Fatoora Partner Signer — Integration Guide

Audience: ISVs and platforms subscribed to a **Fatoora Partner Signer** plan (`partner-signer`, `partner-signer-pro`) who want their own end clients to sign TEIF invoices locally, using a PKCS#11 / USB smart card (TunTrust certificate), through the Fatoora-issued **FatooraSigner** desktop agent — without ever handling the token PIN or transmitting it to your own servers.

Not covered here: the separate **Fatoora Partner API** plan (`partner-starter`, `partner-business`, `partner-max`), a stateless server-to-server REST API for submitting, signing and filing invoices *on behalf of* client organizations — a fundamentally different integration model. See Partner API vs. Partner Signer below, and the companion guide `PARTNER_API_GUIDE.md`.

Two hosts, not one: unlike the Partner API, this integration talks to **two different servers**:

Host	Role
`https://business.fatoora.tn`	Fatoora Cloud — end-client login, session validation, domain whitelist & client pre-registration management
`http://127.0.0.1:38443`	**FatooraSigner**, the local agent installed on the end client's own machine — actual PKCS#11 signing, session lifecycle, optional direct TTN filing

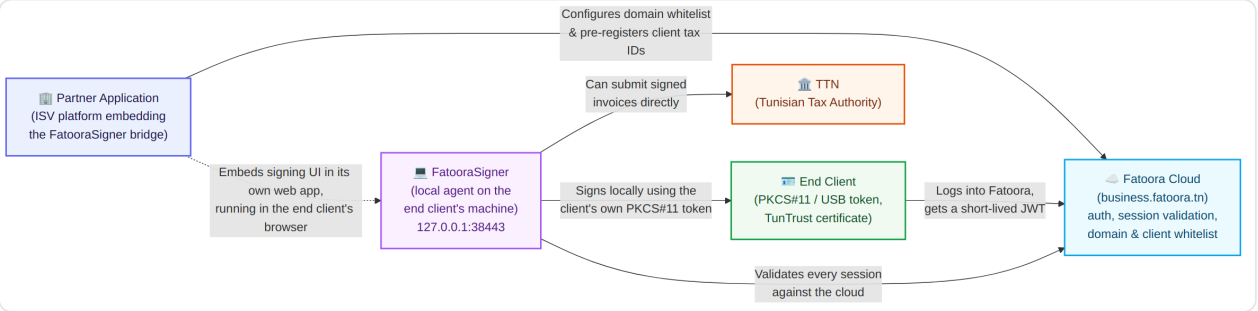
Code samples below use `{{business_url}}` for the first and `{{signer_url}}` for the second — see §3 Two-host architecture.

Table of contents

---

1. How it works

A Partner Signer integration involves **four parties**, and — unlike the Partner API's OTP-gated consent flow — there is **no client-facing approval step**: the partner directly pre-registers which client tax IDs may use the bridge, based on its own existing relationship with that client outside Fatoora.

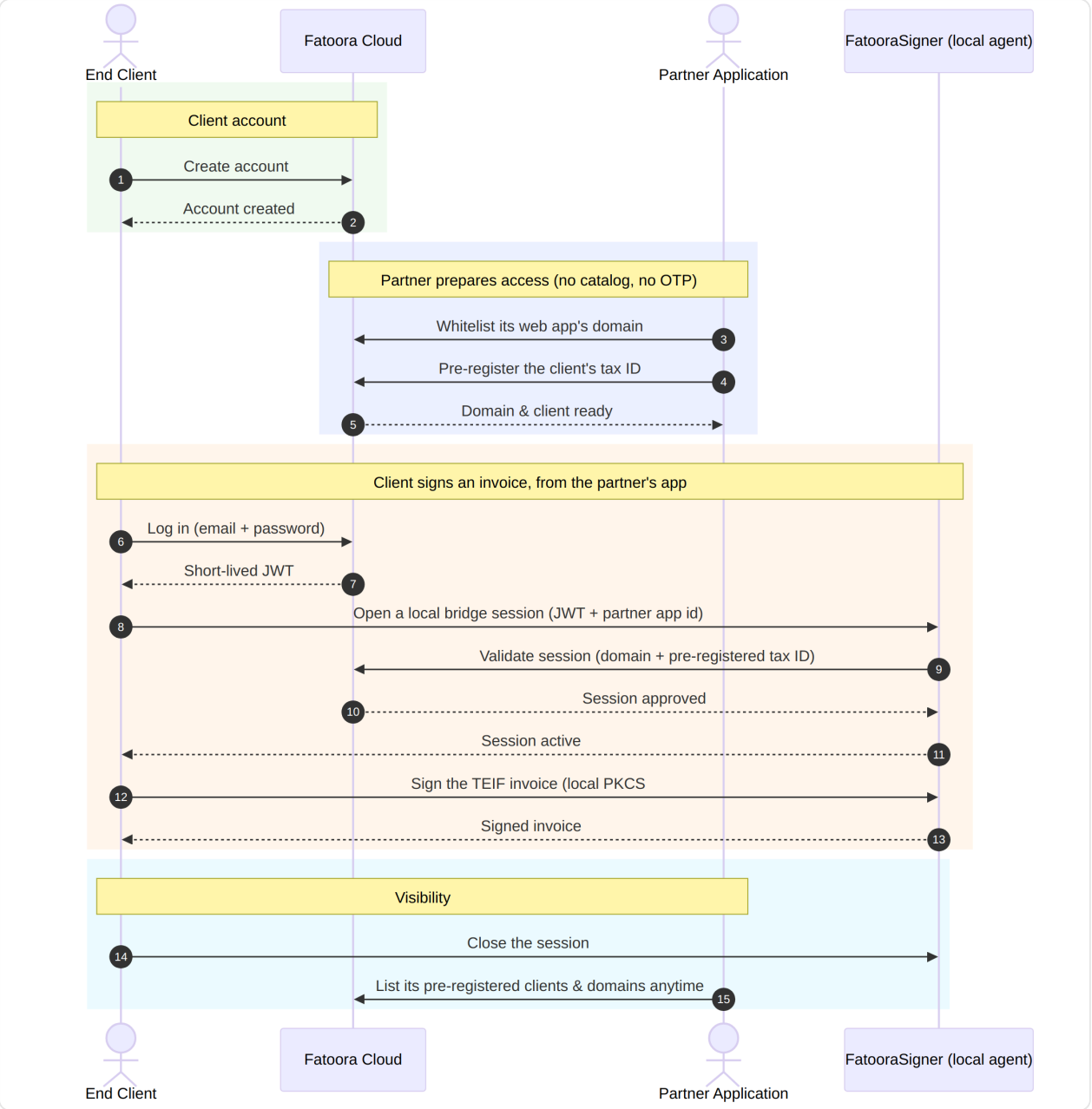


- **You (the Partner Application)** — an ISV embedding a signing UI into your own web app. You never touch the client's PIN, certificate, or private key.
- **Fatoora Cloud** (`business.fatoora.tn`) — issues end-client JWTs, and validates every signing session against your domain whitelist and pre-registered client list.
- **FatooraSigner** — the local agent your end client installs on their own machine. It talks to the PKCS#11 token directly and to Fatoora Cloud for session validation; it never talks to your servers.
- **The End Client** — the person actually signing, using their own TunTrust smart card / USB token.
- **TTN** (Tunisia Trade Net) — the tax authority. FatooraSigner can file signed invoices with TTN directly from the local agent (see §6) — this is a capability the Partner API does **not** expose to partner tokens.

Three principles fall out of this model:

- **No OTP consent step.** You pre-register a client's tax ID yourself (`POST /api/partner/signer/clients`) — Fatoora does not ask that client to approve you. You are responsible for having your own agreement with each client before pre-registering them.
- **The PIN never leaves the client's machine.** It's sent directly from the browser to `127.0.0.1:38443`, encrypted in-memory for the lifetime of the session, and never transmitted to Fatoora Cloud or to your servers.
- **Domain whitelisting protects the cloud-side session check, not the local agent's CORS policy** — these are two separate layers, detailed in §3.

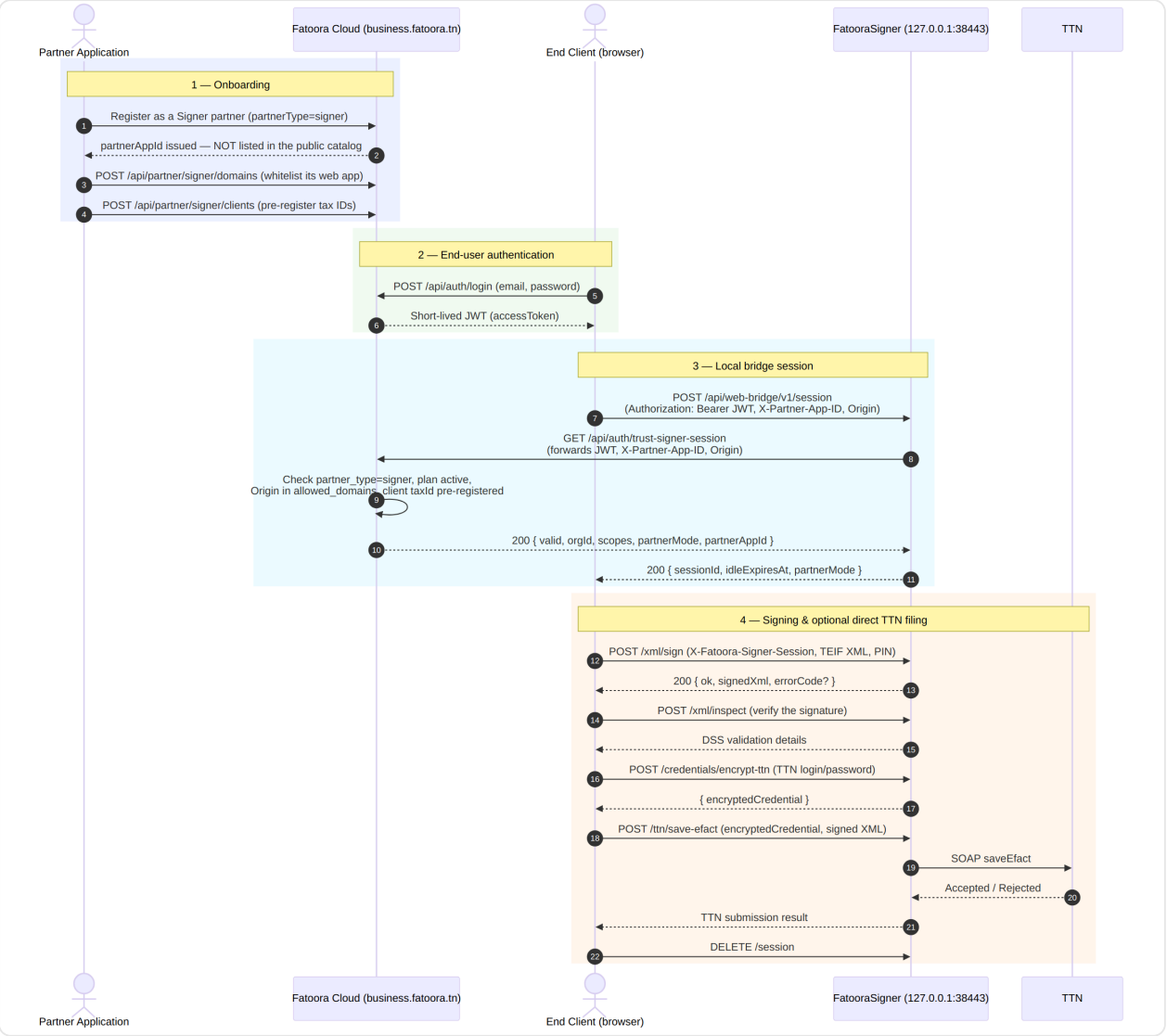
Core interaction sequence



Step	Actor	Action
1	Client	Creates a Fatoora account
2	Partner → Fatoora	Whitelists its web app's domain (once)
3	Partner → Fatoora	Pre-registers the client's tax ID (no client action needed)
4	Client → Fatoora	Logs in, gets a short-lived JWT
5	Client → FatooraSigner	Opens a local bridge session (JWT + partner app id)
6	FatooraSigner → Fatoora	Validates the session (domain + pre-registered tax ID)

