

cs-aihelp

Handbuch

KI-Helpdesk für napp-it 4ai - Antworten auf Basis der lokalen Dokumentation, optionale Web-Recherche, lokales oder Cloud-LLM

Version 1.2.7 · 13. Oktober 2026

Inhalt

1. Überblick	3
2. Funktionen	3
3. Der Go-Daemon	4
4. Voraussetzungen und Installation	4
5. Schnellstart	5
6. Konfiguration	5
6.1 Konfigurationsschlüssel	6
6.2 Provider-Einrichtung	8
6.3 Web-Recherche	8
6.4 Chat-Verlauf	8
7. Bedienung	8
7.1 Das Chat-Menü	8
7.2 Das schwebende Popup	9
8. Sicherheitsmodell	9
9. Architektur	10
9.1 Einhängpunkte in csweb-gui	10
10. Tests, Release, Lizenz	10

1. Überblick

cs-aihelp ist ein chatbasierter **KI-Helpdesk**, der direkt in der napp-it CS Web-GUI läuft. Er beantwortet napp-it-Fragen auf Grundlage der lokalen Dokumentation ([data/howto.ai/*.info](#), Light-RAG), kann die Antwort durch eine Websuche ergänzen und lässt sich mit einem lokalen oder Cloud-LLM verbinden - mit oder ohne API-Key.

Er gehört zur napp-it 4ai (Client-Server-Edition) Cluster-Werkzeugfamilie. csweb-gui verteilt und aktualisiert den Go-Daemon pro Member über das Menü **About > Download cs-tools**.

Version 1.2.7 (stabil) - Lizenz BSD 2-Clause. Das Frontend läuft auf jedem napp-it CS Betriebssystem (FreeBSD, illumos, Linux, macOS, Solaris, Windows); der Go-Daemon wird für 8 Ziele gebaut. Lokale kostenlose KI (Ollama) läuft nativ unter Linux, macOS und Windows; unter FreeBSD, illumos und Solaris nutzt man den Pollinations-Rückfall, ein entferntes Ollama (`provider=ollama` plus Endpoint) oder einen OpenAI-kompatiblen lokalen Server (zum Beispiel llama.cpp).

Das Update auf 1.2.7 ist erforderlich. v1.2.6 und v1.2.7 beheben zwei Fehler aller früheren v1.2.x-Builds: Unter Windows führte die Konsolen-Ausführung (`exec_access=console`) die von der KI vorgeschlagenen Befehle nie tatsächlich aus (ConPTY-Zeilenende), und die Antworten von Reasoning-Modellen wurden auf fast nichts gekürzt (Antwort-Token-Mangel).

