© 2026 Yiguedan Philippe Andy Yaro



Tableau Authoring MCP: Spec In, Workbook Out

Yiguedan Philippe Andy Yaro

View GitHub Repository: github.com/andyyaro/tableau-public-authoring-mcp

Let's connect! Email: yyaro@gmu.edu

LinkedIn: linkedin.com/in/andy-yaro



Tableau Public Authoring MCP lets an AI assistant such as Claude Code write a Tableau workbook from a short JSON description (plain structured text), open it in the real Tableau Public app on a Mac to check it, and publish it through the app's own save dialog. MCP, the Model Context Protocol, is how the assistant calls it.

I built it with Claude Code agents. The feature code was committed within about nine hours on 27 July 2026 (US Eastern time); continuous integration (CI) followed on 19 August, and a 67.5-second launch film in November.

On 3 November 2026 this deck checks the repository at its latest commit: it runs the tests, builds a workbook twice and calls the server's safety checks offline. It also says where the docs, and my notes, claim more than the code. The Mac half could not be run here. I am Yiguedan Philippe Andy Yaro.

A. Why It Exists

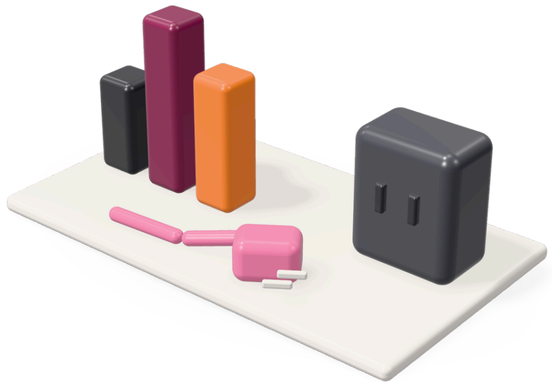


Tableau Public is the free edition of Tableau's desktop app for charts and dashboards. It can publish only to the public website public.tableau.com, where anyone can see the work.

It has no publishing API, no interface other programs can call to publish, and no official library builds a workbook from scratch, so an AI assistant cannot make or publish one on its own.

This project is an MCP server, a program an AI client calls as tools, that writes the workbook, checks it in the real app and publishes it.

1. Six Facts That Shaped the Design

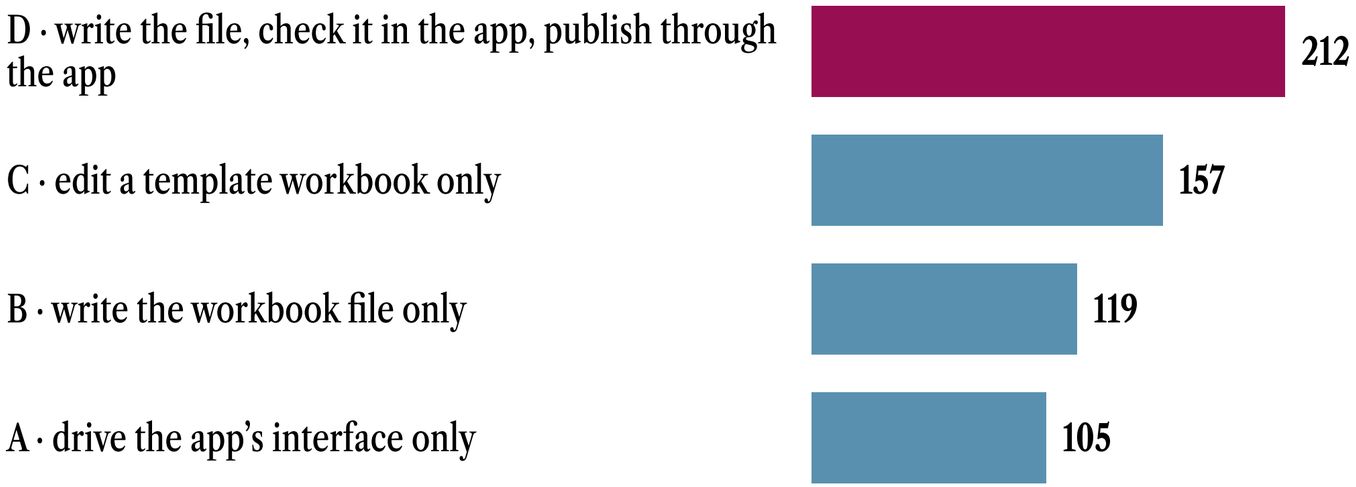
	What the research found	So the design must
1	Tableau's REST API, its web interface for programs, covers Tableau Server and Tableau Cloud, not Tableau Public.	Publish through the desktop app's own Save to Tableau Public command.
2	The Document API, Tableau's Python library for workbook files, cannot create a workbook from scratch.	Write the workbook's XML, text with tags, itself.
3	Tableau Public 2026.2.0 has no AppleScript dictionary, the usual hook for scripting a Mac app.	Drive it with macOS accessibility calls and input events made in code.
4	Tableau Public will not save or publish a workbook whose data is not an extract (error code 3C242D89).	Turn every CSV file, a plain table of comma-separated values, into a Hyper extract.
5	Save to Tableau Public stays disabled until a sheet shows at least one field.	Refuse any sheet without a field before Tableau sees it.
6	Tableau is built with Qt: accessibility clicks report success and do nothing.	Send real mouse and keyboard events (CGEvent).

From the repo's research notes and architecture decision record, both dated 26 July 2026. On 3 November, web searches supported facts 1, 4 and 6; the others are as the research recorded them.

API = an interface other programs call
extract = a copy of the data kept in the workbook
Hyper = Tableau's extract file format
Qt = a toolkit for building app windows
CGEvent = a Mac input event made in code

2. Four Strategies, Scored

Four strategies, scored on 12 weighted criteria (240 points at most)



D was chosen: 88% of the points. Only D both writes workbooks and publishes them.

Python over TypeScript: the work is local files plus macOS automation.

From the architecture decision record, dated 26 July 2026. It names its decision owner as "Claude Code autonomous run": Claude Code agents did the research and design from my brief.

