GUÍA DE AUTOMATIZACIÓN

App de Facturas y Presupuestos

Backend FastAPI en Python · Hugging Face Spaces · Make · Google Drive

ORQUESTADOR	BACKEND	AUTOMATIZACIONES	RESULTADO FINAL
Antigravity (o cualquier CLI) Modelo Opus 4.6 / Gemini 3.1 en modo Planning	FastAPI en Python Extracción de facturas Generación de presupuestos PDF Webhooks	1) Guardar facturas y datos en Drive/Sheets 2) Guardar presupuestos PDF en Drive	App desplegada, probada, con secretos configurados y frontend hosteado en Netlify listo para usar desde el móvil

Requisitos previos

■ Necesitas cuenta en: **OpenRouter** y **Groq** (para las APIs de IA), **HuggingFace** (para hostear tu app) y **Netlify** (para hostear tu front). ***TODAS SON GRATUITAS***.

■ Aunque si ya tienes otros hosts o proveedores de APIs puedes usar los tuyos.

** Si utilizas Claude Code conectado al MCP de Google Drive el proceso se simplifica ya que el orquestador puede crear todo lo necesario en Google Drive. Incluso puedes sustituir Make por n8n para automatizar, que también tiene MCP, y Claude Code podría hacerlo prácticamente todo de forma autónoma. Pero el objetivo con esta app era utilizar únicamente herramientas gratuitas.*

Prompt Maestro para Antigravity

Modelo Opus 4.6 / Gemini 3.1, en modo Planning

■ Usa este prompt tal cual. Al final del prompt encontrarás la instrucción: *«Si tienes dudas pregúntame antes de empezar la ejecución del plan.»* El modelo te pedirá aclaraciones (datos de empresa, etc.) antes de generar los archivos.