2. Funktionen

- **Free-Tier ohne Konfiguration:** `mode=free` funktioniert ohne Konto oder API-Key. Bevorzugt wird ein lokaler Ollama-Daemon, als Rückfall dient ein Pollinations.AI-GET.
- **Eigener Provider:** `mode=provider` für Anthropic, OpenAI (oder jeden OpenAI-kompatiblen Endpoint: vLLM, LM Studio, Groq, ...) und Ollama (ohne Key).
- **Kein stiller Rückfall:** Ein fehlschlagender `mode=provider`-Aufruf (nicht erreichbar, falscher Key, Timeout) wird als Fehler angezeigt und nicht heimlich von einem anderen, unkontrollierten Modell beantwortet (seit v1.2; der Schlüssel `fallback` wird nur noch aus Kompatibilitätsgründen gelesen).
- **Fundierte Antworten mit transparenten Quellen:** Light-RAG über [data/howto.ai/*.info](#) mit dateinamen-bewusstem Ranking, sodass eine Frage, die ein Thema nennt ("was ist cs-sync"), zuverlässig das passende Dokument findet. Jede Antwort hat zwei gekennzeichnete Abschnitte: **Local docs:** (streng an die gefundenen Auszüge gebunden, erfindet nie napp-it-Befehle, Pfade oder Einstellungen) und **General AI:** (das Hintergrundwissen des Modells).
- **Level 1 Diagnose nur lesend:** Optional wird der Live-Zustand (`hostname`, `zpool list`) des gewählten Members als Datenkontext angehängt.
- **Level 2 Exec (agentisch):** Mit `exec_access = exec|console` darf die KI Befehle *vorschlagen*. Die Ausführung wird durch eine Positivliste (`exec_allow`) und eine Sperrliste, die immer gewinnt (`exec_deny`), begrenzt, läuft nur nach Ihrer Bestätigung (`exec_mode=confirm`, Standard) in einem eigenen sitzungsgeschützten CGI, und die Ausgabe fließt zurück in das Gespräch - die KI kann Pools durchsuchen, analysieren und Probleme Schritt für Schritt beheben.
- **Web-Recherche:** `research=ddg` (DuckDuckGo Lite, ohne Key, Standard) oder `research=api` (jeder JSON-Such-Endpoint; die Antwortformen von Google CSE, Brave, Bing, SearXNG und Serper werden automatisch erkannt). Ergebnis-URLs erscheinen in der Quellenzeile.
- **Chat-Verlauf:** je Gespräch eine Datei `_cfg/aihelp/conv_*.json`, Aufbewahrung `off | today | week | month | 6months | all` (Standard `month`), jedes frühere Gespräch lässt sich fortsetzen. Das aktuelle Gespräch wird automatisch geladen, bis man **New** drückt.
- **Zwei Oberflächen:** die Vollbild-Chatseite (**Help > AI Helpdesk**) und ein verschiebbares, kontextsensitives Popup ("Ask AI") auf jeder angemeldeten Seite (`widget=on`).
- **Zwei Provider-Slots (plan/act):** Slot 1 = Plan- / Nur-Lese-Provider, Slot 2 = Act- / Exec-Provider; `plan` oder `act` wird im Helpdesk oder im Popup gewählt. Ein leeres `mode2` greift auf Slot 1 zurück.

- **Lokalisierte Oberfläche:** Formularelemente und kurze Hinweise bleiben Englisch; die Erläuterungen sowie die Texte in Helpdesk und Popup werden über die napp-it Sprachdateien übersetzt.

3. Der Go-Daemon

Seit v1.0 ist cs-aihelp ein **eigenständiger Go-Daemon** (reine Standardbibliothek, `CGO_ENABLED=0`, 8 Plattformen) mit eigenem Gedächtnis: einem RAG-Index im Speicher über `data/howto.ai/*.info`, einem gespeicherten Chat-Verlauf (gleiches Format wie im Perl-Frontend) und einer im Betrieb neu ladbaren Konfiguration. Er ist im Hintergrund immer bereit und über HTTPS auch von außen erreichbar.

```
cs-aihelp serve  [--listen 0.0.0.0:45555] [--config PFAD] [--foreground]
cs-aihelp start  [--config PFAD]      (abgekoppelt und idempotent; Boot-Hook)
cs-aihelp ask    --question "..." [--json]
cs-aihelp status [--json]
cs-aihelp stop   [--config PFAD]      (liest die PID-Datei des Daemons)
cs-aihelp reindex
cs-aihelp version
```

`serve` schreibt eine PID-Datei (Unix `/var/run/cs-aihelp.pid`, Windows neben der Konfiguration) und entfernt sie beim Beenden. `start` ist der Einstiegspunkt des Boot-Hooks (abgekoppelt, überspringt, wenn der Port schon belegt ist). `stop` funktioniert auf allen Plattformen (`taskkill` unter Windows).

Endpunkte (HTTPS, Bearer-Authentifizierung): `GET /health /status /sources?q= /models`, `GET /resume?member=...` (neuestes Gespräch des Members), `POST /ask` (JSON, oder `stream=true` für SSE) und `POST /reload /reindex`. Vom Browser werden sie über den sitzungsgeschützten Proxy `cs-aihelp.pl` erreicht.