Why the Others Lose

A: building a chart means dragging fields onto shelves, which the app does not expose to scripts.

B and C: zero on publishing, and nothing confirms Tableau accepts the file.

E, ruled out: the Document API cannot create files, and Tableau's own publishing endpoint is private.

endpoint = an address a program calls
shelf = where Tableau fields are dropped
TypeScript = JavaScript with types

B. Architecture

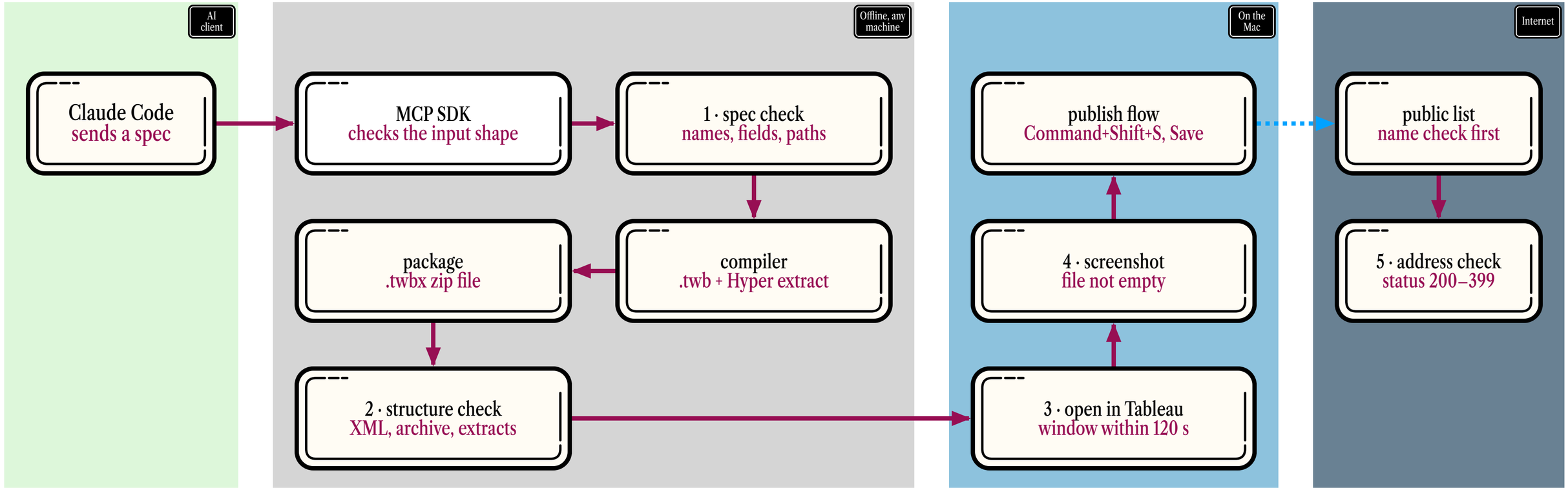


One Python package with two halves. The offline half turns a JSON specification (spec) into a workbook and runs without a Mac; I ran it on Linux.

The desktop half needs a Mac with Tableau Public: it opens the workbook in the real app, takes a screenshot and drives the publish dialog.

An AI client calls ten tools over standard input and output, the plain text channels of a program. The only network calls go to public.tableau.com.

1. From Spec to Published Workbook



Layers 1 and 2, the repo's names for its first two checks, run anywhere. Layers 3 and 4 need a Mac with Tableau Public 2026.2.0 and three privacy permissions; publishing also needs a signed-in session. Layer 5 needs only the internet.

MCP = Model Context Protocol
SDK = software development kit
.twb = a Tableau workbook, as XML
.twbx = the workbook zipped with its data
Hyper = Tableau's extract format

: AI client : Offline : On the Mac : Internet : Next step : Public and irreversible

2. Ten Tools

#	Tool	What it does	Needs
1	get_authoring_capabilities	What the compiler can and cannot express, read from the code	anywhere
2	list_workbook_templates	The seven chart templates and what each needs	anywhere
3	validate_workbook_spec	Dry run: checks the spec against the CSV columns	anywhere
4	build_workbook	The main tool: .twb, .twbx and extracts, then layers 1 and 2	anywhere
5	inspect_local_workbook	Lists a .twbx safely, with size and entry limits	anywhere
6	get_tableau_public_app_status	Install, version, open windows, permission checks	Mac
7	open_in_tableau_public	Opens a workbook and waits for its window	Mac
8	validate_in_tableau_public	Layers 3 and 4: open, wait, screenshot, read any error	Mac
9	publish_to_tableau_public	Guards, then the visible publish flow	Mac, signed in
10	verify_published_workbook	Layer 5: fetches a public.tableau.com address	network

Every tool answers in JSON with an ok field, except input of the wrong shape: the MCP SDK rejects that first, as plain text (section F). No tool can delete a published workbook; that is done by hand in the browser.



C. The Spec



A workbook is described in JSON: its CSV data sources, calculated fields, worksheets, dashboards and where to write the files.

Each sheet picks one of seven templates and names the fields for rows, columns, color, size, label, detail, tooltips and filters.

The spec is checked against the real columns of the CSV data before anything is written; a misspelled field fails, and most such errors list the real field names.

1. What a Spec Can Say

A whole map sheet, from the README

```
$ sed -n 49,57p README.md
{
  "name": "Site Map",
  "template": "map",
  "data_source": "sites",
  "detail": ["Site Name"],
  "color": "Status",
  "size": { "field": "Annual Sessions", "aggregation": "Sum" },
  "title": "Candidate sites"
}
```

Not Supported

Stories, parameters, level-of-detail expressions, table calculations, reference lines.

Joins, custom SQL, live connections, sets, groups, hierarchies.

Filled maps; dashboard filters, legends, actions and floating parts.

