IPA Facharbeit Applikationsentwicklung

OPENAI POWERED SUPPORT CHATBOT

Dominik Dierberger
FUTURELOG AG | BERUFS BILDUNG BADEN

Inhaltsverzeichnis

Teil 1: Obligatorisches Kapitel

1.1 Aufgabenstellung

1.1.1 Ausgangslage

Kundensupport ist ein unverzichtbarer Aspekt jedes Unternehmens, da schnelle und direkte Hilfe einfach grundlegend ist.

Trotzdem bringt die traditionelle Art, diesen Support durch Mitarbeiter zu leisten, einige Nachteile mit sich. Mitarbeiter müssen sich Zeit nehmen, um Kundenanfragen zu bearbeiten, das eigentliche Problem richtig verstehen und dann eine verständliche Antwort formulieren. Dabei verliert man viel Zeit, die eigentlich effizienter eingesetzt werden könnte.

Zudem ist ein 24/7-Support teuer und verlangt nach einem hohen Personaleinsatz. Viele der Anfragen könnten allerdings automatisiert beantwortet werden, was den Mitarbeitern eine Menge Arbeit abnehmen würde.

Das ist der Punkt, an dem meine Abschlussarbeit ins Spiel kommt: Mit den neuesten Entwicklungen im Bereich der künstlichen Intelligenz und den Fortschritten bei LLM[1] ´s ist es mittlerweile machbar, fortschrittliche Chatbots zu entwickeln. Diese können Dokumentationen verarbeiten und Kundenfragen jederzeit beantworten, und das auf eine kosteneffiziente Weise.

1.1.2 Detaillierte Aufgabenstellung

Was soll genau erarbeitet werden?

Die Aufgabe umfasst das Training und die Implementierung eines AI-Chatbots, der in das Suppliermodul integriert wird. Dies beinhaltet:

Integration:

Entwicklung einer Methode zur Einbettung des AI-Chatbots als iframe oder als Pop-Up-Fenster in das Suppliermodul.

Gewährleistung einer nahtlosen Integration in die bestehende Benutzeroberfläche für direkten Zugriff auf Supportleistungen.

Interaktivität:

Implementierung einer direkten, textbasierten Kommunikationsmöglichkeit zwischen Kunden und AI-Chatbot.

Schaffung einer Benutzererfahrung, die einen sofortigen und effizienten Austausch ermöglicht.

[1] Large language model

Testen der Funktionalität:

Durchführung einer umfassenden Überprüfung des AI-Chatbots.

Testen mit verschiedenen Supportbeispielen und Durchführung von Eingabevalidierungen zur Sicherheitsüberprüfung.

Dokumentation:

Erstellung eines Benutzerhandbuchs für Kunden.

Bereitstellung einer technischen Dokumentation, die den Lösungsansatz und die technischen Details erläutert.

Was ist das erwartete Ergebnis?

Das erwartete Ergebnis ist ein funktionsfähiger, sicherer und effizienter AI-Chatbot, der:

In das Suppliermodul integriert ist und über iframe oder als Pop-Up zugänglich gemacht wird.

Direkte, textbasierte Interaktivität mit den Nutzern ermöglicht, um Anfragen in Echtzeit zu bearbeiten.

Gründlich getestet wurde, um Zuverlässigkeit und Leistungsfähigkeit zu gewährleisten.

Durch detaillierte Dokumentation für Nutzer und Techniker unterstützt wird.

Das Ziel ist es, durch den AI-Chatbot eine verbesserte Nutzerzufriedenheit und eine effizientere Supportleistung ohne menschliches Zutun zu erreichen.

Was zählt zur Eigenleistung?

Zur Eigenleistung zählen folgende Aspekte:

Indexierung der vorhandenen Dokumentation:

Analyse und Sortierung der vorhandenen Inhalte: Verstehen der Struktur und des Inhalts der Dokumentation des Suppliermodul, um relevante Informationen für den AI-Chatbot zu identifizieren.

Erstellung eines Index: Aufbau eines Systems zur Kategorisierung und Verschlagwortung der Dokumente, um die Suche und das Retrieval von Informationen zu vereinfachen.

Aufbereitung für maschinelles Lernen: Formatierung und Anpassung der Dokumente in ein für den AI-Chatbot lesbares und interpretierbares Format, um das Training und die Informationsabfrage zu erleichtern.

Integration in den Lernprozess des AI-Chatbots: Einspeisung der indexierten Dokumente in die Lernumgebung des AI-Chatbots, um die Datenbasis für Antworten und Interaktionen zu schaffen.

