Documentación Completa de Métodos - Hacienda API

📋 Tabla de Contenidos

---

1. Controladores

1.1 FacturaController

Ubicación: `src/controllers/facturaController.js`
Propósito: Maneja todas las operaciones de facturas electrónicas en español

Métodos Principales:

`emitirFactura(req, res)` - POST /api/facturas/emitir

Propósito: Emite una nueva factura electrónica
Parámetros:

- `req.body` : Datos completos de la factura
- `res` : Objeto de respuesta HTTP

Funcionalidad:

1. Valida la estructura de la factura usando `FacturaValidator`
2. Inicializa el adaptador ATV si es necesario
3. Genera número consecutivo si no está presente
4. Emite la factura a través del `atvAdapter`
5. Guarda archivos JSON y XML en el sistema
6. Retorna información de la factura emitida

Uso:

```
// Request body ejemplo:
{
  "tipoDocumento": "01",
  "codigoMoneda": "CRC",
  "emisor": { /* datos del emisor */ },
  "receptor": { /* datos del receptor */ },
  "detalleServicio": [/* items de la factura */],
  "resumenFactura": { /* totales */ }
}
```

Respuesta:

```
{
  "success": true,
  "consecutivo": "12345678901234567890",
  "clave": "506...",
  "estado": "emitida",
  "mode": "SIMULATED",
  "archivos": {
    "json": "/path/to/json",
    "xml": "/path/to/xml"
  }
}
```

`validarFactura(req, res)` **- POST /api/facturas/validar**

Propósito: Valida una factura por clave, consecutivo o payload
Parámetros:

- `req.body.clave` : Clave de 50 caracteres (opcional)
- `req.body.consecutivo` : Número consecutivo de 20 dígitos (opcional)
- `req.body.payload` : Datos completos de factura para validación estructural (opcional)

Funcionalidad:

1. Verifica que se proporcione al menos uno de los parámetros
2. Si hay payload, realiza validación estructural
3. Si hay consecutivo sin clave, busca la factura guardada
4. Valida formato de clave usando `FacturaValidator`
5. Ejecuta validación a través del `atvAdapter`
6. Retorna resultado de validación

Casos de Uso:

```
// Validar por clave
{ "clave": "50612345..." }

// Validar por consecutivo
{ "consecutivo": "12345678901234567890" }

// Validar estructura de datos
{ "payload": { /* datos de factura */ } }
```

`enviarFactura(req, res)` **- POST /api/facturas/enviar**

Propósito: Envía una factura al sistema de Hacienda
Parámetros:

- `req.body.clave` : Clave de la factura (opcional)
- `req.body.consecutivo` : Número consecutivo (opcional)

Funcionalidad:

1. Valida que se proporcione clave o consecutivo
2. Si solo hay consecutivo, busca la clave en el archivo guardado

3. Envía la factura usando `atvAdapter.enviarComprobante()`
4. Marca archivos como enviados moviéndolos a carpeta 'sent'
5. Retorna confirmación de envío

`consultarFactura(req, res)` - GET /api/facturas/:consecutivo

Propósito: Consulta una factura específica por número consecutivo
Parámetros:

- `req.params.consecutivo` : Número consecutivo de 20 dígitos
- `req.query.includeContent` : Si incluir contenido completo ('true'/'false')

Funcionalidad:

1. Valida formato del consecutivo
2. Busca la factura en el almacenamiento local
3. Opcionalmente consulta estado en ATV
4. Retorna datos de la factura con metadatos

`listarFacturas(req, res)` - GET /api/facturas

Propósito: Lista facturas con filtros y paginación
Parámetros de Query:

- `status` : Estado de las facturas ('all', 'sent', 'pending', 'error')
- `includeContent` : Si incluir contenido ('true'/'false')
- `limit` : Máximo número de resultados (default: 50)
- `offset` : Offset para paginación (default: 0)

Funcionalidad:

