

MANUAL TÉCNICO HACIENDA API v1.0

Índice

1. [Introducción](#)
 2. [Arquitectura General](#)
 3. [Stack Tecnológico](#)
 4. [Estructura del Proyecto](#)
 5. [Módulos Principales](#)
 6. [Flujos de Datos](#)
 7. [Configuración Detallada](#)
 8. [APIs REST](#)
 9. [Modelos de Datos](#)
 10. [Manejo de Errores](#)
 11. [Testing](#)
 12. [Deploying](#)
-

Introducción

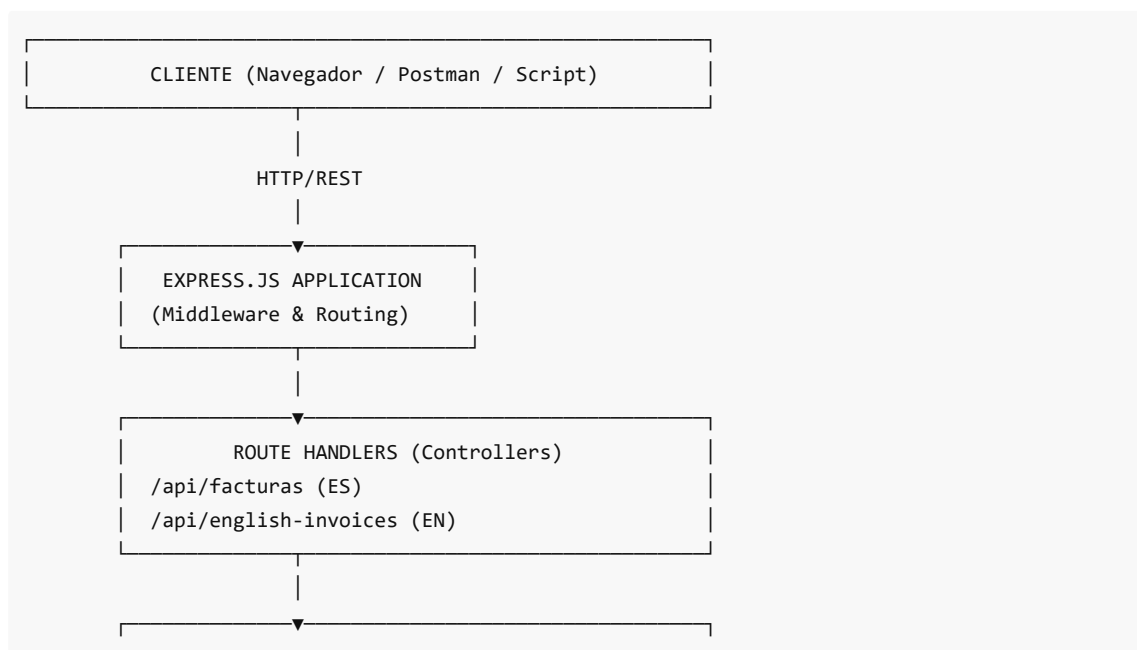
Hacienda API es un sistema de facturación electrónica para Costa Rica que proporciona una capa REST sobre el SDK oficial de ATV (Administración Tributaria Virtual). El sistema funciona en dos modos:

