WAIT

KI-gestütztes IT-Helpdesk

Projektdokumentation

Was WAIT ist, wie es funktioniert und wie man es weiterentwickelt

Stand: November 2026

Repository: `github.com/I3MC-eu/wait` (privat) · Installation: Installationsanleitung · AD-Anbindung: AD-Login-Anleitung · Bedienung: Benutzeranleitung

Inhalt

1. Was WAIT ist 3
1.1 Wichtigste Funktionen 3
2. WAIT benutzen 4
2.1 Für Schüler:innen und Lehrkräfte 4
2.2 Für das IT-Team 4
3. Architektur 5
4. Wie der Assistent funktioniert 6
4.1 Lebenszyklus eines Tickets 6
4.2 Zwei Modi: geskriptet und KI 6
4.3 Bilder 6
4.4 Übergabezusammenfassung 6
5. Datenmodell 7
5.1 Kategorien und Prioritäten 7
6. API-Überblick 8
7. Sicherheit 9
8. Konfigurationsreferenz 10
9. Entwicklung 11
9.1 Aufbau des Repositorys 11
9.2 Lokal ausführen 11
9.3 Tests und Prüfungen 12
9.4 Änderungen vornehmen 12
10. Bekannte Grenzen und nächste Schritte 13

1. Was WAIT ist

WAIT ist ein Helpdesk für die IT-Abteilung einer Schule. Schüler:innen und Lehrkräfte melden Probleme als Tickets. Der **WAIT-Assistent** antwortet sofort mit Hilfe einer Wissensdatenbank aus Schritt-für-Schritt-Anleitungen und eines lokalen KI-Modells. Kann er das Problem nicht lösen, übergibt er das Ticket an das IT-Team, das die übrigen Tickets im Admin-Bereich bearbeitet.

Ziel: Häufige Probleme der ersten Ebene (WLAN, Passwörter, Drucker, Beamer, Teams) werden in Minuten ohne Techniker:in gelöst, und das IT-Team bekommt jedes eskalierte Ticket mit einer Zusammenfassung statt eines langen Chats.

1.1 Wichtigste Funktionen

- **Hilfe-Center zur Selbsthilfe**: 62 Anleitungen auf Deutsch und Englisch, durchsuchbar und nach Kategorien geordnet, mit Daumen-hoch/runter-Bewertung.
- **Tickets mit KI-Assistent**: Er schlägt die passende Anleitung vor, führt mit Buttons Schritt für Schritt hindurch und beantwortet freie Fragen mit einem lokalen Sprachmodell, das sich auf die Wissensdatenbank stützt.
- **Klare Kennzeichnung der KI**: Ein Hinweis im Chat und eine Markierung „KI-Assistent · automatisch erstellt“ an jeder KI-Nachricht zeigen, dass gerade kein Mensch antwortet.
- **Bilder im Chat**: Screenshots anhängen, einfügen (Strg+V) oder hineinziehen; ein Bildmodell liest Fehlermeldungen und -codes daraus.
- **Übergabe an die IT** mit KI-Zusammenfassung: Problem, bereits versucht, wichtige Details, empfohlener nächster Schritt.
- **Admin-Bereich**: Ticket-Warteschlange mit Filtern, Zuweisung, Priorität/Kategorie/Status, interne Notizen, Löschen, Nutzerliste, Team-Verwaltung, Editor für die Wissensdatenbank und Statistiken.
- **CSV-Export** der gefilterten Ticketliste und der Statistiken, direkt in Excel lesbar.
- **Anmeldung über Active Directory** mit dem Schulkonto; Admin-Rechte über AD-Gruppen oder im Admin-Bereich.
- **HTTPS** mit gültigem Zertifikat unter `https://wait.i3mc.fun`.
- **Zweisprachig** (DE/EN) in Oberfläche und Assistent, helles und dunkles Design.
- **Läuft komplett vor Ort**: Keine Daten verlassen die Schule, die KI-Modelle laufen auf der schuleigenen GPU.
- **Produkt-Website** mit Beschreibung und Preisen.