Nur für den Daemon gültige Schlüssel (in derselben Datei `_cfg/cs-aihelp`):

Schlüssel	Beschreibung
<code>listen</code>	<code>127.0.0.1:45555</code> (Standard); <code>0.0.0.0:45555</code> für Zugriff von außen
<code>allowed_ip</code>	kommagetrennte IP/CIDR-Positivliste; leer = nur Loopback; * = beliebig
<code>auth_token</code>	Bearer-Token (Pflicht, wenn <code>listen</code> nicht Loopback ist)
<code>tls_cert / tls_key</code>	Standard = das Zertifikat von <code>webserver.pl</code> (<code>_cfg/webserver/cert/server.crt</code> und <code>.key</code>)
<code>cors_origin</code>	Frontend-Origin zur CORS-Festlegung (optional)
<code>ssrf_allow_private</code>	<code>no</code> oder <code>yes</code> - RFC1918-/private Endpoints erlauben (Ollama nur im LAN); Metadaten- und Link-Local-Adressen bleiben gesperrt
<code>rate_limit</code>	maximale Anfragen pro Minute je Client-IP im Daemon (0 = aus, Standard 60)

Sicherheit: IP-Positivliste und Bearer-Token-Prüfung in konstanter Zeit, ein SSRF-Schutz für alle konfigurierten Endpoints, TLS mit demselben Zertifikat wie die Web-GUI (entfernte Browser vertrauen ihm schon) und Konfigurationsdateien mit Modus `0600`.

Verteilung: `data/cs_server/tools/cs-aihelp/<os>.<arch>/cs-aihelp[.exe]`; gestartet wird er von `autostart.pl` zusammen mit `server.pl`, `auto.pl` und `monitor.pl`.

4. Voraussetzungen und Installation

- Eine installierte napp-it CS Web-GUI (Standardpfad `/opt/csweb-gui`; unter Windows `C:\opt\csweb-gui`).
- Perl mit den Kernmodulen `HTTP::Tiny` und `JSON::PP` sowie `URI::Escape` (`liburi-perl` unter Debian und Ubuntu; bei napp-it CS bereits enthalten).

- Optional für das Free-Tier: **Ollama** (`curl -fsSL https://ollama.com/install.sh | sh`, oder der Windows-Installer von ollama.com, danach `ollama pull llama3.1`). Der Free-Modus erkennt es auf `127.0.0.1:11434`. Für den Pollinations-Rückfall und die Web-Recherche wird Internetzugang benötigt.
- Optional für Cloud-Provider: ein API-Key von console.anthropic.com oder platform.openai.com.

Die Modul-Dateien werden mit folgendem Befehl in die napp-it CS Installation kopiert:

```
perl install.pl /opt/csweb-gui      # Standardziel ist /opt/csweb-gui
```

Datei	Installationsziel
<code>data/wwwroot/cgi-bin/cs-aihelp.pl</code>	sitzungsgeschützter Proxy zum Go-Daemon (Chat, SSE)
<code>data/wwwroot/cgi-bin/cs-aihelp-exec.pl</code>	Exec-CGI (Level 2)
<code>data/menues/_lib/windows/aihelpplib.pl</code>	gemeinsame Bibliothek
<code>data/menues/_lib/windows/cstoolslib.pl</code>	CS-Tools-Registry und GitHub-Download
<code>data/menues/05_Help/00_AI_Helpdesk/action.pl</code>	Chat-Menü (erster Eintrag in Help)
<code>data/menues/10_System/05_Services/12_AI_Helpdesk/action.pl</code>	Einstellungsmenü
<code>data/menues/_lib/tools/CS_Tools_Download/action.pl</code>	Download- und Update-Menü (einschließlich lokaler KI / Ollama)
<code>data/howto.ai/ai-helpdesk.info</code>	Moduldokumentation
<code>config/cs-aihelp.example -> _cfg/cs-aihelp</code>	Standardkonfiguration (nur wenn nicht vorhanden)

