Project Handoff & Developer Diary: MCPO Enhancement *(Updated 2025-08-19)*

Objective: To evolve the baseline `mcpo` orchestrator into a robust, manageable service with a full-featured web UI for real-time control, while aligning with the latest MCP specification and ensuring stability through testing.

Guiding Principle: Every feature, especially in the UI, must be fully functional and backed by a real backend operation. There are to be no placeholder or non-working elements.

---

1. Project State Snapshot (As of Handoff)

- **Overall Status:** The project is in a stable, feature-complete state based on the defined scope. All implemented UI elements are functional.
- **Test Suite:** The backend test suite is comprehensive and passing (`49 passed` as of this update). It covers core logic, configuration, timeout handling, structured output, health endpoint, SSL exposure, tool error envelopes, streamable HTTP transport normalization, and new persistence + read-only gating tests. Some management mutation flows remain untested (see gaps section).
- **Key Deliverables:**
  1. A hardened backend with robust error handling, timeouts, and state management.
  2. A fully integrated React-based web UI for complete operational control.
  3. Alignment with core principles of the June 2025 MCP specification.

Completed Work Breakdown:

Category	Feature	Status	Notes
Core Backend	Robust Timeout Handling	**Complete**	CLI flags for default/max; per-request override via header/query; returns 504 on timeout.
	MCP Spec Alignment	**Complete**	Injects `MCP-Protocol-Version` header; handles structured tool outputs.
	Standardized API Responses	**Complete**	All responses use a `{ok: true/false, ...}` envelope for predictability.
	Pydantic Config Validation	**Complete**	The `mcp.json` file is validated on load and reload, preventing errors from malformed configs.
	Hot Reload Safety	**Complete**	An `asyncio.Lock` prevents race conditions during concurrent file-watch events.
Management & API	In-Memory State Management	**Complete**	Tracks `enabled` state for all servers and individual tools.
	Management API (`/_meta`)	**Complete**	Full suite of endpoints for reading state and triggering actions (reload, restart, toggle, add, remove).
	State Enforcement	**Complete**	Tool invocation path checks the state map and returns a `403 Forbidden` if a server or tool is disabled.
	Dynamic Configuration	**Complete**	Servers can be added or removed via the API, which persists changes to the `mcp.json` file.
Frontend (UI)	Full UI Functionality	**Complete**	All toggles, buttons (Restart, Reload, Add, Remove, Open Config), and status indicators are fully wired.
	Reactivity	**Complete**	UI polls the backend, reflecting changes to server state and tool availability automatically.
	UI served at `/mcp`	**Complete**	The built React application is served directly from the Python backend.

Known Limitations / Next Steps:

1. **State Persistence (Minimal v1 Implemented):** Server/tool enabled flags persist to a versioned, atomically-written sidecar JSON state file (e.g. `<config>_state.json` / `mcpo_state.json`). On startup, stored enable maps are merged (see rules below). Only

booleans + timestamp + `version` are currently tracked (no error counts or last init metadata yet). Atomic write prevents partial/corrupt saves; unknown fields ignored for forward compatibility.

2. **"Install from Git" Feature:** The UI was designed with a tab for automated Git installation. This backend logic is **not implemented**. The UI currently only supports the "Manual Setup" path for adding new servers.
3. **Security of Management API:** The `/_meta` endpoints that perform write actions (toggle, add, remove) are not protected by a separate, more privileged API key. If a global `--api-key` is set, it protects everything, but there is no granular control.

---

2. Developer's Diary & Rationale Log

This section details the logic behind key implementation decisions made during the project.

A. Timeout System: From Simple to Robust

- **Initial State:** The codebase had an unused `--tool-timeout` flag that did nothing.
- **Problem:** A single, global timeout is insufficient. Some tools are fast (e.g., file lookup), while others are slow (e.g., "deep research"). A low timeout would break long-running tools; a high timeout would risk hanging workers on simple calls.
- **Solution & Logic:**
  1. **Enforce the Default:** We first made the `--tool-timeout` functional by wrapping the `session.call_tool` in `asyncio.wait_for`.
  2. **Introduce a Safety Net:** We added `--tool-timeout-max` to act as a hard upper limit, preventing abuse or misconfiguration from creating excessively long timeouts.
  3. **Provide Flexibility:** The crucial step was allowing per-request overrides via the `X-Tool-Timeout` HTTP header or a `?timeout=` query parameter. This allows clients who *know* they are calling a long-running tool to request more time, up to the configured maximum.
  4. **Clear Error Handling:** A timeout now returns a distinct HTTP `504 Gateway Timeout` with a structured error `{ok: false, error: {code: "timeout"}}`, making it easy for clients to handle programmatically.

