ChatGPT

Arcadia Coach Debug – Diagnostic Findings & Recommended Fixes (Oct 2025)

This report is intended for the engineering team working on the Arcadia Coach desktop application. It synthesizes the debug logs and documentation to explain why the ChatKit widget and action buttons are not working and what steps to take to resolve them.

1 ArcadiaChatbot widget fails to render

Likely causes

1. **Server not reachable via MCP/ChatKit** – The widget depends on the `FastMCP` server running with the streamable HTTP transport and an accessible endpoint (e.g. `/mcp`). The **FastMCP** documentation stresses that when serving over the network you must run the server with `transport="http"` and provide `host`/`port` options; this turns the MCP server into an HTTP service at `http://<host>:<port>/mcp/`. Using legacy `sse` transport or placing `host`/`port` on the constructor is unsupported and leads to blank widgets.
2. **Wrong domain key / API URL** – ChatKit only renders when it trusts the domain of the embedding page. The ChatKit docs require registering your domain on OpenAI's **domain allow-list** and supplying the returned domain key as `domainKey` in the `api` configuration [1]. If the macOS app is pointing at a `render.com` domain without the correct `domainKey`, the widget silently refuses to load.
3. **Incorrect ChatKit server responses** – The server must return a **widget envelope** with valid components. The ChatKit Python server examples show how to stream widgets using `stream_widget()` or `stream_agent_response()` and note that only `<Text>` and `<Markdown>` elements with IDs support streaming updates [2]. If your server returns malformed JSON or tries to stream unsupported widgets, the client will display nothing.
4. **Missing environment variables** – If `OPENAI_API_KEY` or `ARCADIA_MCP_URL` are unset, the backend cannot call the OpenAI APIs or connect to the MCP server. The Agents SDK will fail silently or skip guardrail checks.

Recommended fixes

Step	Action
1. **Run the MCP server with the correct transport**	In your `mcp_server/server.py`, call `mcp.run(transport="http", host="0.0.0.0", port=int(os.environ.get("PORT", 8000)))`. This uses FastMCP's recommended streamable HTTP transport and binds to the port provided by Render. Remove unsupported constructor arguments (e.g., `version`, `host` or `port` passed to `FastMCP` itself) and rely on arguments to `run()`. Include a custom `@mcp.custom_route("/health")` endpoint that returns "OK" so you can test `/mcp/health` from your browser.
2. **Verify the endpoint and environment variables**	In the Render dashboard, set `ARCADIA_MCP_URL` to the fully qualified URL of your server (e.g., `https://arcadia-mcp.onrender.com/mcp`). Also set `OPENAI_API_KEY` and any agent IDs used by the backend. Restart the deployment.
3. **Register your domain and set the domain key**	Go to the OpenAI **domain allow-list** at `platform.openai.com/settings/organization/security/domain-allowlist`, add the `render.com` domain or whichever domain you use, and copy the generated domain key. In your macOS app's ChatKit configuration (likely stored in `frontend/src/lib/config.ts` or `Resources/ChatKit/advanced_chatkit.html`), set `CHATKIT_API_DOMAIN_KEY` (or `domainKey` in JavaScript) to this value. The ChatKit docs note that the domain key is required and must be registered [1].
4. **Point the client at the correct API URL**	Ensure `CHATKIT_API_URL` in `config.ts` or the macOS settings points at your MCP endpoint (e.g., `https://arcadia-mcp.onrender.com/mcp`). In local development, this defaults to `/chatkit` because Vite proxies requests; in production it must be overridden.
5. **Return valid widget envelopes**	Check the server's `respond` function. When the agent returns a widget, wrap it using `await stream_widget()` or `await stream_agent_response()` with proper JSON. Only `<Text>` and `<Markdown>` components with IDs are streamable [2]. Invalid structures will cause the widget to remain blank.
6. **Collect logs**	Run the macOS app from Xcode 26.1 beta 2, open Console.app, filter for subsystem `com.arcadiacoach.app`, and reproduce the issue. The JS→Swift bridge should forward `console.log`/`error` messages so you can see whether the ChatKit SDK is throwing registration errors or receiving invalid configuration.

2 Learn / Quiz / Milestone buttons do nothing

Likely causes: The `AgentService` uses the OpenAI Agents API to send messages and call tools. If `OPENAI_API_KEY` or agent IDs are missing on the server, the API calls will fail silently. The macOS UI now shows spinners and error alerts, but the backend must be correctly configured.