cs-aihelp ist nicht in napp-it CS enthalten. Das Daemon-Binary wird über **About > Download cs-tools** von GitHub geladen; bei der ersten Nutzung zeigt das Einstellungsmenü "please download CS tools first". Geladen und unter Beibehaltung der OS-Struktur installiert wird nur das Binary für das Frontend-Betriebssystem (`data/cs_server/tools/cs-aihelp/<platform>.<arch>/`), sodass sich `csweb-gui/data` auf ein anderes Betriebssystem kopieren lässt, wo das passende Binary aufgelöst wird. Dieselbe Seite bietet den Download und die Einrichtung der lokalen KI (Ollama) und einen Modell-Pull im Hintergrund; Start und Stop des Daemons liegen unter **System > Services > AI Helpdesk**. Für die Menüs ist kein Neustart nötig; die Konfigurationsdatei wird bei jeder Anfrage gelesen.

5. Schnellstart

1. Das Modul installieren (`perl install.pl`).
2. Die Web-GUI neu laden und anmelden.
3. Für ein zuverlässiges kostenloses Setup Ollama installieren: `curl -fsSL https://ollama.com/install.sh | sh` und `ollama pull llama3.1`.
4. **Help > AI Helpdesk** öffnen und eine Frage stellen. Es funktioniert sofort in `mode=free` (Ollama, oder der Pollinations-Rückfall, wenn Ollama nicht installiert ist).
5. Optional: **System > Services > AI Helpdesk**, um ein freies Modell, einen Cloud-Provider, die Web-Recherche und die übrigen Optionen zu wählen.

6. Konfiguration

Das Modul wird in **System > Services > AI Helpdesk** konfiguriert und in `_cfg/cs-aihelp` gespeichert, einer flachen `schlüssel = wert`-Datei, die beim ersten Start angelegt wird.

6.1 Konfigurationsschlüssel

Schlüssel	Werte (Standard)	Beschreibung					
mode	off / free / provider (free)	Slot 1 (plan/RO): free = ohne Key, lokales Ollama, dann Pollinations; provider = der konfigurierte Endpoint und ein optionaler Key					
provider	openai / anthropic / ollama (openai)	Slot 1, verwendet bei mode=provider					
endpoint	URL (leer)	Slot 1: vollständige Chat-Completions- / Messages-URL; leer = Standard des Providers					
model	Text (leer)	Slot 1: leer = Standard des Providers					
api_key	Text (leer)	Slot 1: nur Cloud-Provider; serverseitig gespeichert, nie geloggt					
mode2	leer / free / provider (leer)	Slot 2 (Act-/Exec-Provider); leer oder off = nutzt Slot 1					
provider2, endpoint2, model2, api_key2, free_model2		Provider-Einstellungen von Slot 2					
free_model	Text (leer)	Slot 1 mode=free: Tag des lokalen Ollama-Modells; leer = das erste verfügbare					
fallback	off	in v1.2 entfernt, nur noch aus Kompatibilität gelesen					
tool_use	no / yes (no)	Level 1: Live-Zustand nur lesend anhängen (hostname, zpool list)					
research	off / ddg / api (ddg)	Web-Recherche; ddg = DuckDuckGo Lite (ohne Key), api = externer Endpoint					

Schlüssel	Werte (Standard)	Beschreibung					
research_max	Zahl (5)	maximale Zahl der Suchergebnisse im Kontext					
research_endpoint	URL (leer)	research=api: URL-Vorlage mit {q} (oder automatisch ?q=)					
research_key	Text (leer)	research=api: optionaler Key (als Bearer / X-API-Key gesendet)					
history	off / today / week / month / 6months / all (month)	Aufbewahrung des Chat-Verlaufs					
history_turns	Zahl (10)	beim Fortsetzen als Kontext gesendete frühere Runden					
widget	on / off (on)	schwebendes Popup "Ask AI" (nur lesend) auf jeder angemeldeten Seite					
widget_input_lines	Zahl (1)	Fragefeld des Popups: 1 = eine Zeile, 2-10 = mehrzeiliges Textfeld					
widget_answer_height	Zahl (220)	Höhe des Antwortbereichs im Popup in px (100-1200)					
exec_access	ro / exec / console (ro)	Level 2: ro = nur lesend, exec = Positivliste, console = nur exec_deny gilt					
exec_mode	propose / confirm / auto (confirm)	propose = nur anzeigen, confirm = jede Ausführung braucht einen Klick, auto = keine Bestätigung je Schritt					
exec_allow	Kommaliste (leer)	Befehlsklassen / Präfixe, die ausgeführt werden dürfen (erstes Wort der Befehlskette); leer = nichts wird ausgeführt					

