OPEN DATA · ARTIFICIAL INTELLIGENCE · PUBLIC INFRASTRUCTURE

TECHNICAL REPORT · TD-TR-2026-02

ins-tunisia-mcp

A local MCP server for conversational access to Tunisia's national statistics.

AUTHOR

Tarek Gasmi

PUBLISHED

June 2026

STATUS

Independently produced and published

TYPE

Technical report

ABSTRACT

ins-tunisia-mcp is a Model Context Protocol server that makes Tunisia's national statistics portal (dataportal.ins.tn) directly queryable by LLM clients running locally. The portal is an XML-in/XML-out service on the proprietary Prognoz platform; the server's defining job is to own that XML entirely, so its seven tools take structured parameters and return clean JSON — the calling model never sees an angle bracket. A single `GetStructure` call seeds an on-disk SQLite catalogue of all 78 dimensional cubes, and a background probe flags those that publish structure but serve no data. This report documents the system as it ships today — a source-distributed, local stdio server — together with the rationale behind its principal design decisions and the limitations of the running system.

VERSION

v0.1.0

tanitdata — autonomous studio · Tunis, Tunisia

tanitdata.org · contact@tanitdata.org

1. Introduction

Tunisia's **Institut National de la Statistique (INS)** publishes official statistics through a data portal at `dataportal.ins.tn`, built on the proprietary **Prognoz** platform. The portal holds dimensional time-series cubes — national accounts, the 2014 census, population projections, employment, household living conditions, foreign trade, and more. The data is public and governed by the INS *conditions d'utilisation*. In principle it is open; in practice it is awkward to reach.

The friction is structural. The portal speaks **XML in and XML out**: a client POSTs a `QueryMessage` document to operations under a Web API and receives XML back. There is no JSON, no REST convenience layer, and the request envelopes are unforgiving. Worse, the metadata is **dirty in characteristic ways**: dimension and element names carry trailing tabs, embedded markup, typographic noise, outright typos (`Superificie`, `Mileu`), and French/English drift (`Value` versus `Valeur`) — so any code that keys on human-readable labels breaks. And several distinctions that look obvious are traps: a cube's declared period can be a *projection horizon* (e.g. 2050) rather than the range for which data actually exists, and some cubes publish a complete dimensional structure while serving no values at all. The combined effect is that the portal is reachable by a determined developer who will write XML and learn Prognoz's quirks, but effectively closed to the economist, journalist, or researcher who is its natural user.

ins-tunisia-mcp addresses this by making the portal queryable through an AI assistant. It is a server implementing the **Model Context Protocol (MCP)**, the open standard by which LLM clients call external tools. A user runs ins-tunisia-mcp locally, connects it to an MCP-compatible client — Claude Desktop, Cursor, VS Code, or any other — and asks questions in natural language; the model selects and chains the server's tools to discover cubes, resolve the right dimension codes, and return cited figures. The software is published as the repository `ins-tunisia-mcp`, distributed as the `ins-mcp` Python package and run as `python -m ins_mcp`; this report refers to the software as ins-tunisia-mcp throughout, and to **tanitdata** as the publishing studio.

Distribution model and audience. This report documents ins-tunisia-mcp as it actually ships today: as **source code**, distributed for local use. A user clones the repository, installs it into a virtual environment, and launches the server as a local **stdio** process that their MCP client spawns. There is no hosted endpoint, no managed service, and no network listener — the server communicates only over the standard input/output of the process its client started. This is a deliberately modest distribution model, and it bounds the audience honestly: ins-tunisia-mcp in its current form serves a **technically capable user** comfortable cloning a repository and editing a client configuration file. It removes the need to write XML or know Prognoz, but it does not yet remove the need to run a local server. We return to this in Future Work.

The contribution here is not the protocol, which is a standard, nor the idea of wrapping an API, which is routine. It is the **façade**: the specific engineering that turns a dirty, trap-laden XML service into a small set of tools an LLM can call correctly, returning clean JSON with mandatory source attribution and honest signals about what data is and is not actually available.

2. Background

2.1 The Model Context Protocol

The Model Context Protocol is an open standard for connecting LLM applications to external tools and data sources. A client (the LLM application) launches or connects to one or more servers, each advertising a set of *tools* — named, typed, documented functions the model may call. The protocol is transport-agnostic; the transport relevant here is **stdio**, where the client spawns the server as a subprocess and communicates over standard input/output. ins-tunisia-mcp runs over stdio only.