Diese Schritte sind entscheidend für die Funktionsfähigkeit des AI-Chatbots, da sie das Fundament für das Wissen des Bots bilden, auf dem er seine Antworten generiert und die Nutzeranfragen bearbeitet.

Setup in MS Azure:

Einrichtung und Konfiguration des AI-Chatbots im Microsoft Azure Cloud-Service.

Integration in das Suppliermodul[2]:

Technische Umsetzung der Integration des AI-Chatbots in das Suppliermodul als iframe oder Pop-Up, unter Berücksichtigung der Benutzererfahrung und der technischen Anforderungen.

[2] Wird später detailliert erläutert

1.2 Projektaufbauorganisation

Mein 1-jähriges Praktikum absolviere ich bei der FutureLog AG in Baar ZG.

Meine IPA werde ich an meinem normalen Arbeitsplatz austragen.

Während der IPA werde ich nach der IPERKA Methode arbeiten.

- Informieren
- Planen
- Entscheiden
- Realisieren
- Kontrollieren
- Auswerten

Am Anfang werde ich mich zuerst über das Projekt informieren.

Danach folgt die Planung. Hier werde ich mittels Diagramme einen Plan aufstellen, wie die Applikation aufgebaut sein soll und wie die einzelnen Prozesse zusammenhängen.

Nach dem Planen gehe ich zum Entscheiden, hier lege ich fest, wie ich die Arbeit erledigen soll.

Dann kommen wir zum Realisieren. Hier werde ich mein Vorgehen, wie auch mein Programm erklären, wie es funktioniert und wie ich es aufgesetzt habe.

Im Kontrollieren-Teil werde ich meine Arbeit dahingehend kontrollieren, ob alle Anforderungen erreicht wurden, und dass alle IPA-Vorgaben erfüllt wurden.

Am Ende werde ich im Auswerten-Teil noch einmal über die Arbeit reflektieren. Hier werde ich aufzeigen, was mir gut gelungen ist, wie auch, was ich in zukünftigen Projekten besser machen könnte.

1.2.1 FutureLog AG

Die FutureLog AG bietet eine umfassende, Cloud-basierte Procure-to-Pay-Plattform speziell für die Gastgewerbe- und Hotellerie Branche. Die Plattform automatisiert und integriert den gesamten Beschaffungsprozess, von der Bestellung über das Inventarmanagement bis hin zur Rechnungsbearbeitung. Durch die Nutzung der Cloud-Technologie ermöglicht FutureLog seinen Kunden einen sicheren und transparenten Zugriff auf alle Transaktionen und Prozesse.

Im Suppliermodul der FutureLog AG können Lieferanten ihre Artikel verwalten. Hier können sie ebenfalls neue Preise setzen, wie auch Artikeldaten ändern oder neue Artikel erfassen.

Schon vor meinem Praktikum arbeite ich bei der FutureLog AG als Developer hauptsächlich im Backend.

Meine Arbeit während des Praktikums ist es das alte Suppliermodul (Frontend) mit meinem Mitpraktikanten neu zu designen und zu entwickeln.

1.3 Deklaration der Vorkenntnisse

Python: Gute Kenntnisse mit der FLASK library, da ich hauptsächlich für die Entwicklung von API'S in meinem Betrieb verantwortlich war.

ASP.NET: Basic Kenntnisse, in meinem früheren Nebenjob musste ich die bisherige Codebase analysieren und dokumentieren.

Docker: Gute Kenntnisse, da alle entwickelten API'S durch Docker Container deployed wurden.

Azure: Einige API'S wurden schon auf Microsoft Azure deployed, dabei habe ich mich in meiner Freizeit schon einige Male mit Azure auseinandergesetzt.

Svelte: 1-2 kleine Projekte in eigenem Interesse. Basic Kenntnisse in HTML, CSS und JS aus der Ausbildung.

1.4 Deklaration der Vorarbeiten

Direkte Vorarbeiten habe ich keine gelistet.

1.5 Deklaration der Firmenstandards

Hosting in Azure

Alle Dienstleistungen der FutureLog AG werden mit Azure gehostet oder sind im Prozess diese nach Azure zu migrieren. Daher werde auch ich meine API'S wie auch meine Frontend Webapp in Azure hosten.

Deployen mittels Container

Wie auch im Suppliermodul werde ich auch in diesem Projekt meine Scripts und Applikationen mittels Images und Containern hosten. Dies ermöglicht einfaches Scaling wie auch eine hohe Redundanz.

Corporate Identity -> Orange / Grau

Das CI der FutureLog AG besteht hauptsächlich aus den Farben Orange und Grau. Daher werde ich für meine Arbeit diese Farbtöne benutzen.