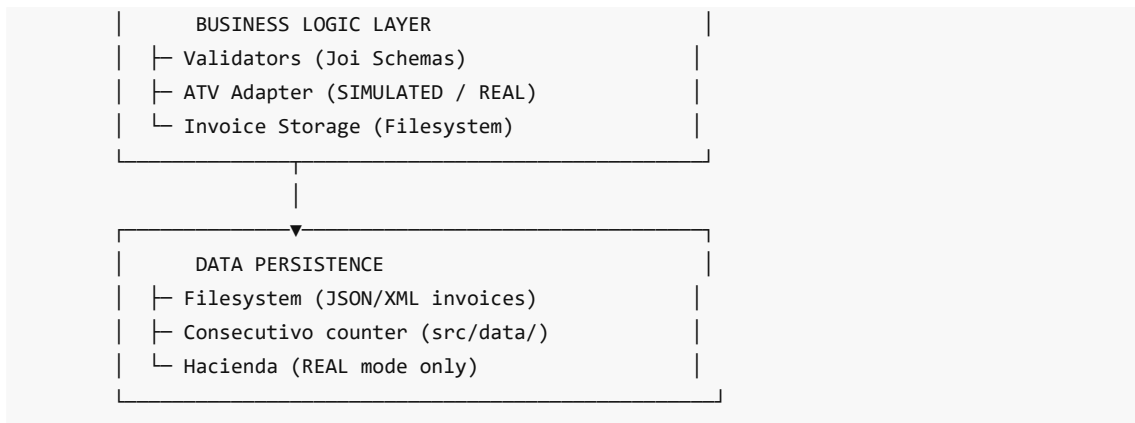
- **Modo SIMULADO:** Simula comportamiento sin necesidad de certificados reales (desarrollo/testing)
- **Modo REAL:** Conecta con servicios reales de Hacienda (producción)

Objetivo

Facilitar la emisión, validación, envío y consulta de comprobantes fiscales electrónicos de manera centralizada y con documentación completa.

Arquitectura General





Patrón de Diseño

- **MVC (Model-View-Controller)**: Controllers manejan requests, Models definen estructura, Views → JSON responses
- **Service Layer**: `atvAdapter` y `invoiceStorage` encapsulan lógica
- **Adapter Pattern**: `atvAdapter` abstrae modo SIMULADO vs REAL
- **Dependency Injection**: Módulos centralizados en `config`

Stack Tecnológico

Core

- **Node.js** (v16+): Runtime JavaScript
- **Express.js** (v4.18): Framework REST
- **CommonJS**: Sistema de módulos

Validación & Mapeo

- **Joi** (v17.11): Validación de esquemas
- **express-validator** (v7): Validación de requests

Almacenamiento & Filesys

- **fs-extra** (v11.1): File I/O mejorado
- **Node.js fs**: Filesystem nativo

Seguridad

- **Helmet** (v7.1): Headers HTTP seguros
- **CORS** (v2.8): Control de acceso cross-origin
- **dotenv** (v16.3): Variables de entorno

Logging

- **Winston** (v3.11): Sistema de logging estructurado

Testing

- **Jest** (v29.7): Test runner y assertions
- **Supertest** (v6.3): Pruebas HTTP/API
- **memfs** (v4.6): Mock de filesystem

Desarrollo

- **Nodemon** (v3): Auto-reload
- **ESLint** (v8.52): Linting
- **Prettier** (v3): Formateo de código

Estructura del Proyecto

```
Hacienda_API/
├─ src/                                # Código fuente principal
│  ├─ server.js                        # Punto de entrada HTTP
│  ├─ app.js                          # Configuración de Express
│  ├─ config/
│  │  └─ index.js                     # Configuración centralizada
│  ├─ controllers/
│  │  ├─ facturaController.js         # Controlador facturas (ES)
│  │  └─ englishInvoiceController.js # Controlador invoices (EN)
│  ├─ routes/
│  │  ├─ facturas.js                 # Rutas facturas (ES)
│  │  └─ englishInvoices.js          # Rutas invoices (EN)
│  ├─ services/
│  │  ├─ atvAdapter.js               # Adaptador ATV (SIMULATED/REAL)
│  │  └─ invoiceStorage.js           # Almacenamiento de facturas
│  ├─ validators/
│  │  ├─ facturaValidator.js          # Validación de facturas (ES)
│  │  └─ englishInvoiceValidator.js   # Validación de invoices (EN)
│  ├─ models/
│  │  └─ facturaModel.js              # Definición de modelos de datos
│  ├─ utils/
│  │  ├─ logger.js                   # Sistema de logging
│  │  └─ filenames.js                # Helpers para nombres de archivo
│  └─ data/
│     └─ consecutivo.json             # Contador de consecutivos
├─ tests/                             # Pruebas unitarias e integración
│  ├─ unit/
│  │  ├─ facturaValidator.test.js
│  │  ├─ invoiceStorage.test.js
│  │  └─ atvAdapter.test.js
│  ├─ integration/
│  │  └─ api.test.js
│  └─ setup.js                       # Configuración global de Jest
├─ docs/                             # Documentación
│  ├─ API_USER_GUIDE.md              # Guía de uso del API
│  ├─ METHODS_INDEX.md               # Índice de métodos
│  ├─ ARCHITECTURE_DIAGRAM.md        # Diagramas de arquitectura
│  ├─ TESTING.md                     # Cómo ejecutar tests
│  ├─ COMPLETE_PAYLOADS.md           # Ejemplos de payloads
│  └─ MANUAL_TECNICO.md              # Este archivo
├─ invoices/                         # Directorio para guardar facturas emitidas
├─ logs/                             # Logs de aplicación
├─ coverage/                         # Reportes de cobertura de tests
├─ postman/                          # Colecciones Postman
└─ scripts/                          # Scripts utilitarios
```