MCP is treated here as settled infrastructure, not a research contribution. What matters for this report is the boundary it gives us: ins-tunisia-mcp's job is to present a small, well-documented set of tools whose descriptions and return values are good enough that an LLM uses them correctly without bespoke prompting on the client side.

2.2 The INS / Prognoz data portal

The portal exposes a Web API under `dataportal.ins.tn/WebApi/` with four operations, all invoked by POSTing an XML `QueryMessage` body: `GetStructure` (the catalogue of cubes and their dimensions), `GetDimensionElements` (the codelist for one dimension), `GetData` (observations for a selection), and `GetDataBorders` (the real year-range available for a selection). The endpoint is **plain HTTP**, requires **no authentication key**, and returns XML for every call.

Several portal characteristics shape the system and recur throughout this report:

- **A single database.** One empty `GetStructure` call returns the entire catalogue — **78 sources** (cubes) — in a single response; a `SourceId` on `GetStructure` is ignored. There is no database tier above "source", and the documentation-template placeholder `C_FOREIGNTRADE` from Prognoz examples does not exist on this instance (foreign trade is an ordinary cube).
- **Typed dimensions.** Every cube's dimensions carry a `DimensionType` of `Indicator`, `Region`, or `Custom`. Across the catalogue this vocabulary is exhaustive (no other type occurs), and every source has **exactly one** `Indicator` dimension — the measure axis. 48 of the 78 sources carry a `Region` dimension; 30 do not.
- **Identity lives in codes, not names.** Dimensions are addressed by a string `Id`, elements by a numeric `KEY`. The human-readable names are dirty (tabs, typos, markup, language drift) and must never be used as identifiers.
- **Declared period is not availability.** A cube's declared `Period` can be a projection horizon; the real coverage of a selection comes only from `GetDataBorders` or from the data actually returned.
- **Mixed grains; soft errors.** The catalogue mixes annual and quarterly sources, and the API answers HTTP 200 even on failure — the authoritative status is an attribute on the

XML root, not the HTTP code.

3. System Architecture

ins-tunisia-mcp is a Python package (`ins_mcp`) built on the official MCP SDK's `FastMCP` framework, with three runtime dependencies: `mcp[cli]`, `httpx` for transport, and `lxml` for XML parsing. At its centre is a strict separation: a transport module is the only place HTTP happens, a build/parse pair is the only place XML is constructed or read, and an on-disk catalogue cache backs the seven tools. Figure 1 shows the structure.

