CENTRO DE E-LEARNING UTN BA
Medrano 951 CABA, Buenos Aires Argentina
(1179) // tel +54 11 7078 – 8073 / fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Documento Clase 9: Automatización de Tareas II

Material de lectura — Diplomatura IA para No Programadores

---

1. Introducción

La automatización de tareas mediante **Inteligencia Artificial (IA)** y herramientas **no-code** está transformando la forma de trabajar de muchos profesionales. En entornos empresariales, integrar IA en flujos automáticos permite delegar labores repetitivas a "asistentes digitales" y enfocarnos en tareas más creativas o estratégicas*[1][2]*. Por ejemplo, en Argentina la empresa NaranjaX logró automatizar la comunicación con sus clientes usando IA, liberando tiempo de sus empleados para otras funciones de mayor valor*[3][4]*. A nivel internacional, compañías líderes también han incorporado IA para optimizar sus procesos, alcanzando mejoras notables en eficiencia y satisfacción de sus usuarios*[5]*.

En esta guía exploraremos **cómo diseñar flujos automáticos** con disparadores (triggers) y acciones inteligentes, cómo integrar servicios de IA (como Google Gemini) para validar, clasificar o analizar contenido, y cómo manejar errores para que nuestros automatismos sean confiables. Todo está orientado a profesionales sin experiencia técnica, por lo que utilizaremos un lenguaje claro y didáctico, acompañando la teoría con ejemplos prácticos, ejercicios guiados paso a paso y casos de uso reales. Al finalizar, tendrás una comprensión profunda de la "Automatización de tareas II", correspondiente a la Clase 9 de la diplomatura, y estarás listo para aplicar estas ideas en tus propios procesos.

> **Caso de uso real (Argentina): Automatización con IA en marketing**
> La fintech NaranjaX implementó **automatización de marketing** para comunicarse con sus usuarios de forma inteligente. Utiliza modelos de analítica avanzada que detectan eventos o intenciones del cliente y activan automáticamente el envío de mensajes personalizados en el momento justo. Por ejemplo, *si un nuevo usuario abandona el registro en la app*, el sistema identifica el paso donde tuvo dificultad (ej. la validación de identidad con foto) y le envía inmediatamente consejos para completar correctamente ese paso.

Gracias a este flujo automático, NaranjaX recuperó al **60% de los usuarios** que no podían avanzar solos en el onboarding[3][4]. Antes, estos envíos se hacían manualmente y con demoras, pero ahora el proceso ocurre en tiempo real y sin intervención humana, mejorando tanto la eficacia (más clientes completan el registro) como la eficiencia del equipo (que ya no debe dedicar medio día a preparar correos cada vez)[6]. Este caso demuestra el poder de la automatización con IA para resolver "puntos de dolor" en la experiencia del cliente de forma práctica y atemporal.

2. Diseño de flujos automáticos con IA

En Make (antes Integromat), un flujo automático se construye como un **escenario** compuesto de módulos interconectados. Podemos imaginarlo como un diagrama de flujo: iniciamos con un **disparador** (trigger) que pone en marcha el proceso cuando ocurre un evento, y luego una serie de **acciones** o pasos que se ejecutan secuencialmente (o en paralelo, si el flujo se bifurca)[7]. Al integrar módulos de **Inteligencia Artificial** dentro de estos flujos, añadimos capacidad de "decisión" o "análisis" avanzado a nuestras automatizaciones, permitiendo que el sistema **valide información**, **clasifique casos** o **genere contenidos** de manera autónoma. A continuación, detallaremos los componentes clave en el diseño de estos flujos inteligentes:

2.1. Disparadores: el inicio del flujo

Un **trigger** o disparador es el evento que inicia automáticamente la ejecución de un escenario en Make. Sin necesidad de intervención manual, el trigger "vigila" cierta condición y, cuando esta se cumple, lanza el flujo de trabajo. Existen diferentes tipos de disparadores:

- **Disparadores temporales o programados:** por ejemplo, ejecutar un flujo cada día a las 9:00, o el último día de cada mes. Útiles para informes periódicos, backups, etc. (Este tipo fue visto en clases previas sobre programación de tareas).
- **Disparadores basados en eventos de aplicaciones:** se activan cuando ocurre algo nuevo en alguna aplicación conectada. Por ejemplo, cuando llega un correo a Gmail, cuando se crea un archivo en Google Drive, cuando alguien envía un formulario, etc.

En Make, los módulos de disparador suelen llamarse "Watch *algo*" ("Vigilar" o "Observar"). **Ejemplos:** el módulo **Gmail – Watch emails** se dispara cuando se recibe

un email nuevo en tu bandeja[8]; el módulo **Google Drive – Watch files** se activa cuando se crea o modifica un archivo en una carpeta específica de Drive[9]. Podemos filtrar los triggers para que reaccionen solo a ciertos eventos: por ejemplo, el trigger de Gmail permite vigilar solo correos no leídos, o de un remitente específico, o con cierta etiqueta[10]. De igual manera, el trigger de Drive puede limitarse a ciertos tipos de archivos (PDF, imágenes, documentos) o a subcarpetas específicas[11][12].

¿Cómo configurar un disparador? Al crear un escenario en Make, primero añadimos un módulo de tipo disparador. La plataforma nos guiará para conectar nuestra cuenta (p.ej., la cuenta de Gmail o Drive) y luego configurar los parámetros del trigger. Por ejemplo, para Gmail "Watch emails" especificamos qué bandeja o etiqueta vigilar, y si queremos marcar los correos como leídos al procesarlos[10]. Una vez activo, Make verificará periódicamente esa condición (o recibirá notificaciones instantáneas, dependiendo del tipo de trigger) y comenzará la ejecución del flujo cada vez que detecte nuevos "bundles" de datos que cumplen el criterio.

Ejemplo sencillo: Podemos tener un escenario cuyo disparador es **"Watch emails" en Gmail** que monitorea la carpeta de "Entrada" en busca de mensajes con asunto "Pedido". Cuando llega un correo nuevo con "Pedido" en el asunto, el flujo se inicia automáticamente. A partir de ahí, podemos encadenar acciones para procesar ese pedido (guardar el adjunto, responder un acuse de recibo, etc.). Si no llegan correos que cumplan la condición, el flujo permanece a la espera sin consumir recursos.

> **Nota:** Es importante definir bien las condiciones del disparador para **evitar ejecuciones innecesarias**. Un criterio demasiado amplio (ej. "vigilar *todos* los correos recibidos") puede disparar el flujo con demasiada frecuencia, incluso para casos que no nos interesan. En cambio, aplicar filtros desde el trigger o justo después con un **filtro** (ver sección 2.3) asegura que el escenario solo procese lo relevante, optimizando las operaciones consumidas y reduciendo complejidad.

2.2. Acciones y módulos en Make

Tras el disparador inicial, el resto de componentes del flujo son **acciones** (también llamadas *módulos de acción*). Cada acción realiza una tarea específica: enviar un correo, leer un archivo, invocar una API de IA, crear un documento, etc. Las acciones toman **datos de entrada** (por lo general, provenientes del paso anterior) y producen **datos de salida** que pueden alimentar al siguiente módulo. Este encadenamiento es posible gracias al mecanismo de **mapeo de datos** en Make: cada módulo expone sus campos (por ejemplo "Asunto del correo", "Cuerpo del correo", "Contenido del

archivo", etc.) para que podamos mapearlos a los campos de módulos posteriores, sin necesidad de código.

En el contexto de **IA integrada**, las acciones más interesantes serán aquellas que envían datos a un servicio de IA y reciben algún resultado de vuelta. Por ejemplo, Make cuenta con módulos específicos para Google Gemini AI que nos permiten aprovechar funciones avanzadas: generar texto o imágenes, analizar contenido, extraer datos estructurados, etc*[13][14]*. Veremos esto a detalle en la siguiente subsección. Además, podemos usar acciones para interactuar con Gmail (enviar un correo, mover un correo a una carpeta, añadir una etiqueta) o con Google Drive (subir un archivo, descargarlo, crear una nueva carpeta, etc.) según la lógica de nuestro flujo.