Objetivo general y Stack

```
Quiero que construyas el backend completo de una app en Python lista para desplegar en Hugging Face Spaces.

OBJETIVO GENERAL
La app tendrá 2 funcionalidades:
1) Leer facturas desde imágenes o PDFs
2) Crear presupuestos en PDF para clientes

STACK Y REQUISITOS
- Backend en Python
- Framework: FastAPI
- Código modular, limpio y bien comentado
- Preparado para Hugging Face Spaces
- Uso de variables de entorno para:
  GROQ_API_KEY
  OPENROUTER_API_KEY
  MAKE_FACTURAS_WEBHOOK
  MAKE_PRESUPUESTOS_WEBHOOK
- Añadir endpoint /health
- Manejo robusto de errores
- Respuestas JSON consistentes
- requirements.txt completo
- README.md con instrucciones de despliegue en Hugging Face Spaces
- La app debe ser capaz de hacer fotos desde el móvil o desde la webcam del PC

FUNCIONALIDAD 1: LECTOR DE FACTURAS
La app debe aceptar múltiples archivos a la vez y acceder a la cámara del movil o del PC para hacer fotos y subirlas a la
app.
Tipos de archivo permitidos:
- jpg
- jpeg
- png
- pdf

Si el archivo es una imagen:
- detectar el documento automáticamente
- recortar el documento
- corregir perspectiva
- mejorar contraste y legibilidad
- dejar la imagen lista para enviarla al modelo vision

Si el archivo es PDF:
- no recortar
- procesar el PDF adecuadamente
- renderizar páginas a imagen si hace falta para análisis visual

La extracción de datos debe usar:
- proveedor principal: Groq con meta-llama/llama-4-scout-17b-16e-instruct
- fallback: OpenRouter con meta-llama/llama-4-scout


Diseña un servicio de extracción con esta lógica:
1. intentar Groq
2. si falla, usar OpenRouter
3. si todo falla, devolver error controlado y marcar revisión manual

La extracción debe devolver JSON ESTRICTO con los siguientes campos:
- proveedor
- cif_nif_proveedor
- num_factura
- fecha_factura
- fecha_vencimiento
- base_imponible
- tipo_iva
- cuota_iva
- total_factura
- moneda
- descripcion
- categoria
- metodo_pago
- confianza_modelo
- archivo_original

Requisitos adicionales:
- fechas en formato YYYY-MM-DD
- importes como números decimales
- no inventar datos
- si un campo no aparece, usar null
- marcar needs_review si faltan campos importantes
```

```
Crea estos endpoints:
1. POST /api/invoices/extract
   - recibe múltiples archivos
   - devuelve resultados estructurados
2. POST /api/invoices/confirm
   - recibe datos ya revisados
   - los envía a MAKE_FACTURAS_WEBHOOK

AUTOMATIZACIÓN FUNCIONALIDAD 1:
Una vez enviemos los datos por el webhook el flujo de Make debe coger esos datos e incluirlos en mi Google Sheets de facturación alojado en mi Drive. Dame el blueprint completo para importarlo en Make y dime como configurarlo. Admeás debe guardar una copia original de la factura en mi carpeta FACTURA de Drive con el nombre de archivo AAAA-MM-DD_num_factura_Factura {Cliente}.pdf/jpg/png

FUNCIONALIDAD 2: GENERADOR DE PRESUPUESTOS
Crear un endpoint:
POST /api/quotes/generate

Debe recibir un JSON con:
- cliente
- correo_cliente
- descripcion
- precio_sin_iva
- tipo_iva
- plazo_ejecucion

La app debe:
- cargar los datos de empresa desde un archivo local de configuración (pideme los datos y crea el archivo company_config.json)
- calcular cuota_iva
- calcular precio_con_iva
- generar un PDF bonito y profesional
- incluir logo
- incluir datos de empresa
- incluir datos del cliente
- incluir fecha
- incluir número de presupuesto
- incluir tabla de conceptos
- incluir pie legal
- nombrar el archivo como:
  AAAA-MM-DD_Número de presupuesto_Presupuesto {Cliente}.pdf

Después debe enviar el PDF y los metadatos al webhook MAKE_PRESUPUESTOS_WEBHOOK. El flujo de Make debe coger este pdf y guardarlo en mi carpeta de Drive "PRESUPUESTOS" y guardarlo con el nombre AAAA-MM-DD_Número presupuesto_Presupuesto {Cliente}.pdf

PDF
Quiero que el PDF se genere con una solución robusta y fácil de desplegar en Hugging Face Spaces. Prioriza estabilidad antes que complejidad visual extrema.

ESTRUCTURA DE PROYECTO
Quiero una estructura modular parecida a esta:
- app.py
- requirements.txt
- README.md
- company_config.json
- utils/
  - vision.py
  - image_processing.py
  - pdf_processing.py
  - invoice_schema.py
  - quote_pdf.py
  - webhooks.py
  - validators.py
- assets/
  - logo_placeholder.png

PROMPTS INTERNOS DEL MODELO
Incluye prompts internos para extracción de facturas pidiendo JSON estricto sin texto adicional.

VALIDACIÓN
Incluye validaciones para:
- tamaño máximo de archivo
- extensiones permitidas
- email válido
- importes válidos
- campos obligatorios

SALIDA
Quiero que me entregues:
1. todos los archivos del backend
2. requirements.txt
3. README
4. company_config.json
5. ejemplos de respuesta JSON
6. Blueprints en JSON de los dos flujos de automatización para importarlos en Make


FRONTEND
Una vez tengamos la app creada y funcionando quiero que crees un frontend en html que se conecte al space de HuggingFace por API para poder utilizar la app desde este html que alojaré en Netlify


Si tienes dudas preguntame antes de empezar la ejecución del plan.
```

Funcionalidad 1: Lector de Facturas

La app debe aceptar múltiples archivos a la vez y acceder a la cámara del móvil o del PC para hacer fotos y subirlas a la app.

Tipos de archivo permitidos: jpg · jpeg · png · pdf

Si el archivo es una imagen:

- Detectar el documento automáticamente
- Recortar el documento
- Corregir perspectiva
- Mejorar contraste y legibilidad
- Dejar la imagen lista para enviarla al modelo vision

Si el archivo es PDF:

- No recortar
- Procesar el PDF adecuadamente
- Renderizar páginas a imagen si hace falta para análisis visual

Servicio de extracción — lógica de fallback

Paso	Acción
1	Intentar Groq con meta-llama/llama-4-scout-17b-16e-instruct (proveedor principal)
2	Si falla, usar OpenRouter con meta-llama/llama-4-scout (fallback)
3	Si todo falla, devolver error controlado y marcar revisión manual