1. Aplica filtros de estado
2. Implementa paginación
3. Opcionalmente incluye contenido de facturas
4. Retorna lista con metadatos de paginación

`obtenerEstadoSistema(req, res)` - GET /api/facturas/status

Propósito: Obtiene información del estado del sistema
Funcionalidad:

1. Recopila estadísticas del adaptador ATV
2. Obtiene estadísticas de almacenamiento
3. Información de configuración del sistema
4. Retorna estado completo del sistema

`eliminarFactura(req, res)` - DELETE /api/facturas/:consecutivo

Propósito: Elimina una factura (solo en desarrollo)
Restricciones: Solo funciona en modo desarrollo (`NODE_ENV !== 'production'`)
Funcionalidad:

1. Valida formato de consecutivo
2. Elimina archivos del almacenamiento
3. Retorna confirmación de eliminación

1.2 EnglishInvoiceController

Ubicación: `src/controllers/englishInvoiceController.js`
Propósito: Maneja operaciones de facturas en inglés para clientes internacionales

Métodos Principales:

`issueInvoice(req, res)` - POST /api/en/invoices/issue

Propósito: Emite facturas usando terminología en inglés
Funcionalidad:

1. Valida estructura usando `EnglishInvoiceValidator`
2. Convierte datos de inglés a español para compatibilidad ATV
3. Procesa la factura igual que el controlador español
4. Guarda datos originales en inglés y convertidos en español

Campos en Inglés:

```
{
  "documentType": "01",
  "currencyCode": "USD",
  "issuer": { /* datos del emisor */ },
  "receiver": { /* datos del receptor */ },
  "serviceDetail": [/* detalles de servicios */],
  "invoiceSummary": { /* resumen de factura */ }
}
```

`validateInvoice(req, res)` - POST /api/en/invoices/validate

Propósito: Validación de facturas en formato inglés
Funcionalidad:

1. Acepta validación por key/consecutive o payload
2. Valida estructura en inglés
3. Convierte a español para validación ATV
4. Retorna resultados en formato inglés

`sendInvoice(req, res)` - POST /api/en/invoices/send

Propósito: Envío de facturas con interface en inglés
Funcionalidad: Similar a `enviarFactura` pero con terminología inglesa

`queryInvoice(req, res)` - GET /api/en/invoices/:consecutive

Propósito: Consulta facturas con respuesta en inglés
Funcionalidad:

1. Busca factura por consecutive number
2. Convierte datos a formato inglés si están en español
3. Retorna datos bilingües cuando están disponibles

`listInvoices(req, res)` - GET /api/en/invoices

Propósito: Lista facturas con metadatos en inglés
Funcionalidad:

1. Lista facturas con filtros
2. Convierte metadatos a inglés
3. Proporciona datos bilingües cuando posible

`convertToSpanish(englishInvoice)` - Método Estático

Propósito: Convierte estructura de factura de inglés a español
Funcionalidad:

1. Mapea todos los campos de inglés a español
2. Conserva estructura compatible con ATV
3. Traduce unidades de medida y códigos

Mapeo de Campos:

```
{
  issuer → emisor,
  receiver → receptor,
  serviceDetail → detalleServicio,
  invoiceSummary → resumenFactura,
  documentType → tipoDocumento,
  // ... más mapeos
}
```

---

2. Servicios

2.1 ATVAdapter

Ubicación: `src/services/atvAdapter.js`
Propósito: Adaptador para el SDK de ATV con modo dual (REAL/SIMULADO)

Métodos Principales:

`init(customConfig = {})`

Propósito: Inicializa el adaptador en modo REAL o SIMULADO
Funcionalidad:

1. Determina el modo basado en configuración
2. En modo REAL: Inicializa SDK oficial de ATV
3. En modo SIMULADO: Configura respuestas simuladas
4. Establece `isInitialized = true`

`emitirComprobante(facturaData)`

Propósito: Emite un comprobante electrónico
Parámetros: `facturaData` - Datos completos de la factura