Al diseñar las acciones debemos pensar en **el orden y la lógica**: ¿qué debe ocurrir primero, qué después? Por ejemplo, si queremos guardar en Drive los adjuntos de un correo, el flujo típico sería: *Trigger: nuevo correo → Acción 1: obtener los adjuntos → Acción 2: subir archivo a Drive*. Si además quisiéramos responder automáticamente al remitente, habría una Acción 3: enviar email de respuesta. Es importante notar que algunas acciones *requieren información de pasos previos*: en el ejemplo, para subir a Drive necesitamos el archivo binario extraído del correo; para responder necesitamos quizás el correo original o ciertos datos de él. Make nos facilita esto permitiendo mapear directamente esas salidas a las entradas necesarias.

Consejo: Antes de implementar, dibuja en un papel el flujo con sus pasos: "Si ocurre X, entonces hacer Y y Z". Identifica qué datos vas a necesitar en cada paso. Esto ayuda a traducir el proceso a módulos Make de forma ordenada. La presentación de Clase 9 sugiere pensar en **"disparadores y acciones con IA"** como piezas de Lego: primero la pieza que detecta el evento, luego las piezas de procesamiento (incluida la IA si hace falta) y finalmente las piezas que actúan (ej. enviar un aviso, guardar un resultado). Mantener esta mentalidad modular te permitirá armar escenarios paso a paso sin perder de vista el panorama general.

2.3. Validación con IA: el semáforo verde/rojo

Una de las aplicaciones potentes de la IA en flujos automáticos es la **validación automática de información**. Por "validación" entendemos comprobar si un contenido cumple ciertos criterios o reglas para decidir automáticamente si damos "luz verde" (procede sin intervención) o "luz roja" (requerir atención humana u otra ruta de procesamiento). En la presentación se aludió a esta idea con la metáfora de un **"semáforo verde/rojo"** controlado por IA.

¿Cómo implementamos una validación automatizada con IA? Básicamente, insertamos un paso en el flujo donde enviamos ciertos datos a un modelo de IA que

actúa como “juez” o “filtro inteligente”, y según su respuesta tomamos acciones diferentes. La IA podría, por ejemplo: evaluar si un texto es apropiado (moderación de contenido), verificar si un formulario está completo, determinar si un documento cumple con ciertos requisitos, etc. El modelo devuelve un resultado que podemos interpretar como *aprobado* (green) o *rechazado* (red).

Ejemplo (moderación de contenido): Imagina un flujo que publica automáticamente comentarios en una web. Podríamos querer evitar publicar contenido inapropiado. Podemos usar un servicio de IA que analice el texto del comentario y devuelva un resultado como “OK” o “Not OK” (o incluso una puntuación de toxicidad). Si el resultado es “OK”, la automatización sigue y publica el comentario (semáforo verde). Si es “Not OK”, entonces se detiene la publicación y quizá envía una alerta al moderador (semáforo rojo). Todo esto sin que un humano tenga que leer el comentario primero.

En Make, concretamente con **Gemini AI**, podríamos implementar lo anterior usando el módulo **“Generate a response”** configurado con un prompt del estilo: *“Lee el siguiente texto y responde solo con la palabra '*GREEN*' si el texto es apropiado, o '*RED*' si viola las normas: <<texto del comentario>>”*. La IA procesará el contenido y nos devolverá “GREEN” o “RED” según el caso. Luego, para ramificar la lógica según ese resultado, podemos utilizar un **Filtro** o un **Router**:

- **Con un Filtro:** colocaríamos un filtro después del módulo de IA que deje pasar solo los bundles donde la respuesta de la IA **existe** y es "*GREEN*" por ejemplo. Dentro de ese filtro irían las acciones de aprobar/publicar. En paralelo, podríamos tener otro camino (otro módulo con su propio filtro) que detecte la condición contraria (respuesta “RED”) y ejecute las acciones de rechazo (por ejemplo, enviar un email al responsable con el comentario bloqueado para revisión).

- **Con un Router:** un Router nos permite dividir el flujo en rutas exclusivas. Podríamos añadir un Router inmediatamente después de la validación con IA, creando dos rutas: Ruta 1 con condición “si respuesta contiene GREEN” y Ruta 2 con condición “si respuesta contiene RED”. En la Ruta 1 conectamos los módulos de acción para caso aprobado; en la Ruta 2 los módulos para caso rechazado. De este modo, cuando corra el escenario, la ejecución tomará **una u otra ruta** dependiendo del resultado de la IA (o ambas rutas si ambas condiciones se cumplen, pero aquí serían mutuamente excluyentes).

¿Filtro o Router? Ambos logran condicionar el flujo. Un **filtro** es más simple cuando solo hay que decidir sí/no continuar con *un* camino. Un **router** es útil para ramificar en *múltiples* caminos desde un punto (por ejemplo: “positivo”, “negativo” o “neutro”; o como veremos luego, clasificar en varias categorías). Internamente, un router en Make agrega múltiples salidas donde podemos poner condiciones a cada una. En la

práctica, usar uno u otro depende de la preferencia y legibilidad: con dos opciones un par de filtros puede bastar, pero con más de dos, un router es más claro.

En todo caso, la IA actúa como nuestro "semáforo inteligente". Un acierto de usar IA aquí es que podemos validar con criterios complejos que serían muy difíciles de programar manualmente. Por ejemplo, evaluar *el tono* de un correo para ver si es legítimo o spam no es trivial con reglas fijas, pero un modelo entrenado (p.ej. Gemini o GPT) puede capturar sutilezas. En la sección 2.5 veremos un ejercicio de validación automática aplicado a documentos.

2.4. Clasificación y ruteo automático de casos

Relacionado a la validación binaria, otro uso típico de IA en automatizaciones es la **clasificación automática** de entradas, para luego ejecutar acciones diferentes según la clase detectada. Podríamos llamarlo un ruteo avanzado: en lugar de solo "verde/rojo", la IA podría asignar una categoría entre varias posibilidades, y el flujo elige qué hacer en cada caso.

Ejemplos de clasificación automática:

- Clasificar correos entrantes en categorías: *consulta de soporte*, *solicitud comercial*, *spam*, *feedback positivo*, etc., y según la categoría tomar distintas acciones (enviar a equipo de soporte, derivar a ventas, responder con información, ignorar…).
- Analizar respuestas de una encuesta de satisfacción (texto libre) y determinar si el sentimiento es *positivo, negativo o neutral*. Luego, quizás, guardar conteos de cada tipo o alertar si hay feedback muy negativo.
- Procesar documentos y decidir su tipo: por ejemplo, un sistema recibe archivos y la IA determina si es un *CV*, una *factura*, un *contrato*, etc., para almacenarlos en diferentes carpetas o iniciar distintos workflows de aprobación.

En Make, la implementación es similar al caso anterior pero con **más de dos resultados** posibles. Podemos lograrlo combinando un módulo de IA y luego un **Router con múltiples ramas**:

1. **Módulo IA de clasificación:** Podríamos usar *Gemini – Extract structured data* o *Gemini – Generate a response*, según nos convenga. Por ejemplo, Gemini tiene una acción para *extraer datos estructurados* que podríamos adaptar para que nos devuelva una etiqueta o un JSON con una clasificación. O más sencillo: usar **Generate a response** con un prompt como *"Clasifica el siguiente correo en 'Soporte', 'Ventas' o 'Otro'. Responde solo con una de esas palabras:"*. La salida sería texto con la categoría.

2. **Router de ruteo por categoría:** Añadimos un Router y creamos, digamos, tres rutas. En cada ruta, definimos una condición que evalúa la salida de la IA:

3. Ruta 1 (Soporte): condición *Category (salida IA)* = "*Soporte*"
4. Ruta 2 (Ventas): condición *Category (salida IA)* = "*Ventas*"
5. Ruta 3 (Otro): podría ser simplemente un *filtro de captura* sin condición específica, de modo que si no cumplió las anteriores, caiga aquí (o poner explícitamente *Category* = "*Otro*").

Make verificará en orden las rutas; las que cumplan la condición ejecutarán sus módulos internos. Es importante asegurarse de marcar en el Router la opción de **"Ejecutar rutas restantes si alguna se cumple"** según sea el caso. Normalmente, el Router intenta todas las rutas que cumplan (no es estrictamente un if-elseif a menos que especifiquemos prioridad y exclusive flow).