B. The "Restart" vs. "Reload" Distinction

- **Problem:** Simply re-reading the config file (`reload`) is not enough to make newly added servers functional. Starlette's `lifespan` events, which initialize the MCP connection and discover tools, only run on application startup.
- **Solution & Logic:**
  1. **`Reload` (`/_meta/reload`):** This is the "lightweight" action. It uses the `watchdog`-based `ConfigWatcher` to diff the config file, mounting new server sub-apps and unmounting removed ones. It's fast but doesn't initialize new servers.
  2. **`Restart` (`/_meta/reinit`):** This is the "heavy" action required after adding a server. We created a `reinit_servers` function that explicitly iterates through all server sub-apps and manually runs the `lifespan` startup logic for them. This connects to the MCP process, discovers tools, and makes the server fully operational without killing the main Uvicorn process. The UI "Restart" button triggers this `reinit` flow.

C. State Management & Enforcement

- **Problem:** The user required the ability to disable a misbehaving or unused server/tool instantly without removing it from the configuration.
- **Solution & Logic:**
  1. **In-Memory State:** We introduced two dictionaries on the main FastAPI `app.state`, `server_enabled` and `tool_enabled`, to act as the single source of truth. This is fast and avoids file I/O on every check.
  2. **Management API:** A suite of `.../enable` and `.../disable` endpoints were created to modify this state.
  3. **Enforcement at the Core:** The most important decision was where to enforce this state. We modified the `get_tool_handler` factory. Before any tool is called, the handler now checks the state maps. If the server or the specific tool is marked as `False`, it immediately returns an HTTP `403 Forbidden`. This is secure and efficient, as it prevents any attempt to even communicate with the underlying MCP process.
  4. **Parent App Reference:** Sub-apps needed access to the main app's state. We achieved this by passing the `parent_app` reference into each sub-app when it is mounted.

D. Frontend Philosophy: "No Placeholders"

- **Initial Conflict:** Early UI iterations included non-functional elements (like the enable/disable toggles) which led to user frustration and accusations of "glazing over" the work.

- **Corrective Action & Logic:** The development pivot was to ensure **every single interactive element in the UI was fully functional**.
    - The "Add Server" modal was initially removed because its backend didn't exist. It was then **restored** only *after* the `POST /_meta/servers` endpoint was built.
    - The toggles were re-implemented to call the real `.../enable` and `.../disable` endpoints. Their visual state is now a direct reflection of the backend's state, refreshed via polling.
    - The result is a UI with slightly fewer features than the original *design brief* (e.g., no "Install from Git"), but where **100% of what is visible to the user is real and operational.** This builds trust and provides genuine utility.

---

3. Endpoint Catalog & Capabilities

Endpoint	Method	Purpose	Mutates State	Auth (if api_key set + strict)	Idempotent	Notes
`/healthz`	GET	Basic health snapshot	No	Optional	Yes	Includes generation + last reload time
`/_meta/servers`	GET	List mounted servers + flags	No	Optional	Yes	Enabled flag derived from in-memory map
`/_meta/servers/{server}/tools`	GET	List tools for server	No	Optional	Yes	Sorted by name
`/_meta/config`	GET	Return config path	No	Optional	Yes	Path only, not contents
`/_meta/reload`	POST	Diff config + mount/unmount + init new	Yes	Optional	Partially	Existing servers untouched unless changed
`/_meta/reinit/{server}`	POST	Re-handshake a single server	Yes	Optional	No	Tears down session + rebuilds endpoints
`/_meta/servers`	POST	Add server then reload	Yes	Optional	No	Persists to config file
`/_meta/servers/{server}`	DELETE	Remove server then reload	Yes	Optional	No	Persists removal
`/_meta/servers/{server}/enable`	POST	Enable server	Yes	Optional	Yes	In-memory only currently