Modo REAL:

- Utiliza SDK oficial de ATV
- Conecta con sistema real de Hacienda

Modo SIMULADO:

- Genera clave simulada de 50 caracteres

- Crea XML simulado
- Retorna respuesta con estructura idéntica al modo real

Respuesta:

```
{
  success: true,
  consecutivo: "12345678901234567890",
  clave: "506...",
  estado: "emitida",
  xml: "<FacturaElectronica>...</FacturaElectronica>",
  mode: "SIMULATED",
  timestamp: "2025-11-07T..."
}
```

`validarComprobante(clave)`

Propósito: Valida un comprobante por su clave
Parámetros: `clave` - Clave de 50 caracteres

Funcionalidad:

- Modo REAL: Consulta validación real en Hacienda
- Modo SIMULADO: Retorna validación simulada positiva
- Incluye hash de validación y mensajes

`enviarComprobante(clave)`

Propósito: Envía un comprobante al sistema de Hacienda
Parámetros: `clave` - Clave de 50 caracteres

Funcionalidad:

- Simula o ejecuta envío real
- Genera número de comprobante
- Retorna respuesta de Hacienda

`consultarComprobante(clave)`

Propósito: Consulta el estado de un comprobante
Parámetros: `clave` - Clave de 50 caracteres

Funcionalidad:

- Consulta estado actual del comprobante
- Retorna información de procesamiento
- Incluye timestamps y estados de validación

`getStatus()`

Propósito: Obtiene estado actual del adaptador
Retorna:

```
{
  initialized: boolean,
```

```
  mode: "REAL" | "SIMULATED",
  sdkVersion: string,
  lastActivity: timestamp
}
```

2.2 InvoiceStorage

Ubicación: `src/services/invoiceStorage.js`
Propósito: Gestiona almacenamiento de facturas en sistema de archivos

Métodos Principales:

`saveInvoiceJSON(consecutivo, facturaData)`

Propósito: Guarda factura en formato JSON
Parámetros:

- `consecutivo` : Número consecutivo
- `facturaData` : Datos de la factura

Funcionalidad:

1. Genera nombre de archivo con timestamp
2. Agrega metadatos de almacenamiento
3. Guarda en directorio de facturas
4. Retorna ruta del archivo guardado

Formato de Archivo: `FACTURA_{consecutivo}_{timestamp}.json`

`saveInvoiceXML(consecutivo, xmlContent)`

Propósito: Guarda XML de factura
Parámetros:

- `consecutivo` : Número consecutivo
- `xmlContent` : Contenido XML como string

Formato de Archivo: `FACTURA_{consecutivo}.xml`

`markAsSent(consecutivo, envioMeta = {})`

Propósito: Marca factura como enviada moviéndola a carpeta 'sent'
Funcionalidad:

1. Busca archivos relacionados con el consecutivo
2. Mueve archivos JSON y XML a carpeta 'sent'
3. Actualiza metadatos con información de envío
4. Retorna resumen de archivos movidos

`getInvoice(consecutivo, includeContent = true)`

Propósito: Recupera una factura específica
Funcionalidad:

1. Busca archivos por consecutivo en todas las carpetas
2. Opcionalmente carga contenido completo
3. Retorna metadatos y contenido si se solicita

Respuesta:

```
{
  found: true,
  files: {
    json: { path: "...", clave: "..." },
    xml: { path: "..." }
  },
  metadata: {
    createdAt: "...",
    status: "sent",
    fileSize: 1024
  },
  content: {
    json: { /* datos de factura */ },
    xml: "XML content"
  }
}
```

`listInvoices(options = {})`

Propósito: Lista facturas con filtros y paginación
Opciones:

- `status` : Filtrar por estado
- `includeContent` : Incluir contenido
- `limit` : Límite de resultados
- `offset` : Offset para paginación

Funcionalidad:

1. Escanea directorios de facturas
2. Aplica filtros de estado
3. Implementa paginación
4. Calcula estadísticas
5. Opcionalmente carga contenido

`deleteInvoice(consecutivo)`

Propósito: Elimina factura (solo desarrollo)
Funcionalidad:

1. Busca todos los archivos relacionados
2. Elimina archivos JSON y XML
3. Retorna resumen de eliminación

`getStatistics()`

Propósito: Obtiene estadísticas de almacenamiento
Retorna:

```
{
  total: 150,
  byStatus: {
```

```
    pending: 25,
    sent: 120,
    error: 5
  },
  totalSize: "15.6 MB",
  directories: {
    main: "/path/invoices",
    sent: "/path/invoices/sent"
  }
}
```

---

3. Validadores

3.1 FacturaValidator

Ubicación: `src/validators/facturaValidator.js`
Propósito: Validación completa de facturas electrónicas usando Joi

Métodos Estáticos:

`validateFactura(factura, options = {})`

Propósito: Validación completa de factura
Funcionalidad:

1. Valida estructura usando esquema Joi
2. Ejecuta validaciones de lógica de negocio
3. Verifica cálculos matemáticos
4. Retorna resultado detallado con errores

Opciones:

- `strict` : Validación estricta (default: true)
- `allowUnknown` : Permitir campos desconocidos
- `skipBusinessLogic` : Saltar validación de negocio

`validateClave(clave)`

Propósito: Valida formato de clave de comprobante
Reglas:

- Exactamente 50 caracteres
- Solo caracteres alfanuméricos
- Campo requerido

`validateConsecutivo(consecutivo)`

Propósito: Valida formato de número consecutivo
Reglas:

- Exactamente 20 dígitos
- Solo números
- Campo requerido

`validateEmisor(emisor)`

Propósito: Valida datos del emisor
Campos Validados:

- `nombre` : String, requerido, 1-100 caracteres
- `identificacion` : String, requerido, formato específico
- `tipoIdentificacion` : Código válido ('01', '02', '03', '04')
- `correoElectronico` : Email válido
- `telefono` : Formato de teléfono

`validateReceptor(receptor)`

Propósito: Valida datos del receptor
Similar a emisor con campos específicos del receptor

`validateDetalle(detalle)`

Propósito: Valida array de líneas de servicio
Reglas:

- Array de 1-1000 elementos
- Cada elemento valida cantidad, precios, impuestos
- Validación matemática de totales por línea

`validateTotales(totales)`

Propósito: Valida resumen de totales
Campos Validados:

- Totales de servicios gravados/exentos
- Total de impuestos
- Total de descuentos
- Total final de comprobante

`validateBusinessLogic(factura)`

Propósito: Validaciones de lógica de negocio
Validaciones:

1. **Coherencia Matemática**: Suma de líneas = totales
2. **Impuestos**: Cálculo correcto de IVA
3. **Descuentos**: Aplicación correcta
4. **Totales**: Verificación de sumas finales

Ejemplo de Error:

```
{
  field: 'resumenFactura.totalComprobante',
  message: 'Total calculado (113.00) no coincide con total declarado (115.00)',
  code: 'MATHEMATICAL_ERROR',
  calculated: 113.00,
  declared: 115.00
}
```

`getSchemas()`

Propósito: Retorna todos los esquemas Joi disponibles
Uso: Para validaciones granulares o debugging

3.2 EnglishInvoiceValidator

Ubicación: `src/validators/englishInvoiceValidator.js`
Propósito: Validación de facturas en inglés con conversión automática

Métodos de la Clase:

`validateInvoice(invoiceData)`

Propósito: Validación completa en formato inglés
Funcionalidad:

1. Valida estructura en inglés usando esquemas Joi
2. Ejecuta validaciones de lógica de negocio
3. Retorna errores en terminología inglesa