6. **Acciones en cada ruta:** Dentro de cada rama del router, colocamos los módulos que correspondan. Ejemplo:
7. En la ruta Soporte, podríamos poner un módulo Gmail "Send an email" que reenvía el correo original al equipo de soporte o crea un ticket.
8. En la ruta Ventas, quizás enviar un correo distinto (o añadir una fila a una hoja de cálculo de leads, etc., aunque en nuestro ejercicio nos limitaremos a Gmail/Drive).
9. En la ruta Otro, tal vez archivar el correo en Drive o aplicarle una etiqueta en Gmail para revisarlo luego.

Este mecanismo nos da un **enrutamiento automático**. La IA hace la parte difícil (entender de qué se trata el contenido) y luego el flujo sigue la "ruta" adecuada sin intervención humana. Un punto importante es **evaluar la confianza** de la IA: podemos, por ejemplo, incorporar en el resultado un nivel de confianza y usarlo para decidir. Si la confianza es baja, podríamos en lugar de actuar directamente, mandar el caso a revisión manual. Esto sería un refinamiento para evitar errores de clasificación graves (un modelo no es 100% infalible). Por simplicidad, nuestros ejemplos asumen que la IA acierta la mayor parte del tiempo en su clasificación.

Nota: En Make también se podría realizar un ruteo sin router usando solo **filtros secuenciales**: p.ej., poner un módulo con filtro "Soporte", luego conectarlo a otro módulo con filtro "Ventas", etc. Sin embargo, esto puede enredar el escenario con cadenas largas. La **herramienta Router** existe para simplificar estos múltiples bifurcaciones. Según la documentación oficial, se agrega un router desde el menú "Flow control" y luego se conectan módulos a cada rama; podemos sumar condiciones a cada ruta como si fueran filtros independientes*[15]*. Una vez configurado, el router enviará los datos entrantes por las ramas cuyos criterios se

cumplan. *Un ejemplo ilustrativo:* la documentación muestra un escenario donde tras procesar datos con OpenAI, se usan routers para enviar los resultados a distintas ramas según el tipo de evento*[16]*. Esto es exactamente el patrón que recomendamos para clasificar con IA.

> **Ejemplo práctico de filtro simple:** La documentación de Make indica cómo usar filtros para restringir el paso de datos. Supongamos un escenario que *al recibir un correo, sube un archivo adjunto a Drive*. Podríamos poner un filtro entre el trigger de Gmail y la acción de Drive para asegurarnos de que solo se procesen correos **que tienen adjunto**. El filtro verificaría que el bundle contenga un archivo; si no, ese correo simplemente no continúa al módulo de subida*[17]*. Este filtro actuaría como una **condición de ruta**: "si hay adjunto, sigue; si no, detiene aquí para ese bundle". En nuestro contexto de clasificación, los filtros (o condiciones en routers) examinarán la **categoría** devuelta por la IA de forma análoga.

2.5. Integrando Gmail con Gemini: análisis automático de contenido

Veamos ahora una integración concreta de lo explicado, combinando las herramientas de Google (Gmail, Drive) con la IA **Gemini** para automatizar un proceso real. Imaginemos la siguiente situación: recibimos muchos correos en nuestra casilla Gmail con cierta información que necesitamos analizar o resumir, y queremos agilizar esa tarea usando IA.

Caso de ejemplo: Una casilla de correo de soporte recibe consultas de clientes. Queremos que automáticamente el sistema lea cada correo, **analice su contenido con IA** (por ejemplo, resuma el problema o detecte la urgencia), y luego tome una acción: quizás guardar un **registro** de ese resumen en Google Drive, y si es urgente, notificar por email a un supervisor. Todo sin que el equipo de soporte tenga que leer y procesar manualmente cada mensaje entrante.

¿Cómo lo construimos en Make? Siguiendo la secuencia lógica:

- **Trigger:** Gmail – *Watch emails*. Configuramos este disparador para vigilar la bandeja de entrada (o una etiqueta específica "Soporte" donde se filtran estos correos). Podemos limitarlo a correos no leídos o marcar como leídos los que ya procese para no duplicar. Cada correo nuevo disparará el escenario con sus datos (remitente, asunto, cuerpo, adjuntos, etc. en un bundle).
- **Acción 1 (IA):** Gemini – *Generate a response* (Generar respuesta) con un prompt diseñado para nuestro fin. Por ejemplo: *"Resumen de soporte: Lee el siguiente correo del cliente y genera un breve resumen (1 o 2 oraciones) indicando el problema principal y su urgencia percibida. Texto: <<Cuerpo del correo>>"*. Aquí aprovechamos la

capacidad de **Gemini** para comprender texto y generar una respuesta. Este módulo enviará el cuerpo del email al modelo de IA y obtendrá como salida un texto generado: el resumen con mención de la urgencia. (Gemini está diseñado para recibir una entrada y devolver una respuesta en lenguaje natural acorde al pedido*[18]*). Otra opción: podríamos usar *Extract structured data* si quisiéramos un resultado JSON (por ejemplo *{ "resumen": "...", "urgencia": "alta" }*), pero para simplificar podemos trabajar con texto libre formateado por prompts.

- **Acción 2 (Ruteo según urgencia):** Vamos a decidir qué hacer según la urgencia que detectó la IA en el resumen. Aquí aplicamos lo de **validación IA (verde/rojo)**: si la IA mencionó que es urgente/alta prioridad, actuaremos de una forma; si no, de otra. Para esto, podemos usar un **Router** con dos rutas:

- Ruta “Urgente”: condición que verifica si el texto de resumen contiene palabras clave como “urgente” o “alta prioridad”.
- Ruta “No urgente”: podría ser la ruta por defecto (sin condición específica, cualquier cosa no capturada por la primera caerá aquí).

- **Acción 3A (para urgentes):** En la rama urgente, añadimos un **Gmail – Send an email** que envía una alerta. Por ejemplo, un correo interno a *supervisor@empresa.com* con asunto “⚠️ Ticket urgente de [Cliente]” y cuerpo con el resumen generado: “Resumen del problema: ... (Urgencia: alta)”. De esta manera, cada vez que llegue un caso crítico, el supervisor lo sabrá de inmediato por correo. (También podríamos aquí marcar el correo original con una etiqueta “Urgente” usando *Gmail – Update email labels*, o moverlo a una carpeta prioritaria, según convenga).

- **Acción 3B (para no urgentes):** En la rama no urgente, podemos automatizar el registro de la consulta sin molestar al supervisor. Por ejemplo, usar **Google Drive – Create a file from text**: creamos un documento de texto o un archivo *.txt* en una carpeta “Resúmenes Soporte”, cuyo contenido sea el resumen generado por Gemini. Podemos nombrar el archivo con fecha y asunto, p. ej. “2025-11-07 - Consulta de [Cliente].txt”. Así, todas las consultas quedan resumidas y archivadas ordenadamente. Alternativamente, podríamos simplemente enviar una respuesta automática al cliente indicando recepción (un “Gracias, estamos en ello”) – esa sería otra acción Gmail –, pero enfoquémonos en Drive para este ejercicio.

¿Qué logramos con este escenario? Para *cada correo* que llega: 1. **IA** lee y comprende el mensaje, produciendo un resumen inteligible. 2. Según la evaluación de la IA, el flujo se bifurca: - Si detectó urgencia, *notifica inmediatamente* al responsable humano. - Si no es urgente, *almacena la info* en un lugar centralizado para seguimiento, sin interrumpir a nadie. 3. Todo esto ocurre en segundos y de forma consistente. El equipo de soporte, al comenzar su día, podría revisar la carpeta de Drive con resúmenes para atender los casos normales, sabiendo que los urgentes ya les fueron avisados por email.

Esta integración **Gmail + Gemini** ejemplifica cómo **"conectar con IA para analizar contenido"** agiliza tareas cognitivas: leer y resumir texto es algo que a un humano le toma tiempo y concentración, pero que una IA bien entrenada puede hacer al instante. Además, al integrarse en Make, el *output* de la IA se convierte en *input* para otras acciones inmediatas (enviar alertas, ordenar documentos, etc.), cerrando el circuito de automatización completa.

