HOOX · AXIS · DOCUMENTATION

TRADERS · RESEARCHERS · SELF-HOSTERS

END USER MANUAL

Install the PWA, compose plugins, research workflows

Open charting PWA — own the axes, swap the engine

2026-08-03 DIN A4 · HTTPS://HOOX.SH/AXIS/DOCS OPEN SOURCE

CONTENTS

OVERVIEW

title: "AXIS Documentation" description: "Open charting PWA — orthogonal sources, streams, engines, and storage. Own the axes; swap the engine."

AXIS DOCUMENTATION

AXIS is an installable charting PWA that treats **price**, **time**, and **calculation** as separable axes. Sources load history. Streams advance the present. Engines evaluate Pine Script™ via **PYNE**. Storage keeps your script library. Compose any lawful combination without a proprietary charting host.

Pine Script™ and TradingView® are trademarks of TradingView, Inc. This project is independent and not affiliated with or endorsed by TradingView, Inc.

Abstract

AXIS is a charting PWA, not a language rewrite. Language fidelity lives in PYNE. The PWA contributes:

1. A **unified plugin registry** (`source | stream | engine | storage`)
2. A **Solid + Vite** product UI (chart, editor, results, manager)
3. Optional backends: local Flask, offline Pyodide, Cloudflare® Worker (+ DO/KV/D1/R2)

Invariant (AXIS ≠ engine): the shell never embeds a closed interpreter. Evaluation is always an engine plugin.

Product map

Track	Role	Start here
End User	Install, compose recipes, research workflows	End User hub
Architecture	ADRs, topologies, state namespaces	Architecture
Plugins	Contracts and built-in catalogs	Plugins
UI	Chart, editor, panels, store	UI
Worker	CF Pages + Worker data plane	Worker
DevOps	Build, VPS, CORS, CI	DevOps
PYNE (separate)	Grammar & evaluator	PYNE docs

Conceptual model

```
flowchart TB
  subgraph Browser["Browser PWA"]
    SRC[Sources]
    STR[Streams]
    ENG[Engines]
    STO[Storage]
    UI[Chart · Editor · Results]
    SRC --> UI
    STR --> UI
    ENG --> UI
    STO --> UI
  end
  ENG -->|server| FLASK[Flask Pro API]
  ENG -->|pyodide| PY[In-browser pynescript]
  ENG -->|worker proxy| CF[Cloudflare Worker]
  CF --> FLASK
```

Formal active set:

```
Active set = { source, stream, engine, storage }
Contract namespace: pynescript.axis.plugins.v1
App state: pynescript.axis.v1
```

Compose in one glance

Recipe	Source	Stream	Engine	Storage
Offline lab	`mock-walk`	`mock-poll`	`pyodide`	`local`
Desk research	`binance-rest`	`binance-ws`	`server` → Flask	`local`
AXIS at the edge	venue REST	venue WS / DO	`server` → Worker	`cloud`
Repo library	any	any	any	`git`

Full recipes: Compose recipes.

Infrastructure names (do not rename lightly)

Surface	Value
CF Wrangler / Pages project	`pynescript-axis` (frozen)
Health JSON service	`pynescript-axis-worker`
Product brand	**AXIS**

Demo topologies

- Dev: Vite `:3000` + Flask `:5002`
- Static: `bun run build` + `axis_pwa_server.py` `:8081`
- Worker: `wrangler` on `:8787`

See topologies and local dev.

Offline & agent exports

Auto-generated like HOOX manuals (`bun run docs:exports`):

Kind	Path
Full-corpus LLM pack	`llm.txt`
LLM site map (llmstxt.org)	`llms.txt`
Track PDFs (A4)	`/exports/axis-*-manual.pdf`

See also

- Quick start
- Plugin contracts
- ADRs
- PYNE runtime

OVERVIEW

title: "End User" description: "Install AXIS, compose source/stream/engine/storage recipes, and run research workflows without a proprietary chart host."

END USER

AXIS is an installable charting PWA: load history, stream the present, evaluate Pine Script™ through a pluggable engine, and keep a script library on disk, IndexedDB, cloud, or git. This track is for operators and researchers who use the AXIS—not for plugin authors or Worker deployers.

Language fidelity (grammar, builtins, runtime semantics) lives in PYNE. AXIS never reimplements the language; it **hosts** evaluation via engine plugins.

Abstract

You compose four axes:

Axis	Question it answers
Source	Where do historical OHLCV bars come from?
Stream	How does the live bar advance?
Engine	Who evaluates the script?
Storage	Where do saved scripts live?

Any lawful combination is valid. Offline lab, desk research against Binance + Flask, AXIS at the edge through a Cloudflare Worker, and a git-backed library are all first-class recipes—see Compose recipes.

Conceptual model

```
flowchart LR
  WL[Watchlist] --> SRC[Source]
  SRC --> CH[Chart panes]
  STR[Stream] --> CH
  ED[Editor] --> ENG[Engine]
  SRC --> ENG
  ENG --> RES[Results + overlays]
  STO[Storage] --> ED
```

The shell (topbar, chart, editor, results, manager) is **AXIS**. Calculation is always `engine.run({ script, bars, config })`. Persistence of scripts is always a storage plugin.

Interface surface

Surface	Role
Topbar	Symbol, interval, source/stream/engine pickers, Load, Run, Live
Watchlist	Quick symbol switch + quote poll
Chart	lightweight-charts panes, drawings, trade markers
Editor	CodeMirror 6 Pine document (docked / popout)
Results	Events, Strategy, Plots, Metrics, Raw + export
Manager	Plugin catalog, URL install, Script Library
Settings	Endpoint, engine, storage, interval defaults

Getting started

1. Installation — Vite dev, static `dist/`, or hosted PWA
2. Quick start — first load + first Run
3. Compose recipes — offline, desk, multi-venue, CSV, edge, git

Guides

- Manager and settings
- Script library
- Strategy and results
- Drawings
- Troubleshooting

Reference

- Glossary
- FAQ

Invariants (operator-facing)

1. **AXIS ≠ engine** — switching engine does not change the chart library; switching source does not change the language runtime.
2. **Active set is four-tuple** — source, stream, engine, storage are selected independently (with sensible defaults when source implies stream).
3. **API keys stay in the browser** — cloud storage and Pro keys are not uploaded by the shell except as request headers you configure.
4. **OHLCV bars are not persisted** — reload re-fetches history; layout, script draft, and drawings survive in `pynescript.axis.v1`.

