ItkChatbot

Entwicklerdokumentation

André Fischer

30.03.2026

Inhaltsverzeichnis

1 Technischer Überblick **3**
1.1 Symfony-Begriffe, die du hier brauchst . . . 3
1.2 KI-Request (vereinfacht) . . . 4
1.3 Sicherheit (Kurz) . . . 4

2 Konfigurationsgruppen **4**
2.1 KI / LLM . . . 4
2.2 Wissen / Embeddings / RAG . . . 5
2.3 Live-Chat . . . 5
2.4 UI / Storefront . . . 5

3 `services.xml` und technische Verdrahtung **5**

4 Datenmodell (DAL) **6**

5 Storefront-Routen **7**
5.1 KI-Chat (**ChatbotController**) . . . 7
5.2 Live-Chat (**LiveChatStorefrontController**) . . . 7

6 Admin-API (`_routeScope api`) **8**
6.1 **ChatbotKnowledgeController** . . . 8
6.2 **LiveChatAdminController** . . . 8
6.3 **ChatPresetAdminController** . . . 9

7 Services (Kernlogik) **9**

8 Administration **10**

9 Subscriber **10**

10 Console Commands **11**

11 Vollständige PHP-Klassenliste (Plugin `src/`) **11**

12 Custom-Modus (Middleware + CSV) **12**

13 Rate-Limiting (KI) **12**

14 ACL (Berechtigungen) **13**

15 Debugging-Reihenfolge **13**

1 Technischer Überblick

Plugin-Klasse: `ItkComputerGmbh\\ItkChatbot\\ItkChatbot`
Technischer Name: ItkChatbot
Haupt-Namespace: `ItkComputerGmbh\\ItkChatbot`
Konfigurationspräfix: `ItkChatbot.config.*`
Admin-Label: „ITK Live-Chat & KI-Chatbot“

Das Plugin registriert **DAL-Entities** (Wissen, Live-Chat, Embeddings, Presets), **Storefront-Controller** (KI + Live-Chat + CSV-Export), **Admin-API-Controller** (Wissen, Live-Chat, Presets), **Subscriber** (Seitenkontext, Widget-Injektion, Knowledge-Embeddings), **Services** (API-Client, Kontext, Embeddings, CSV) und **Console Commands**.

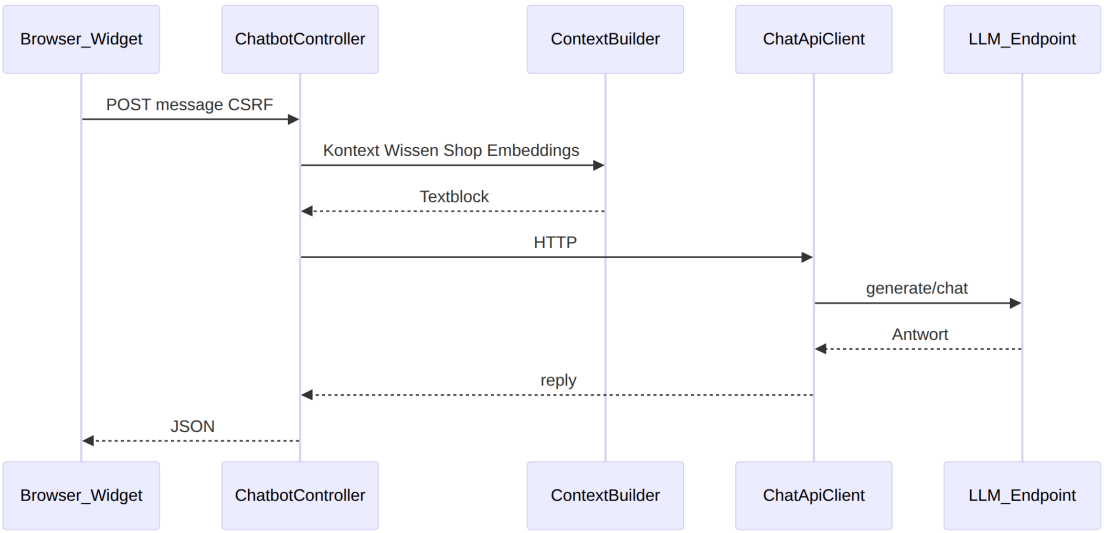
Technisch kannst du das Plugin in vier Ebenen lesen:

1. **Storefront-Ebene**: Widget, KI-Chat, Live-Chat, Session-Handling
2. **Service-Ebene**: Kontextaufbau, API-Kommunikation, Embeddings, Filter, Öffnungszeiten
3. **Persistenz-Ebene**: DAL-Tabellen für Wissen, Sessions, Nachrichten, Presets und Embeddings
4. **Admin-Ebene**: Wissenspflege, Live-Chat-Bearbeitung und Presets über Admin-API und Administration-Module

