

Memoria Práctica 7

Jaime Parra Jiménez

INTEGRACIÓN DE TECNOLOGÍAS Y SERVICIOS INFORMÁTICOS

Universidad de Almería

Correo: jpj451@inlumine.ual.es

1 de diciembre de 2025

Índice general

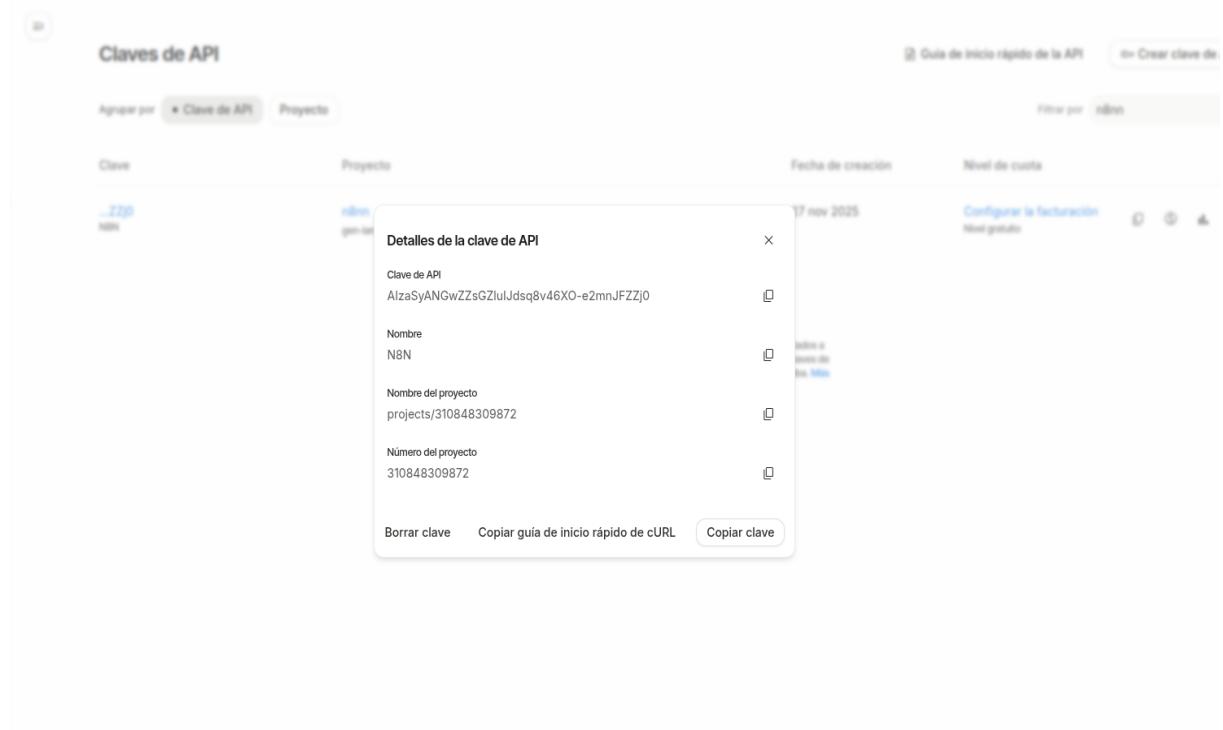
1. Ejercicio Guiado 1	2
2. Ejercicio 1	8
3. Ejercicio 2	11
4. Ejercicio 3	14

Capítulo 1

Ejercicio Guiado 1

El objetivo del ejercicio guiado es conectar la IA a nuestra base de datos de tareas de la Práctica 6. Crear un flujo que lea tareas sin analizar, usar Gemini para determinar su prioridad basándose en la descripción, y actualizar la base de datos.

El primer paso es acceder a Google AI para obtener una nueva clave de la API.