See also

- Architecture — ADRs and topologies
- UI — chart, editor, store internals
- Plugins — contracts and catalogs
- PYNE runtime — evaluation semantics

INSTALLATION

title: "Installation" description: "Run AXIS as Vite+Solid dev, static dist PWA, or against Flask / Cloudflare Worker backends."

INSTALLATION

AXIS ships as the `frontend/` package (`axis-chart`). Primary path is **Vite + Solid**. Legacy static shell (`main.js`, root `style.css`) is not the product UI—prefer `bun run dev` or built `dist/`.

Abstract

Three operator modes:

Mode	Command surface	When
Dev AXIS	Vite `:3000`	Day-to-day UI work
Static PWA	`dist/` + `axis_pwa_server.py` `:8081`	Offline demo / VPS
AXIS at the edge	CF Pages + Worker	Production topology

Backend is optional when **Engine = Client-Side (Pyodide)** and sources/streams are mock or CSV.

Prerequisites

- **Bun** (or Node 22+) for package install and Vite
- **Python 3.10+** if you use Flask Pro API (`make run`) or `axis_pwa_server.py`
- Modern Chromium / Firefox / Safari (PWA install works best on Chromium)

Dev AXIS (recommended)

```
# Terminal 1 — Pro API (server engine)
make run              # Flask :5002

# Terminal 2 — AXIS
cd frontend
bun install
bun run dev           # Vite :3000, proxies /run → :5002 when configured
```

Open `http://localhost:3000` . Default symbol `BTCUSDT` , engine often `server` with endpoint `http://localhost:5002` (or a demo host if preconfigured).

Production build (static)

```
cd frontend
bun install
bun run build         # → frontend/dist/
python3 axis_pwa_server.py   # serves dist/ on :8081
```

Confirm:

- App shell loads; chart requests history
- DevTools → Application → Manifest + Service Worker
- Icons 192/512 from `public/assets/`

Pyodide assets and vendor wheels under `public/pyodide/` and `public/vendor/` must be present in `dist/` for offline engine— `bun run build` copies them via Vite `public/` .

Offline-first lab (no Flask)

1. Source → **Mock Walk**
2. Stream → **Mock Poll** (or **None**)
3. Engine → **Client-Side (Pyodide)**
4. Storage → **Local**

Disable network in DevTools; **Run** still executes. First Pyodide boot downloads/loads self-hosted runtime from the origin—allow that once while online, then go offline.

Cloudflare Worker endpoint

```
cd frontend/worker
npm install
npm run dev           # wrangler :8787
```

In AXIS Settings, set **Backend URL** to `http://127.0.0.1:8787` (or `/api` path as your Worker routes). Production project id is frozen as `pynescript-axis`—see topologies.

CORS when using server engine

Browser origin → Pro API must allow your AXIS origin. Flask uses `ALLOWED_ORIGINS`. Localhost regex is typically included; for a VPS demo host, set an explicit origin list. Symptom of failure: `POST /run` blocked in Network tab (no `Access-Control-Allow-Origin`).

PWA install

- Manifest: void theme `#0a0b10`, name AXIS
- Service Worker: cache-first shell; network-first `/api/*`; offline API returns structured failure so Pyodide path remains usable
- Chrome/Edge: install icon in the omnibox

Verification checklist

Check	Expect
Load	Bars on chart for default symbol
Topbar	Source / Stream / Engine pickers populated
Run (server)	Flask or Worker responds; plots overlay
Run (pyodide)	Offline OK after warm-up
Manager	Catalog lists built-ins
Theme	Dark/light toggle persists

Failure modes

Symptom	Likely cause	Fix
Empty chart	Source network / CORS / wrong symbol	Load again; try mock-walk
Engine errors immediately	Endpoint down or wrong URL	Settings → Test endpoint
Pyodide “BadZipFile” / HTML	SPA fallback instead of wheels	Ensure `public/vendor` and pyodide in `dist/`
SW stale UI	Aggressive cache	Unregister SW or hard reload

Internals (repo paths)

Path	Role
`frontend/src/index.tsx`, `app.tsx`	Solid entry
`frontend/vite.config.ts`	Build
`frontend/axis_pwa_server.py`	Static host
`frontend/manifest.webmanifest`, `sw.js`	PWA
`frontend/worker/`	CF Worker data plane

See also

- Quick start
- Compose recipes
- DevOps local dev
- Worker

QUICK START

title: "Quick start" description: "First historical load, first Pine Run, live stream, and reading the Results drawer in AXIS."

QUICK START

Fifteen-minute path from empty AXIS to plots, strategy stats, and optional live updates.

Abstract

1. Ensure AXIS is running (Installation).
2. Load bars for a symbol.
3. Paste or write a minimal indicator / strategy.
4. **Run** → inspect chart overlays + Results.

5. Optionally enable **Live**.

Conceptual model

```
sequenceDiagram
  participant U as Operator
  participant T as Topbar
  participant S as Source
  participant E as Engine
  participant C as Chart
  participant R as Results
  U->>T: Load
  T->>S: fetchHistorical
  S-->>C: bars
  U->>T: Run
  T->>E: run(script, bars)
  E-->>C: plots / markers / drawings
  E-->>R: lastRun payload
```

Interface surface

1. Load history

Topbar fields that matter:

Control	Default-ish	Notes
Symbol	`BTCUSDT`	Venue-specific for `binance-rest`
Interval	`1d`	Watchlist + chart share interval vocabulary
Source	`binance-rest`	Or mock / CSV
Stream	`binance-ws`	Paused until Live
Engine	`server` or `pyodide`	Settings for endpoint

Click **Load** (or pick a watchlist row). Status bar should move through loading → ready with bar count context.

CSV path: Source = **CSV upload** → pick file when prompted → bars inject via upload store.

2. Minimal script

Open the editor (docked right by default). Example indicator:

```
//@version=5
indicator("AXIS smoke", overlay=true)
plot(close, "close")
plot(ta.sma(close, 20), "sma20")
```

Strategy smoke (events for Results → Strategy):

```
//@version=5
strategy("AXIS strat smoke", overlay=true)
longCond = ta.crossover(ta.sma(close, 10), ta.sma(close, 30))
if longCond
    strategy.entry("L", strategy.long)
if ta.crossunder(ta.sma(close, 10), ta.sma(close, 30))
    strategy.close("L")
```

Language semantics: PYNE runtime.