La extracción debe devolver JSON ESTRICTO con los siguientes campos:

Campo	Tipo	Notas
proveedor	Texto	
cif_nif_proveedor	Texto	null si no aparece
num_factura	Texto	
fecha_factura	Texto	Formato YYYY-MM-DD
fecha_vencimiento	Texto	Formato YYYY-MM-DD · null si no aparece
base_imponible	Decimal	Como número, no como texto
tipo_iva	Decimal	
cuota_iva	Decimal	
total_factura	Decimal	
moneda	Texto	null si no aparece
descripcion	Texto	
categoria	Texto	
metodo_pago	Texto	null si no aparece
confianza_modelo	Decimal	Valor entre 0 y 1
archivo_original	Texto	Nombre del archivo original

■ Requisitos adicionales: fechas en formato YYYY-MM-DD · importes como números decimales · no inventar datos · si un campo no aparece, usar null · marcar needs_review si faltan campos importantes.

Endpoints de Facturas

Endpoint	Descripción
POST /api/invoices/extract	Recibe múltiples archivos · devuelve resultados estructurados
POST /api/invoices/confirm	Recibe datos ya revisados · los envía a MAKE_FACTURAS_WEBHOOK

Funcionalidad 2: Generador de Presupuestos

Endpoint: POST /api/quotes/generate

Debe recibir un JSON con:

Campo	Tipo
cliente	Texto
correo_cliente	Email
descripcion	Texto
precio_sin_iva	Decimal
tipo_iva	Decimal
plazo_ejecucion	Texto

La app debe:

- Cargar los datos de empresa desde un archivo local de configuración (`company_config.json`) — pedirle los datos al modelo y que cree el archivo.
- Calcular cuota_iva.
- Calcular precio_con_iva.
- Generar un PDF bonito y profesional.
- Incluir logo.
- Incluir datos de empresa.
- Incluir datos del cliente.
- Incluir fecha.
- Incluir número de presupuesto.
- Incluir tabla de conceptos.
- Incluir pie legal.
- Nombrar el archivo como: `AAAA-MM-DD_Número de presupuesto_Presupuesto {Cliente}.pdf`
- Después enviar el PDF y los metadatos al webhook `MAKE_PRESUPUESTOS_WEBHOOK`

El flujo de Make debe coger este PDF y guardarlo en la carpeta **PRESUPUESTOS** de Drive con el nombre:

`AAAA-MM-DD_Número presupuesto_Presupuesto {Cliente}.pdf`

■ PDF: Quiero que el PDF se genere con una solución robusta y fácil de desplegar en Hugging Face Spaces. Prioriza estabilidad antes que complejidad visual extrema.

Estructura de proyecto requerida

```
app.py
requirements.txt
README.md
company_config.json
utils/
vision.py
image_processing.py
pdf_processing.py
invoice_schema.py
quote_pdf.py
webhooks.py
validators.py
assets/
logo_placeholder.png
```

Prompts internos del modelo

■ Incluir prompts internos para extracción de facturas pidiendo JSON estricto sin texto adicional.

Validación

Incluir validaciones para:

- Tamaño máximo de archivo
- Extensiones permitidas
- Email válido
- Importes válidos
- Campos obligatorios

Salida que debe entregar el modelo

Quiero que me entregues:

- 1) Todos los archivos del backend
- 2) requirements.txt
- 3) README
- 4) company_config.json
- 5) Ejemplos de respuesta JSON
- 6) Blueprints en JSON de los dos flujos de automatización para importarlos en Make

Frontend

■ Una vez tengamos la app creada y funcionando, quiero que crees un frontend en HTML que se conecte al Space de HuggingFace por API para poder utilizarla desde este HTML que alojaré en Netlify.

Recuerda decirle en el prompt: «Si tienes dudas pregúntame antes de empezar la ejecución del plan.»

1 Revisión inicial de los archivos generados

■ Antes de desplegar nada, abre el proyecto localmente y confirma que la estructura coincide con lo esperado. No empieces por Hugging Face sin revisar primero que el backend esté completo.