Vale aclarar que para usar Gemini en Make primero debemos **conectar** nuestra cuenta o API de Google AI. Make ofrece módulos nativos para Google Gemini AI, lo que simplifica mucho la integración – no hace falta manejar peticiones HTTP manualmente ni autenticación compleja, solo añadimos el módulo y configuramos qué queremos que haga (respuesta, imagen, datos, etc.)*[13][14]*. Como Gemini es una tecnología nueva de IA generativa de Google, su potencia nos permite no solo entender texto sino también generar imágenes o videos a partir de indicaciones*[19][20]*, aunque en automatizaciones típicas de texto (emails, documentos) lo usaremos principalmente para análisis de lenguaje natural. En nuestro flujo ejemplo, usamos **Generate a response** para obtener texto en lenguaje natural, pero ten en cuenta que existe también **Extract structured data**, muy útil si quisieras, digamos, extraer campos específicos de un documento (por ejemplo, leer una factura escaneada y retornar un JSON con importe, fecha, proveedor, etc.). La capacidad de Gemini de *extraer datos estructurados de texto o archivos* puede ahorrarnos tener que programar expresiones regulares o usar OCR por separado*[13]* – la IA lo hace de un tirón.

> **Recomendación:** Cuando diseñes prompts para los módulos de IA, sé **específico y claro** en lo que esperas. En el ejemplo pedimos "responde solo con una palabra (GREEN/RED)" o "genera un breve resumen indicando X". Esto reduce la ambigüedad en la respuesta de la IA, facilitando luego la lógica del flujo (es más fácil buscar la palabra "urgente" que interpretar un párrafo largo y variado). Esta práctica se llama *prompt engineering* y es crucial para obtener resultados útiles de modelos generativos.

Tras haber repasado la concepción de flujos automáticos con IA, pasemos ahora a cómo garantizar que estos flujos corran de manera confiable en el tiempo, abordando el tema de **errores y su manejo**, así como ejercicios prácticos para afianzar lo aprendido.

3. Monitoreo y gestión de errores en flujos automáticos

Cuando automatizamos procesos, es fundamental considerar **qué pasa si algo falla**. Un flujo puede interrumpirse por múltiples motivos: datos inesperados, falta de conexión a un servicio (p. ej., caída de internet o de la API), límites de cuota excedidos,

etc. Sin una estrategia de manejo de errores, nuestra automatización podría detenerse y quedar inactiva sin que nos demos cuenta. De hecho, **más del 60% de las fallas en automatizaciones se deben a un manejo inadecuado de errores***[21]*, lo cual subraya la importancia de este tema.

"Monitoreo y gestión de errores" implica dos aspectos: 1. **Monitoreo proactivo**: estar al tanto del estado de nuestros escenarios (Make ofrece un registro de ejecuciones donde podemos ver si hubo errores, cuántos y en qué módulo). Podemos configurar notificaciones o revisar periódicamente el panel de Make para detectar escenarios detenidos o ejecuciones fallidas. Un buen hábito es, tras crear un escenario, probar casos extremos (datos inválidos, ausencia de datos) para ver cómo se comporta y luego verificar en el *log* de Make esos intentos. 2. **Gestión reactiva (y preventiva) de errores**: configurar nuestros escenarios para que, en caso de error, sigan un plan predefinido en lugar de simplemente "crashear". Make proporciona una funcionalidad llamada **Error Handlers** (manejadores de error) que podemos agregar a cada módulo o escenario para controlar el flujo en situaciones de error.

3.1. Manejadores de error en Make

Los **Error Handlers** son rutas alternativas que se ejecutan cuando un módulo falla. En Make existen cinco tipos principales de manejador de error, cada uno con un comportamiento distinto*[22]*:

- **Rollback (Revertir)**: Deshace ciertos cambios y detiene la ejecución. Solo aplica a acciones de tipo base de datos (transacciones ACID). Es como presionar *Ctrl+Z*: revierte lo que se hizo antes del error, en lo posible*[23][24]*. Por ejemplo, si el escenario añadió registros a una base de datos antes de fallar en un paso siguiente, **Rollback** intentará borrar esos registros añadidos, para no dejar datos a medias. Útil cuando queremos mantener consistencia y preferimos abortar todo si algo sale mal. *(No puede, claro, "des-enviar" un email ya enviado o "des-borrar" un archivo eliminado – solo revierte transacciones marcadas como reversibles.)*

- **Ignore (Ignorar)**: Omite el error y continúa con el siguiente paso como si nada hubiera pasado. Es como decir "si falla, *sigue adelante*". El escenario no se detiene ni se apaga; de hecho, la ejecución se marcará como exitosa a pesar del fallo en ese módulo*[25]*. Esto tiene sentido cuando un error no es crítico. Ejemplo: Si un módulo intenta agregar una fila a una hoja pero esa hoja no existe, podríamos ignorar el error y seguir, quizás registrándolo en otro lado. **Ojo**: ignorar indiscriminadamente puede ocultar problemas, úsese solo cuando el fallo potencial no compromete el objetivo final.

- **Commit (Guardar y detener)**: Salva permanentemente los cambios hechos **antes** del error y luego detiene la ejecución en ese punto*[26]*. Básicamente dice "mantén lo logrado hasta aquí, pero no sigas". De nuevo, aplicable a operaciones de base de datos

principalmente: por ejemplo, en un escenario que realiza varias operaciones ACID, un Commit se asegura de consolidar lo que se hizo antes del fallo (que no haya rollback automático de la transacción) y luego para el flujo. Para acciones no ACID (ej. envío de email, que no tiene "deshacer"), Commit actúa similar a una detención normal pero sin intentar revertir nada[27]. Se usa cuando *queremos conservar* los resultados parciales aunque algo falló después.

- **Break (Pausar/Reintentar)**: Este manejador es muy útil para **reintentos automáticos**. *Break* detiene la ejecución **en el módulo que falló**, marca la ejecución como **incompleta** y **permite reanudarla después**[28][29]. Es como un "pausa y guarda para luego". Lo valioso es que Make puede luego **reintentar esa ejecución incompleta automáticamente** (o puedes tú revisarla y reejecutarla manualmente). Mientras tanto, el escenario puede seguir procesando otros bundles en paralelo (no detiene toda la operación, solo esa rama)[30]. Para que Break funcione, debemos habilitar la opción *"Allow storing of incomplete executions"* en la configuración del escenario[31].
  ¿Cuándo usar Break? Cuando los errores podrían ser temporales o intermitentes, por ejemplo: fallos de conexión, timeouts de API, errores de tasa de límite (rate limit). En vez de abortar por completo, pausamos ese caso y dejamos que Make lo intente de nuevo más tarde (automáticamente, suele reintentar poco después hasta cierto número de veces). Esto previene que pequeñas interrupciones externas tumben nuestro flujo.
  Ejemplo: al llamar a un servicio web, a veces puede no responder; con Break podríamos reintentar la llamada 5 minutos después sin intervención manual. Durante el Break, Make guarda toda la información necesaria del intento fallido para reanudarlo tal cual quedó[28]. Si el error persiste tras varios intentos automáticos (lo maneja Make internamente), la ejecución queda como *incompleta* en el log para que un usuario la atienda. Al usar Break, el escenario **no se desactiva** (no se pone *Off*, sigue activo para próximas ejecuciones), lo cual es crucial para flujos permanentes.

- **Resume (Reemplazar datos y continuar)**: Este manejador ofrece un "Plan B" inmediato cuando algo falla[32]. En lugar de pausar o ignorar, *Resume* permite suministrar **datos alternativos de reemplazo** para que el módulo que falló pueda "simular" un éxito y el escenario continúe con esos datos ficticios[33]. Es decir, anticipamos qué haríamos si X falla. Por ejemplo, si un módulo debía devolver un valor numérico pero falla, con Resume podemos dar un valor por defecto. El flujo sigue usando ese valor. Es como un *fallback*. Se usa cuando podemos definir un comportamiento de respaldo. Ejemplo: tratar de obtener el precio de un producto de una API externa; si no responde, usar un precio aproximado de una base local mediante Resume para no interrumpir un proceso de generación de reporte. Resume mantiene el flujo funcionando "no matter what", aunque los datos no sean los reales, evitando un bloqueo[34]. Obviamente hay que usarlo con precaución y sentido común: conviene para escenarios donde es mejor continuar con datos incompletos que detener todo.

En resumen, **Make nos da control sobre qué hacer ante un error en lugar de simplemente fallar**. Podemos decidir, según el caso de uso, aplicar uno u otro. Por ejemplo, en flujos críticos de negocios quizás preferimos Break (reintentar) o Rollback (consistencia) antes que ignorar, para no perder transacciones. En flujos menos críticos, tal vez Ignore es aceptable para que nunca se detengan.