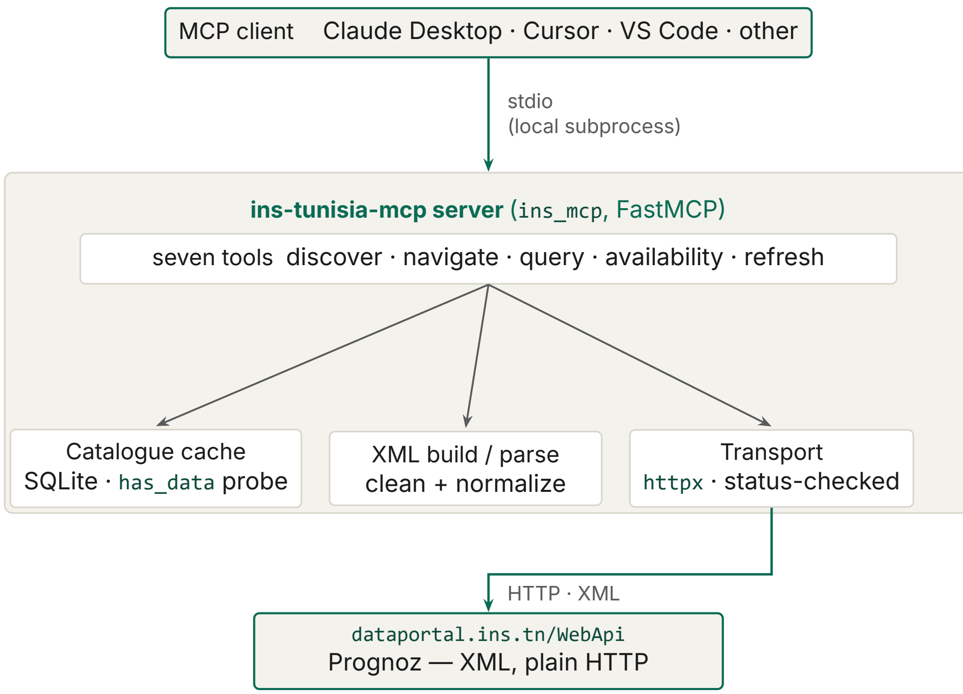

Figure 1. System architecture. An MCP client spawns the local ins-tunisia-mcp server over stdio. Seven tools resolve through an on-disk SQLite catalogue cache, a single XML build/parse layer, and a single status-checked transport against the live Prognoz portal. The server owns all XML; tools return JSON only.

3.1 The tool suite

ins-tunisia-mcp exposes **seven tools**, organized as a discovery and querying path over the portal's cubes. They are uniform in spirit — all generic, none hard-coded to a specific cube's schema — because the portal's own `DimensionType` metadata already provides the routing a per-domain tool would otherwise encode. Table 1 summarizes them.

Tool	Role	Purpose
`ins_list_sources`	discovery	List all cubes with dimensions, declared period, and `has_data`.
`ins_search_sources`	discovery	Accent/case-insensitive substring search over source titles and descriptions.
`ins_get_source`	discovery	Full detail for one cube: typed dimensions, declared period, `has_data`.
`ins_get_dimension_elements`	navigation	Browse (drill-down) or keyword-search a dimension's codelist to find element `KEY`s.
`ins_get_data`	query	Tidy observations for a selection; each carries period, frequency, value, unit, scale, attribution.
`ins_get_availability`	query	Real start/finish years for a specific selection, via `GetDataBorders`.
`ins_refresh_catalogue`	maintenance	Re-pull the catalogue from the live API.

Table 1. The seven tools. All are generic over the catalogue; the portal's `DimensionType` metadata supplies domain routing, so no per-cube tools are needed.

The discovery path. A user's question typically names a *measure* ("superficie agricole utilisée") and a *place* or *period*. The intended flow is: `ins_search_sources` to find the broad cube by theme, then `ins_get_dimension_elements` with a `search=` term to resolve the indicator's numeric `KEY` inside that cube, and likewise for a region. Codelists are hierarchical and large (the shared regions codelist runs to several thousand nodes across six levels), so this tool never dumps the whole tree: it either drills one level by `parent_key` or returns substring matches across all levels. Search is **accent- and case-insensitive** but remains substring, not synonym-aware.

The query path. `ins_get_data` is the workhorse. It validates the selection before any network call, builds the `GetData` envelope, and returns observations as flat records — one per cell — each carrying its own period, frequency, value, measurement unit, scale, and the dimension `KEY`s that produced it, plus a coverage summary and a mandatory attribution block (Section 3.4). `ins_get_availability` answers the distinct question "which years actually exist for this selection?" via `GetDataBorders`, returning real start/finish years rather than the cube's declared horizon.

Maintenance. `ins_refresh_catalogue` re-pulls the catalogue from the live portal on demand, returning the source count. The catalogue is otherwise seeded once on first use and reused.

3.2 The catalogue cache and the `has_data` probe

Tools need to know what cubes exist, what dimensions they carry, and which serve data — without paying a network round-trip on every call. An on-disk **SQLite** cache provides this. It is versioned through a `meta` table; when the on-disk shape predates the running code, the cache **auto-heals** — it drops and recreates its tables from the current schema and re-seeds on next use, and deserialization tolerantly drops unknown fields, so a stale cache can never crash a read tool.

Fast seed. Seeding is one `GetStructure` call (a few seconds): the 78 sources and their dimensions are written, and codelists are fetched lazily on first use and cached per dimension. The shared national regions codelist, reused across most cubes, is fetched once.

The empty-cube problem and the `has_data` flag. Some cubes on the portal publish a complete dimensional structure but serve **no values** — notably a family of 2014-census household cross-tabs (housing connections, equipment, water, energy, ICT). Without a signal, an AI agent drills through all of a cube's dimensions before discovering the emptiness. ins-tunisia-mcp surfaces this up front as a per-source boolean `has_data`, computed by a root `GetDataBorders` probe. Across the catalogue, **10 of the 78 sources** are flagged empty — exactly the census household cross-tab family.