Archivos y carpetas que deben existir

- Comprobar que existan al menos estos archivos y carpetas: `app.py`, `requirements.txt`, `README.md`, `company_config.json`, `utils/` y `assets/`.
- Verificar que en utils/estén los módulos clave: `vision.py`, `image_processing.py`, `pdf_processing.py`, `invoice_schema.py`, `quote_pdf.py`, `webhooks.py` y `validators.py`.
- Asegurarse de que `assets/` contenga un logo placeholder o el logo definitivo.
- Confirmar que el README incluya instrucciones para Hugging Face Spaces y los nombres exactos de variables de entorno.

Checklist mínimo del backend

Elemento	Debe existir	Observación
`Endpoint /health`	**Sí**	Debe devolver estado OK para diagnóstico rápido.
`POST /api/invoices/extract`	**Sí**	Debe aceptar múltiples archivos jpg, jpeg, png y pdf.
`POST /api/invoices/confirm`	**Sí**	Debe enviar datos validados al webhook de Make.
`POST /api/quotes/generate`	**Sí**	Debe crear el PDF y enviarlo al webhook de Make.
Variables de entorno	**Sí**	Nunca dejes claves hardcodeadas.
Validadores	**Sí**	Tamaño, extensión, email, importes y campos obligatorios.

2 Completar los datos que Antigravity no puede adivinar

Aunque el código esté generado, todavía faltan datos de negocio y credenciales. Esta parte es obligatoria antes del despliegue.

- Rellena `company_config.json` con los datos reales de tu empresa: nombre fiscal, CIF/NIF, dirección, teléfono, email, web, pie legal y numeración base de presupuestos.
- Sustituye `assets/logo_placeholder.png` por tu logo final.
- Define el nombre exacto de tus carpetas de Google Drive: **FACTURA** y **PRESUPUESTOS**.
- Decide el formato definitivo del Google Sheets de facturación para que Make pueda escribir siempre en columnas fijas.

Estructura recomendada para el Google Sheets de facturación

Columna	Tipo	Obligatoria	Ejemplo
fecha_factura	Fecha	Sí	2026-04-19
proveedor	Texto	Sí	Papelería X
cif_nif_proveedor	Texto	No	B12345678
num_factura	Texto	Sí	F-2026-018
base_imponible	Número	Sí	100.00
tipo_iva	Número	Sí	21
cuota_iva	Número	Sí	21.00
total_factura	Número	Sí	121.00
moneda	Texto	No	EUR
descripcion	Texto	No	Compra material oficina
categoria	Texto	No	Material
metodo_pago	Texto	No	Transferencia
archivo_original	Texto/URL	Sí	2026-04-19_F-2026-018_Factura Papelería X.pdf
needs_review	Booleano	Sí	false

3 Despliegue en Hugging Face Spaces

Una vez que el proyecto funcione en local, súbelo a un Space. Para FastAPI suele funcionar bien un Space tipo Docker o Gradio que arranque el servidor Python según la estructura generada.

- Crea un nuevo Space en Hugging Face.
- Sube todos los archivos del proyecto tal y como los generó Antigravity, incluyendo utils/, assets/y company_config.json.
- Define el puerto esperado por tu app si el proyecto lo necesita. En muchos despliegues de Space el servicio escucha en el **puerto 7860**.
- Comprueba que `requirements.txt` instala correctamente: OpenCV, FastAPI, Pillow, pydantic, reportlab o la librería de PDF elegida, y cualquier dependencia para PDFs e imágenes.

■ Después del primer deploy, abre la URL pública del Space y prueba /health. Si /health falla, no continúes con Make todavía. Si te da algún error, copia los logs y dáselos a Antigravity para que lo solucione y vuelve a deployear con los archivos corregidos.

Secrets / variables de entorno que debes crear en el Space

Secret	Uso
GROQ_API_KEY	Proveedor principal para extracción con llama-4-scout.
OPENROUTER_API_KEY	Fallback cuando Groq falle.
MAKE_FACTURAS_WEBHOOK	Webhook del escenario de Make para facturas confirmadas.
MAKE_PRESUPUESTOS_WEBHOOK	Webhook del escenario de Make para presupuestos generados.