3. Run

Run (topbar or editor). Pipeline:

1. Active engine `run({ script, bars, config })`
2. `lastRun` stored in memory (not fully persisted)
3. Results drawer opens
4. Chart applies overlay lines, trade markers, Pine drawings

Success: status “Completed in Nms”; plots on price pane (or dedicated indicator pane if non-overlay).

4. Read Results

Tabs:

Tab	Content
Events	Normalized entry/exit/order stream
Strategy	Closed trades + win rate, PF, max DD
Plots	Series names, point counts, last value
Metrics	Runtime, engine, bar count, errors
Raw	Full JSON payload

Export: JSON run dump; trades CSV. Click a trade to scroll the chart to entry/exit time.

5. Live (optional)

Toggle **Live**. Stream plugin pushes bars; AXIS may re-run silently when `needsRerun` is set. Stream = **None** freezes time. Mock poll works offline after engine warm-up.

6. Save a script (optional)

Manager → **Script Library** → name → Save. Backend is active storage (`local` default = IndexedDB). See Script library. Editor **Pull / Push** uses the same storage when configured for git/cloud.

7. Power tools (optional)

Tool	Where	Purpose
⌘/Ctrl+K	Global	Command palette — panels, Run, layout, editor toggles
Layouts	Topbar	Multi-chart 1 / 2H / 2V / 4 + recipes
Replay	Topbar	Scrub history (full bars, cursor on last)
Compare	Topbar	Second symbol overlay
Debug / Pins	Editor header	Line chips vs chart markers — Debugging
Layers / Alerts	Panel toggles	Drawings list; local price alerts

Worked example: offline lab

Setting	Value
Source	mock-walk
Stream	mock-poll
Engine	pyodide
Storage	local

Load → Run smoke indicator → toggle Live. No Flask required.

Worked example: desk + Flask

Setting	Value
Source	binance-rest
Stream	binance-ws
Engine	server
Endpoint	`http://localhost:5002`
Storage	local

`make run` in repo root; Vite or static AXIS as installed.

Invariants

- Empty editor **Run** is a no-op.
- Errors set status `error` and still populate Results Metrics/Raw when payload exists.
- Switching source auto-aligns default stream (e.g. mock-walk → mock-poll).

Failure modes

Symptom	Action
CORS on `/run`	Fix `ALLOWED_ORIGINS` / use Pyodide
No trades in Strategy	Need entry+exit pair events; open strategy script
Live no updates	Stream not `none`; network; check logs drawer
Pyodide first run slow	Expected warm-up; assets self-hosted under origin

See also

- Compose recipes
- Strategy and results
- Troubleshooting

COMPOSE RECIPES

title: "Compose recipes" description: "Lawful source × stream × engine × storage combinations: offline, desk, multi-venue, CSV, edge, and git library."

COMPOSE RECIPES

AXIS treats composition as the product. This page lists **recipes**—complete four-tuples with intent, constraints, and failure modes—not marketing tiers.

Abstract

Formal active set:

```
Active set = { source, stream, engine, storage }
```

Built-in ids (catalog may grow via URL plugins):

Kind	Built-ins
Source	`binance-rest`, `mock-walk`, `csv-upload`, (+ examples e.g. CoinGecko)
Stream	`binance-ws`, `mock-poll`, `none`, (+ CF DO example)
Engine	`server`, `pyodide`
Storage	`local`, `cloud`, `git`

Recipe matrix

Recipe	Source	Stream	Engine	Storage	Needs net	Needs backend
Offline lab	mock-walk	mock-poll	pyodide	local	first boot only	no
Desk research	binance-rest	binance-ws	server → Flask	local	yes	Flask
Multi-venue desk	binance-rest + URL sources	binance-ws / none	server	local	yes	Flask
CSV desk	csv-upload	none / mock-poll	server or pyodide	local	no*	optional
AXIS at the edge	venue REST	venue WS / DO	server → Worker	cloud	yes	Worker (+ optional Flask proxy)
Repo library	any	any	any	git	yes (save/list)	git host API

*CSV needs local file; engine may still call network if `server`.

Offline lab

Intent: airplane mode, demos, CI-less teaching, pure AXIS UX.

Axis	Id	Why
Source	`mock-walk`	Synthetic OHLCV in-process
Stream	`mock-poll`	Synthetic bar advance
Engine	`pyodide`	In-browser `pynescript` via Pyodide
Storage	`local`	IndexedDB script library

Setup

1. Install static or dev AXIS with `public/pyodide` + `public/vendor` present.
2. Select the four-tuple in topbar / settings.
3. Load → Run while online once (warm Pyodide).
4. DevTools → Offline; re-run.

Invariants: no `POST /run`; no exchange traffic; drawings + library remain local.

Failure modes: missing wheels → HTML SPA fallback errors; cold start timeout on huge bar sets.

Desk research

Intent: real venue history + live klines + full PYNE server fidelity.

Axis	Id	Why
Source	`binance-rest`	Public REST klines
Stream	`binance-ws`	Public WS kline stream
Engine	`server`	Flask Pro API `POST {endpoint}/run`
Storage	`local`	Fast offline library

Setup

```
make run                # :5002
cd frontend && bun run dev
```

Settings → Backend URL `http://localhost:5002` → Test. Symbol like `BTCUSDT`, interval `1h` / `1d`.

CORS: AXIS origin must be allowed by Flask `ALLOWED_ORIGINS`.

Failure modes: rate limits; symbol not on venue; CORS; engine timeout on long history (AXIS scales timeout with bar count).

Multi-venue / multi-source

Intent: compare feeds without forking the AXIS.

Axis	Pattern
Source	Built-in `binance-rest` **or** Manager → Install URL plugin (e.g. CoinGecko example) → **Use**
Stream	Match venue when available; else `none` for historical-only
Engine	Usually `server` for parity
Storage	`local` or `git` for shared research

Workflow

1. Manager → Catalog → capability badges (offline / auth / network / proxy).
2. **Use** sets active source.
3. Load and Run as usual.

Dynamic plugins rehydrate from localStorage install list on boot (`restoreInstalledPlugins`).

Failure modes: plugin URL not CORS-readable; invalid export shape; built-ins cannot be unregistered without force flags.

CSV desk

Intent: proprietary or offline CSV/OHLCV files.