Endpoint	Method	Purpose	Mutates State	Auth (if api_key set + strict)	Idempotent	Notes
`/_meta/servers/{server}/disable`	POST	Disable server	Yes	Optional	Yes	In-memory only currently
`/_meta/servers/{s}/tools/{t}/enable`	POST	Enable tool	Yes	Optional	Yes	In-memory only
`/_meta/servers/{s}/tools/{t}/disable`	POST	Disable tool	Yes	Optional	Yes	In-memory only
`/{serverMount}/{tool}`	POST	Invoke tool	No*	Optional	No	Underlying MCP tool call; *internal side-effects unknown

Planned (Not Implemented): metrics, version, bulk operations, state export, WebSocket/SSE updates.

4. State Model (Current vs. Proposed)

Current (persisted minimal structure in sidecar state file):

```
{
    "server_enabled": { "<server>": true/false },
    "tool_enabled": { "<server>": { "<tool>": true/false } }
}
```

Proposed enriched schema (future superset of current minimal state file):

```
{
    "version": 1,
    "servers": {
        "<server>": {
            "enabled": true,
            "tools": { "<tool>": { "enabled": true } },
            "lastInit": "2025-08-18T00:00:00Z",
            "lastError": null
        }
    },
    "updatedAt": "2025-08-18T00:00:00Z"
}
```

Merge Rules (on startup):

1. If state file missing -> all enabled.
2. If tool missing in persisted record -> default enabled.
3. If persisted tool absent in live list -> drop silently.
4. Unknown fields ignored (forward compatibility).

5. Error & Timeout Taxonomy

Code	HTTP	Trigger	Client Guidance
`timeout`	504	Tool exceeded effective timeout	Retry with adjusted timeout if safe
`not_found`	404	Missing server or endpoint	Refresh server list; user may have removed it
`exists`	409	Add server name collision	Choose different name
`invalid`	422	Validation failure (payload/command)	Correct request; re-submit
`io_error`	500	File write failure persisting config	Check filesystem perms/disk space
`reload_failed`	500	Exception during reload apply	Inspect logs; retry once
`reinit_failed`	500	Exception during re-handshake	Retry; if persistent inspect server process
`no_config_mode`	400	Mutation when not config-driven	Run with config file or disable feature
`no_config`	400	Reload invoked without config	Provide config path
`disabled` (planned)	403	Server/tool disabled execution attempt	Enable first or skip tool

Action Item: unify current 403 responses to include `code: "disabled"`.

6. Test Coverage Matrix (Post-Snapshot Gaps)

Domain	Scenario	Status
Enable server	Basic enable/disable + list reflects	Missing
Tool disable enforcement	Disabled tool 403	Missing
Add server (stdio)	Happy path	Missing
Add server (duplicate)	409 exists	Missing
Remove server	Tools inaccessible post-remove	Missing
Reinit	Success + preserves enable flags	Missing
Reload diff	Add/remove/update config changes	Missing
Timeout override	Below default / above max clamp	Present (base)
Structured output edge	Complex nesting classification	Partial
Concurrency	Two reload POSTs serialized	Missing
Envelope uniformity	404/422/403/500 shapes	Partial

Priority: Implement missing rows before next release. (Add tests for persistence load/merge as well.)

7. Reliability & Performance Opportunities

Short Term:

- Add exponential backoff (2,4,8...) for failed init with cap.
- Hash tool schemas to skip redundant endpoint rebuilds.
- Collapse per-tool routes into generic /invoke (optional optimization) when tool count > threshold.

Long Term:

- Replace polling with SSE/WebSocket event channel (server_added, server_removed, state_changed, tool_list_changed).
- Circuit breaker for repeatedly failing servers (auto-disable after N failures).

8. Observability Roadmap

Logging Fields (recommend): `ts, level, action, server, tool, duration_ms, outcome, error_code`.
Metrics (MVP): counters (tool_calls_total, tool_errors_total{code}), gauges (servers_enabled, tools_enabled), histogram (tool_call_duration_seconds).
Expose via: `/ _meta/metrics` (JSON) or optional Prometheus format.

9. Security & Threat Model

Risks:

1. Unauthorized config mutation.
2. DoS via many long-running tool invocations (mitigated by max timeout; still concurrency risk).
3. Malicious command injection in added servers.
4. Information leakage if future verbose logs include tool inputs.

