Módulo 4. Asistentes de OpenAI, Custom Actions y manejo de errores en Make

CÓRDOBA CREA FUTURO

1. Asistentes de OpenAI

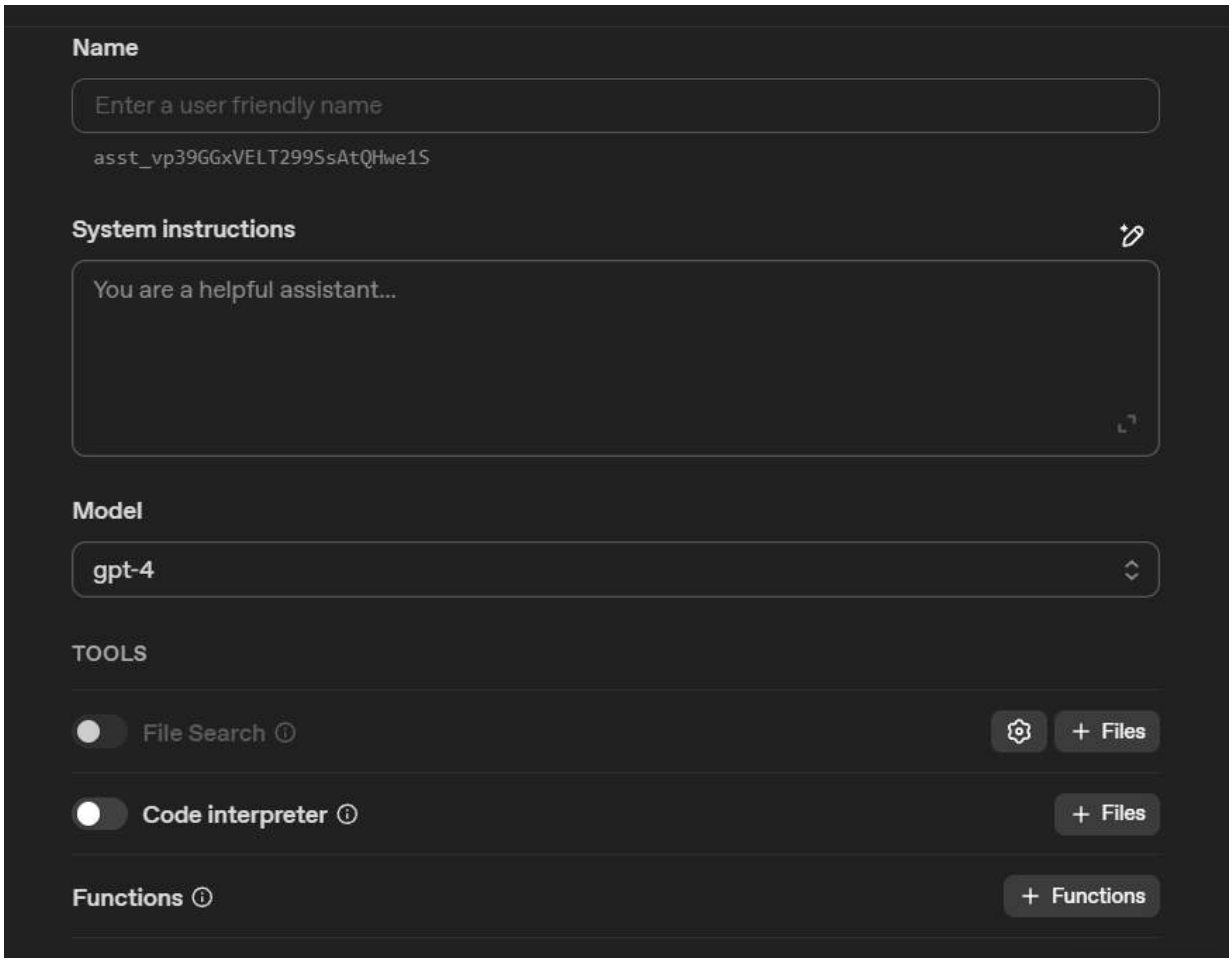
Los asistentes de OpenAI (OpenAI *assistant*) son asistentes de IA personalizados que se crean en la plataforma de OpenAI con configuraciones específicas de comportamiento, conocimientos adicionales y *skills* o herramientas integradas (OpenAI, 2023; OpenAI Platform, s. f.a). En esencia, tiene instrucciones de sistema (un *prompt* inicial fijo que define su rol y personalidad), puede incorporar herramientas avanzadas provistas por OpenAI (ejecución de código o búsqueda de información) y también puede poseer una base de conocimiento privada mediante archivos adjuntos. Esto significa que, mientras que al utilizar la API de GPT tradicional, este responde basándose únicamente en su entrenamiento general (y en cualquier *prompt* que le proporcione el usuario en cada consulta), un OpenAI *assistant* actúa conforme a directrices predeterminadas y con capacidades que le configuremos.

Para crear un asistente personalizado, debemos dirigirnos a la sección de “Asistentes del *Dashboard*” en la plataforma de OpenAI (platform.openai.com). Allí podremos definir varios componentes clave que mencionamos anteriormente:

- **Definir el modelo y nombre:** al iniciar la creación, se elige un modelo subyacente (por ejemplo, GPT-4o mini, GPT-5), que será la base de las respuestas del asistente. Luego se le asigna un nombre identificatorio al asistente, que sirve para reconocerlo en la lista de recursos de nuestra cuenta (n8n, s. f.). Por ejemplo, podríamos llamarlo “Asistente VentasAI” si estuviera orientado a informes de ventas.