└─ .env.example	# Ejemplo de variables de entorno
└─ jest.config.js	# Configuración de Jest
└─ package.json	# Dependencias y scripts
└─ README.md	# Leeme general

Módulos Principales

1. Server (src/server.js)

Responsabilidad: Crear servidor HTTP, normalizar puertos, manejar errores

```
// Normaliza puerto (número, named pipe o false)
function normalizePort(val)

// Manejo de errores de servidor
function onError(error)

// Logging de servidor en escucha
function onListening()
```

Flujo:

1. Cargar configuración
2. Crear servidor HTTP con Express
3. Escuchar en puerto configurado
4. Registrar errores

2. App (src/app.js)

Responsabilidad: Configurar middleware, rutas y manejo global de errores

Middleware configurado:

- `helmet()` : Headers de seguridad HTTP
- `cors()` : Control de origen según `NODE_ENV`
- `express.json()` : Parsing JSON con validación y límite de tamaño
- `express.urlencoded()` : Parsing de formularios
- Logging de requests
- Inyección de config y logger en request

Rutas principales:

GET /health	# Health check
GET /info	# Información del sistema
POST /api/facturas/emitir	# Emitir factura (ES)
POST /api/facturas/validar	# Validar factura (ES)
POST /api/facturas/enviar	# Enviar a Hacienda (ES)
GET /api/facturas	# Listar facturas (ES)
GET /api/facturas/:consecutivo	# Obtener factura (ES)
POST /api/english-invoices/emit	# Emitir invoice (EN)

```
POST /api/english-invoices/validate # Validar invoice (EN)
GET  /api/english-invoices         # Listar invoices (EN)
```

3. Config (src/config/index.js)

Responsabilidad: Centralizar configuración y detectar modo de operación

Propiedades principales:

```
port          // Puerto del servidor (default: 3000)
nodeEnv       // Ambiente (development/test/production)
logLevel      // Nivel de logging (error/warn/info/debug)
mode          // Modo detectado (SIMULATED/REAL)
atv           // Configuración ATV (keys, certs, credentials)
invoicesDir   // Directorio de almacenamiento
```

Lógica de detección de modo:

```
|— ¿Hay ATV_KEY_PATH + ATV_CERT_PATH + ATV_CLIENT_ID?
| |— SÍ — ¿Archivos existen en filesystem?
| |   |— SÍ → REAL
| |   |— NO → SIMULATED
| |— NO — ¿SIMULATE_IF_NO_KEYS=true?
| |   |— SÍ → SIMULATED
| |   |— NO → ERROR
```

4. ATV Adapter (src/services/atvAdapter.js)

Responsabilidad: Abstracción de comunicación con ATV (Hacienda)

Modo SIMULATED:

- Genera UUIDs como claves de validación
- Retorna respuestas realistas sin contactar Hacienda
- Consecutivos pseudo-aleatorios

Modo REAL:

- Importa @facturacr/atv-sdk (actualmente comentado)
- Usa certificados .p12 y credenciales
- Comunica con APIs reales de Hacienda

Métodos principales:

```
async init(customConfig)           // Inicializar adaptador
async emitirComprobante(facturaData) // Emitir factura
async validarComprobante(facturaData) // Validar estructura
async enviarComprobante(clave, pin)  // Enviar a Hacienda
async obtenerEstado(clave)           // Obtener estado de emisión
```

Respuesta tipo:

```
{
  "success": true,
  "clave": "50602272020110310000100010010000000001",
  "consecutivo": "00100010010000000001",
  "timestamp": "2025-11-20T10:30:00Z",
  "mode": "SIMULATED"
}
```

5. Invoice Storage (src/services/invoiceStorage.js)

Responsabilidad: Persistencia de facturas en filesystem

Estructura de directorios:

```
invoices/
├── FACTURA_[CONSECUTIVO]_[TIMESTAMP].json
├── FACTURA_[CONSECUTIVO]_[TIMESTAMP].xml
└── sent/
    ├── FACTURA_[CONSECUTIVO]_[TIMESTAMP].json
    └── FACTURA_[CONSECUTIVO]_[TIMESTAMP].xml
```

Métodos principales:

```
async saveInvoiceJSON(consecutivo, data)    // Guardar JSON
async saveInvoiceXML(consecutivo, xmlData)   // Guardar XML
async getInvoice(consecutivo)                // Obtener factura
async listInvoices(filters)                 // Listar con filtros
async markAsSent(consecutivo, clave)         // Marcar como enviada
async deleteInvoice(consecutivo)             // Eliminar factura
```

Metadata guardada:

```
{
  consecutivo: "...",
  clave: "...",
  savedAt: "2025-11-20T10:30:00Z",
  sentAt: null,
  status: "PENDING",
  format: "JSON",
  filename: "FACTURA_00100010010000000001_20251120-103000.json"
}
```

6. Validators (src/validators/)

facturaValidator.js (Esquema ES)

Valida estructura completa usando Joi:

- Emisor (nombre, cédula, ubicación)
- Receptor (nombre, cédula)

- Líneas de detalle (cantidad, precio, impuestos)
- Resumen (totales, impuestos, moneda)
- Condiciones (crédito/contado, plazos)

englishInvoiceValidator.js (Esquema EN)

Esquema traducido a inglés con misma lógica

Características:

- Validación de identificación (9-12 dígitos)
- Validación de email y teléfono
- Validación de impuestos (IVA, exención, etc.)
- Validación de moneda (CRC por defecto)
- Mensajes de error detallados en español/inglés

7. Controllers (src/controllers/)

facturaController.js

Métodos principales:

```
static async emitirFactura(req, res) // POST /api/facturas/emitir
static async validarFactura(req, res) // POST /api/facturas/validar
static async enviarFactura(req, res) // POST /api/facturas/enviar
static async obtenerEstadoSistema(req, res) // GET /api/facturas/status
static async listarFacturas(req, res) // GET /api/facturas
static async obtenerFactura(req, res) // GET /api/facturas/:consecutivo
static async descargarFacturaXML(req, res) // GET /api/facturas/:consecutivo/xml
```

Flujo de emitirFactura:

1. Validar payload con FacturaValidator
2. Inicializar adaptador ATV si es necesario
3. Generar consecutivo si no existe
4. Agregar timestamp de emisión
5. Llamar atvAdapter.emitirComprobante()
6. Guardar en invoiceStorage (JSON)
7. Retornar respuesta con clave y consecutivo

englishInvoiceController.js

Mismo flujo pero para esquema EN

8. Logger (src/utls/logger.js)

Sistema de logging con Winston

Transportes configurados:

- **Console:** Output coloreado con timestamp
- **File** (error): logs/error.log
- **File** (combined): logs/combined.log