Feature	What a spec can say
Templates	bar, line, text table, heatmap, scatter, circle, map
Encodings	color, size and label; detail and tooltips as lists
Aggregations	Sum, Avg, Min, Max, Count, CountD, None, Attribute
Filters	category filters on dimensions only, shown as filter cards
Calculations	Tableau formula text, passed through unchecked
Maps	latitude and longitude points on Tableau's background map
Dashboards	one row or column of up to 20 sheets, fixed or automatic size
Data	UTF-8 CSV only: up to 64 MB, 200,000 rows, 256 columns

Field types are guessed from the first 500 rows. A later value that does not fit becomes empty in the extract, without a warning.



dimension = a category field, such as State
measure = a number that is summed or averaged
encoding = which field sets color, size or label
aggregation = how values combine, as Sum
CountD = count of distinct values

1. One Spec, Built Twice

```
$ .venv/bin/python -I ../checks/build_twice.py tests/fixtures/validation_spec.json \
    tests/fixtures/data/ev_site_readiness.csv /tmp/tab_out
sheets: Readiness by Site, Site Detail, Site Map
dashboards: Overview
inside the .twbx:
   MCP_Authoring_Validation_-_EV_Charging_-_20260727-0050.twb
   Data/sites/sites.hyper
   Data/sites/ev_site_readiness.csv
warnings: none
build a  .twb 13c63da1f7490563  twbx c456a24e2a8b8e8b
build b  .twb 13c63da1f7490563  .twbx 4bfb6ed7687a5dca
.twb identical: True   .twbx identical: False
entries that differ inside the .twbx: ['Data/sites/sites.hyper']
```

1 Inside the .twbx: the workbook XML, one Hyper extract per CSV, and a copy of the CSV itself.

2 The .twb is byte for byte the same in both builds: its internal IDs are hashes of names, not random.

3 The .twbx differs only in the Hyper file, which Tableau's library writes differently each time. The repo's limitations page says so.

Run on 3 November 2026 on a clean copy at commit de0a8a7, with Python 3.13 and the locked dependencies, the exact package versions in the repo's lock file. build_twice.py, written for this deck, builds the repo's validation spec plus the README's map sheet and a two-sheet dashboard, into two folders.

i

hash = a fingerprint of a file's bytes
byte for byte = every byte identical
CSV = comma-separated data file

D. Checking in the Real App



Valid XML, text with tags, is not enough: Tableau's file format has no published schema, or formal description, and the validation report lists five load failures found only by opening files in the app.

So the server opens each workbook in Tableau Public on the Mac, waits for its window, takes a screenshot and reads any error dialog.

This section shows what that proves, and what it does not.

1. Five Layers of Checks

Layer	Runs	Checks	Does not check
1	anywhere	Keys, names and paths; every field exists in the CSV; template rules	Calculated-field formulas
2	anywhere	Well-formed XML with no self-expanding entities; every reference found; an extract per CSV; safe archive names	Whether Tableau will load it
3	Mac	A window whose title matches the file name, ignoring case and underscores, appears within 120 seconds, before any error dialog	Whether an older copy was already open
4	Mac	A screenshot of that window is written and is not empty; then any error dialog is read	What the screenshot shows
5	network	The public address answers with a success or redirect status (200 to 399)	That the chart renders

The tool answers ok: true, with a flag per layer, even when the file never opened: the caller must read the flags. Only the live tests look at the image, 11 of the 15; the other 4 test the publish dialog. At least 0.05% of sampled pixels must differ from the background and 0.02% must be in color, inside a fixed crop box.

schema = a formal description of a file's shape
entity = an XML shortcut that can expand to fill memory
live tests = tests that drive the real app

1. What “Validated” Means in the Code

```
$ sed -n 366,384p src/tableau_public_authoring/server.py
        res = await asyncio.to_thread(app.open_workbook, p, timeout=120)
        payload: dict[str, Any] = {"open": res, "layers": {"layer3_open": res["opened"]}}
        if not res["opened"]:                                                    1
            payload["layers"]["layer4_screenshot"] = False
            return _ok({"validation": payload})

        await asyncio.sleep(float(args.get("settle_s", 8)))
        win = app.find_window(p.stem) or {"id": res.get("window_id")}            2
        shot = shot_dir / f"validate-{p.stem}.png"
        try:
            app.screenshot_window(int(win["id"]), shot)
            payload["screenshot"] = str(shot)
            payload["layers"]["layer4_screenshot"] = True                        3
        except (app.DesktopError, TypeError, ValueError) as exc:
            payload["layers"]["layer4_screenshot"] = False
            payload["screenshot_error"] = str(exc)
        err = app.read_error_dialog(dismiss=True)
        payload["error_dialog"] = err
        payload["layers"]["no_error_dialog"] = err is None
$ sed -n 629,630p src/tableau_public_authoring/desktop/tableau_app.py
    proc = _run(["screencapture", "-x", "-o", "-l", str(window_id), str(out_path)], timeout=30)
    if proc.returncode != 0 or not out_path.exists() or out_path.stat().st_size == 0:   4
```

Code at commit de0a8a7, from src/tableau_public_authoring/server.py and desktop/tableau_app.py. Point 2 is my reading of the code: the repo itself notes that Tableau brings an open copy forward instead of loading a second one.

1. If the file never opens, the tool still answers through _ok, which sets ok to true. Only the flags and an error field say it failed.

2. The window is found by its title alone. An older copy already open would pass, and be the one photographed.

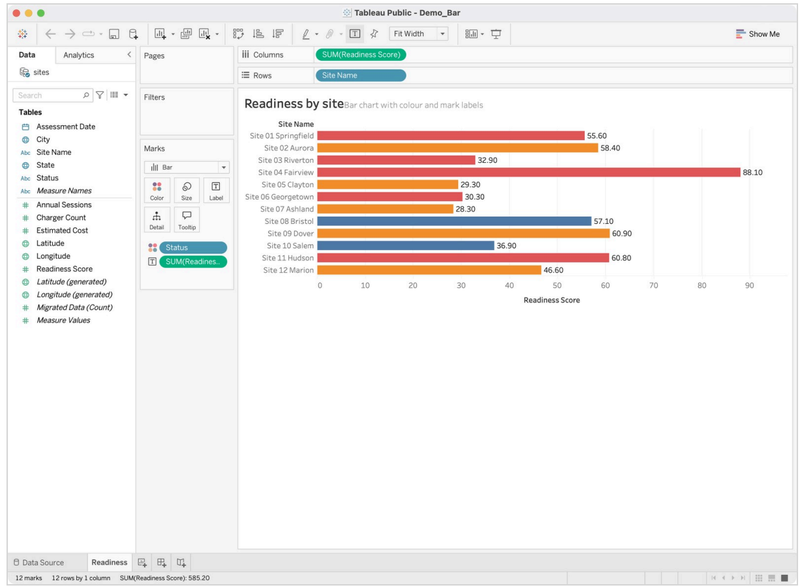
3. Layer 4 passes as soon as the screenshot call succeeds...