1.1 Symfony-Begriffe, die du hier brauchst

- **Controller:** Eine Klasse mit Methoden, die auf eine URL gemappt sind (**#[Route]**). Gibt JSON oder Response zurück.
- **DAL (Data Abstraction Layer):** Shopware-ORM – Entities, Repositories, **Criteria**/**Filter** statt rohem SQL.
- **SalesChannelContext vs. Context:** Im Storefront hast du oft Kontext **mit** Verkaufskanal; CLI und manche API-Calls nutzen `Context::createDefaultContext()`.
- **Subscriber:** Reagiert auf Lifecycle-Events (hier z. B. Seiten-Rendering, Wissens-Änderungen).

1.2 KI-Request (vereinfacht)



1.3 Sicherheit (Kurz)

- Storefront-Route für KI-Nachrichten nutzt **CSRF** (Token wird per **ChatbotInjectSubscriber** ins Layout gegeben) – nicht blind von fremden Origins aufrufbar ohne Token.
- Live-Chat: **Session-Token** für Gäste; Admin-Routen mit `itk_live_chat.view`.

---

2 Konfigurationsgruppen

Die tatsächliche `config.xml` ist deutlich breiter als die paar Keys, die in den Controllern sofort sichtbar sind. Für die tägliche Arbeit lohnt sich diese Einteilung:

2.1 KI / LLM

Typische Schlüssel:

- **aiChatEnabled**
- **apiUrl**
- **apiFormat**

- **apiKey**
- **modelName**
- **timeout**
- **maxTokens**
- **apiSkipSslVerify**
- **rateLimitPerMinute**

2.2 Wissen / Embeddings / RAG

Hier steuerst du, wie stark Wissenseinträge, Produktdaten und Kategorien in den Prompt-Kontext einfließen und ob Embeddings genutzt werden. Diese Gruppe ist für Antwortqualität und Laufzeit entscheidend.

2.3 Live-Chat

Öffnungszeiten, Bad-Word-Filter, geschlossene Meldungen, Session-Verhalten und Polling-relevante Fachregeln hängen an einem eigenen Konfigurationsblock.

2.4 UI / Storefront

Sichtbarkeit des Widgets, Darstellung und bestimmte Frontend-Funktionen hängen ebenfalls an SystemConfig. Deshalb setzen Subscriber im Code oft nur Flags und nicht die komplette Fachlogik selbst um.

---

3 `services.xml` und technische Verdrahtung

Das Plugin nutzt in `services.xml` fast alle klassischen Shopware-Schemata:

- `shopware.entity.definition` für alle DAL-Definitionen
- `controller.service_arguments` für Storefront- und Admin-Controller
- `kernel.event_subscriber` für Storefront- und Embedding-Subscriber
- `console.command` für Warmup-/Embedding-/Optimierungsbefehle

Wichtige technische Muster:

- **ChatApiClient** und **EmbeddingService** sprechen über **HttpClientInterface**

- **KnowledgeService**, **ShopDataService** und **ShopEmbeddingService** hängen direkt an Repositories
- **ContextBuilder** ist die zentrale Orchestrierungsklasse für den Prompt-Kontext
- Logger sind bewusst getrennt in `itk_chatbot.log` und `itk_chatbot_embeddings.log`

Wenn du neue Fachlogik ergänzen willst, landet sie in diesem Plugin fast nie direkt im Controller, sondern zuerst in einem Service, der danach in `services.xml` verdrahtet wird.

---

4 Datenmodell (DAL)

Entity-Name (DAL)	PHP-Definition	Inhalt
itk_chatbot_knowledge	**ChatbotKnowledgeDefinition**	Wissensbasis-Einträge inkl. optionalem Embedding-Feld (Migration).
itk_chatbot_chat_session	**ChatSessionDefinition**	Live-Chat-Session: Sales Channel, Kunde/Gast, Status, Claim, Seiten-URL/Typ, Produkt, Notizen, Session-Token (Gast).
itk_chatbot_chat_message	**ChatMessageDefinition**	Einzelne Nachrichten, Sender-Typ (Kunde/Mitarbeiter), Inhalt, Zeitstempel.
itk_chatbot_chat_preset	**ChatPresetDefinition**	Textvorlagen pro Admin-Benutzer.
itk_chatbot_product_embedding	**ProductEmbeddingDefinition**	Embedding-Vektoren zu Produkten.

Entity-Name (DAL)	PHP-Definition	Inhalt
itk_chatbot_category_embedding	**CategoryEmbeddingDefinition**	Embedding-Vektoren zu Kategorien.

Zu jeder Definition gehören **Entity**-, **Collection**-Klassen (wo vorhanden) unter `Core/Content/...`.

