ChatGPT

Cortex-OS Management Dashboard for ChatGPT

Purpose

This design document proposes a **ChatGPT Dashboard** tailored for brAInwav's **Cortex-OS** platform. The goal is to create an accessible, low-friction interface that administrators can open inside ChatGPT to **grant access** and **manage Cortex-OS** operations. Unlike the earlier connector-focused widget, this dashboard monitors the health and activity of the Cortex-OS runtime, surfaces logs and traces, exposes core metrics, and gives operators fine-grained control over agents and workflows.

Context and Requirements

Cortex-OS is an Autonomous Software Behavior Reasoning (ASBR) runtime designed for multi-agent workflows with strong governance, security and observability [1]. Operators need visibility into:

- **Health** of the ASBR stack: Model Context Protocol (MCP), Agent-to-Agent (A2A) channels and REST endpoints [2].
- **Logs**: searchable, filterable by run ID, with live streaming for active tasks [2].
- **Traces**: visualisation of distributed spans and relationships [2].
- **Metrics**: counters, rates and durations across agents and workflows [2].
- **Controls**: ability to pause/resume runs, retry tasks, drain queues and run test flows [2].

Cortex-OS emphasises quality, security and observability [1], so the dashboard must provide **role-based access**, avoid exposing secrets [3] and honour accessibility requirements (keyboard navigation, screen-reader labels) [4]. Tasks should be **time-bound and summarizable** to respect the conversational context of ChatGPT apps [5], and the UI should be **conversational, simple, responsive and accessible** [6].

Primary Use Cases

1. **Grant access to Cortex-OS**: show a list of available environments/tenants and allow the operator to connect via OAuth 2.1/PKCE. Once authenticated, the user can browse panels.
2. **Monitor system health**: display live status of MCP/A2A/REST endpoints, agent heartbeat and memory service. Show a **TTL countdown** for any time-limited tokens or signatures to encourage timely refreshes.
3. **Browse logs and traces**: provide search and filters to locate logs by run ID, agent or severity. Allow drilling into a trace tree to inspect spans.
4. **View metrics**: summarise key counters (tasks executed, errors, active workflows) and rates (requests per second, latency). Show small sparkline graphs for trend spotting.
5. **Control agents and workflows**: list active agents and workflows, grouped by **tags or categories** (e.g., "core", "beta", "RAG"). Let users pause/resume, retry or drain them. Include a **search bar** to filter agents/workflows by name or tag.

6. **Accessibility & help**: provide keyboard shortcuts for panel navigation (`g` / `G` to move next/prev, `Enter` to open, `Esc` to close [4]). Include a `?` or `Ctrl-/` global help overlay explaining shortcuts and panel functions.

High-Level Layout

The dashboard adopts a **three-pane structure** that scales from mobile to desktop:

1. **Header bar** (horizontal across the top) – shows the brAInwav logo, current Cortex-OS environment, a global **search bar** and action icons (refresh, help, user menu). A subtle TTL countdown indicator appears next to the environment name when tokens are time-limited.

2. **Side navigation** (vertical on the left) – lists the main panels:

3. **Home** – overall summary with quick statistics and notifications.
4. **Health** – detailed endpoint and service status.
5. **Logs** – searchable log viewer.
6. **Traces** – distributed span explorer.
7. **Metrics** – charts of counters, rates and durations.
8. **Agents** – list of registered agents with status and controls.
9. **Workflows** – current and scheduled workflows (task state machine) with controls.
10. **Connectors** (optional) – shows external integrations via MCP; inherits TTL countdown and grouping by tags.
11. **Settings** – access controls, roles, preferences.

The side nav uses labelled icons and collapses to icons only on narrow screens. The active panel is highlighted and provides a visible focus ring for keyboard users.

1. **Main content area** – displays the selected panel. For example:

2. **Home**: four summary cards for Health (green/yellow/red indicator), Logs (recent error count), Agents (active vs. inactive) and Workflows (running/queued). Each card links to its panel.

3. **Health**: table of services with status (Up/Down/Degraded), response time, last heartbeat and **TTL countdown** if signed responses expire. Rows can be grouped by category (MCP, A2A, REST) and filtered via tags.
4. **Logs**: a toolbar with search and severity filters, followed by a virtualised list of log entries. Clicking an entry opens details in a side drawer. The search bar accepts free text or structured filters (e.g., `runId:1234`).
5. **Traces**: timeline or tree view of spans with collapsible branches. Selecting a span reveals metadata and associated logs.
6. **Metrics**: grid of metric cards (counters, rates, durations). Each card has a small sparkline; clicking it opens a larger chart and raw numbers. Metrics can be grouped by agent or workflow.
7. **Agents & Workflows**: each row shows name, status (running/idle/error), tags, start time and actions (pause/resume, retry, stop). A **search bar** and **tag filter** appear above the table. Group items by tag or category for easier navigation.

