Der KI-Chatbot „BibKI“ der KIT-Bibliothek

Uwe Dierolf, Tobias Kurze, Frank Polgart, Michael Skarupianski, Bernadette Breyer

Abstract

Am Nikolaustag im Dezember 2024 wurde an der KIT-Bibliothek in Karlsruhe der KI-Chatbot „BibKI“ zur Beantwortung von Nutzerfragen öffentlich für Tests für eingeweihte Personen zugänglich gemacht (quasi „unter dem Radar“) und am 7. Januar ganz offiziell auch in die Website und den Katalog der KIT-Bibliothek integriert.
Er kann von dort oder direkt mittels https://chatbot.bibliothek.kit.edu aufgerufen werden.

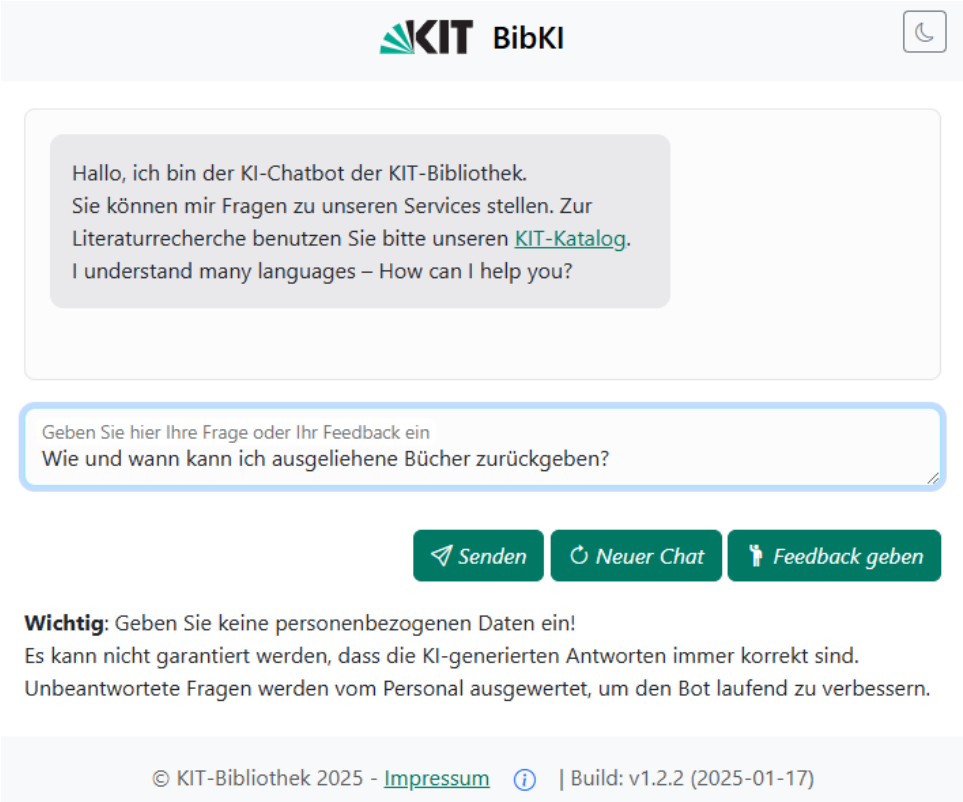

Abbildung: KI-Chatbot „BibKI“ im Light Mode (ein Dark Mode oben rechts ist ebenfalls möglich)

Dieser Chatbot ist ein Service-Chatbot zur Beantwortung von FAQ-Fragen rund um die KIT-Bibliothek. Es handelt sich **nicht** um einen Recherche-Chatbot im Stil des Berliner VÖBB-Chatbots oder SHAI, der KI-Assistenz der Bibliotheken Schaffhausens.

Im Fokus lag eine leichtgewichtige, einfach zu hostende Web-Anwendung, die von Dritten nachgenutzt werden kann. Einzige Voraussetzung ist ein OpenAI Account bei https://platform.openai.com/

Idee und erste Schritte

Mit dem Aufkommen der großen Sprachmodelle entstand bei vielen Anwendern schnell der Wunsch nach KI-Lösungen, die Antworten auf Fragen zu einer eigenen lokalen Wissensbasis

(local knowledge base) ermöglichen. Diese Thematik ist unter dem Begriff RAG (Retrieval Augmented Generation) bekannt geworden.

Hierzu wurden unzählige Veröffentlichungen, Blogbeiträge, YouTube-Videos etc. gemacht. Die Grundidee besteht darin, eine semantische Suche über Dokumente verschiedener Typen (PDF, Word, Markdown, Text, JSON etc.) durchzuführen und das Ergebnis (also die Antworten) von einem LLM ausformulieren zu lassen.

So ermöglicht die semantische Suche, Dokumente zu finden, in denen das Wort 'Tier' zwar nicht direkt vorkommt, aber beispielsweise 'Katze' erwähnt wird. Damit unterscheidet sie sich grundlegend von der Keyword-Suche, wie sie typischerweise in Bibliothekskatalogen und anderen Datenbanksystemen verwendet wird.

Technische Begriffe, die man bei RAG immer wieder antrifft, sind Embeddings und Vector Stores. Bei **Embeddings** handelt es sich um eine interne, mathematische Notation eines Dokuments oder Dokument-Teils (auch Chunk), einem sog. Vektor. Je nach Embedding-Modell können mehr oder weniger viele Dimensionen für einen solchen n-dimensionalen Vektor erzeugt werden. Diese Anzahl schwankt aktuell zwischen einigen hundert bis hin zu einigen tausend.
Die Vektoren werden in einem **Vector Store**, einem spezialisierten Datenbanksystem, abgelegt. Ein Beispiel dafür ist ChromaDB. Solche Spezial-Datenbanken können sehr schnell anhand einer Suchanfrage, zu der auch ein Embedding berechnet wird, die passenden Dokumente ermitteln. Die Treffer einer solchen semantischen Suche sind Dokumente bzw. Dokument-Teile (chunks). Chunks werden aktuell deswegen verwendet, da Sprachmodelle (noch) nicht beliebig viel Text auf einmal verarbeiten können. Es muss die sog. „context size“ berücksichtigt werden. Das Sprachmodell erzeugt dann anhand der Treffer (-Chunks) eine passende Antwort zur Frage.

Nach einigen Monaten der Implementierung solcher RAG-Anwendungen, die eine semantische Suche lokal durchführen und zur Erzeugung der Antworten entweder lokale (llama3.x etc.) oder externe Sprachmodelle (OpenAI, Claude, etc.) genutzt haben, waren die Ergebnisse häufig nur dann von guter Qualität, wenn sowohl die Erzeugung der Embeddings als auch die Erstellung der Antworten von externen großen KI-Anbietern durchgeführt wurde.

Parallel dazu wurden Versuche mit Custom GPTs von ChatGPT durchgeführt, bei denen man Dokumente unterschiedlichen Typs hochladen und Fragen dazu stellen kann. Die Bereitstellung solcher Custom GPTs für Endnutzer wurde jedoch vom zuständigen Datenschutz-Team am KIT nicht erlaubt, da es seitens der Nutzer einen ChatGPT-Account erfordert hätte und somit personenbezogene Daten an OpenAI abgeflossen wären.