- **Instrucciones del sistema:** se provee un conjunto de **instrucciones** que actuará como el *prompt* permanente del asistente (n8n, s. f.), de la misma forma que cuando creamos un GPT personalizado en ChatGPT. Aquí se establece el rol, el objetivo, las especificaciones y demás indicaciones que seguirá la IA. Estas instrucciones pueden ser extensas y detalladas para guiar el comportamiento del modelo (n8n, s. f.).

Figura 1: Creación de asistente de OpenAI



Fuente: captura de pantalla de OpenAI.

HERRAMIENTAS | UNA VEZ CREADO EL ASISTENTE

- *Code Interpreter*: se trata de un intérprete de código Python a través del cual el asistente puede ejecutar código para

leer/escribir archivos temporales y devolver resultados (por ejemplo, cálculos, generación de gráficos, análisis de datos). Habilitar esta herramienta le permite al asistente realizar tareas que involucren programación o análisis avanzado de datos durante la conversación (OpenAI Platform, s. f.a).

- *File Search*: es una herramienta de búsqueda en archivos vectorizados que funciona como base de conocimiento adjunta. Esto permite que el asistente consulte documentos proporcionados por el creador (PDF, manuales, bases de datos de texto) para extraer información relevante a las preguntas del usuario (OpenAI Platform, s. f.a; Skroumpelou, 2023). En la práctica, se realiza cargando archivos a la plataforma (propósito "*assistants*") y asociándolos al asistente; el modelo, entonces, puede recuperar fragmentos pertinentes de esos archivos para fundamentar sus respuestas. Por ejemplo, un asistente de soporte técnico podría tener cargado el manual de usuario de un producto y utilizar esta herramienta para encontrar respuestas exactas en ese texto.
- ***Function Calling*:** es una herramienta que da la posibilidad de definir funciones personalizadas que el asistente puede "llamar" cuando necesita realizar acciones externas. Configuramos, a través de código, una función para que el asistente acceda a una API externa cuando se lo pidamos. Como verán, es una herramienta parecida a las *Custom Actions* en los GPT personalizados. Dado que en este caso ya conectamos los asistentes a Make, esta opción de configurar funciones no resulta tan útil.
    - **Archivos y recursos asociados:** si el asistente va a usar la herramienta de búsqueda en archivos o el intérprete de código, es necesario asociar los recursos correspondientes. En el caso de *File Search*, se deben cargar los archivos que constituyen la base de conocimiento del asistente (OpenAI

Platform, s. f.a; Skroumpelou, 2023). En el caso de *Code Interpreter*, se pueden asociar archivos de datos (por ejemplo, CSV, imágenes, etc.), que el asistente podrá leer y procesar en sus sesiones (OpenAI Platform, s. f.a). Los archivos se suben mediante la consola de OpenAI o la API, se especifica el propósito "assistants", y cada archivo recibe un ID único (OpenAI Platform, s. f.a). También pueden cargarse desde la misma interfaz de creación del asistente, en la sección de cada herramienta. Es importante planificar qué documentos o datos necesita el asistente para cumplir su objetivo y cargarlos de antemano, ya que sus respuestas se limitarán a esa información cuando use la herramienta de conocimiento. De hecho, es recomendable indicarlo así en las instrucciones: por ejemplo, "Responde basándote únicamente en la información proporcionada en los archivos adjuntos; si la pregunta se sale de ese alcance, indica cortésmente que no puedes ayudar con eso" (Skroumpelou, 2023). Esto garantiza que el asistente no "invente" respuestas fuera de su conocimiento validado.

- **Configuraciones avanzadas:** OpenAI provee parámetros adicionales al crear asistentes, como el temperature, que define qué tan fiel al *prompt* será el asistente (menor temperatura para mayor obediencia al *prompt*, mayor temperatura para más creatividad y salirse del rol), o el top_p, que controla la variabilidad en las salidas del modelo (n8n, s. f.). También se puede ajustar un parámetro de esfuerzo de razonamiento (reasoning_effort) en algunos modelos razonadores para equilibrar la profundidad de análisis frente a la velocidad (OpenAI Platform, s. f.b). Otra opción interesante es definir formatos de respuesta estructurados (por ejemplo, JSON) usando response_format, lo cual es útil si deseamos que el asistente devuelva datos en un esquema

