ChatGPT

1. Restated Context & Assumptions

A single developer with ADHD is building a local “AI Life OS” on Windows 11. The core system uses Claude Desktop as the **Agentic Kernel** (reasoning engine), connected via the Model Context Protocol (MCP) to deterministic tools (“the Hands”) such as a filesystem driver, Git, an n8n automation engine, a (possibly sandboxed) shell, and web fetch tools [1] [2]. The system maintains all state in a **Git-backed Truth Layer** (YAML/JSON files under version control) [2] [3]. Work is performed in a strict **Chat→Spec→Change** cycle: the user “chats” high-level goals, the agent generates a formal “Spec” (including a diff preview and validation report) for user approval, then on approval the agent applies the “Change” to the Truth Layer (via filesystem writes and git commits) and executes it (e.g. deploying an n8n workflow) [4] [5]. This creates a *cognitive pause* that mitigates impulsivity and ensures deterministic execution [4].

Key constraints: the system must be robust despite one human’s limited time and working memory. All subsystems should guard against hallucination, drift, or accidental destructive actions (e.g. strict MCP scoping and spec approvals) [2] [6]. The design assumes the user follows minimal rituals (like a “daily standup” status report) [7] and that the Truth Layer is the single source of state (any divergence between state files and the real environment is catastrophic [8]). We assume *no* large external teams or redundant operators – the solo builder must trust the AI tools and do only 1–2 hour bursts of work. The research documents (architectural blueprints, implementation plan, playbooks) are the primary source, with all recommendations grounded in these assumptions.

2. System Map & Dependency Overview

Component	Depends On	Provides	If It Fails
Agentic Kernel (Claude Desktop)	Windows 11, Claude API; MCP configuraton file; network (if using cloud models)	Natural-language reasoning, spec generation, orchestration; user interface	**Breaks**: entire orchestration stops. No new actions, user stuck in chat.
Filesystem MCP server	NodeJS, permission to user repo (Truth Layer)	Read/write access to truth files (YAML/JSON) and log files [3]	**Breaks**: cannot read or update state. Agent loses “grounding” – hallucinations ensue [2]; no new state updates.
Git MCP server	Git CLI/binary, access to Truth Layer repo	Version control: commits, diffs, rollbacks, change history [9]	**Breaks**: no commit ability. Changes can’t be saved or reverted; loss of audit trail; increases risk of undetected errors.

Component	Depends On	Provides	If It Fails
n8n MCP (API bridge)	n8n API endpoint (localhost:5678), Docker running	Automations: create, inspect, execute workflows on schedule (cron, webhooks) [10] [11]	**Breaks**: scheduled jobs and integrations stop. Any automation-dependent tasks fail silently or miss deadlines.
n8n Engine (Docker)	Docker Desktop/ WSL2, host networking	Executes workflow logic asynchronously; decouples timing from Claude [12] [13]	**Breaks**: Cron-like and webhook triggers are disabled; tasks dependent on n8n will not run.
Shell MCP (cmd/ PowerShell)	MCP-shell tool, constrained environment	Ad-hoc system commands (file exec, tests). In Phase 2.3 typically whitelisted to safe tools [6]	**Breaks**: dangerous. If unrestricted, agent could run any cmd (risk catastrophic changes). If missing, some tasks (e.g. pip installs) cannot run.
Web Fetch / Browser MCP	Internet connection, Puppeteer or fetch server	Retrieves external documentation or data (e.g. n8n docs, web search)	**Breaks**: no external lookup ability. Agent cannot learn new tools autonomously (missing docs) or do web search fallbacks.
Truth Layer (Git repo)	Filesystem, Git	Canonical state store: system config, active workflows, memory [2] ; "safety net" with undo [2] [8]	**Breaks**: if repo is corrupt or overwritten, system loses reliable state. Potential irrecoverable loss of history.
Observed-State Monitor (new)	Filesystem/Git MCP + agent logic	Checks for "drift" between truth and reality; raises change requests if out-of-sync [8]	**Breaks**: drift goes undetected. State divergence erodes trust (user abandons) [8] .
Windows 11 Host (WSL2, Docker)	Hardware, OS stability	Underpins all local services and containers	**Breaks**: full outage. Updates or crashes kill Docker and tools.

Each component has clear dependencies. For example, the Agentic Kernel requires both the local OS and network access to APIs. The Filesystem and Git MCP servers depend on correct configuration scope (to limit paths [14]). n8n needs Docker running. The Truth Layer critically depends on Git and file I/O integrity. Failures often cascade: e.g. filesystem inability leads to hallucination or inability to persist state [2] .