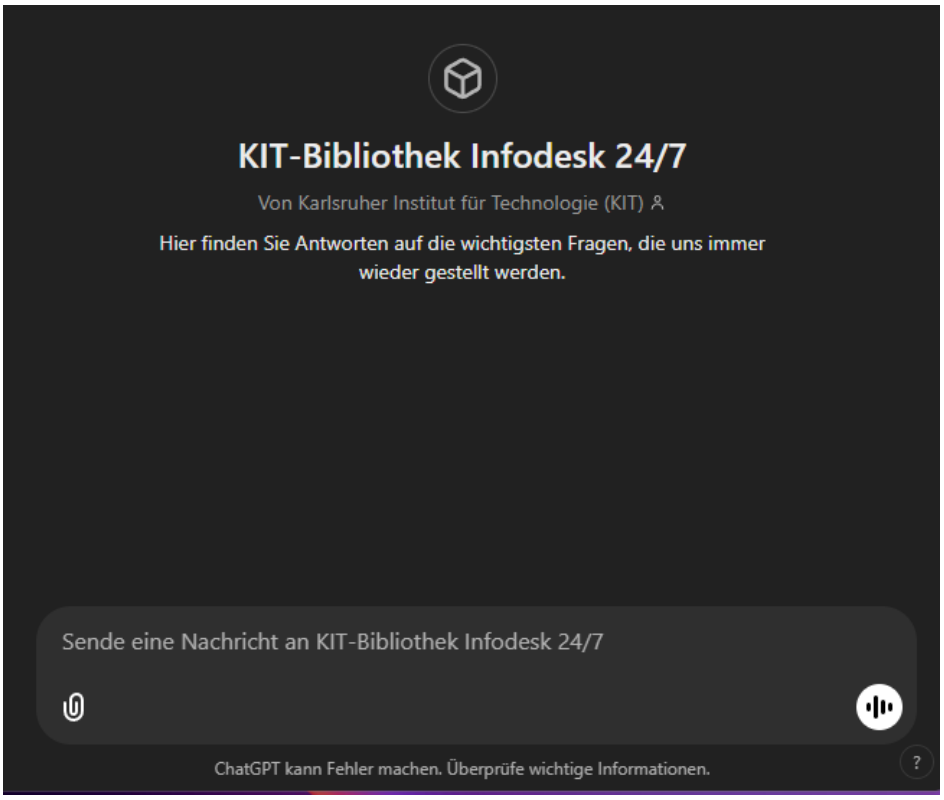

Abbildung: Custom GPT

Daher war schnell klar, dass eine eigene Lösung für eine Web-Anwendung eines Service-Chatbots (quasi ein KIT Custom GPT) implementiert werden muss. Dieser Lösungsweg wird hier vorgestellt. Die Grundlage bildet ein OpenAI Assistant.

OpenAI Assistant

Unter der tendenziell weniger bekannten URL https://platform.openai.com/ findet man im Dashboard u.a. „Assistants“. Diese erlauben das Hochladen von Dokumenten, auf denen der Assistant eine sogenannte „File search“ durchführen kann. D.h. der RAG-Teil wird komplett vom Assistant erledigt.
In den „System instructions“ teilt man dem Assistant mit, dass er nur darauf basierend antworten und nicht sein großes, ChatGPT zugrundeliegendes Sprachmodell benutzen soll.

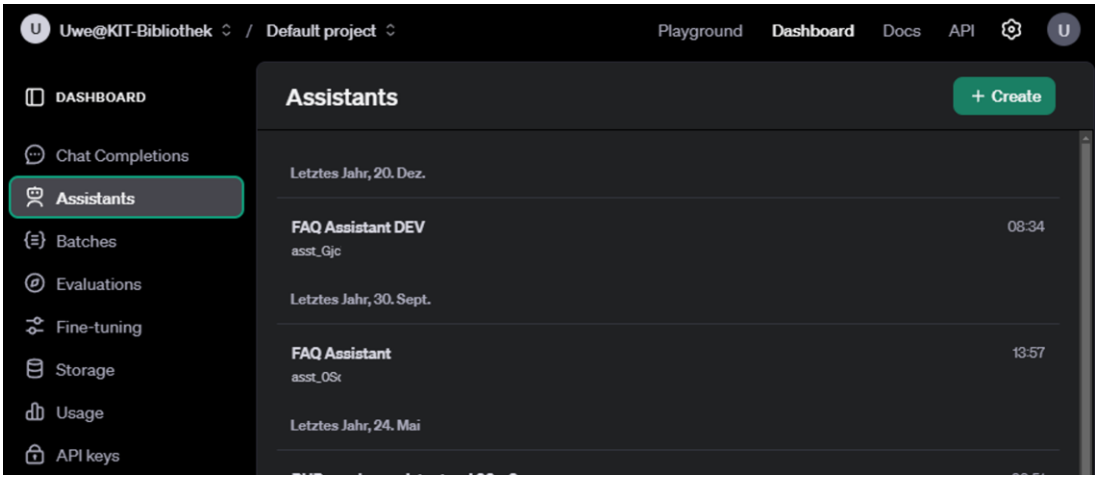

Abbildung: OpenAI Assistants (Assistant IDs sind verkürzt)

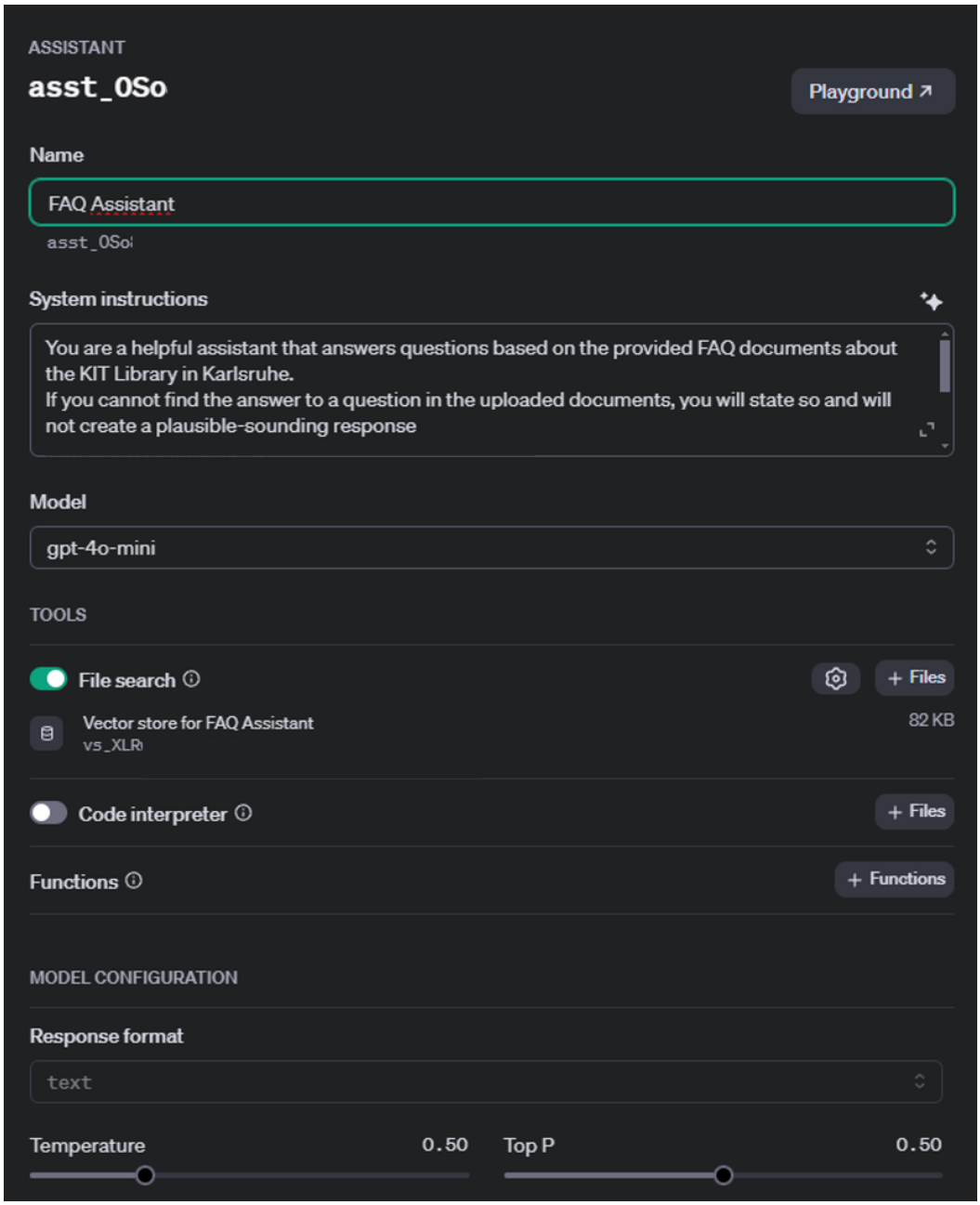

Abbildung: FAQ Assistant

Mittels des Playground-Buttons oben rechts, kann man direkt Anfragen an den eigenen Assistant stellen, wie man es von ChatGPT gewohnt ist.