4. ...and the only test of that file is that it exists and is not empty. No pixel is examined.

i

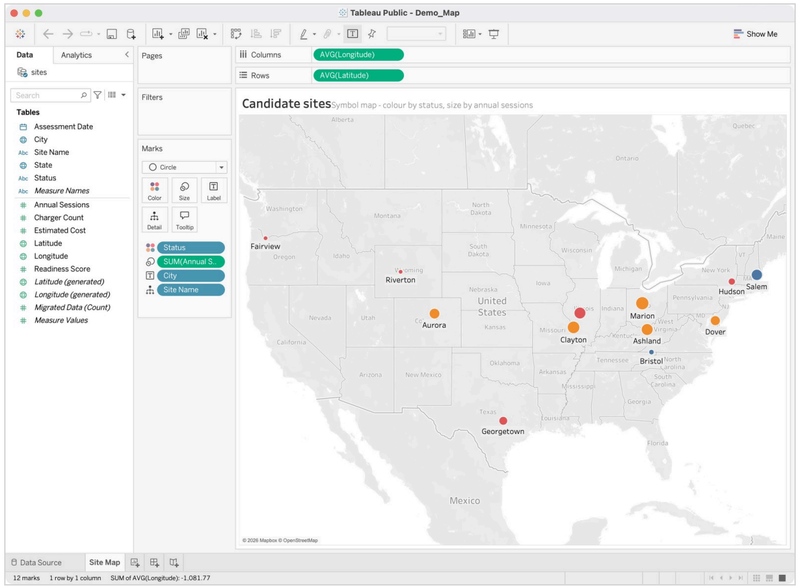
stem = a file name without its extension
_ok = the server’s success reply

2. Three Generated Workbooks, Open in Tableau



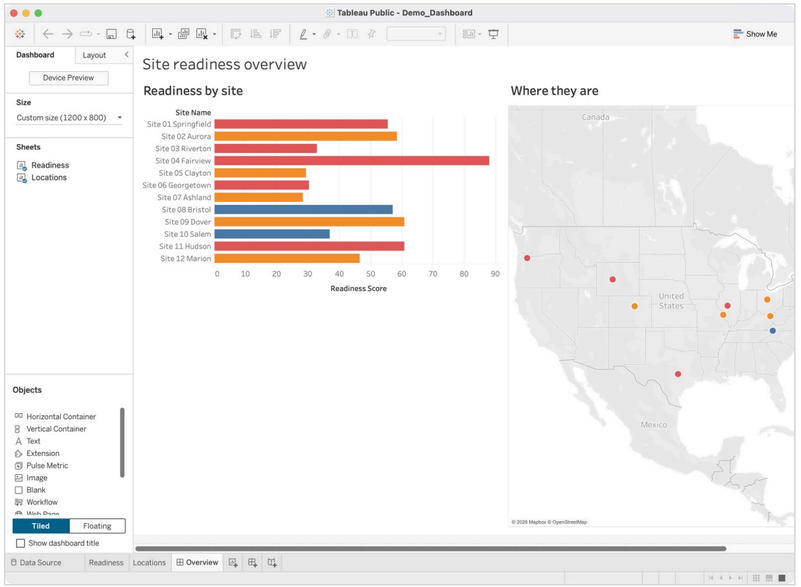
Bar chart

Bar chart: 12 marks with color and labels. The status bar reads SUM(Readiness Score) 585.20.



Symbol map

Symbol map: 12 marks on Tableau's background map, colored by status and sized by sessions.



Dashboard

Dashboard, 1200 × 800: the window is narrower than the canvas, so the map is cut off on the right.

These are the repo's own screenshots, committed on 27 July 2026 and taken on macOS with Tableau Public 2026.2.0. I could not make new ones for this deck: the desktop half needs a Mac with Tableau installed.