Panels should degrade gracefully: if no data is present, show informative empty states; when loading, use skeleton placeholders; when offline or error, show accessible error messages with remediation suggestions. All cards hide secondary sections when no data is present to keep the layout compact (e.g., only show the “Retry” button if a workflow is in error).

Interaction Flow

1. **Initial load:** the widget asks the user to select or authenticate to a Cortex-OS environment. After authentication, the Home panel loads summary cards.
2. **Viewing health:** clicking the Health nav item shows the status table. The user can click a row to expand details or press the **Refresh** icon in the header to fetch latest status; the TTL countdown resets accordingly.
3. **Searching logs:** the user selects Logs, types a query in the search bar and hits Enter (or presses `/` to focus the search field). The results update with a spinner; pressing `Enter` on a log row opens details.
4. **Drilling into a trace:** from the Logs panel, the user clicks a trace ID; the Traces panel opens with the tree view pre-filtered to that trace. The user can use `g` / `G` keys to navigate between spans.
5. **Controlling agents/workflows:** from Agents or Workflows panel, the user selects a row and chooses an action (Pause, Resume, Retry). The UI surfaces a confirmation prompt before performing any write actions (per security best practices).

Accessibility and Design Considerations

- **Conversational & summarizable:** tasks such as “pause workflow X” should complete in a single turn wherever possible and include summarised outputs. Long-running operations should return periodic status via follow-up messages.
- **Visual hierarchy:** rely on system colours, consistent spacing and typography as prescribed by the Apps SDK design guidelines [7]. Use accessible icons and avoid color-only signals by pairing icons with text or patterns [4].
- **Keyboard support:** provide shortcuts for panel navigation and actions (`g` / `G`, `Enter`, `Esc`) [4]. Ensure focus order follows the visual order and announce state changes via `aria-live`.
- **Role-based access:** hide controls that the current user is not allowed to perform; require confirmation for any external side-effects (pause, resume, drain). Do not expose secrets in UI [3].
- **Grouping & search:** allow grouping of items (agents, workflows, connectors) by tags or categories; provide a search/filter bar to quickly locate entries. This reduces cognitive load and helps complete tasks within the conversation boundary.
- **TTL countdown:** display a subtle countdown indicator (e.g., pill or progress bar) next to any resource with a time-to-live so the user knows when a refresh is needed.

Implementation Notes

- Build the dashboard as a React widget using the ChatGPT Apps SDK. Use the `window.openai` API for tool calls, follow-up messages and layout switches (inline vs. full screen). Persist UI state via `setWidgetState` to maintain filters and selections between conversations.
- Structure the server so that each panel corresponds to one or more tools: a **health** tool to fetch service status; **log** and **trace** tools to stream data; **metrics** tool to fetch counters and rates; **agent/**

workflow tools to list and control agents. Each tool should have a clear contract and descriptive metadata [5].
- Represent read-only operations with `readOnlyHint: true` and add explicit confirmation steps for write actions (pause, resume, retry). Use secure authentication (OAuth 2.1 + PKCE) and honour Cortex-OS standards.

Future Enhancements

- **Custom analytics:** integrate with the RAG system to surface retrieval metrics, such as documents fetched and cost per query.
- **Alerting:** allow users to configure notification thresholds (e.g., when error rates exceed 5%) and have ChatGPT proactively message them.
- **Audit trail:** provide a panel to view past administrative actions with timestamps and actors.
- **Plugin marketplace:** integrate connectors and third-party tools into the same dashboard so that all Cortex-OS resources are discoverable from one place.

This design aligns the ChatGPT Dashboard with Cortex-OS's operational needs and the ChatGPT Apps SDK's guidelines for conversational, accessible UIs [6]. It emphasises clarity, grouping, search and summarised tasks, providing operators with the tools they need to manage a complex multi-agent runtime while staying within the conversational flow.

---

[1] raw.githubusercontent.com
https://raw.githubusercontent.com/jamiescottcraik/Cortex-OS/main/README.md

[2] [3] [4] raw.githubusercontent.com
https://raw.githubusercontent.com/jamiescottcraik/Cortex-OS/main/apps/dashboard/README.md

[5] [6] [7] App design guidelines
https://developers.openai.com/apps-sdk/concepts/design-guidelines