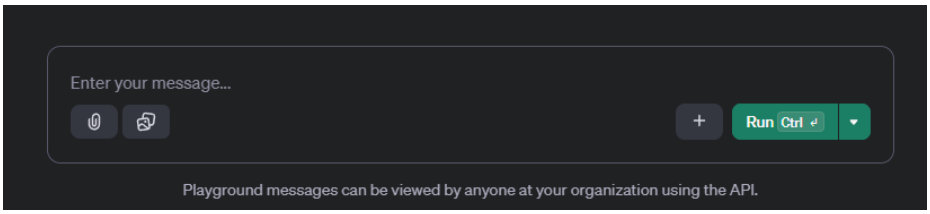

Abbildung: Playground

Die Nutzung des Playgrounds ist übrigens auch ein sehr kostengünstiger Weg, um ChatGPT beliebig oft zu nutzen und nur für die Tokens zu bezahlen, die zur Beantwortung der Fragen

benötigt wurden. Das erspart die laufenden Kosten von ca. 20 EUR pro Monat. Zusätzlich erhält man mit den Assistants die Möglichkeit, RAG zu testen. Wichtig ist, dass man hierbei die Kosten gut im Blick behält und das Modell am besten sofort von gpt-4o auf gpt-4o-mini umstellt. Gegenüber ChatGPT kann man so auch experimentieren, welchen Einfluss die Parameter „Temperature“ und „Top P“ etc. auf die Qualität der Antworten der KI haben. So soll die KI bei einem niedrigeren Temperature-Wert deutlich weniger zum Halluzinieren neigen. Sie wird so aber auch zugleich weniger kreative Antworten liefern.

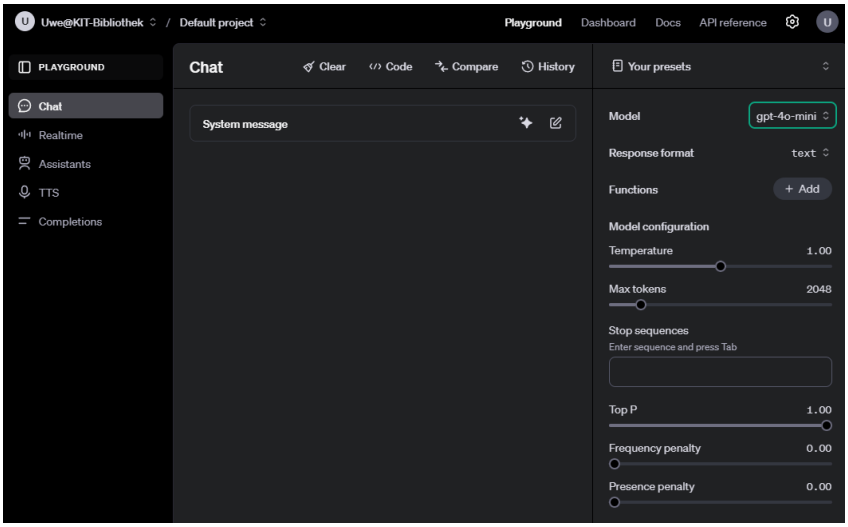

Abbildung: Chats im Playground

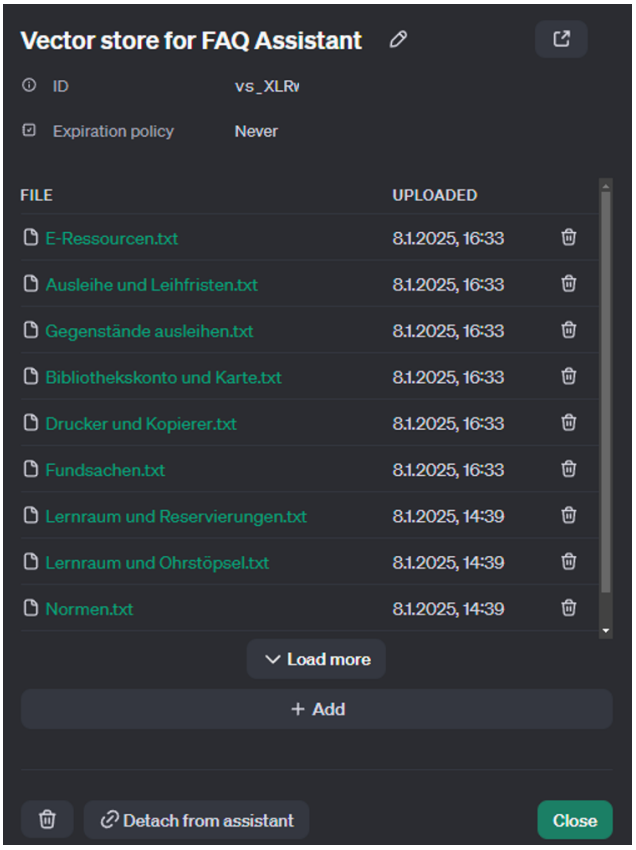

Abbildung: Hochgeladene FAQ-Dokumente im Vectorstore

Die Integration in die eigene KI-Chatbot-Webanwendung erfolgt anhand des Assistant API von OpenAI. Man kann Anfragen stellen und erhält fortlaufend (gestreamt) die jeweiligen Antworten und kann diese in der eigenen Weboberfläche ausgeben.

Das Verfahren erfordert lediglich die Kenntnis eines OpenAI API Keys und einer Assistant ID.

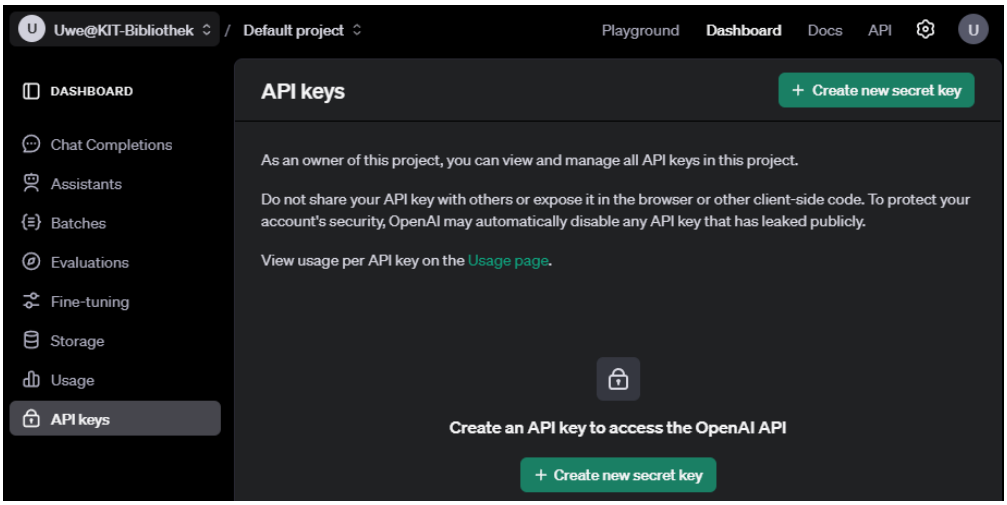

Abbildung: OpenAI API Keys

OpenAI stellt eine sehr ausführliche Dokumentation zu allen APIs zur Verfügung. Exemplarisch hier ein Beispiel für die Assistants API.