Campos en Inglés:

```
{
  issuer: { name, identification, email, ... },
  receiver: { name, identification, email, ... },
  serviceDetail: [{ lineNumber, description, quantity, unitPrice, ... }],
  invoiceSummary: { totalTaxable, totalTax, totalInvoice, ... }
}
```

`validateIssuer(issuer)`

Propósito: Validación específica de emisor en inglés
Campos: `name`, `identification`, `identificationType`, `email`, `phone`

`validateReceiver(receiver)`

Propósito: Validación específica de receptor en inglés

`validateServiceDetail(serviceDetail)`

Propósito: Validación de líneas de servicio en inglés
Campos por línea:

- `lineNumber`: Número de línea
- `description`: Descripción del servicio
- `quantity`: Cantidad
- `unitPrice`: Precio unitario
- `tax.rate`: Tasa de impuesto
- `totalLineAmount`: Total de línea

`validateInvoiceSummary(summary)`

Propósito: Validación de resumen en inglés
Campos:

- `totalTaxableServices`: Total servicios gravados
- `totalTax`: Total impuestos

- `totalInvoice` : Total de factura

`convertFromSpanish(spanishInvoice)` - Estático

Propósito: Convierte factura de español a inglés
Funcionalidad:

1. Mapea campos de español a inglés
2. Traduce códigos y valores
3. Mantiene estructura compatible

`convertToSpanish(englishInvoice)` - Estático

Propósito: Convierte factura de inglés a español
Funcionalidad: Inverso del método anterior

---

4. Utilidades

4.1 Logger

Ubicación: `src/utils/logger.js`
Propósito: Sistema de logging estructurado usando Winston

Métodos Principales:

`info(message, meta = {})`

Propósito: Log de información general
Uso: Eventos normales del sistema

`warn(message, meta = {})`

Propósito: Log de advertencias
Uso: Situaciones que requieren atención pero no son errores

`error(message, meta = {})`

Propósito: Log de errores
Uso: Errores que requieren investigación

`debug(message, meta = {})`

Propósito: Log de debugging
Uso: Información técnica detallada

`logRequest(req, res, next)`

Propósito: Middleware para logging de requests HTTP
Funcionalidad:

1. Registra método, URL, IP
2. Mide tiempo de respuesta
3. Registra código de estado
4. Filtra información sensible

`logError(error, requestMeta = {})`

Propósito: Logging estructurado de errores
Funcionalidad:

1. Registra stack trace completo

2. Incluye contexto de request
3. Categoriza tipos de error
4. Facilita debugging

Configuración de Transports:

- **Console**: Output colorizado para desarrollo
- **File (error.log)**: Solo errores, rotación automática
- **File (combined.log)**: Todos los niveles, rotación automática
- **File (access.log)**: Logs de acceso HTTP

4.2 Filenames

Ubicación: `src/utils/filenames.js`
Propósito: Utilidades para generación de nombres de archivo y consecutivos

Funciones:

`generateConsecutivo()`

Propósito: Genera número consecutivo único
Formato: 20 dígitos basado en timestamp
Ejemplo: `20251107143025123456`

`generateTimestamp()`

Propósito: Genera timestamp para nombres de archivo
Formato: `YYYYMMDD_HHMMSS`
Ejemplo: `20251107_143025`

`sanitizeFilename(filename)`

Propósito: Sanitiza nombres de archivo
Funcionalidad:

1. Remueve caracteres no válidos
2. Reemplaza espacios con guiones bajos
3. Limita longitud

`validateConsecutivoFormat(consecutivo)`

Propósito: Valida formato de consecutivo
Reglas: Exactamente 20 dígitos numéricos

---

5. Modelos

5.1 FacturaModel

Ubicación: `src/models/facturaModel.js`
Propósito: Define estructuras de datos y ejemplos de facturas

Estructuras Principales:

`EjemploFactura`

Propósito: Factura completa de ejemplo para testing
Contenido:

- Datos completos de emisor y receptor

- Línea de servicio con impuestos
- Resumen de totales calculados
- Metadatos de ejemplo

`EjemploEmisor`

Propósito: Estructura de emisor válida

`EjemploReceptor`

Propósito: Estructura de receptor válida

`EjemploDetalleServicio`

Propósito: Array de líneas de servicio de ejemplo

`EjemploResumenFactura`

Propósito: Resumen de totales de ejemplo

Uso:

```
const { EjemploFactura } = require('../models/facturaModel');

// Usar en tests
const validationResult = FacturaValidator.validateFactura(EjemploFactura);

// Usar como template
const nuevaFactura = {
  ...EjemploFactura,
  emisor: { ...EjemploFactura.emisor, nombre: 'Mi Empresa' }
};
```

---

6. Rutas

6.1 Facturas (Español)

Ubicación: `src/routes/facturas.js`

Rutas Definidas:

`POST /api/facturas/emitir`

Controlador: `FacturaController.emitirFactura`
Propósito: Emisión de facturas en español

`POST /api/facturas/validar`

Controlador: `FacturaController.validarFactura`
Propósito: Validación de facturas

`POST /api/facturas/enviar`

Controlador: `FacturaController.enviarFactura`
Propósito: Envío a Hacienda

`GET /api/facturas/status`

Controlador: `FacturaController.obtenerEstadoSistema`
Propósito: Estado del sistema

`GET /api/facturas`

Controlador: `FacturaController.listarFacturas`
Propósito: Listado con paginación

`GET /api/facturas/:consecutivo`

Controlador: `FacturaController.consultarFactura`
Propósito: Consulta específica

`DELETE /api/facturas/:consecutivo`

Controlador: `FacturaController.eliminarFactura`
Propósito: Eliminación (solo desarrollo)

6.2 English Invoices

Ubicación: `src/routes/englishInvoices.js`

Rutas Definidas:

`POST /api/en/invoices/issue`

Controlador: `EnglishInvoiceController.issueInvoice`
Propósito: Emisión en inglés

`POST /api/en/invoices/validate`

Controlador: `EnglishInvoiceController.validateInvoice`
Propósito: Validación en inglés

`POST /api/en/invoices/send`

Controlador: `EnglishInvoiceController.sendInvoice`
Propósito: Envío en inglés

`GET /api/en/invoices/:consecutive`

Controlador: `EnglishInvoiceController.queryInvoice`
Propósito: Consulta en inglés

`GET /api/en/invoices`

Controlador: `EnglishInvoiceController.listInvoices`
Propósito: Listado en inglés

`GET /api/en/invoices/health/check`

Propósito: Health check para API en inglés

---

7. Configuración

7.1 Config

Ubicación: `src/config/index.js`
Propósito: Configuración centralizada del sistema

Métodos y Propiedades:

`mode`

Propósito: Determina modo de operación ('REAL' o 'SIMULATED')
Basado en: Variables de entorno y disponibilidad de credenciales

`isDevelopment()`

Propósito: Verifica si está en modo desarrollo
Retorna: `true` si `NODE_ENV === 'development'`

`isProduction()`

Propósito: Verifica si está en modo producción
Retorna: `true` si `NODE_ENV === 'production'`

`atv`

Propósito: Configuración específica de ATV
Propiedades:

- `keyPath` : Ruta de llaves privadas
- `certPath` : Ruta de certificados
- `environment` : Entorno ('sandbox' o 'production')

`database`

Propósito: Configuración de base de datos (futura implementación)

`storage`

Propósito: Configuración de almacenamiento
Propiedades:

- `invoicesDir` : Directorio de facturas
- `maxFileSize` : Tamaño máximo de archivo
- `retentionDays` : Días de retención

---

8. Aplicación Principal

8.1 App.js

Ubicación: `src/app.js`
Propósito: Configuración principal de Express

Middleware Configurado:

Seguridad

- `helmet()` : Headers de seguridad
- `cors()` : Control de acceso entre dominios