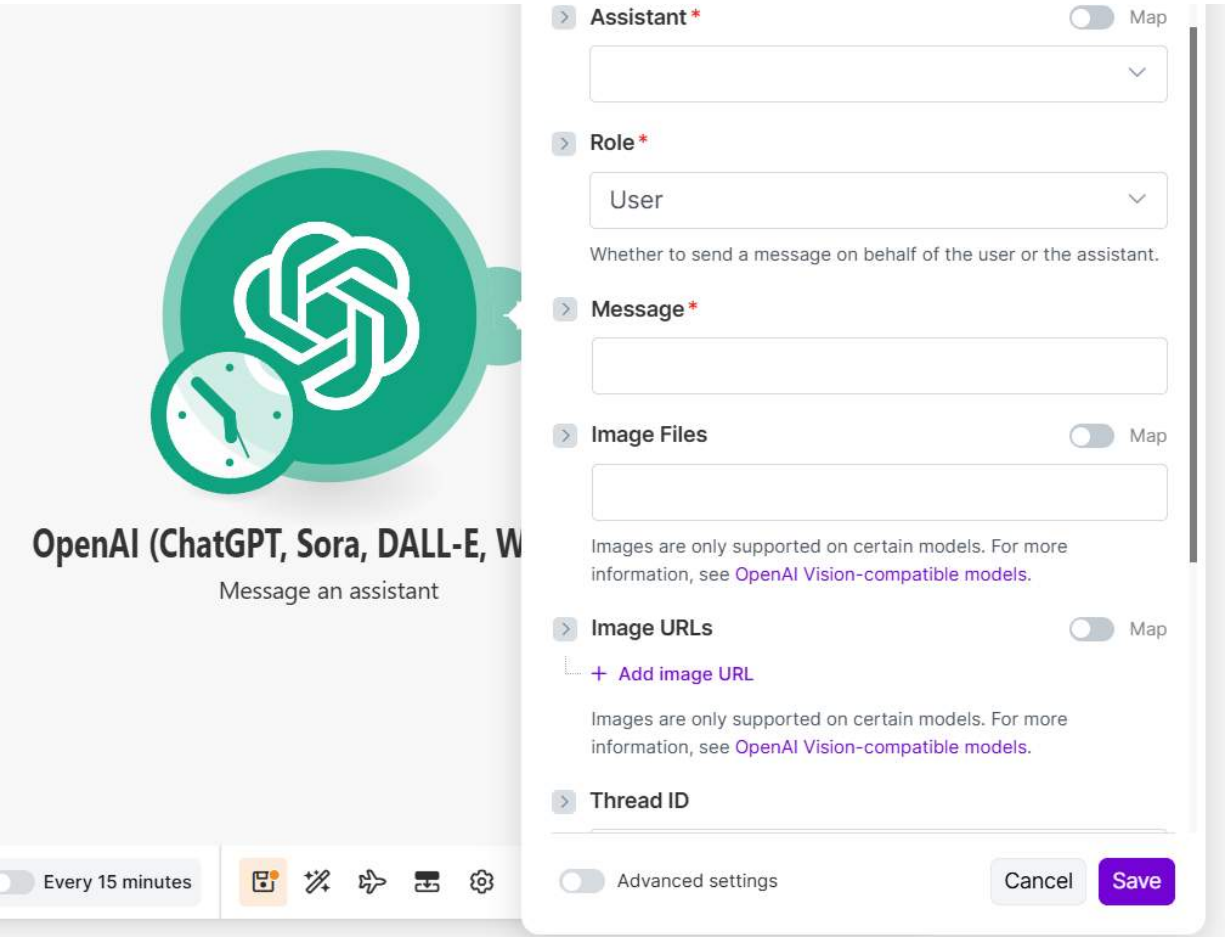
fijo para procesarlos automáticamente (OpenAI Platform, s. f.b), como vimos en módulos anteriores.

en Make podemos utilizarlo en cualquier escenario con el módulo OpenAI: *Message an Assistant,* que permite enviar un mensaje a uno de nuestros asistentes y obtener la respuesta de manera directa dentro de un flujo automatizado. A continuación, describimos cómo se integran estos componentes, incluyendo requisitos y mejores prácticas:

- Selección del asistente*:* una vez conectada la cuenta, el módulo *Message an Assistant* mostrará un campo para elegir cuál asistente utilizar. En el menú desplegable, deberían aparecer todos los asistentes existentes en tu cuenta OpenAI (Justas, 2024).
- Ingreso del mensaje *(prompt):* el módulo solicita el *prompt* o mensaje que deseamos enviar al asistente (n8n, s. f.). Esto corresponde típicamente al mensaje del usuario en la conversación. Como hemos visto, podemos ingresar texto estático o mapear variables de módulos anteriores del escenario. Cabe aclarar que es suficiente con el mensaje de usuario al asistente; no es necesario ningún *prompt* con contexto ni instrucciones, ya que tendrá las instrucciones que le brindamos cuando lo creamos.
- Parámetros y opciones adicionales*:* el módulo *Message an Assistant* en Make permite configuraciones adicionales; sin

embargo, son las mismas que podemos configurar en la plataforma de OpenAI, por lo que conviene dejar de lado esas opciones y realizar la configuración desde la misma plataforma.

Figura 2: Configuración del módulo “Message an Assistant” en Make



Fuente: captura de pantalla de Make.

A continuación, se ilustran tres escenarios prácticos donde se integran asistentes personalizados en flujos de Make, demostrando el valor agregado de estas automatizaciones inteligentes:

- Ejemplo 1: asistente para atención al cliente automatizada.

  Imaginemos una empresa que recibe consultas de clientes vía correo electrónico o formulario web. Usando Make, podríamos automatizar la atención inicial: el escenario se activa con cada nueva consulta recibida (p. ej., módulo Outlook para *email* entrante o *webhook* desde el formulario). A continuación, emplea *OpenAI: Message an Assistant* seleccionando un asistente entrenado con información de productos, políticas de garantía y preguntas frecuentes de la empresa. Este asistente tiene cargados documentos de FAQ y manuales mediante la herramienta de búsqueda en archivos, y está configurado para dar respuestas cordiales y precisas. El *prompt* que se envía podría ser el texto de la consulta del cliente más algún contexto adicional (por ejemplo: “Consulta: {{contenido_del_email}} Información Cliente:

{{nombre}}, {{producto_adquirido}}, etc.”). El asistente genera una respuesta elaborada respondiendo la duda del cliente y citando datos relevantes del manual, si aplica. Make toma la respuesta y, en la siguiente ruta del escenario, la envía de vuelta al cliente (vía *email* de respuesta automatizada). Adicionalmente, se podría registrar la interacción en un sistema CRM mediante otro módulo; también un humano podría revisar la respuesta antes de ser enviada, si se quiere mayor seguridad.