E. Publishing

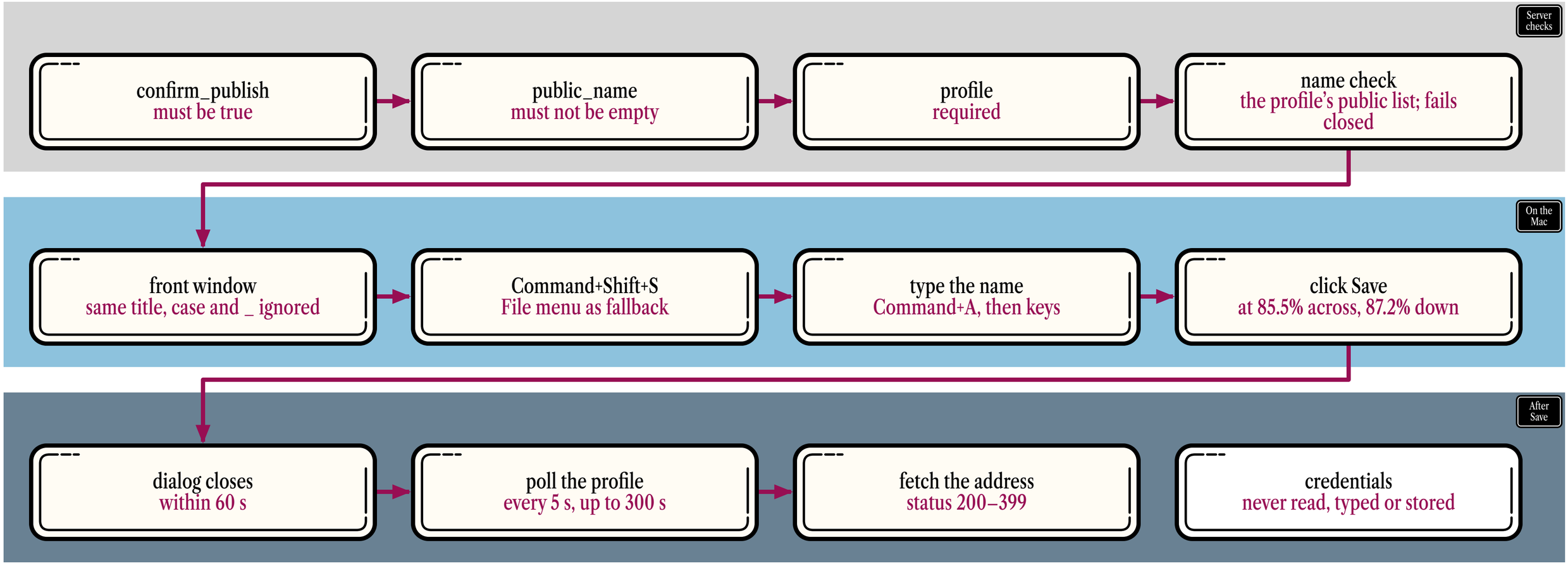


Publishing is public at once, and replacing a workbook cannot be undone, so the server refuses unless the caller sets confirm_publish to true and names the Tableau Public profile.

It then drives Tableau's own Save to Tableau Public dialog in the user's signed-in session. It never reads, types or stores a password.

This flow has run end to end once, on 27 July 2026 during development, and the address answered. No automated test has repeated it, and none presses Save.

1. The Publish Flow, Step by Step



By hand, the user installs Tableau Public, signs in, and grants Accessibility, Screen Recording and Automation permissions to the exact Python the setup check names. Deleting a published workbook is also done by hand: the server cannot.

fails closed = stops when it cannot check
poll = ask again and again

: Server : Mac : After Save : Order of steps

2. Publishing: What Holds and Where It Is Thin

What Holds

No confirm_publish, no profile or an empty name: refused before Tableau is touched.

The name check reads the profile's public list and stops when it cannot read it, unless overwriting is on.

Overwriting is off unless the caller sets overwrite to true.

Before the keystroke, the front window must be the named workbook, or the run stops.

A sign-in window that stays up for 25 seconds stops the run; I sign in and run it again.

Where It Is Thin

With set_title false, which keeps the title Tableau fills in, it publishes under a title the name check never saw.

Hidden workbooks are invisible to the name check.

The front-window check matches names ignoring case, with underscores read as spaces.

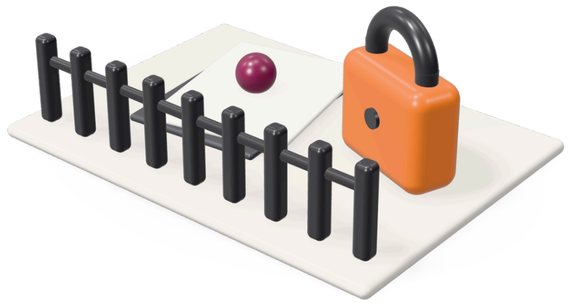
An overwrite can never be confirmed: the public list has no publish time.

The name check relies on an undocumented public address.

One end-to-end run, during development; no automated test presses Save.

Left: read in the publishing code; two of the refusals also run offline in section F. Right: my reading of the code, except the last line, which comes from the tests and the validation report. None of the right column was tested on a Mac.

F. Security



The server takes text from an AI model and turns it into files, commands and clicks on a real desktop. Its threat model, a list of what could go wrong and the defense for each, names ten threats across four points where data passes between parties.

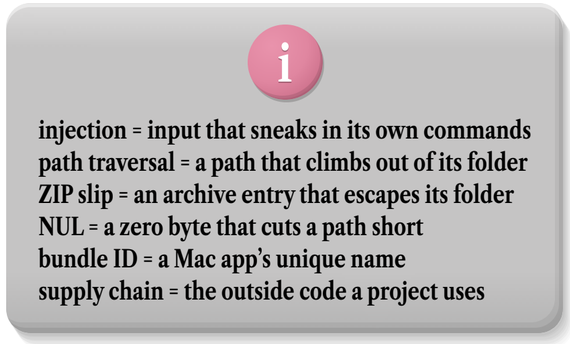
Most of the controls are in the code: XML is built safely, archive and file paths are checked, and no command goes through a shell, the interpreter that could run injected text.

A few controls the documents describe are missing. The next slides show both, run offline.

1. Ten Threats, Checked Against the Code

Threat	In the code	Verdict
T1 · XML injection	XML built with ElementTree only; names escaped	Enforced
T2 · ZIP slip	Archive entry names checked before writing	Enforced
T3 · Path traversal	".." and NUL refused; relative paths kept inside base_dir; system folders blocked for output	Enforced; absolute paths can still read any file the user can
T4 · Shell injection	Argument lists; AppleScript passed on standard input	Enforced
T5 · Credentials	None handled; sign-in handed to the user	Consistent
T6 · Machine details	The .twb is scanned for home paths before packaging	Enforced for the .twb; the CSV is copied as is
T7 · Wrong window	Title match, case and _ ignored, and front-window checks	Partial: no bundle-ID targeting; clicks by position
T8 · Accidental publishing	confirm_publish, overwrite off, name check, dry run	Partial: no tool annotations or timestamped name
T9 · Untrusted CSV	64 MB, 200,000 rows, 256 columns	Enforced for size
T10 · Supply chain	Four runtime dependencies; only mcp capped below version 2	Partial: the model names two; lock file only since 19 Aug 2026

From docs/architecture/THREAT_MODEL.md, checked control by control against the code at commit de0a8a7.