Axis	Id
Source	`csv-upload`
Stream	`none` (typical) or `mock-poll` for artificial advance
Engine	`pyodide` (airgap) or `server`
Storage	`local`

Workflow

1. Source → CSV upload (file picker opens if no file yet).
2. AXIS parses via `parseOhlcvFile` into upload store.
3. Load injects bars; symbol field less critical for synthetic series.
4. Run strategies against that tape.

Invariants: bars stay in memory/session upload store—not the long-lived app state blob (size).

Failure modes: bad headers/time units; empty parse; re-select same file requires clearing input (AXIS resets file input after pick).

AXIS at the edge

Intent: browser AXIS + Cloudflare Worker data plane (keys, usage, optional D1 scripts, DO fan-out).

Axis	Id	Why
Source	venue REST (built-in or proxy-aware)	History
Stream	venue WS **or** DO-backed example stream	Single upstream → N clients
Engine	`server` endpoint = Worker	Worker proxies `/api/run` → Flask or future in-worker runtime
Storage	`cloud`	`/api/scripts` + Bearer Pro key

Setup

1. `wrangler dev` / deploy Worker project **`pynescript-axis`** (frozen name).
2. Endpoint → Worker origin.
3. Storage cloud → API key from admin `/api/keys`.
4. Script Library uses cloud list/read/write; If-Match on concurrent writes.

Invariants: CF project id is infrastructure, not brand; health JSON may say `pynescript-axis-worker`.

Failure modes: missing Bearer key; D1 unbound → memory store (dev only); DO session query params wrong.

Git library

Intent: Pine sources as first-class repo files; Save = commit.

Axis	Id
Source / Stream / Engine	any research tuple
Storage	`git`

Config (Manager → Script Library or pluginsConfig)

Field	Notes
provider	`github` | `gitlab`
token	contents:write / api
owner, repo	GitHub path; GitLab projectId optional
branch	default `main`
basePath	default `pine-library`
apiBaseUrl	self-hosted forges

Invariants: drafts stay local; only explicit Save/remove hit the remote. Commit message template supports `{{name}}` / `{{iso}}`.

Failure modes: 401 token scope; wrong base path; GitLab path vs numeric id confusion.

Choosing a recipe

```
flowchart TD
  A{Need real venue data?} -->|no| B[Offline lab or CSV]
  A -->|yes| C{Need live?}
  C -->|no| D[Source + stream none + engine]
  C -->|yes| E{Own Worker / multi-client?}
  E -->|no| F[Desk research]
  E -->|yes| G[AXIS at the edge]
  H{Share scripts in git?} -->|yes| I[Storage git]
  H -->|team cloud keys| J[Storage cloud]
  H -->|solo| K[Storage local]
```

Internals

Concern	Path
Active selection	`frontend/src/store` `activePlugins`
Built-in catalogs	`sources/catalog.ts`, `streams/*`, `engines/catalog.ts`, `storage/catalog.ts`
URL plugins	`plugins/loader.ts`
Compose UI	`ui/Topbar.tsx`, `ui/PluginManager.tsx`, `ui/SettingsDialog.tsx`

See also

- Plugin contracts
- Manager and settings
- Topologies
- ADR-001 pluggable registry

MANAGER AND SETTINGS

title: "Manager and settings" description: "Plugin Manager catalog, URL install, capability badges, and Settings dialog for engine, storage, and endpoints."

MANAGER AND SETTINGS

Two orthogonal surfaces configure AXIS: the **Plugin Manager** (what exists and is active) and **Settings** (endpoint, engine, storage, interval defaults). Neither embeds language semantics.

Abstract

Surface	Opens from	Mutates
Plugin Manager	Topbar Plugins	Registry membership, active source via Use, library ops
Settings	Topbar gear	`endpoint`, active engine/storage, chart interval, watchlist refresh

Active plugins live in `store.activePlugins`; per-plugin fields in `store.pluginsConfig` keyed by `` `${kind}:${id}` ``.

Conceptual model

```
flowchart TB
  REG[PluginRegistry]
  CAT[Catalog tab]
  INS[Install tab]
  LIB[Script Library tab]
  SET[Settings dialog]
  REG --> CAT
  INS -->|loadPluginFromUrl| REG
  LIB --> STO[Storage plugin]
  SET --> AP[activePlugins + endpoint]
  CAT -->|Use| AP
```

Plugin Manager

Catalog

Lists sources, streams, engines, storages from the unified registry.

Action	Effect
Use	`setActivePlugin(kind, id)` for that row's kind
Capability badges	`offline`, `needsAuth`, `needsNetwork`, `needsProxy` from `plugin.capabilities`
Built-in flag	Built-ins resist casual unregister

Status bar shows active engine id and storage backend after changes.

Install (URL)

Paste a module URL exporting a default plugin object (`kind`, `id`, `name`, contract methods). AXIS:

1. Fetches and validates export
2. Registers into the TypeScript registry
3. Persists install record so `restoreInstalledPlugins()` reloads next visit

Security notes (operator-level): only https/http schemes suitable for module load; treat third-party URLs as code execution in your browser origin. See security tests under `frontend/tests/security/`.

Example plugins ship in `public/plugins/`:

- `example-coingecko-source.js`
- `example-tiny-pine-engine.js`

- `example-cf-do-stream.js`

Script Library tab

See Script library. Manager hosts the panel so library and plugins share one modal.

Settings dialog

Field	Behavior
Engine	Registry engine list; labels via `engineOptionLabel`
Execution mode	When the active engine exposes `configSchema.mode` (server): `interpret` | `compile` | `auto`. Stored in `pluginsConfig.engine:<id>.mode` and sent on each run (WS and REST).
Prefer WebSocket run	Server engine: prefer `/ws/run` over REST when available
Backend URL	Shown when engine is `server` or has `configSchema.endpoint`
Test / Probe	`GET` health against endpoint; status message
Storage	`local` | `cloud` | `git`
Default interval	Updates store; may reload bars for current symbol
Watchlist refresh	Clamp 5–120 seconds

Execution mode maps to PYNE `Runtime.run(..., mode=…)`:

Mode	Behavior
`interpret`	Full AST interpreter (default)
`compile`	Numba/numpy compiled path
`auto`	Try compile; fall back to interpret on failure

The Connection HUD engine chip shows the selected mode. Save persists via Solid store `persist()` into `pynescript.axis.v1`. Escape closes; Ctrl/Cmd+Enter saves.

Pyodide engines hide endpoint—calculation is same-origin assets, not Flask.