2. WAIT benutzen

Die ausführliche Bedienung mit Screenshots steht in der **Benutzeranleitung**. Kurzfassung:

2.1 Für Schüler:innen und Lehrkräfte

1. Unter `https://wait.i3mc.fun` mit dem Schulkonto anmelden (Benutzername oder E-Mail und Schul-Passwort).
2. Zuerst im **Hilfe-Center** suchen. Jede Anleitung endet mit „Immer noch nicht gelöst?“; das öffnet ein Ticket und sagt dem Assistenten, welche Anleitung schon versucht wurde.
3. **Neues Ticket**: Betreff, Kategorie, Priorität, Beschreibung, optional Anhänge. Der Assistent antwortet in wenigen Sekunden.
4. Den vorgeschlagenen Schritten mit **Erledigt, nächster Schritt** folgen, bei Erfolg **Das hat geholfen** wählen oder mit **An die IT weitergeben** eskalieren.
5. Fragen frei eintippen oder Screenshots schicken; der Assistent liest sie.
6. **Meine Tickets** zeigt alle eigenen Tickets und ihren Status. Gelöste Tickets lassen sich wieder öffnen.

2.2 Für das IT-Team

Konten mit der Rolle `admin` sehen im Menü den **Admin-Bereich** mit diesen Seiten:

Seite	Zweck
Tickets	Warteschlange der offenen Tickets nach Priorität, mit Suche, Filtern (Status, Priorität, Kategorie, Zuständigkeit) und CSV-Export.
Ticketdetails	KI-Übergabezusammenfassung, der ganze Chat, interne Notizen (gelb, nie für die Person sichtbar), Anhänge mit KI-Beschreibung, Zuweisen, Priorität, Kategorie, Lösen, Wieder öffnen, Löschen.
Automatisch gelöst	Tickets, die der Assistent ohne das IT-Team gelöst hat.
Nutzer:innen	Alle Personen mit erstellten und gelösten Tickets und ihrem letzten Ticket.
Team	Lokale Admin-Konten anlegen und Rollen ändern. AD-Admins sind mit „über AD“ markiert und nur im AD änderbar.
Wissensdatenbank	Anleitungen in beiden Sprachen anlegen und bearbeiten (Titel, Zusammenfassung, Text, Schritte, Suchbegriffe).
Auswertung	Statistiken: offene und nicht zugewiesene Tickets, KI-Lösungsquote, Eskalationen, Zeit bis zur ersten Antwort und bis zur Lösung, Tickets nach Status und Kategorie; CSV-Export.
Einstellungen	Nur lesen: API-Status, erlaubte Registrierungs-Domains, Passwortregel, Upload-Grenze.

Sobald eine Techniker:in ein Ticket zuweist oder antwortet, antwortet der Assistent in diesem Ticket nicht mehr. Interne Notizen sind nur für Kolleg:innen.

3. Architektur

WAIT ist eine klassische dreischichtige Webanwendung mit lokalem KI-Server, verpackt mit Docker Compose:

```
Browser ──► caddy (HTTPS :443) ──► frontend (nginx) ──/api──► backend (FastAPI :8000)
──► db (PostgreSQL)
                                    │ React-App                │
                                    │                          ├──► ollama (:11434,
GPU)
Browser ──► caddy (:80) ──► site    Produkt-Website            │      Textmodell +
Bildmodell
                                                               └──► Active Directory
(LDAPS :636)
```

Teil	Technik	Ordner
Frontend	React 19, Vite, Tailwind CSS, React Router	`wait-frontend/`
Backend	Python 3.12, FastAPI, SQLAlchemy 2, Alembic, Pydantic, ldap3	`wait-backend/`
Datenbank	PostgreSQL 17	Docker-Volume `db-data`
KI	Ollama mit `qwen2.5:7b` (Text) und `qwen2.5vl:3b` (Bild)	Docker-Volume `ollama-data`
HTTPS	Caddy mit Cloudflare-DNS-Plugin, Let's Encrypt	`~/wait/https/` (nur auf dem Server)
Website	Statisches HTML/CSS über nginx	`wait-site/`
Betrieb	Docker Compose, NVIDIA Container Toolkit	`docker-compose.yml`

4. Wie der Assistent funktioniert

4.1 Lebenszyklus eines Tickets

Status	Bedeutung
`submitted`	Gerade erstellt; der Assistent analysiert es im Hintergrund.
`analyzed`	Der Assistent hat mit einer vorgeschlagenen Anleitung geantwortet.
`chatting`	Gespräch läuft.
`escalated`	An die IT übergeben (durch die Person, den Assistenten oder eine Zuweisung). Der Assistent schweigt.
`resolved`	Gelöst. `resolved_by` hält fest, ob durch KI, IT-Team oder die Person selbst.

