Administración de hilos
(unidades de ejecución más pequeñas dentro de un proceso)

```
const threadIds = await ThreadsTable.findRecords({ filter: { conversationId:
event.conversationId }, limit: 1 })
workflow.threadId = threadIds[0]?.threadId
```

const threadIds = await ThreadsTable.findRecords(...)

Esta línea es asíncrona, lo que significa que podría esperar a que se complete alguna operación antes de continuar.
Define una variable constante llamada threadIds.

const es una palabra clave que se utiliza para declarar una variable con un valor constante, es decir, una variable cuyo valor no puede ser reasignado una vez que se ha inicializado.

findRecords toma un argumento que es un objeto que contiene un filtro y un límite.

- El filtro especifica que está buscando hilos donde la propiedad conversationId coincide con el valor de event.conversationId. Este event.conversationId probablemente proviene de un evento externo o datos recibidos por el programa.

- La propiedad limit se establece en 1, lo que indica que la función solo debe devolver un máximo de un registro (hilo) que coincida con los criterios del filtro.

workflow.threadId = threadIds[0]?.threadId

Esta línea asigna un valor a una propiedad llamada threadId en un objeto llamado workflow.

(Un objeto es una estructura de datos que puede contener datos y funciones (llamadas métodos) que operan en esos datos.)

El valor que se asigna se obtiene de la constante threadIds. Sin embargo, aquí hay una parte interesante:

- threadIds[0] intenta acceder al primer elemento del arreglo threadIds (suponiendo que findRecords devuelve un arreglo de objetos de hilo).
- El operador ? se usa antes de .threadId. Este es un operador de coalescencia nula. Comprueba si threadIds[0] es nulo o indefinido. Si lo es, entonces la expresión después del ? (que es .threadId en este caso) no se evalúa y se asigna undefined a workflow.threadId. De lo contrario, se asigna el valor de threadId del primer elemento de threadIds.

En resumen, este fragmento de código recupera un máximo de un registro de hilo de una ThreadsTable basado en el ID de conversación recibido en un evento. Luego, asigna el threadId de ese hilo recuperado (si existe) a la propiedad threadId de un objeto workflow. Esto sugiere que el threadId podría ser importante para rastrear el hilo actual dentro del flujo de trabajo o la conversación.

Creación de hilos con Open Ai

```
if (!workflow.threadId) {
 let data = JSON.stringify({})

 let config = {
  method: 'post',
  maxBodyLength: Infinity,
  url: 'https://api.openai.com/v1/threads',
  headers: {
   'OpenAI-Beta': 'assistants=v1',
   Authorization: 'Bearer ' + env.api_key,
   'Content-Type': 'application/json'
  },
  data: data
 }

 const response = await axios.request(config)

 workflow.threadId = response.data.id
 await ThreadsTable.createRecord({
  threadId: workflow.threadId,
  conversationId: event.conversationId
 })
}
```

```
if (!workflow.threadId) {
```

if: Esta palabra clave introduce una instrucción condicional. El código dentro de las llaves {} solo se ejecutará si la condición después de if es verdadera.
!workflow.threadId: Esta es la condición que se verifica. El signo de exclamación (!) actúa como un operador lógico NO. Entonces, la condición verifica si workflow.threadId tiene un valor falso. En JavaScript, los valores falsos incluyen false, 0, "" (cadena vacía), null y undefined. En este caso, el código verifica si workflow.threadId aún no tiene un valor asignado (probablemente undefined).

```
 let data = JSON.stringify({})
```

let: Esta palabra clave declara una variable con alcance de bloque. La variable data solo será accesible dentro del bloque de código donde se declara (el código dentro de la instrucción if).
data = JSON.stringify({}): Esta línea asigna un valor a la variable data.

