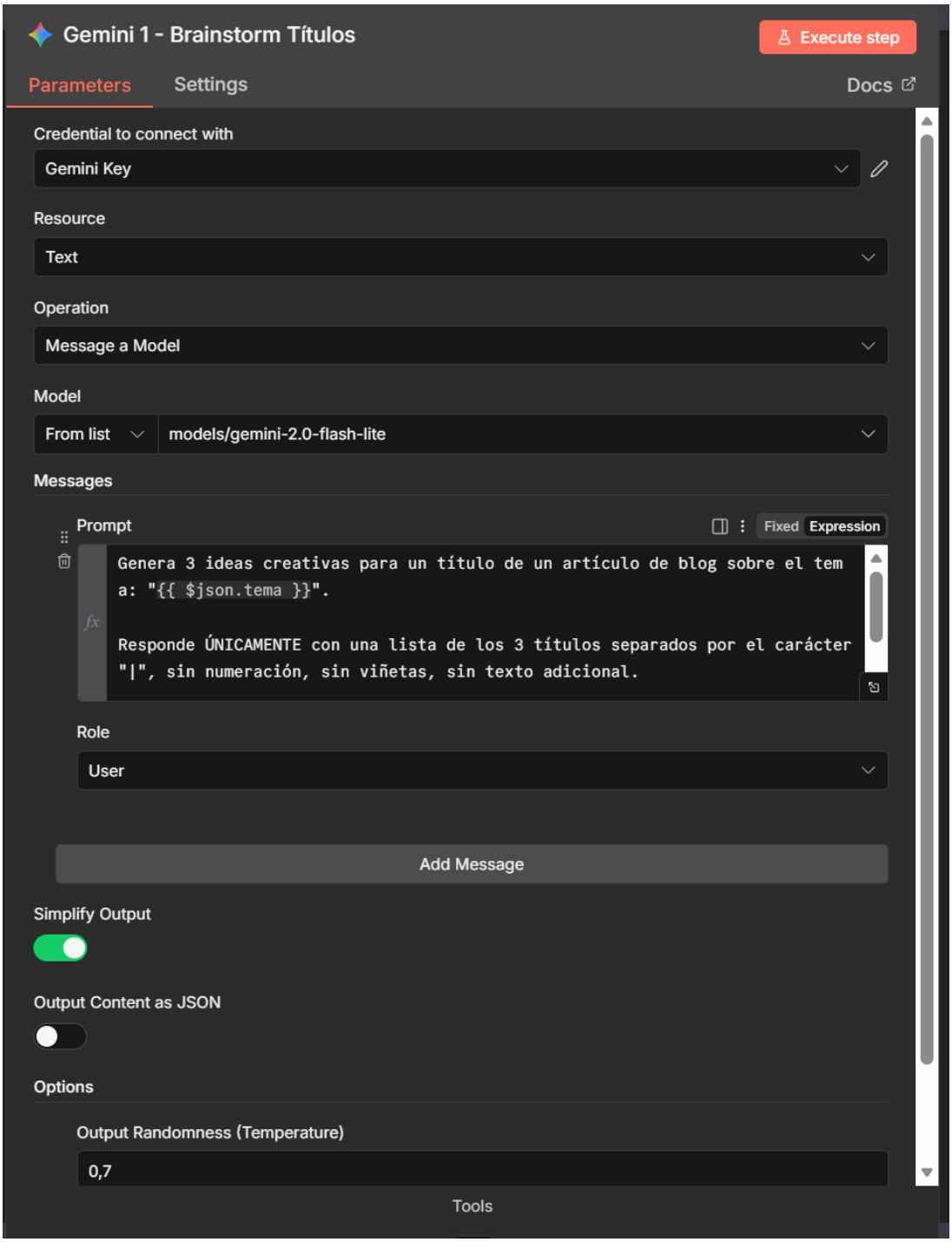

Figura 18: Primera llamada a Gemini para generar títulos

3. Split Out Se añadió un nodo "Split Out"para convertir el array en ítems individuales:

- **Fields To Split Out**: titulos

Esto genera 3 ítems separados, uno por cada título.