4.2 Zwei Modi: geskriptet und KI

Das Durchgehen der Schritte ist geskriptet, damit es sich immer gleich verhält: Beim Erstellen wählt das Backend die passendste Anleitung (Schlüsselwort-Ranking über Titel, Suchbegriffe, Zusammenfassung und Text, gefiltert nach Sprache und Kategorie, ohne eine bereits versuchte Anleitung) und bietet ihre Schritte einzeln mit Buttons an. Gehen die Schritte aus, versucht es eine Alternative und eskaliert dann.

Freie Nachrichten gehen an das Sprachmodell. Für jede Nachricht:

1. Holt das Backend die zwei passendsten Anleitungen (der „Knowledge Agent“) und setzt sie in den System-Prompt.
2. Wurden Bilder geschickt, ergänzt es die Nachricht um einen Vermerk `[Bild Name: Beschreibung]`. Die Beschreibung stammt vom Bildmodell beim Hochladen; hat die Person zusätzlich eine Frage gestellt, bekommt das Bildmodell auch diese Frage zum Bild.
3. Schickt es System-Prompt, die letzten acht Chat-Runden und die neue Nachricht über Ollama an `qwen2.5:7b`.
4. Ist Ollama nicht erreichbar, antwortet es geskriptet, damit der Chat nie abbricht.

Jede Ausgabe des Modells läuft durch einen Filter, der Gedankenstriche entfernt; der Prompt verbietet sie zusätzlich.

4.3 Bilder

Erlaubt sind PNG, JPG, GIF und WebP bis 10 MB, fünf pro Nachricht. Bevor das Bildmodell ein Bild sieht, verkleinert das Backend es auf höchstens 1024 Pixel: Bei Full-HD-Bildern geriet das Modell in eine Schleife, bis Ollama abbrach. Meldet das Modell trotzdem einen Fehler, lädt das Backend es neu und versucht es einmal erneut.

4.4 Übergabezusammenfassung

Wird ein Ticket `escalated`, bittet eine Hintergrundaufgabe das Textmodell um eine vierteilige Kurzfassung in der Sprache des Tickets: **Problem**, **Bereits versucht**, **Wichtige Details** (Fehlercodes, was Screenshots zeigen) und **Empfohlener nächster Schritt**. Sie wird am Ticket gespeichert, nur dem IT-Team gezeigt und lässt sich auf der Ticketseite neu erzeugen.

5. Datenmodell

Tabelle	Inhalt
`users`	Name, E-Mail, bcrypt-Passwort-Hash (nur lokale Konten), Rolle (`customer` oder `admin`), Sprache, `auth_source` (`local` oder `ldap`), `directory_admin` (Admin über AD-Gruppe), `panel_admin` (im Admin-Bereich befördert).
`tickets`	Betreff, Kategorie, Priorität, Beschreibung, Status, Sprache, Ersteller:in, Zuständige:r, versuchte/vorgeschlagene Anleitung, aktueller Schritt, Zeitstempel, `resolved_by`, `handoff_summary`.
`messages`	Chat-Nachrichten: Absender (`user`, `assistant`, `staff`), Text, Schnellantwort-Buttons, Kennzeichen „intern“, Autor:in.
`attachments`	Hochgeladene Dateien: Name, Typ, Größe, Dateiname auf dem Server, KI-Beschreibung, zugehörige Nachricht.
`articles`	Anleitungen: Slug + Sprache (gleicher Slug = gleiche Anleitung auf DE/EN), Kategorie, Zusammenfassung, Text, Schritte, Suchbegriffe.
`article_feedback`	Daumen-Bewertung pro Person und Anleitung.

Die Rolle ergibt sich bei AD-Konten bei jeder Anmeldung aus `directory_admin` oder `panel_admin`. So überlebt eine Beförderung im Admin-Bereich die nächste Anmeldung, und ein AD-Admin kann im Admin-Bereich nicht herabgestuft werden (das würde AD beim nächsten Login rückgängig machen).

Schemaänderungen erfolgen in `wait-backend/app/models.py` und werden als Alembic-Migrationen in `wait-backend/alembic/versions/` festgehalten. Migrationen laufen beim Start des Backends automatisch.

5.1 Kategorien und Prioritäten

Kategorien: Erste Schritte, Netzwerk & VPN, Konto & Zugang, Hardware, Software, Drucken, E-Mail & Kommunikation, Sicherheit, Klassenzimmer-Technik, Sonstiges. Prioritäten: Niedrig, Mittel, Hoch, Dringend. Beides ist einmal in `app/categories.py` definiert, zusammen mit den Statusbezeichnungen für den Export.

6. API-Überblick

Alle Endpunkte liegen unter `/api` und nutzen JSON mit einem `Bearer`-Token aus der Anmeldung. Eine interaktive Dokumentation gibt es unter `http://<server>:8000/docs`.