4 Configuración en Google Drive

Qué debes preparar en Make ANTES de importar los Blueprints

Para que luego no falle nada, deja preparado esto antes:

Google Drive

- Crea la carpeta **FACTURA**
- Crea la carpeta **PRESUPUESTOS**

Google Sheets y conexiones en Make

- Ten creado ya el archivo de facturación con sus columnas.
- Si vas a guardar también presupuestos en Sheets, crea también esa pestaña.

Conexiones en Make:

- Asegúrate de tener conectada tu cuenta de Google Drive en Make.
- Asegúrate de tener conectada tu cuenta de Google Sheets en Make.

5 Escenario de Make para Facturas

Crea un nuevo flujo en Make e importa el Blueprint.json generado por Antigravity. Este flujo debe arrancarse desde el webhook `MAKE_FACTURAS_WEBHOOK` y terminar guardando datos en Sheets y el archivo original en Drive.

Módulo	Tipo	Descripción
1	Custom Webhook	Recibir el JSON confirmado desde /api/invoices/confirm.
2	Router (opcional)	Separar facturas correctas de facturas con needs_review = true.
3	Google Drive > Upload a file	Subir a la carpeta FACTURA.
4	Google Sheets > Add a row	Añadir fila con los campos normalizados.
5	Email o Slack (opcional)	Notificación si needs_review es true.

Nombre recomendado del archivo en Drive: `AAAA-MM-DD_num_factura_Factura {Cliente}.pdf / .jpg / .png`

Ejemplo: `2026-04-19_F-2026-018_Factura Papelería X.pdf`

Configuración paso a paso:

1. Webhooks > Custom webhook

Aquí realmente no hay mucho que tocar, pero sí hay 3 cosas clave:

Qué tendrás que configurar

- **Nombre del webhook:** ponle un nombre claro, por ejemplo: *facturas_confirmadas*
- **Copiar la URL del webhook:** la URL que te genera Make será la que tendrás que pegar en tu app como `MAKE_FACTURAS_WEBHOOK`
- **Redetectar estructura si hace falta:** si el blueprint no trae bien mapeados los campos, entra al módulo y usa **Redetermine data structure** o haz una prueba real desde la app para que Make «vea» el JSON exacto.

Qué debes comprobar

- El webhook debe recibir, como mínimo, esto por cada factura: datos de la factura · nombre del archivo · contenido del archivo o URL temporal del archivo · indicador de revisión si existe.
- Si ves que Make recibe un array tipo `facturas[]`, perfecto. Si recibe una sola factura suelta, el flujo cambiará un poco.

2. Tools > Iterator

■ Este nodo solo se usa si el webhook recibe varias facturas a la vez.

Qué tendrás que configurar

■ En el campo **Array** debes seleccionar la colección que contiene las facturas, normalmente algo como: `facturas[]`

Qué debes comprobar

■ Después del Iterator, Make ya trabajará factura por factura.

■ Haz una prueba y verifica que en la salida del Iterator aparecen campos individuales como: `num_factura` · `fecha_factura` · `proveedor` · `file_name` · `file_content_base64`

■ Si en vez de campos separados sigue saliendo todo anidado, es que el array seleccionado no era el correcto.

3. Tools > Set variable / Set multiple variables

Aquí lo importante es preparar nombres limpios y reutilizables para los módulos siguientes.

Variables que se recomienda crear:

Variable	Descripción
fecha_formateada	Usa la fecha de factura. Si ya llega como YYYY-MM-DD, puedes usarla tal cual. Si no, conviértela.
cliente_o_proveedor_limpio	Nombre del proveedor sin caracteres raros para usarlo en el nombre del archivo.
nombre_archivo_final	Construye aquí el nombre final, p.ej.: 2026-04-19_F-2026-0012_Factura Papeleria XYZ.pdf o .jpg

■ Quita o sustituye estos caracteres problemáticos en el nombre del archivo: `/ \ : * ? comillas` — porque si no, Google Drive puede darte problemas. No hace falta complicarlo demasiado: este nodo es solo para dejar preparado el nombre final y quizá algún campo limpio.

