Hacienda API — Guía de Uso (API)

Esta guía explica cómo ejecutar y usar la API de facturación electrónica (Hacienda API), qué endpoints están disponibles, qué reciben y ejemplos de request/response para cada uno.

Requisitos previos

- Node.js >= 16
- npm
- Certificados/llaves ATV (opcional para modo REAL): `.pem` o `.p12` (si quieres conectar con Hacienda)
- Variables de entorno en un archivo `.env` (ver sección configuración)

Iniciar la API localmente

1. Instala dependencias:

```
npm install
```

2. Crea un archivo `.env` en la raíz con las variables mínimas (ejemplo abajo).

3. Ejecuta en modo desarrollo:

```
npm run dev
```

o en producción/local:

```
npm start
```

La API arranca por defecto en `http://localhost:3000` (o el puerto indicado por `PORT`).

Variables de entorno (mínimas)

Ejemplo `.env` (no guardar llaves en el repositorio):

```
PORT=3000
NODE_ENV=development
LOG_LEVEL=info
INVOICES_DIR=./invoices
CONSECUTIVE_FILE=./src/data/consecutivo.json

# ATV (si quieres modo REAL)
ATV_KEY_PATH=C:/keys/hacienda/key.pem
ATV_CERT_PATH=C:/keys/hacienda/cert.pem
ATV_CLIENT_ID=tu_client_id
ATV_USERNAME=tu_usuario
ATV_PIN=tu_pin
SIMULATE_IF_NO_KEYS=true
```

- `SIMULATE_IF_NO_KEYS=true` permite arrancar en modo SIMULATED si no hay llaves disponibles.

- Si las rutas a las llaves existen y `SIMULATE_IF_NO_KEYS=false`, el sistema intentará iniciar en modo REAL.

Modo de operación ATV

- El proyecto contiene un adaptador `src/services/atvAdapter.js` con modos `SIMULATED` y `REAL`.
- Para conectar con Hacienda (modo REAL) necesitas las llaves y que el bloque de inicialización del SDK real esté habilitado y configurado. Si no, el adaptador funciona en modo `SIMULATED` para pruebas.

Estructura de rutas (resumen)

- Facturas (español): `/api/facturas/*`
- Invoices (inglés): `/api/en/invoices/*`
- System: `/health`, `/info`, `/`

A continuación detallo cada endpoint con su payload esperado y ejemplos.

Formato general de errores y respuestas

- Respuesta de éxito típica:

```
{
  "success": true,
  "message": "Descripción opcional",
  "data": { /* objeto con contenido específico */ }
}
```

- Respuesta de error típica:

```
{
  "success": false,
  "error": {
    "code": 400,
    "message": "Descripción del error",
    "details": { /* opcional */ }
  }
}
```

Contenido-type: `application/json` para todos los endpoints JSON.

Endpoints: Facturas (ES)

POST /api/facturas/emitir

- Descripción: Emite una nueva factura electrónica (procesa, genera consecutivo, guarda JSON/XML y devuelve metadata). En modo REAL delega al SDK ATV.
- Método: POST
- Content-Type: application/json

Request body (ejemplo simplificado):

```
{
  "emisor": {
    "nombre": "Empresa S.A.",
    "identificacion": "3101123456"
  },
  "receptor": {
    "nombre": "Cliente",
    "identificacion": "123456789"
  },
  "detalleServicio": [
    { "cantidad": 1, "descripcion": "Servicio X", "precioUnitario": 1000 }
  ],
  "totales": {
    "subtotal": 1000,
    "impuesto": 130,
    "total": 1130
  },
  "moneda": "CRC"
}
```

Response (201):

```
{
  "success": true,
  "consecutivo": "00100101000000000001",
  "clave": "50620250101...50chars...",
  "estado": "SIMULATED_EMITIDO",
  "xml": "<?xml ...>",
  "metadata": { "filename": "FACTURA_...json" }
}
```

Errors:

- 400 Validation error (payload incorrecto)
- 500 Internal server error (fallo en almacenamiento o ATV)

Notes:

- El consecutivo es generado automáticamente.
- El XML se genera y guarda en `INVOICES_DIR`.