- Ejemplo 2: generación automatizada de reportes de ventas con análisis de IA.

  Supongamos que el Departamento de Ventas necesita un informe semanal de desempeño con gráficas y hallazgos clave. Mediante un escenario de Make programado (por ejemplo, cada lunes a las 8 a. m.), podemos orquestar lo siguiente: primero, un módulo extrae los datos de ventas de la semana desde una fuente (p. ej., un Google Sheets, una base de datos o un CSV exportado del CRM). Luego, se utiliza *OpenAI: Message an Assistant* con un asistente llamado “AnalistaVentasAI”. Este asistente tiene la herramienta de *Code*

Interpreter activada y tal vez archivos de plantilla o históricos cargados. Make le envía un mensaje que incluye las instrucciones para el reporte y los datos correspondientes; por ejemplo: “Aquí están los datos de ventas de la última semana: [adjuntar archivo CSV o JSON con los datos]. Por favor, analiza las tendencias, calcula las métricas principales (ingresos totales, variación vs. semana anterior, etc.), y provee un resumen de 2 párrafos con hallazgos importantes”. Dado que el asistente tiene capacidad de código, tomará el archivo de datos y ejecutará cálculos (OpenAI Platform, s. f.a). El escenario de Make, entonces, puede tomar esa respuesta y proceder a distribuir el reporte: por ejemplo, enviar un *email* a la gerencia con el texto, o bien publicar el resumen en un canal de Slack de ventas y guardar la imagen en una carpeta compartida. Todo este proceso ocurre automáticamente cada semana.

- Ejemplo 3: asistencia educativa personalizada.

  En un contexto educativo o de capacitación interna, podríamos crear un asistente tutor. Supongamos que hay un programa de inducción donde los nuevos empleados

pueden hacer preguntas sobre la empresa, políticas o sistemas internos. Se construye un asistente “TutorAI” con instrucciones para ser pedagógico y paciente, y se le carga como conocimiento el manual del empleado, políticas de la empresa, y documentación de los procesos internos (usando la herramienta de búsqueda en archivos para esos PDF). Integrado en Make, cada vez que un empleado llena un pequeño formulario web con su pregunta (por ejemplo, “¿Cómo puedo solicitar vacaciones?”), el escenario captura esa pregunta, llama al módulo “Message an Assistant con TutorAI”, pasa la consulta y devuelve la respuesta para desplegarla al usuario. El asistente utilizará únicamente la información aprobada de la empresa en sus respuestas y entregará explicaciones detalladas, incluso citando secciones del manual si fuera necesario (porque tiene esas referencias en sus datos). Este sistema funciona como un FAQ interactivo potenciado por IA, siempre disponible.

Estos ejemplos ilustran cómo los asistentes de OpenAI integrados en Make permiten automatizaciones

inteligentes, en las que la IA no solo contesta preguntas, sino que además entiende contexto, accede a datos relevantes, realiza cálculos si es preciso y entrega resultados accionables. Las posibilidades son muy amplias: asistentes para Recursos Humanos (respondiendo preguntas frecuentes de empleados), para TI (guiando en solución de problemas técnicos comunes), para *Marketing* (generando borradores de publicaciones a partir de datos de tendencias), etc.