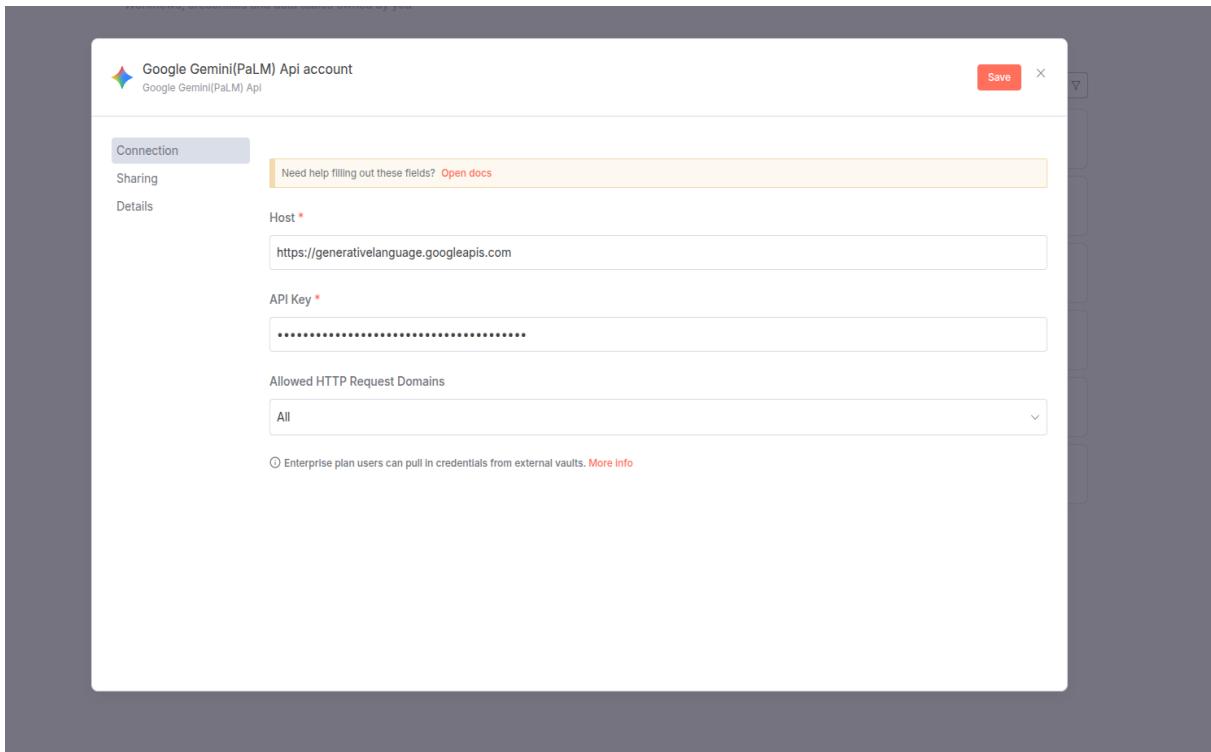


The screenshot shows the 'Claves de API' (API Keys) page in the Google Cloud console. A single API key is listed:

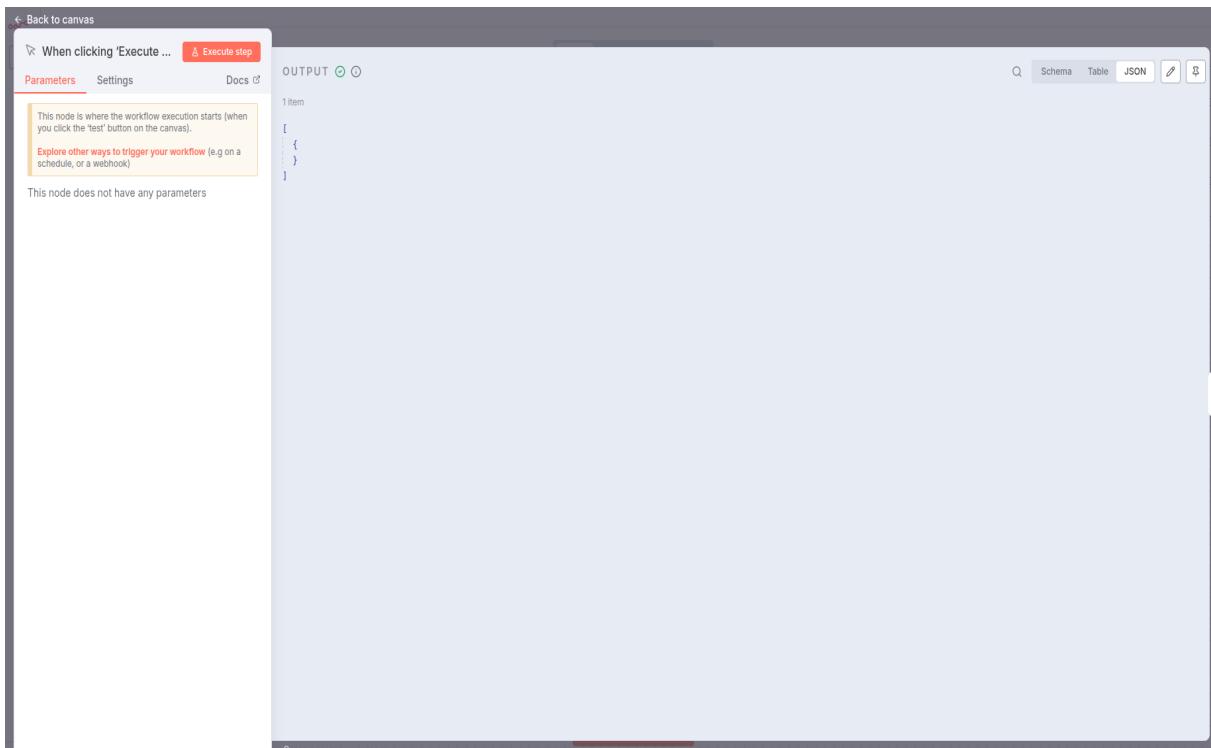
Clave	Proyecto	Fecha de creación	Nivel de cuenta
..._ZJ0 n8N	Detalles de la clave de API Clave de API: AlzaSyANGwZZsGZlulJdsq8v46XO-e2mnJFZZj0 Nombre: N8N Nombre del proyecto: projects/310848309872 Número del proyecto: 310848309872	17 nov 2025	Configurar la facturación Nivel gratuito