The probe is governed by a strict **fail-open** invariant: `has_data` is `false` *only* on a clean probe that returns the portal's empty-range sentinel; any error, timeout, ambiguity, or not-yet-probed state yields `true`. A wrong `false` would hide real data, which is worse than the honest default. To keep this off the request path, the probe runs as a **bounded-concurrency background pass** (eight workers) after the fast seed: the first tool call is never blocked on probing, and until a source's probe completes its flag reads `true` (provisional). The probe selects all members of the measure dimension rather than a specific node, which avoids both an expensive codelist download and a known false-flagging pitfall (aggregate nodes return the empty sentinel even on populated cubes). Figure 2 shows how the flag and the two period concepts layer into three distinct, honest signals.

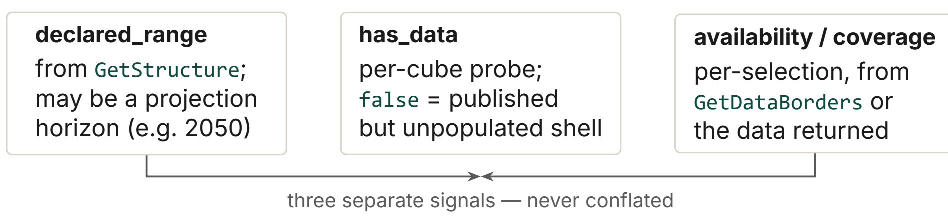

Figure 2. Three honest availability signals. The cube's *declared* period (which may be a horizon), the per-cube `has_data` flag (does it serve any values at all), and the per-selection real *coverage* are distinct and reported separately; none stands in for another.

3.3 The XML façade

The architectural principle is that **the server owns all the XML**. The calling model never sees a `QueryMessage` envelope or an angle bracket; it passes structured parameters and receives JSON. This is enforced by confining XML to two modules — one that *builds* request envelopes for the four operations, one that *parses* responses — and a transport module that is the sole HTTP boundary.

Normalization on ingest. Because identity is by code and names are dirty, every human-readable string is cleaned as it is parsed: tabs and stray whitespace stripped, HTML entities decoded, embedded markup removed, stray combining marks normalized. The measure dimension is resolved by its `DimensionType`, never by matching a label. Aggregate ("total") codelist nodes are distinguished from leaves so a query can target real data rather than a computed roll-up.

Status checking that fails closed. The portal answers HTTP 200 even on error, so transport does not trust the HTTP code: it reads the status attribute on the XML root and raises on a non-success value — and, for the named operation roots, raises if the status attribute is **missing** entirely, so a truncated or non-API response can never be mistaken for an empty-but-successful one. A legitimately empty success still passes. The XML parser is additionally **hardened for untrusted plain-HTTP content**: external entity resolution, DTD loading, network access, and unbounded tree expansion are all disabled.

3.4 Correctness and compliance handling

Much of the code exists to make tool output trustworthy and lawful to reuse. These are shipped behaviors:

- **Frequency is explicit and grain-honest.** The portal only slices by frequency when a time window is present; given a frequency without one, it silently returns a mix of native grains. ins-tunisia-mcp closes this trap: a frequency without bounds is given the source's declared range as a window so the requested grain is honored, and a window *without* a frequency returns a structured `frequency_required` error rather than silently widening. A returned grain that differs from the one requested is flagged in the response notes. Observation periods are strings — `"2015"` (annual), `"2015-Q1"` (quarterly). Confirmed accepted codes are annual and quarterly; monthly is accepted by the API but returned no data on tested sources, and semestrial is rejected outright.
- **Missing cells stay missing.** The portal omits absent observations rather than returning zeros, so a present value of `0` is a real zero and a sparse selection simply yields fewer rows — reported as "no data for this selection," explicitly not an error.
- **Structured errors at the tool edge.** Unknown source, empty selection, unknown dimension, and an empty per-dimension key list are all caught *before* the API call and returned as structured `{error}` objects; downstream API rejections and unsupported frequencies are likewise translated to structured errors. The agent never sees a raw exception or the transport layer.
- **Mandatory source attribution.** Every data-returning tool embeds an `attribution` block — producer, source URL, license URL, last-updated date, and a neutral disclaimer — present even on error returns. The `last_updated` value is derived from the data actually returned (the element timestamp, falling back to the dimension's last-updated date) and normalized to an ISO date; it is never invented, and is `null` when genuinely unavailable. This satisfies the attribution requirement of the INS *conditions d'utilisation*; the block is documented as a compliance field that consumers must reproduce and must not alter.