3. Technical Failure Modes

LLM Hallucinations / Context Drift (Reasoning Layer): Claude may generate incorrect or spurious plans if its context is incomplete or if the AI drifts from the real state. This is a *known critical failure mode* [2] . For example, the agent might hallucinate a file or misinterpret a spec, producing a destructive command. The Truth Layer is designed to mitigate this by grounding the agent in deterministic data [2], but it still relies on the agent correctly using it. Without additional safeguards, hallucinations can cause state corruption.

Concurrency & State Corruption (Truth Layer): With multiple actors (Claude and n8n both writing to the Truth Layer), **race conditions** can occur. A classic *lost update* happens if Claude and n8n both edit `tasks.json` around the same time: the last write overwrites the other's changes [15]. Current MCP file tools do not enforce locking by default, so without custom version checks or lockfiles, state files can silently corrupt [16] [17]. Similarly, agents might introduce **schema drift** – e.g. hallucinating a new field or changing a type – making files invalid [18]. Without concurrency control (optimistic locking or mutex locking), data integrity is compromised [19] [20].

Infinite Loops / Recursion: The orchestration layer (e.g. LangGraph) can enter unintended loops. If a coding loop fails repeatedly, it could consume all tokens or resources. LangGraph allows setting a recursion limit [21], but if not tuned, a Research or Builder agent might spin (e.g. repeatedly retrying a failing fix) until hitting limits. If no fallback is defined, this could crash or freeze the system. Proper retry policies and loop termination (ask user after N attempts [21]) are essential to avoid runaway scripts.

MCP Server Failures: Each MCP tool is a single point of failure. If the filesystem or git MCP crashes (due to bugs, memory, or misconfiguration), Claude will lose access to core functionality. A hung or misbehaving `server-filesystem` might hang the agent or cause timeouts. A broken `git` tool could leave the Truth Layer unreachable. Similarly, if the n8n-mcp bridge breaks, Claude cannot inspect or deploy workflows. Because MCP servers often run as local subprocesses, failures may manifest as JSON-RPC timeouts or parse errors (not always obvious to the user). For example, forgetting to set `MCP_MODE=stdio` can cause silent parse failures [22].

Docker/WSL/Network Issues: The n8n engine and other containers run in Docker. If Docker or the WSL2 backend stops (e.g. Windows update, Docker crash), all containerized services (n8n, possibly fetch tools) will stop. Host–container networking can also fail (the agents may lose connectivity to `http://localhost:5678` for n8n). This halts all scheduling and automation. Similarly, internet outages disable any external API or web lookup (e.g. Tavily search), making certain features unavailable.

Shell Injection / Unsafe Commands (Execution Layer): If the shell MCP is too permissive, the LLM could execute destructive commands. For example, without filtering, a bad prompt might lead Claude to run `rm -rf C:\*`. The research warns that shell access must be *whitelisted* (only safe binaries like `python` or `npm`) [6]. A misconfiguration here is catastrophic: it gives the agent effectively full OS privileges. Even allowed commands can have side effects (network calls, disk changes), so uncontrolled shell use is high risk.

n8n Workflow Errors: The n8n engine itself may fail workflows. Scripts in n8n (e.g. API calls) can time out or return errors (e.g. "API limit" as in the standup example [23]). Because n8n is stateful per execution, errors within workflows can halt tasks silently. Since n8n is demoted to a "worker" role [24], such failures don't directly break the Kernel, but they break intended automations. Complex integrations (e.g. cyclic logic) can be particularly fragile [24]. If a workflow is malformed, Claude's `n8n_validate_workflow` should catch it before deployment, but run-time failures remain possible.

Data Loss or Drift: If data in n8n or external systems changes outside the Truth Layer, the system can drift. For example, a user might manually edit a file in the mounted Truth directory, or reset part of the n8n DB without updating the YAML config. Such *configuration drift* means the agent's model is wrong.

The docs state drift is "catastrophic" for trust [8]. Without a live drift-detection mechanism (observer), these discrepancies accumulate until the system behaves unpredictably.