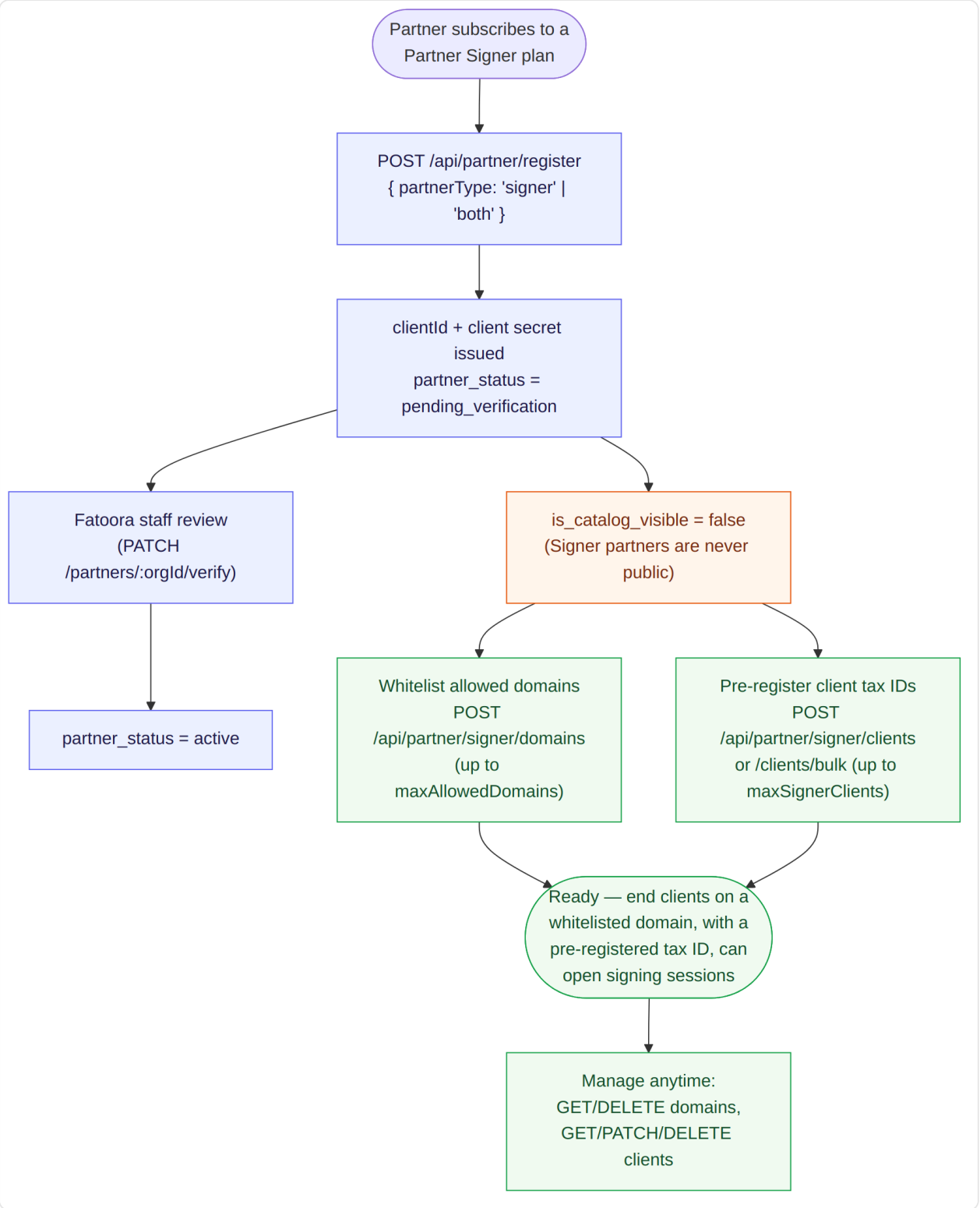
Step	Actor	Action
7	Client → FatooraSigner	Signs the TEIF invoice with the local PKCS#11 PIN
8	Partner → Fatoora	Can list its pre-registered clients & domains anytime

The full technical workflow, end to end



Typical use case: an accounting software vendor embeds FatooraSigner into their desktop/web product so their clients — who already have TunTrust smart cards — can sign invoices generated by the vendor's own software, without the vendor ever building its own PKCS#11 integration or handling signing credentials.

2. Onboarding



Step 1 — Register

```
POST /api/partner/register
{ "partnerType": "signer", "companyName": "...", "contactEmail": "..." }
```

Provisions a `clientId` + client secret (same credential mechanism as the Partner API, though the web-bridge flow itself doesn't use HMAC — these credentials matter only if you also call other partner REST endpoints). Your account starts `pending_verification`.

> **Signer partners are never listed in the public partner catalog** (`is_catalog_visible = false` is forced for `partnerType: "signer"`) — unlike API partners, clients don't "discover" you through Fatoora; they already use your product.

If your product needs both a REST invoice-submission integration *and* local signing, register with `partnerType: "both"` — this unlocks both capability sets (see §12).

Step 2 — Whitelist your domain(s)

```
GET    /api/partner/signer/domains
POST   /api/partner/signer/domains         { "domain": "yourapp.example.com" }
DELETE /api/partner/signer/domains/:domain
```

`domain` must be a bare hostname (no scheme) matching `^([a-zA-Z0-9]([a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?\.)+[a-zA-Z]{2,}$` — 3 to 253 characters, lowercased on save. This is the `Origin`/`Referer` host that Fatoora Cloud checks when your end client's browser opens a signing session; subdomains of a whitelisted domain also match (`app.yourapp.example.com` matches `yourapp.example.com`).

Capped at `maxAllowedDomains` for your plan (1 for `partner-signer`, 5 for `partner-signer-pro`) — `403 DOMAIN_LIMIT_REACHED` beyond that, confirmed live with the exact message `"Your plan allows a maximum of {N} whitelisted domain(s). Upgrade to add more."`. Re-adding an already-whitelisted domain returns `409 DOMAIN_ALREADY_EXISTS`.

> **This entire flow was broken until this guide's verification pass, and has now been fixed.** `POST`/`DELETE` on this endpoint previously crashed with `500 INTERNAL_ERROR` on every call — server logs showed a raw Postgres `malformed array literal` error, because the write path didn't serialize the domain list into valid Postgres array syntax. Fixed in `fatooraBusiness/src/routes/partner-signer.routes.ts`; verified live afterward — add, duplicate-detect, limit-enforce, and delete (including deleting your last domain back to an empty list) all now work as documented above.

Step 3 — Pre-register your clients' tax IDs

```
GET    /api/partner/signer/clients?search=&active=true
POST   /api/partner/signer/clients          { "taxId": "1234567ABC000", "clientName": "Client
SARL" }
POST   /api/partner/signer/clients/bulk     { "clients": [{ "taxId": "...", "clientName": "..." },
...] }
PATCH  /api/partner/signer/clients/:id      { "isActive": false }
DELETE /api/partner/signer/clients/:id
```

Capped at `maxSignerClients` for your plan (50 for `partner-signer`, 300 for `partner-signer-pro`) — `403 CLIENT_LIMIT_REACHED` beyond that (bulk import checks the limit up front and rejects the whole batch if it would be exceeded). `POST` on an existing `taxId` reactivates it (`isActive: true`) rather than erroring — this doesn't count against the limit for genuinely-new tax IDs. This flow was unaffected by the domain-whitelist bug above (pre-registered clients are individual table rows, not an array column) and is confirmed live end-to-end: create, search, bulk-import, activate/deactivate, delete.

> `createdAt` on a client record is a raw timestamp, not strict ISO-8601 — confirmed live as `"2026-07-11 20:54:12.487115"` (space-separated, microsecond precision, no `T`/`Z`), the same format quirk as the Partner API's `authorizedAt` field.

Once a client's tax ID is pre-registered and active, and their browser is on a whitelisted domain, they can open signing sessions immediately — no further action from either side.

---

3. Two-host architecture: cloud vs. local agent

	Fatoora Cloud (`business.fatoora.tn`)	FatooraSigner (`127.0.0.1:38443`)
Runs where	Fatoora's servers	The end client's own machine (desktop-installed agent)
Protocol	HTTPS	**Plain HTTP** — no TLS by default
You manage	Domain whitelist, client pre-registration, your credentials	Nothing — it's the client's local install
End client talks to it for	Login (`/api/auth/login`)	Opening a session, signing, inspecting, optional TTN filing

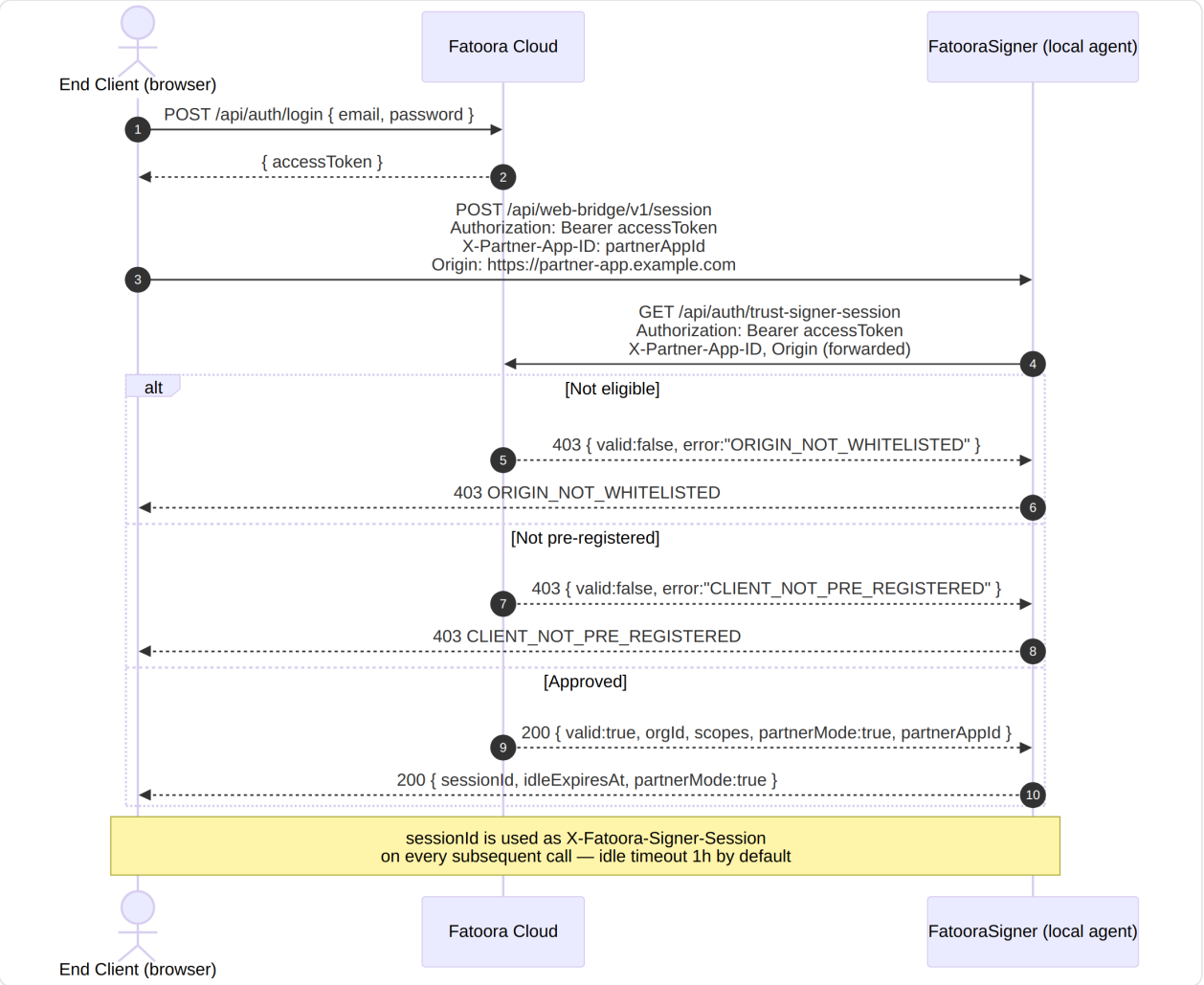
	Fatoora Cloud (`business.fatoora.tn`)	FatooraSigner (`127.0.0.1:38443`)
Enforces	Your `allowed_domains` + `partner_signer_clients` allowlist (via `Origin` / `Referer` check at session-open time)	A **separate, broader** CORS policy (`localhost`, `127.0.0.1`, `*.fatoora.tn`, plus any origins your agent config explicitly allows)

Important nuance: the domain whitelist you configure in §2 governs the **cloud-side session-authorization check** — it does not change the local agent's own CORS policy. For your own domain's browser JavaScript to successfully call `127.0.0.1:38443` at all (before the cloud check even runs), that origin must already be allowed by the agent's CORS configuration. By default the agent allows `localhost` / `127.0.0.1` and any `*.fatoora.tn` subdomain; a genuinely third-party domain needs to be added to the agent's `app.http.cors.additional-origin-patterns` — talk to Fatoora support about how this is provisioned for your distribution of the agent.