1.6 Zeitplan

				06.03.2024		07.03.2024		08.03.2024		11.03.2024		12.03.2024		13.03.2024		14.03.2024		15.03.2024		18.03.2024		19.03.2024	
Tätigkeit	Erfüllt	Stunden	Werte	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00	08:00-12:00	13:00-17:00
1. Informieren																							
Informieren über OWASP Top Ten Kriterien	100%	1	Soll																				
		1	Ist																				
Analyse Bewertungskriterien IPA	100%	1	Soll																				
		1	Ist																				
Analyse Aufgabenstellung	100%	1	Soll																				
		1	Ist																				
2. Planen																							
Inhaltsverzeichnis und Teil 1 der Doku erstellen	100%	5	Soll																				
		5	Ist																				
PAP erstellen	100%	1	Soll																				
		1	Ist																				
Front End MockUP erstellen	100%	2	Soll																				
		2	Ist																				
Datei Struktur planen	100%	1	Soll																				
		1	Ist																				
Deployment planen	100%	3	Soll																				
		3	Ist																				
3. Entscheiden																							
Vergleich Varianten mit Nutzwertanalyse	100%	1	Soll																				
		1	Ist																				
4. Realisieren																							
Zusammenstellung der Dokumentation des Supplier Moduls	100%	4	Soll																				
		4	Ist																				
Erstellung des GPTS	100%	2	Soll																				
		2	Ist																				
Einbindung des GPTS mit API	100%	6	Soll																				
		6	Ist																				
Erstellung des Python scripts	100%	5	Soll																				
		10	Ist																				
Erstellung des Frontends	100%	6	Soll																				
		10	Ist																				
5. Kontrollieren																							
Erfüllung der Aufgabenstellung überprüfen	100%	3	Soll																				
		3	Ist																				
Erfüllung der Bewertungskriterien überprüfen	100%	3	Soll																				
		3	Ist																				
Testen	100%	8	Soll																				
		6	Ist																				
Bug Fixing	100%	10	Soll																				
		8	Ist																				
6. Auswerten																							
Reflexion der Arbeit	100%	2	Soll																				
		2	Ist																				
Quellenverzeichnis und Glossar erstellen	100%	1	Soll																				
		1	Ist																				
Speziell																							
Expertenbesuche	100%	2.5	Soll																				
		0.5	Ist																				
Reserve	100%	11.5	Soll																				
		8.5	Ist																				
Gesamtzahl der Stunden		80	Soll																				
		80	Ist																				
		0	Diff																				
Meilensteine																							
Zeitplan Erstellen																							
Gerüst Dokumentation																							
Planen Fertigstellen																							
Entscheidungen gefällt																							
Umsetzung der Arbeit																							
Alles Kontrolliert																							
Reflexion der Arbeit																							
Dokumentation Fertigstellen																							
Arbeit abgeben																							

Abbildung 1 Zeitplan

1.7 Arbeitsjournal

DATUM	6.3.2024
TÄTIGKEITEN	Zeitplan erstellt Dazu habe ich angefangen mich zu Informieren und konnte diesen Teil ebenfalls abschliessen. Als Letztes habe ich den 1. Teil der Dokumentation fertig gestellt.
UNGEPLANTE TÄTIGKEITEN	-
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Da ich schon einen Plan habe, wie ich den praktischen Teil der IPA umsetzten möchte, konnte ich gezielt mich auf das Informieren fokussieren und dieses ebenfalls abschliessen.
MISSERFOLGE	-
FAZIT	Ich konnte das erledigen, was ich mir vorgenommen habe.

DATUM	7.3.2024
TÄTIGKEITEN	Ich habe alle Diagramme und Planungen fertiggestellt. Daraufhin habe ich mir 3 Varianten für die Programmierung überlegt und nach der Bewertung für eine entschieden.
UNGEPLANTE TÄTIGKEITEN	-
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Ich konnte Kapitel 2 “Planen” und Kapitel 3 “Entscheiden” erfolgreich abschliessen
MISSERFOLGE	-
FAZIT	Kapitel 2 und 3 der Doku konnten erfolgreich abgeschlossen werden.

DATUM	8.3.2024
TÄTIGKEITEN	Nachdem ich alle Dokumente bezüglich der Dokumentation des Suppliermoduls zusammengestellt habe, konnte ich angefangen das GPT-Model zu erstellen. Dazu habe ich ebenfalls noch die AI Search Funktion eingerichtet und das Handbuch für das Suppliermodul in unser Azure Storage hochgeladen.
UNGEPLANTE TÄTIGKEITEN	Aufsetzten und Einbinden der Azure AI Search Dienstleistung
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Handbuch zusammengestellt, GPT-Model erstellt.
MISSERFOLGE	-
FAZIT	Alles konnte vorbereitet werden, um das nächste Mal mit dem Programmieren des Scripts zu beginnen.