Migrationen unter `src/Migration/` legen Tabellen an und erweitern sie (z. B. **sessionToken**, **guestName**, **notes**, **updatedAt**, Embedding-Spalten).

---

5 Storefront-Routen

5.1 KI-Chat (ChatbotController)

Methode	Pfad	Name	Verhalten
POST	/itk-chatbot/message	frontend.itk-chatbot.message	Prüft **aiChatEnabled**, Rate-Limit (Session), **apiUrl**, baut Kontext (**ContextBuilder**), System-Prompt (**PromptBuilder**), ruft **ChatApiClient**. Request: **message**, optional **pageContext** (JSON), **conversationHistory[]** (max. letzte 10 Paare **user**/**assistant**). JSON: **{ reply }** oder **{ error }** mit 400/429/502/503.
GET	/itk-chatbot/export/products-categories.csv	frontend.itk-chatbot.export.products-categories	CSV via **ProductCsvExport Service**, Cache-Control **private, max-age=86400**.

5.2 Live-Chat (LiveChatStorefrontController)

Methode	Pfad	Name	Kurzbeschreibung
POST	/itk-chatbot/live-chat/start	frontend.itk-chatbot.live-chat.start	Neue Session oder Update bestehender Session (Seiteninfos); schließt vorher aktive Chats desselben Kunden/Gasts; Gast bekommt **sessionToken**.
GET	/itk-chatbot/live-chat/session	frontend.itk-chatbot.live-chat.session	Session-Status; Ownership-Prüfung.
POST	/itk-chatbot/live-chat/message	frontend.itk-chatbot.live-chat.message	Nachricht max. 5000 Zeichen, `strip_tags`, Bad-Word-Filter, Speichern immer; außerhalb Öffnungszeiten JSON mit **outsideHours** + **infoMessage**.
GET	/itk-chatbot/live-chat/messages	frontend.itk-chatbot.live-chat.messages	Polling neuer Nachrichten seit **since** (Timestamp).

Alle genannten Routen: **routeScope** storefront, typisch **XmlHttpRequest** / **noStore** gesetzt.

---

6 Admin-API (`_routeScope` api)

6.1 ChatbotKnowledgeController

Routen unter Präfix `/api/_action/itk-chatbot/knowledge` für CRUD, Suche und Pagination der Wissenseinträge. Technisch arbeitet der Controller direkt gegen `itk_chatbot_knowledge.repository`.

6.2 LiveChatAdminController

Berechtigung: `itk_live_chat.view` für alle Aktionen.

Routen (Auszug, vollständig in `LiveChatAdminController.php`):

- **GET /api/_action/itk-chatbot/live-chat/sessions** – Liste offener + eigener übernommener Sessions.

- **GET /api/_action/itk-chatbot/live-chat/sessions/{id}** – Detail inkl. Nachrichten, Produkt, Kunde.
- Weitere Routen: Session übernehmen (**claim**), schließen, Nachricht senden, Nachrichten laden, Notizen, Produkt-Suche für Befehle, Session löschen, etc.

6.3 ChatPresetAdminController

Ebenfalls **Berechtigung** `itk_live_chat.view`. CRUD für `itk_chatbot_chat_preset` (Presets pro User).

---

7 Services (Kernlogik)

Klasse	Aufgabe
ChatApiClient	HTTP-Client für LLM: Formate **ollama**, **openai**, **custom**; unterstützt API-Key, Timeout, maxTokens, optional SSL-Verify aus.
ContextBuilder	Orchestriert **KnowledgeService**, **ShopDataService**, optional **EmbeddingService** + **ShopEmbedding Service**, baut den Textkontext für die KI; Logging für Debug.
PromptBuilder	Erzeugt System-Prompt aus Kontext + optionalem Bot-Namen.
KnowledgeService	Lädt Wissen aus DAL; mit/ohne semantische Suche über **EmbeddingService**.
ShopDataService	Lädt Produkt- und Kategoriedaten für Kontext (Repositories).
EmbeddingService	Ruft Embedding-API auf (Ollama-kompatibel), Konfiguration aus SystemConfig.
ShopEmbeddingService	Ähnlichkeitssuche auf Produkt-/Kategorie-Embeddings.
ProductCsvExportService	CSV-Inhalt + Caching (cache.object) für Export und Warmup.
OpeningHoursService	Parsed JSON-Öffnungszeiten, Zeitzone, Closed-Message aus Config.

Klasse	Aufgabe
BadWordFilterService	Filtert Live-Chat-Text je nach Config.