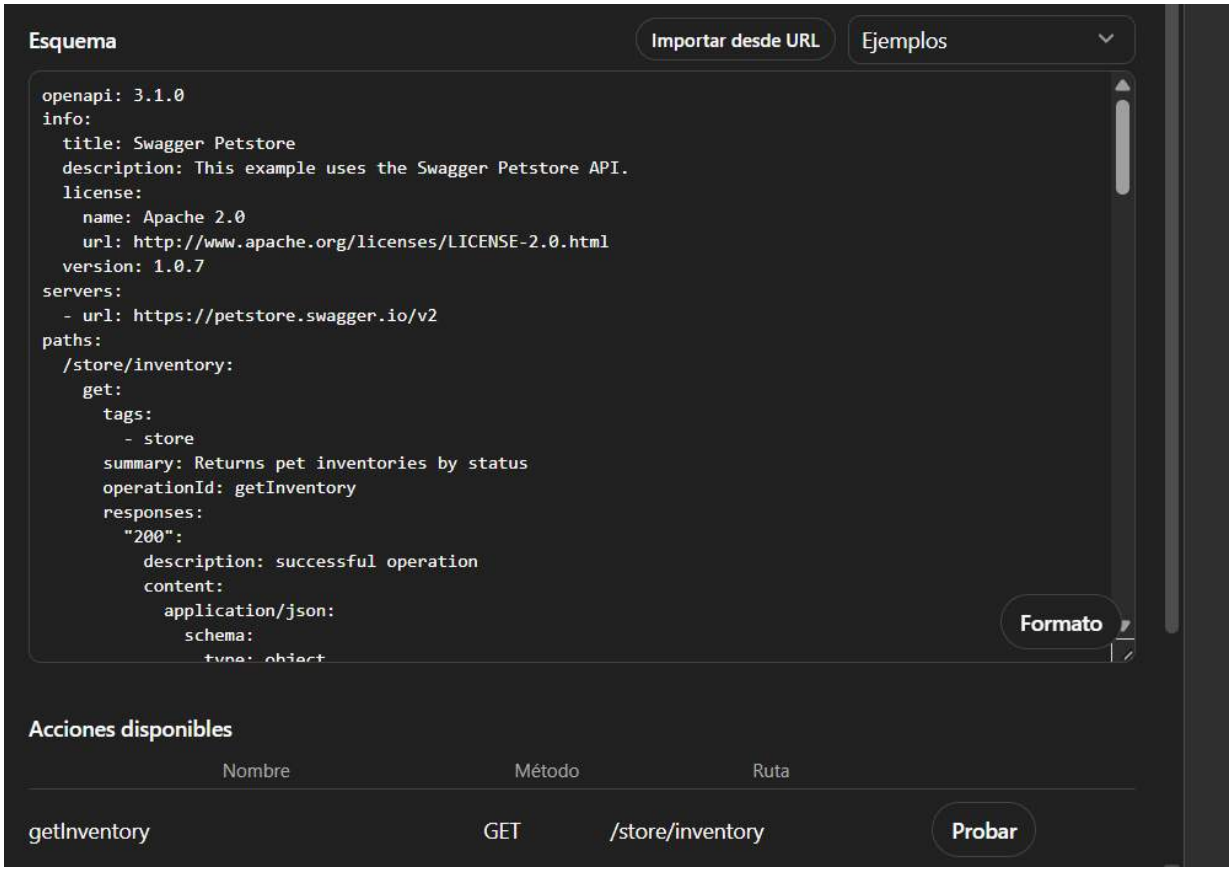
CONTINUAR

2. Custom Actions

La potencia de un asistente de IA aumenta exponencialmente cuando puede no solo conversar, sino también realizar acciones en entornos externos en nombre del usuario. A estas acciones externas desencadenadas por los GPT las llamamos "*Custom Actions*". Las *Custom Actions* se refieren a la capacidad de los asistentes GPT de ejecutar funciones específicas definidas por el creador. Esto permite que, por ejemplo, un asistente no solo responda "Deberías agregar el cliente X a tu CRM", sino que efectivamente lleve a cabo esa operación conectándose al CRM, todo dentro del flujo automatizado. En ChatGPT, esto se logra definiendo acciones con un esquema OpenAPI o JSON que describe, por ejemplo, una operación "EnviarCorreo" con los campos destinatario, asunto, mensaje. Cuando el usuario solicite "Envía un correo a mi equipo avisando que alcanzamos la meta", el GPT puede reconocer que debe usar la acción "EnviarCorreo" en vez de solo responder. Al hacerlo, produce una llamada de función estructurada (p. ej. {"function":

"EnviarCorreo", "arguments": {...}}), la cual debe ser manejada por la aplicación que lo utiliza.

Figura 3: Definición de Custom Actions en un GPT personalizado



Fuente: captura de pantalla de ChatGPT.

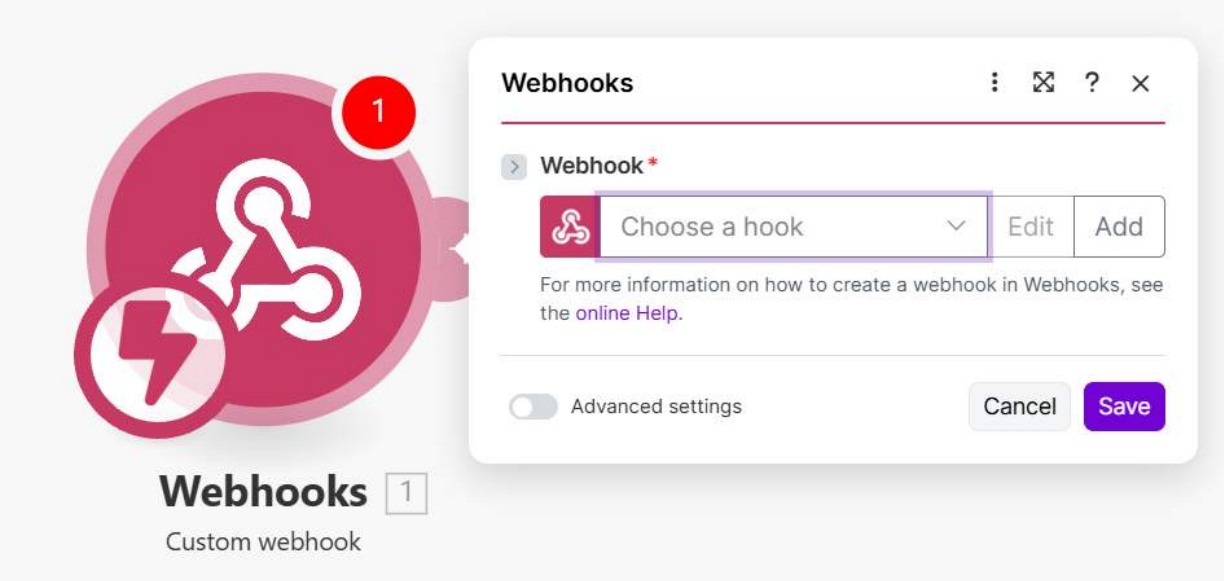
Estas acciones pueden configurarse para llamar a la API de cualquier aplicación; sin embargo, ahora que utilizamos