- JSON.stringify({}): Esta parte convierte un objeto JavaScript vacío {} en una cadena JSON. JSON (Notación de Objetos JavaScript) es un formato para almacenar y transmitir datos.

```
let config = {
  method: 'post',
  maxBodyLength: Infinity,
  url: 'https://api.openai.com/v1/threads',
  headers: {
   'OpenAI-Beta': 'assistants=v1',
   Authorization: 'Bearer ' + env.api_key,
   'Content-Type': 'application/json'
  },
  data: data
}
```

Similar a la línea 2, esta línea declara una variable llamada config con alcance de bloque y le asigna un objeto literal. El objeto define la configuración para realizar una solicitud HTTP.
Esta solicitud es necesaria para los API REST

Objeto de configuración (config)

- method: 'post': Esta propiedad especifica el método HTTP para la solicitud. Aquí, se establece en 'post', que generalmente se usa para enviar datos a un servidor.
- maxBodyLength: Infinity': Esta propiedad establece la longitud máxima permitida para el cuerpo de la solicitud (el data). Configurarla como Infinity elimina cualquier restricción en el tamaño de los datos.
- url: 'https://api.openai.com/v1/threads': Esta propiedad especifica el punto final de la URL para la solicitud de API. En este caso, apunta al punto final de la API de OpenAI para crear nuevos hilos.
- headers: Este objeto define los encabezados que se enviarán con la solicitud.
    - 'OpenAI-Beta': 'assistants=v1': Este encabezado podría ser un encabezado personalizado requerido por la API de OpenAI para una funcionalidad específica (probablemente relacionada con asistentes).
    - Authorization: 'Bearer ' + env.api_key: Este encabezado se utiliza para la autenticación. Incluye la cadena "Bearer " seguida del valor de una variable llamada env.api_key. Esta variable probablemente contenga su clave API de OpenAI almacenada en una variable de entorno.
    - 'Content-Type': 'application/json': Este encabezado especifica el formato de los datos que se envían en el cuerpo de la solicitud. Aquí, se establece en 'application/json', lo que indica que los datos son una cadena JSON.
- data: data: Esta propiedad asigna la cadena JSON creada previamente (data) al cuerpo de la solicitud.

5. const response = await axios.request(config)

- const: Esta palabra clave declara una variable llamada response con alcance de bloque. Contendrá los datos de respuesta de la llamada a la API.
- await: Esta palabra clave se usa porque axios.request probablemente es una función asíncrona. Las funciones asíncronas realizan operaciones que podrían tardar en completarse (como realizar una solicitud de red). La palabra clave await le indica al código que espere a que finalice la solicitud antes de continuar.
- axios.request(config): Esta parte llama a una función llamada axios.request (suponiendo que axios es una biblioteca para realizar solicitudes HTTP). La función se llama con el objeto config como argumento, que proporciona todos los detalles sobre la solicitud. El resultado de la llamada (la respuesta de la API) se almacena en la variable response.

```
const response = await axios.request(config)
```

Esta línea realiza una solicitud HTTP asíncrona a la API de OpenAI utilizando la función axios.request
El objeto config (definido anteriormente en el código) contiene todos los detalles sobre la solicitud, incluido el método (probablemente POST), la URL, los encabezados y los datos.
La palabra clave await se usa porque axios.request es asíncrona. Esto significa que el código espera la respuesta de la API antes de continuar.
El resultado de la llamada a la API (los datos de la respuesta) se almacena en la variable constante response.

```
 workflow.threadId = response.data.id
 await ThreadsTable.createRecord({
   threadId: workflow.threadId,
   conversationId: event.conversationId
 })
}
```

1. Asignar el ID del hilo al flujo de trabajo:
    - workflow.threadId = response.data.id:
        - Esta línea extrae la propiedad id de los datos de respuesta (response.data). Este id probablemente representa el identificador único para el hilo recién creado en OpenAI.
        - El id extraído luego se asigna a la propiedad threadId del objeto workflow. Esto asocia el hilo recién creado con el flujo de trabajo actual.
        -
    - await ThreadsTable.createRecord({ ... }):
        - Esta línea probablemente interactúa con una tabla de base de datos llamada ThreadsTable.