Logger (Monolog):

- `itk_chatbot` → `var/log/itk_chatbot.log`
- `itk_chatbot_embeddings` → `var/log/itk_chatbot_embeddings.log`

---

8 Administration

Die Admin-Oberfläche lebt unter `Resources/app/administration/` und besteht technisch aus:

- Live-Chat-Modul
- Knowledge-Modul
- Preset-bezogenen API-Services
- ACL-Registrierung
- Admin-spezifischen JS-API-Services für die Controller unter `Administration/Controller`

Wenn im Admin etwas nicht sichtbar ist, musst du deshalb immer drei Ebenen unterscheiden:

1. liefert der PHP-Controller korrekt JSON?
2. ruft der Admin-API-Service die richtige Route auf?
3. ist das Modul / die Komponente in der Administration korrekt registriert?

---

9 Subscriber

Klasse	Event / Zweck
ChatbotPageSubscriber	Setzt u. a. ob Chat für Layout aktiv ist; Logging.

Klasse	Event / Zweck
ChatbotInjectSubscriber	Injiziert Widget-HTML/JS in Storefront-Response; CSRF-Token für Nachrichten-Route.
KnowledgeEmbeddingSubscriber	Reagiert auf Änderungen an Wissenseinträgen und hält die Embedding-Daten synchron, damit gespeicherte Inhalte auch semantisch durchsuchbar bleiben.

---

10 Console Commands

Service-Klasse	Registrierter Name
GenerateProductCsvCommand	itk:chatbot:products-csv:warmup
OptimizeKnowledgeCommand	itk:chatbot:knowledge:optimize
GenerateEmbeddingsCommand	itk:chatbot:embeddings:generate
GenerateShopEmbeddingsCommand	itk:chatbot:embeddings:generate-shop

---

11 Vollständige PHP-Klassenliste (Plugin `src/`)

Plugin-Bootstrap: `ItkChatbot.php`

Commands: **GenerateProductCsvCommand**, **GenerateEmbeddingsCommand**, **GenerateShopEmbeddingsCommand**, **OptimizeKnowledgeCommand**

Core / Content:
ChatbotKnowledge (Definition, Entity, Collection),
ChatSession (Definition, Entity, Collection),
ChatMessage (Definition, Entity, Collection),
ChatPreset (Definition, Entity, Collection),
ProductEmbedding (Definition, Entity, Collection),
CategoryEmbedding (Definition, Entity, Collection)

Administration / Controller: ChatbotKnowledgeController, **LiveChatAdminController**, **ChatPresetAdminController**

Storefront / Controller: ChatbotController, **LiveChatStorefrontController**

Storefront / Subscriber: ChatbotPageSubscriber, **ChatbotInjectSubscriber**

Subscriber: KnowledgeEmbeddingSubscriber

Service: BadWordFilterService, **ChatApiClient**, **ContextBuilder**, **EmbeddingService**, **KnowledgeService**, **OpeningHoursService**, **ProductCsvExportService**, **PromptBuilder**, **ShopDataService**, **ShopEmbeddingService**

Migration: alle **Migration17…** Klassen unter `src/Migration/`

---

12 Custom-Modus (Middleware + CSV)

Im API-Format **custom** erwartet **ChatApiClient** weiterhin eine Ollama-nahe Chat-Schnittstelle. Der Unterschied ist nicht der Storefront-Flow, sondern die Gegenstelle:

- Shopware baut Kontext, Prompt und History wie gewohnt
- eine Middleware kann zusätzliche Daten wie CSV-Kontext oder andere externe Informationen ergänzen
- die Antwort kommt wieder in einer Form zurück, die in den normalen Reply-Flow passt

Praktisch heißt das: „custom“ ist kein zweites Plugin, sondern dieselbe Chat-Pipeline mit einer freieren Server-Gegenstelle.

---

13 Rate-Limiting (KI)

Implementierung in **`ChatbotController::checkRateLimit`**: Session-Key `itk_chatbot_ratelimit`, Zeitfenster 60 Sekunden, Zähler aus Timestamps. Kein Redis – bei verteilten Sessions ggf. eigenes Rate-Limit vorschalten (Reverse Proxy).

---

14 ACL (Berechtigungen)

Privileg: `itk_live_chat.view` – in Administration unter `Resources/app/administration/src/acl/` registriert. Ohne dieses Recht keine Live-Chat-API und keine Preset-Verwaltung über die geschützten Controller.

---

15 Debugging-Reihenfolge

Wenn du einen Fehler eingrenzen musst, hilft diese Reihenfolge:

1. **Storefront-Controller** prüfen: Kommt der Request richtig an, stimmen Statuscodes und Validierung?
2. **Services** prüfen: Kontextaufbau, Embedding-Suche, API-Call zum LLM, Öffnungszeiten, Filter.
3. **DAL / Admin** prüfen: Wurden Session, Nachrichten, Presets oder Wissenseinträge korrekt persistiert?

So deckst du den typischen Weg von der Widget-Anfrage bis zur Datenhaltung in einer sinnvollen Reihenfolge ab.