Make, puede permitirnos sacarle provecho a la función. Dado que Make puede conectarse con multitud de servicios, podemos diseñar nuestras **Custom Actions** de GPT de forma que llamen a *webhooks* de Make. Un *webhook* es una URL donde se pueden enviar datos para desencadenar una automatización; en Make, un módulo de tipo *webhook* (escuchar *webhooks*) puede iniciar un escenario al recibir una petición HTTP. ¿Cómo conectamos esto con GPT? Como vimos, a la hora de crear un GPT personalizado, podemos configurar *Custom Actions* para ese GPT. En ese flujo, se suele proveer un archivo de especificación (YAML/JSON) siguiendo el formato de ChatGPT *plugins* u OpenAPI para describir las acciones. En dicha especificación, se indica la URL del servicio web que realiza la acción. Aquí es donde se aprovecha Make: sin tener que montar un servidor propio, podemos usar un *webhook* de Make como el *backend* de la acción. El procedimiento sería: configurar el *webhook* en Make, obtener su URL única, introducirla en la definición de acción del GPT, y luego, en Make, diseñar la secuencia necesaria.

Para quienes no posean conocimiento sobre cómo programar un esquema para la *Custom Action*, basta con copiar la URL del *webhook* de Make y pasársela al asistente GPT desarrollado por OpenAI para asistirnos en la generación de estos. Podemos acceder a este GPT en el campo donde se

pega el esquema en la *Custom Action*. Al pasarle la URL y especificarle qué información queremos enviar al *webhook*, este nos creará el esquema listo para pegarlo.

Figura 4: *Webhooks*



Fuente: captura de pantalla de Make.

Al poder conectar un GPT con Make, las posibilidades son tantas como aplicaciones tiene disponible la plataforma. Algunos ejemplos de acciones que podríamos implementar mediante Make y *webhooks* incluyen:

- Operaciones en sistemas corporativos: crear un *ticket* en un sistema de *helpdesk*, registrar un cliente en el CRM, actualizar un campo en la base de datos, iniciar un proceso de aprobación en un sistema de *workflow*, etc. Por ejemplo, un asistente de RR. HH. podría tener la acción “aprobar_vacaciones”, que automáticamente marque la solicitud de vacaciones de un empleado en el sistema, si así se lo indica un gerente.

- Comunicación y notificaciones: enviar correos electrónicos y mensajes de chat (Slack, Teams). Enviar *email* es un caso común y muy útil.

Para resumir, las *Custom Actions* amplían el potencial de los GPT desde la generación de texto hacia la automatización integral de tareas. Usando Make como aliado, incluso sin programar desde cero, podemos orquestar que un asistente converse, decida y actúe: una simbiosis poderosa entre IA y automatización.

CONTINUAR

3. Funciones adicionales de Make: manejo de errores

Al desarrollar flujos automatizados robustos, es crucial considerar qué ocurrirá si algo sale mal en alguna parte del proceso. Make proporciona módulos y configuraciones especiales para el manejo de errores, que permiten capturar y gestionar fallos sin interrumpir por completo la ejecución de escenarios. Dado que en escenarios podemos enfrentar errores de API, tiempos de espera o datos incompatibles, el manejo de errores se vuelve fundamental para garantizar la consistencia de una automatización.

¿Qué son los módulos de manejo de errores en Make? Son una característica de Make que actúa como un plan B cuando una operación falla (Fajry, 2025). Por defecto, si cualquier módulo en un escenario lanza un error, el escenario marcará esa ejecución como fallida y se detendrá. Pero al adjuntar un error *handler* a un módulo, le indicamos a Make cómo proceder ante ese fallo en vez de simplemente

abortar. Técnicamente, un error *handler* es una ruta alternativa que se conecta a un módulo (visualmente, aparece como una rama que sale del módulo con un icono de advertencia). Solo se activa si dicho módulo arroja un error; en ese caso, la ruta de error se ejecuta y la ruta principal del escenario puede quedar interrumpida o modificada según el tipo de manejo elegido (Make Help Center, s. f.b). Si no hay error, la rama de error es ignorada. Esto equivale a los bloques *try/catch* en programación, donde se "captura" la excepción y se define un comportamiento compensatorio.

Make ofrece cinco tipos de manejadores de error estándar, cada uno con una semántica particular (Fajry, 2025):

1. *Ignore* (omitir el error): el escenario básicamente ignora el error como si no hubiera sucedido y continúa con los siguientes módulos como de costumbre (Fajry, 2025). La ejecución del módulo que falló se considera resuelta (aunque no produjo resultados), y el escenario no se marca como fallido. Este modo es útil para errores tolerables o datos faltantes que no son críticos. Por ejemplo, si un módulo opcional de búsqueda de datos falla, podemos ignorarlo y

seguir sin esa información. Al usar *Ignore*, asegúrate de que saltarse ese paso no compromete la lógica posterior; solo debe emplearse para errores menores donde continuar es seguro.