4. Design Decisions and Rationale

4.1 The server owns all XML, and identity is by code

The single most consequential decision is the façade itself: the server's entire reason to exist is to absorb the Prognoz XML so the model never touches it. Everything else follows. Because the portal's names are dirty and unstable, identity is by `Id`/`KEY` and the measure axis is resolved by `DimensionType`, never by label — the only way to be robust against tabs, typos, and French/English drift. Cleaning happens once, on ingest, at the single parse boundary, so no tool re-implements it.

4.2 Declared range, `has_data`, and a fail-open probe

The portal offers no honest "is there data here, and for what years" signal, so the server constructs one — and deliberately splits it into three non-interchangeable pieces (Figure 2): the declared period (possibly a horizon), the per-cube `has_data` flag, and per-selection coverage. The `has_data` flag was the subtlest to get right. The governing constraint is that it must **never mislabel a populated cube as empty**, because a wrong "no data" hides real data permanently in an archival sense — worse than the prior honest-but-tedious behavior. Hence the strict fail-open rule (only `false` on a cleanly proven empty sentinel) and the choice to probe all measure members rather than a single node (a specific aggregate node returns the empty sentinel even on populated cubes, which would false-flag). Probing all 78 sources was validated to flag exactly the 10 known-empty census household cubes and no populated cube. The probe was also moved off the request path into a bounded background pass after an earlier inline version made the first call after a cache rebuild block for minutes; the fast structure seed now returns in seconds and the flags fill in shortly after, fail-open until they do.

4.3 Strict frequency handling over a convenient default

When a caller supplies a time window but no frequency, the server returns an error rather than guessing a grain. This is a considered choice: the portal's silent grain-mixing is exactly the kind of failure that produces plausible, wrong, hard-to-notice answers, and a lenient default would reintroduce that risk in friendlier clothing. The cost is one extra round-trip for a caller who forgets the frequency; the benefit is that a returned series is always the grain that was asked for, or an explicit error.

4.4 No authentication, by the nature of the deployment

ins-tunisia-mcp ships **no authentication layer** and exposes **no network listener**. It runs only as a local stdio subprocess of the client that launched it, against a public, read-only, key-less upstream API. There is no HTTP transport to secure and no secret to manage; adding auth to a local stdio tool would be friction for no security gain. This is a narrower posture than some sibling servers adopt, and it is intentional: the distribution model (local, source-run) and the data (public official statistics) make it the honest fit.

4.5 Keyword search, not embeddings

Discovery search is deliberately keyword/substring with accent- and case-folding, not a vector or synonym layer. The folding handles the portal's mixed accented/unaccented French spellings, which is the common real failure; a synonym/embedding layer was held back as unjustified until plain keyword search demonstrably fails at scale. The known consequence — that a user's word may not match the portal's term ("surface" will not find "superficie") — is documented rather than papered over.

5. Limitations and Known Issues

This section is deliberately complete; for an archival document, candid limitations are a credibility feature.

Single live dependency on the upstream portal (sharpest limitation). Every tool that touches data proxies `dataportal.ins.tn` live, over plain HTTP. There is no offline data snapshot. The on-disk cache holds the catalogue and fetched codelists, so discovery and navigation can survive a brief outage, but any `ins_get_data` or `ins_get_availability` call needs the portal. **When the portal is down, data queries are non-functional.** The plain-HTTP transport also means the connection is unencrypted; the XML parser is hardened against malicious content, but the channel itself is not private.

Defined-but-unpopulated cubes. 10 of the 78 sources publish full structure but serve no values (the 2014-census household cross-tab family). The `has_data` flag surfaces this, but the underlying data simply does not exist on the portal; the server can flag the gap, not fill it.

The `has_data` flag is provisional after a reseed. Because probing runs in the background under the fail-open rule, every source reads `has_data: true` until its probe completes (on the order of a couple of minutes for the full catalogue after a cold rebuild). During that window an empty cube is not yet flagged. This is the deliberate, safe direction — never a wrong `false` — but it means the flag is authoritative only once the pass has run.