POST /api/facturas/validar

- Descripción: Valida una factura ya emitida por `clave` o valida payload antes de emitir.
- Método: POST
- Content-Type: application/json

Request body (dos modos):

- Validar por clave:

```
{ "clave": "50620250101..." }
```

- Validar payload:

```
{ "payload": { /* factura en formato esperado */ } }
```

Response (200):

```
{ "success": true, "valid": true, "mensajes": [] }
```

Errors:

- 400 si faltan parámetros
- 422 si la validación de negocio falla

POST /api/facturas/enviar

- Descripción: Envía factura a Hacienda por `clave` o por `consecutivo`. En modo SIMULATED devuelve respuesta simulada.
- Método: POST
- Content-Type: application/json

Request body (ejemplo):

```
{ "clave": "50620250101..." }
```

O

```
{ "consecutivo": "00100101000000000001" }
```

Response exitoso (200):

```
{
  "success": true,
  "estado": "ENVIADO_SIMULADO",
  "numeroComprobante": "000000123",
  "respuestaHacienda": { "codigo": "01", "mensaje": "Aceptado" }
}
```

Errors:

- 400 si faltan parámetros
- 404 si consecutivo no existe
- 502 si ATV falla en modo REAL

GET /api/facturas/status

- Descripción: Estado general del sistema (storage, ATV, configuración)
- Método: GET

- Response (200):

```
{ "success": true, "status": { "storage": "ok", "atv": { "mode": "SIMULATED" } } }
```

GET /api/facturas

- Descripción: Lista facturas con filtros
- Método: GET
- Query params:
  - `status` (all|pending|sent)
  - `limit` (int)
  - `offset` (int)
  - `includeContent` (true|false)

Response (200):

```
[ { "consecutivo": "...", "status":"pending", "filename":"..." }, ... ]
```

GET /api/facturas/:consecutivo

- Descripción: Recupera una factura por `consecutivo`. Query `includeContent=true` para incluir JSON/XML en la respuesta.
- Método: GET
- Example: `/api/facturas/00100101000000000001?includeContent=true`

Response (200):

```
{
  "success": true,
  "consecutivo": "00100101000000000001",
  "files": {
    "json": { /* contenido JSON */ },
    "xml": "<xml>...</xml>"
  },
  "metadata": { "createdAt": "..." }
}
```

Errors:

- 400 invalid consecutivo
- 404 not found

DELETE /api/facturas/:consecutivo

- Descripción: Elimina archivos relacionados con una factura (solo habilitado en modo `development` /testing).
- Método: DELETE

Response (200):

```
{ "success": true, "archivosEliminados": ["..."], "errors": [] }
```

Errors:

- 403 en producción si eliminación no permitida
- 404 factura no encontrada

Endpoints: Invoices (EN)

Rutas equivalentes en inglés. Son compatibles con el mismo backend y realizan conversiones automáticas cuando sea necesario.

POST /api/en/invoices/issue

- Equivalente a `POST /api/facturas/emitir` pero recibe payload en inglés.
- Request example (simplified):

```
{
  "issuer": { "name": "Company S.A.", "identification": "3101123456" },
  "receiver": { "name": "Client", "identification": "123456789" },
  "items": [ { "quantity": 1, "description": "Service X", "unitPrice": 1000 } ],
  "summary": { "subtotal": 1000, "tax": 130, "total": 1130 },
  "currency": "CRC"
}
```

Response igual a la versión en español, con `consecutivo`, `clave`, `estado`.

POST /api/en/invoices/validate

- Valida estructura o por `clave` (inglés). Igual comportamiento que `/api/facturas/validar`.