2

Resume (reanudar con datos alternativos): en este manejo, si ocurre un error, entramos por la rama de error donde podemos proporcionar una ruta alternativa o datos de respaldo, y luego volver al flujo principal como si el error se hubiese resuelto (Fajry, 2025). En Make, esto se logra poniendo módulos en la ruta de error que, por ejemplo, generan un valor por defecto o toman un camino alternativo, y luego usando un módulo especial “Resume” para reincorporarse al proceso normal. Es como tener un plan de contingencia: *Resume* permite que pese al fallo, la automatización continúe pero con un *fallback*. Por ejemplo, si un asistente de OpenAI no responde (*timeout*), podríamos en la rama de error generar una respuesta estándar (“Lo siento, estoy teniendo dificultades técnicas”) y, con *Resume,* enviar esa en lugar de la respuesta real, y así continuar el flujo de notificación al usuario. Así, la experiencia no se trunca, aunque la acción original no se completó como se esperaba.

3

Break (interrumpir guardando estado): este es un manejo que detiene la ejecución actual en el punto del error, pero a diferencia de una falla normal, *Break* le dice a Make que la ejecute más tarde o la marque para reintento (Fajry, 2025). En la práctica, *Break* crea una "ejecución incompleta" que almacena el estado del escenario en el momento del error (los datos procesados hasta allí, el error ocurrido, etc.) (Make Help Center, s. f.a). Luego, el escenario continúa con el siguiente ítem o transacción (si el *trigger* tenía múltiples ítems, p. ej., procesando varias filas, sigue con la siguiente) (Make Help Center, s. f.a). Las ejecuciones incompletas quedan accesibles en la interfaz de Make para que un usuario las revise y decida resolverlas manualmente o reintentar. Además, *Break* ofrece la opción de reintentos automáticos: podemos configurar en el módulo de error cuántas veces reintentar y con qué intervalo, ideal para errores transitorios (Make Help Center, s. f.a).

Por ejemplo, si OpenAI responde con un error de *rate limit* o un fallo de conexión, se podría usar *Break* con 2 reintentos, de forma que Make espere unos minutos y vuelva a intentar la

llamada a la API sin intervención humana (lo cual suele resolver problemas temporales de red o límites, y es más eficiente que notificar manualmente). En casos en que, tras X reintentos, aún falla, la ejecución queda incompleta para intervención manual.

Break es sumamente útil en procesos por lotes: evita que un error con un solo elemento frene todo el lote. Supongamos que estamos procesando 100 solicitudes de clientes con ayuda de un asistente AI, si una falla por datos inválidos, *Break* aislará esa y permitirá que las demás 99 se procesen normalmente, mientras esa una se guarda aparte para revisión.

4 *Commit* (*commit* y detener): este manejador aplica principalmente en escenarios que involucran transacciones o acciones acumulativas en bases de datos. *Commit* indica que, ante un error, queremos confirmar los cambios realizados hasta antes del error y luego detener la ejecución (Fajry, 2025). Es decir, “salvar lo que sí se hizo bien, y no

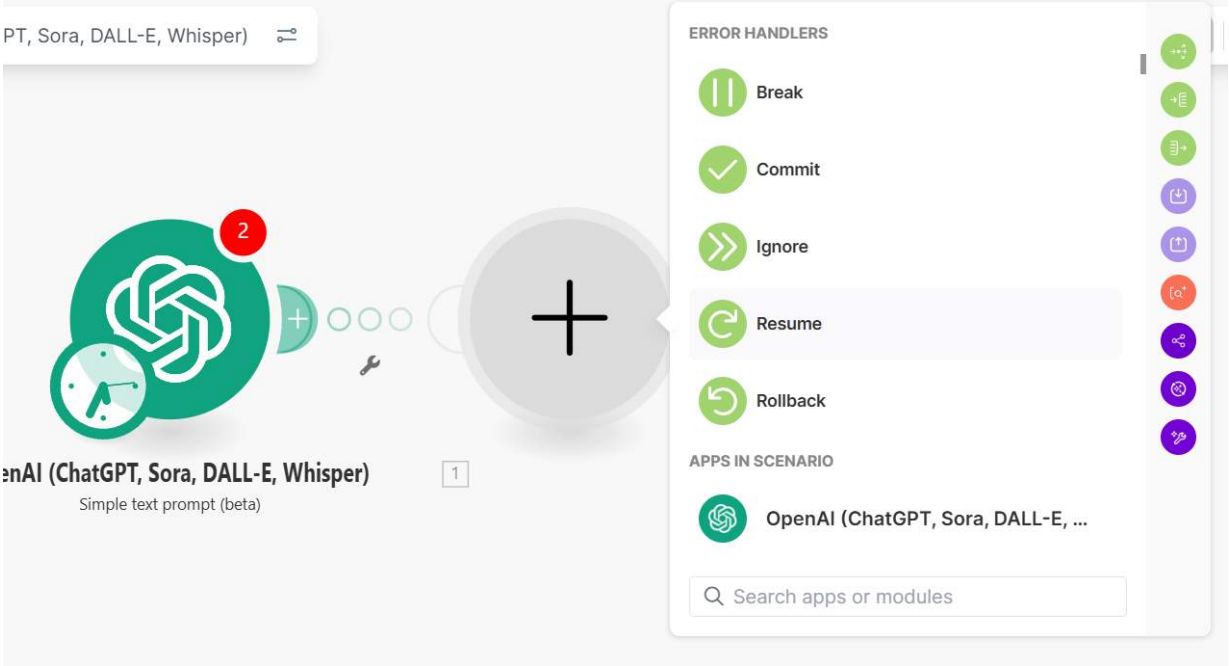
continuar con nada más". En Make, ciertos módulos tienen la capacidad de ejecutar en modo transaccional (poseen una etiqueta que dice "Acid"). Usando *Commit*, nos aseguramos de que esas operaciones se confirmen en la base de datos en vez de revertirse, a pesar de que posteriormente ocurra un error (Fajry, 2025). Un ejemplo: imaginemos un escenario que registra a un cliente en dos sistemas. El primer registro en la Base A fue exitoso, pero el segundo en la Base B falló. Con *Commit*, podríamos hacer que el registro en A quede firme (no se deshaga) y luego parar ahí para solucionar manualmente el de B. Sin *Commit*, por defecto, Make haría *rollback* (revertiría) en A si es una operación reversible.