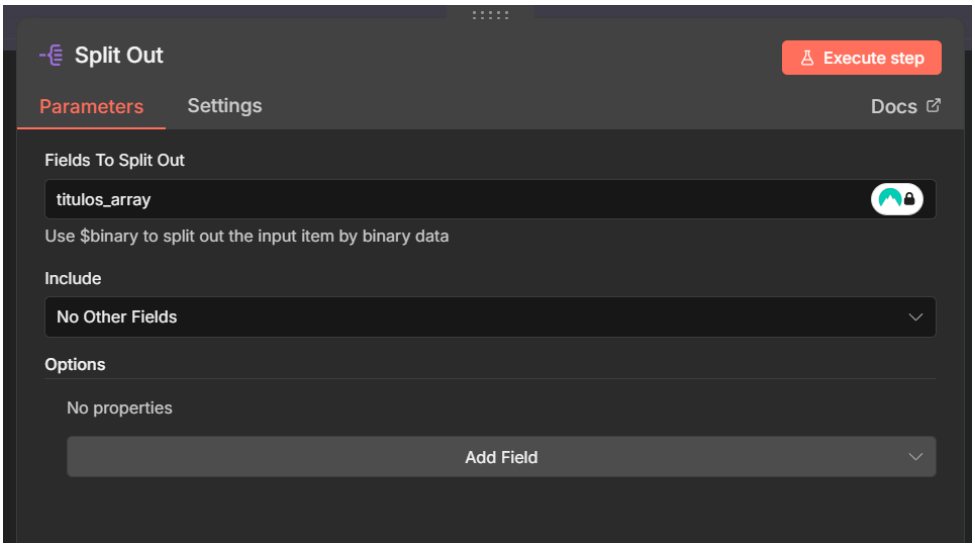

Figura 19: Split Out para separar títulos individuales

5. Segunda llamada a Gemini (Expand) Cada título se envía individualmente a un segundo nodo Gemini:

```
Escribe un parrafo de introduccion (3-4 frases)
para un articulo de blog con el siguiente titulo:
'{{ $json.titulos }}'
```

Este nodo procesa cada ítem por separado, generando un párrafo para cada título.