DATUM	11.3.2024
TÄTIGKEITEN	Programmierung der Python API
UNGEPLANTE TÄTIGKEITEN	-
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Der Chatbot konnte erfolgreich mit dem in Azure gespeicherten Handbuch angesprochen werden.
MISSERFOLGE	-
FAZIT	Funktion läuft -> Funktion muss nun nur noch in die API eingebunden werden.

DATUM	12.3.2024
TÄTIGKEITEN	Hosting der Python API in Azure
UNGEPLANTE TÄTIGKEITEN	Einerseits konnte ich die Python API sofort hosten, jedoch musste ich 1-2 Stunden die Verbindung debuggen.
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Python Flask API konnte erfolgreich gehostet werden. Ebenfalls konnten gute Fortschritte im Frontend erzielt werden.
MISSERFOLGE	Frontend Container startet nicht
FAZIT	Das Backend konnte abgeschlossen werden.

DATUM	13.3.2024
TÄTIGKEITEN	Heute habe ich das Frontend erstellt und dieses in Azure gehostet. Danach konnte ich die URL des Frontends nehmen und diese mit einem Pop-Up im Suppliermodul einbinden.
UNGEPLANTE TÄTIGKEITEN	Ich musste das Dockerfile des Frontends überarbeiten. Da, sich der Container am Anfang nicht starten liess, lag das Problem am Ende bei den Building Instructions. Ebenfalls können Ports in Azure nicht aus dem Dockerfile gewonnen werden. Daher musste ich den Port als Umgebungsvariable definieren.
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Ich konnte heute die Realisierungsphase abschliessen.
MISSERFOLGE	-
FAZIT	Ich konnte heute die Realisierungsphase erfolgreich abschliessen.

DATUM	14.3.2024
TÄTIGKEITEN	Nachdem ich gestern mit der Realisierungsphase fertig wurde, konnte ich heute mit Kapitel 2.6 dem Kontrollieren anfangen. Ich konnte ebenfalls anfangen, zu testen.
UNGEPLANTE TÄTIGKEITEN	-
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Fertigstellung der ersten zwei Unterkapiteln des Kontrollieren Kapitels.
MISSERFOLGE	-
FAZIT	Ich konnte heute gute Fortschritte im Kontrollieren-Teil absolvieren. Morgen sollte ich mit dem Testing-Teil fertig werden.

DATUM	15.3.2024
TÄTIGKEITEN	Heute konnte ich das Testing abschliessen, welches ich gestern begonnen habe. Dazu konnte ich noch allfällige Bugs beheben, so dass die Applikation wie gewünscht funktioniert.
UNGEPLANTE TÄTIGKEITEN	-
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Testing abgeschlossen, und fertige App veröffentlicht.
MISSERFOLGE	-
FAZIT	Ich konnte heute das Testing abschliessen. Dazu habe ich einige kleine Bugs behoben und Version 1.0 auf Azure veröffentlicht.

DATUM	18.3.2024
TÄTIGKEITEN	Heute stand das letzte Kapitel auf dem Zeitplan. In Kapitel 2.7 “Auswerten” habe ich die Arbeit reflektiert und ein Schlusswort geschrieben. Dazu habe ich noch einige Rechtschreibfehler ausgebessert und generell die Dokumentation noch feingeschliffen.
UNGEPLANTE TÄTIGKEITEN	-
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Fertigstellung Grobtext IPA
MISSERFOLGE	-
FAZIT	Ich bin heute mit dem Grobtext der IPA fertig geworden, ebenfalls habe ich mit einem letzten Feinschliff der Dokumentation angefangen.

DATUM	19.3.2024
TÄTIGKEITEN	Am letzten Tag der IPA habe ich noch kleine Änderungen in der IPA-Dokumentation vorgenommen, wie auch alle Dateien aufgeräumt. Am Ende des Tages habe ich die Dokumentation abgegeben.
UNGEPLANTE TÄTIGKEITEN	-
ALLFÄLLIGE ÜBERZEIT	-
ERFOLGE	Abgabe der IPA
MISSERFOLGE	-
FAZIT	Heute konnte ich die IPA abschliessen. Nach einigem Feinschliff gab es für mich heute nichts weiter zu tun, als die Dokumentation auf Pk.Org hochzuladen.