At the bottom of the key details, there are buttons: 'Borrar clave', 'Copiar guía de inicio rápido de cURL', and 'Copiar clave' (which is highlighted).

Con la clave ya generada, en n8n se añadirá una nueva credencial de Google Gemini API donde se copiará la API key generada.



A continuación, se crea el flujo dde trabajo. Primero se coloca un nodo Manual Trigger.



A este se le añade un nodo PostgreSQL para obtener las tareas de la base da datos task, que previamente debe estar iniciada.

The screenshot shows a 'Select rows from a table' step in a data integration tool. The input is a JSON array of task objects. The output shows 5 items, each a task object with fields like id, title, description, and done status.

```

[{"id": 2, "title": "Probar notifier", "description": "Desde curl", "done": false}, {"id": 3, "title": "Tarea para probar notifier", "description": "Probando envio al webhook", "done": false}, {"id": 4, "title": "Tarea B", "description": "Probando consumidor", "done": false}, {"id": 123, "title": "Tarea de prueba", "description": "Esto es una prueba", "done": true}, {"id": 45, "title": "Tarea de prueba", "description": "Esto es una prueba", "done": true}
]

```

Añadir un nodo Google Gemini al PostgreSQL para analizar las tareas. Se usará un prompt para que Gemini defina la prioridad de las tareas.

The screenshot shows a 'Message a model' step in a data integration tool. The input is the same JSON array of tasks. The output shows 5 items, each a message object with a content field containing a priority classification (ALTA, MEDIA, BAJA).

```

[{"content": {"parts": [{"text": "ALTA"}], "role": "model"}, {"content": {"parts": [{"text": "MEDIA"}], "role": "model"}, {"content": {"parts": [{"text": "BAJA"}], "role": "model"}, {"content": {"parts": [{"text": "ALTA"}], "role": "model"}, {"content": {"parts": [{"text": "ALTA"}], "role": "model"}}
]

```

Se añadirá un nodo merge para unificar la salida de Gemini con la de PostgreSQL.

The screenshot shows the Data Pipeline interface with a 'Merge' node selected. The input section displays a list of 5 items, each representing a task with fields like id, title, description, done, and content. The output section shows the result of merging these items, where tasks with the same id are combined into a single object, and their content is aggregated into an array of parts. The 'Mode' dropdown is set to 'Combine'.

Con un nodo Edit Fields (Set) se filtrarán los campos de ID, title y prioridad de cada tarea.

The screenshot shows the Data Pipeline interface with an 'Edit Fields' node selected. The input is the same list of tasks as before. The node configuration shows manual mapping for three fields: 'ID' (String, value: \$json.id), 'title' (String, value: \$json.title), and 'Prioridad' (String, value: \$json.content.parts[0].text). The output section shows the modified tasks, now including the 'ID', 'title', and 'Prioridad' fields.

El siguiente paso es conectar un nodo Switch para separar las tareas según su prioridad.

The screenshot shows the MuleSoft Anypoint Studio interface with a Switch node configuration. The input is a JSON array of tasks:

```
[{"ID": "2", "title": "Probar notifier", "Prioridad": "MEDIA"}, {"ID": "3", "title": "Tarea para probar notifier", "Prioridad": "ALTA"}, {"ID": "4", "title": "Tarea B", "Prioridad": "ALTA"}, {"ID": "123", "title": "Tarea de prueba", "Prioridad": "BAJA"}, {"ID": "45", "title": "Tarea de prueba", "Prioridad": "BAJA"}]
```

The Switch node has three output routes:

- ALTA** (Output 0): Tasks with Prioridad "ALTA".
- MEDIA** (Output 1): Tasks with Prioridad "MEDIA".
- BAJA** (Output 2): Tasks with Prioridad "BAJA".

Se conectará un nodo Send Email a la salida 0, las tareas con prioridad alta, para mandar un aviso de que la tarea ha sido calificada de prioridad alta por la IA.

The screenshot shows the MuleSoft Anypoint Studio interface with a Send email node configuration. The input is the output from the Switch node (Priority ALTA tasks).