Interface surface (UI map)

Control	Store field
Engine select	`activePlugins.engine`, flat `engine`
Storage select	`activePlugins.storage`
Endpoint	`endpoint`
Source/stream topbar	`activePlugins.*`, `source`, `live.streamId`
Theme toggle	`theme` + `data-theme` on `<html>`

Internals

Path	Role
`frontend/src/ui/PluginManager.tsx`	Manager modal
`frontend/src/ui/SettingsDialog.tsx`	Settings
`frontend/src/ui/plugin-badges.tsx`	Capability badges
`frontend/src/plugins/registry.ts`	Unified registry
`frontend/src/plugins/loader.ts`	URL load + restore
`frontend/src/plugins/bootstrap.ts`	Built-ins

Invariants

1. Settings never write script bodies—only layout/config.
2. Catalog **Use** does not auto-Run; Load/Run remain explicit.
3. Source change may re-default stream (`defaultStreamForSource`).
4. API keys for cloud/git belong in `pluginsConfig` , not in git-committed defaults.

Worked examples

Point AXIS at local Worker

1. Settings → Engine Server-Side → Endpoint `http://127.0.0.1:8787`
2. Probe → OK
3. Save → Run

Install example source

1. Manager → Install → URL to `.../plugins/example-coingecko-source.js` (served origin)
2. Catalog → Source appears → Use → Load

Failure modes

Symptom	Fix
Empty catalog	Built-ins not registered—hard reload; check console
URL install fails	CORS on plugin host; bad export; mixed content
Probe fails	Backend down; wrong path; HTTPS mixed content
Settings not sticky	localStorage blocked / quota

See also

- Script library
- Compose recipes
- Plugins

SCRIPT LIBRARY

title: "Script library" description: "Save, load, import/export Pine scripts via local IndexedDB, cloud Worker, or git (GitHub/GitLab)."

SCRIPT LIBRARY

The script library is the operator-facing surface of **storage plugins**. The editor holds the working document; the library is durable, named documents with metadata.

Abstract

Backend	Id	Medium	Auth
Local	`local`	IndexedDB (`pynescript.axis.storage`), LS fallback	none
Cloud	`cloud`	Worker `/api/scripts` (D1 or memory)	Bearer Pro API key
Git	`git`	GitHub/GitLab Contents API	PAT

Active backend: `store.activePlugins.storage` (Settings or library picker).

Conceptual model

```
flowchart LR
  ED[Editor doc]
  SVC[storage/service.ts]
  P[Active StoragePlugin]
  ED -->|Save| SVC
  SVC --> P
  P -->|list/read/write/remove| MED[IDB / Worker / Forge]
  SVC -->|export/import JSON| FILE[File]
```

Service layer (`listScripts` , `readScript` , `writeScript` , …) never talks to engines.

Interface surface

Manager → **Script Library**:

Control	Behavior
Backend select	`setActivePlugin('storage', id)`
Refresh	`list()`
Name / description	Metadata for next write
Save	Write current editor doc
Load	`read` → inject editor (`loadLibraryDoc` / setDoc)
Delete	`remove`
Export JSON	Full library dump
Import JSON	Bulk write
Cloud endpoint + key	Saved under `pluginsConfig['storage:cloud']`
Git fields	provider, token, owner, repo, branch, basePath, …

Status line: backend · remote · branch · connected · count.

Local backend

- DB name `pynescript.axis.storage` v1 stores `scripts` + `kv`
- Older library keys migrate once
- Drafts: optional `saveDraft` / `loadDraft` without polluting remote backends

- Works fully offline

Invariant: browser profile wipe deletes local library—export JSON for backup.

Cloud backend

- `Authorization: Bearer <api_key>`
- Scripts partitioned by key hash on Worker
- Concurrent updates may use `If-Match` / revision → HTTP 409 on conflict
- Default endpoint falls back to `store.endpoint` or `http://127.0.0.1:8787`

Configure key in library panel or Settings-related config; keys stay in this browser's localStorage state blob.

Git backend

Invariant	Detail
Save commits	Each write/remove is a commit (+ push via API)
Drafts local	No remote commit for draft buffer
basePath	Default `pine-library/`
Providers	`github`, `gitlab` (+ self-hosted `apiBaseUrl`)

Token scopes: GitHub `contents:write`; GitLab `api` or `write_repository`.

Worked examples

Solo offline library

1. Storage = local
2. Write indicator in editor
3. Name `sma-ribbon` → Save
4. Reload page → Library → Load

Team cloud

1. Admin creates key on Worker
2. Storage = cloud; paste endpoint + key
3. Save shared strategies
4. Second browser: same key → list

Research monorepo

1. Storage = git; GitHub owner/repo; basePath `research/pine`
2. Save → commit message from template
3. PR review on GitHub UI

Internals

Path	Role
`frontend/src/ui/ScriptLibraryPanel.tsx`	UI
`frontend/src/storage/service.ts`	Facade
`frontend/src/storage/local.ts`	IDB
`frontend/src/storage/cloud.ts`	Worker API
`frontend/src/storage/git.ts`	Git plugin
`frontend/src/storage/git-github.ts`, `git-gitlab.ts`	Forges

Contract: Plugin types `StoragePlugin`.

Failure modes

Symptom	Cause	Mitigation
Empty list after switch	Wrong backend / auth	Check status line + key
401 cloud	Bad/missing key	Regenerate; re-save config
409 cloud	Revision conflict	Re-read then write
Git 404	Wrong owner/repo/basePath	Fix config
Quota local	Huge scripts in LS fallback	Prefer IDB; export prune

See also

- Manager and settings
- Compose recipes — git / edge
- State namespaces

STRATEGY AND RESULTS

title: "Strategy and results" description: "How AXIS turns engine events into trades, stats, chart markers, equity, and exportable Results tabs."

STRATEGY AND RESULTS

After **Run**, the engine returns a structured payload. AXIS **AXIS** normalizes events, pairs trades, paints markers, and exposes a Results drawer. Strategy math here is a **viewer**—not a brokerage.

Abstract

Engine `RunResult` (simplified):

```
status: success | error
plots[] | series{}
events[]     — entries, exits, orders, custom
drawings[]   — Pine line/label/box objects
meta{}       — ms, overlay, script_name, plot_meta, …
error?
```

AXIS stores this as `store.lastRun` (in-memory; not fully rehydrated from localStorage).