- Se llama a la función createRecord en ThreadsTable. Es probable que esta función se use para crear un nuevo registro en la tabla de la base de datos.
- Se pasa un objeto como argumento a la función createRecord. Este objeto define los datos para el nuevo registro:
  - threadId: Esta propiedad se establece en el valor del workflow.threadId (al que se le acaba de asignar el ID de la respuesta de OpenAI).
  - conversationId: Esta propiedad probablemente proviene de la variable event.conversationId (mencionada anteriormente en el código). Esto podría usarse para asociar el hilo con una conversación específica.
- La palabra clave await se usa nuevamente porque createRecord podría ser asíncrona (dependiendo de la implementación de la interacción con la base de datos).

En resumen, estas líneas de código hacen una llamada a la API para crear un nuevo hilo en OpenAI, almacena el ID del hilo en el objeto del flujo de trabajo y luego guardan la información del hilo junto con el ID de la conversación en una tabla de la base de datos. Esto ayuda a rastrear los hilos de conversación asociados con cada flujo de trabajo.

```
let data = JSON.stringify({
  role: 'user',
  content: event.preview
})

let config = {
  method: 'post',
  maxBodyLength: Infinity,
  url: `https://api.openai.com/v1/threads/${workflow.threadId}/messages`,
  headers: {
    'OpenAI-Beta': 'assistants=v1',
    Authorization: `Bearer ${env.api_key}`,
    'Content-Type': 'application/json'
  },
  data: data
}

await axios.request(config)
```

```
let data = JSON.stringify({role: 'user',
  content: event.preview
})
```

Preparando los datos del mensaje (JSON)
Esta línea crea un objeto literal de JavaScript y luego lo convierte en una cadena JSON usando JSON.stringify.
El objeto define los datos que se enviarán en el mensaje:

- role: 'user': Esta propiedad especifica que el mensaje proviene del usuario (a diferencia del asistente de inteligencia artificial).
- content: event.preview: Esta propiedad establece el contenido del mensaje. El valor proviene de event.preview, que probablemente contenga la entrada del usuario o una vista previa de su mensaje.

```
let config = {
 method: 'post',
 maxBodyLength: Infinity,
 url: `https://api.openai.com/v1/threads/${workflow.threadId}/messages`,
 headers: {
  'OpenAI-Beta': 'assistants=v1',
  Authorization: `Bearer ${env.api_key}`,
  'Content-Type': 'application/json'
 },
 data: data
}
```

Configuración de la solicitud HTTP:

Esta línea define un objeto literal de JavaScript que especifica la configuración para la solicitud HTTP.

method: 'post': Esta propiedad establece el método HTTP en 'post'. El método POST se usa típicamente para enviar datos a un servidor, lo cual es apropiado para enviar un mensaje.
maxBodyLength: Infinity': Esta propiedad establece la longitud máxima permitida para el cuerpo de la solicitud (el data) en Infinity. Esto elimina cualquier restricción en el tamaño del contenido del mensaje.
url: \https://api.openai.com/v1/threads/${workflow.threadId}/messages`...`:

- Esta propiedad define el punto final de la URL para la solicitud de API. Utiliza literales de plantilla (...) para insertar dinámicamente el valor de workflow.threadId en la URL. Esto probablemente crea una URL específica para el hilo de conversación asociado con el flujo de trabajo actual. La URL apunta al punto final de la API de OpenAI para enviar mensajes a hilos existentes.

headers: Este objeto define los encabezados que se incluirán en la solicitud:

- 'OpenAI-Beta': 'assistants=v1': Este podría ser un encabezado personalizado requerido por la API de OpenAI para una funcionalidad específica relacionada con asistentes.
- Authorization: 'Bearer ${env.api_key}': Este encabezado se utiliza para la autenticación. Incluye la cadena "Bearer " seguida de un literal de plantilla que inserta el valor de la variable env.api_key. Esta variable probablemente contenga su clave API de OpenAI almacenada en una variable de entorno.
- 'Content-Type': 'application/json': Este encabezado especifica el formato de los datos que se envían en el cuerpo de la solicitud. Aquí, se establece en 'application/json', lo que indica que los datos son una cadena JSON (la cual creamos anteriormente en la variable data).

```
await axios.request(config)
```