The Send email node settings:

- Credential to connect with: SMTP account
- Operation: Send
- From Email: jpj451@inlumine.ual.es
- To Email: jpj451@inlumine.ual.es
- Subject: Alerta de Tarea Prioridad ALTA!
- Email Format: HTML
- HTML body (fixed expression): La tarea "{{ \$json.title }}" (ID: {{ \$json.ID }}) ha sido clasificada como ALTA prioridad por la IA.
- Options: No properties

The output is a JSON array of accepted and rejected emails:

```
[{"accepted": [{"to": "jpj451@inlumine.ual.es"}], "rejected": [{"to": ""}], "ehlo": [{"size": 35882577, "bitTime": "8BITTIME", "auth": "AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH", "enhancedStatusCodes": "ENHANCEDSTATUSCODES", "pipelining": "PIPELINING", "chunking": "CHUNKING", "smtpUTF8": "SMTPUTF8"}, {"envelopeTime": 309, "messageTime": 588, "messageSize": 766, "response": "250 2.8.0 OK 1764245064 ffacd0b85a97d-42e1c5d61813m3155618fef.18 - gsmtp", "envelope": {"from": "jpj451@inlumine.ual.es", "to": [{"to": "jpj451@inlumine.ual.es"}]}], "messageId": "<9570f18c-06a8-2b83-9e83-3c7b5616fc2@inlumine.ual.es>"}, {"accepted": [{"to": "jpj451@inlumine.ual.es"}], "rejected": [{"to": ""}], "ehlo": [{"size": 35882577}]}]
```

A las salidas se le conectarán un Merge para unificar las tareas de las tres salidas para poder actualizarlas en la base de datos.

```
[{"ID": "3", "title": "Tarea para probar notifier", "Prioridad": "ALTA"}, {"ID": "4", "title": "Tarea B", "Prioridad": "ALTA"}, {"ID": "2", "title": "Probar notifier", "Prioridad": "MEDIA"}, {"ID": "123", "title": "Tarea de prueba", "Prioridad": "BAJA"}, {"ID": "45", "title": "Tarea de prueba", "Prioridad": "BAJA"}]
```

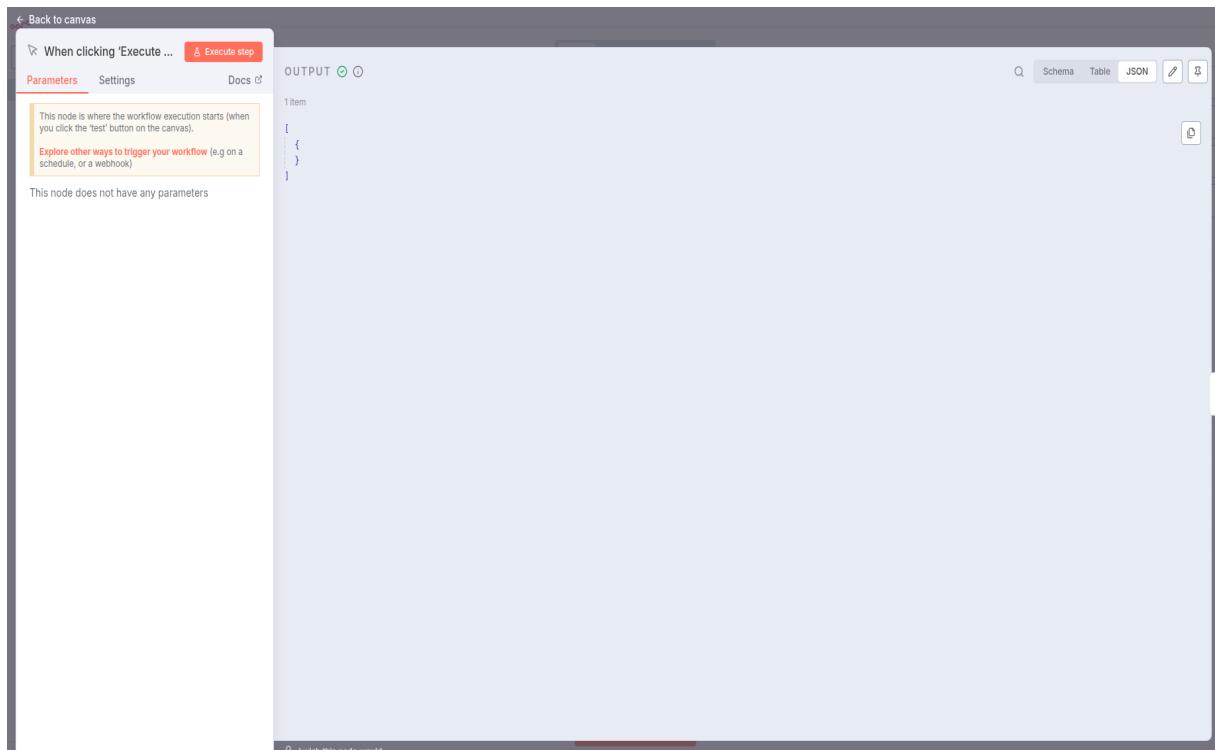
Por último, con un nodo PostgreSQL se actualizará la base de datos indicando la prioridad de cada tarea.