These and other technical risks are summarized below:

Subsystem	Failure Mode	Impact	Notes / Examples
Claude (LLM)	Hallucination / context drift [2]	High (dangerous ops)	Agent generates wrong spec or action (e.g. destructive command) due to memory loss.
	Loss of API / model (no model)	High	No reasoning possible (system idle).
Filesystem MCP	I/O error / crash	High	Truth Layer inaccessible; agent can't read/write state. [2]
	Mis-scoped access (over-permissive)	High	Agent can modify system files (e.g. delete system32) [14].
Git MCP	Conflicts / locking error	High	Unable to commit; potential data overwrite.
	Corrupted repo	High	State loss, must recover from backup.
n8n Bridge / API	Unavailable / auth error	Medium	Claude can't inspect or deploy workflows.
n8n Engine	Crash / out-of-memory	Medium/High	All scheduled tasks fail; workloads not executed.
	Workflow bug (invalid data)	Medium	Single workflow fails, others unaffected.
Shell MCP	Unsafe command executed [6]	Catastrophic	Destructive shell command runs (security risk).
Web Fetch MCP	Connectivity or parse error	Low/Medium	External lookup fails; agent loses knowledge source.
Truth Layer	Split-brain / lost update [25]	High	Two actors overwrite each other's changes.
	Schema drift / invalid format [18]	High	Files break parsers or logic (unexpected types/fields).
Infrastructure	Docker/WSL crash or update	High	Entire system down until host is fixed.

4. Human / Cognitive / Process Failure Modes

- **Maintenance Overload:** A solo ADHD user can be overwhelmed by too many details or tasks. The research explicitly warns "maintenance is the enemy" for ADHD users [26]. Every required manual step (e.g. resolving conflicts, approving specs, performing standup) is a potential churn point. If the governance burden is high, the user may skip steps or abandon the system [26] [8].

- **Complexity Trap ("God Agent"):** As the system adds more tools and subsystems, cognitive load grows sharply [27] [28]. For example, exposing dozens of MCP tools floods Claude's context with irrelevant options [27]. Likewise, numerous YAML/JSON files and workflows can clutter the user's mental model. The docs call out the "God Agent" anti-pattern: too many tools degrade performance [27]. An ADHD mind is especially sensitive to clutter: presenting too many choices or technical details risks decision paralysis.

- **Impulsivity vs Pause:** The Chat→Spec→Change protocol enforces a "cognitive pause" [4], but if the user ever bypasses it (for speed or frustration), risky actions may follow. For instance, if a user impulsively instructs "delete old files," the agent must pause and spec that out. If the user ignores the spec or approves without reading (due to impatience), unintended deletions can occur. The system prompt has a rule to "present options clearly" and "always provide a 'Revert' option in specs" [29], recognizing that human oversight is fallible.

- **Standup Fatigue / Rituals:** The suggested "daily standup" routine [30] keeps the human in loop without overload. However, if the user skips the standup (or finds it too frequent), they may miss critical alerts (e.g. workflow failures). Conversely, making the standup too verbose would exhaust attention. Finding the right balance is tricky. Similarly, expecting the user to follow a rigid reset protocol [31] or manually investigate diffs might exceed their willingness.

- **Fragmented Attention:** ADHD implies short attention span. If the system's outputs are too verbose or low-level (stack traces, raw diffs), the user may get lost. The playbook advises "be concise, present options clearly" [29]. Ignoring this leads to cognitive overload. Also, if the agent forgets context (user was interrupted and returns later), the user must re-orient the system – a potential failure point. The system design must minimize such context-switch costs.

- **Trust & Abandonment:** If the agent makes errors or drift accumulates, the user's trust breaks down. The research notes that drift "degrades trust…leading to abandonment" [8]. An ADHD user who already struggles with executive function is likely to abandon a system that seems unreliable. Even perfectly safe errors (e.g. a needed validation prompt) can frustrate, so minimizing friction and providing easy fallbacks ("revert" tools, undo, backups) is crucial [8] [32].

- **Cognitive Load of Configuration:** There are many config details (MCP setup, YAML schemas, Docker mounts). Remembering all parameters and performing updates is error-prone. For example, missetting the allowed directories in `claude_desktop_config.json` can cause security holes [14]. These low-level tasks may be forgotten or mis-done by an inattentive user, causing vulnerabilities or failures later.

- **Dependency on Human in the Loop:** Ultimately, the user must approve specs, resolve conflicts, and reset the system when needed. During those moments, an ADHD user might rush or miss something. The framework does include governance gates (e.g. requiring a specific "CONFIRM" phrase for destructive actions [33]), but if the user overlooks instructions, the gate fails. Process complexity (multiple levels of confirmation) can itself trip up a distracted user.