5

Rollback (revertir cambios y detener): este es el inverso del anterior. *Rollback* indica que, si ocurre un error, se deben deshacer las operaciones previas que sean reversibles, y luego se debe detener el escenario (Fajry, 2025). Solo aplica a módulos que soportan reversión (p. ej., módulos de base de datos que tengan transacciones abiertas); es decir, si anteriormente hay un módulo para enviar un correo, esa operación no podrá deshacerse.

Continuemos con el ejemplo de dos registros A y B: con *Rollback*, si falla el registro en B, se revertiría automáticamente el registro A (y dejaría la base de datos sin el cliente) para mantener consistencia, ya que la secuencia completa no pudo completarse. Después se detendría la ejecución para intervención. Muchos escenarios no utilizan transacciones avanzadas, pero es vital conocerlo en procesos financieros o de inventario donde un error a la mitad requeriría anular lo anterior para no generar inconsistencias. Cabe destacar que, si no hay nada que revertir (porque las acciones anteriores no eran ACID), *Rollback* termina comportándose parecido a un *Break* que detiene la ejecución, pero la distinción es importante cuando hay operaciones críticas encadenadas.

Figura 5: Opciones de manejo de errores en un escenario de Make



Fuente: captura de pantalla de Make.

Además de estos tipos, Make permite combinar manejadores con rutas de notificación o filtrado. Por ejemplo, se puede configurar un *Ignore* condicional que solo ignore ciertos tipos de error y otros no. La interfaz de Make incluso sugiere atajos: cuando ocurre un error en pruebas, ofrece botones para "Añadir *Ignore*" (ignorar siempre ese tipo de error) o "*Ignore* con filtro" (ignorar solo si coincide con ese error específico) (Make Help Center, s. f.b). También es común en la rama de error incluir módulos de notificación:

por ejemplo, si cualquier error pasa por *Break*, añadimos un módulo de “Enviar *email*” al administrador con detalle del error, para garantizar que alguien esté al tanto aunque el sistema lo maneje automáticamente. Recordemos: si un error no es manejado por ninguno de estos métodos, hará que el escenario falle y posiblemente se desactive (Make desactiva escenarios que fallan muchas veces seguidas como medida de seguridad). Por eso es recomendable manejar los errores esperables, sobre todo si se realiza un proceso importante, para asegurar que el escenario no se detenga abruptamente sin nuestro conocimiento (Make Help Center, s. f.b).

Figura 6: Rutas de manejo de errores y notificaciones en Make

Fuente: captura de pantalla de Make.

Nota aclaratoria sobre uso de IA

Este material fue asistido con herramientas de IA generativa para tareas de borrador, síntesis, reescritura y apoyo en la organización de contenidos. Cada sección fue revisada, editada y validada por el equipo humano, que verificó la precisión conceptual, la coherencia pedagógica y las fuentes citadas. Se invita a contrastar con las referencias

bibliográficas incluidas y la documentación oficial. Dado que los modelos de IA evolucionan con rapidez, ciertas especificaciones técnicas podrían actualizarse; este texto refleja el estado del conocimiento al momento de su elaboración.

CONTINUAR

Referencias

Fajry, A. (2025). *How To Handle Errors In Make: A Complete Guide*. Amine Fajry. https://aminefajry.com/handle-errors-in-make

Justas. (17 de septiembre de 2024). *OpenAI module "Message an Assistant" doesn't return any assistants.* [Publicación en foro]. Make Community. https://community.make.com/t/openai-module-message-an-assistant-doesnt-return-any-assistants/55228

Make Help Center. (s. f.a). *Break error handler. Make Help Center*. https://help.make.com/break-error-handler

Make Help Center. (s. f.b). *Overview of error handling. Make Help Center*.

n8n. (s. f.). *OpenAI Assistant operations*. https://docs.n8n.io/integrations/builtin/app-nodes/n8n-

nodes-langchain.openai/assistant-operations/

OpenAI Platform. (s. f.a). *Assistants API deep dive*. https://platform.openai.com/docs/assistants/deep-dive

OpenAI. (2023). *Presentamos GPT*. https://openai.com/index/introducing-gpts/

OpenAI Platform. (s. f.b). *Assistants. OpenAI API Reference*. https://platform.openai.com/docs/api-reference/assistants

Skroumpelou, K. (2023). *OpenAI's Assistants API — A hands-on demo. Medium.* https://pakotinia.medium.com/openais-assistants-api-a-hands-on-demo-110a861cf2d0

CONTINUAR