ID	Title	Prioridad	done
3	Tarea para probar notifier	ALTA	false
2	Probar notifier	MEDIA	false
123	Tarea de prueba	BAJA	false
45	Tarea de prueba	BAJA	false

Capítulo 2

Ejercicio 1

El objetivo del ejercicio 1 es modificar el flujo de trabajo guiado para que, en lugar de clasificar la prioridad, genere un resumen de la tarea. Para empezar con el flujo de trabajo se añade un nodo Manual Trigger.



A continuación se le conecta un nodo PostgreSQL para obtener las tareas de la base de datos.

The screenshot shows a data processing interface with the following details:

- INPUT:** A JSON array containing five items, each representing a task with fields like id, title, description, and done.
- Step Type:** "Select rows from a table".
- Parameters:**
 - Credential to connect with: Postgres account
 - Operation: Select
 - Schema: From list public
 - Table: From list task
 - Return All: Off
 - Limit: 5
- Select Rows:** Two sets of filters are applied:
 - Column: description, Operator: Is Not Null
 - Column: description, Operator: Not Equal, Value: (empty)
- OUTPUT:** The resulting JSON array contains five items, each representing a task that has not been completed (done: false).

El siguiente paso es modificar el prompt del ejercicio guiado para que en vez de definir la prioridad haga un resumen de la tarea de máximo 15 palabras.

The screenshot shows a data processing interface with the following details:

- INPUT:** A JSON array containing five items, each representing a task with fields like id, title, description, and done.
- Step Type:** "Message a model".
- Parameters:**
 - Credential to connect with: Google Gemini(PaLM) Api account
 - Resource: Text
 - Operation: Message a Model
 - Model: From list models/gemini-2.5-flash
 - Messages:
 - Prompt: "Genera un resumen conciso de una sola frase (máximo 15 palabras) para la siguiente tarea: {{ \$json.description }}"
 - Role: User
- OUTPUT:** The resulting JSON array contains five items, each representing a task with an additional content field containing a summary generated by the model.

A continuación se añade un nodo Merge para unificar el resumen a su respectiva tarea.

The screenshot shows the Gemini workspace interface. On the left, the INPUT panel displays a JSON array of five items, each representing a task with fields like ID, title, description, and done status. In the center, a 'Merge' node is selected, with its parameters set to 'Combine' mode and 'Number of Inputs' set to 2. The output of this merge is shown in the OUTPUT panel, where the tasks are combined into a single document. This document includes the tasks' details and adds a new 'Resumen' field containing a summary of the tasks' descriptions. A note at the bottom of the output panel says 'I wish this node would...'.

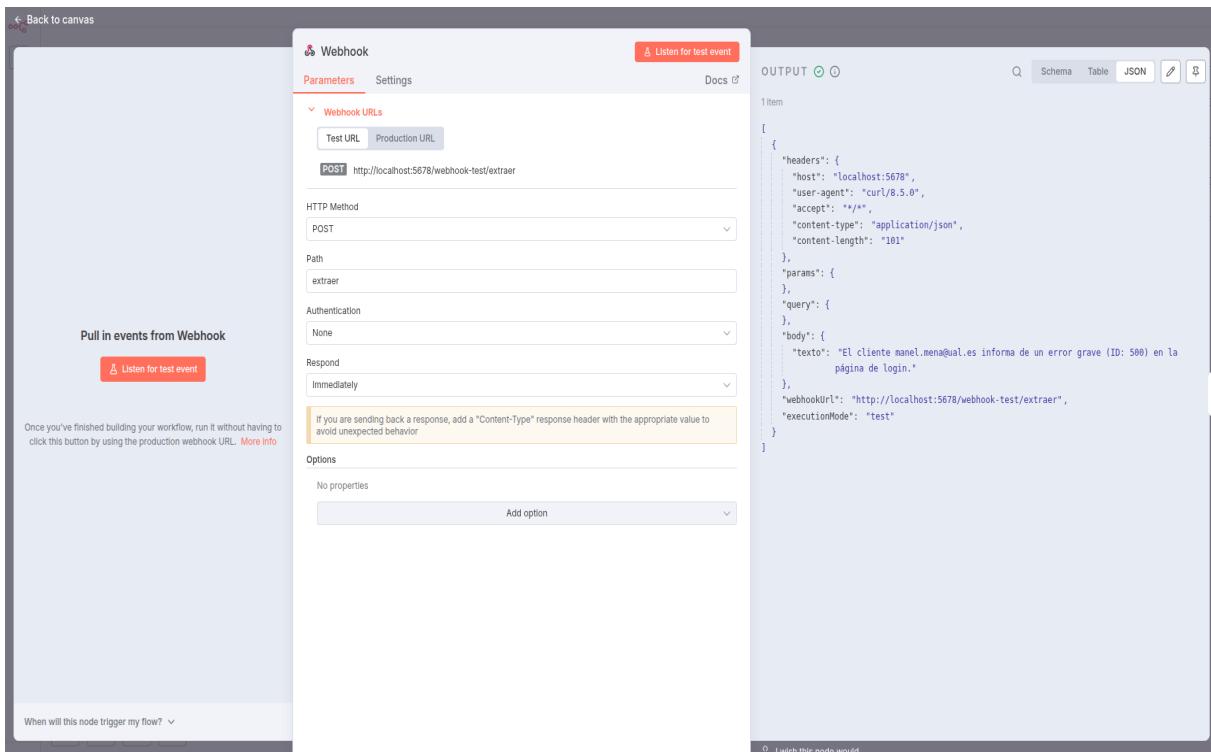
Por último, se añade un Edit Fields (Set) para añadir un campo resumen donde se guardará la salida de Gemini.