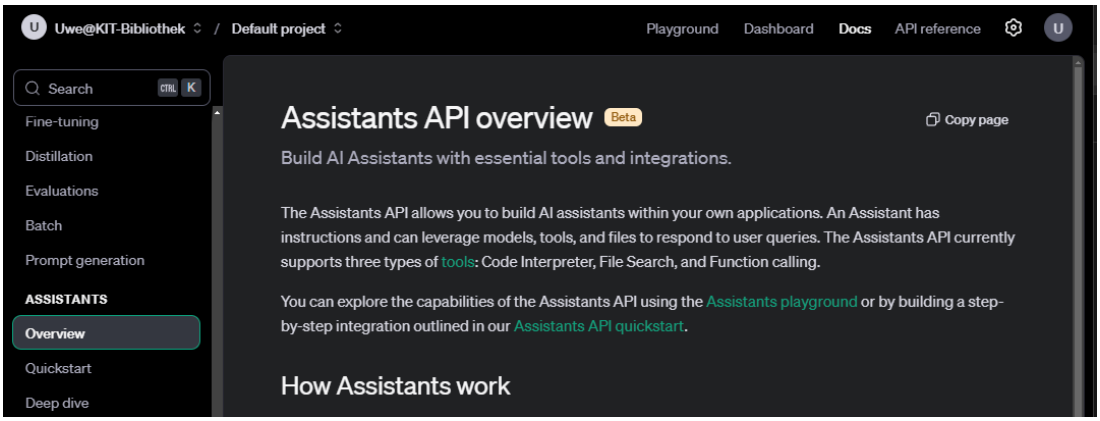

Abbildung: Docs – Assistants API

Architektur & Hosting

Der KI-Chatbot ist eine Python-Anwendung. Jede Instanz läuft in einem eigenen Docker-Container. Vorgeschaltet ist der Web-Proxy „Caddy“, der auch als Docker-Container seine Arbeit verrichtet. Er verteilt die eingehenden Anfragen an die KI-Chatbots und kümmert sich zusätzlich um das Holen und Erneuern von Let’s Encrypt Zertifikaten, damit die KI-Chatbots HTTPS unterstützen. Die KI-Chatbots kommunizieren mit OpenAI über die oben bereits erwähnte API.

Sofern ein DNS-Eintrag auf den KIT-Chatbot-Server verweist und dazu eine Konfiguration bestehend aus Hostname (z.B. chatbot.meine-domaine.de) sowie dem OpenAI API Key und der Assistant ID auf dem Server vorhanden ist, kann sofort ein weiterer Service-Chatbot auch Dritten bereitgestellt werden.

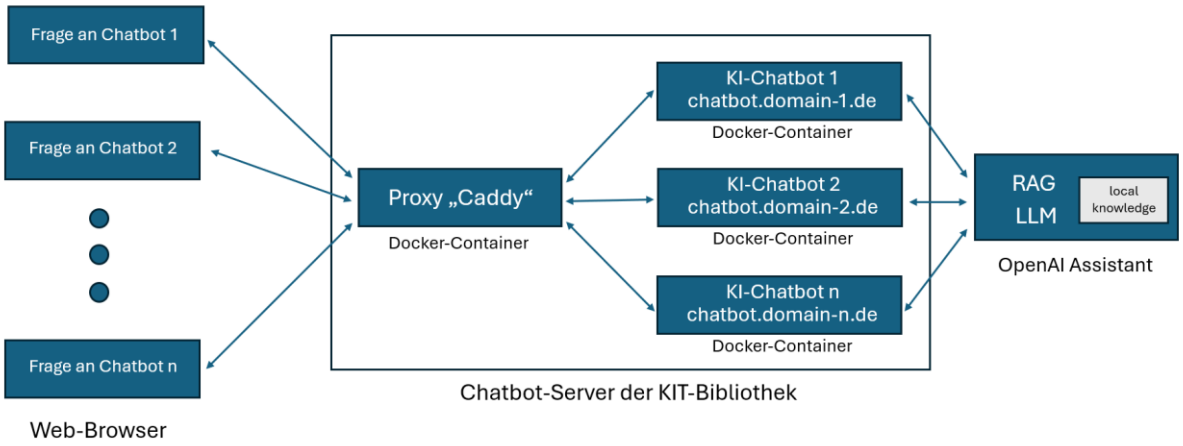

Abbildung: Architektur der Webanwendung „KI-Chatbot“

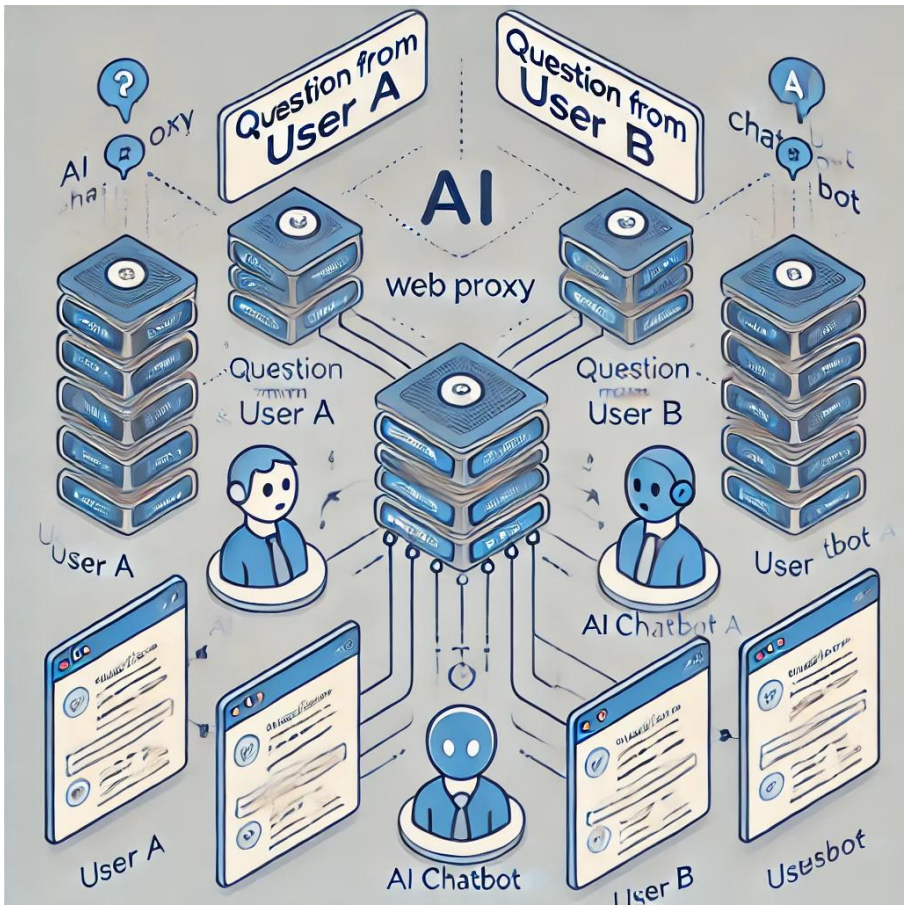

Abbildung: Architektur der Chatbot-Webanwendung von KI erstellt

Prompt:
kannst du mir ein architekturbild erstellen, das folg. Szenario beschreibt:
Ein Nutzer A schickt von seinem Browser eine Anfrage "Frage von Nutzer A" an "KI-Chatbot A".
Ein Nutzer B schickt von seinem Browser eine Anfrage "Frage von Nutzer B" an "KI-Chatbot B".
Auf dem Chatbot-Server der KIT-Bibliothek gehen all diese Anfragen ein.
Auf dem Chatbot-Server läuft ein Web-Proxy "Caddy" in einem Docker-Container.
Außerdem läuft ein Docker-Container für "KI-Chatbot A" sowie ein Docker-Container für "KI-Chatbot B"
Caddy als Proxy verteilt die Anfrage "Frage von Nutzer A" an den Docker-Container für "KI-Chatbot A".
Caddy als Proxy verteilt die Anfrage "Frage von Nutzer B" an den Docker-Container für "KI-Chatbot B".