The agent binds to all network interfaces by default (not restricted to `127.0.0.1` at the OS/socket level) — the "local-only" guarantee comes from it being a desktop-installed application behind the client's own machine/firewall, plus the CORS policy above, not a hard network restriction. Don't advertise it as network-isolated in your own documentation.

> ⚠️ **A third, undocumented CORS gate — confirmed live, currently blocks legitimate partner domains.** There are actually **three** layers a session-open call passes through, not two:
>
> 1. Browser → local agent: the agent's own CORS policy (above).
> 2. Local agent → Fatoora Cloud (`GET /api/auth/trust-signer-session`): **fatooraBusiness's global Express `cors()` middleware**, driven by the server-side `CORS_ORIGINS` env var plus a handful of hardcoded `*.fatoora.tn` domains — completely separate from, and unaware of, your self-service `allowed_domains` whitelist from §2.
> 3. Cloud business logic: your `allowed_domains` check, returning the documented `403 ORIGIN_NOT_WHITELISTED`.
>
> Tested live: calling `trust-signer-session` with `X-Partner-App-ID` set and `Origin: https://a-real-partner-domain.com` — a domain **not** in the server's `CORS_ORIGINS` — returns `500 {"message":"Origin https://a-real-partner-domain.com not allowed by CORS"}`, rejected by layer 2 **before layer 3's whitelist check ever runs**. In other words: whitelisting your domain via `POST /api/partner/signer/domains` is necessary but **not sufficient** — Fatoora staff also need to add it to the server's `CORS_ORIGINS` configuration, which partners cannot self-service. If your session-open calls fail with a raw `500` mentioning "not allowed by CORS" rather than the documented `403 ORIGIN_NOT_WHITELISTED`, this is why — contact Fatoora support to have your domain added server-side, in addition to your own whitelist entry.

4. Authentication & session flow



Step A — End-client login (cloud)

```
POST https://business.fatoora.tn/api/auth/login
Content-Type: application/json

{ "email": "client@example.com", "password": "..." }

→ 200 { "accessToken": "eyJhbGc..." }
```

This is a standard Fatoora user JWT (short-lived) — not the OAuth2 client-credentials token used by the Partner API.

Step B — Open a local bridge session (agent, validated by cloud)

```
POST http://127.0.0.1:38443/api/web-bridge/v1/session
Authorization: Bearer <accessToken>
X-Partner-App-ID: <your partnerAppId>
Origin: https://yourapp.example.com     # sent automatically by the browser
```

The agent forwards your JWT, `X-Partner-App-ID`, and `Origin` to `GET https://business.fatoora.tn/api/auth/trust-signer-session`, which runs this validation chain — order confirmed live (a paused subscription surfaces `PLAN_NOT_ELIGIBLE` at step 3 even with no domain/client configured, i.e. it's genuinely checked before steps 4–5, not just documented that way):

0. **Before any of this runs**, the request must first clear fatooraBusiness's own global CORS layer — see the callout in §3. A domain rejected there never reaches step 1.
1. Partner app exists (`403 PARTNER_NOT_FOUND` if not)
2. `partner_type ∈ {signer, both}` (`403 NOT_SIGNER_PARTNER` if not)
3. Partner's own subscription is `partner-signer`/`partner-signer-pro`, `active` or `trial` (`403 PLAN_NOT_ELIGIBLE` if not)
4. `Origin`/`Referer` host matches (or is a subdomain of) an entry in your `allowed_domains` — **only enforced if you've configured at least one domain and the header is parseable**; skipped otherwise (`403 ORIGIN_NOT_WHITELISTED` if it fails)
5. The end client's organization tax ID is in your `partner_signer_clients` table with `is_active = true` — **only enforced if the org has a tax ID set** (`403 CLIENT_NOT_PRE_REGISTERED` if it fails)

Success response, from the agent:

```
{
  "sessionId": "sess_xxxxxxxx",
  "idleExpiresAt": "2026-07-11T15:00:00Z",
  "absoluteExpiresAt": null,
  "partnerMode": true,
  "partnerAppId": "partn_xxxxxxxx"
}
```

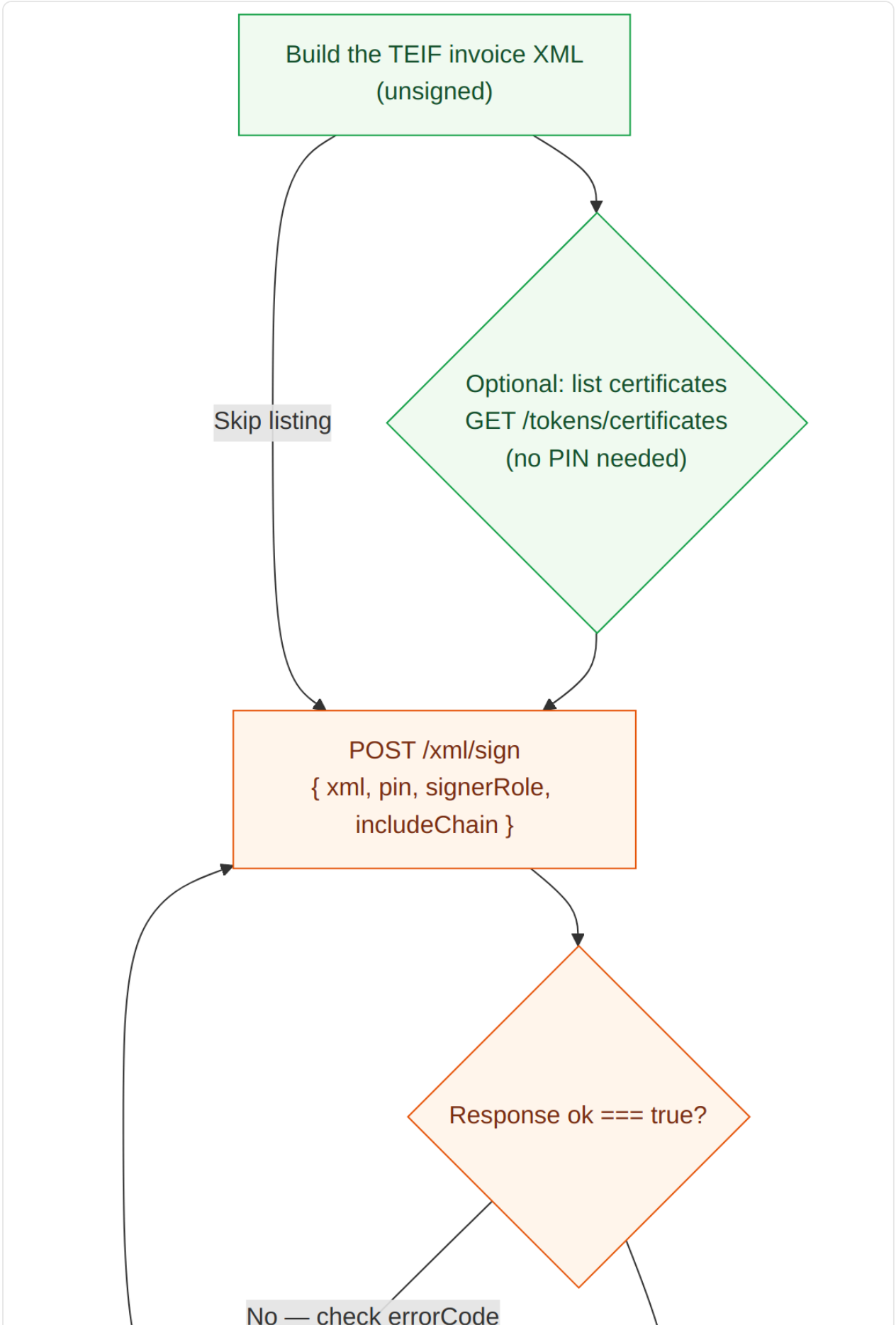
Use `sessionId` as the `X-Fatoora-Signer-Session` header on every subsequent agent call. Idle timeout defaults to **1 hour**; there's no absolute maximum lifetime by default. Close it explicitly when the client is done:

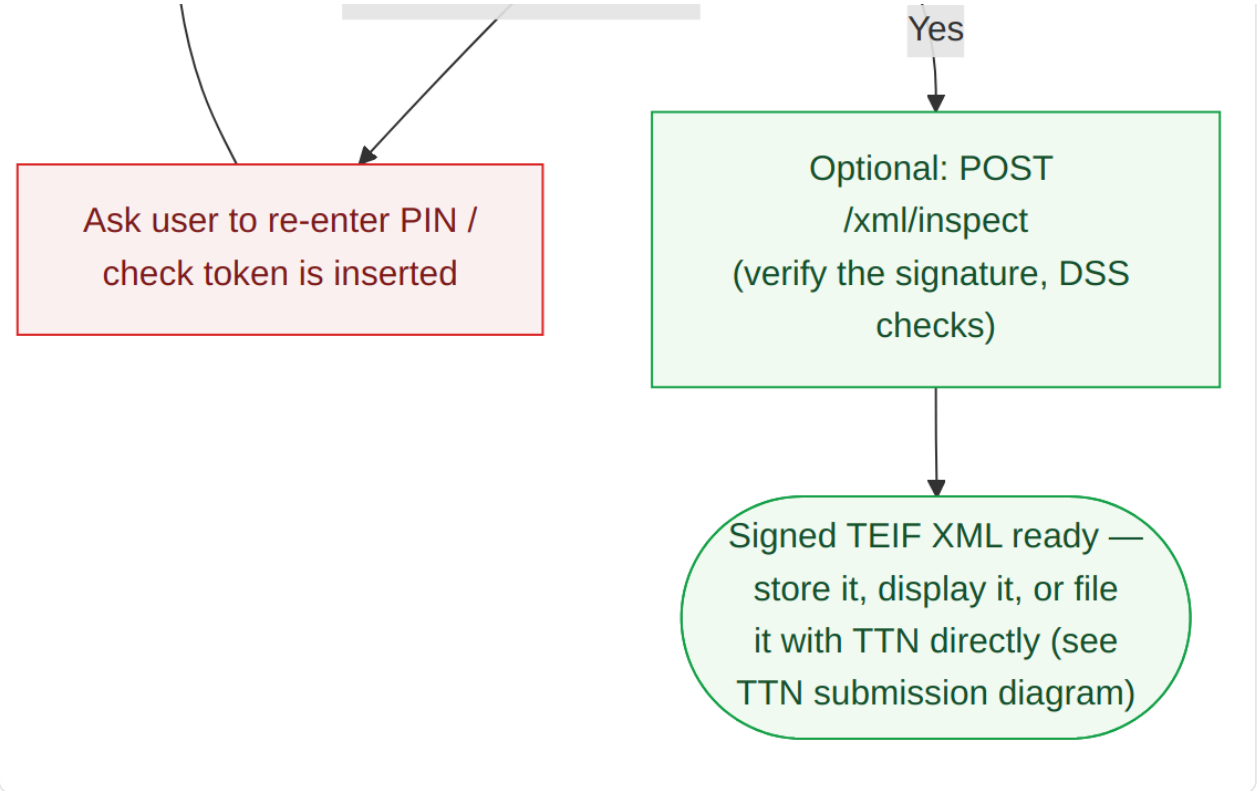
```
DELETE http://127.0.0.1:38443/api/web-bridge/v1/session
X-Fatoora-Signer-Session: <sessionId>

→ 204 No Content
```

---

5. Signing operations


Build the TEIF invoice XML
(unsigned)
Skip listing
Optional: list certificates
GET /tokens/certificates
(no PIN needed)
POST /xml/sign
{ xml, pin, signerRole,
includeChain }
Response ok === true?
No — check errorCode



All of these require `X-Fatoora-Signer-Session` and target the **local agent** (`http://127.0.0.1:38443`).

`GET /api/web-bridge/v1/tokens/certificates`

Lists public certificates on the connected USB token — **no PIN required**. Optional `driverId` query param to target a specific PKCS#11 driver.

`POST /api/web-bridge/v1/tokens/detect-with-certificates`

```
{ "pin": "1234", "driverId": null }
```

Detects the token and reads full certificate details. Requires the PIN (or a previously-obtained `encryptedPin` — see §6).