This screenshot shows the addition of an 'Edit Fields (Set)' node. The INPUT panel shows the same JSON array of tasks. The 'Edit Fields' node is configured with 'Manual Mapping' mode, mapping specific fields from the input to output. The output panel shows the tasks again, but now each task includes a new 'Resumen' field that contains a summary of the task's description. A note at the bottom of the output panel says 'I wish this node would...'.

Capítulo 3

Ejercicio 2

El objetivo del ejercicio 2 es crear un flujo que reciba texto no estructurado y extraiga datos en formato JSON, demostrando el patrón de "Prompt Fuerte". El primer paso es iniciar el flujo con un Webhook Trigger.



A continuación se añade un nodo Google Gemini donde se le pasará un prompt para que extraiga el email, el tipo de problema y la ID, y que responda únicamente en formato JSON.

The screenshot shows the 'Message a model' node configuration in MuleSoft Anypoint Studio. The input is a webhook message with a JSON payload. The output shows the generated JSON response from Google Gemini, which includes an email, a problem type, and an ID.

```

{
  "headers": {
    "host": "localhost:5678",
    "user-agent": "curl/8.5.0",
    "accept": "*/*",
    "content-type": "application/json",
    "content-length": "101"
  },
  "params": {},
  "query": {},
  "body": {
    "texto": "El cliente manel.mena@ual.es informa de un error grave (ID: 500) en la página de login."
  },
  "webhookUrl": "http://localhost:5678/webhook-test/extrair",
  "executionMode": "test"
}

```

Prompt:

- O USES formato de markdown.
- O uses bloques de código.
- O uses comillas triples.
- O uses json en ningún caso.
- Responde ÚNICAMENTE con un JSON puro y válido.

Role: User

Output Content as JSON: On

Output:

```

{
  "content": {
    "parts": [
      {
        "text": "\n\"email_cliente\":\n\"manel.mena@ual.es\",\\n\"tipo_problema\": \"error\ngrave en la página de login\",\\n\"id_incidencia\":\n500\\n\""
      }
    ],
    "role": "model"
  },
  "finishReason": "STOP",
  "index": 0
}

```

Se le conecta un nodo Edit Fields (Set) para crear los campos Email e Incidencia partir del JSON.

The screenshot shows the 'Edit Fields' node configuration in MuleSoft Anypoint Studio. It maps the JSON fields 'Email' and 'Incidencia' to specific values extracted from the JSON response of the previous node.

Fields to Set:

- Email: `JSON.parse($json.content.parts[0].text).email_cliente` (Value: manel.mena@ual.es)
- Incidencia: `JSON.parse($json.content.parts[0].text).id_incidencia` (Value: 500)

Output:

```

{
  "Email": "manel.mena@ual.es\\n",
  "Incidencia": "500\\n"
}

```

Por último, se añade un nodo Switch para enrutar el flujo basándose en el campo email.

The screenshot shows a data processing interface with a central 'Switch' node configuration window and two side panels: 'INPUT' and 'OUTPUT'.

INPUT Panel: Shows a JSON input document with one item containing fields 'Email' and 'Incidencia'.