Parsing

- `express.json()` : Parsing JSON con límite de tamaño
- `express.urlencoded()` : Parsing de forms

Logging

- `logger.logRequest` : Middleware de logging de requests

Rutas Principales

- `/health` : Health check básico
- `/info` : Información del sistema
- `/api/facturas/*` : Rutas en español
- `/api/en/invoices/*` : Rutas en inglés

Manejo de Errores

- Middleware de rutas no encontradas (404)

- Middleware de manejo de errores global
- Logging estructurado de errores
- Respuestas diferentes para desarrollo y producción

8.2 Server.js

Ubicación: `src/server.js`
Propósito: Punto de entrada del servidor

Funcionalidad:

1. Carga configuración
2. Inicializa logger
3. Arranca servidor Express
4. Maneja señales de terminación
5. Configuración de puerto dinámico

---

9. Patrones de Uso Comunes

9.1 Flujo Completo de Facturación

```
// 1. Validar datos
const validation = FacturaValidator.validateFactura(facturaData);
if (!validation.valid) {
  return { error: validation.errors };
}

// 2. Inicializar ATV
if (!atvAdapter.isInitialized) {
  await atvAdapter.init();
}

// 3. Emitir factura
const emissionResult = await atvAdapter.emitirComprobante(validation.data);

// 4. Guardar archivos
const jsonPath = await invoiceStorage.saveInvoiceJSON(
  emissionResult.consecutivo,
  emissionResult
);

// 5. Enviar a Hacienda (opcional)
const sendResult = await atvAdapter.enviarComprobante(emissionResult.clave);

// 6. Marcar como enviada
await invoiceStorage.markAsSent(emissionResult.consecutivo);
```

9.2 Manejo de Errores Estándar

```
try {
  // Operación
  const result = await operation();
  res.status(200).json({ success: true, data: result });
} catch (error) {
  logger.error('Descripción del error:', error);

  // Determinar código de estado apropiado
  const statusCode = error.statusCode || 500;

  res.status(statusCode).json({
    success: false,
    error: error.message,
    timestamp: new Date().toISOString(),
    // Incluir detalles solo en desarrollo
    details: config.isDevelopment() ? error.stack : undefined
  });
}
```

9.3 Validación con Múltiples Niveles

```
// 1. Validación estructural
const structuralValidation = FacturaValidator.validateFactura(data);
if (!structuralValidation.valid) {
  return { errors: structuralValidation.errors };
}

// 2. Validación de lógica de negocio
const businessValidation = FacturaValidator.validateBusinessLogic(data);
if (!businessValidation.valid) {
  return { errors: businessValidation.errors };
}

// 3. Validación externa (ATV)
const atvValidation = await atvAdapter.validarComprobante(clave);
return { valid: atvValidation.valid, details: atvValidation };
```

---

10. Mejores Prácticas

10.1 Logging

- Usar niveles apropiados (`info` , `warn` , `error` , `debug`)
- Incluir contexto relevante en metadatos
- No loggear información sensible
- Usar logging estructurado para facilitar análisis

10.2 Validación

- Validar entrada lo más pronto posible
- Proporcionar mensajes de error claros y específicos

- Usar validación en capas (estructural + negocio)
- Mantener consistencia entre validadores

10.3 Manejo de Errores

- Usar try-catch apropiadamente
- Propagar errores con contexto
- Retornar códigos de estado HTTP correctos
- Loggear errores para debugging

10.4 Configuración

- Usar variables de entorno para configuración sensible
- Proporcionar valores por defecto apropiados
- Documentar todas las opciones de configuración
- Validar configuración al inicio

Esta documentación cubre todos los métodos principales del proyecto Hacienda API, proporcionando información detallada sobre su propósito, uso, parámetros y funcionalidad. La documentación está organizada por módulos para facilitar la navegación y referencia durante el desarrollo y mantenimiento.