4. Google Drive > Upload a file

Qué tendrás que configurar

- **Conexión:** conecta tu cuenta de Google.
- **Folder:** selecciona la carpeta de Drive donde quieres guardar las facturas: **FACTURA**. Lo ideal es que esa carpeta ya exista antes de importar el escenario.
- **File name:** aquí mapea la variable que preparaste antes: `nombre_archivo_final`
- **Data:** aquí va el contenido real del archivo. Esto dependerá de cómo lo mande tu app: si manda en base64 ☒ mapear ese campo; si el blueprint ya trae la conversión preparada ☒ solo revisar que el campo correcto esté conectado; si manda una URL ☒ necesitarás un nodo intermedio para descargarlo.

Qué debes comprobar

- Haz una prueba y revisa: que el archivo se sube a la carpeta correcta · que conserva la extensión correcta · que el nombre queda bien · que el archivo se abre bien en Drive.
- Si se sube corrupto, casi siempre el problema está en el campo del contenido del archivo.

5. Google Sheets > Add a row

Qué tendrás que configurar

- **Spreadsheet:** selecciona tu archivo de Google Sheets de facturación.
- **Sheet:** selecciona la pestaña concreta, por ejemplo: *Facturas*.
- **Columnas:** aquí tendrás que mapear manualmente cada columna de tu hoja con cada campo recibido.

Mapea las siguientes columnas:

- Fecha factura
- Proveedor
- CIF/NIF
- Número factura
- Fecha vencimiento
- Base imponible
- Tipo IVA
- Cuota IVA
- Total factura
- Moneda
- Descripción
- Categoría
- Método de pago
- Confianza modelo
- Archivo original
- Needs review
- URL archivo Drive

■ Muy importante: los nombres de columnas de Google Sheets deben estar ya creados antes de usar el módulo. Make no organiza bien esto si la hoja está vacía o mal estructurada. Verifica: que cada dato cae en su columna · que los importes entran como número · que la fecha queda correcta · que puedes guardar también el enlace del archivo subido a Drive si el módulo de Drive lo devuelve.

6. Filtros opcionales

Si quieres, puedes poner un filtro antes de añadir fila a Sheets. Por ejemplo:

- Guardar solo si `num_factura` existe.
- Guardar solo si `needs_review` es false.
- O al revés, guardar todas pero marcar aparte las dudosas.

■ Recomendación: para empezar, guarda todas y usa la columna needs_review en la hoja. Es más simple.

6 Escenario de Make para Presupuestos

Crea un nuevo flujo en Make e importa el Blueprint.json generado por Antigravity. Este flujo debe recibir el PDF generado por /api/quotes/generatey guardarlo en Drive, idealmente junto con sus metadatos.

Módulo	Tipo	Descripción
1	Custom Webhook	Recibir el presupuesto generado.
2	Google Drive > Upload a file	Subir a la carpeta PRESUPUESTOS.
3	Google Sheets o BD (opcional)	Registrar número de presupuesto, cliente, importe y fecha.
4	Gmail u Outlook (opcional)	Enviar el PDF al cliente automáticamente.

Nombre recomendado del PDF: `AAAA-MM-DD_NumeroPresupuesto_Presupuesto {Cliente}.pdf`

Ejemplo: `2026-04-19_P-2026-007_Presupuesto Autos Levante.pdf`

Configuración paso a paso:

1. Webhooks > Custom webhook

Qué tendrás que configurar

- Ponle un nombre tipo: **presupuestos_generados**
- Copia la URL y úsala en tu backend como: `MAKE_PRESUPUESTOS_WEBHOOK`

Qué debes comprobar

- Este webhook debería recibir: datos del presupuesto · nombre de archivo · PDF en base64 o archivo binario · metadatos del cliente.
- Haz una prueba real para que Make detecte correctamente todos los campos.

2. Tools > Set variable / Set multiple variables

Aquí solo necesitas dejar listo el nombre final del PDF.

Ejemplo: `2026-04-19_PRES-0001_Presupuesto Cliente XYZ.pdf`

Variables recomendadas:

Variable	Descripción
fecha_presupuesto	Fecha de emisión del presupuesto.
cliente_limpio	Nombre del cliente sin caracteres problemáticos.
numero_presupuesto	Número correlativo del presupuesto.
nombre_pdf_final	Nombre completo del archivo PDF.

