TESTING.md — Cómo ejecutar las pruebas y usar las herramientas de validación

Esta guía cubre cómo ejecutar la suite de tests (Jest + Supertest), cómo usar Postman/Newman para las colecciones (si existen) y cómo usar los scripts PowerShell incluidos (si están disponibles). Incluye resolución de problemas comunes que pueden aparecer en este proyecto.

1) Ejecutar la suite de tests (Jest)

Desde la raíz del proyecto:

```
npm install
npm test
```

Comandos útiles:

- `npm test` — Ejecuta Jest una vez (usa `--runInBand` por defecto en el `package.json`).
- `npm run test:watch` — Ejecuta Jest en modo watch (re-ejecuta al guardar).
- `npm run test:coverage` — Ejecuta tests y genera reporte de cobertura en `coverage/`.

Si ves errores de configuración de Jest (por ejemplo watchPlugins o moduleNameMapper), abre `jest.config.js` y verifica las claves: `moduleNameMapper` es la correcta (no `moduleNameMapping`). Si hay `watchPlugins` que no estén instalados puedes quitar esas líneas o instalar el paquete.

Errores comunes y soluciones

- "Unknown option moduleNameMapping" → en `jest.config.js` renombrar `moduleNameMapping` a `moduleNameMapper`.
- "watch plugin cannot be found" → quitar o instalar el plugin indicado (por ejemplo `jest-watch-typeahead`).
- Fallos por dependencias nativas: revisa `tests/setup.js` y mocks (`__mocks__`) que puedan necesitar adaptaciones.

2) Ejecutar un solo test

Puedes ejecutar un archivo de test concreto:

```
npx jest tests/unit/invoiceStorage.test.js --runInBand
```

O ejecutar por nombre de test:

```
npx jest -t "InvoiceStorage Unit Tests" --runInBand
```

3) Mocking y tests de integración

- Unit tests deben usar mocks para acceso a FS y SDKs externos. Este repo usa `jest.mock('fs-extra')` en pruebas unitarias. Revisa `tests/__mocks__` si existe.
- Tests de integración (Supertest) arrancan el servidor y hacen requests HTTP. Asegúrate de que `process.env.NODE_ENV='test'` y que `SIMULATE_IF_NO_KEYS=true` si no quieres que el adaptador

ATV intente modo REAL.

4) Postman / Newman (colecciones)

Si en el proyecto existen colecciones Postman (por ejemplo `docs/postman/*.json` o `postman/*.postman_collection.json`) puedes ejecutarlas con Newman:

Instala Newman globalmente (o usar npx):

```
npm install -g newman
# o
npx newman run docs/postman/EnglishInvoices.postman_collection.json -e docs/postman/env-dev.json
```

Opciones útiles:

- `-e <env>` para usar environment file.
- `--reporters cli,html` para generar reporte HTML.

Si no tienes Newman instalado, puedes ejecutar la colección desde la app Postman.

5) Scripts PowerShell incluidos

Si el repo contiene scripts de prueba en `scripts/` o `tools/`, por ejemplo `Test-EnglishInvoiceAPI.ps1` o `Quick-Test-API.ps1`, puedes ejecutarlos desde PowerShell:

```
# Desde la raíz del repo
.\scripts\Test-EnglishInvoiceAPI.ps1
```

Revisa el contenido de los scripts y las variables que usan (pueden requerir `.env` o parámetros como URL del servidor).

6) Cómo preparar el entorno para tests que tocan ATV o filesystem

- Para evitar tocar el filesystem real en tests, usa `jest.mock('fs-extra')` o la biblioteca `memfs`.
- Para evitar llamadas reales a ATV, asegúrate que `SIMULATE_IF_NO_KEYS=true` o que en `config` las rutas de llaves no estén configuradas en CI.

7) Añadir cobertura y subir a CI

- `npm run test:coverage` genera `coverage/lcov` y `coverage/overview`.
- En GitHub Actions puedes usar `actions/setup-node`, luego `npm ci` y `npm test`.

Ejemplo mínimo de job en GitHub Actions (archivo `.github/workflows/tests.yml`):

```
name: Tests
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
```

```
      - uses: actions/setup-node@v4
        with:
          node-version: '18'
      - run: npm ci
      - run: npm test
      - run: npm run test:coverage
        continue-on-error: true
```

8) Reporte de errores en tests (qué incluir)

Si un test falla y quieres reportarlo, incluye:

- Salida completa de `npm test`.
- El archivo de test que falla y la línea del error.
- Cualquier mock o config relevante (`jest.config.js`, `tests/setup.js`).
- Versión de Node (`node -v`) y OS.

9) Checklist antes de ejecutar integración con Hacienda (modo REAL)

- .env con rutas `ATV_KEY_PATH`, `ATV_CERT_PATH`, `ATV_CLIENT_ID`, `ATV_PIN`.
- `SIMULATE_IF_NO_KEYS=false` y `NODE_ENV=test` o `production` según corresponda.
- Ejecutar `node -e "const a=require('./src/services/atvAdapter'); a.init().then(()=>console.log(a.getStatus())).catch(e=>console.error(e));"` para comprobar inicialización del SDK.

Si quieres, puedo:

- Añadir ejemplos de colecciones Postman (exportadas) o exportar la colección que hemos usado en la sesión.
- Crear el `workflow` de GitHub Actions y añadirlo al repo.

Fin de `TESTING.md`.