`POST /api/web-bridge/v1/xml/sign` — the core signing call

```
{
  "xml": "<Invoice>...</Invoice>",
  "pin": "1234",
  "signerRole": "CEO",
  "driverId": "opensc",
  "includeChain": true
}
```

Response — **always `200`, even on failure**; check the `ok` field:

```
{ "mode": "sign", "ok": true, "signedXml": "<Invoice>...(signed)...</Invoice>", "signatureLevel":
        "..." }
```

or, on failure:

```
{ "mode": "sign", "ok": false, "error": "Invalid PIN", "errorCode": "SIGN_FAILED" }
```

`500` is reserved for native PKCS#11 library load failures (`LinkageError` / `ClassNotFoundException`) — not for ordinary signing errors like a wrong PIN.

`POST /api/web-bridge/v1/xml/inspect`

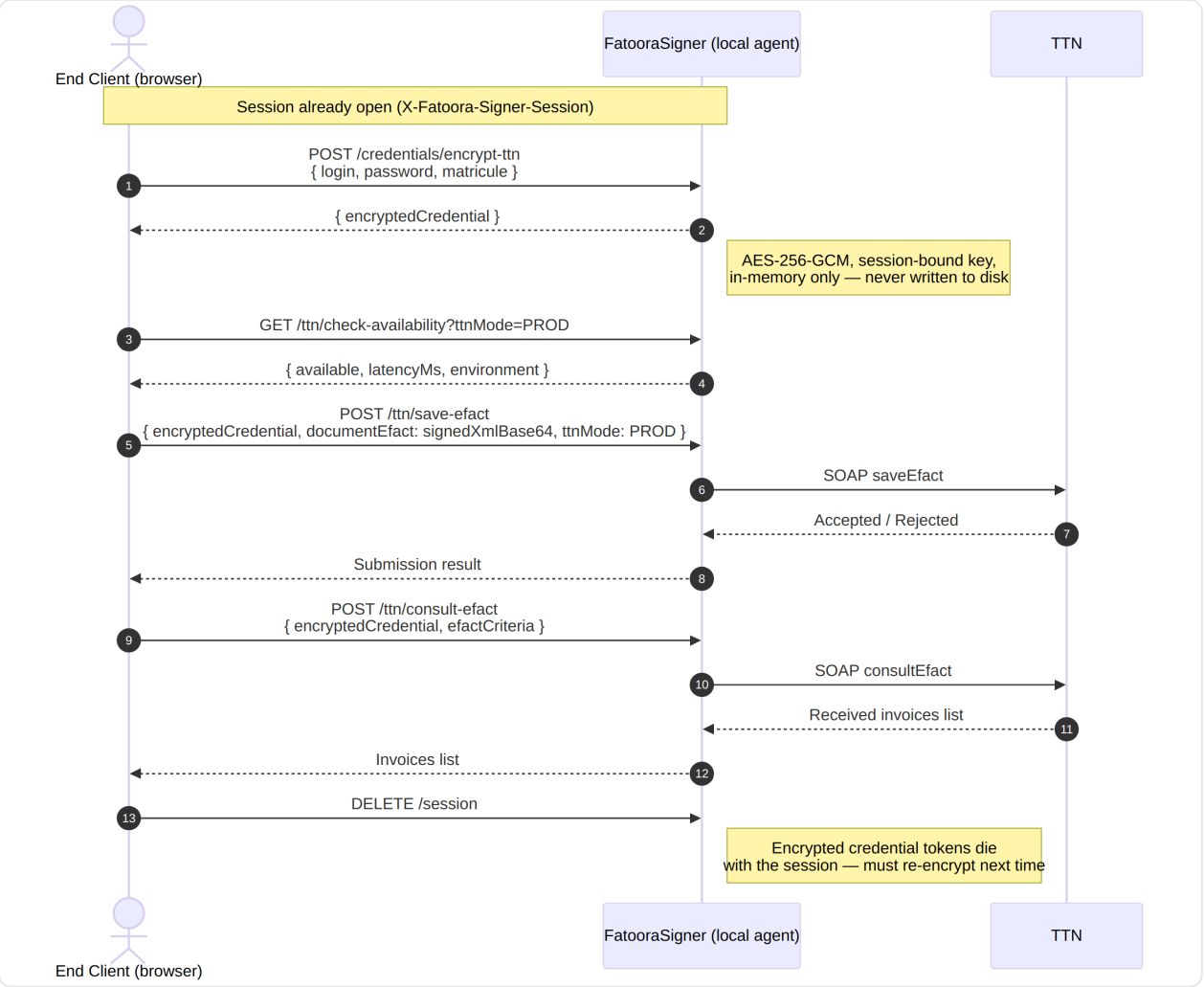
```
{ "xml": "<SignedInvoice>...</SignedInvoice>", "revocationOnline": true }
```

Verifies signatures already present in an XML document and returns DSS validation details — useful for a "verify before you trust it" step, or to check a document you received rather than signed yourself.

---

6. Submitting directly to TTN

Unlike the Partner API (where partner tokens are explicitly blocked from direct TTN submission — see the companion guide's §11), the **local FatooraSigner agent can file signed invoices with TTN directly**, using the end client's own TTN credentials, entered locally and never sent to your servers.



`POST /api/web-bridge/v1/credentials/encrypt-ttn`

```
{ "login": "12345678", "password": "ttn-password", "matricule": "1234567ABC000" }
→ { "encryptedCredential": "AES256GCM:..." }
```

AES-256-GCM, session-bound key, **in-memory only — never written to disk, never sent to Fatoora Cloud or your servers**. The encrypted token dies with the session; re-encrypt on every new session.

`GET /api/web-bridge/v1/ttn/check-availability?ttnMode=PROD`

Reachability + latency check before submitting. `ttnMode` is `TEST` or `PROD`.

`POST /api/web-bridge/v1/ttn/save-efact`

```
{
  "encryptedCredential": "AES256GCM:...",
  "documentEfact": "<base64-encoded signed TEIF XML>",
  "ttnMode": "PROD"
}
```

Submits the signed document to TTN via SOAP (`saveEfact`).

`POST /api/web-bridge/v1/ttn/consult-efact`

```
{ "encryptedCredential": "AES256GCM:...", "efactCriteria": { /* ... */ }, "ttnMode": "PROD" }
```

Pulls received invoices from TTN (`consultEfact`).

> **Zero DB access on these routes.** The agent never persists TTN credentials or invoice content locally beyond the in-memory session — every call re-decrypts the session-bound token to make the SOAP request.

---

7. Domain whitelist & client management

Full CRUD reference (all served from **Fatoora Cloud**, `https://business.fatoora.tn`, bearer-authenticated with your own user/partner session — not the end client's):

Method	Path	Purpose
GET	`/api/partner/signer/domains`	List whitelisted domains
POST	`/api/partner/signer/domains`	Add a domain (`403 DOMAIN_LIMIT_REACHED`, `409 DOMAIN_ALREADY_EXISTS`)
DELETE	`/api/partner/signer/domains/:domain`	Remove a domain (`404 DOMAIN_NOT_FOUND`)
GET	`/api/partner/signer/clients?search=&active=`	List pre-registered clients, filterable
POST	`/api/partner/signer/clients`	Pre-register (or reactivate) one client (`403 CLIENT_LIMIT_REACHED`)

Method	Path	Purpose
POST	`/api/partner/signer/clients/bulk`	Pre-register up to 200 clients at once (`403 CLIENT_LIMIT_REACHED`)
PATCH	`/api/partner/signer/clients/:id`	Toggle `isActive` (`404 CLIENT_NOT_FOUND`)
DELETE	`/api/partner/signer/clients/:id`	Remove a pre-registered client (`404 CLIENT_NOT_FOUND`)

Manage these from **Fatoora dashboard → Partner → Signer → Domains / Clients** as an alternative to the API.

> All eight endpoints above were exercised live end-to-end against a dev instance (create/list/limit/duplicate/delete for both domains and clients, plus bulk import and search) — see §2 for the domain-whitelist bug that was found and fixed during that pass.

---

8. What partner signer apps cannot do

- **No catalog visibility.** Signer partners are never shown in Fatoora's public partner catalog — clients reach you through your own product, not through Fatoora.
- **No client-facing OTP consent.** You pre-register a client's tax ID unilaterally; Fatoora does not notify or ask that client to approve you. This is a trust model, not a technical restriction — you're expected to have your own agreement with each client.
- **No REST invoice submission through this bridge.** The web-bridge signs and (optionally) files XML you already built — it has no endpoint to create/list/manage invoices the way the Partner API does. If you also need that, register with `partnerType: "both"`.
- **The client's PIN and TTN credentials are never accessible to you.** They're entered directly into the browser page talking to `127.0.0.1:38443` and encrypted in-memory for the session's lifetime — your backend never sees them, by design.
- **No network isolation guarantee.** As noted in §3, the agent isn't hard-restricted to `127.0.0.1` at the socket level — don't represent it as network-isolated to your own clients.

---

9. Error handling

Cloud session-validation errors (`GET /api/auth/trust-signer-session`, surfaced through the agent's `POST /session`)

Code	HTTP	Meaning / fix
"Origin <url> not allowed by CORS"	**500**	⚠️ **Not the same as `ORIGIN_NOT_WHITELISTED` below** — confirmed live. This fires at fatooraBusiness's global CORS layer, *before* your domain whitelist is even checked, if your domain isn't also in the server's `CORS_ORIGINS` config. See the callout in §3 — contact Fatoora support.
`PARTNER_NOT_FOUND`	403	Your `partnerAppId` doesn't match a registered partner
`NOT_SIGNER_PARTNER`	403	Your profile's `partner_type` isn't `signer` / `both` — check `PATCH /api/partner/profile`
`PLAN_NOT_ELIGIBLE`	403	Your subscription isn't an active/trial `partner-signer` / `partner-signer-pro` plan — confirmed live (checked *before* the origin/client checks below, so a paused subscription masks those errors)
`ORIGIN_NOT_WHITELISTED`	403	The calling page's `Origin` / `Referer` isn't in your domain whitelist — see §2
`CLIENT_NOT_PRE_REGISTERED`	403	The end client's org tax ID isn't pre-registered (or is inactive) — see §2
"Active Fatoora Trust subscription required"	403	Direct-Trust-mode error (no `X-Partner-App-ID` sent) — not applicable in partner mode; confirmed live
"User context required" / "Organization context required"	401 / 403	Missing/invalid JWT
"Internal error"	500	Unexpected server error

Local agent errors

Code	HTTP	Meaning / fix
"Invalid or expired token"	401	Re-authenticate the end client and open a new session
(forwarded cloud error code)	403	See table above — the agent passes through the cloud's `error` field verbatim
Missing/invalid `X-Fatoora-Signer-Session`	401	Session expired or was never opened — reopen it
"Missing required scope: ..."	403	Only if the agent is configured with `required-scopes` — not the default
`SIGN_FAILED` (and other business `errorCode` values)	**200**	Check the PIN, that the token is inserted, and the PKCS#11 driver — this is not an HTTP-level error
(native library failure)	500	PKCS#11 driver/native library couldn't load — reinstall/reconfigure the agent
(cloud unreachable)	502	The agent couldn't reach `business.fatoora.tn` — check the end client's internet connection

Domain/client management errors (Fatoora Cloud)

Code	HTTP	Meaning
`DOMAIN_ALREADY_EXISTS`	409	Domain already whitelisted
`DOMAIN_LIMIT_REACHED`	403	At your plan's `maxAllowedDomains`
`DOMAIN_NOT_FOUND`	404	No such whitelisted domain
`CLIENT_LIMIT_REACHED`	403	At your plan's `maxSignerClients`
`CLIENT_NOT_FOUND`	404	No such pre-registered client (or not owned by your partner app)
`VALIDATION_ERROR`	400	Malformed request body (Zod validation)
`USER_NOT_PARTNER` / `USER_NO_ORG`	403	Calling user isn't a partner-org member
`INTERNAL_ERROR`	500	Unexpected server error

10. Rate limits and quotas

Limiter	Scope	Limit
`POST /api/auth/login`	per IP	Standard Fatoora auth rate limiting (see companion Partner API guide, §12)
`/api/partner/signer/*` (domain & client management)	per IP	General API limiter — 300 requests / 15 min in production
Local agent (`127.0.0.1:38443/*`)	—	No rate limiting — it's a single-user local process

Plan-level quotas (no signing-volume cap on either plan — both `monthlyDocLimit: null`):

Plan	`maxAllowedDomains`	`maxSignerClients`	`maxServiceApps`	`maxSeats`	Trial
`partner-signer`	1	50	1	3	14 days
`partner-signer-pro`	5	300	3	5	14 days

Session idle timeout defaults to **1 hour** (agent-side, not plan-dependent) — configurable per agent deployment (`WEB_BRIDGE_SESSION_IDLE`), not something you control via the partner API.