1. The Guards, Run Offline

```
$ .venv/bin/python -I ../checks/guards.py tests/fixtures/data/ev_site_readiness.csv
publish, confirm_publish false -> refusing to publish: confirm_publish must be true  (1)
publish, no profile            -> refusing to publish: 'profile' is required
verify, http:// URL            -> url must be a https://public.tableau.com/ address
tools: 10; with MCP annotations (e.g. destructive): 0  (2)
dry run: new output folder exists before: False
dry run says:                              dry run - no files were written  (3)
dry run: new output folder exists after:  True
unknown key, server's own check: {"ok": false, "error": "specification error: unknown top-level keys: ['bogus']"}
unknown key, through the MCP SDK: isError True  (4)
   Input validation error: Additional properties are not allowed ('bogus' was unexpected)
```

1. Publishing is refused without confirm_publish or a profile; checking accepts only a public.tableau.com address over HTTPS.
2. None of the ten tools carries an MCP annotation, though the threat model says publishing is marked destructive.
3. The dry run says no files were written, yet it creates the output folder.
4. Input of the wrong shape never reaches the server's check: the MCP SDK answers first, in plain text, not the JSON the docs promise.

Run on 3 November 2026 with guards.py, written for this deck. It calls the server's own functions in one process, with no Tableau and no network. The refused publish without a profile still created a hard-coded screenshot folder under the home directory; I deleted it afterwards. Through an MCP client, a missing profile is stopped earlier by the SDK.

annotation = a hint that a tool is risky
HTTPS = encrypted web traffic

2. Where the Documents Promise More

```
$ grep -n "bundle identifier\|annotated destructive\|timestamped" docs/architecture/THREAT_MODEL.md
102:**Mitigation:** automation targets the process by **bundle identifier**
110:**Mitigation:** `publish_to_tableau_public` is annotated destructive, defaults to
112:timestamped name, and refuses any name matching an existing user workbook. A
$ grep -n "^BUNDLE_ID\|kCGWindowOwnerName\|bundle_id=app" \
    src/tableau_public_authoring/desktop/tableau_app.py
21:BUNDLE_ID = "com.tableausoftware.tableaupublic"
169:        if "Tableau" not in (w.get("kCGWindowOwnerName") or ""):
297:        bundle_id=app_bundle_id(app),
$ grep -rlw "golden" tests src | wc -l
0
$ git log --format="%ad %h %s" --date=short -- uv.lock
2026-08-19 923c866 style: re-wrap the docstring lines the em dash pass left too long
$ grep -n "Server > Tableau Public" src/tableau_public_authoring/server.py
182:                "Public account through the visible Server > Tableau Public > Save to "
410:                        "menu_path": "Server > Tableau Public > Save to Tableau Public",
$ grep -n "real_key(VK_ANSI_S" src/tableau_public_authoring/publishing/publisher.py
398:            real_key(VK_ANSI_S, command=True, shift=True)
```

Run on 3 November 2026 on the clean copy at commit de0a8a7. A golden test compares output with a saved, known-good file.

1. The threat model credits bundle-ID targeting, a destructive annotation and a timestamped name.
2. Windows are matched by an owner name containing “Tableau”; the bundle ID is only reported.
3. No golden tests exist, though the threat model relies on them to catch format drift.
4. The lock file of exact versions that the threat model cites arrived on 19 August, in a commit labeled formatting only.
5. The tool description and dry-run reply still name the Server menu path of paid Tableau Desktop...
6. ...while the code presses Command+Shift+S and falls back to the File menu.

G. Tests



507 tests run without Tableau, on Linux, in about 22 seconds. Another 15 drive the real app on a Mac and are switched off by default, by two separate guards.

Continuous integration runs the offline tests on Python 3.10 and 3.13 on macOS, plus a linter (a tool that flags risky code) and a package build, on each push to the main branch and each pull request.

No test presses Save, none goes through the MCP connection layer, and the live tests never run in CI.

1. The Tests, Run

```
$ .venv/bin/pytest -q -p no:cacheprovider | tail -1
507 passed, 15 deselected in 22.10s
$ .venv/bin/pytest --collect-only -q -p no:cacheprovider | tail -1
507/522 tests collected (15 deselected) in 0.22s
$ .venv/bin/pytest -m live --collect-only -q -p no:cacheprovider | tail -1
15/522 tests collected (507 deselected) in 0.23s
$ grep -rhE "^\s*(async )?def test_" tests | wc -l
419
$ .venv/bin/ruff check --no-cache .
All checks passed!
$ .venv/bin/ruff format --check --no-cache . | tail -1
13 files would be reformatted, 62 files already formatted
```

Run on 3 November 2026 on a clean copy at commit de0a8a7, with Python 3.13 and the locked dependencies. On Python 3.10, 505 pass and 2 skip; both skips need a newer standard library.

i

pytest = Python's test runner
ruff = a Python linter and formatter
linter = a tool that flags risky code
deselected = found but not run

1. All 507 offline tests pass.
2. pytest finds 522 in all; the 15 live tests are left out unless asked for.
3. Asked for, they are collected. Without a Mac and Tableau they skip; they did not run here.
4. 419 counts test functions, the figure in my notes. Some functions run several cases.
5. The linter passes.
6. 13 files do not follow the project's formatting rules. CI only reports this; it does not fail.