Schlüssel	Werte (Standard)	Beschreibung					
<code>exec_deny</code>	Pipe-Liste	immer angewandt, gewinnt gegen <code>exec_allow</code> ; Standard <code>`zfs destroy</code>	<code>zpool d destroy</code>	<code>rm -rf</code>	<code>dd</code>	<code>mkfs</code>	<code>format</code>
<code>autostart</code>	<code>on</code> / <code>off</code> (<code>on</code>)	den Go-Daemon bei jedem Start von <code>server.pl</code> starten (wenn <code>mode != off</code> und das Binary vorhanden ist)					
<code>max_context</code>	Zahl (<code>8000</code>)	Budget des System-Prompts in Zeichen					

6.2 Provider-Einrichtung

Option	Was es ist	Key nötig
<code>free</code>	lokales Ollama (zuverlässig, privat) mit Pollinations.AI-GET-Rückfall (sofort, experimentell)	nein
<code>provider = ollama</code>	lokales Ollama über eine explizite Provider-Konfiguration	nein
<code>provider = anthropic</code>	Claude (beste Qualität)	ja (console.anthropic.com)
<code>provider = openai</code>	OpenAI oder jeder OpenAI-kompatible Dienst	ja (bei lokalen Servern keiner)

Datenschutz: `free` (Pollinations) und die Cloud-Provider senden die Frage (einschließlich des Tool-Use-Kontexts) an einen externen Dienst. In sensiblen Umgebungen sollte Ollama lokal genutzt werden.

6.3 Web-Recherche

- `research=ddg` (Standard) nutzt lite.duckduckgo.com - kostenlos, ohne Key, direkter Internetzugang. Direktes Scrapen von Google wird von Google blockiert; für Google also `research=api` mit einem Google-CSE-Key verwenden.
- `research=api` ruft jeden JSON-Such-Endpoint auf, zum Beispiel `research_endpoint = https://www.googleapis.com/customsearch/v1?key=...&cx=...&q={q}`. Der Platzhalter `{q}` wird durch die Frage ersetzt; fehlt er, wird `?q=` angehängt. Die Antwortformen von Google CSE, Brave, Bing, SearXNG und Serper werden automatisch erkannt.

6.4 Chat-Verlauf

Gespräche werden in `_cfg/aihelp/conv_<zeitstempel>_<id>.json` gespeichert (`{created, updated, member, title, messages[]}`). Gespeichert werden nur Frage-Antwort-Paare - nie System-Prompts oder Geheimnisse. Die Bereinigung läuft bei jeder Anfrage verzögert nach `history`. Beim Fortsetzen werden die letzten `history_turns` Runden als Kontext an das Modell gesendet.