■ Qué debes comprobar: que el cliente no meta caracteres raros en el nombre del archivo. Este es el ajuste más importante de este nodo.

3. Google Drive > Upload a file

Qué tendrás que configurar

- **Folder:** selecciona la carpeta **PRESUPUESTOS**.
- **File name:** mapea la variable `nombre_pdf_final`.
- **Data:** mapea el contenido del PDF que envía tu backend.

Qué debes comprobar

- Que el archivo realmente sea PDF.
- Que se abra correctamente.
- Que llegue a la carpeta adecuada.
- Que el nombre quede como quieres.
- Si el archivo pesa mucho, comprueba también que tu backend no lo está enviando truncado.

7 Pruebas funcionales que deberías hacer

Prueba	Qué validar	Resultado esperado	Estado
Factura JPG simple	Recorte, OCR visual, JSON	Campos principales rellenados y needs_review coherente	**Pendiente**
Factura PDF	Procesado PDF	Extracción sin recorte, sin error	**Pendiente**
Lote de 3 archivos	Proceso múltiple	Respuesta por cada archivo	**Pendiente**
Fallo de Groq	Fallback	Usa OpenRouter automáticamente	**Pendiente**
Quote PDF	Diseño y naming	PDF bonito, datos correctos y nombre correcto	**Pendiente**
Webhook facturas	Integración Make	Drive + Sheets actualizados	**Pendiente**
Webhook presupuestos	Integración Make	PDF guardado en Drive	**Pendiente**

8 Siguiente fase: Frontend HTML en Netlify

Cuando el backend ya esté estable y los dos escenarios de Make funcionen, ya puedes construir el frontend HTML que llamará a la API del Space.

- El frontend debe consumir la URL pública de Hugging Face Spaces por fetch.
- Necesitarás formularios separados para: extracción de facturas y generación de presupuestos.
- Conviene mostrar previsualización, estados de carga, errores y confirmación visual antes de enviar a Make.
- Si el Space no permite CORS por defecto como necesitas, tendrás que habilitarlo desde FastAPI.
- Una vez publicado en Netlify ya puedes acceder desde el móvil y guardarla como APP en tu pantalla de inicio del móvil.

9 Errores típicos y cómo detectarlos rápido

✗ La app arranca pero /health falla

Suele ser un problema de importaciones, rutas, dependencias o arranque incorrecto del servidor.

✗ Make no recibe nada

Revisa que las variables MAKE_FACTURAS_WEBHOOK y MAKE_PRESUPUESTOS_WEBHOOK estén definidas en el Space y que el backend realmente haga el POST.

✗ OpenCV da errores en Hugging Face

Revisa requirements y que la build del Space incluya dependencias del sistema si el proyecto las necesita.

✗ El PDF sale feo o roto

Comprueba rutas de logo, fuentes, saltos de línea y que company_config.json tenga todos los campos.

✗ La factura extrae datos inventados

Refuerza el prompt interno del modelo para JSON estricto, null cuando falte información y needs_review cuando haya duda.

10 Orden recomendado de ejecución

1	Revisar estructura de archivos generados por Antigravity.
2	Completar company_config.json y sustituir el logo.
3	Definir estructura del Google Sheets de facturación.
4	Probar la app localmente (si quieres).
5	Crear los dos escenarios en Make.
6	Crear carpetas FACTURA y PRESUPUESTOS en Drive.
7	Subir el proyecto a Hugging Face Spaces.
8	Añadir los cuatro secrets.
9	Probar /health y después los endpoints reales.
10	Validar que Make guarda archivos y filas correctamente.
11	Solo entonces construir el frontend HTML para Netlify.

Checklist final

✓	El Space arranca sin errores.
✓	/health responde OK.
✓	Groq funciona y OpenRouter actúa como fallback.
✓	Las facturas múltiples responden con JSON consistente.
✓	needs_review se marca cuando toca.
✓	Los PDFs de presupuestos salen bien maquetados.
✓	Make guarda facturas en FACTURA.
✓	Make guarda presupuestos en PRESUPUESTOS.
✓	Google Sheets recibe los datos correctos.
✓	Ya puedes pasar a construir el frontend en Netlify.