Enviando la solicitud

Esta línea envía la solicitud HTTP usando axios.request (suponiendo que axios es una biblioteca para realizar solicitudes HTTP).
El objeto config que definimos anteriormente contiene todos los detalles sobre la solicitud.
La palabra clave await se usa porque axios.request probablemente es asíncrona, lo que significa que podría tardar un tiempo en recibir una respuesta de la API. El código espera la respuesta antes de continuar.

```
const creationResponse = await axios.request({
  method: 'post',
  maxBodyLength: Infinity,
  url: `https://api.openai.com/v1/threads/${workflow.threadId}/runs`,
  headers: {
    'OpenAI-Beta': 'assistants=v1',
    Authorization: `Bearer ${env.api_key}`,
    'Content-Type': 'application/json'
  },
  data: JSON.stringify({
    assistant_id: env.assistant_id
  })
})

const runId = creationResponse.data.id

const waitTillRunComplete = async () => {
```

```
  const statusResponse = await axios.request({
    method: 'get',
    maxBodyLength: Infinity,
    url: `https://api.openai.com/v1/threads/${workflow.threadId}/runs/${runId}`,
    headers: {
      'OpenAI-Beta': 'assistants=v1',
      Authorization: `Bearer ${env.api_key}`,
      'Content-Type': 'application/json'
    }
  })

  if (['queued', 'in_progress'].includes(statusResponse.data.status) === false) {
    console.log('the status is:', statusResponse.data.status)
    return
  }
  await new Promise((resolve) => {
    setTimeout(resolve, 1000)
  })
  await waitTillRunComplete()
}

await waitTillRunComplete()
```

```
const creationResponse = await axios.request({
  method: 'post',
  maxBodyLength: Infinity,
  url: `https://api.openai.com/v1/threads/${workflow.threadId}/runs`,
  headers: {
    'OpenAI-Beta': 'assistants=v1',
    Authorization: `Bearer ${env.api_key}`,
    'Content-Type': 'application/json'
  },
  data: JSON.stringify({
    assistant_id: env.assistant_id
  })
})
```

Creando una ejecución de OpenAI:

- const creationResponse = await axios.request({ ... }):

- Esta línea envía una solicitud HTTP POST a la API de OpenAI para crear una nueva "ejecución".
- El method se establece en 'post', lo que indica que se trata de una solicitud de envío de datos.
- maxBodyLength se establece en Infinity, permitiendo grandes cantidades de datos.
- La url se construye usando un literal de plantilla para incluir dinámicamente el workflow.threadId. Esto probablemente crea una URL única para la ejecución asociada con el flujo de trabajo actual.
- Los headers contienen la información de autenticación necesaria y especifican el tipo de contenido como JSON.
- Los data son una cadena JSON que contiene el assistant_id, que probablemente sea un identificador para su asistente específico de OpenAI.

```
const runId = creationResponse.data.id
```

Extrayendo el ID de ejecución:

- const runId = creationResponse.data.id:
  - Esta línea extrae la propiedad id de los datos de respuesta, que representa el identificador único para la ejecución recién creada.
  - Este runId se usará en pasos posteriores para rastrear y administrar la ejecución.

```
onst waitTillRunComplete = async () => {
```

Definiendo una función para esperar la finalización de la ejecución:

- const waitTillRunComplete = async () => { ... }:
  - Esto define una función asíncrona llamada waitTillRunComplete que verifica repetidamente el estado de la ejecución de OpenAI hasta que esté completa.

```
const statusResponse = await axios.request({
  method: 'get',
  maxBodyLength: Infinity,
  url: `https://api.openai.com/v1/threads/${workflow.threadId}/runs/${runId}`,
  headers: {
    'OpenAI-Beta': 'assistants=v1',
    Authorization: `Bearer ${env.api_key}`,
    'Content-Type': 'application/json'
  }
})
```