7. Bedienung

7.1 Das Chat-Menü

Help > AI Helpdesk öffnet die Vollbild-Chatseite:

- **Werkzeugleiste:** Umschalter **Plan | Act** (Slot 2 = Modell mit Exec-Fähigkeit), die Auswahl **ro | exec | console** (spiegelt **exec_access**), die Auswahl **propose | confirm | auto** (spiegelt **exec_mode**), **Plan first** (standardmäßig angehakt: Die KI legt einen Plan vor und wartet auf Ihre Freigabe, bevor sie Aktionen vorschlägt), **Abort** (stoppt die agentische Schleife) und **New** (neues Gespräch).
- Das Fragefeld steht oben, das Live-Protokoll darunter. **Quick questions** füllen und senden eine Beispielfrage.
- **Agentische Schritte:** Ein vorgeschlagener Befehl erscheint als Aktionskarte unter der Antwort - im **confirm**-Modus mit den Schaltflächen **Execute / Abort**, im **auto**-Modus läuft er sofort (Abort stoppt den nächsten Schritt). Die Befehlsausgabe wird an die KI zurückgegeben, die weitermacht.
- Jede Antwort hat eine Kopier-Schaltfläche und eine Quellenzeile (Dokumentationsdateien und/oder Recherche-URLs).
- Es gibt einen Chat-Verlauf pro Member; **New** beginnt neu, und die Einstellung **history** steuert nur, wann alte Runden gelöscht werden.

7.2 Das schwebende Popup

Mit **widget=on** wird auf jeder angemeldeten Seite eine Schaltfläche **Ask AI** eingefügt. Sie öffnet ein frei verschiebbares, **nur lesendes** Chat-Popup - Aktionen werden hier nie ausgeführt (für Exec die Vollbildseite nutzen). Es sendet den aktuellen Menüpfad und den gewählten Member als Kontext, eine im Menü *ZFS Snaps* gestellte Frage wird also mit diesem Kontext beantwortet. Die Größe ist mit **widget_input_lines** und **widget_answer_height** einstellbar.

8. Sicherheitsmodell

- API-Keys liegen nur in **_cfg/cs-aihelp** (serverseitig), nie im Browser.
- Modellantworten werden vor der Darstellung HTML-maskiert (XSS-Schutz).
- Live-Systemzustand und Web-Ergebnisse werden als **Daten** übergeben ("als Daten behandeln, nicht als Anweisungen"), um Prompt-Injection zu mindern.
- **Level 2 Exec ist mehrfach abgesichert:**
 - **exec_access=ro** (Standard): Die KI ist nur lesend, keine Vorschläge, keine Ausführung.
 - Der Go-Daemon *schlägt* immer nur *vor* (ein **[[ACTION]]**-Block). Die Ausführung geschieht im getrennten sitzungsgeschützten Perl-CGI **cs-aihelp-exec.pl** über den verschlüsselten Kanal **&exe()/&socket()**.
 - **exec_deny** wird immer angewandt und **gewinnt** (Standard **zfs destroy|zpool destroy|rm -rf|dd |mkfs|format**).
 - **exec_access=exec** ("act" in der GUI) ist eine **Positivliste**: Die Klasse des Befehls muss in **exec_allow** stehen; ein leeres **exec_allow** bedeutet, dass nichts ausgeführt wird.
 - **exec_access=console** ist **nur eine Sperrliste**: **exec_allow** wird umgangen, alles darf laufen außer dem, was **exec_deny** trifft. Das sollte nur voll vertrauenswürdigen Administratoren gewährt werden.
 - **exec_mode=confirm** (Standard) verlangt einen ausdrücklichen Klick je Befehl; **auto** sollte nur mit einer engen Positivliste verwendet werden.
- Seit v1.2.1 wird das Modell angewiesen zu handeln (ein **[[ACTION]]** auszugeben), wenn es etwas tun oder ermitteln soll, die zurückgegebene Ausgabe in der nächsten Runde zu lesen und weiterzumachen.
- Seit v1.2.2 schlägt das Modell nie einen Installationsbefehl ohne seine nicht-interaktive Option vor (**apt-get install -y, winget install --silent**, ...). Als Absicherung umhüllt **server.pl** jeden Exec-Pfad mit **IPC::Run** und einem 60-s-Timeout, der ein hängendes Kindprogramm beendet.
- Der Chat-Verlauf enthält nur Frage-Antwort-Paare (keine Geheimnisse) und wird nach der Aufbewahrungseinstellung bereinigt.