Mitigations (planned):

- Role-based API keys or signed JWT with `role=admin` for mutation endpoints.
- Command whitelist or sandbox directory for stdio servers.
- Rate limiting (token bucket) on tool execution.
- Read-only mode flag (CLI: `--read-only` or env: `MCPO_READ_ONLY=1`) (implemented) blocks all mutation endpoints.

10. Roadmap (Now / Next / Later)

Stage	Items
Now	Persistence for enable flags; mutation endpoint tests; disabled error code normalization; accountability doc
Next	Metrics endpoint; SSE updates; security (role separation); state file schema v1
Later	Circuit breaker; tool invocation history; WebSocket live logs; Git install automation; plugin discovery

11. Operational Runbook (Essential)

Reload Config: POST `/_meta/reload` → Expect `{ok:true}`; if failure inspect logs then retry once.
Add Server: POST `/_meta/servers {name, command|url,type?}` → On success tool endpoints appear after reload (auto inside handler).
Disable Misbehaving Tool: POST disable tool → Next call returns 403 (planned code `disabled`).
Recover Stuck Server: POST reinit; if still disconnected check underlying MCP process manually.
Emergency Read-Only Mode: restart with `--read-only` (or set `MCPO_READ_ONLY=1`) and all mutation endpoints return 403 (non-destructive containment mode).

12. Ethical & Professional Guidelines

1. No Placeholders: Ship only controls with real backend behavior (maintain trust).
2. Transparent Failures: Always return structured envelopes—never swallow errors silently.
3. Principle of Least Power: Default to read-only; require explicit opt-in for mutation features in production deployments.
4. Auditability: Log every mutation (who/when/what) without sensitive payload contents.
5. User Agency: Provide clear disable & rollback paths for newly added servers/tools.
6. Avoid Dark UX: No silent auto-disables—surface reasons (error count, circuit breaker) to the operator.
7. Privacy Respect: If tool inputs may contain sensitive data, allow redaction in logs (config flag).
8. Security by Configuration: Document safe defaults; insecure options must be explicit (e.g., `--allow-unsafe-commands`).
9. Minimal Attack Surface: Do not expose shell execution beyond defined MCP server commands.
10. Integrity Over Feature Velocity: Add tests before expanding surface area.

13. Immediate Low-Effort High-Impact Actions (Refreshed)

```
1. Add accountability mapping (UI control → endpoint) in `DECISIONS.md` or a new
`ACCOUNTABILITY.md`.
2. Normalize all 403 disabled responses to `{ok:false,error:{code:"disabled"}}`.
3. Add unit tests: enable/disable, add/remove, reinit, reload diff (add/remove server),
persistence merge edge (missing tool, stale server).
4. Add metrics stub endpoint returning counts (servers_enabled, tools_enabled, tool_calls_total,
tool_errors_total{code}).
5. Draft design note for SSE/WebSocket event channel before implementation.
```

14. Handoff Verification Checklist

Item	Present	Notes
Snapshot Commit Ref	Yes	330876d7fcb200a0... (original); ongoing updates continue
Working Tests	Yes	49 passing (2025-08-19); mutation / enable-disable still needs coverage
UI Functional	Yes	All visible controls wired
Read-Only Mode Flag	Yes	`--read-only` / `MCPO_READ_ONLY=1` blocks mutations
Mutations Protected by API Key (strict)	Optional	Single shared key; no RBAC
Error Envelope Uniformity	Improving	Disabled returns code="disabled"; audit remaining 403 paths
Persistence of Enable Flags	Partial	Minimal versioned atomic sidecar (booleans + timestamp + version)
Structured Output Classification	Basic	Images/resources tagged; circular refs -> Any
Documentation (README+Diary)	Extensive	Diary updated; consolidation later

15. Closing Note (Updated)

State persistence advanced to minimal versioned, atomic implementation and read-only mode is live. Immediate strategic priorities: (1) add tests for enable/disable + add/remove/reinit + reload diff + persistence merge edges, (2) optionally introduce a flag to disable admin endpoints entirely (in addition to read-only), (3) extend state file schema with tool/server metadata (lastInit, lastError), (4) metrics & disabled error code normalization. This diary is a living intent ledger—keep tables in sync to avoid drift.

Last editorial update: 2025-08-19.