Verificando el estado de la ejecución:

- const statusResponse = await axios.request({ ... }):
    - Dentro de waitTillRunComplete, esta línea envía una solicitud HTTP GET a la API de OpenAI para verificar el estado de la ejecución utilizando el runId.
    - El method se establece en 'get', ya que está recuperando datos.
    - La url se construye usando un literal de plantilla para incluir el runId.
    - Los headers contienen la información de autenticación necesaria.

```
if (['queued', 'in_progress'].includes(statusResponse.data.status) === false) {
  console.log('the status is:', statusResponse.data.status)
  return
}
```

Evaluando el estado de la ejecución:

- if (['queued', 'in_progress'].includes(statusResponse.data.status) === false) { ... }:
    - Esto verifica si el estado de la ejecución es 'queued' o 'in_progress'. Si no lo es, significa que la ejecución ha finalizado (con éxito o con un error).
        - Si el estado no es 'queued' o 'in_progress', el código registra el estado actual y sale de la función waitTillRunComplete.

```
await new Promise((resolve) => {
  setTimeout(resolve, 1000)
})
await waitTillRunComplete()
```

Retrasando y verificando el estado de forma recursiva:

await new Promise((resolve) => { setTimeout(resolve, 1000) }):

Esta línea introduce un retraso de 1 segundo utilizando una Promise y setTimeout. Esto le permite a la ejecución tener algo de tiempo para avanzar antes de volver a verificar su estado.

await waitTillRunComplete():

Esta línea llama recursivamente a la función waitTillRunComplete, creando efectivamente un bucle que verifica repetidamente el estado de la ejecución hasta que finalice.

    -

```
await waitTillRunComplete()
```

Iniciando la verificación de finalización de ejecución:

- await waitTillRunComplete():
    - Esta línea fuera de la función waitTillRunComplete llama a la función, comenzando el proceso de monitorear la ejecución hasta que esté completa.

Este fragmento de código gestiona la creación y el monitoreo de ejecuciones de OpenAI para la generación de texto. Crea una nueva ejecución usando el assistant_id, extrae el ID de ejecución y luego verifica repetidamente el estado de la ejecución hasta que finaliza. Esto permite que el programa espere el texto generado antes de continuar con otras acciones.

```
const creationResponse = await axios.request({
 method: 'post',
 maxBodyLength: Infinity,
 url: `https://api.openai.com/v1/threads/${workflow.threadId}/runs`,
 headers: {
  'OpenAI-Beta': 'assistants=v1',
  Authorization: `Bearer ${env.api_key}`,
  'Content-Type': 'application/json'
 },
 data: JSON.stringify({
  assistant_id: env.assistant_id
 })
})

const runId = creationResponse.data.id

const waitTillRunComplete = async () => {
 const statusResponse = await axios.request({
```

```
  method: 'get',
  maxBodyLength: Infinity,
  url: `https://api.openai.com/v1/threads/${workflow.threadId}/runs/${runId}`,
  headers: {
    'OpenAI-Beta': 'assistants=v1',
    Authorization: `Bearer ${env.api_key}`,
    'Content-Type': 'application/json'
  }
 })


 if (['queued', 'in_progress'].includes(statusResponse.data.status) === false) {
  console.log('the status is:', statusResponse.data.status)
  return
 }
 await new Promise((resolve) => {
  setTimeout(resolve, 1000)
 })
 await waitTillRunComplete()
}


await waitTillRunComplete()
```

Creando una ejecución de OpenAI (Solicitud POST Asíncrona):