Agregar un manejador en Make es sencillo: en el editor del escenario, haces clic derecho sobre el módulo donde quieres manejar errores y seleccionas "Add error handler". Aparecerá una ruta de error (similar a un router rojo) donde puedes arrastrar los módulos a ejecutar en caso de fallo, y en las opciones escoger si esa ruta será Break, Ignore, etc. (Make te deja elegir el tipo). Una vez configurado, si ese módulo lanza una excepción, el flujo en lugar de abortar tomará la ruta del handler.

Ejemplo aplicado: Supongamos en nuestro flujo Gmail+Gemini+Drive, la acción de *Enviar email de alerta* podría fallar (quizá la cuenta Gmail llegó a un límite de envío). Podemos colocar un **Error Handler tipo Break** en ese módulo Gmail. Así, si no pudo enviar el correo, la ejecución se pausará e intentará reintentarlo después automáticamente*[29]*. Mientras tanto, otras ejecuciones del escenario para otros correos seguirán independientemente. De este modo, no perdemos la alerta, solo la posponemos hasta que tal vez Gmail permita enviar de nuevo (p.ej., un minuto después). Alternativamente, podríamos usar **Resume** en un caso así para notificar por otro canal: por ejemplo, Resume podría alimentar un módulo alternativo que envíe la alerta vía otro medio (Slack, SMS) y seguir el flujo. Eso sería un diseño de alta disponibilidad.

> **Importante:** Si no se configuran handlers, por defecto cuando un módulo falla, la **ejecución entera** se marca como error y el escenario podría detenerse (si acumula varios errores seguidos, Make pone el escenario en *Estado Off* para prevenir loops). Por eso, es buena práctica envolver llamadas a APIs externas o puntos débiles con manejadores, o al menos monitorear. Como dice un especialista: *"El manejo de errores es el seguro de tu flujo de trabajo: en lugar de que todo se caiga ante un problema, tú decides cómo debe responder el sistema"*[35].

3.2. Reintentos automáticos y mejores prácticas

Dado que uno de los pedidos específicos fue tratar **reintentos en flujos automáticos**, profundicemos en ello. **¿Cómo lograr que un flujo reintente una tarea fallida automáticamente?** La respuesta es: usando la combinación de **Break + Incomplete Executions activas**.

Con Break, como vimos, el error genera una "ejecución incompleta" guardada. Make entonces puede (en *Automatic mode*) volver a intentar completar esa ejecución sin intervención*[29]*. En la configuración de escenario se puede activar cuántos reintentos automáticos se hacen o cada cuánto (Make por defecto intenta completar los incompletos en los siguientes ciclos de ejecución, dependiendo del tipo de trigger; si es trigger programado, lo hará en la siguiente ejecución programada; si es webhook, creo que reintentará después de unos minutos). La documentación indica que hay **modo automático y manual**: en automático, Make reintenta solo (ideal para problemas transitorios como timeouts); en manual, las ejecuciones incompletas quedan para que un usuario las revise y manualmente le dé "Run" de nuevo cuando el problema se haya resuelto*[29]*. Podemos optar por automático en la mayoría de casos prácticos.

Mejor práctica 1: Habilitar "Incomplete Executions" – Asegúrate de entrar a las opciones avanzadas del escenario y activar la casilla que permite almacenar ejecuciones incompletas*[31]*. Sin esto, el Break no funcionará como esperamos.

Mejor práctica 2: Limitar reintentos o notificar tras X fallos – Un error puntual puede reintentarse, pero si falla repetidamente quizá requiere atención humana. Podemos configurar un **contador** o lógica adicional: por ejemplo, usar un *aggregator* o data store para contar cuántas veces ese mismo bundle ha fallado, y si supera 3 intentos, entonces enviar un correo de alerta o hacer Commit/Rollback en lugar de seguir con Break. Esto es más avanzado, pero mencionarlo nos recuerda que los reintentos infinitos pueden no ser deseables (podrían generar loops).

Mejor práctica 3: Logs y alertas – Aunque tengamos reintentos automáticos, conviene monitorear el histórico. Make permite ver todas las ejecuciones (Completadas, Incompletas, Fallidas). Revisar ahí ayuda a identificar patrones de error. Se puede crear un *escenario de monitoreo* aparte: por ejemplo, Make puede enviar un email a un administrador si detecta más de cierto número de errores al día, o si un escenario se apaga. Incluso sin llegar a tanto, es bueno tener al menos un vistazo semanal a las estadísticas que proporciona la plataforma.

Mejor práctica 4: Pruebas con datos reales y extremos – Antes de confiar en producción, hay que testear el escenario con diferentes inputs, especialmente los que podrían causar errores (¿Qué pasa si el correo viene sin asunto? ¿Y si el adjunto es muy pesado? ¿Y si la IA devuelve algo inesperado?). Realiza simulaciones y mira cómo actúan los manejadores de error. Ajusta los handlers según la necesidad observada.

Resumiendo, la gestión de errores no es opcional: es parte integral del diseño de flujos robustos. Al anticipar los posibles fallos y definir cómo responder, nuestras

automatizaciones ganan resiliencia. Como resultado, tendremos menos interrupciones y más confianza para delegar tareas importantes al sistema.

Pasemos ahora a la sección final, donde proponemos **ejercicios prácticos** para poner en práctica todo lo aprendido. Estos ejercicios están diseñados para realizarse en **Make** en 20 minutos o menos, usando exclusivamente módulos de **Gmail, Google Drive y Gemini AI**, y se basan en escenarios reales con documentación oficial de referencia. ¡Manos a la obra!

4. Ejercicios prácticos

A continuación se presentan dos ejercicios paso a paso. Cada ejercicio plantea una situación a automatizar y utiliza nodos de Gmail, Google Drive y Gemini, combinándolos de las maneras que discutimos. Están pensados para que los realices en Make.com, probándolos con tus propias cuentas de Google (preferiblemente cuentas de prueba o con datos ficticios para evitar efectos no deseados en información real). Los ejercicios refuerzan los conceptos de **triggers**, **acciones**, **IA integrada**, **ruteo condicional**, y **manejo básico de errores**.

4.1. Ejercicio 1: Resumen automático de correos y archivo en Drive (Gmail + Gemini + Drive)

Situación: Tienes una casilla de Gmail donde recibes regularmente correos largos (por ejemplo, reportes diarios, noticias o comunicaciones internas extensas). Quieres ahorrar tiempo obteniendo un **resumen automático** de cada correo, y almacenar esos resúmenes en archivos de texto dentro de una carpeta de Google Drive para leerlos más tarde o compartirlos con tu equipo.

Objetivo: Configurar un escenario en Make que, *cada vez que llegue un correo nuevo* a Gmail (que cumpla ciertos criterios), genere un resumen con IA del contenido y guarde ese resumen en un archivo de texto dentro de Google Drive. Adicionalmente, marcará el correo original como *procesado* (p. ej., aplicándole una etiqueta) para no duplicar trabajo.

Pasos detallados:

1. **Crear escenario y disparador Gmail:** Inicia sesión en Make y crea un nuevo escenario. Agrega como primer módulo el **disparador Gmail – Watch emails** (Vigilar emails). Al arrastrarlo, Make te pedirá conectar tu cuenta de Gmail. Sigue los pasos para autorizar a Make (deberás darle permisos de lectura de correos; Make utiliza la API oficial de Gmail*[36]*).

2. Configura el disparador: Elige **Inbox** como carpeta a vigilar (o “Primary” si quieres solo principal, evitando Promociones/SPAM).
3. En “Criteria” selecciona *“Unread Emails”* (solo no leídos) para que solo dispare en correos nuevos.
4. Si quieres ser más específico: en *Has the words* podrías poner una palabra clave común a los correos que te interesa resumir (por ejemplo “Reporte:” o algún remitente específico). Si no, déjalo genérico para cualquier correo entrante.
5. Marca la casilla **“Mark email as read when fetched”** para que una vez que Make detecte el correo, Gmail lo pase a leído (así no lo vuelve a procesar)*[37][38]*.
6. Pon un límite (Limit) de, digamos, 1 o 2 para pruebas, para que no intente agarrar muchos de golpe en la primera ejecución.

7. **Añadir módulo Gemini – Generate a response:** Ahora agrega el módulo de **Google Gemini AI** al escenario. Selecciona la opción **“Generate a response”** (Generar una respuesta)*[18]*. Te pedirá conectar tu cuenta de Google AI o API Key; realiza la conexión (si es la primera vez, es posible que necesites habilitar el servicio, pero Make lo guía).