5. Risk Rating & Prioritization

Failure Mode	Type	Impact	Likelihood	Priority	Notes
LLM hallucination / context drift [2]	Technical	High	High	High	Can lead to wrongful actions. Truth Layer mitigates this [2] but agent must use it correctly.
Shell injection / destructive commands [6]	Technical	Catastrophic	Medium	High	Unlimited shell access could destroy OS. Must whitelist safe tools only [6].
Truth Layer conflicts (lost updates) [25]	Technical	High	Medium	High	Concurrent writes by Claude and n8n can overwrite data. Custom locking/OCC needed [16] [17].
Data schema / drift errors [18]	Technical	High	Medium	High	Hallucinated fields or type changes break parsing. Schema enforcement needed [18].
n8n engine crash / unavailability	Technical	High	Medium	High	Stops all scheduled tasks. Recovery (restart/reset) needed.
Docker/WSL failure	Technical	High	Medium	High	Whole system down until host is fixed.
MCP server outage (fs/git)	Technical	High	Low/Med	Medium	Agent loses key capabilities temporarily.
Infinite loop / runaway execution [21]	Technical	Medium	Low/Med	Medium	Without limits, agents might loop on errors. LangGraph recursion limit and fallbacks must catch this [21].
Privacy / Security leak (credentials exposure)	Technical	Medium	Low	Medium	If scripts/tools mishandle keys or send data externally. (No direct source, but implicit risk.)

Failure Mode	Type	Impact	Likelihood	Priority	Notes
User cognitive overload / abandonment [26] [8]	Human	High	High	High	Complexity or drift breaks trust. “Maintenance is the enemy” [26] – if system is hard to maintain, user quits.
Impulsive skip of Spec step	Human/ Process	High	Medium	High	Bypassing the safe spec check invites errors. Strong vetting and clear prompts needed.
Configuration errors (MCP, Docker setup)	Human/ Process	Medium	Medium	Medium	Easy to misconfigure: e.g. wrong MCP_MODE causes parsing errors [22].
Standup/ritual fatigue	Human	Medium	Medium	Medium	If user ignores daily checks, then issue detection is delayed.
Excessive detail in output	Human	Medium	High	Medium	Too much log/flow detail will overwhelm ADHD user [29].

This unified table highlights **high-priority** risks: those that combine severe impact with likelihood. In particular, LLM hallucination and shell misuse are top technical risks, while **cognitive overload** and maintenance burden are top human risks. (All entries are informed by the research; for example, hallucination is explicitly noted as critical [2], and drift leading to abandonment is noted for ADHD users [8].)

6. Recommendations to Simplify / Shrink / Change Assumptions

- **Eliminate Unnecessary Agents:** Phase 2.3 should avoid multi-agent complexity. For now, **drop any cloud LLM (e.g. GPT) integration** and rely solely on Claude Desktop/MCP. This removes one class of failures (network/API issues, additional hallucinations) and reduces configuration overhead [34] [35].

- **Dual Truth Layer (Static+Observer):** Introduce an *observer-read-only* layer as suggested in the snapshot [36]. That is, keep the existing Git truth (static) plus a dynamic observed-state ledger. This simplifies drift handling: Claude can query a read-only summary of actual system state (event ledger or hash manifest) instead of trying to remember everything. In practice, have n8n or a script periodically record actual files/workflows (the “verified reality” [37]). Then let Claude generate change requests (CRs) to reconcile, rather than making live changes. This “Risk-Informed Observability” approach avoids changing state during detection and keeps the main truth static.

- **Merge Config Layers:** Currently configurations are split between YAML (user-facing) and JSON (machine) [38]. For simplicity, consider **unifying where possible**. For example, keep all core state in one format (e.g. YAML for easier editing) with JSON only for heavy n8n flows. Fewer file types means less cognitive overhead for the user.

- **Minimize Tools Exposed:** Remove any MCP tools that are not immediately needed. For instance, avoid broad shell access – restrict to a minimal safe subset [6]. Omit experimental tools (e.g. experimental web-scrapers) until proven essential. Fewer tools reduce context clutter [27].

- **Stricter System Prompts / Governance:** Harden the Chat→Spec→Change protocol: e.g. automatically **insert mandatory spec reviews** on every change, and require explicit "CONFIRM" for destructive actions [33]. The system prompt should keep outputs terse and always offer a one-click "revert" rollback for safety [29]. This guards against impulsivity and missing steps.