Dokumente

Man kann viele verschiedene Dokumenttypen wie Word, PDF, Excel, aber auch einfache Textdokumente beim OpenAI Assistant für die File-Search hochladen. Wir haben uns für Textdokumente entschieden, die auch Markdown enthalten dürfen. Diese Dateien werden von der Benutzungsabteilung gepflegt. In den Dokumenten dürfen Überschriften (#, ##, ###) und wichtige Begriffe mit Markdown-Syntax (**..**) ausgezeichnet werden. Das hat den Vorteil, dass bei der Aufbereitung der Antwort diese Auszeichnungen erhalten bleiben. So können z.B. Telefonnummern oder wichtige Begriffe auch fettgedruckt vom Service-Chatbot ausgegeben werden. Telefonnummern können auf mobilen Endgeräten auch direkt für die Durchführung eines Telefonanrufs angeklickt werden.

Web-Crawling vs. FAQ-Dokumente

Beim Lesen dieses Beitrags stellt sich unweigerlich die Frage, warum man Daten doppelt pflegen sollte, wenn man doch bereits einen Webauftritt verwaltet, der alle wichtigen Informationen enthält.
An der KIT-Bibliothek werden daher auch Versuche durchgeführt, bei denen der Inhalt der Seiten einer Website durch einen GPT Crawler (s. github) ermittelt und als JSON-Daten beim OpenAI Assistant hochgeladen werden.

Abbildung: GPT-Crawler (https://github.com/BuilderIO/gpt-crawler)

Zukünftig wird angestrebt, die lokale Wissensbasis nur einmal zu pflegen. Die Kunst liegt beim Crawlen in dem Ermitteln der inhaltlich relevanten Abschnitte einer Seite. Texte aus Menüs, die auf allen Seiten vorkommen, sollten eliminiert werden. Dies erfordert beispielsweise Anpassungen des in Typescript geschriebenen GPT Crawlers oder die Erstellung von Skripten, die das Ergebnis in weiteren Läufen bereinigen. Es gibt jede Menge weiterer Crawler wie z.B. Crawl4AI, so dass die Auswahl schwerfällt.

Man muss sich auch Gedanken darüber machen, wie man die Änderungen an den Seiten einer Website erfasst und die „local knowledge base“ aktualisiert. Für kleine Websites wäre der

simpelste Ansatz, jede Nacht alles abzuziehen und den Vector Store vollständig neu aufzubauen. Auch hierzu bietet OpenAI API-Calls für den Umgang mit Dateien und Vector Stores, mit denen solch ein Vorgang vollautomatisch durchgeführt werden kann. Das (Neu-)Berechnen von Embeddings für Datenmengen im MB-Bereich ist nicht teuer, wie das folg. Kapitel zeigt.

Kosten

Die Nutzung eines solchen KI-Service wird auf Token-Basis (input und output) abgerechnet. Dazu muss man zu seinem OpenAI Account ein Zahlungsmittel (Billing) hinterlegen. Im Normalfall ist dies eine Kreditkarte. Daran scheitern viele Einrichtungen des öffentlichen Dienstes, da nur selten eine Kreditkarte vorhanden ist. Sobald man diese Bezahl-Hürde jedoch überwunden hat, können erste Tests selbst durch das Aufladen von wenigen Euro durchgeführt werden.

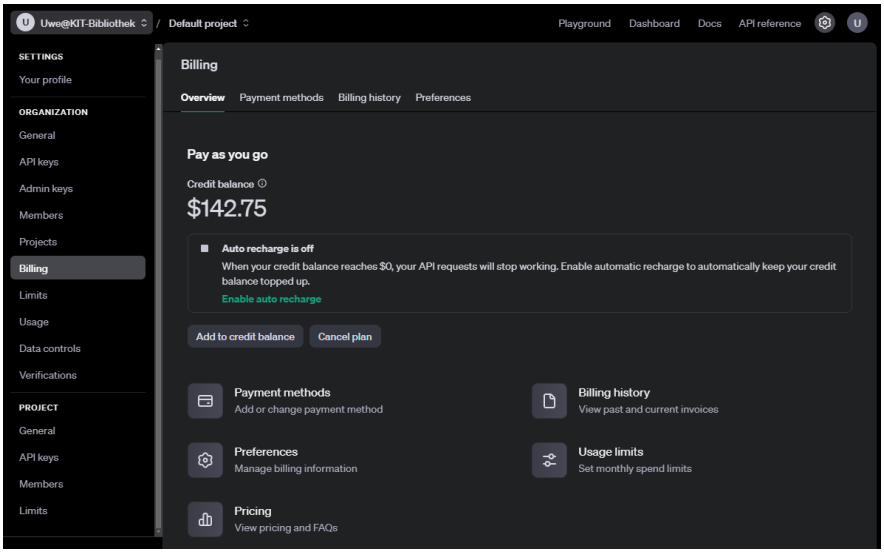

Abbildung: Billing - Zahlungsmittelinformation

Im Dashboard findet man dann sowohl die API-Aufrufe als auch deren Kosten.

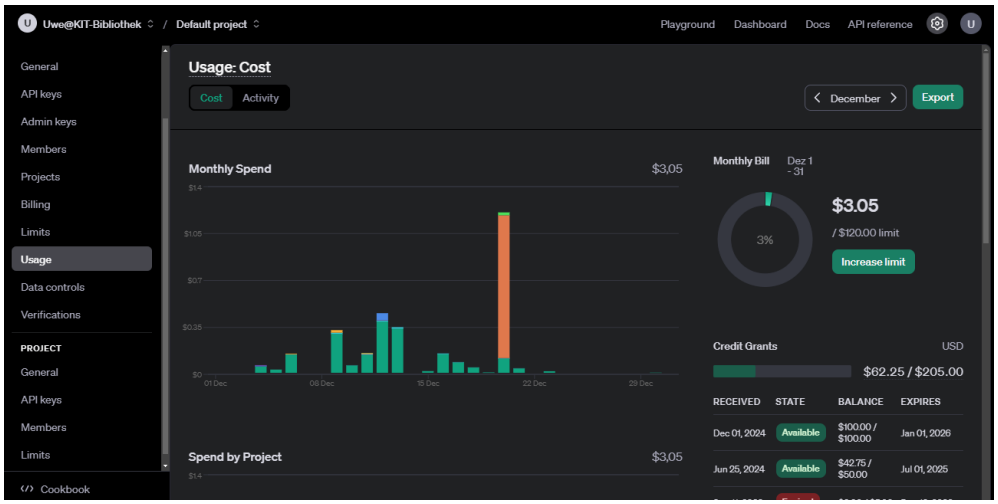

Abbildung: Usage

Wichtig ist, dass beim OpenAI Assistant darauf geachtet werden muss, welches Modell von OpenAI zum Einsatz kommt. So kostet gpt-4o ca. das 15-fache gegenüber gpt-4o-mini. Nach dem Erzeugen eines neuen OpenAI Assistant sollte daher der Defaultwert für „Model“ von gpt-4o auf gpt-4o-mini geändert werden.