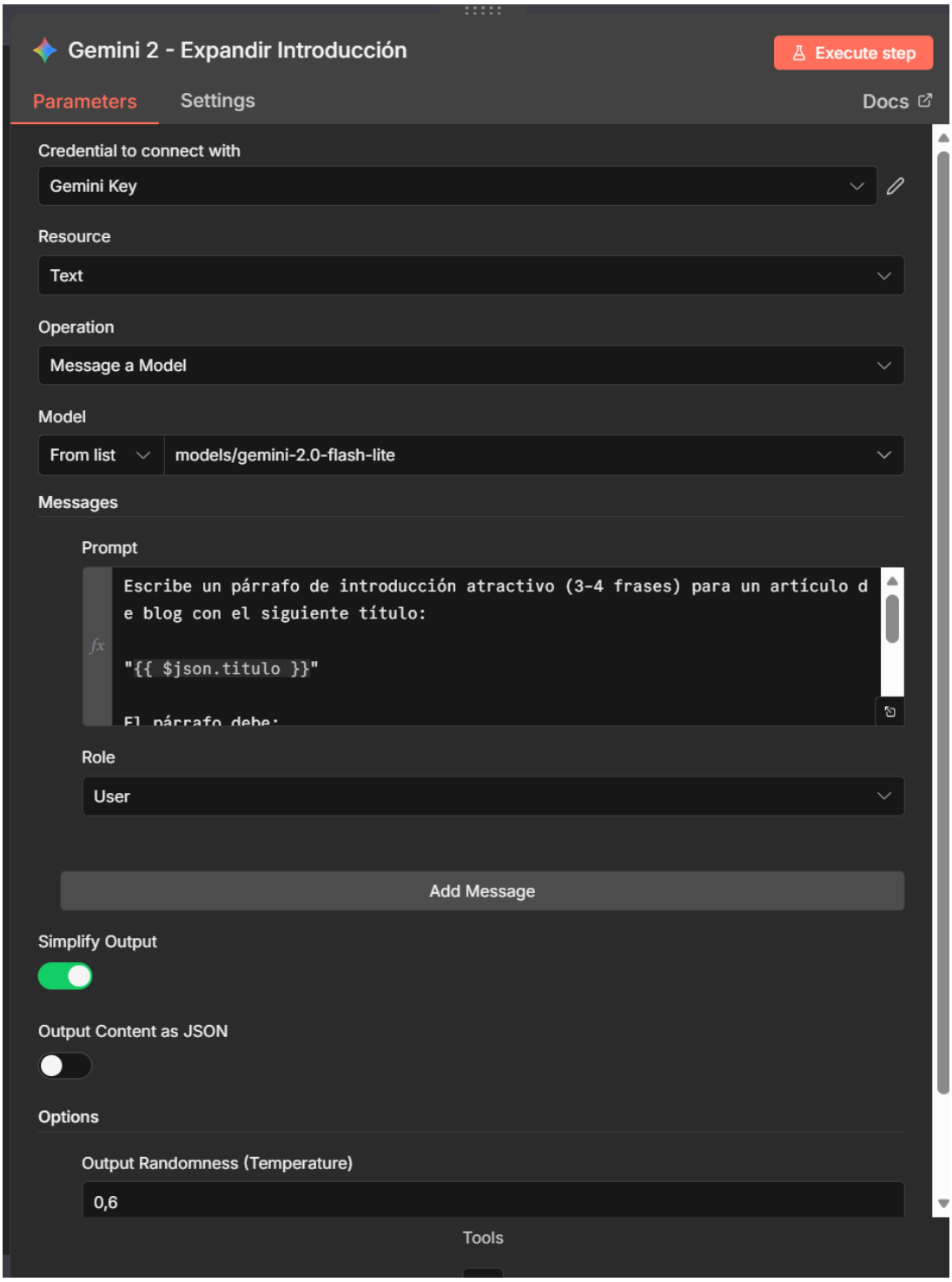

Figura 20: Segunda llamada a Gemini para expandir cada título

6. Merge Se conectó un nodo "Merge"para recopilar los tres párrafos generados:

- **Mode**: Append
- **Number of Inputs**: 2 (datos originales + párrafos generados)

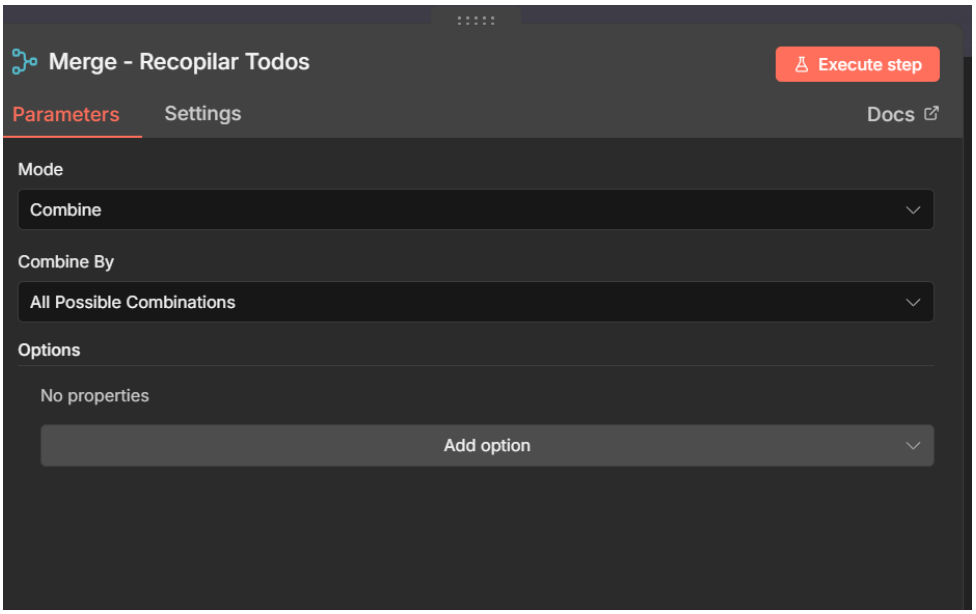

Figura 21: Merge para consolidar todos los resultados

3.3.4. Resultado

El flujo genera exitosamente:

1. 3 títulos creativos basados en el tema inicial
2. 3 párrafos de introducción, uno para cada título
3. Todos consolidados en un único conjunto de datos

Ejemplo de salida final:

```
[
  {
    "titulo": "Integra RabbitMQ en n8n: Tu Guia Completa...",
    "intro": "RabbitMQ es una poderosa herramienta de..."
  },
  {
    "titulo": "Desbloquea el Poder de los Workflows...",
    "intro": "En el mundo de la automatizacion..."
  },
  {
    "titulo": "RabbitMQ y n8n: La Fusion Perfecta...",
    "intro": "La integracion de sistemas es clave..."
  }
]
```