Niveles: error, warn, info, debug, verbose

Uso:

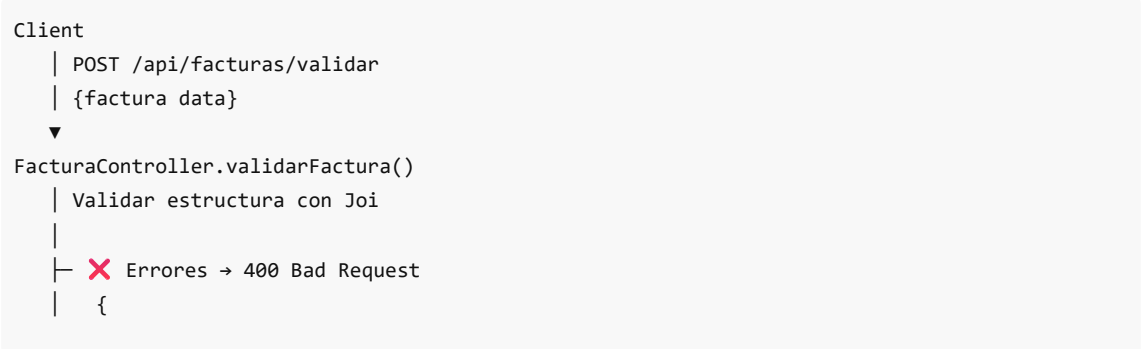
```
logger.info('Mensaje informativo')
logger.error('Error importante', { errorData: true })
logger.debug('Información de debug')
```

Flujos de Datos

Flujo 1: Emitir Factura (Modo SIMULADO)



Flujo 2: Validar Factura



```
|      success: false,
|      errors: [{field, message}]
|    }
|
└─ ✓ Válido
    | Validar lógica de negocio
    | (totales cuadren, impuestos correctos)
    ▼
    Response 200 OK
    {
      success: true,
      message: "Factura válida"
    }
```

Flujo 3: Enviar a Hacienda (Modo REAL)

```
Client
| POST /api/facturas/enviar
| {clave, pin}
▼
FacturaController.enviarFactura()
| Validar clave y PIN
| Recuperar factura del storage
▼
atvAdapter.enviarComprobante(clave, pin)
| (Modo REAL)
├─ Cargar certificado .p12
├─ Usar @facturacr/atv-sdk
├─ Conectar con servicios de Hacienda
▼
Respuesta de Hacienda
├─ ✓ Aceptada → Status "SENT"
└─ ✗ Rechazada → Status "REJECTED"
    |
    ▼
invoiceStorage.markAsSent(consecutivo)
| Actualizar metadata
▼
Response
{
  success: boolean,
  status: "SENT|REJECTED",
  haciendaResponse: {...}
}
```

Configuración Detallada

Variables de Entorno (.env)

```
# SERVIDOR
PORT=3000
NODE_ENV=development

# ATV (Hacienda) - Deixar vacíos para modo SIMULADO
ATV_KEY_PATH=
ATV_CERT_PATH=
ATV_CLIENT_ID=
ATV_USERNAME=
ATV_PIN=

# SIMULACIÓN
SIMULATE_IF_NO_KEYS=true

# STORAGE
INVOICES_DIR=./invoices
CONSECUTIVE_FILE=./src/data/consecutivo.json

# LOGGING
LOG_LEVEL=info

# LIMITS
MAX_FILE_SIZE=10MB
ALLOWED_MIME_TYPES=application/json,application/xml,application/pdf

# CORS
ALLOWED_ORIGINS=http://localhost:3000,http://localhost:3001
```

Detectar Modo Actual

Hacer GET a `/info` :

```
{
  "status": "OK",
  "mode": "SIMULATED",
  "nodeEnv": "development",
  "version": "1.0.0"
}
```