Conceptual model

```
flowchart TB
  ENG[Engine.run] --> LR[lastRun]
  LR --> NORM[normalizeStrategyEvents]
  NORM --> REP[buildStrategyReport]
  NORM --> MARK[eventsToMarkers]
  REP --> UI[Results tabs]
  MARK --> CH[Chart markers]
  LR --> PLOTS[Overlay series]
  LR --> EQ[Equity curve pane]
  LR --> EXP[JSON / CSV export]
```

Results drawer tabs

Tab	Operator value
Events	Chronological normalized stream (kind, time, price, id, dir)
Strategy	Closed trades table + summary metrics
Plots	Series inventory (points, last value)
Metrics	Runtime, status, engine, source, bars, high-level stats
Raw	Pretty-printed JSON for support / diffs

Strategy metrics

Computed by `buildStrategyReport`:

Stat	Definition (viewer)
totalPnl	Sum of closed trade PnL (price units)
winRate	% wins among closed trades
profitFactor	Gross profit / gross loss
avgTrade / avgWin / avgLoss	Means over closed set
maxDD	Peak-to-trough on cumulative trade PnL path
wins / losses / trades	Counts

Pairing rules: entry-like kinds open by `id`; exit/close kinds match id or sole open trade. Direction flips short PnL. Missing prices drop the event from pairing (with normalization attempting OHLC fill from bars when available).

Chart coupling

- Trade markers on price pane (`eventsToMarkers`)
- Scroll-to-time on trade row click
- Equity curve series when strategy events warrant (`buildEquityCurve`)
- Status bar snippet: closed trade count · net PnL when available

Export

Export	Content
JSON	Full `lastRun`
CSV	Closed trades (`tradesToCsv`)
Clipboard	CSV of trades when supported

Filenames: `axis-run-<ts>.json` , `axis-trades-<ts>.csv` .

Interface surface

- Drawer height persisted (`resultsPanel.height`); open state toggled after runs (`openResults` default true for interactive runs)
- Silent live re-runs can skip auto-open
- Equity / indicator panes created on demand when non-overlay or strategy paths need them

Worked example

1. Load `BTCUSDT` daily history.
2. Run a crossover strategy (see Quick start).
3. Results → Strategy: inspect win rate.
4. Click a green trade → chart jumps to entry.
5. Export CSV for a notebook.

Internals

Path	Role
`frontend/src/ui/ResultsPanel.tsx`	Drawer UI
`frontend/src/results/strategy.ts`	Trade pairing + stats
`frontend/src/results/events.ts`	Normalize events, markers, equity
`frontend/src/indicators/runner.ts`	`runAndApply` orchestration
`frontend/src/chart/series-factory.ts`	Plot colors / series

Language-level strategy semantics: PYNE runtime.

Invariants

1. Results are a function of **last successful/error payload + current bars**—changing bars without re-run can desync prices used for fill.
2. Viewer PnL is not currency-normalized; treat as relative tape units.
3. `meta.overlay === false` routes plots to an indicator pane, not price.
4. Pine drawings (script) are distinct from user drawing tools (Drawings).

Failure modes

Symptom	Explanation
Events but 0 trades	Only entries, or unpaired ids
Empty plots	Series all null / name filtered (`__` prefix hidden)
Markers missing	Manager not mounted or events lack times
Huge Raw JSON	Many events—export file rather than copy

See also

- Quick start
- AXIS — results and strategy
- AXIS — indicators

DRAWINGS

title: "Drawings" description: "Interactive chart annotations in AXIS—tools, persistence, and how they differ from Pine script drawings."

DRAWINGS

AXIS supports **user drawings** (interactive tools on the chart) and **script drawings** (objects returned by the engine). They share a canvas layer but different lifecycles.

Abstract

Class	Origin	Cleared when	Persisted
User drawings	Drawing toolbar	Explicit delete / clear	Yes (`store.drawings` in `pynescript.axis.v1`)
Script drawings	Engine `drawings[]`	Next Run (re-applied)	No (recomputed)

Conceptual model

```
flowchart TB
  TB[DrawingToolbar] --> ST[store.drawingTool]
  ST --> DL[DrawingLayer]
  U[Pointer gestures] --> DL
  DL --> SD[store.drawings]
  ENG[Engine drawings] --> DL
  RUN[runAndApply] -->|clearScriptDrawings| DL
```

Tools

Tool id	Kind	Geometry
`cursor`	—	Select / pan mode (not a drawable)
`hline`	hline	Horizontal price
`vline`	vline	Vertical time
`trend`	trend	Two points
`ray`	ray	Two points (extends)
`extend`	extend	Extended line
`rect`	rect	Two corners
`ellipse`	ellipse	Ellipse / oval
`arrow`	arrow	Directional segment
`fib`	fib	Two points; levels 0, 0.236, 0.382, 0.5, 0.618, 0.786, 1
`measure`	measure	Two points (delta readout)
`text`	text	Anchor + label

Colors: default accent `#939fff` , up/down/measure variants in `DRAWING_COLORS` .

Layers & templates

- **Layers** panel lists user drawings (select, hide, delete)
- **Templates**: save/load drawing packs (`axis.drawingTemplates.v1`)
- **Duplicate / tag symbol** helpers for multi-chart workflows

Interface surface

- Toolbar on chart host (`DrawingToolbar`)

- Active tool in store; layer sync via `createEffect` on `store.drawingTool`
- Pane manager hosts price chart; drawings bind to time/price coordinates (not pixel-only)

Workflows

Annotate a level

1. Select Horizontal line.
2. Click price.
3. Reload page—line remains (persisted drawings array).

Fibonacci between swing points

1. Select Fib.
2. Click swing low → swing high (or reverse).
3. Levels render from `FIB_LEVELS`.

After Run with Pine lines/labels

1. Run script that emits drawings.
2. Layer clears previous script drawings, then applies new set.
3. User drawings remain.

Internals

Path	Role
`frontend/src/chart/drawing-types.ts`	Tool ids, types, fib levels
`frontend/src/chart/drawing-layer.ts`	Interaction + render
`frontend/src/chart/DrawingToolbar.tsx`	UI
`frontend/src/chart/pine-drawings.ts`	Map engine drawings → layer
`frontend/src/chart/ChartHost.tsx`	Mount / dispose

Invariants

1. User drawings ≠ strategy results; deleting drawings does not alter `lastRun`.
2. Time base is bar time (unix-compatible chart times).
3. `cursor` never creates geometry.
4. Persistence omits bars/logs but **includes** drawings—large freehand-less sets only.