Bereich	Endpunkte
Anmeldung `/api/auth`	`GET /config`, `POST /register`, `POST /login`, `GET/PATCH /me`, `POST /change-password`
Tickets `/api/tickets`	`GET /` (eigene Tickets), `POST /` (erstellen), `GET /{id}`, `POST /{id}/messages`, `POST /{id}/attachments`
Anhänge	`GET /api/attachments/{id}` (Download, nur Ersteller:in oder IT-Team)
Wissen `/api/articles`	`GET /`, `GET /categories`, `GET /{slug}`, `GET/POST /{slug}/feedback`
Wissen bearbeiten `/api/admin/articles`	`GET /`, `POST /`, `PUT /{id}`, `DELETE /{id}` (nur IT-Team)
IT-Team `/api/admin`	`GET /tickets`, `GET /tickets/export`, `GET/PATCH/DELETE /tickets/{id}`, `POST /tickets/{id}/messages`, `POST /tickets/{id}/summary`, `GET /stats`, `GET /stats/export`, `GET /customers`, `GET/POST /users`, `PATCH /users/{id}/role`, `GET /settings`

Die beiden Export-Endpunkte liefern CSV (UTF-8 mit BOM, Trennzeichen `;` bei Deutsch, `,` bei Englisch) und nehmen dieselben Filter wie die Ticketliste sowie `lang=de|en`. Admin-Rechte werden für jeden `/api/admin`-Endpunkt auf dem Server geprüft, nicht nur in der Oberfläche versteckt.

7. Sicherheit

- HTTPS mit Let's-Encrypt-Zertifikat; `http://wait.i3mc.fun` leitet auf HTTPS um.
- AD-Anmeldung über LDAPS mit Zertifikatsprüfung; AD-Passwörter werden nie gespeichert.
- Lokale Passwörter mit bcrypt gehasht, mindestens 12 Zeichen.
- JWT-Anmelde-Tokens (24 Stunden), signiert mit `JWT_SECRET`.
- Begrenzung der Anmeldeversuche: 5 Fehlversuche pro IP und Konto sperren diese Kombination für 15 Minuten.
- Optionale Selbstregistrierung nur für die E-Mail-Domain der Schule (`ALLOWED_EMAIL_DOMAINS`).
- Personen sehen nur ihre eigenen Tickets; interne Notizen und Übergabezusammenfassungen werden nie an sie geschickt.
- Grenzen für Dateityp und -größe; Dateien werden nur an Ersteller:in und IT-Team ausgeliefert.
- Die ganze KI-Verarbeitung ist lokal: Kein Ticketinhalt geht an externe Dienste.
- Geheimnisse liegen in `.env`, das nicht in git ist. **Vor dem echten Einsatz ein neues JWT-Geheimnis erzeugen und die Seed-Passwörter ändern.**

> **Hinweis:** WAIT ist bewusst nur im Schulnetz erreichbar. Für Zugriff von außen bräuchte es eine Freigabe in der Firewall (zum Beispiel ausgehend Port 7844 für Cloudflare Tunnel) und eine zusätzliche Zugangssperre.

8. Konfigurationsreferenz

Variable	Standard	Zweck
DATABASE_URL	von Compose gesetzt	PostgreSQL-Verbindung
JWT_SECRET	Pflicht	Schlüssel für Tokens
JWT_EXPIRE_MINUTES	1440	Gültigkeit der Anmeldung
CORS_ORIGINS	http://localhost:8080	Erlaubte Browser-Adresse(n)
ALLOWED_EMAIL_DOMAINS	leer	Domain-Beschränkung der Registrierung
UPLOAD_DIR / MAX_UPLOAD_MB	/data/uploads / 10	Speicherort und Grenze für Anhänge
SEED_ON_START	1	Demo-Konten und Anleitungen beim Start anlegen
OLLAMA_BASE_URL	http://ollama:11434	KI-Server
OLLAMA_MODEL	llama3.2:3b (Server: qwen2.5:7b)	Textmodell
OLLAMA_VISION_MODEL	moondream (Server: qwen2.5vl:3b)	Bildmodell
OLLAMA_TIMEOUT_SECONDS	60	Maximale Wartezeit pro KI-Aufruf
AUTH_MODE	local (Server: both)	Anmeldequelle: lokal, AD oder beides
LDAP_*	leer	Active-Directory-Anbindung (Installationsanleitung 6.5)
LDAP_TIMEOUT_SECONDS	5	Zeitlimit für AD-Anfragen (ganze Sekunden)
APP_URL / SITE_PORT	http://localhost:8080 / 80	Website-Links und -Port
CF_API_TOKEN	nur Server	Cloudflare-Token für das HTTPS-Zertifikat