9. Architektur

```
Browser (Menüseite oder Popup)
| POST /cgi-bin/cs-aihelp.pl (Sitzungsprüfung + nur lesender live_state)
v
cs-aihelp.pl (Perl-Proxy, Loopback, HTTPS Webserver-Zertifikat / Bearer auth_token)
| /ask (gepuffert oder SSE stream:true)
v
cs-aihelp (Go-Daemon, 127.0.0.1:45555)
| RAG-Index im Speicher (data/howto.ai/*.info)
| optionale Web-Recherche (ddg / api)
| Provider-Aufruf: free (Ollama -> Pollinations) | provider (anthropic/openai/ollama)
| Chat-Verlauf (_cfg/aihelp/conv_*.json), /resume
v
Browser: Token als SSE gestreamt oder gepuffertes JSON; Antwort ("Local docs:" / "General AI:") + Quellen
```

Das Perl-Frontend (Menüs, Einstellungen, Popup, Chatseite) ist eine dünne Schicht: Es schreibt `_cfg/cs-aihelp`, sammelt den nur lesenden Live-Zustand und den Menükontext und leitet jedes `/ask` über Loopback an den Go-Daemon weiter. Der Exec-Kanal bleibt im sitzungsgeschützten Perl-CGI.

Agentische Schleife Level 2: Der Browser führt die Schleife, der Daemon antwortet nur. Die Frage geht über den Proxy an `/ask`; die Antwort kann ein `[[ACTION]]` mit Befehl und Begründung enthalten. Im Confirm-Modus klickt der Benutzer Execute, im Auto-Modus läuft es automatisch; `cs-aihelp-exec.pl` prüft den Befehl gegen Positiv- und Sperrliste und führt ihn über den verschlüsselten Kanal aus; die Ausgabe geht als `tool_results` zurück an `/ask`, und die Schleife läuft weiter. Die Ausführung geschieht nie im Go-Daemon - er ist nur Vorschlagender.

9.1 Einhängpunkte in csweb-gui

Vier UI- und Boot-Einhängpunkte berühren Kerndateien von napp-it und werden auf der napp-it-Seite angewandt (sie gehören nicht zur Dateiliste von `install.pl`):

- die **Statusampel** (KI-Punkt) in `get_async.pl` und `admin-wlib.pl`,
- die **Popup-Einbindung** im Header von `interface.pl` (`ai_popup(...)`) und in `admin.pl`,
- der **Daemon-Autostart** in `server_boot_tasks.pl` (`ai_boot_autostart()` bei jedem Start von `server.pl`),
- die **Lokalisierung** über die Dateien `lang/*/ai_helpdesk.txt` und `lang/*/help.txt` sowie die beiden Menüschlüssel in `about_menus.txt`.

10. Tests, Release, Lizenz

```
tests/run_tests.sh # Linux / macOS / illumos
powershell -ExecutionPolicy Bypass -File tests/run_tests.ps1 # Windows
```

Die Perl-Suite startet einen lokalen Mock-HTTP-Server (OpenAI-kompatibler Chat, Ollama, DuckDuckGo-Lite-HTML, Google-CSE-JSON) und führt 49 Funktionsprüfungen aus: Konfigurations-Rundlauf, Provider-Auflösung, RAG-Suche, Verlauf speichern/laden/auflisten/bereinigen, Provider-Aufrufe und Fehlerpfade, den Free-Tier-Ollama-Pfad, Recherche-Auswertung, den SSRF-Schutz und den Schlüssel `log`. Der Go-Daemon hat eigene Unit-Tests (`go test ./...`).

GitHub Actions baut das Release bei einem `v*`-Tag: `ci.yml` (Go vet und test plus die Perl-Suite bei jedem Push) und `release.yml` (der Go-Daemon für 8 Plattformen als `cs-aihelp-<os>.<arch>.tar.gz` plus das Modul-Archiv, mit `checksums.txt`).

Lizenz: BSD 2-Clause - Copyright (c) 2026 Guenther Alka / napp-it.org.