Ein anderer Kostenpunkt ist die Menge der hochgeladenen Dokumente. Im Fall der KIT-Bibliothek sind derzeit nur wenige MB im Vector Store des OpenAI Assistant hochgeladen worden. Derzeit sind ca. 30 Dokumente hinterlegt, so dass bei Änderungen nicht immer für alles die Embeddings neu berechnet werden müssen. Die Kosten für eine einmalige (Neu-) Berechnung der Embeddings aller Dokumente liegt noch im einstelligen Cent-Bereich. Aber selbst große Datenmengen im kleinen GB-Bereich kosten bei OpenAI pro Monat nur wenige Euro. In solchen Fällen kann das häufige Neuberechnen der Embeddings jedoch einen echten Kostenfaktor darstellen.

Die meisten dieser Kosten findet man unter dem Punkt „Pricing“, der sich wiederum im Menüpunkt „Billing“ findet (https://openai.com/api/pricing/).

Man kann sog. „Projects“ in OpenAI anlegen. Diesen Projekten kann man Project-Owner zuordnen, die sich dann selbst einen eigenen OpenAI Assistant anlegen und dort Dokumente hochladen dürfen. Die Abrechnung erfolgt dann über das zentral hinterlegte Zahlungsmittel, wie z.B. die Kreditkarte. Projekte ermöglichen es also, Nutzern eine isolierte Umgebung für ihre(n) Chatbot(s) bereitzustellen, die aber über ein Zahlungsmittel abgerechnet werden.

Sollte eine interne Verrechnung pro Projekt erforderlich sein, findet man dazu im OpenAI Dashboard die pro Projekt entstandenen Kosten.

In den letzten Tagen wurde das Open Source Modell „DeepSeek“ vorgestellt. Die Preise für die API-Nutzung liegen deutlich unter denen von OpenAI. Man darf daher auch bei OpenAI von einer Anpassung der Preise nach unten ausgehen.

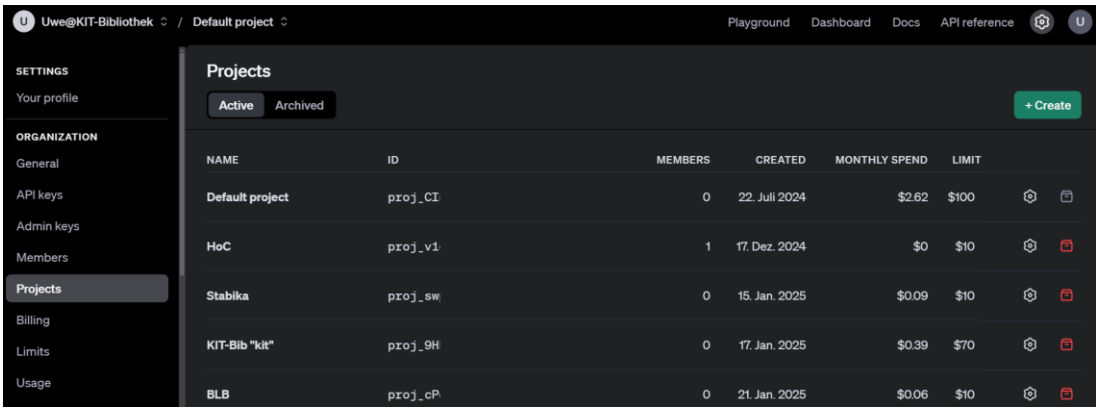

Abbildung: Projects findet man in den Settings (das Rad rechts oben)

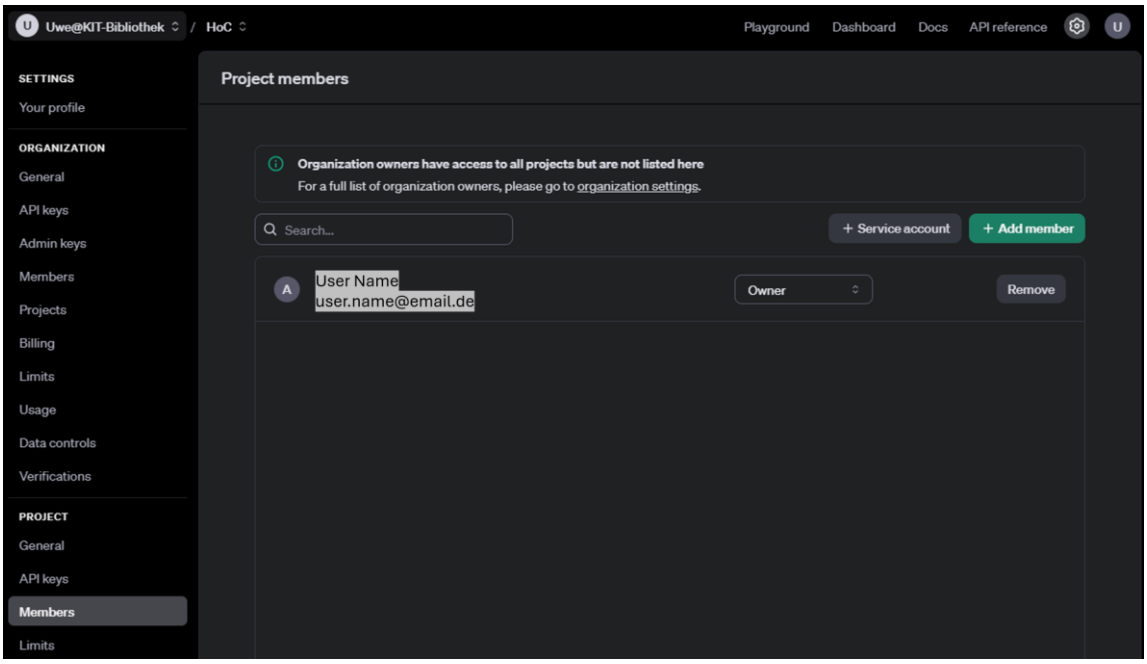

Abbildung: Ebenso das Project-Management (Bsp. Members)

Pflege des Chatbots

Je OpenAI Assistant können mehreren Personen Owner-Rechte zur Pflege der hinterlegten Dateien gegeben werden. So kann ein kleines Team die Datenbasis pflegen und man ist nicht vom Besitzer des OpenAI Accounts abhängig. Auf diese Weise kann auch das Sharen eines OpenAI Accounts durch eine Gruppe von Personen und somit die Preisgabe eines Passwort an mehrere Admins vermieden werden.

Wenn sich Dokumente ändern, müssen diese derzeit noch manuell in den OpenAI Assistant hochgeladen werden. Dabei muss (noch!) darauf geachtet werden, dass eine Datei immer zum Vector Store hinzugefügt wird, auch wenn deren Dateiname bereits im Vector Store vorhanden ist. Sie wird also nicht überschrieben. Anhand des Zeitstempels muss dann die alte Datei – sofern erwünscht - manuell gelöscht werden.

Ursächlich dafür ist, dass beim Upload einer Datei diese immer als Objekt im File Storage abgelegt und mit einer File ID versehen wird. Es wird nicht geprüft, ob eine Datei mit demselben Namen bereits vorhanden ist; nur die File ID muss eindeutig sein, nicht der Name. Im File-Storage-Bereich von OpenAI werden alle Versionen solcher hochgeladenen Dokumente gespeichert. Das Löschen einer Datei aus dem Vector Store löscht diese nicht automatisch im File Storage.