APIs REST

1. POST `/api/facturas/emitir`

Emitir nueva factura

Parámetros (Body JSON):

```
{
  "emisor": {
    "nombre": "Mi Empresa S.A.",
```

```

    "identificacion": "3101234567",
    "tipoIdentificacion": "02",
    "correoElectronico": "facturacion@miempresa.cr",
    "provincia": "01",
    "canton": "01",
    "distrito": "01"
  },
  "receptor": {
    "nombre": "Cliente S.A.",
    "identificacion": "123456789",
    "tipoIdentificacion": "01"
  },
  "detalleServicio": [
    {
      "numeroLinea": 1,
      "cantidad": 1,
      "unidadMedida": "UNI",
      "detalle": "Servicio profesional",
      "precioUnitario": 1000.00,
      "montoTotalLinea": 1000.00,
      "impuestos": [
        {
          "codigo": "01",
          "tarifa": 13.00,
          "monto": 130.00
        }
      ]
    }
  ]
},
"resumenFactura": {
  "totalVenta": 1000.00,
  "totalDescuentos": 0.00,
  "totalVentaNeta": 1000.00,
  "totalImpuesto": 130.00,
  "totalComprobante": 1130.00,
  "codigoMoneda": "CRC"
},
"condicionVenta": "Contado",
"mediosPago": ["01"]
}

```

Respuesta (201 Created):

```

{
  "success": true,
  "clave": "50602272020110310000100010010000000001",
  "consecutivo": "00100010010000000001",
  "numero": "00100010",
  "serie": "010",
  "timestamp": "2025-11-20T10:30:00Z",
  "savedAt": "2025-11-20T10:30:00Z",

```

```
"filePath": "./invoices/FACTURA_00100010010000000001_20251120-103000.json",
"mode": "SIMULATED"
}
```

2. POST /api/facturas/validar

Validar estructura de factura

Parámetros: Mismo formato que emitir

Respuesta (200 OK):

```
{
  "success": true,
  "valid": true,
  "message": "Factura válida",
  "warnings": []
}
```

Respuesta (400 Bad Request - Validación fallida):

```
{
  "success": false,
  "valid": false,
  "errors": [
    {
      "field": "emisor.identificacion",
      "message": "La identificación debe contener entre 9 y 12 dígitos"
    }
  ]
}
```

3. POST /api/facturas/enviar

Enviar factura a Hacienda

Parámetros (Body JSON):

```
{
  "clave": "50602272020110310000100010010000000001",
  "pin": "1234"
}
```

Respuesta (200 OK):

```
{
  "success": true,
  "status": "SENT",
  "message": "Factura enviada exitosamente",
}
```

```
"sentAt": "2025-11-20T10:30:00Z"
}
```

4. GET /api/facturas

Listar facturas

Parámetros (Query):

- `status` : "all" | "pending" | "sent" (default: "all")
- `limit` : número (default: 50)
- `offset` : número (default: 0)
- `includeContent` : "true" | "false" (default: "false")

Respuesta (200 OK):

```
{
  "success": true,
  "total": 5,
  "limit": 50,
  "offset": 0,
  "facturas": [
    {
      "consecutivo": "00100010010000000001",
      "clave": "50602272020110310000100010010000000001",
      "savedAt": "2025-11-20T10:30:00Z",
      "status": "PENDING",
      "emisor": "Mi Empresa S.A.",
      "receptor": "Cliente S.A.",
      "monto": 1130.00,
      "moneda": "CRC"
    }
  ]
}
```

5. GET /api/facturas/:consecutivo

Obtener factura específica

Parámetros (URL):

- `consecutivo` : string (ej: "00100010010000000001")

Respuesta (200 OK):

```
{
  "success": true,
  "factura": {
    "consecutivo": "...",
    "clave": "...",
    "emisor": {...},
    "receptor": {...},
  }
}
```

```
"detalleServicio": [...],
"resumenFactura": {...},
"metadata": {
  "savedAt": "...",
  "sentAt": null,
  "status": "PENDING"
}
}
```

6. GET /api/facturas/:consecutivo/xml

Descargar factura en formato XML

Parámetros (URL):