- **Simplify Rituals:** Keep rituals minimal. The Daily Standup [30] is useful; implement it automatically (Claude posts a brief report each morning) so the user need only glance and approve. Avoid extra meetings or detailed reports. Possibly add a single "Are you okay?" prompt at start of session to let user quickly resume context.

- **Flatten Automation:** Because n8n can't do complex loops [24], minimize multi-step n8n logic for now. Instead of building elaborate webhooks, have Claude handle logic loops and use n8n only for trivial sequential tasks (e.g. one API call or schedule trigger) [10] [24]. This "kernel-centric" approach reduces the pain of debugging n8n spaghetti.

- **Use Defaults & Convention:** Wherever possible, preset sensible defaults so the user rarely has to tweak settings. For example, predefine MCP config JSON with safe scopes [14], and create YAML templates for common workflows. This shrinks the surface for human error.

- **Limit Scope / MVP Focus:** Drop low-priority features. Focus on core use cases (e.g. email parsing, file organization). Defer advanced retrieval (LanceDB vectors) or deep learning components (DeepEval/E2B) – they add huge complexity [39] [21]. The next 6–12 months' architecture should be risk-informed: a small set of robust automations, rather than an ambitious multi-agent system.

- **Self-Healing Defaults:** Configure "nuclear" fallback procedures as defaults: e.g. always commit **before** any critical change [40], and possibly even auto-commit on errors. Provide an obvious one-click "reset to last commit" button or alias (like the reset protocol above [31]). Lower the friction of recovery to near zero.

Each suggestion above ties to the research: e.g. trimming toolsets addresses the **Kernel Architecture** complexity issues [27]; adding an observer/read-only mode is motivated by current-state drift notes [36]; minimizing rituals and output verbosity responds to **Cognitive Load** warnings [29] [8]; focus on stability over features echoes **n8n Reliability** guidance (n8n as simple worker) [24].

7. Suggested Implementation Roadmap

- **Critical Pre-Launch Fixes (Weeks 0–2):**
- *Drift Detection:* Implement a read-only observer process. For example, schedule an n8n job that nightly logs the current set of files and workflows, comparing to the Truth Layer (hashes or

listings) and alerting Claude to any mismatch [8]. This ensures latent drift doesn't undermine trust.

- *Spec Enforcement:* Configure the system prompt and interface so that **every change must be confirmed** via a spec review [4] [40]. Test that no file writes happen without a preceding spec diff and user approval.
- *Concurrency Guard:* Introduce a simple locking mechanism or version field for shared files [19]. At minimum, update MCP filesystem tools to check a `_version` field before writing, so a conflict aborts. This prevents lost updates.
- *Shell Safety:* Lock down shell access. Remove or blacklist dangerous commands (rm, del, etc.) in the shell MCP, and only allow known safe utilities [6].
- *Recovery Path:* Build a one-click reset script. For example, a batch or PowerShell script that does "`docker-compose down; git reset --hard; docker-compose up -d`" [31]. Ensure the user can run it easily. This makes "oh shit" recovery trivial.

- **Months 1–3 (Basic Features and Stability):**

- *Slice 1 & 2 (Bootstrapping & Self-Wiring):* Follow the playbook slices [41] [42]. Have Claude initialize the Truth repo and test n8n connectivity (create a simple workflow). Then build a recurring health-check workflow (as in Slice 2) that writes a log file to Truth. [41] [43]. This proves the full loop works end-to-end.
- *Core Automations:* Identify 2–3 high-impact, low-complexity automations (e.g. email-to-notes, backup check) and implement them. Validate each through the Chat→Spec→Change process and real tests. Focus on reliability, not many features.
- *User Prompts & Coaching:* Develop and refine system prompts. For instance, incorporate the "be concise" and "do not overwhelm" rules [29]. Create any necessary help text or tooltips so the user isn't guessing. Use the daily standup routine routinely to catch issues quickly [30].
- *Error Handling:* Add basic error-handling in flows. E.g. in LangGraph, set a reasonable `recursion_limit` and have failed nodes route back to the spec phase (as per LangGraph best practice [21]). Ensure that if Claude's proposed change fails validation (e.g. `n8n_validate_workflow` returns errors), Claude reports it and does not deploy.
- *Monitoring:* Start tracking simple KPIs. For example, count how often reconciling-run detects drift, or how many specs are undone by user. Use these to adjust priorities (if drift is frequent, focus more on observation).