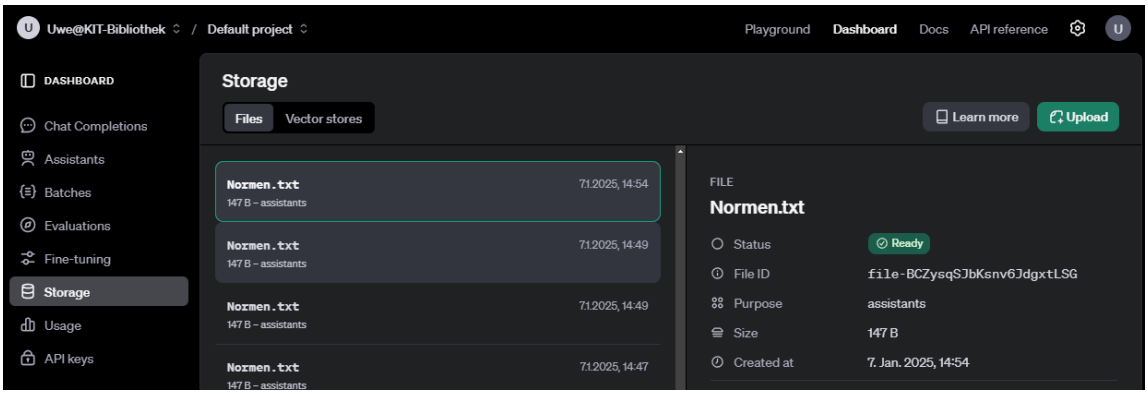

Abbildung: Doppelte Dateinamen im Bereich File-Storage

Hier kann per File API für Ordnung gesorgt werden, solange OpenAI dazu noch kein nutzerfreundlicheres GUI anbietet.

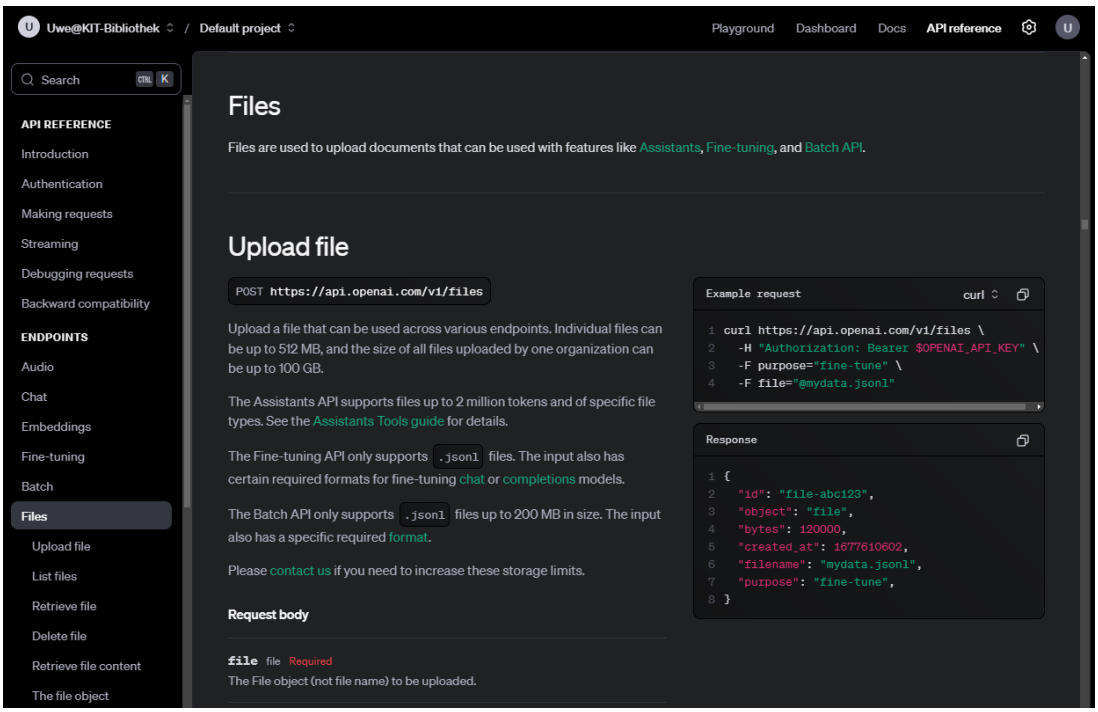

Abbildung: API reference „Files“

Am KIT wird daher überlegt, ob dazu eine eigene Upload-Funktion implementiert werden soll, damit die Pflege komplett via KI-Chatbot Admin-Interface von Endnutzern ohne OpenAI Account durchgeführt werden kann. Aber für den Start des Bots sollte mit nur geringem Entwicklungsaufwand eine Lösung erstellt werden, da all diese Komfort-Features nur dann Sinn machen und nötig sind, wenn die Antwort-Qualität zufriedenstellend ist. Und das entscheidet am Ende die Praxis! Man darf nicht vergessen, dass es sich beim KI-Chatbot um ein Experiment handelt.

Administration

Das Wichtigste bei einem Chatbot ist die Kenntnis darüber, ob die Fragen, die von den Endnutzern gestellt werden, vom Bot beantwortet werden können. Dazu bietet der Bot momentan drei Web-Zugänge für Administratoren (IT-Abteilung, Benutzung etc.).

- user_messages
- unanswered_questions
- feedback

Die **user_messages** enthalten alle Fragen und verschaffen einen generellen Überblick.

Der **feedback** Endpunkt listet die Rückmeldungen von Nutzern auf, die wir hoffentlich dann erhalten, wenn der Bot keine zufriedenstellende Antwort gegeben hat. In diesem Fall wird der gesamte Chat-Verlauf mit allen Nutzerfragen und Antworten der KI sowie dem Feedback-Text erfasst.

Der unserer Einschätzung nach wichtigste Teil dürfte jedoch aus den **unanswered_questions**, also den nicht beantworteten Fragen bestehen, zu denen kein Feedback gesendet wurde. Diese werden von uns intern per Software erkannt und in einer separaten Datenbank-Tabelle abgespeichert. So können Lücken in der zugrundeliegenden Wissensbasis (local knowledge base) erkannt und an die tatsächlich eingehenden Fragen angepasst werden. Dieses Finetuning wird durch das betreuende Personal durchgeführt und hat nichts mit dem Finetuning von Sprachmodellen zu tun.

Erste Erfahrungen

Zum Zeitpunkt des Entstehens dieses Beitrags liegen nur geringe Erfahrungswerte vor. In den ersten Tagen gingen neben echten Nutzerfragen zu den Services der KIT-Bibliothek auch etliche Anfragen ein, die erkennen ließen, dass die Nutzenden eine Literaturrecherche im Sinn hatten. Es musste deutlicher hervorgehoben werden, dass es sich beim KI-Chatbot um einen reinen Service-Chatbot und kein Rechercheinstrument handelt. Die erste Abbildung in diesem Beitrag zeigt die inzwischen geänderte Welcome Message beim Start des Bots.

Geplant ist eine Klassifikation der Fragen durch KI, um z.B. im Fall des Versuchs einer Literaturrecherche via KI-Chatbot einen gezielten Hinweis zu geben. Denkbar wäre auch die Erstellung einer Suchanfrage für den KIT-Katalog und die Ausgabe eines entsprechenden Recherche-Links seitens des Chatbots.