Failure modes

Symptom	Fix
Clicks do nothing	Tool is cursor; chart empty (no bars/time scale)
Drawings vanish on Run	Those were script drawings—re-run or convert notes to user tools
Lost after wipe	Cleared site data / different browser profile

See also

- AXIS — charting
- Strategy and results

TROUBLESHOOTING

title: "Troubleshooting" description: "Diagnose empty charts, CORS, engine failures, Pyodide assets, streams, storage, and PWA cache issues in AXIS."

TROUBLESHOOTING

Systematic triage for AXIS. Prefer logs drawer + Network tab + status bar before reinstalling.

Abstract

Failure domains:

1. **Data plane** — source / stream
2. **Calc plane** — engine / endpoint / Pyodide assets
3. **Shell plane** — store, SW cache, plugins

4. **Storage plane** — library backends

Decision tree

```
flowchart TD
  A[Problem] --> B{Chart empty?}
  B -->|yes| C[Source / Load / CORS / symbol]
  B -->|no| D{Run fails?}
  D -->|yes| E[Engine endpoint / Pyodide / script error]
  D -->|no| F{Live dead?}
  F -->|yes| G[Stream id / WS / none]
  F -->|no| H{Library fail?}
  H -->|yes| I[Storage auth / IDB]
  H -->|no| J[UI / SW / plugin]
```

Data plane

Symptom	Checks	Fix
Infinite loading	Network klines; status message	Correct symbol; try `mock-walk`
CORS on REST	Binance/public APIs usually OK; custom sources may fail	Proxy plugin or Worker
CSV no bars	Parse error in status	Fix columns/time; re-upload
Wrong interval bars	Store interval vs request	Settings interval + Load

Calc plane — server engine

Symptom	Checks	Fix
Failed fetch `/run`	Endpoint, CORS preflight	`ALLOWED_ORIGINS`; Settings probe
HTTP 4xx/5xx	Response JSON `message`	Fix script; backend logs
Timeout	Bar count × complexity	Shorter history; raise server resources
Plots missing	`status: success` but empty series	Script has no plots; check Raw

Probe: Settings → Test endpoint (health GET).

Calc plane — Pyodide

Symptom	Checks	Fix
HTML instead of wheel	Content-Type / ZIP magic	Deploy `public/vendor`, `public/pyodide` into `dist/`
Slow first run	Cold runtime	Wait; `preloadPyodide` on idle
Offline fail first visit	Assets never cached	One online warm-up
Interpreter error	Results Raw / logs	Language issue → PYNE

Live streams

Symptom	Checks	Fix
Live toggle no effect	Stream = `none`?	Select `binance-ws` or `mock-poll`
WS errors	Console; venue symbol format	Uppercase `BTCUSDT` style
Reconnect loop	Network flap	Logs drawer; stop/start Live
DO stream	Worker session query	Worker docs; example plugin

Storage / library

Symptom	Checks	Fix
Save no-op	Active storage; errors in panel	Read status line
Cloud 401	Bearer key	Settings/library key field
Git denied	Token scopes	PAT permissions
Lost local scripts	Cleared site data	Import JSON backup

Chart / editor (recent surfaces)

Symptom	Checks	Fix
Indicator blank / missing on chart	Oscillator on price scale; wrong pane id	Prefer `overlay=false`; re-run; see Indicators
Editor no lines / zero height	Dock height chain	Detach/reattach or toggle panel; editor should fill right strip
Editor covers Indicators	Side-by-side dock	Both open on right → Indicators left of Editor; resize each panel
Replay shows one candle only	Session start	Replay starts at **last bar** with full history; Play at end restarts from bar 0
No Debug chips / Pins	Toggles + log shape	Enable **Debug** / **Pins**; logs need `line` and/or `bar_index` — Debugging
Completions thin	Builtins corpus	Re-sync `pine-builtins.json` from PYNE; optional remote LSP

Shell / PWA

Symptom	Checks	Fix
Old UI after deploy	Service Worker	Unregister SW; hard reload
State weird	`localStorage['pynescript.axis.v1']`	Export scripts; clear key; reload
Plugin vanished	URL install list	Re-install URL
Theme wrong	`data-theme`	Toggle theme once
Lost multi-chart slots	`chartLayout` in store	Named layouts menu; re-apply recipe

Logging

- **System logs** drawer: boot, pyodide ready, live re-run errors (`appendLog`)
- **Status bar**: ready / loading / running / error + short message
- **Results** → **Raw**: last engine payload

Security-related

- Poisoned localStorage: app should tolerate parse failures (see security tests)
- Plugin URL schemes restricted
- Never paste production PATs into shared screen recordings

Internals for deeper digs

Area	Path
Load history	`data/load-symbol.ts`
Run pipeline	`indicators/runner.ts`
Streams	`streams/multiplex.ts`
Engines	`engines/catalog.ts`
Store	`store/index.ts`

See also

- Installation
- FAQ
- DevOps
- Worker

GLOSSARY

title: "Glossary" description: "AXIS end-user glossary: axes, AXIS, plugins, engines, storage keys, and result terms."

GLOSSARY

Terms as used in AXIS documentation and UI. Language runtime terms → PYNE.

Product

Term	Definition
AXIS	Charting PWA product : chart, editor, panels, and store—not the language engine.
UI track	Docs track for presentation surfaces (chart host, editor, shell, store) under `/axis/docs/ui`.
PYNE	Pine Script™ toolchain (parse/eval); engines embed or call it.
Active set	Tuple `{ source, stream, engine, storage }` currently selected.
Compose / recipe	A purposeful active set for a workflow (offline lab, desk, edge, …).

Plugin kinds

Term	Definition
Source	Historical OHLCV provider (`fetchHistorical`).
Stream	Live bar/tick push (`start` → dispose).
Engine	Calculator (`isReady` , `run`).
Storage	Script library backend (`list` / `read` / `write` / `remove`).
Component	Reserved UI slot plugins (phase 2).
Registry	In-memory catalog of plugins by kind (`PluginRegistry`).
configSchema	Declarative field schema for Settings / plugin config UI.
Capabilities	Flags: offline, needsAuth, needsNetwork, needsProxy.

Engines & backends

Term	Definition
server engine	`POST {endpoint}/run` with script + bars.
pyodide engine	In-browser Python + pynescript wheels.
Pro API / Flask	Local Python backend (`make run` , typically `:5002`).
Worker	Cloudflare Worker data plane for AXIS.
Endpoint	Base URL the server engine (and often cloud storage) calls.