Recommended fixes

1. **Check environment variables on Render** – The Agents SDK requires `OPENAI_API_KEY` plus the specific **agent IDs** you created in the OpenAI dashboard. If these values are blank, the agent will not respond. Set them in Render and redeploy.
2. **Confirm backend routing** – The macOS settings must point at the Render backend (not `localhost`). If you used `VITE_CHATKIT_API_URL` for the chat widget, ensure the other API base URL (`FACTS_API_URL` or `AgentService` base) is also set. Hard-coded `localhost` URLs will not work in production.
3. **Inspect `AgentService` logs** – Use the OS log traces you added. Look for non-2xx HTTP status codes or JSON decode failures. If you see 401/403 responses, your API key may be invalid. If you see 404 or 405, your MCP server may be mounted incorrectly.

3 Backend ChatKit + Agents integration

1. **Validate environment variables** – Besides `OPENAI_API_KEY` and `ARCADIA_MCP_URL`, your server may expect variables such as `FACTS_DATABASE_URL`, agent IDs (e.g., `ARCADIA_AGENT_ID`), or keys for file storage. Any missing value may cause the server to skip guardrails or refuse to create sessions. Redeploy after setting them.
2. **Use streamable HTTP** – The Agents SDK expects the MCP server to support the new **streamable HTTP** transport. The Model Context Protocol spec says the server should expose a single endpoint that accepts POST (JSON-RPC batches) and optionally GET for SSE streaming. Validate your `/mcp` endpoint responds correctly to `POST` and streams events.
3. **Test MCP endpoints** – Use `curl` or Postman to call:
4. `GET https://arcadia-mcp.onrender.com/mcp/health` → should return `OK`.
5. `POST /mcp/tools/list` with an empty JSON batch (e.g., `[{"jsonrpc":"2.0","id":1,"method":"mcp.tools.list"}]`) → should return a JSON array of tools. If this fails, there is a server bug or misconfiguration.

4 MCP widget server deployment hiccups

1. **Use the correct `run()` call** – The FastMCP docs recommend starting HTTP servers via `mcp.run(transport="http", host="0.0.0.0", port=PORT)` and note that SSE is legacy. Passing unsupported arguments like `version` or `host`/`port` to `FastMCP()` will throw errors. Move those arguments to the `run()` call instead.
2. **Bind to Render's port** – Render injects a `PORT` environment variable. Use `int(os.environ['PORT'])` when calling `run()`. Setting `host="0.0.0.0"` ensures the server listens on all interfaces.
3. **Expose a health endpoint** – Use FastMCP's `@custom_route` decorator to add `/health` so the platform can check the service. Example:

```
from fastmcp import FastMCP
from starlette.responses import PlainTextResponse

mcp = FastMCP("ArcadiaServer")

```

```
@mcp.custom_route("/health", methods=["GET"])
async def health_check(request):
    return PlainTextResponse("OK")

if __name__ == "__main__":
    mcp.run(transport="http", host="0.0.0.0", port=int(os.environ["PORT"]))
```

1. **Redeploy and test** – After fixing the server, redeploy to Render. Verify `https://arcadia-mcp.onrender.com/mcp/health` returns 200 and that `ARCADIA_MCP_URL` in the backend is updated.

Next diagnostic steps (hand off to Codex)

1. **Capture logs** – Open Xcode 26.1 beta 2, run the macOS app, and keep **Console.app** open filtered on subsystem `com.arcadiacoach.app`. Reproduce the missing widget and button clicks. Share the logs (JS/Swift) with the backend team; they should show whether the ChatKit SDK is failing to register or whether the server returns malformed configuration.
2. **Validate environment variables** – Double-check all required variables in the Render dashboard. Set `OPENAI_API_KEY`, `ARCADIA_MCP_URL`, agent IDs, and domain key. Redeploy the backend.
3. **Confirm backend connectivity** – Launch `uvicorn` (or `uv run`) locally with a test key and confirm the macOS settings point to the reachable URL. Ensure the `Services/WidgetResource.arcadiaChatbotWidgetBase64()` function correctly reads the widget JSON and passes it to ChatKit. Inspect file sizes and logs to make sure the base64 payload is valid.
4. **Test MCP endpoints** – Use `curl` to call `/mcp/tools/list`, `/mcp/tools/call` and ensure they return expected widget envelopes. Compare the results to the ChatKit Python sample; adjust your server if necessary. Only when these endpoints work should you expect the widget and action buttons to function.

Conclusion

The blank ChatKit widget and unresponsive buttons are typically the result of misconfigured infrastructure rather than a bug in ChatKit itself. By running the MCP server with the correct transport, configuring environment variables, registering your domain and domain key, and returning valid widget envelopes, you can restore the Arcadia Coach chat experience. Capturing detailed logs will help confirm that the client receives valid configuration and errors are surfaced. Once these changes are applied, the ChatKit widget should render correctly and the Learn / Quiz / Milestone actions should invoke your agents successfully.

---

1 Custom backends | OpenAI Agent Embeds
https://openai.github.io/chatkit-js/guides/custom-backends/

2 Server Integration - Chatkit Python SDK
https://openai.github.io/chatkit-python/server/