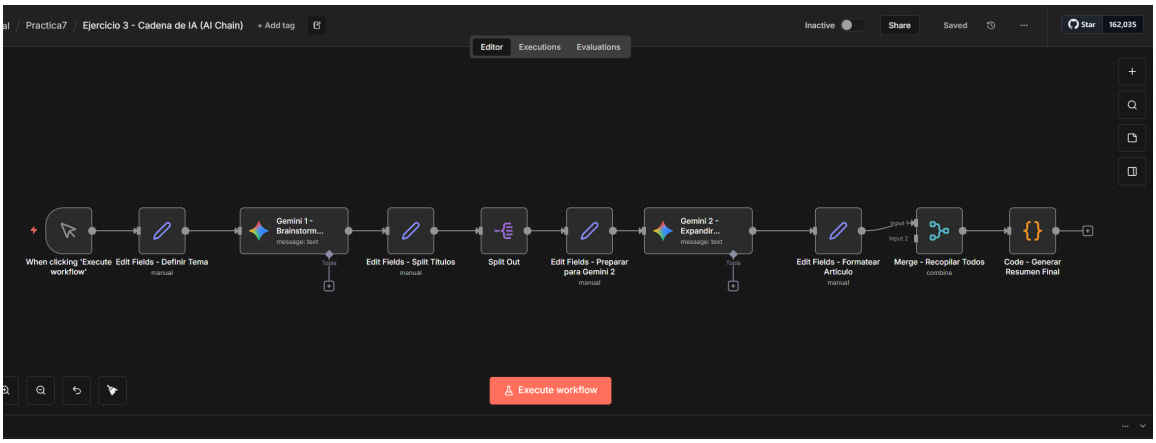

Figura 22: Flujo completo del Ejercicio 3 mostrando la cadena de IA

3.3.5. Patrón de diseño demostrado

Este ejercicio ilustra el patrón de .AI Chaining":

1. **Generación inicial**: Una IA genera múltiples opciones
2. **Procesamiento intermedio**: Los datos se transforman y dividen
3. **Expansión**: Cada opción se expande individualmente con otra llamada a IA
4. **Consolidación**: Los resultados se recopilan en un conjunto final

Este patrón es útil para:

- Generación de contenido a escala
- Refinamiento iterativo de ideas
- Procesamiento paralelo con IA
- Creación de variaciones sobre un tema base

4. Conclusiones

4.1. Aprendizajes clave

Esta práctica ha permitido comprender y aplicar varios conceptos fundamentales:

4.1.1. Integración de IA en flujos de trabajo

- La orquestación de modelos de lenguaje requiere un diseño cuidadoso de prompts
- La calidad de las respuestas depende directamente de la claridad y especificidad del prompt
- Es fundamental establecer formatos de salida estrictos cuando se necesita procesamiento automatizado

4.1.2. Patrones de diseño

Se han identificado y aplicado tres patrones principales:

1. **Clasificación simple**: IA → Decisión → Acción
   - Usado en: Flujo guiado de clasificación de prioridades
   - Ventaja: Implementación directa y resultados predecibles
2. **Extracción estructurada**: Texto no estructurado → IA → JSON → Procesamiento
   - Usado en: Ejercicio 2 de extracción de entidades
   - Ventaja: Convierte datos ambiguos en estructuras procesables
3. **Encadenamiento de IA**: IA1 → Procesamiento → IA2 → Consolidación
   - Usado en: Ejercicio 3 de generación de contenido
   - Ventaja: Permite workflows complejos con múltiples etapas de generación

4.2. Desafíos encontrados

4.2.1. Estructura de datos de Gemini

El principal desafío fue comprender la estructura de la respuesta de Google Gemini:

```
{
  "content": {
    "parts": [
      {
        "text": "ALTA"
      }
    ]
  }
}
```

Solución: Usar la expresión {{ `$json.content.parts[0].text` }} para acceder al texto de respuesta.

4.2.2. Manejo de formato JSON

En el Ejercicio 2, Gemini ocasionalmente incluía formato markdown:

```
‘‘‘json
{"email": "..."}
‘‘‘
```

Solución: Especificar explícitamente en el prompt "NO incluyas triples comillas de markdownz usar `JSON.parse()` para validar.

4.2.3. Concurrencia en AI Chains

En el Ejercicio 3, n8n procesa los ítems del Split Out secuencialmente, lo cual puede ser lento para grandes volúmenes.

Solución potencial: Usar el nodo "Loop Over Itemsçon procesamiento por lotes.

4.3. Aplicaciones prácticas

Los patrones implementados tienen aplicaciones directas en:

- **Automatización de tickets**: Clasificación automática de incidencias
- **Análisis de sentimientos**: Procesamiento de feedback de clientes
- **Generación de contenido**: Creación automatizada de documentación
- **Extracción de datos**: Digitalización de documentos no estructurados
- **Asistentes virtuales**: Orquestación de respuestas inteligentes

La orquestación de modelos de IA mediante herramientas como n8n democratiza el acceso a capacidades avanzadas de procesamiento de lenguaje natural. Los patrones aprendidos en esta práctica son fundamentales para construir sistemas empresariales inteligentes que combinen automatización tradicional con capacidades cognitivas de IA.

La clave del éxito radica en el diseño cuidadoso de los prompts y en la comprensión de cómo estructurar los flujos de datos para aprovechar al máximo las capacidades de los modelos de lenguaje.