11. Code examples

Full flow: login → open session → sign

```
# 1. Authenticate the end client
ACCESS_TOKEN=$(curl -s -X POST "https://business.fatoora.tn/api/auth/login" \
  -H "Content-Type: application/json" \
  -d '{"email":"client@example.com","password":"..."}' | jq -r '.accessToken')

# 2. Open a local bridge session (runs from the client's browser in practice —
#    shown here with curl for clarity; the Origin header matters when called from a browser)
SESSION_ID=$(curl -s -X POST "http://127.0.0.1:38443/api/web-bridge/v1/session" \
  -H "Authorization: Bearer $ACCESS_TOKEN" \
  -H "X-Partner-App-ID: YOUR_PARTNER_APP_ID" | jq -r '.sessionId')

# 3. Sign a TEIF XML document
curl -s -X POST "http://127.0.0.1:38443/api/web-bridge/v1/xml/sign" \
  -H "Content-Type: application/json" \
  -H "X-Fatoora-Signer-Session: $SESSION_ID" \
  -d '{"xml":"<Invoice>...</Invoice>","pin":"1234","signerRole":"CEO","includeChain":true}'

# 4. Close the session
curl -s -X DELETE "http://127.0.0.1:38443/api/web-bridge/v1/session" \
  -H "X-Fatoora-Signer-Session: $SESSION_ID"
```

```
// Runs in the end client's browser
const loginRes = await fetch('https://business.fatoora.tn/api/auth/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({ email, password }),
});
const { accessToken } = await loginRes.json();

const sessionRes = await fetch('http://127.0.0.1:38443/api/web-bridge/v1/session', {
  method: 'POST',
  headers: {
    Authorization: `Bearer ${accessToken}`,
    'X-Partner-App-ID': 'YOUR_PARTNER_APP_ID',
  },
});
const { sessionId } = await sessionRes.json();

const signRes = await fetch('http://127.0.0.1:38443/api/web-bridge/v1/xml/sign', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json', 'X-Fatoora-Signer-Session': sessionId },
  body: JSON.stringify({
    xml: teifXmlString,
    pin: userPin,        // entered by the user, never sent to your servers
    signerRole: 'CEO',
    includeChain: true,
  }),
});
const { ok, signedXml, errorCode } = await signRes.json();
if (!ok) throw new Error(errorCode);
```

Managing domains and clients

```
# Whitelist your domain (once)
curl -X POST "https://business.fatoora.tn/api/partner/signer/domains" \
  -H "Authorization: Bearer YOUR_USER_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"domain":"yourapp.example.com"}'

# Pre-register a client
curl -X POST "https://business.fatoora.tn/api/partner/signer/clients" \
  -H "Authorization: Bearer YOUR_USER_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"taxId":"1234567ABC000","clientName":"Client SARL"}'
```

A ready-to-import Postman collection is available from **Fatoora dashboard → Partner → Credentials → Signer guide → Download Postman collection**. See §14 for the exact same collection embedded in this guide.

---

12. Partner API vs. Partner Signer

	Partner API	**Partner Signer** (this guide)
Plans	`partner-starter/business/max`	`partner-signer`, `partner-signer-pro`
Integration model	Stateless server-to-server REST API (OAuth2 client_credentials)	Local PKCS#11/browser signing bridge, running on the end client's machine
Hosts involved	One (`business.fatoora.tn`)	Two (`business.fatoora.tn` + local agent `127.0.0.1:38443`)
What you submit	Full TEIF invoices, on behalf of clients	Nothing — you broker local signing sessions for pre-registered clients
Client onboarding	OTP-based authorization (client approves you)	Direct pre-registration by tax ID (no client action, no OTP)
Catalog visibility	Yes (once verified)	Never
Direct TTN submission	Blocked for partner tokens	**Allowed**, from the local agent, using the client's own TTN credentials
Domain restriction	None	Required — `allowed_domains` gates the cloud-side session check

If your product needs both, register with `partnerType: "both"` — this unlocks both capability sets on your account.

---

13. Appendix — reference tables

Session descriptor shape

```
{
  "valid": true,
  "sub": "user_xxxx",
  "userId": "user_xxxx",
  "orgId": "org_xxxx",
  "roles": ["owner"],
  "scopes": [],
  "partnerMode": true,
  "partnerAppId": "partn_xxxx"
}
```

`partnerMode` / `partnerAppId` are omitted entirely (not `null`) in direct Fatoora Trust mode (no `X-Partner-App-ID` header sent).

Endpoint summary

Method	Host	Path	Auth
POST	Cloud	`/api/auth/login`	—
GET/POST/DELETE	Cloud	`/api/partner/signer/domains[/:domain]`	Bearer (partner user)
GET/POST/PATCH/DELETE	Cloud	`/api/partner/signer/clients[/:id]`, `/clients/bulk`	Bearer (partner user)
GET	Cloud	`/api/auth/trust-signer-session`	Bearer (end client) — called by the agent, not directly
POST/DELETE	Local agent	`/api/web-bridge/v1/session`	Bearer / `X-Fatoora-Signer-Session`
GET	Local agent	`/api/web-bridge/v1/tokens/certificates`	`X-Fatoora-Signer-Session`

Method	Host	Path	Auth
POST	Local agent	`/api/web-bridge/v1/tokens/detect-with-certificates`	`X-Fatoora-Signer-Session`
POST	Local agent	`/api/web-bridge/v1/xml/sign`	`X-Fatoora-Signer-Session`
POST	Local agent	`/api/web-bridge/v1/xml/inspect`	none
POST	Local agent	`/api/web-bridge/v1/credentials/encrypt-pin`	`X-Fatoora-Signer-Session`
POST	Local agent	`/api/web-bridge/v1/credentials/encrypt-ttn`	`X-Fatoora-Signer-Session`
GET	Local agent	`/api/web-bridge/v1/ttn/check-availability`	`X-Fatoora-Signer-Session`
POST	Local agent	`/api/web-bridge/v1/ttn/save-efact`	`X-Fatoora-Signer-Session`
POST	Local agent	`/api/web-bridge/v1/ttn/consult-efact`	`X-Fatoora-Signer-Session`

Sign request/response fields

```
// request
{ "xml": "...", "pin": "1234", "encryptedPin": null, "pkcs11Path": null,
  "slotId": null, "keyAlias": null, "sha1": null, "subjectContains": null,
  "includeChain": true, "signerRole": "CEO" }
// response
{ "mode": "sign", "ok": true, "signedXml": "...", "signatureLevel": "...",
  "error": null, "errorCode": null }
```