9. Entwicklung

9.1 Aufbau des Repositorys

Pfad	Inhalt
`wait-backend/app/main.py`	FastAPI-App und Registrierung der Router
`wait-backend/app/ai.py`	Der Assistent: Anleitungssuche, Schritte, LLM-Antworten, Bildvermerke, Übergabezusammenfassung
`wait-backend/app/llm.py`	Ollama-Client: Chat, Bildbeschreibung, Verkleinern, Wiederholung, Gedankenstrich-Filter
`wait-backend/app/ldap_auth.py`	Active-Directory-Anmeldung und Gruppenprüfung
`wait-backend/app/routers/`	API-Endpunkte: `auth`, `tickets`, `attachments`, `articles`, `admin` (inkl. CSV-Export)
`wait-backend/app/knowledge.py`, `knowledge_de.py`	Seed-Anleitungen (EN, DE)
`wait-backend/app/i18n.py`	Geskriptete Bot-Nachrichten und Button-Beschriftungen
`wait-backend/tests/`	pytest-Testsuite (42 Tests)
`wait-frontend/src/pages/`	Seiten für Nutzer:innen; `admin/` für den Admin-Bereich
`wait-frontend/src/components/ChatImages.jsx`	Bilder anhängen, einfügen, hineinziehen und anzeigen
`wait-frontend/src/i18n/`	Texte der Oberfläche (`de.js`, `en.js`)
`wait-site/`	Produkt-Website

9.2 Lokal ausführen

Voraussetzungen: Node 20+, Python 3.12+, PostgreSQL 15+ (Ollama optional; ohne Ollama antwortet der Assistent geskriptet).

```
# Backend
cd wait-backend
python -m venv venv && venv\Scripts\activate
pip install -r requirements-dev.txt
copy .env.example .env          # JWT_SECRET und DATABASE_URL setzen
alembic upgrade head
python -m app.seed
uvicorn app.main:app --reload --port 8000

# Frontend (zweites Terminal)
cd wait-frontend
npm install
npm run dev                     # http://localhost:5173
```

`wait-frontend/.env.local` setzt `VITE_API_URL`: auf `http://localhost:8000` für ein lokales Backend oder auf einen Server.

9.3 Tests und Prüfungen

```
cd wait-backend && python -m pytest tests -q     # braucht die Datenbank "wait_test"
cd wait-frontend && npm run lint && npm run build
```

9.4 Änderungen vornehmen

1. Einen Branch von `main` anlegen.
2. Code ändern; bei Datenbankänderungen `models.py` anpassen und `alembic revision --autogenerate -m "…"` ausführen.
3. Tests ergänzen oder anpassen, Testsuite und Frontend-Build laufen lassen.
4. In `main` mergen und pushen.
5. Auf dem Server ausrollen: zuerst eine Datenbanksicherung, dann `git pull` und `docker compose up -d --build` (siehe Installationsanleitung, Abschnitt 8.1).

10. Bekannte Grenzen und nächste Schritte

- **Keine Benachrichtigungen**: Personen und IT-Team müssen die App öffnen, um Antworten zu sehen. E-Mail- oder Push-Benachrichtigungen würden helfen.
- **Abfragen statt Live-Verbindung**: Offene Tickets aktualisieren sich alle 8 Sekunden; WebSockets würden den Chat sofort machen.
- **Schlüsselwortsuche** für Anleitungen; eine semantische Suche (zum Beispiel mit Qdrant) würde Anleitungen nach Bedeutung finden, ohne die Schnittstelle von `find_guides()` zu ändern.
- **Kleine GPU**: 6 GB fassen ein 7B-Modell auf einmal; der Wechsel zwischen Text- und Bildmodell kostet ein paar Sekunden. Mehr VRAM erlaubt größere, klügere Modelle.
- **Antworten werden nicht gestreamt**: Man wartet auf die ganze Antwort. Streaming würde sich deutlich schneller anfühlen.
- **Kein Krisenschutz**: Der Assistent lehnt schädliche Anfragen ab, aber Nachrichten über Selbstverletzung werden nicht erkannt oder an das IT-Team gemeldet. Vor dem Einsatz mit Schüler:innen empfohlen.
- **Nur im Schulnetz**: Zugriff von zu Hause ist nicht möglich, solange ausgehend Port 7844 gesperrt ist.