Switch Node Configuration:

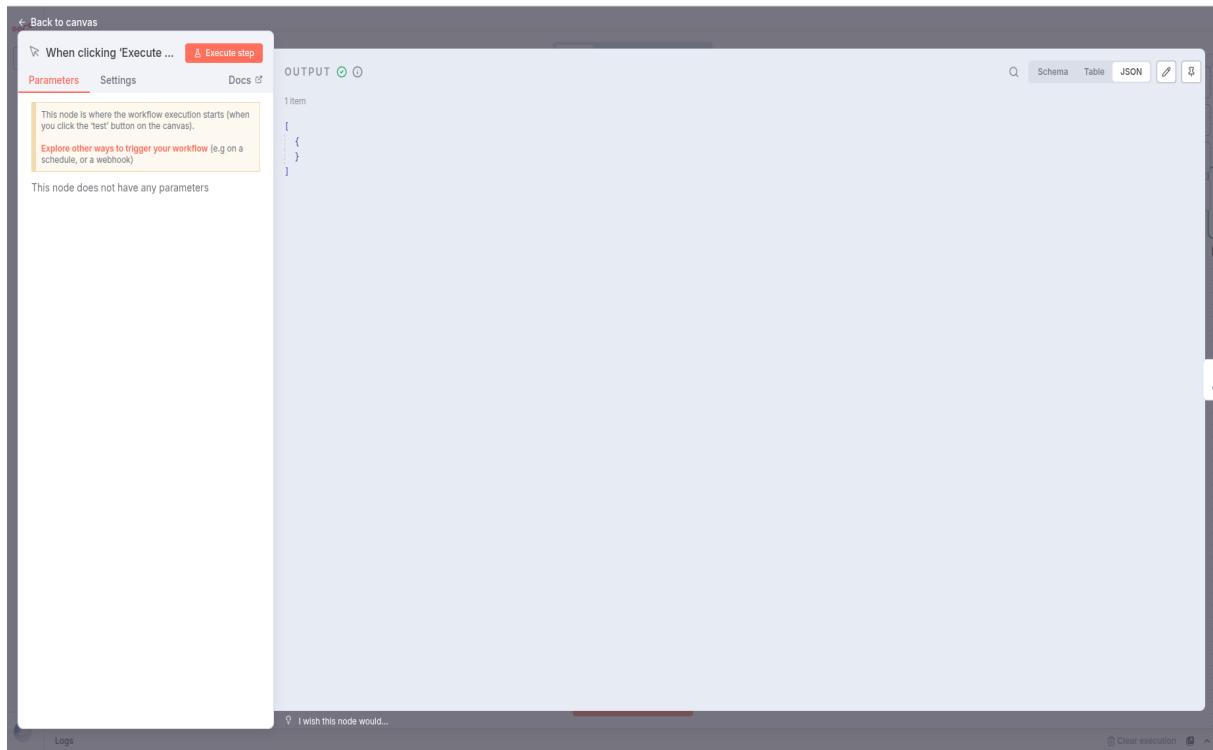
- Mode:** Set to 'Rules'.
- Routing Rules:** Contains two rules:
 - Rule 1: Condition `{{ $json.Email }}` contains `@ual.es`.
 - Rule 2: Condition `{{ $json.Email }}` contains `@gmail.es`.
- Outputs:** Two outputs are defined:
 - Output 1: Value `value1` if condition is met.
 - Output 2: Value `value2` if condition is met.
- Options:** Includes 'Add Routing Rule' and 'Convert types where required' buttons.

OUTPUT Panel: Shows the resulting output document with one item, which includes the original 'Email' and 'Incidencia' fields and the selected output value ('value1' or 'value2').

Capítulo 4

Ejercicio 3

El objetivo del ejercicio 3 es crear un flujo donde la salida de una llamada a la IA se utiliza como entrada para una segunda llamada a la IA. Para empezar el flujo de trabajo se añade un Manual Trigger.



El siguiente paso es añadir un nodo Edit Fields(Set) para definir el campo tema con el valor Integración de RabbitMQ en n8n.

Al Edit Fields se le conecta un nodo Google Gemini para que cree tres titulos para un artículo de blog.

Para separar los tres títulos y darle formato de JSON se usa un nodo Code de JavaScript.

```

Code in JavaScript
Parameters Settings
Mode Run Once for All Items
Language JavaScript
JavaScript
1 // Recogemos el texto generado por Gemini
2 const text = items[0].json.content.parts[0].text || "";
3
4 // Dividimos por el carácter | y limpiamos espacios/saltos de línea
5 const titulos = text
6 .split("|")
7 .map(t => t.trim())
8 .filter(t => t.length > 0);
9
10 // Devolvemos un único ítem con titulos como un ARRAY real
11 return [
12   {
13     json: {
14       titulos
15     }
16   }
17 ];

```

Type \$ for a list of **special vars/methods**. Debug by using `console.log()` statements and viewing their output in the browser console.

A continuación se usa un nodo Split Out para dividir el array en ítems individuales.

```

Split Out
Parameters Settings
Fields To Split Out
titulos
titulos
Include
No Other Fields
Options
No properties
Add Field

```

OUTPUT Schema Table JSON
3 items
[{"titulos": "Guía Paso a Paso: Integrando RabbitMQ en n8n"}, {"titulos": "Potencia tus Automatizaciones: Conectando RabbitMQ y n8n"}, {"titulos": "n8n y RabbitMQ: La Sinergia Perfecta para Flujos Asincronos"}]

Al Split Out se le conecta un segundo nodo Google Gemini para escribir un párrafo de unas 3 o 4 frases para cada título.