- consecutivo : string

Respuesta (200 OK - Content-Type: application/xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<Factura>
  <!-- XML de factura en formato Hacienda -->
</Factura>
```

7. GET /api/facturas/status

Obtener estado del sistema

Respuesta (200 OK):

```
{
  "success": true,
  "status": "OK",
  "mode": "SIMULATED",
  "totalFacturas": 5,
  "storageDir": "./invoices",
  "lastError": null,
  "timestamp": "2025-11-20T10:30:00Z"
}
```

Endpoints EN (English)

Equivalentes bilingües:

POST	/api/english-invoices/emit	≈	POST /api/facturas/emitir
POST	/api/english-invoices/validate	≈	POST /api/facturas/validar
GET	/api/english-invoices	≈	GET /api/facturas
GET	/api/english-invoices/:number	≈	GET /api/facturas/:consecutivo

Modelos de Datos

Estructura Emisor

```
{
  nombre: string (1-100 chars, requerido),
  identificacion: string (9-12 dígitos, requerido),
  tipoIdentificacion: "01"|"02"|"03"|"04" (default: "02"),
  // 01: Física, 02: Jurídica, 03: DIMEX, 04: NITE
  correoElectronico: string (email válido, opcional),
  telefono: string (máx 20 chars, opcional),
  codigoPais: string (default: "506"),
  provincia: string (opcional),
  canton: string (opcional),
  distrito: string (opcional),
  direccion: string (máx 200 chars, opcional)
}
```

Estructura Receptor

```
{
  nombre: string (1-100 chars, requerido),
  identificacion: string (9-12 dígitos, requerido),
  tipoIdentificacion: "01"|"02"|"03"|"04" (default: "01"),
  correoElectronico: string (opcional),
  telefono: string (opcional)
}
```

Estructura Línea de Detalle

```
{
  numeroLinea: integer (requerido),
  cantidad: number (requerido, > 0),
  unidadMedida: string (UNI, KG, L, HR, etc.),
  detalle: string (1-1000 chars, descripción del servicio),
  precioUnitario: number (requerido, >= 0),
  montoTotalLinea: number (cantidad × precioUnitario),
  descuentoMonto: number (opcional, descuento línea),
  impuestos: [
    {
      codigo: string ("01" para IVA, etc.),
      tarifa: number (0-100, %),
      monto: number (monto del impuesto)
    }
  ]
}
```

Estructura Resumen

```
{
  totalServGravados: number (total servicios gravados),
  totalServExentos: number (total servicios exentos),
  totalMercanciasGravadas: number,
  totalMercanciasExentas: number,
  totalGravado: number,
  totalExento: number,
  totalVenta: number (suma de líneas),
  totalDescuentos: number,
  totalVentaNeta: number (totalVenta - descuentos),
  totalImpuesto: number (suma de todos los impuestos),
  totalComprobante: number (totalVentaNeta + totalImpuesto),
  codigoMoneda: "CRC"|"USD"|... (default: "CRC"),
  tipoCambio: number (si moneda != CRC)
}
```

Manejo de Errores

Códigos HTTP

- **200 OK:** Solicitud exitosa
- **201 Created:** Recurso creado exitosamente
- **400 Bad Request:** Datos inválidos o estructura incorrecta
- **404 Not Found:** Recurso no encontrado
- **500 Internal Server Error:** Error del servidor
- **503 Service Unavailable:** ATV no disponible (REAL mode)

Formato de Error Estándar

```
{
  "success": false,
  "error": "Descripción del error",
  "details": {
    "field": "valor",
    "message": "Explicación específica"
  },
  "timestamp": "2025-11-20T10:30:00Z"
}
```

Errores Comunes

Validación fallida

```
{
  "success": false,
  "error": "Datos de factura inválidos",
  "details": [
```

```
{
  "field": "emisor.identificacion",
  "message": "La identificación debe contener entre 9 y 12 dígitos"
}
]
```

Consecutivo no encontrado