14. OpenAPI spec and Postman collection

OpenAPI specification

Hand-authored — no generated partner-signer-scoped OpenAPI source exists in the codebase (fatooraBusiness's generic `/open-api` spec doesn't cover it, and the local agent doesn't expose Swagger/OpenAPI publicly). Covers both hosts, using per-operation `servers` overrides for the local-agent paths. Also saved as a standalone file: `partner-signer-openapi.yaml`.

```yaml
openapi: 3.0.0

info:

  title: Fatoora Partner Signer

  version: "1.0"

  description: >
    Two-host API for Fatoora Partner Signer integrators (partner-signer / partner-signer-pro
        plans).

    Domain-whitelist and client-pre-registration management run on the Fatoora cloud

    (https://business.fatoora.tn). Session opening, local PKCS#11 signing, and optional direct TTN

    filing run on the FatooraSigner agent installed on the end client's own machine

    (http://127.0.0.1:38443 by default) — see the per-operation `servers` override on each path.

    Not extracted from a single generated source file; hand-authored from

    fatooraBusiness/src/routes/{signer-bridge,partner-signer}.routes.ts and

    fatooraSigner's tn.fatoora.signer.webbridge package. Cloud-side paths (domains/clients CRUD,
    session validation) were live-verified against a dev instance on 2026-07-11, which found and
        fixed
    a critical bug (domain whitelist writes crashed with 500 on every call) and flagged an
        unresolved

    CORS architecture gap — see the trust-signer-session path description. Local-agent paths

    (127.0.0.1:38443) remain code-derived; no agent instance was available to test against live.

  contact:

    name: Fatoora Support

    email: support@fatoora.tn

servers:

  - url: https://business.fatoora.tn

    description: Fatoora Cloud (production)


paths:

  /api/auth/login:

    post:

      summary: Authenticate the end client and obtain a short-lived JWT

      security: []

      requestBody:

        required: true

        content:

          application/json:

            schema:

              type: object

              required: [email, password]

              properties:

                email: { type: string }

                password: { type: string }
```

```
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                accessToken: { type: string }

/api/partner/signer/domains:
  get:
    summary: List whitelisted domains for this partner app
    security: [{ bearerAuth: [] }]
    responses:
      "200":
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties:
                domains:
                  type: array
                  items:
                    type: object
                    properties: { id: { type: string }, domain: { type: string } }
                total: { type: integer }
                partnerId: { type: string }
  post:
    summary: Whitelist a domain
    security: [{ bearerAuth: [] }]
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required: [domain]
            properties:
```

```
              domain:
                type: string
                description: Bare hostname, e.g. "yourapp.example.com" (no scheme)
    responses:
      "201":
        description: Created
      "403":
        description: DOMAIN_LIMIT_REACHED
      "409":
        description: DOMAIN_ALREADY_EXISTS

/api/partner/signer/domains/{domain}:
  delete:
    summary: Remove a whitelisted domain
    security: [{ bearerAuth: [] }]
    parameters:
      - name: domain
        in: path
        required: true
        schema: { type: string }
    responses:
      "200":
        description: OK
      "404":
        description: DOMAIN_NOT_FOUND

/api/partner/signer/clients:
  get:
    summary: List pre-registered client tax IDs
    security: [{ bearerAuth: [] }]
    parameters:
      - name: search
        in: query
        schema: { type: string }
      - name: active
        in: query
        schema: { type: string, enum: ["true", "false"] }
    responses:
      "200":
        description: OK
```

```
          content:
            application/json:
              schema:
                type: object
                properties:
                  clients:
                    type: array
                    items: { $ref: "#/components/schemas/SignerClient" }
                  total: { type: integer }
    post:
      summary: Pre-register (or reactivate) a client tax ID
      security: [{ bearerAuth: [] }]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [taxId]
              properties:
                taxId: { type: string }
                clientName: { type: string }
      responses:
        "201":
          description: Created / reactivated
          content:
            application/json:
              schema: { $ref: "#/components/schemas/SignerClient" }
        "403":
          description: CLIENT_LIMIT_REACHED

  /api/partner/signer/clients/bulk:
    post:
      summary: Bulk pre-register client tax IDs
      security: [{ bearerAuth: [] }]
      requestBody:
        required: true
        content:
          application/json:
            schema:
```

```
              type: object
              required: [clients]
              properties:
                clients:
                  type: array
                  minItems: 1
                  maxItems: 200
                  items:
                    type: object
                    required: [taxId]
                    properties:
                      taxId: { type: string }
                      clientName: { type: string }
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties:
                  success: { type: boolean }
                  inserted: { type: integer }
                  updated: { type: integer }
                  errors:
                    type: array
                    items:
                      type: object
                      properties: { taxId: { type: string }, error: { type: string } }
        "403":
          description: CLIENT_LIMIT_REACHED

  /api/partner/signer/clients/{id}:
    patch:
      summary: Activate or deactivate a pre-registered client
      security: [{ bearerAuth: [] }]
      parameters:
        - name: id
          in: path
          required: true
```

```
          schema: { type: string }
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            required: [isActive]
            properties: { isActive: { type: boolean } }
    responses:
      "200":
        description: OK
      "404":
        description: CLIENT_NOT_FOUND
  delete:
    summary: Remove a pre-registered client
    security: [{ bearerAuth: [] }]
    parameters:
      - name: id
        in: path
        required: true
        schema: { type: string }
    responses:
      "200":
        description: OK
      "404":
        description: CLIENT_NOT_FOUND


/api/auth/trust-signer-session:
  get:
    summary: Validate a signing session (called by the FatooraSigner agent, not directly by
      partners)
    description: >
      Not called directly by partner apps — the FatooraSigner local agent calls this on the
      partner's/client's behalf when opening a session. Documented here for
      completeness/troubleshooting.
      WARNING (confirmed live, 2026-07-11): this route sits behind fatooraBusiness's global
      Express
      cors() middleware, driven by the server's CORS_ORIGINS env var — a separate allowlist,
      unaware
      of the partner's self-service allowed_domains. An Origin not in CORS_ORIGINS is rejected
      with
      500 {"message":"Origin <url> not allowed by CORS"} BEFORE the allowed_domains check (which
      would
```

```
        otherwise return the documented 403 ORIGIN_NOT_WHITELISTED) ever runs. A partner's own
        domain
        whitelist entry is necessary but not sufficient in production; Fatoora staff must also add
        the
        domain to CORS_ORIGINS server-side.
        Validation order confirmed live: PARTNER_NOT_FOUND → NOT_SIGNER_PARTNER →
        PLAN_NOT_ELIGIBLE →
        ORIGIN_NOT_WHITELISTED → CLIENT_NOT_PRE_REGISTERED (a paused subscription surfaces
        PLAN_NOT_ELIGIBLE even with no domain/client configured).
      security: [{ bearerAuth: [] }]
      parameters:
        - name: X-Partner-App-ID
          in: header
          required: false
          schema: { type: string }
        - name: Origin
          in: header
          required: false
          schema: { type: string }
      responses:
        "500":
          description: >
            "Origin <url> not allowed by CORS" — global CORS rejection, confirmed live, occurs
        before
            any of the 403 checks below. Not represented in the response body schema (plain-text-
        ish
            JSON {message} from the Express cors() error handler, not the app's own ErrorResponse
        shape).
        "200":
          description: OK
          content:
            application/json:
              schema: { $ref: "#/components/schemas/SessionDescriptor" }
        "403":
          description: PARTNER_NOT_FOUND / NOT_SIGNER_PARTNER / PLAN_NOT_ELIGIBLE /
        ORIGIN_NOT_WHITELISTED / CLIENT_NOT_PRE_REGISTERED


  /api/web-bridge/v1/session:
    post:
      summary: Open a local signing session
      servers:
        - url: http://127.0.0.1:38443
          description: FatooraSigner local agent (default port; user-configurable)
      security: [{ bearerAuth: [] }]
      parameters:
```

```
        - name: X-Partner-App-ID
          in: header
          required: false
          schema: { type: string }
      requestBody:
        required: false
      responses:
        "200":
          description: Session opened
          content:
            application/json:
              schema:
                type: object
                properties:
                  sessionId: { type: string }
                  idleExpiresAt: { type: string, format: date-time }
                  absoluteExpiresAt: { type: string, format: date-time, nullable: true }
                  partnerMode: { type: boolean }
                  partnerAppId: { type: string }
        "401":
          description: Invalid or expired token
        "403":
          description: Forwarded cloud error code (ORIGIN_NOT_WHITELISTED,
        CLIENT_NOT_PRE_REGISTERED, PLAN_NOT_ELIGIBLE, ...)
        "502":
          description: Cloud unreachable
    delete:
      summary: Close a signing session
      servers:
        - url: http://127.0.0.1:38443
          description: FatooraSigner local agent
      security: [{ sessionAuth: [] }]
      responses:
        "204":
          description: Session closed


  /api/web-bridge/v1/tokens/certificates:
    get:
      summary: List public certificates on the connected PKCS#11 token (no PIN required)
      servers:
        - url: http://127.0.0.1:38443
```

```
        description: FatooraSigner local agent
    security: [{ sessionAuth: [] }]
    parameters:
      - name: driverId
        in: query
        required: false
        schema: { type: string }
    responses:
      "200":
        description: OK

/api/web-bridge/v1/tokens/detect-with-certificates:
  post:
    summary: Detect PKCS#11 tokens and read certificate details (PIN required)
    servers:
      - url: http://127.0.0.1:38443
        description: FatooraSigner local agent
    security: [{ sessionAuth: [] }]
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              pin: { type: string }
              encryptedPin: { type: string }
              driverId: { type: string, nullable: true }
    responses:
      "200":
        description: OK

/api/web-bridge/v1/xml/sign:
  post:
    summary: Sign a TEIF XML document with the client's local PKCS#11 token
    servers:
      - url: http://127.0.0.1:38443
        description: FatooraSigner local agent
    security: [{ sessionAuth: [] }]
    requestBody:
```

```
      required: true
      content:
        application/json:
          schema: { $ref: "#/components/schemas/SignRequest" }
    responses:
      "200":
        description: >
          Always 200 unless the native PKCS#11 library fails to load — check the `ok` and
          `errorCode` fields for business-level success/failure.
        content:
          application/json:
            schema: { $ref: "#/components/schemas/SignResponse" }
      "500":
        description: Native library load failure (LinkageError / ClassNotFoundException)

  /api/web-bridge/v1/xml/inspect:
    post:
      summary: Inspect / verify a signed TEIF XML document
      servers:
        - url: http://127.0.0.1:38443
          description: FatooraSigner local agent
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [xml]
              properties:
                xml: { type: string }
                revocationOnline: { type: boolean, default: true }
                normalizeTtnReturnedXmlForDss: { type: boolean, default: false }
      responses:
        "200":
          description: DSS validation details

  /api/web-bridge/v1/credentials/encrypt-pin:
    post:
      summary: Encrypt a PKCS#11 PIN, session-bound, in-memory only
      servers:
```

```
        - url: http://127.0.0.1:38443
          description: FatooraSigner local agent
      security: [{ sessionAuth: [] }]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [pin]
              properties: { pin: { type: string } }
      responses:
        "200":
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties: { encryptedPin: { type: string } }

  /api/web-bridge/v1/credentials/encrypt-ttn:
    post:
      summary: Encrypt TTN (TradeNet) credentials, session-bound, in-memory only
      servers:
        - url: http://127.0.0.1:38443
          description: FatooraSigner local agent
      security: [{ sessionAuth: [] }]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [login, password, matricule]
              properties:
                login: { type: string }
                password: { type: string }
                matricule: { type: string }
      responses:
        "200":
```

```
        description: OK
        content:
          application/json:
            schema:
              type: object
              properties: { encryptedCredential: { type: string } }

  /api/web-bridge/v1/ttn/check-availability:
    get:
      summary: Check TTN availability and latency
      servers:
        - url: http://127.0.0.1:38443
          description: FatooraSigner local agent
      security: [{ sessionAuth: [] }]
      parameters:
        - name: ttnMode
          in: query
          schema: { type: string, enum: [TEST, PROD] }
      responses:
        "200":
          description: OK

  /api/web-bridge/v1/ttn/save-efact:
    post:
      summary: Submit a signed invoice directly to TTN
      servers:
        - url: http://127.0.0.1:38443
          description: FatooraSigner local agent
      security: [{ sessionAuth: [] }]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [encryptedCredential, documentEfact]
              properties:
                encryptedCredential: { type: string }
                documentEfact: { type: string, description: "Base64-encoded signed TEIF XML" }
                ttnMode: { type: string, enum: [TEST, PROD], default: PROD }
```

```
                ttnUrl: { type: string, nullable: true }
      responses:
        "200":
          description: TTN submission result

  /api/web-bridge/v1/ttn/consult-efact:
    post:
      summary: Pull received invoices from TTN
      servers:
        - url: http://127.0.0.1:38443
          description: FatooraSigner local agent
      security: [{ sessionAuth: [] }]
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [encryptedCredential, efactCriteria]
              properties:
                encryptedCredential: { type: string }
                efactCriteria: { type: object }
                ttnMode: { type: string, enum: [TEST, PROD], default: PROD }
      responses:
        "200":
          description: Received invoices list

components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
      description: End-client JWT from POST /api/auth/login
    sessionAuth:
      type: apiKey
      in: header
      name: X-Fatoora-Signer-Session
      description: Session id returned by POST /api/web-bridge/v1/session
```

```
  schemas:
    SessionDescriptor:
      type: object
      properties:
        valid: { type: boolean }
        sub: { type: string }
        userId: { type: string }
        orgId: { type: string }
        roles: { type: array, items: { type: string } }
        scopes: { type: array, items: { type: string } }
        partnerMode: { type: boolean }
        partnerAppId: { type: string }

    SignerClient:
      type: object
      properties:
        id: { type: string }
        taxId: { type: string }
        clientName: { type: string, nullable: true }
        isActive: { type: boolean }
        createdAt: { type: string, format: date-time }

    SignRequest:
      type: object
      required: [xml]
      properties:
        xml: { type: string }
        pin: { type: string }
        encryptedPin: { type: string }
        pkcs11Path: { type: string }
        slotId: { type: integer }
        keyAlias: { type: string }
        sha1: { type: string }
        subjectContains: { type: string }
        includeChain: { type: boolean }
        signerRole: { type: string }

    SignResponse:
      type: object
      properties:
```

```
        mode: { type: string, example: sign }
        ok: { type: boolean }
        signedXml: { type: string, nullable: true }
        signatureLevel: { type: string, nullable: true }
        error: { type: string, nullable: true }
        errorCode: { type: string, nullable: true, example: SIGN_FAILED }
```

Postman collection

Not hand-written — it's the literal output of `buildSignerPostmanCollection()` in `fatooraLive/src/components/partner/credentials/PartnerSignerGuide.tsx`, extracted and executed with Node to guarantee fidelity with the code (group names translated to English; the code's own French labels are `Authentification utilisateur`, `Gestion session bridge locale`, etc.). Also saved as a standalone file: `partner-signer-postman-collection.json`.

Every request now carries one or more **saved example responses** (success and, where instructive, the most common error) directly in the collection — Postman shows these in the "Examples" tab so partners can see a realistic response body and status code without needing a live agent running first.