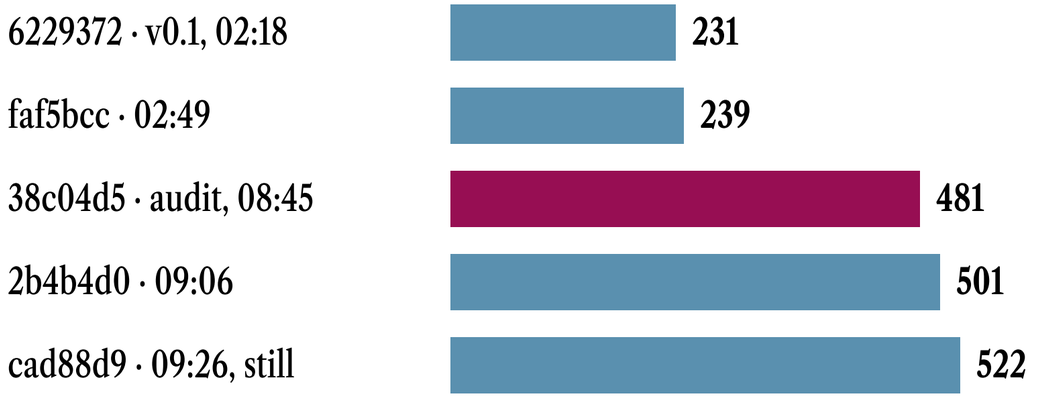
2. Size and Growth

Lines of Python, source and tests



"Code only" leaves out blank lines, comments and docstrings. Tests lead by 6% on all lines and are about equal on code. About a quarter (24%) of the source lines are comments or docstrings.

Tests collected, offline + live, by commit (all on 27 July)



The tests doubled in one commit, which added maps and fixed the findings of an audit by parallel Claude Code agents. The count has not changed since 27 July; later commits only changed punctuation.

What CI Runs

On macOS, Python 3.10 and 3.13: the offline tests, the linter, and the format check as a report only.

A package build, installed into a clean environment and imported.

Never the live tests: CI has no Tableau and no signed-in session.

docstring = a description written inside Python code
CI = checks that run on each change to main

H. History and the Film



Thirteen commits. All the feature code was committed on 27 July 2026 between 00:14 and 09:26 (US Eastern time), with its tests and docs.

I built it with Claude Code agents: the architecture decision record, the document of the design choice, names a Claude Code run as its decision owner, and the last commit credits Claude as co-author.

In November came a logo and a 67.5-second launch film, made without Tableau installed.

1. Thirteen Commits

```
$ git log --reverse --date=format:"%Y-%m-%d %H:%M" --format="%ad %h %s" | cut -c1-98
2026-07-27 00:14 92176d8 Research, architecture, and working workbook compiler
2026-07-27 02:18 6229372 Tableau Public authoring MCP v0.1 (verified end-to-end)
2026-07-27 02:49 faf5bcc fix(desktop): stop treating destroyed windows as minimized ones (#2)
2026-07-27 08:45 38c04d5 Maps, full audit remediation, and public-repo prep (#3)
2026-07-27 08:48 9928bd6 docs: preserve the release write-ups as documentation
2026-07-27 08:50 aaa6503 docs: inline the uninstall steps, drop the machine-specific rollback doc
2026-07-27 09:06 2b4b4d0 feat: one-command install and a preflight that names the real problem
2026-07-27 09:26 cad88d9 feat(spec): relative paths anchored to base_dir, and contained by it
2026-08-19 15:30 38f8ea8 ci: add GitHub Actions workflow and cap the mcp dependency
2026-08-19 18:42 464cc98 style: replace em dashes with hyphens
2026-08-19 18:51 923c866 style: re-wrap the docstring lines the em dash pass left too long
2026-11-01 10:57 d77186f Add project logo
2026-11-02 15:40 de0a8a7 Add the launch film to the README, with its source and watch page
$ git show -s --format=%B de0a8a7 | grep "Co-Authored-By"
Co-Authored-By: Claude Opus 5.5 <noreply@anthropic.com>
$ grep -n "Decision owner" docs/architecture/ARCHITECTURE_DECISION.md
5:**Decision owner:** Claude Code autonomous run (MASTERPROMPT_AUTHORING.md §6)
```

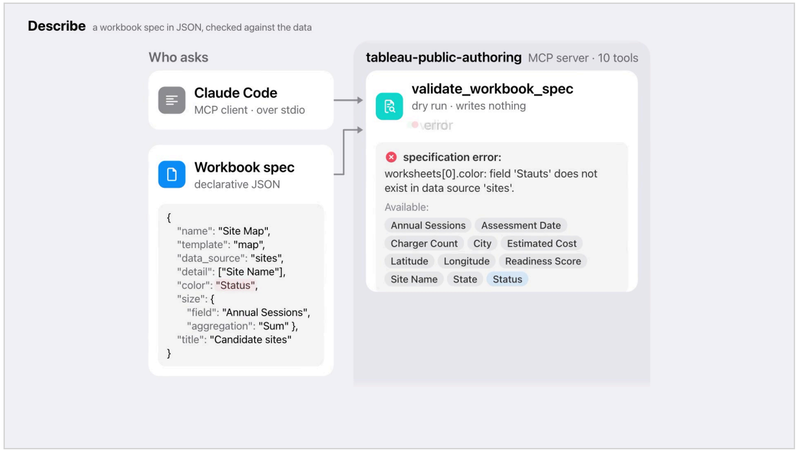
1. Eight commits in about nine hours on 27 July: compiler, v0.1, maps, audit fixes, installer.
2. 19 August: CI and an upper version limit on the mcp library, then punctuation.
3. 2 November: the film commit credits Claude as co-author.
4. The design record names its decision owner: a Claude Code autonomous run.

My notes say “Solo”. Every commit has one human author, but the work was done with Claude Code agents, and the history says so. Two commit titles end in (#2) and (#3): pull-request numbers from a private copy this public repo was started from. Those pull requests are not in this repository.