Search is substring, monolingual-leaning, and title-only for sources. Source search matches cube titles and descriptions, not the indicators inside them, so a user must find the broad cube first and then search its measure dimension. Within-dimension search is accent- and case-insensitive but not synonym-aware ("surface" $\neq$ "superficie") and not transliteration-aware; codelist labels are Latin-script and French-dominant, so an Arabic-script query for a place name will not match.

Frequency coverage is partial and asymmetric. Annual and quarterly queries are confirmed to return data; monthly is accepted by the API but returned nothing on the sources tested (no monthly source is confirmed to exist), and semestrial is rejected by the API. The server passes monthly through (empty is not the same as unsupported) and reports a returned grain that differs from the one requested, but it cannot manufacture a grain the portal does not serve.

Declared period is not availability. A cube's declared `Period` can be a projection horizon, not the range of real data. The server keeps the two separate and exposes real coverage via `ins_get_availability` and the per-query coverage summary, but a consumer that reads only the declared range will be misled — a portal property the server flags rather than removes.

Catalogue freshness is on-demand. The catalogue is seeded once and then reused until `ins_refresh_catalogue` is called or a cache-version change forces a rebuild; there is no automatic refresh interval. Cubes added to or changed on the portal between refreshes are not reflected until a manual refresh.

Testing leans on saved fixtures and a thin live smoke test. The suite is offline-first — it validates the XML build/parse layers, normalization, the catalogue and `has_data` logic, and the tool-edge error contracts against committed XML fixtures, with a small opt-in set of live tests. This is fast and deterministic, but most coverage asserts against captured responses rather than the live portal, and the live surface exercised each run is intentionally minimal.

6. Future Work

Concrete next steps, in rough priority order:

- **Lower the barrier to running the server.** The current source-only, local distribution serves a technical user. A packaged install or one-command launcher would extend reach toward the non-technical users the data is meant for.
- **Optional offline resilience.** A cached or snapshotted data path, opt-in rather than default, would let discovery and some queries survive a portal outage without giving up live freshness as the default.
- **Sharpen discovery.** Surface a hint on empty source-search results toward within-dimension search, and consider transliteration/Arabic-alias support for place names — the most likely real-user dead end today.
- **Broaden tested surface.** Add end-to-end tool-output checks against the live portal beyond the current smoke test, and periodically refresh fixtures as the portal evolves.
- **Confirm sub-annual coverage.** Establish whether any monthly source exists and exercise the quarterly serialization path more broadly end-to-end.

A separate evaluation of the system's effect on answer quality, and any systematic audit of the portal's data quality, are left to future, separate work.

7. Conclusion

ins-tunisia-mcp turns an awkward XML statistical portal into something an LLM can query in natural language. Its value is not the protocol it speaks or the API it wraps, but the façade between them: a small generic tool suite, a self-healing catalogue cache, careful XML normalization and fail-closed status checking, honest and separated availability signals — including a fail-open `has_data` flag that never hides real data — strict frequency handling, and mandatory source attribution on every result. The system as documented here is a local, source-distributed, stdio server with a single live dependency on its upstream portal; its limitations are real and stated plainly.

About the Author

Tarek Gasmi

FOUNDER, TANITDATA

Tarek Gasmi is Founder of tanitdata and Head of Data and AI at datadoit, with an academic affiliation at the University of Manouba. His work focuses on AI governance, digital infrastructure, sustainability, and technology policy, with particular attention to transitional economies and the geopolitical economy of AI systems.

Citation and License

Suggested citation. Gasmi, T. (2026). *ins-tunisia-mcp: A Local MCP Server for Conversational Access to Tunisia's National Statistics.* Technical report TD-TR-2026-02. tanitdata, Tunis.

Version. v0.1.0, June 2026.

References

Model Context Protocol — open standard specification and SDKs. modelcontextprotocol.io.

Portail de données de l'Institut National de la Statistique (INS), Tunisie. dataportal.ins.tn. Conditions d'utilisation: ins.tn/conditions-utilisation.

ins-tunisia-mcp source repository, github.com/tanitdata/ins-tunisia-mcp (source tree, `README`, `NOTICE`, `LICENSE`, `pyproject.toml`).