- const creationResponse = await axios.request({ ... }):
    - Esta línea envía una solicitud HTTP POST asíncrona a la API de OpenAI usando axios.request.
    - method: 'post': Especifica el método POST para enviar datos.
    - maxBodyLength: Infinity: Permite enviar cantidades potencialmente grandes de datos en el cuerpo de la solicitud.
    - url: \https://api.openai.com/v1/threads/${workflow.threadId}/runs` ...`**:
        - Construye una URL dinámica usando un literal de plantilla.
        - https://api.openai.com/v1/threads/: El punto final de la API de OpenAI para crear ejecuciones.
        - ${workflow.threadId}: Inserta el ID del hilo del flujo de trabajo actual para crear la ejecución dentro de la conversación específica.
        - /runs: La ruta específica para crear ejecuciones dentro de un hilo.
    - headers:
        - 'OpenAI-Beta': 'assistants=v1': Podría ser un encabezado personalizado para la funcionalidad relacionada con asistentes.
        - Authorization: 'Bearer ${env.api_key}': Encabezado de autenticación con su clave API de OpenAI almacenada en env.api_key (probablemente una variable de entorno).
        - 'Content-Type': 'application/json': Especifica el formato de datos JSON en el cuerpo de la solicitud.
    - data: JSON.stringify({ assistant_id: env.assistant_id })**:
        - Prepara los datos del cuerpo de la solicitud como una cadena JSON.
        - assistant_id: Probablemente el identificador específico de su asistente de OpenAI, obtenido de env.assistant_id.
    - await: Asegura que el programa espere la respuesta asíncrona antes de continuar.

Extrayendo el ID de la ejecución (Procesamiento de la respuesta):

- const runId = creationResponse.data.id:
    - Extrae la propiedad id de la respuesta exitosa de la solicitud POST (creationResponse.data).
    - Este runId identifica de manera única la ejecución de OpenAI recién creada.

Definiendo la función waitTillRunComplete (Función asíncrona):

- const waitTillRunComplete = async () => { ... }:
    - Define una función asíncrona llamada waitTillRunComplete.
    - Las funciones asíncronas pueden usar await para esperar operaciones asíncronas.

Comprobando el estado de la ejecución (Solicitud GET Asíncrona):

- const statusResponse = await axios.request({ ... }):
    - Envía una solicitud HTTP GET asíncrona para recuperar el estado de la ejecución.

    - method: 'get': Especifica el método GET para recuperar datos.
    - url: \https://api.openai.com/v1/threads/workflow.threadId/runs/{runId}` ...**: Construye una URL dinámica similar a la anterior, pero usandorunId` para consultar el estado de la ejecución específica.
    - headers: Los mismos encabezados que en la solicitud POST para autenticación y tipo de contenido.
    - await: Espera la respuesta de la solicitud GET.

Evaluando el estado de la ejecución (Salida condicional):

- if (['queued', 'in_progress'].includes(statusResponse.data.status) === false) { ... }:
    - Comprueba si el estado de la ejecución (statusResponse.data.status) es 'queued' o 'in_progress'.
    - Si no lo es, significa que la ejecución ha finalizado (con éxito o con un error).
        - console.log('the status is:', statusResponse.data.status): Registra el estado real para fines de depuración o

```
const response = await axios.request({

  method: 'get',

  maxBodyLength: Infinity,

  url:
`https://api.openai.com/v1/threads/${workflow.threadId}/messages`,

  headers: {

    'OpenAI-Beta': 'assistants=v1',

    Authorization: `Bearer ${env.api_key}`,

    'Content-Type': 'application/json'

  }

})



workflow.response = response.data.data[0].content[0].text.value
```

¡Claro! Aquí está el texto sin números:

Enviando una solicitud HTTP GET:

`const response = await axios.request({ ... })`

Esta línea envía una solicitud HTTP GET asíncrona usando `axios.request`.

`await`: Asegura que el programa espere la respuesta antes de continuar.

Configuración de la solicitud:

`method: 'get'`

Especifica el método GET para recuperar datos.

`maxBodyLength: Infinity`

Permite que el cuerpo de la respuesta tenga una cantidad grande de datos (potencialmente).

`url: \https://api.openai.com/v1/threads/${workflow.threadId}/messages…`

Construye una URL dinámica usando un literal de plantilla.

`https://api.openai.com/v1/threads/`: El punto final de la API de OpenAI para acceder a mensajes en un hilo.