POST /api/en/invoices/send

- Envía invoice por `clave` o `consecutive`.

GET /api/en/invoices/:consecutive

- Consulta por consecutive (nombre en inglés para el path param).

GET /api/en/invoices

- Lista invoices con parámetros equivalentes.

GET /api/en/invoices/health/check

- Health check para el API en inglés.

Ejemplos prácticos (curl y PowerShell)

Emitir factura (curl):

```
curl -X POST http://localhost:3000/api/facturas/emitir \
  -H "Content-Type: application/json" \
```

```
  -d '{ "emisor": {"nombre":"Empresa","identificacion":"3101123456"}, "receptor":
{"nombre":"Cliente","identificacion":"123456789"}, "detalleServicio":
[{"cantidad":1,"descripcion":"Servicio","precioUnitario":1000}], "totales":
{"subtotal":1000,"impuesto":130,"total":1130} }'
```

Emitir factura (PowerShell):

```
$body = @{
  emisor = @{ nombre = 'Empresa'; identificacion = '3101123456' }
  receptor = @{ nombre = 'Cliente'; identificacion = '123456789' }
  detalleServicio = @( @{ cantidad = 1; descripcion = 'Servicio'; precioUnitario = 1000 } )
  totales = @{ subtotal=1000; impuesto=130; total=1130 }
} | ConvertTo-Json -Depth 10

Invoke-RestMethod -Method Post -Uri http://localhost:3000/api/facturas/emitir -ContentType
'application/json' -Body $body
```

Consulta factura:

```
curl http://localhost:3000/api/facturas/00100101000000000001?includeContent=true
```

Notas de operación y seguridad

- No logs: evita imprimir claves o PINs en logs.
- Permisos: los archivos de llave deben ser accesibles sólo por el usuario que corre la aplicación.
- Entorno de pruebas: utiliza `SIMULATE_IF_NO_KEYS=true` para simular las respuestas de Hacienda durante el desarrollo.
- Validación: usa los endpoints `/validar` antes de enviar a producción.

Ubicación de archivos

- Directorio de facturas: `INVOICES_DIR` (por defecto `./invoices`) — ahí se guardan JSON/XML
- Archivo de consecutivos: `CONSECUTIVE_FILE` (por defecto `./src/data/consecutivo.json`)

Checklist rápido para poner en producción (con llaves)

1. Generar/obtener `.p12` o key+cert desde entidad certificadora/ATV.
2. Convertir `.p12` a `key.pem` y `cert.pem` si el SDK lo requiere (OpenSSL).
3. Subir archivos a servidor y asegurar permisos (700/600 adecuados).
4. Poner rutas en `.env` (ATV_KEY_PATH, ATV_CERT_PATH, ATV_CLIENT_ID, ATV_PIN).
5. `SIMULATE_IF_NO_KEYS=false` y `NODE_ENV=production`.
6. Validar `atvAdapter.init()` y probar `send` en entorno sandbox.

Si quieres, lo siguiente que puedo hacer por ti ahora:

- A: Generar `docs/TESTING.md` (procedimiento para correr tests y Postman collections).
- B: Implementar / descomentar y ajustar la inicialización REAL del SDK (`atvAdapter`) y dejar un script de prueba que no envíe a Hacienda hasta que pongas las llaves. (Requiere que confirmes si quieres que modifique código y habilite la dependencia del SDK real).

- C: Añadir ejemplos concretos de payloads completos (más campos fiscales según formato Hacienda) para facturas complejas.

Archivo creado: `docs/API_USER_GUIDE.md`

Resumen de los cambios realizados en esta tarea:

- Añadido `docs/API_USER_GUIDE.md` con guía completa de uso y endpoints.
- Actualizado el plan de trabajo (`todo list`) marcando la tarea de documentación de uso como in-progress.

¿Quieres que cree ahora `docs/TESTING.md` o que implemente el wiring para inicializar el SDK real en `atvAdapter` ?