```
{
  "success": false,
  "error": "Factura no encontrada",
  "consecutive": "00100010010000000001"
}
```

Error de almacenamiento

```
{
  "success": false,
  "error": "Error al guardar factura",
  "details": "Error message from fs"
}
```

Testing

Ejecución de Tests

```
# Todos los tests
npm test

# Con coverage
npm run test:coverage

# Modo watch
npm run test:watch

# Test específico
npx jest tests/unit/facturaValidator.test.js
```

Estructura de Tests

```
tests/
├─ unit/                                # Tests unitarios (aislados)
│  ├─ facturaValidator.test.js          # Validación ES
│  ├─ englishInvoiceValidator.test.js   # Validación EN
│  ├─ invoiceStorage.test.js           # Persistencia
│  └─ atvAdapter.test.js               # ATV adapter
```

```
|— integration/                # Tests de integración (full stack)
|   |— api.test.js            # Endpoints HTTP completos
|   |— setup.js               # Configuración global
```

Cobertura Esperada

- **Validators:** 95%+ (lógica crítica)
- **Services:** 80%+ (lógica de negocio)
- **Controllers:** 75%+ (integración)
- **Global:** 80%+ (threshold del proyecto)

Mocking

- **fs-extra:** Mock en `tests/__mocks__/fs-extra.js`
- **ATV SDK:** Mock en `atvAdapter` (modo SIMULATED)
- **HTTP:** Supertest para requests reales

Deploying

Requisitos de Producción

- Node.js 16+ en servidor
- Variables de entorno configuradas
- Certificados ATV (.p12) en ruta segura
- Acceso a filesystem para almacenamiento
- Base de datos (opcional, para persistencia mejorada)

Paso a Modo REAL

1. Obtener certificados de Hacienda (.p12 + key)
2. Configurar variables `ATV_KEY_PATH`, `ATV_CERT_PATH`, `ATV_CLIENT_ID`
3. Instalar SDK: `npm install @facturacr/atv-sdk`
4. Descomentar código real en `atvAdapter.js`
5. Probar en ambiente de staging primero
6. Validar respuestas reales

Ejemplo Configuración REAL

```
PORT=3000
NODE_ENV=production
ATV_KEY_PATH=/etc/secrets/hacienda/private.key
ATV_CERT_PATH=/etc/secrets/hacienda/certificate.p12
ATV_CLIENT_ID=123456789-XXXXX
ATV_USERNAME=usuario
ATV_PIN=1234
SIMULATE_IF_NO_KEYS=false
LOG_LEVEL=warn
```

Docker (Recomendado)

```
FROM node:18-alpine
WORKDIR /app
```

```
COPY package*.json ./
RUN npm ci --only=production
COPY src ./src
EXPOSE 3000
CMD ["npm", "start"]
```

```
docker build -t hacienda-api .
docker run -p 3000:3000 -e NODE_ENV=production hacienda-api
```

CI/CD (GitHub Actions)

Incluido en `.github/workflows/tests.yml` :

- Ejecutar tests en cada push/PR
- Generar reporte de cobertura
- Build y deploy automático a staging/production

Troubleshooting

Problema: "Error al conectar con ATV"

Solución: Verificar que está en modo SIMULADO o que certificados existen

Problema: "Directorio invoices no encontrado"

Solución: Crear manualmente `./invoices` o usar script `setup.sh`

Problema: "Module not found: fs-extra"

Solución: Ejecutar `npm install`

Problema: "Jest config error"

Solución: Verificar `jest.config.js` tiene `moduleNameMapper` (no `moduleNameMapping`)

Contacto & Soporte

Para reportar bugs o sugerencias:

1. Revisar logs en `logs/`
2. Ejecutar tests para aislar problema
3. Verificar variables de entorno
4. Consultar documentación en `docs/`

Versión: 1.0.0

Última actualización: Noviembre 2025

Mantener actualizado: Revisar cada 3 meses