`${workflow.threadId}`: Inserta el ID del hilo del flujo de trabajo actual para recuperar mensajes de esa conversación específica.

`/messages`: La ruta para acceder a los mensajes dentro de un hilo.

`headers`:

`'OpenAI-Beta': 'assistants=v1'`

Podría ser un encabezado personalizado para la funcionalidad relacionada con asistentes.

`Authorization: 'Bearer ${env.api_key}'`

Encabezado de autenticación con tu clave API de OpenAI almacenada en `env.api_key`.

`'Content-Type': 'application/json'`

Especifica el formato de datos JSON, potencialmente utilizado para futuras solicitudes (esta solicitud recupera datos, no los envía).

Extrayendo el contenido del mensaje:

`workflow.response = response.data.data[0].content[0].text.value`

Asigna el contenido del mensaje recuperado a la variable `workflow.response`.

`response.data.data`: Accede a la lista de datos de la respuesta.

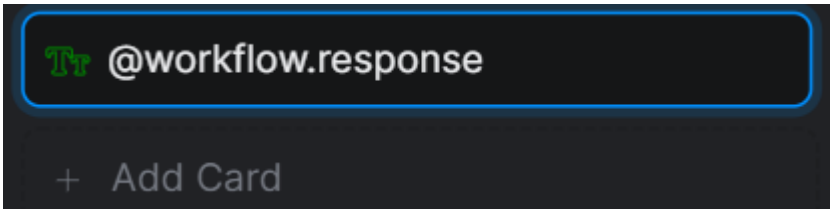
`[0]`: Selecciona el primer mensaje (índice 0) asumiendo que solo quieres el mensaje más reciente.

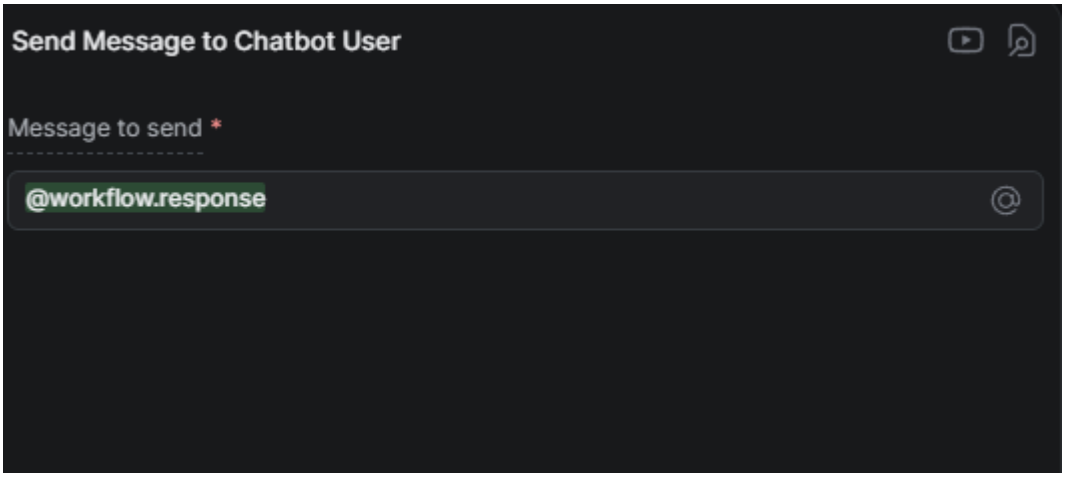
`content[0]`: Selecciona el primer elemento de contenido (podría estar anidado dependiendo de la estructura de la respuesta de la API).

`text.value`: Extrae el valor de texto real de la estructura de contenido anidada.

Funcionalidad general:

Este código recupera mensajes de una conversación de OpenAI y almacena el contenido del primer mensaje (asumiendo que quieres el más reciente) en la variable `workflow.response`. Esta variable puede ser utilizada por tu flujo de trabajo para procesar posteriormente o realizar acciones basadas en el mensaje recibido.





Último módulo del bot

Configuración de salida que permite al bot entregar la información en str (texto)