8. En el campo de **input/prompt** del módulo, vas a escribir la instrucción para la IA. Ejemplo de prompt:

   Resumí el siguiente correo en máximo 3 oraciones, destacando los puntos clave:
   ===
   {{imap_body_plain}}

   Aquí estamos asumiendo que **imap_body_plain** es el mapeo del cuerpo del correo obtenido del módulo Gmail (Make normalmente te permite mapear el contenido del correo; búscalo en la lista de campos del trigger Gmail, posiblemente “Body” o “Texto del mensaje”). El separador “===” no es obligatorio pero ayuda a delimitar claramente el texto original.
9. Asegúrate de indicar en el prompt cualquier detalle: si quieres el resumen en puntos, o en texto corrido, idioma (por defecto Gemini responderá en el mismo idioma del texto dado; puedes forzar “El resumen en español…” si los correos estuvieran en otro idioma).
10. No es necesario configurar mucho más en este módulo; la salida será un texto (campo “Choices” o similar contendrá la respuesta generada por la IA). Mapea ese resultado para usarlo luego.

11. **Añadir módulo Google Drive – Create a file from text:** Ahora arrastra un módulo de Google Drive. Usa la acción **"Create a file from text"** (Crear un archivo a partir de texto)*[39]*. Conecta tu cuenta de Google Drive si no lo hiciste ya.
12. En "Choose a Drive" selecciona "My Drive" (o una unidad compartida si quisieras).
13. En "Folder" navega y escoge/crea la carpeta donde guardarás los resúmenes, por ejemplo una carpeta llamada **"Resúmenes Emails"**. Si no existe, puedes crearlo manualmente en Drive de antemano o usar el módulo "Create a folder" antes (como ejercicio adicional, podrías automatizar que si la carpeta no existe, la cree).
14. En "File name", asigna un nombre dinámico. Por ejemplo: *Resumen - {{subject}} - {{date}}.txt*. Puedes mapear el **Asunto (Subject)** del correo desde el trigger Gmail, y quizá la fecha ({{date}} suele estar en metadatos; sino puedes usar la función formatdate actual). Esto generará nombres únicos por correo.
15. En "File content", mapea el **resultado del módulo Gemini**. Probablemente verás algo como *"Choices Message Content"* o simplemente el output textual del paso 2. Selecciónalo para que ese texto sea el contenido del archivo.
16. Opcional: habilita "Convert to Google Docs" si prefieres que en vez de un .txt se guarde como documento Google Docs (formato Google). Si lo dejas como texto plano (.txt), es también válido y quizás más sencillo de descargar luego.
17. Este módulo creará el archivo con el resumen en la carpeta especificada.
18. **(Opcional) Marcar el correo con una etiqueta en Gmail:** Para fines de auditoría, quizás quieras marcar de algún modo el correo original como "resumido" o moverlo del inbox. Puedes añadir otro módulo Gmail, por ejemplo **"Update email labels"***[40]*.
19. Conecta la misma cuenta Gmail. En "Message ID" mapea el ID del correo del trigger (campo *Message ID* o similar del primer módulo).
20. En "Labels to add", escribe el nombre de una etiqueta en tu Gmail, por ejemplo "Resumido". (Debes tener creada esa etiqueta en Gmail; si no, créala antes de ejecutar el escenario).
21. En "Labels to remove", podrías quitar "Inbox" para sacarlo de recibidos si ya lo procesaste, aunque como lo marcamos leído quizás con eso basta.
22. Este paso es opcional pero muestra cómo interactuar de vuelta con Gmail para organización.
23. **Ejecuta el escenario en modo prueba:** En la esquina inferior izquierda, haz clic en "Run once" (Ejecutar una vez) en Make. Ahora envía un correo de prueba a la cuenta de Gmail (desde otra cuenta, o duplica uno existente y márcalo como no leído). Observa en Make cómo el trigger detecta el correo (verás el módulo Gmail volverse verde y mostrar 1 bundle). Luego el módulo Gemini se ejecutará (puede tardar 1-2 segundos

mientras la IA genera el resumen) y debería mostrar su salida. Finalmente, el módulo Drive creará el archivo. Verifica que no haya errores (todo verde).

24. Abre tu Google Drive y navega a la carpeta “Resúmenes Emails”: deberías ver el archivo recién creado. Ábrelo para comprobar que contiene un resumen coherente del correo original.
25. Mira tu Gmail: el correo de prueba debería ahora estar marcado como leído (por el trigger) y con la etiqueta “Resumido” si hiciste el paso opcional. Así sabrás que fue procesado.
26. **Programar el escenario:** Si todo salió bien, regresa a Make y activa el escenario (turn on). Configura la frecuencia del trigger Gmail. Gmail triggers en Make funcionan típicamente en modo pooling (consulta cada X minutos). Puedes ajustar en la configuración del módulo la intervalicidad mínima (algunos planes permiten cada 5 minutos, otros 15 min). Ajusta según necesites; 15 minutos suele estar bien para un resumen que no es urgente. Ahora el escenario quedará ejecutándose automáticamente en segundo plano.
27. Cada vez que llegue un correo nuevo (no leído) al inbox, en el próximo ciclo programado Make lo procesará, generará el archivo de resumen y marcará el correo. Tendrás todos los resúmenes almacenados ordenadamente.

Referencia a documentación: Este ejercicio hace uso de funciones estándar de Gmail (ver documentación de Gmail en Make para filtros de búsqueda y manejo de IDs*[41][42]*) y de Google Drive (crear archivos de texto*[39]*). Además, emplea la integración con Google Gemini AI (acción *Generate a response*[18]). Todos estos componentes están cubiertos en la documentación oficial de Make y Google, pero la ventaja es que no tuvimos que escribir ni una línea de código para conectarlos.

Con este ejercicio, has practicado: **disparadores por evento (email entrante)**, **mapeo de datos entre módulos**, **uso práctico de IA generativa para resumir texto**, **acciones de creación de archivo en Drive**, e incluso un toque de **post-procesamiento en Gmail** (etiquetado). Es un flujo sencillo pero poderoso: piensa cuánto tiempo ahorrarías si en vez de leer 100 correos largos al día, lees 100 resúmenes generados automáticamente.

4.2. Ejercicio 2: Clasificación de documentos en Drive con IA y envío de alertas (Drive + Gemini + Gmail)

Situación: Tienes una carpeta de Google Drive donde distintos usuarios suben documentos de texto (por ejemplo, propuestas o informes). Quieres implementar un sistema que automáticamente **lea cada documento nuevo** y determine si requiere atención inmediata o no, enviando un correo de alerta si es importante. Supongamos

que los documentos son subidos a una carpeta "Por Revisar", y que si la IA detecta cierto criterio (por ejemplo, el documento menciona “URGENTE” o es una propuesta de alto valor), entonces quieres recibir un email notificándote. Si no, simplemente archiva el documento moviéndolo a otra carpeta para revisar en routine.

Objetivo: Configurar un escenario en Make que vigile una carpeta de Drive por archivos nuevos, use Gemini para analizar el contenido del archivo (en texto) y según el análisis, envíe un correo de alerta o realice otra acción organizativa (mover el archivo).

Supuestos simplificados: Para facilitar el ejercicio, asumiremos que los documentos son **Google Docs** de texto plano (o archivos .txt) para poder extraer fácilmente el texto. Así evitamos complejidades de OCR en PDFs o formatos binarios. También definiremos que el criterio de “importante” es que el contenido contenga la palabra “URGENTE” (en mayúsculas), para no depender de conocimiento de dominio complejo. La IA la usaremos para resumir el documento o detectar su tema, y de paso que mencione si vio “urgencia”.