Für einige Bibliotheken aus Karlsruhe wurden Demo-Instanzen des KI-Chatbots aufgesetzt. Da bei diesen Bibliotheken noch keine eigenen OpenAI Accounts vorlagen, und somit auch kein OpenAI Assistant von ihnen mit deren Daten befüllt werden konnte, wurden von der IT der KIT-Bibliothek deren Webpräsenzen mit dem oben erwähnten GPT Crawler abgezogen und als OpenAI Projects in den OpenAI Account der KIT-Bibliothek eingepflegt. Die Rückmeldungen dazu waren bisher sehr gut. Dies lässt hoffen, dass auch in vielen anderen Fällen die Pflege der Website und deren automatisches Crawlen für den Aufbau einer guten lokalen Wissensbasis für einen KI-Chatbot ausreicht. Am KIT wird derzeit ein hybrider Ansatz angewandt. Teile der Daten sind manuell gepflegte Textdokumente, welche um gecrawlte JSON-Dateien angereichert werden.

Mehrsprachigkeit

Die Wissensbasis besteht aus Dokumenten in deutscher Sprache. Für die gestellten Fragen werden wie bereits oben erwähnt Embeddings berechnet. Die Suche nach den Dokumenten, in denen die Antwort zu finden ist, erfolgt semantisch anhand der Embeddings. Daher kann eine Frage in einer beliebigen Sprache gestellt werden, die vom Embedding-Modell und vom Sprachmodell, welches die Antwort erstellt unterstützt werden. Die Oberfläche ist momentan nur einsprachig, i18n (internationalization) ist aber angedacht.

Datenschutz und mögliche weitere Anwendungen

Da die Dokumente derzeit von einem US-Service-Provider verarbeitet werden, sollte man nur solche Informationen hochladen, die öffentlich zugänglich sind. Das ist bei der KIT-Bibliothek jedoch der Fall.
Die Gesellschaft für wissenschaftliche Datenverarbeitung in Göttingen (GWDG) bietet für deren Hosting-Lösung „Chat AI“ eine API an, die zur OpenAI API kompatibel ist. Eventuell wird sich deren RAG-Anwendung ebenfalls an die OpenAI Assistant API anlehnen. Dann wäre eine Umstellung auf einen deutschen KI-Service-Provider sehr leicht möglich und dort könnten dann auch guten Gewissens eher intern benutze Dokumente hochgeladen werden.

Fazit und Ausblick

Der vorgestellte Ansatz basiert auf der Grundidee, möglichst viele technische Verfahren (wie RAG, Nutzung von LLMs) von KI-Service-Providern zu nutzen und nicht alles selbst zu implementieren bzw. zu betreiben. Die Entwicklung in diesem Bereich ist sehr schnell. Mit kleinen Teams kann man dieser nur schwer folgen. Daher wurde auf die Cloud und API-Anbindung gesetzt. Das Hosting des Service-Chatbots musste implementiert werden, um die Datenschutzbestimmungen zu erfüllen. Der Fokus liegt auf der Optimierung der bereitgestellten Dokumente, damit die KI möglichst gute Antworten geben kann.

Prinzipiell können Bots auch zur Beantwortung von Fragen zu abteilungsinternen Workflows erstellt werden, sofern solche Unterlagen keine kritischen Informationen (s. Datenschutz) enthalten. Auch öffentlich zugängliches Fragenmaterial von Fakultäten oder Dienstleistern in Universitäten (RZ, DAAD etc.) könnte die Grundlage für einen Service-Chatbot bilden, der einen Großteil aller Fragen schon im Vorfeld und rund um die Uhr beantworten kann.

Das Stellen von Fragen in natürlicher Sprache hat in den letzten zwei Jahren seit dem Erscheinen von ChatGPT stark zugenommen und die klassische Google-Suche zwar nicht ersetzt, aber zumindest sehr gut ergänzt. Daher bietet das Bereitstellen eines Service-Chatbots neben der Pflege der eigenen Webpräsenz bzw. Website einen zusätzlichen Mehrwert.

Zum Zeitpunkt der Erstellung dieses Beitrags liegen nur geringe Erfahrungswerte vor. Daher sind wir gespannt, wie der Versuch „KI-Chatbot“ ankommt. Das letzte Wort haben am Ende die Nutzerinnen und Nutzer!

Danksagung

Ein Projekt lebt vom Austausch mit anderen Menschen, deren Inspirationen und der Teamarbeit. Daher sei allen ausdrücklich gedankt, die uns durch ihr Interesse an unserem KI-Projekt und dem in vielen Gesprächen erhaltenen Feedback gute Anregungen gegeben haben.

Ein besonderer Dank gilt Thomas Griesbaum von der Fak. f. Informatik, dessen simple Frage "Wie erfahrt ihr denn, was der Chatbot nicht beantworten konnte?" einen wichtigen Anstoß gegeben hat.

Auch der gute Dialog mit den Karlsruher Partner-Bibliotheken hat unser Projekt befruchtet und die Idee und Umsetzung des Hostings reifen lassen.

Ebenso die Unterstützung der Direktion der KIT-Bibliothek und Arne Upmeiers Rückhalt und Geduld und immer gute Laune haben uns sehr beflügelt.

Last but not least ein großes Dankeschön an die DGI und Margarita Reibel-Felten, deren Begeisterung für KI ungebrochen ist und auf deren Nachfrage dieser Beitrag entstanden ist.

Anlage 1: Hosting für Dritte

Erforderliche Schritte, um diesen Dienst zu nutzen:

- API Key von OpenAI besorgen - dies erfordert keinen ChatGPT Plus Account!
- Zahlungsmittel hinterlegen und Geld aufbuchen zur Nutzung der OpenAI API
- Einen OpenAI Assistant anlegen und die Assistant ID notieren
- Dokumente bei File-Search hochladen
- DNS-Eintrag besorgen

Nachdem man bei https://platform.openai.com den Assistant angelegt hat, kann man alle Tests selbst durchführen und prüfen, wie gut die Antworten der KI sind.

Für unsere Container benötigen wir nur den DNS-Hostnamen, den API-Key und die Assistant-ID.

Die Hosting-Kosten seitens des KIT sind derzeit noch in Klärung.

Fragen?

Fragen beantworten wir gerne telefonisch unter 0721-608-46076 oder per Mail an Uwe.Dierolf@kit.edu

Über die Autoren

Uwe Dierolf ist Diplom-Informatiker und ist seit 2001 Leiter der IT-Abteilung an der KIT-Bibliothek in Karlsruhe. Er beschäftigt sich seit dem Erscheinen von ChatGPT mit KI und hat einige Praxis-Workshops u.a. bei der DGI zu diesem Thema gegeben.

Tobias Kurze ist Diplom-Informatiker und seit 2015 Mitglied der IT-Abteilung an der KIT-Bibliothek. Er hat große Teile des Chatbots in Python implementiert und dafür eine Docker basierte CI/CD Pipeline realisiert, die die Grundlage für das Hosting des KI-Chatbots bildet.

Frank Polgart ist Diplom-Physiker und seit 2024 Mitglied der IT-Abteilung an der KIT-Bibliothek in Karlsruhe. Er hat etliche RAG-Anwendungen aufgesetzt und entwickelt und den Einsatz von selbsterstelltem RAG und alternativen zu OpenAI untersucht.

Michael Skarupianski ist Master of Science und seit 2013 Mitglied der IT-Abteilung der KIT-Bibliothek in Karlsruhe. Er entwickelte und optimierte das Web-Design und die Usability.

Bernadette Breyer ist Bibliotheksassistentin an wissenschaftlichen Bibliotheken und seit 2003 an der KIT-Bibliothek in der Abteilung Benutzung tätig. Sie pflegt die „local knowledge base“ des KI-Chatbots und ist schon seit Projektbeginn Mitglied des KI-Teams.