Data & chart

Term	Definition
Bar	`{ time, open, high, low, close, volume? }`
Pane	Chart strip: price, volume, indicator, equity.
Overlay	Plot drawn on price pane (`meta.overlay !== false`).
Sub-pane	Non-overlay indicator strip; stable store id `indicator` .
User drawing	Interactive annotation tool geometry.
Script drawing	Engine-emitted line/label/box objects.
Marker	Trade/event marker on series.
Pane badge	Corner chip with script actions (settings / eye / re-run / remove).
Multi-chart	Grid layouts (1 / 2H / 2V / 4) with per-slot symbol/interval.
Bar Replay	Scrub/play over loaded OHLCV without live stream.
Compare	Second-symbol series overlaid on the price pane.
Volume profile	OHLCV volume-at-price histogram (Layers).
Watchlist	Side panel of symbols + quote refresh.
Layers	Panel listing panes, indicators, and user drawings.
Alert	Local price condition (+ optional webhook); Alerts panel.

Editor & debug

Term	Definition
Debug	Inline end-of-line chips from last-run logs with line refs.
Pins	Chart markers + 📍 gutter for logs with bar_index/time.
Problems	Clickable list of engine diagnostics for the last run.
Ruler	80-character column guide in the Pine editor.
Command palette	⌘/Ctrl+K jump menu for panels, Run, toggles, git.
Git bar	Pull/Push via active storage plugin (local/cloud/git).

Results

Term	Definition
lastRun	In-memory last engine payload.
Event	Strategy/order/plot-adjacent event from engine.
Closed trade	Paired entry/exit with PnL in the viewer.
Equity curve	Cumulative PnL series derived from trades/events.
Scriptlogs	Floatable panel of Pine `log.*` from last run (not System Logs).

Persistence

Term	Definition
`pynescript.axis.v1`	Primary localStorage app state key.
`pynescript.axis.v2`	Older app-state key; migrated on read.
pluginsConfig	Per-plugin configuration map.
Library export	JSON dump of storage documents.

Infrastructure names

Term	Definition
`pynescript-axis`	Frozen CF Wrangler/Pages project id.
`pynescript-axis-worker`	Health JSON service identity.
void	Brand pack / dark chrome aesthetic for AXIS docs/UI.

See also

- FAQ
- State namespaces
- Architecture overview

FAQ

title: "FAQ" description: "Frequently asked questions about AXIS, engines, offline use, TradingView relation, and data privacy."

FAQ

Product scope

Is AXIS TradingView?

No. AXIS is an independent charting PWA. Pine Script™ and TradingView® are trademarks of TradingView, Inc. This project is not affiliated with or endorsed by TradingView, Inc. Trademark notice appears on the AXIS docs home.

Does AXIS implement Pine Script?

Evaluation is performed by **engines** that call **PYNE** (or proxies that do). The AXIS does not embed a closed interpreter. See PYNE for language fidelity.

AXIS vs engine?

AXIS = UI + orchestration. Engine = `run(script, bars)`. You can swap engines without rewriting the chart host. That is **ADR-002** territory—ADRs.

Running & offline

Can I use AXIS fully offline?

Yes, with recipe **Offline lab**: `mock-walk` + `mock-poll` + `pyodide` + `local`, after one warm-up so Pyodide assets cache. Server engine and live venue streams require network.

Why is first Pyodide run slow?

Bootstraps Python + wheels in-browser. Subsequent runs reuse the runtime when still warm. Assets are self-hosted under the PWA origin when deployed correctly.

Default endpoint points at a VPS—is that required?

No. Change Settings → Backend URL to `http://localhost:5002` or your Worker. Defaults may reference a public demo API host for convenience.

Data & privacy

Where do API keys live?

In **this browser's** localStorage state / `pluginsConfig`. They are sent as HTTP headers only to endpoints you configure (cloud storage, etc.). They are not part of the Pine script body.

Are OHLCV bars uploaded?

Server engine sends script + bars to the configured endpoint for that run. **Pyodide** keeps bars local. Choose engine accordingly.

Are bars saved on disk?

App state persistence **omits** `bars`, `lastRun`, and `logs` to control size. History is re-fetched on Load.

Features

How do I share a setup?

Legacy hash state (`state-hash.js`) can encode symbol/interval/engine/source/stream (and short scripts). Prefer documenting the four-tuple + library export for durable sharing. URL-load plugins are re-fetched by install list.

Can I use my own data vendor?

Yes—implement a **source** plugin (and optional stream) and install via URL or built-in registration. See Plugins.

Why Strategy tab shows no trades?

Need paired entry/exit-style events with prices. Indicators that only `plot` will fill Plots/Metrics, not Strategy stats.

Drawings disappeared after Run

Script drawings are cleared and re-applied each run. User toolbar drawings persist across runs.

Popout editor out of sync?

Editor bridge uses shared document storage + message bridge between windows. Use **Reattach** on the main AXIS if the popout closed uncleanly.

What is Debug vs Pins?

	Debug	Pins
Shows	End-of-line chips in the editor	Markers on the chart + 📍 gutter
Needs	Line reference in logs/errors	`bar_index` and/or bar time
Toggle	Editor header **Debug**	**Pins** or **Alt+P**

Full workflow: Debugging.

Multi-chart, Compare, Replay, Alerts?

- **Layouts** topbar menu: 1 / 2H / 2V / 4 + named recipes
- **Compare**: second symbol overlay (% or absolute)
- **Replay**: scrub loaded history (starts with full history at last bar)
- **Alerts**: local price alerts panel (webhook optional)
- **⌘/Ctrl+K**: command palette (panels, Run, editor toggles, git)

UI detail: UI shell, Charting.

Why are Indicators next to the Editor?

Left/right docks with multiple open panels lay out **side-by-side** (row), so Indicators can sit left of Editor on the right strip. Bottom dock still stacks.

Ops

What is `pynescript-axis`?

Frozen Cloudflare project name. Renaming breaks bindings and CI. Brand is AXIS; infrastructure id stays.

Legacy localStorage keys?

Migrated automatically into `pynescript.axis.v1` and library keys. See State namespaces.

Tests for the PWA?

```
cd frontend && bun run test:unit
bun run test:e2e:smoke
```

See `frontend/TESTING.md`.

See also

- Troubleshooting
- Glossary
- Compose recipes