✅ **1. Estructura general obligatoria (Factura Electrónica 4.4)**

Según el esquema oficial, los nodos **mínimos requeridos** son:

➤ **Nivel raíz**

FacturaElectronica

🟦 ➤ **Campos obligatorios dentro del comprobante:**

✅ 1️⃣ **Clave**

- Tipo: string (50 caracteres exactos)

✅ 2️⃣ **Código Actividad**

- CodigoActividad

✅ 3️⃣ **Número consecutivo**

- NumeroConsecutivo

✅ 4️⃣ **Fecha de emisión**

- FechaEmision

✅ 5️⃣ **Emisor (completo)**

Campos obligatorios dentro de Emisor:

- Nombre
- Identificacion (Tipo + Número)
- Correo electrónico (Email)
- Ubicación (Provincia, Cantón, Distrito, Barrio opcional, OtrasSenas)
- Teléfono (opcional, pero recomendable)

✅ 6️⃣ **Receptor (si aplica)**

Para factura “normal” (cliente registrado):

- Nombre
- Identificacion (Tipo + Número)

Si es consumidor final (< ₡4.000), se puede omitir, pero depende del uso.

✅ 7️⃣ **Condición de venta (CondicionVenta)**

Ejemplos válidos:

- 01 = Contado

- 02 = Crédito

✅ 8️⃣ **Medio de pago (MedioPago)**

Por lo menos uno:

- 01 = Efectivo
- 02 = Tarjeta
- 03 = Transferencia
- 04 = Cheque
- 05 = Otros

✅ 9️⃣ **Detalle del servicio**

Debe existir al menos **una (1) línea**:

Campos obligatorios en LineaDetalle:

- Número de línea (NumeroLinea)
- Cantidad (Cantidad)
- Unidad de medida (UnidadMedida)
- Detalle (descripción)
- Precio unitario (PrecioUnitario)
- Subtotal (SubTotal)
- Base imponible (BaseImponible)
- Impuesto (solo si aplica)
- Monto total de la línea (MontoTotalLinea)

✅ 🔟 **Resumen de la factura (ResumenFactura)**

Obligatorio:

- CodigoMoneda (ejemplo: CRC, USD)
- TipoCambio (solo si no es CRC)
- TotalServGravados, TotalServExentos (según corresponda)
- TotalVenta
- TotalImpuesto (si existe)
- TotalComprobante

✅ 1️⃣ 1️⃣ **Firma digital (ds:Signature)**

Siempre obligatoria al enviar a Hacienda.

.env.example → variables de entorno con credenciales falsas.

send-sandbox.js → script CommonJS que:

1. Carga el JSON de la factura que ya tenemos.
2. Genera XML sin firmar (usando el SDK tal como lo veníamos usando).
3. (Opción) Firma con un .p12 si lo tienes (si no, se salta la firma).
4. Envía al sandbox usando el cliente del SDK si existe la función; si no, usa axios hacia la URL de sandbox configurada.
5. Muestra la respuesta y errores en consola.

```
// send-sandbox.js

require('dotenv').config();

const fs = require('fs');

const path = require('path');

const axios = require('axios');


const { FacturaElectronica } = require('@facturacr/atv-sdk'); // usado antes en tu flujo


// --- Config ---

const SANDBOX_URL = process.env.SANDBOX_URL ||
'https://sandbox.facturacr.example/api/v1/recepcion';

const USE_SDK_CLIENT = process.env.USE_SDK_CLIENT !== 'false';


// Carga tu JSON de factura (el que definimos anteriormente)

const facturaData = require('./factura-base.json'); // crea este JSON con el contenido que ya
tienes
```

```
async function main() {
  try {
    // 1) Instanciar factura con el SDK
    const factura = new FacturaElectronica(facturaData);

    // 2) Obtener XML sin firmar (método que usaste antes)
    let xmlSinFirma;
    if (typeof factura.getXMLSinFirma === 'function') {
      xmlSinFirma = factura.getXMLSinFirma();
      console.log('XML sin firma generado (long):', xmlSinFirma.length);
    } else if (typeof factura.toXML === 'function') {
      // alternativa de nombre de método
      xmlSinFirma = factura.toXML({ signed: false });
      console.log('XML sin firma (toXML) generado');
    } else {
      console.warn('El SDK no expone getXMLSinFirma() ni toXML(). Ajusta según la API del SDK v2.0.11-alpha.');
      xmlSinFirma = null;
    }

    // 3) (Opcional) Firmar con P12 si lo tienes
    let xmlFirmado = xmlSinFirma;
    if (process.env.P12_PATH && fs.existsSync(process.env.P12_PATH)) {
      try {
        const p12Buffer = fs.readFileSync(path.resolve(process.env.P12_PATH));
        const p12Password = process.env.P12_PASSWORD || '';

        // Intento de firma via SDK si existe el método signWithP12 o similar
        if (typeof factura.signWithP12 === 'function') {
```

```
      xmlFirmado = factura.signWithP12(p12Buffer, p12Password);
      console.log('Factura firmada con P12 usando SDK.');
    } else if (typeof factura.sign === 'function') {
      // algunos SDKs usan sign(buffer, password)
      xmlFirmado = factura.sign(p12Buffer, p12Password);
      console.log('Factura firmada (método sign) con P12 usando SDK.');
    } else {
      console.warn('No se detectó un método de firma en el SDK. Puedes firmar externamente y colocar el XML firmado aquí.');
    }
   } catch (err) {
    console.warn('Error al intentar firmar con P12 (continuando sin firma):', err.message);
   }
  } else {
   console.log('No se encontró P12 en P12_PATH — se enviará (temporalmente) sin firma si el sandbox lo acepta.');
  }

  // 4) Envío al sandbox
  if (USE_SDK_CLIENT) {
   // --- Intento usando cliente del SDK (si existe) ---
   try {
    // algunos SDKs exportan un cliente para enviar, por ejemplo:
    // const { AtvClient } = require('@facturacr/atv-sdk');
    // const client = new AtvClient({ apiKey: process.env.SANDBOX_API_KEY, sandbox: true });
    // const resp = await client.sendFactura(xmlFirmado || xmlSinFirma);

    // Dado que las firmas de métodos varían entre versiones, hacemos un *mejor esfuerzo*:
    if (typeof factura.sendToSandbox === 'function') {
      const resp = await factura.sendToSandbox({
```

```
        apiKey: process.env.SANDBOX_API_KEY,
        sandbox: true
       });
       console.log('Respuesta SDK sendToSandbox:', resp);
      } else if (typeof factura.send === 'function') {
       const resp = await factura.send({
        environment: 'sandbox',
        apiKey: process.env.SANDBOX_API_KEY
       });
       console.log('Respuesta SDK send:', resp);
      } else {
       throw new Error('No se detectó API de envío en el objeto factura. Procederé con fallback HTTP (axios).');
      }
     } catch (err) {
      console.warn('Envio via SDK falló o no está disponible: ', err.message);
      console.log('Intentando fallback HTTP (axios) al SANDBOX_URL ...');
      await sendWithAxios(xmlFirmado || xmlSinFirma);
     }
    } else {
     // Directo por HTTP
     await sendWithAxios(xmlFirmado || xmlSinFirma);
    }

  } catch (err) {
    console.error('Error general:', err);
    process.exit(1);
  }
}
```

```
async function sendWithAxios(xmlPayload) {
  if (!xmlPayload) {
    throw new Error('No hay XML para enviar. Generá o firma el XML antes de intentar enviar.');
  }

  // En el sandbox puede esperarse JSON u otro wrapper; muchos endpoints aceptan XML en el body.
  // Ajustá headers y body según la documentación oficial / SDK.
  const headers = {
    'Content-Type': 'application/xml',
    'x-api-key': process.env.SANDBOX_API_KEY || '',
    // si el sandbox pide Authorization Bearer, ajustá aquí:
    // Authorization: `Bearer ${process.env.SANDBOX_CLIENT_SECRET}`
  };

  try {
    const resp = await axios.post(SANDBOX_URL, xmlPayload, { headers, timeout: 20000 });
    console.log('Respuesta HTTP sandbox:', resp.status, resp.data);
  } catch (err) {
    if (err.response) {
      console.error('Error enviado al sandbox. Status:', err.response.status, 'Data:', err.response.data);
    } else {
      console.error('Error de conexión o timeout:', err.message);
    }
  }
}

main();
```

CAMBIAR DESPUES:

SANDBOX_URL: Usé https://sandbox.facturacr.example/api/v1/recepcion como placeholder. Cuando obtengas la URL oficial del sandbox la colocás en .env.

Credenciales falsas: reemplazá SANDBOX_API_KEY, SANDBOX_CLIENT_ID, SANDBOX_CLIENT_SECRET por las que te entregue Hacienda / proveedor.

Firma: Hacienda exige factura firmada (p12) en muchos flujos; en sandbox algunas veces aceptan pruebas sin firma, otras requieren firma — chequeá el comportamiento del sandbox. El script intenta firmar si existe P12_PATH.

Métodos del SDK: En la versión ^2.0.11-alpha los nombres exactos de métodos pueden cambiar (getXMLSinFirma, toXML, signWithP12, send, sendToSandbox, etc.). Si alguno no existe en tu SDK, inspeccioná el paquete (node_modules/@facturacr/atv-sdk) o revisá su README y ajustá las llamadas en el script. El script ya incluye manejo y fallback a axios.

Clave y NumeroConsecutivo: en tu JSON de ejemplo están con valores dummy. Antes de emitir facturas reales necesitás generar la Clave conforme al algoritmo oficial y usar NumeroConsecutivo correcto para tu emisor en el ambiente (sandbox/producción).

Validación XSD: puedes (y deberías) validar el XML contra el XSD v4.4 antes de enviarlo. Muchos SDKs lo hacen internamente; si no, validá con una librería XML+XSD