Pasos detallados:

1. **Crear escenario y disparador Drive:** Inicia un nuevo escenario en Make. Arrastra el módulo **Google Drive – Watch files in a folder** (Vigilar archivos en carpeta)*[9]*. Conecta tu cuenta de Drive si no lo hiciste ya.
2. En *Folder*, navega hasta la carpeta de Drive que será monitoreada (por ejemplo, “Por Revisar”). Selecciónala.
3. En *Watch files*, selecciona “New files” (archivos creados; así ignorará solo modificaciones).
4. En *File types to watch*, puedes especificar **Documents** si la carpeta puede tener varios tipos y quieres solo docs de Google, o **Text** si son txt. Si dejas “All”, vigilará cualquier archivo nuevo. Para nuestro ejercicio, supongamos que se suben Google Docs, así que podrías filtrar por ese tipo.
5. Deja Limit en 1 o 2 para pruebas, y guarda configuración. Este trigger cada X intervalos detectará archivos nuevos añadidos a la carpeta.
6. **Acción: Descargar el archivo (obtener texto)**: Añade un módulo **Google Drive – Download a file** (Descargar archivo)*[43]*.
7. En “File ID”, mapea el ID del archivo proveniente del trigger (campo *File ID* del bundle del Watch files).
8. Marca la opción **“Convert to plain text”** si el archivo es Google Doc. Es posible que Make te deje elegir el formato de conversión: selecciona *text/plain* para obtener un TXT. (En algunos casos el módulo Download tiene un campo “target

format" donde pones por ejemplo *Google Docs -> Plain text*[44]). El objetivo es que la salida de este módulo sea el contenido textual del documento.

9. Este módulo producirá un bundle con el archivo (probablemente con campos: Data (que contiene el texto) y nombre, etc.). Asegúrate de identificar el campo donde está el texto puro. Si fuera un archivo binario, vendría codificado; pero con convert a texto, debe darlo en legible.

10. **Acción IA: Analizar contenido con Gemini**. Añade módulo **Gemini – Generate a response**.

11. En el prompt, escribe algo como:

    Eres un asistente que analiza documentos. Lee el contenido abajo y responde con:
    - Un breve resumen (una frase) del documento,
    - La palabra "URGENTE" si el documento requiere atención inmediata, u "OK" si no.
    Contenido: <<<{{texto_del_archivo}}>>>

    Aquí {{texto_del_archivo}} es el mapeo del **contenido** obtenido del paso 2 (Download file). Puede que tengas que mapear *Data* o similar, según cómo exponga el texto. Hemos estructurado el prompt para que la salida de la IA sea algo como: "*Resumen: ...* | URGENTE" o "*Resumen: ...* | OK". Esto facilita luego parsear la palabra URGENTE vs OK.

12. Alternativamente, podríamos usar *Extract structured data* para pedir JSON, pero mantendremos Generate a response.
13. Configura el módulo. La salida esperada es un string.

Tip: Para probar rápidamente, puedes ejecutar hasta este paso con un archivo de ejemplo. Por ahora, supón que la IA devolverá por ejemplo: "Resumen: Informe de ventas del Q3 con recomendaciones. | OK" o "Resumen: Se solicita aprobación de gasto extraordinario. | URGENTE".

1. **Decisión/ruteo según resultado IA:** Vamos a bifurcar según si la IA marcó "URGENTE" o no. Para simplificar, usemos un **Filtro** en este caso (aunque un Router también valdría, usaremos un filtro con dos ramas secuenciales):
2. **Opción A (Urgente)**: Añade un módulo **Gmail – Send an email** inmediatamente después del módulo Gemini, pero vamos a ponerle un **filtro** antes de que se ejecute. Cuando conectas el módulo, haz clic en la línea de conexión y agrega una condición. Configura la condición: que el **texto devuelto por Gemini contiene "URGENTE"**. (En Make, podrías usar operador "contains" o

simplemente “Gemini Output CONTAINS URGENTE”). Nombrar el filtro como “Si Urgente”.

3. En el módulo Gmail Send Email:
    - Conecta tu Gmail.
    - En “To” pon tu dirección (o la de quien deba ser alertado).
    - Asunto: “📂 Documento URGENTE: {{File name}}” (puedes mapear el nombre del archivo de Drive aquí).
    - Cuerpo: Puedes incluir el resumen y quizás un link al archivo. Por ejemplo: “Se detectó un documento urgente en la carpeta. **Resumen:** {{resumen_parte_del_output}}”. Aquí necesitas extraer el resumen del output de IA. Si el formato es “Resumen: X | URGENTE”, podrías usar funciones de texto para recortar antes del “|”. Una forma sencilla: en vez de hacer un solo campo, podrías modificar el prompt para que devuelva en dos campos separados, pero mantengámoslo simple: pon todo el output como cuerpo por ahora, o usa la función split de Make (en el mapeo avanzado, se podría partir por “|”). Para no complicar, dejemos que ponga todo el mensaje de Gemini en el cuerpo, que incluye el resumen y la marca URGENTE.
    - Opcional: en el cuerpo añade un enlace directo al archivo de Drive. Si en el trigger tienes el campo “File URL” o “Link to file”, mapéalo. Así quien reciba el correo puede hacer clic y abrir el doc.
4. **Opción B (No urgente)**: Agrega otro módulo después, puede ser **Google Drive – Move a file** (Mover archivo). Conecta Drive.
    - En “File ID” mapea el mismo ID del archivo original.
    - En “Destination folder” selecciona otra carpeta, por ejemplo “Archivados” o “Revisar luego”. De este modo, si no es urgente, simplemente se quita de “Por Revisar” y se guarda en Archivados.
    - Pon un filtro a este módulo también: condición que **el output de IA *no* contiene "URGENTE"**. (O alternativamente, un filtro *CONTAINS OK*, asumiendo siempre vendrá una u otra palabra). De esa forma, este módulo solo se ejecutará para los casos no urgentes.
5. **Nota:** Estamos usando dos filtros mutuamente excluyentes en secuencia, lo cual en la práctica funciona similar a un router de dos ramas. Primero, Make evaluará el filtro “Si Urgente”; si cumple, envía el email (y seguirá luego al siguiente módulo pero ese siguiente tiene su propio filtro). Si el doc era urgente, la segunda rama “no urgente” fallará su condición y no moverá el archivo (podríamos, si quisiéramos, aún moverlo también aunque sea urgente, pero supongamos que urgentes los dejamos en la carpeta para que un humano los vea ahí). Si el doc no era urgente, el primer filtro no pasa y no envía email, pero el segundo sí pasa y entonces mueve el archivo. Si no ponemos ninguna ruta más, los urgentes se quedarían en "Por Revisar" marcados como urgentes

por la IA (quizá podríamos en el email instructivo decir “ve a carpeta Por Revisar para verlo”). Esta lógica se puede ajustar a la necesidad, pero lo importante es ver cómo filtrar por el contenido analizado por IA.

6. **Manejo básico de errores (opcional):** Podemos añadir un **Error Handler** al módulo Gemini por si falla la API de IA. Por ejemplo, añadir un *Break* handler: clic derecho en el módulo Gemini, “Add error handler”, elige *Break*. Así si no pudo leer el archivo o la IA falló, la ejecución se guarda incompleta para reintentar luego. Mientras tanto, podríamos notificar de otra forma, pero mantengámoslo sencillo: con Break ya no se apagaría el escenario si un día Gemini está inaccesible.

7. Igualmente, podríamos poner un handler en Gmail Send email (por si el email no se envía, reintentar) y en Move file (por si falla el movimiento, quizás usar Resume para intentarlo más tarde). Estas consideraciones de robustez son buenas prácticas, aunque no obligatorias para completar el ejercicio.

8. **Probar el escenario:** Ejecuta el escenario en modo prueba (Run once). Sube un documento de ejemplo a la carpeta “Por Revisar”. Asegúrate que en el texto del doc incluyes la palabra “URGENTE” en mayúsculas si quieres probar la rama urgente, o no la incluyas para probar la rama normal.

9. Observa en Make: el trigger detectará el archivo (bundle con ID y nombre). El módulo Download file obtendrá el contenido (si todo va bien, el campo Data contendrá el texto; puedes inspeccionarlo en la ejecución). Luego Gemini procesará: en la consola de ejecución verás su salida. Luego, según el caso, el Gmail module se ejecutará (si urgencia) o se saltará (verás un icono de filtro que indica condición falsa), y el Move file lo propio.
10. Verifica los resultados: Si se envió correo, revisa tu bandeja de entrada para ver el mensaje de alerta con el resumen. Si se movió el archivo, mira en Drive que el doc ya no está en “Por Revisar” sino en “Archivados” (o la carpeta que elegiste). Si era urgente, debería seguir en “Por Revisar” (no lo movimos en nuestro diseño) y tú habrás recibido aviso para atenderlo.
11. Testea ambos caminos cambiando el contenido del doc y subiendo otro. También prueba con documentos más largos. Observa cómo la IA resume y marca. Si quieres, ajusta el prompt para afinar qué consideraría urgente (ahora solo busca la palabra literal, pero podrías hacer que la IA juzgue urgencia por contexto).