> The domain-format inconsistency previously flagged here (`POST /api/partner/signer/domains` example using a full URL instead of a bare hostname) has been fixed at the source — the example now uses `"domain": "yourapp.example.com"`, consistent with the endpoint's validation regex (see §2).

```
{
  "info": {
    "name": "Fatoora Partner Signer — Web Bridge",
    "description": "Integration guide for Fatoora Partner Signer. Covers user authentication,
        local bridge session management, XML signing and inspection via FatooraSigner running at
        127.0.0.1:38443.",
    "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json"
  },
  "variable": [
    {
      "key": "business_url",
      "value": "https://business.fatoora.tn",
      "type": "string"
    },
    {
      "key": "signer_url",
      "value": "http://127.0.0.1:38443",
      "type": "string"
    },
    {
      "key": "partner_app_id",
      "value": "YOUR_PARTNER_APP_ID",
      "type": "string"
    },
    {
      "key": "user_email",
      "value": "",
      "type": "string"
    },
    {
      "key": "user_password",
      "value": "",
      "type": "string"
    },
    {
      "key": "access_token",
      "value": "",
      "type": "string"
    },
    {
      "key": "session_id",
```

```
      "value": "",
      "type": "string"
    },
    {
      "key": "token_pin",
      "value": "",
      "type": "string"
    }
  ],
  "item": [
    {
      "name": "1 — User authentication",
      "item": [
        {
          "name": "POST /api/auth/login (get JWT)",
          "request": {
            "method": "POST",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              }
            ],
            "body": {
              "mode": "raw",
              "raw": "{\n  \"email\": \"{{user_email}}\",\n  \"password\": \"
      {{user_password}}\"\n}",
              "options": {
                "raw": {
                  "language": "json"
                }
              }
            },
            "url": {
              "raw": "{{business_url}}/api/auth/login",
              "host": [
                "{{business_url}}/api/auth/login"
              ]
            },
            "description": "Authenticate the end-user and retrieve an access token. Save
      'accessToken' to the 'access_token' variable."
```

```
  },
  "response": [
    {
      "name": "200 OK",
      "originalRequest": {
        "method": "POST",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          }
        ],
        "url": {
          "raw": "{{business_url}}/api/auth/login",
          "host": [
            "{{business_url}}/api/auth/login"
          ]
        }
      },
      "status": "OK",
      "code": 200,
      "_postman_previewlanguage": "json",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        }
      ],
      "cookie": [],
      "body": "{\n  \"accessToken\":
\"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJ1c2VyX3h4eHgifQ.SIGNATURE\",\n
\"refreshToken\": \"eyJhbGciOiJIUzI1NiIs...\",\n  \"expiresIn\": 900\n}"
    },
    {
      "name": "401 Unauthorized — wrong credentials",
      "originalRequest": {
        "method": "POST",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
```

```
            }
          ],
          "url": {
            "raw": "{{business_url}}/api/auth/login",
            "host": [
              "{{business_url}}/api/auth/login"
            ]
          }
        },
        "status": "Unauthorized",
        "code": 401,
        "_postman_previewlanguage": "json",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          }
        ],
        "cookie": [],
        "body": "{\n  \"error\": \"INVALID_CREDENTIALS\",\n  \"message\": \"Invalid email or password.\"\n}"
      }
    ]
  },
  {
    "name": "POST /api/auth/refresh (refresh JWT)",
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        },
        {
          "key": "Authorization",
          "value": "Bearer {{access_token}}"
        }
      ],
      "url": {
        "raw": "{{business_url}}/api/auth/refresh",
        "host": [
```

```
            "{{business_url}}/api/auth/refresh"
          ]
        },
        "description": "Refresh an expiring access token."
      },
      "response": [
        {
          "name": "200 OK",
          "originalRequest": {
            "method": "POST",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              },
              {
                "key": "Authorization",
                "value": "Bearer {{access_token}}"
              }
            ],
            "url": {
              "raw": "{{business_url}}/api/auth/refresh",
              "host": [
                "{{business_url}}/api/auth/refresh"
              ]
            }
          },
          "status": "OK",
          "code": 200,
          "_postman_previewlanguage": "json",
          "header": [
            {
              "key": "Content-Type",
              "value": "application/json"
            }
          ],
          "cookie": [],
          "body": "{\n  \"accessToken\": \"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...NEW\",\n
\"expiresIn\": 900\n}"
        }
      ]
```

```
        }
      ]
    },
    {
      "name": "2 — Local bridge session management",
      "item": [
        {
          "name": "POST /session (open bridge session)",
          "request": {
            "method": "POST",
            "header": [
              {
                "key": "Authorization",
                "value": "Bearer {{access_token}}"
              },
              {
                "key": "X-Partner-App-ID",
                "value": "{{partner_app_id}}"
              }
            ],
            "url": {
              "raw": "{{signer_url}}/api/web-bridge/v1/session",
              "host": [
                "{{signer_url}}/api/web-bridge/v1/session"
              ]
            },
            "description": "Opens a local bridge session on the client's machine. Requires
          FatooraSigner running at 127.0.0.1. The X-Partner-App-ID header activates partner-signer
          validation (allowed domains + pre-registered client check). Save 'sessionId' to the
          'session_id' variable."
          },
          "response": [
            {
              "name": "200 OK",
              "originalRequest": {
                "method": "POST",
                "header": [
                  {
                    "key": "Authorization",
                    "value": "Bearer {{access_token}}"
                  },
                  {
```

```
            "key": "X-Partner-App-ID",
            "value": "{{partner_app_id}}"
          }
        ],
        "url": {
          "raw": "{{signer_url}}/api/web-bridge/v1/session",
          "host": [
            "{{signer_url}}/api/web-bridge/v1/session"
          ]
        }
      },
      "status": "OK",
      "code": 200,
      "_postman_previewlanguage": "json",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        }
      ],
      "cookie": [],
      "body": "{\n  \"sessionId\": \"sess_xxxxxxxxxxxxxxxx\",\n  \"idleExpiresAt\":
\"2026-07-11T15:00:00Z\",\n  \"absoluteExpiresAt\": null,\n  \"partnerMode\": true,\n
\"partnerAppId\": \"{{partner_app_id}}\"\n}"
    },
    {
      "name": "403 Forbidden — domain not whitelisted",
      "originalRequest": {
        "method": "POST",
        "header": [
          {
            "key": "Authorization",
            "value": "Bearer {{access_token}}"
          },
          {
            "key": "X-Partner-App-ID",
            "value": "{{partner_app_id}}"
          }
        ],
        "url": {
          "raw": "{{signer_url}}/api/web-bridge/v1/session",
```

```
            "host": [
              "{{signer_url}}/api/web-bridge/v1/session"
            ]
          }
        },
        "status": "Forbidden",
        "code": 403,
        "_postman_previewlanguage": "json",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          }
        ],
        "cookie": [],
        "body": "{\n  \"error\": \"ORIGIN_NOT_WHITELISTED\"\n}"
      },
      {
        "name": "403 Forbidden — client not pre-registered",
        "originalRequest": {
          "method": "POST",
          "header": [
            {
              "key": "Authorization",
              "value": "Bearer {{access_token}}"
            },
            {
              "key": "X-Partner-App-ID",
              "value": "{{partner_app_id}}"
            }
          ],
          "url": {
            "raw": "{{signer_url}}/api/web-bridge/v1/session",
            "host": [
              "{{signer_url}}/api/web-bridge/v1/session"
            ]
          }
        },
        "status": "Forbidden",
        "code": 403,
```

```
            "_postman_previewlanguage": "json",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              }
            ],
            "cookie": [],
            "body": "{\n  \"error\": \"CLIENT_NOT_PRE_REGISTERED\"\n}"
          }
        ]
      },
      {
        "name": "DELETE /session (close bridge session)",
        "request": {
          "method": "DELETE",
          "header": [
            {
              "key": "X-Fatoora-Signer-Session",
              "value": "{{session_id}}"
            }
          ],
          "url": {
            "raw": "{{signer_url}}/api/web-bridge/v1/session",
            "host": [
              "{{signer_url}}/api/web-bridge/v1/session"
            ]
          },
          "description": "Closes the bridge session. Call on user logout."
        },
        "response": [
          {
            "name": "204 No Content",
            "originalRequest": {
              "method": "DELETE",
              "header": [
                {
                  "key": "X-Fatoora-Signer-Session",
                  "value": "{{session_id}}"
                }
```

```
            ],
            "url": {
              "raw": "{{signer_url}}/api/web-bridge/v1/session",
              "host": [
                "{{signer_url}}/api/web-bridge/v1/session"
              ]
            }
          },
          "status": "No Content",
          "code": 204,
          "_postman_previewlanguage": "json",
          "header": [
            {
              "key": "Content-Type",
              "value": "application/json"
            }
          ],
          "cookie": [],
          "body": ""
        }
      ]
    }
  ]
},
{
  "name": "3 — PKCS#11 token",
  "item": [
    {
      "name": "GET /tokens/certificates (public certs, no PIN)",
      "request": {
        "method": "GET",
        "header": [
          {
            "key": "X-Fatoora-Signer-Session",
            "value": "{{session_id}}"
          }
        ],
        "url": {
          "raw": "{{signer_url}}/api/web-bridge/v1/tokens/certificates",
          "host": [
```

```
            "{{signer_url}}/api/web-bridge/v1/tokens/certificates"
          ]
        },
        "description": "Lists public certificates on the USB token without requiring a PIN."
      },
      "response": [
        {
          "name": "200 OK",
          "originalRequest": {
            "method": "GET",
            "header": [
              {
                "key": "X-Fatoora-Signer-Session",
                "value": "{{session_id}}"
              }
            ],
            "url": {
              "raw": "{{signer_url}}/api/web-bridge/v1/tokens/certificates",
              "host": [
                "{{signer_url}}/api/web-bridge/v1/tokens/certificates"
              ]
            }
          },
          "status": "OK",
          "code": 200,
          "_postman_previewlanguage": "json",
          "header": [
            {
              "key": "Content-Type",
              "value": "application/json"
            }
          ],
          "cookie": [],
          "body": "{\n  \"certificates\": [\n    {\n      \"slotId\": 0,\n      \"keyAlias\": \"TunTrust Signing Key\",\n      \"subject\": \"CN=John Doe,O=Client SARL,C=TN\",\n      \"sha1\": \"0123456789ABCDEF0123456789ABCDEF01234567\",\n      \"validFrom\": \"2025-01-01T00:00:00Z\",\n      \"validTo\": \"2028-01-01T00:00:00Z\"\n    }\n  ]\n}"
        }
      ]
    },
    {
      "name": "POST /tokens/detect-with-certificates (with PIN)",
```

```
        "request": {
          "method": "POST",
          "header": [
            {
              "key": "Content-Type",
              "value": "application/json"
            },
            {
              "key": "X-Fatoora-Signer-Session",
              "value": "{{session_id}}"
            }
          ],
          "body": {
            "mode": "raw",
            "raw": "{\n  \"pin\": \"{{token_pin}}\",\n  \"driverId\": null\n}",
            "options": {
              "raw": {
                "language": "json"
              }
            }
          },
          "url": {
            "raw": "{{signer_url}}/api/web-bridge/v1/tokens/detect-with-certificates",
            "host": [
              "{{signer_url}}/api/web-bridge/v1/tokens/detect-with-certificates"
            ]
          },
          "description": "Detects PKCS#11 tokens and reads certificate details. PIN is required.
      The PIN never leaves the client machine."
        },
        "response": [
          {
            "name": "200 OK",
            "originalRequest": {
              "method": "POST",
              "header": [
                {
                  "key": "Content-Type",
                  "value": "application/json"
                },
                {
```

```
            "key": "X-Fatoora-Signer-Session",
            "value": "{{session_id}}"
          }
        ],
        "url": {
          "raw": "{{signer_url}}/api/web-bridge/v1/tokens/detect-with-certificates",
          "host": [
            "{{signer_url}}/api/web-bridge/v1/tokens/detect-with-certificates"
          ]
        }
      },
      "status": "OK",
      "code": 200,
      "_postman_previewlanguage": "json",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        }
      ],
      "cookie": [],
      "body": "{\n  \"driverId\": \"opensc\",\n  \"slots\": [\n    {\n      \"slotId\":
0,\n      \"label\": \"TunTrust Card\",\n      \"certificates\": [\n        {\n
\"keyAlias\": \"TunTrust Signing Key\",\n          \"subject\": \"CN=John Doe,O=Client
SARL,C=TN\",\n          \"sha1\": \"0123456789ABCDEF0123456789ABCDEF01234567\"\n
}\n      ]\n    }\n  ]\n}"
    },
    {
      "name": "200 OK — wrong PIN (business error)",
      "originalRequest": {
        "method": "POST",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          },
          {
            "key": "X-Fatoora-Signer-Session",
            "value": "{{session_id}}"
          }
        ],
        "url": {
```

```
                "raw": "{{signer_url}}/api/web-bridge/v1/tokens/detect-with-certificates",
                "host": [
                  "{{signer_url}}/api/web-bridge/v1/tokens/detect-with-certificates"
                ]
              }
            },
            "status": "OK",
            "code": 200,
            "_postman_previewlanguage": "json",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              }
            ],
            "cookie": [],
            "body": "{\n  \"ok\": false,\n  \"error\": \"Invalid PIN\",\n  \"errorCode\":
    \"INVALID_PIN\"\n}"
          }
        ]
      }
    ]
  },
  {
    "name": "4 — XML signature",
    "item": [
      {
        "name": "POST /xml/sign (sign TEIF document)",
        "request": {
          "method": "POST",
          "header": [
            {
              "key": "Content-Type",
              "value": "application/json"
            },
            {
              "key": "X-Fatoora-Signer-Session",
              "value": "{{session_id}}"
            }
          ],
          "body": {
```

```
      "mode": "raw",
      "raw": "{\n  \"xml\": \"{{teif_xml}}\",\n  \"pin\": \"{{token_pin}}\",\n
\"signerRole\": \"CEO\",\n  \"includeChain\": true\n}",
      "options": {
        "raw": {
          "language": "json"
        }
      }
    },
    "url": {
      "raw": "{{signer_url}}/api/web-bridge/v1/xml/sign",
      "host": [
        "{{signer_url}}/api/web-bridge/v1/xml/sign"
      ]
    },
    "description": "Signs a TEIF XML document using the client's local USB token. Returns
'signedXml'. Check 'ok' and 'errorCode' fields — a 200 response can still contain a
business error."
  },
  "response": [
    {
      "name": "200 OK — signed",
      "originalRequest": {
        "method": "POST",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          },
          {
            "key": "X-Fatoora-Signer-Session",
            "value": "{{session_id}}"
          }
        ],
        "url": {
          "raw": "{{signer_url}}/api/web-bridge/v1/xml/sign",
          "host": [
            "{{signer_url}}/api/web-bridge/v1/xml/sign"
          ]
        }
      },
      "status": "OK",
```

```
      "code": 200,
      "_postman_previewlanguage": "json",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        }
      ],
      "cookie": [],
      "body": "{\n  \"mode\": \"sign\",\n  \"ok\": true,\n  \"signedXml\": \"<Invoice>...
(signed)...</Invoice>\",\n  \"signatureLevel\": \"XAdES-BASELINE-B\"\n}"
    },
    {
      "name": "200 OK — business error (wrong PIN)",
      "originalRequest": {
        "method": "POST",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          },
          {
            "key": "X-Fatoora-Signer-Session",
            "value": "{{session_id}}"
          }
        ],
        "url": {
          "raw": "{{signer_url}}/api/web-bridge/v1/xml/sign",
          "host": [
            "{{signer_url}}/api/web-bridge/v1/xml/sign"
          ]
        }
      },
      "status": "OK",
      "code": 200,
      "_postman_previewlanguage": "json",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        }
```

```
        ],
        "cookie": [],
        "body": "{\n  \"mode\": \"sign\",\n  \"ok\": false,\n  \"error\": \"Invalid PIN\",\n  \"errorCode\": \"SIGN_FAILED\"\n}"
      }
    ]
  },
  {
    "name": "POST /xml/inspect (inspect signed XML)",
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        },
        {
          "key": "X-Fatoora-Signer-Session",
          "value": "{{session_id}}"
        }
      ],
      "body": {
        "mode": "raw",
        "raw": "{\n  \"xml\": \"{{signed_teif_xml}}\"\n}",
        "options": {
          "raw": {
            "language": "json"
          }
        }
      },
      "url": {
        "raw": "{{signer_url}}/api/web-bridge/v1/xml/inspect",
        "host": [
          "{{signer_url}}/api/web-bridge/v1/xml/inspect"
        ]
      },
      "description": "Verifies signatures in a signed XML. Returns DSS validation details."
    },
    "response": [
      {
        "name": "200 OK",
```

```
            "originalRequest": {
              "method": "POST",
              "header": [
                {
                  "key": "Content-Type",
                  "value": "application/json"
                },
                {
                  "key": "X-Fatoora-Signer-Session",
                  "value": "{{session_id}}"
                }
              ],
              "url": {
                "raw": "{{signer_url}}/api/web-bridge/v1/xml/inspect",
                "host": [
                  "{{signer_url}}/api/web-bridge/v1/xml/inspect"
                ]
              }
            },
            "status": "OK",
            "code": 200,
            "_postman_previewlanguage": "json",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              }
            ],
            "cookie": [],
            "body": "{\n  \"valid\": true,\n  \"signatureLevel\": \"XAdES-BASELINE-B\",\n
      \"signerCertificate\": {\n    \"subject\": \"CN=John Doe,O=Client SARL,C=TN\",\n
      \"issuer\": \"CN=TunTrust CA\"\n  },\n  \"revocationStatus\": \"GOOD\"\n}"
          }
        ]
      }
    ]
  },
  {
    "name": "5 — Domain & client management (partner)",
    "item": [
      {
```

```
      "name": "GET /api/partner/signer/domains",
      "request": {
        "method": "GET",
        "header": [
          {
            "key": "Authorization",
            "value": "Bearer {{access_token}}"
          }
        ],
        "url": {
          "raw": "{{business_url}}/api/partner/signer/domains",
          "host": [
            "{{business_url}}/api/partner/signer/domains"
          ]
        },
        "description": "Lists domains allowed to initiate bridge sessions for this partner
    app."
      },
      "response": [
        {
          "name": "200 OK",
          "originalRequest": {
            "method": "GET",
            "header": [
              {
                "key": "Authorization",
                "value": "Bearer {{access_token}}"
              }
            ],
            "url": {
              "raw": "{{business_url}}/api/partner/signer/domains",
              "host": [
                "{{business_url}}/api/partner/signer/domains"
              ]
            }
          },
          "status": "OK",
          "code": 200,
          "_postman_previewlanguage": "json",
          "header": [
            {
```

```
            "key": "Content-Type",
            "value": "application/json"
          }
        ],
        "cookie": [],
        "body": "{\n  \"domains\": [\n    {\n      \"id\": \"domain_0\",\n      \"domain\": \"yourapp.example.com\"\n    }\n  ],\n  \"total\": 1,\n  \"partnerId\": \"{{partner_app_id}}\"\n}"
      }
    ]
  },
  {
    "name": "POST /api/partner/signer/domains",
    "request": {
      "method": "POST",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        },
        {
          "key": "Authorization",
          "value": "Bearer {{access_token}}"
        }
      ],
      "body": {
        "mode": "raw",
        "raw": "{\n  \"domain\": \"yourapp.example.com\"\n}",
        "options": {
          "raw": {
            "language": "json"
          }
        }
      },
      "url": {
        "raw": "{{business_url}}/api/partner/signer/domains",
        "host": [
          "{{business_url}}/api/partner/signer/domains"
        ]
      },
      "description": "Whitelists a bare hostname (no scheme) — subdomains of a whitelisted domain also match."
```

```
    },
    "response": [
      {
        "name": "201 Created",
        "originalRequest": {
          "method": "POST",
          "header": [
            {
              "key": "Content-Type",
              "value": "application/json"
            },
            {
              "key": "Authorization",
              "value": "Bearer {{access_token}}"
            }
          ],
          "url": {
            "raw": "{{business_url}}/api/partner/signer/domains",
            "host": [
              "{{business_url}}/api/partner/signer/domains"
            ]
          }
        },
        "status": "Created",
        "code": 201,
        "_postman_previewlanguage": "json",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          }
        ],
        "cookie": [],
        "body": "{\n  \"success\": true,\n  \"domain\": \"yourapp.example.com\",\n
\"totalDomains\": 1\n}"
      },
      {
        "name": "409 Conflict — already whitelisted",
        "originalRequest": {
          "method": "POST",
          "header": [
```

```
            {
              "key": "Content-Type",
              "value": "application/json"
            },
            {
              "key": "Authorization",
              "value": "Bearer {{access_token}}"
            }
          ],
          "url": {
            "raw": "{{business_url}}/api/partner/signer/domains",
            "host": [
              "{{business_url}}/api/partner/signer/domains"
            ]
          }
        },
        "status": "Conflict",
        "code": 409,
        "_postman_previewlanguage": "json",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          }
        ],
        "cookie": [],
        "body": "{\n  \"error\": \"DOMAIN_ALREADY_EXISTS\",\n  \"message\": \"Domain already whitelisted.\"\n}"
      },
      {
        "name": "403 Forbidden — plan limit reached",
        "originalRequest": {
          "method": "POST",
          "header": [
            {
              "key": "Content-Type",
              "value": "application/json"
            },
            {
              "key": "Authorization",
              "value": "Bearer {{access_token}}"
```

```
            }
          ],
          "url": {
            "raw": "{{business_url}}/api/partner/signer/domains",
            "host": [
              "{{business_url}}/api/partner/signer/domains"
            ]
          }
        },
        "status": "Forbidden",
        "code": 403,
        "_postman_previewlanguage": "json",
        "header": [
          {
            "key": "Content-Type",
            "value": "application/json"
          }
        ],
        "cookie": [],
        "body": "{\n  \"error\": \"DOMAIN_LIMIT_REACHED\",\n  \"message\": \"Your plan
allows a maximum of 1 whitelisted domain(s). Upgrade to add more.\",\n  \"limit\": 1\n}"
      }
    ]
  },
  {
    "name": "GET /api/partner/signer/clients",
    "request": {
      "method": "GET",
      "header": [
        {
          "key": "Authorization",
          "value": "Bearer {{access_token}}"
        }
      ],
      "url": {
        "raw": "{{business_url}}/api/partner/signer/clients",
        "host": [
          "{{business_url}}/api/partner/signer/clients"
        ]
      }
    },
```

```
  "response": [
    {
      "name": "200 OK",
      "originalRequest": {
        "method": "GET",
        "header": [
          {
            "key": "Authorization",
            "value": "Bearer {{access_token}}"
          }
        ],
        "url": {
          "raw": "{{business_url}}/api/partner/signer/clients",
          "host": [
            "{{business_url}}/api/partner/signer/clients"
          ]
        }
      },
      "status": "OK",
      "code": 200,
      "_postman_previewlanguage": "json",
      "header": [
        {
          "key": "Content-Type",
          "value": "application/json"
        }
      ],
      "cookie": [],
      "body": "{\n  \"clients\": [\n    {\n      \"id\": \"client_xxxx\",\n
\"taxId\": \"1234567ABC000\",\n      \"clientName\": \"Client SARL\",\n      \"isActive\":
true,\n      \"createdAt\": \"2026-07-11T00:00:00Z\"\n    }\n  ],\n  \"total\": 1\n}"
    }
  ]
},
{
  "name": "POST /api/partner/signer/clients",
  "request": {
    "method": "POST",
    "header": [
      {
        "key": "Content-Type",
```

```
            "value": "application/json"
          },
          {
            "key": "Authorization",
            "value": "Bearer {{access_token}}"
          }
        ],
        "body": {
          "mode": "raw",
          "raw": "{\n  \"taxId\": \"{{client_tax_id}}\",\n  \"clientName\": \"
{{client_name}}\"\n}",
          "options": {
            "raw": {
              "language": "json"
            }
          }
        },
        "url": {
          "raw": "{{business_url}}/api/partner/signer/clients",
          "host": [
            "{{business_url}}/api/partner/signer/clients"
          ]
        },
        "description": "Pre-registers a client tax ID so FatooraSigner accepts bridge sessions
for this org."
      },
      "response": [
        {
          "name": "201 Created",
          "originalRequest": {
            "method": "POST",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              },
              {
                "key": "Authorization",
                "value": "Bearer {{access_token}}"
              }
            ],
```

```
            "url": {
              "raw": "{{business_url}}/api/partner/signer/clients",
              "host": [
                "{{business_url}}/api/partner/signer/clients"
              ]
            }
          },
          "status": "Created",
          "code": 201,
          "_postman_previewlanguage": "json",
          "header": [
            {
              "key": "Content-Type",
              "value": "application/json"
            }
          ],
          "cookie": [],
          "body": "{\n  \"id\": \"client_xxxx\",\n  \"taxId\": \"1234567ABC000\",\n
\"clientName\": \"Client SARL\",\n  \"isActive\": true,\n  \"createdAt\": \"2026-07-
11T00:00:00Z\"\n}"
        },
        {
          "name": "403 Forbidden — plan limit reached",
          "originalRequest": {
            "method": "POST",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              },
              {
                "key": "Authorization",
                "value": "Bearer {{access_token}}"
              }
            ],
            "url": {
              "raw": "{{business_url}}/api/partner/signer/clients",
              "host": [
                "{{business_url}}/api/partner/signer/clients"
              ]
            }
```

```
            },
            "status": "Forbidden",
            "code": 403,
            "_postman_previewlanguage": "json",
            "header": [
              {
                "key": "Content-Type",
                "value": "application/json"
              }
            ],
            "cookie": [],
            "body": "{\n  \"error\": \"CLIENT_LIMIT_REACHED\",\n  \"message\": \"Your plan
        allows a maximum of 50 pre-registered client(s). Upgrade to add more.\",\n  \"limit\":
        50\n}"
          }
        ]
      }
    ]
  }
```