The screenshot shows the Meltano ETL interface. A central modal window titled "Message a model" is open. In the "Parameters" tab, the "Credential to connect with" is set to "Google Gemini(PaLM) Api account". The "Resource" is set to "Text", "Operation" is "Message a Model", and the "Model" is "From list" with "models/gemini-2.5-flash" selected. The "Messages" section contains a prompt: "Escribe un párrafo de introducción (3-4 frases) sin opciones ni nada, solamente una breve frase de introducción para un artículo de blog con el siguiente título: {{ \$json.titulos }} No pongas nada de frases ni nada, simplemente la introducción sin nada más". Below this, there's a "Role" dropdown set to "User". The "Output Content as JSON" toggle is off. The "Simplify Output" toggle is on. The "Options" section has a "No properties" message. The "Execute step" button is at the top right. To the left is an "INPUT" panel showing the JSON input, and to the right is an "OUTPUT" panel showing the resulting JSON output with two paragraphs generated by the model.

Además, se añade un nodo Merge para recopilar los tres párrafos generados.

This screenshot shows the "Merge" step in the Meltano interface. The "Mode" is set to "Combine" and "Position". The "Number of Inputs" is set to 2. There are no options or properties defined. The "Execute step" button is at the top right. The "INPUT" panel shows the same JSON array as before. The "OUTPUT" panel shows the merged JSON, which includes both paragraphs generated by the Gemini model, effectively combining them into a single document.

Por último, se añade un nodo Edits Fields(Set) para separar los campos de Título e Introducción.

The screenshot shows a data processing interface with two main panels: INPUT and OUTPUT.

INPUT: Shows a JSON object with several sections. One section is highlighted with a red border, containing the following text:

```

{
  "titulos": "Guía Paso a Paso: Integrando RabbitMQ en n8n",
  "content": [
    {
      "parts": [
        {
          "text": "En el dinámico mundo de la automatización, la gestión eficiente y fiable de los datos es clave para construir sistemas robustos. RabbitMQ, un potente broker de mensajes, se erige como una solución ideal para asegurar una comunicación asíncrona y resistente entre tus servicios. Cuando lo combinamos con la versatilidad de n8n, una herramienta de automatización de código abierto, las posibilidades de crear flujos de trabajo escalables y a prueba de fallos se multiplican. Esta guía te mostrará cómo integrar RabbitMQ en tus automatizaciones de n8n, paso a paso, para llevar tus proyectos al siguiente nivel."
        }
      ],
      "role": "model"
    },
    "finishReason": "STOP",
    "index": 0
  ],
  "titulos": "Potencia tus Automatizaciones: Conectando RabbitMQ y n8n",
  "content": [
    {
      "parts": [
        {
          "text": "En el dinámico mundo de la automatización, optimizar la comunicación entre tus sistemas y servicios es crucial para la eficiencia. Este artículo explora cómo la potente combinación de RabbitMQ, un robusto broker de mensajes, y n8n,"
        }
      ]
    }
  ]
}

```

OUTPUT: Shows the resulting JSON object after the 'Edit Fields1' step. The 'Titulos' and 'Introduccion' fields have been extracted and mapped to new fields in the JSON object.

```

{
  "Titulos": "Guía Paso a Paso: Integrando RabbitMQ en n8n",
  "Introduccion": "En el dinámico mundo de la automatización, la gestión eficiente y fiable de los datos es clave para construir sistemas robustos. RabbitMQ, un potente broker de mensajes, se erige como una solución ideal para asegurar una comunicación asíncrona y resistente entre tus servicios. Cuando lo combinamos con la versatilidad de n8n, una herramienta de automatización de código abierto, las posibilidades de crear flujos de trabajo escalables y a prueba de fallos se multiplican. Esta guía te mostrará cómo integrar RabbitMQ en tus automatizaciones de n8n, paso a paso, para llevar tus proyectos al siguiente nivel.",
  "Titulos": "Potencia tus Automatizaciones: Conectando RabbitMQ y n8n",
  "Introduccion": "En el dinámico mundo de la automatización, optimizar la comunicación entre tus sistemas y servicios es crucial para la eficiencia. Este artículo explora cómo la potente combinación de RabbitMQ, un robusto broker de mensajes, y n8n, una flexible herramienta de automatización, puede transformar tus flujos de trabajo. Descubre cómo integrar estas dos herramientas para construir automatizaciones más resistentes, escalables y capaces de manejar cualquier volumen de tareas. Prepárate para llevar tus procesos automatizados a un nivel superior de rendimiento y fiabilidad."
}

```