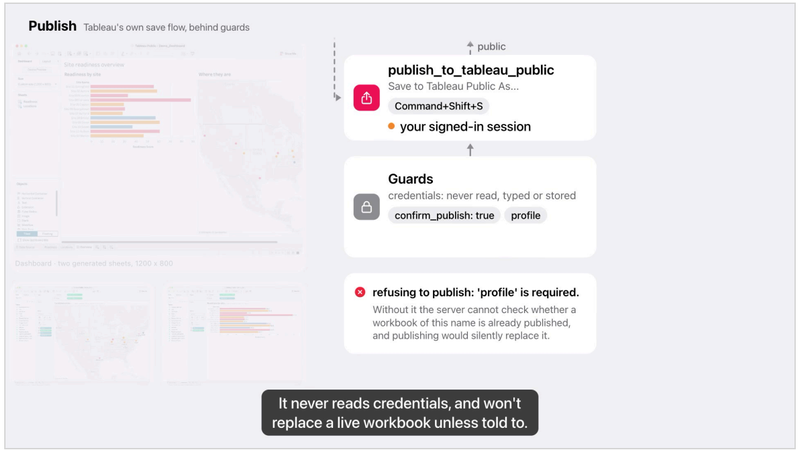
i

commit = one saved change in git
pull request = a proposed change, reviewed before merging

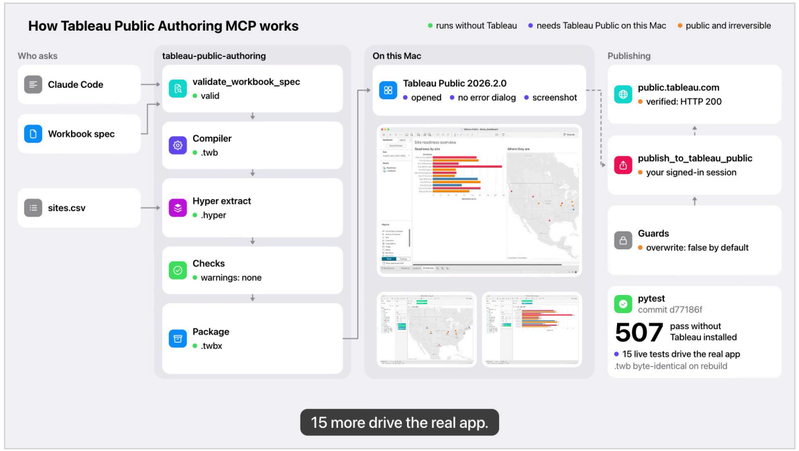
2. The 67.5-Second Film



Describe. A typo fails with the real field names. The label says the dry run writes nothing; it does create the output folder (section F).



Publish. The save flow behind its guards; the refusal shown is the server's own message, captured by calling it directly.



The board. 507 offline tests, 15 live ones, and the one address check from 27 July.

How It Was Made

67.5 seconds, 1920 × 1080, 13.8 MB; the README rounds it to 68.

A web page (HTML) animated with GSAP, a JavaScript animation library, and made into video with HyperFrames; narrated by Kokoro, an open text-to-speech model.

Made without Tableau installed: the server's replies were captured offline; the app screenshots are the repo's own.

Where to Watch

andyyaro.github.io/tableau-public-authoring-mcp/film.html

GitHub Pages serves the page from the repo and the video from the film-v1 release; both answered on 3 November 2026.

I. Proof and Limits



Checked on 3 November 2026: tests, linter and a double build on a clean copy, the server's guards called offline, the code read against its docs, and the history read.

The offline half holds up. The Mac half rests on one setup and one end-to-end publish, and several documents promise more than the code.

The last slides set my claims against the repo, list fixes, and say what the project shows.

1. My Claims Against the Repo

My notes said	The repo shows	So I now say
Jul 27 – Aug 19, 2026	Research dated 26 Jul; all feature code 27 Jul, 00:14–09:26; CI and style 19 Aug; logo and film 1–2 Nov	Built 26–27 Jul 2026; CI 19 Aug; film Nov 2026
Solo	One person in git; a Claude Code run owns the design decision; audits by parallel agents; Claude co-authored the film commit	Solo, built with Claude Code agents
419 tests	522 collected: 507 offline, 15 live. 419 counts test functions	507 offline tests and 15 live tests
~17,000 lines	16,925 lines of Python in source and tests	About 17,000 lines of Python
More test code than source	8,716 against 8,209 lines; level once blank, comment and docstring lines are dropped	About as much test code as source
Validates in the real app	Opens it, waits for the window, takes a screenshot, reads errors; the tool never looks at the image	Opens each workbook in Tableau Public and checks for load errors
Publishes through the desktop workflow	Built and guarded; run end to end once, during development, on 27 Jul 2026	Publishes through the app's own dialog; run once end to end
1 star	Not checkable here: GitHub's API was not available to this session	Leave it out

From my career-hub notes, checked against the repo on 3 November 2026. CI = continuous integration; a docstring is a description written inside Python code.

2. What to Fix Next

1. **Close the set_title gap**
   Check the title Tableau will really use, or refuse set_title false, and add the missing test.

2. **Make validation say what it saw**
   Answer ok: false when nothing opened, refuse when the workbook is already open, and add the live tests' pixel check.

3. **Add tool annotations**
   Mark publishing as destructive, as the threat model already claims.

4. **Make the dry run write nothing**
   Create the output folder only when building.

5. **Bring the docs in line**
   The menu path, the threat-model controls, "always JSON" and the changelog's "39 tests"; and 522 tests, not 419, in my notes.

6. **Fix the setup check**
   Test Accessibility rather than Automation, look at the Screen Recording image, and advise mcp below version 2.

7. **Remove personal traces**
   Drop the hard-coded folder that names my private repo; decide about my email on commits and an old log file in history.

8. **Tighten CI**
   Make the format check fail the build, test through the MCP connection layer, and run the live tests on a second Mac.

In order of risk: publishing first, then what the tools report, then the docs and housekeeping.

3. What It Shows, and What It Does Not

It Shows

A working compiler: a JSON spec becomes a Tableau workbook with its extract, and the same spec gives the same .twb every time.

A careful way to drive a desktop app that offers no scripting interface, with most of its threat model enforced in code.

507 offline tests passing and a clean lint, on a clean copy.

A publish flow that refuses by default, and worked once end to end.

It Does Not Show

That publishing works today: the only end-to-end run was on 27 July 2026, during development.

That validation catches a blank or wrong chart: the tool never looks at the image.

That it works beyond one setup: macOS, Tableau Public 2026.2.0 in English, window sizes measured on one machine.

That the documents can be taken at their word.

The offline half is solid and well tested. The next step is making the Mac half report what it really saw, and making the documents say exactly what the code does.