12. **Activar el escenario:** Satisfecho con las pruebas, activa el escenario. Ahora cada vez que alguien suba un documento a “Por Revisar”, en pocos minutos Make lo procesará. Tendrás un ayudante virtual triageando documentos: te enviará correos solo cuando detecte urgencias y ordenará el resto.

Comentarios: Este ejercicio integró los tres servicios de manera interesante: - Usamos Drive como fuente (trigger de archivo nuevo y descarga de contenido). - Usamos la IA

Gemini para *leer y entender* el contenido del archivo. La IA en este caso hizo un breve análisis y tomó una decisión de categorización (aunque apoyada por nosotros en el prompt con la palabra clave). - Usamos Gmail como salida para notificaciones, además de reordenar archivos en Drive.

Podríamos ampliarlo: por ejemplo, almacenar también los resúmenes en un Spreadsheet o en un archivo (similar al ejercicio 1) para llevar registro de todos los documentos entrantes y sus clasificaciones. También podríamos en lugar de mover, simplemente etiquetar el archivo en Drive o renombrarlo agregando "[URGENTE]" al nombre, etc. Las posibilidades de acciones son muchas, pero lo esencial es el flujo de información: **Drive -> IA -> decisión -> Gmail/Drive**.

Este patrón es común en automatizaciones con IA: fuente de datos, análisis inteligente, actuación basada en resultado. Sin la IA, haríamos reglas estáticas (por ej. buscar "URGENTE" con un filter simple, sin entender contexto). Con IA, tenemos flexibilidad de comprensión del lenguaje: podríamos detectar urgencias aunque no esté la palabra exacta (ej. "lo antes posible" podría inferirse como urgente). Siéntete libre de experimentar cambiando el prompt para ver cómo varía el comportamiento.

Referencia a documentación: Nos basamos en documentación oficial de Drive para descargar archivos (convirtiendo Google Docs a texto)*[45]*, en la integración de Gemini (similar a ejercicio 1) y en Gmail para enviar correos*[46]*. Las condiciones de filtro en Make siguen la sintaxis descrita en la documentación de filtering*[47]*, permitiendo usar operadores "contains" para texto, etc. Siempre que tengas dudas de cómo mapear algo (como obtener el link público de un archivo de Drive, o extraer parte de un texto), la comunidad y help de Make tiene ejemplos útiles.

Estos ejercicios prácticos solidifican los conceptos presentados en la Clase 9: vimos cómo usar **triggers** de Gmail y Drive para iniciar flujos automáticos, incorporamos **IA (Gemini)** para validar, resumir y clasificar información (simulando el "semáforo verde/rojo" y ruteos múltiples), y concluimos con diferentes **acciones** en Gmail/Drive según criterios definidos. También aplicamos nociones de **gestión de errores** (aunque mínimas, mencionamos el uso de Break), demostrando cómo construir flujos resilientes y útiles sin escribir código, solo conectando módulos.

5. Conclusiones y próximos pasos

Hemos profundizado en los contenidos de "Automatización de tareas II", mostrando que incluso profesionales sin conocimientos de programación pueden diseñar

soluciones de automatización **potentes** combinando herramientas no-code como Make con servicios de **Inteligencia Artificial**. Siguiendo el orden temático de la clase, abordamos el **diseño de flujos automáticos con disparadores y acciones inteligentes**, aprendiendo a usar triggers de eventos (correo nuevo, archivo nuevo) para iniciar procesos automáticamente. Detallamos cómo insertar pasos de **IA generativa** para realizar validaciones (semáforo verde/rojo) o **clasificaciones** y así tomar rutas distintas en función del contenido analizado – algo antes impensable sin un programador o un equipo de data science. También cubrimos la importancia del **monitoreo y manejo de errores**, pues una automatización robusta no solo hace la tarea feliz en el caso ideal, sino que también sabe recuperarse o al menos alertarnos cuando algo no sale según lo previsto.

Los **casos de uso reales** presentados (tanto de Argentina con NaranjaX, como internacionales con Leroy Merlin) demuestran que estas no son solo ideas de laboratorio, sino prácticas que hoy aportan valor en empresas de diversos sectores. Integrar IA en la automatización permite ahorrar tiempo, reducir errores humanos y obtener insights rápidos. Por ejemplo, Leroy Merlin logró agilizar el **90% de sus procesos manuales** en devoluciones integrando IA generativa en sus flujos de RPA*[5]*, lo que nos inspira a pensar cuántos procesos cotidianos en nuestras organizaciones podrían beneficiarse de una transformación similar.

Para continuar tu aprendizaje, podrías explorar más módulos de Make (hay miles de apps integrables) y otras capacidades de Gemini (como generación de imágenes, que abre posibilidades en marketing automatizado, por ejemplo). Asimismo, practicar el uso de **routers avanzados, iteradores** (para procesar múltiples elementos dinámicamente), y combinarlo con herramientas vistas en clases anteriores (formularios, bases de datos no-code, etc.) te permitirá construir automatizaciones cada vez más completas. No olvides la parte de **ética y seguridad**: al automatizar con IA, maneja datos sensibles con cuidado (anonimiza si es necesario) y supervisa los resultados al principio para asegurarte de que el modelo se comporte como esperas.

En definitiva, estás terminando esta guía con una comprensión sólida de cómo **diseñar, implementar y mantener** flujos automáticos apoyados por IA. Te animamos a aplicar estas técnicas en un pequeño proyecto piloto en tu ámbito profesional: identifica una tarea repetitiva, aplícale un poco de inteligencia con Gemini, automatiza los pasos con Make, y verás cómo en poco tiempo tendrás un “asistente digital” trabajando para ti. Esa es la esencia de la diplomatura: empoderarte para ser creador de soluciones no-code con IA, sin miedo a la tecnología, adoptando una mentalidad de experimentación y mejora continua.

¡Buena suerte con tus automatizaciones, y que todos tus semáforos sean verdes! 🚀

Bibliografía seleccionada:

- Documentación oficial de Make (Make.com Help Center) – Gmail, Google Drive y uso de filtros*[41][39][17]*.
- Documentación de Make – Manejo de errores y buenas prácticas*[30][48]*.
- Google Make + AI Integrations – Detalles de módulos de Google Gemini AI*[13][18]*.
- Slotnisky, D. (2024). *Cómo las empresas argentinas integran la IA en sus tareas cotidianas*. LA NACIÓN*[3][4]*.
- Appian Blog (2024). *5 casos de uso de IA generativa para potenciar la productividad* – Caso Leroy Merlin*[5]*.
- Fajry, A. (2025). *How To Handle Errors In Make: A Complete Guide*[21][23]*.
(Traducción y adaptación didáctica de fragmentos técnicos).

---

[1] [2] [3] [4] [6] Cómo las empresas argentinas integran la IA en sus tareas cotidianas - LA NACION

https://www.lanacion.com.ar/tecnologia/como-las-empresas-argentinas-integran-la-ia-en-sus-tareas-cotidianas-nid17042024/

[5] 5 casos de uso de IA generativa para su empresa

https://appian.com/es/blog/acp/ai/generative-ai-use-cases-enterprise

[7] [13] [14] [18] [19] [20] Google Gemini AI and GPT Chatbot Integration | Workflow Automation | Make

https://www.make.com/en/integrations/gemini-ai/gpt-chatbot

[8] [10] [37] [38] [40] [41] [42] [46] Gmail modules - Apps Documentation

https://apps.make.com/gmail-modules

[9] [11] [12] [39] [43] [44] [45] Google Drive modules - Apps Documentation

https://apps.make.com/google-drive-modules

[15] How to Use Routers in Make or Integromat Scenarios - XRay.Tech

https://www.xray.tech/post/using-routers-in-make-integromat

[16] Need help with -> make.com scenario stopping at router - Skool

https://www.skool.com/brendan/need-help-with-makecom-scenario-stopping-at-router

[17] [47] Filtering - Help Center

https://help.make.com/filtering

[21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [48] How To Handle Errors In Make: A Complete Guide

https://aminefajry.com/handle-errors-in-make

[36] Gmail - Apps Documentation

https://apps.make.com/google-email