- **Hardening & Later Phases (3+ Months):**

- *Simplifications from Research:* If still needed, consider adopting lightweight validation frameworks (e.g. Pydantic for spec schemas [44]) to catch field-type errors earlier. However, weigh this against added complexity.
- *Incremental Tool Safety:* Investigate integrating simple sandboxing for code (E2B) or verification loops (Reflexion) [45], but only after core stability is proven. For now, rely on git undo and testing in shell.
- *LangGraph Exploration:* If future needs grow, prototype a minimal LangGraph integration (Option B in the research) to handle complex loops [24]. But keep n8n as the main worker.
- *Cognitive Feedback:* After a few weeks of use, collect the user's feedback: which tasks feel most burdensome, which prompts confuse, etc. Use this to simplify the workflow or UI. For example, if spec diffs are too detailed, add a summary line highlighting only changes.

8. Limitations & Open Questions

- **User Usage Patterns:** We lack real data on which automations the user needs most. Future work should involve observing the user's actual tasks to prioritize features. (E.g., if the user never asks for web search, the docs-fetch tool is lower priority.)
- **Drift Handling Details:** We assume a read-only observer solves drift, but details (frequency of checks, how to present change requests) need testing. For instance, should Claude auto-commit certain benign fixes, or always ask user?
- **LangGraph Integration Trade-offs:** The research (Phase 3.0 architecture) suggests moving to LangGraph for complex workflows [24]. But LangGraph introduces heavy new complexity (state definitions, Python code). We must question whether it's worth it for a single user with limited time. Follow-up research could evaluate a lightweight agent framework or enhanced LangChain as an alternative.
- **n8n Dependence:** n8n is central for scheduling, but it's not truly "stateful" as an agent. If the user's needs evolve (e.g. complex logic or need for loops), we might hit n8n's limits [24]. The plan above assumes basic linear tasks suffice for now.
- **Failure of Observability:** Current tools (filesystem MCP) can't push events [46]. We rely on periodic polling or standups to detect issues. Is this enough? Could future work add file-system event watching (via WSL or native hooks) to raise alerts more promptly?
- **Evaluation of Cognitive Load:** We have heuristics from research, but how will this particular user handle the system? Ongoing testing is needed. For example, the recommended "concise" prompt style [29] might be too terse in practice or might omit needed context. A/B testing of prompt phrasing could be valuable.
- **Security vs Usability:** Strict whitelisting prevents most hazards [6], but may also block legitimate future commands (e.g. installing new tools). A mechanism for safely expanding the whitelist, or an emergency permission override, may be needed. This is a subtle policy decision that requires careful design.

In summary, this risk review—grounded in the user's own research—finds that the biggest threats are **state drift and cognitive overload**. The high-level recommendation is to simplify the architecture to essentials (Claude + a few MCP tools + Git) and automate drift-detection and recovery. Every change suggestion references the relevant research: e.g. reducing tools targets the "God agent" problem [27], and reinforcing the Chat→Spec→Change loop targets hallucination risk [2] [4]. The proposed roadmap prioritizes "safety wiring" (locking, spec gating, resets) before adding novel features. Following this plan should yield a lean, testable Life OS that a busy, distracted user can actually maintain and trust.

---

[1] [3] [6] [8] [9] [10] [12] [35] [46] 09_agentic_kernel_claude_desktop_mcp.md
file://file_000000002ec8722fa013c45a2f95ae21

[2] [4] [5] [7] [11] [13] [23] [26] [29] [30] [31] [32] [33] [38] [40] [41] [42] [43]
10_claude_desktop_agentic_kernel_playbook_windows.md
file://file_00000000e71071fb936a770503768c3d

[14] [15] [16] [17] [18] [19] [20] [25] 01_architectural-blueprint_phase-3-ai-os.md
file://file_0000000017a071f5a58aa89e7e9b58de

[21] [27] 04_personal-ai-life-os_implementation-strategy.md
file://file_0000000053fc71f581eb86a2d3623e37

[22] n8n-MCP.md
https://github.com/zf13883922290/TeyMCP-Server/blob/39915e5c78ed85a22be8ec1e4bdb868df3531f28/docs/n8n-MCP.md

[24] [28] [34] [37] 07_agentic-kernel_self-correcting-ai-life-os.md
file://file_000000002828722fb14797702a1964f0

[36] 08_ai_os_current_state_snapshot.md
file://file_000000007878722fb6f72ca2b0654c3f

[39] [44] [45] 03_personal-ai-life-os_orchestration-reliability-secure-execution.md
file://file_00000000f94871f58c2fc5181579d4ab