---

*This guide reflects the Fatoora Partner Signer surface as implemented in `fatooraBusiness` (routes: `signer-bridge.routes.ts`, `partner-signer.routes.ts`, `partner-self-register.routes.ts`) and `fatooraSigner` (`tn.fatoora.signer.webbridge` package: `WebBridgeController`, `WebBridgeCredentialsController`, `WebBridgeTtnController`, `WebBridgeSessionService`), and has been **live-verified** against a dev instance (fatooraLive :3600 / fatooraBusiness :4100, test partner-signer account) on 2026-07-11: login, profile/credentials, the full domain-whitelist CRUD (add/duplicate/limit/delete), the full pre-registered-client CRUD (add/patch/bulk/search/delete), and the session-validation error chain (`PLAN_NOT_ELIGIBLE`, direct-Trust-mode rejection) were all exercised against live responses. The local FatooraSigner agent itself (`127.0.0.1:38443`) could not be exercised in this pass — no agent instance was running in the dev environment — so §5/§6's request/response shapes remain code-derived, not live-confirmed.

> That pass found and fixed one **critical bug**: `POST` / `DELETE /api/partner/signer/domains` crashed with `500 INTERNAL_ERROR` on every call (Postgres `malformed array literal`, from an array not being serialized into valid array-literal syntax before a `::text[]` cast) — fixed in `fatooraBusiness/src/routes/partner-signer.routes.ts`. It also found and **flagged (not fixed)** a significant architecture gap: fatooraBusiness's global CORS middleware is a separate, server-configured allowlist unaware of a partner's self-service domain whitelist — see the callout in §3 and §9. The domain-format inconsistency previously flagged in §14 was also fixed at the source (`fatooraLive/src/components/partner/credentials/PartnerSignerGuide.tsx`); the agent's non-restricted bind address flagged in §3 is still worth verifying/fixing before this guide is published externally.*

Diagrams are rendered from Mermaid sources in `partner-signer-diagrams/src/*.mmd` using `@mermaid-js/mermaid-cli` (`mmdc`) and embedded as base64 PNG data URIs directly in this file (self-contained — no dependency on sibling files). Regenerate with `partner-signer-diagrams/regenerate_and_embed.sh` (same pattern as the companion Partner API guide).