Teil 2: Projekt Dokumentation

2.1 Kurzfassung des IPA-Berichts

In folgendem Abschnitt werde ich meine IPA kurz zusammenfassen.

2.1.1 Ausgangssituation

Da wir in der Firma immer wieder kontaktiert werden, um Kunden (in diesem Falle Lieferanten) bei Problemen zu unterstützen, war es an der Zeit eine automatisierte Lösung für diese Art der Supportanfragen zu entwickeln.

Einen Chatbot zu erstellen, welcher durch eine hochgeladene Dokumentation / Handbuch den Nutzer unterstützen kann, ist das Ziel.

Dieser Chatbot sollte auch in der Zukunft noch einfach zu warten und zu erweitern sein und die Qualität seiner Antworten permanent verbessern.

2.1.2 Umsetzung

Da ein Grossteil unserer Infrastruktur auf Azure gehostet wird, wurde auch hier beim Hosting des Chatbots auf Azure gesetzt. Jedoch werden für dieses Projekt auch noch einige andere Azure Services benötigt, welche mit dem Hosting eher weniger zu tun haben.

In meiner Arbeit werde ich auf 3 Technologien setzen.

1. Wird die API mit Python und Flask programmiert.
2. Das Frontend wird mittels des JavaScript Frameworks Svelte programmiert.
3. Um alle meine Skripte und Programme zu starten, benutze ich Docker, um sie danach auch gleich als Images zu deployen.

Alle Projekt Daten werden auf GitHub in unserem firmeninternen Team zur Verfügung gestellt.

2.1.3 Ergebnis

Am Ende der Arbeit konnte ich den Chatbot voll funktionstüchtig fertigstellen.

Die Ladezeiten und Antwortzeiten des Bots sind gering, und die App läuft Ressourcen optimiert auf Azure.

In Zukunft kann der Bot auch einfach erweitert werden. Neue Dokumentationen / Handbücher können jederzeit auf Azure hochgeladen werden.

Die Azure Services erledigen den Rest.

2.2 Informieren

(OWASP, 2021)

2.2.1 OWASP Top Ten

2.2.1.1 Broken Access Control

Dies bezieht sich auf Schwächen in den Zugriffskontrollmechanismen. Wenn Zugriffskontrollen nicht korrekt umgesetzt werden, können unbefugte Benutzer Zugriff auf geschützte Bereiche oder Funktionen erhalten.

2.2.1.2 Cryptographic Failure

Früher bekannt als "Sensitive Data Exposure". Dieses Risiko betrifft das Versagen in der Umsetzung sicherer Kryptografie, was zur Offenlegung sensibler Daten oder zur Kompromittierung des Systems führen kann.

2.2.1.3 Injection

Hierunter fallen Sicherheitslücken, die entstehen, wenn unvertrauenswürdige Daten als Befehle oder Abfragen interpretiert werden. Das bekannteste Beispiel ist SQL-Injection. In dieser Kategorie ist auch Cross-Site Scripting (XSS) enthalten.

2.2.1.4 Insecure Design

Eine neue Kategorie, die sich auf Schwachstellen in der Gestaltung und Architektur der Software bezieht. Um diese Risiken zu mindern, wird der Einsatz von Bedrohungsmodellierung und sicheren Entwurfsmustern empfohlen.

2.2.1.5 Security Misconfiguration

Dieses Risiko entsteht durch falsche Konfigurationen in der Software oder den Systemen. Dazu gehört auch die unsachgemäße Verwaltung von XML External Entities (XXE).

2.2.1.6 Vulnerable and outdated Components

Dies bezieht sich auf die Nutzung von Komponenten mit bekannten Sicherheitslücken. Diese Kategorie unterstreicht die Bedeutung der Aktualität und Sicherheit von Softwarekomponenten.

2.2.1.7 Identification and Authentication failures

Früher bekannt als "Broken Authentication". Diese Kategorie befasst sich mit Schwächen in Identifikations- und Authentifizierungsprozessen.

2.2.1.8 Software and Data Integrity Failures

Diese neue Kategorie bezieht sich auf die Annahmen bezüglich der Integrität von Software-Updates und kritischen Daten, insbesondere im Kontext von Continuous Integration/Continuous Deployment (CI/CD) Pipelines.

2.2.1.9 Security Logging and Monitoring Failures

Früher bekannt als "Insufficient Logging & Monitoring". Dieses Risiko bezieht sich auf unzureichendes Protokollieren und Überwachen von Sicherheitsvorfällen, was die Erkennung und Untersuchung von Sicherheitsverletzungen erschwert.

2.2.1.10 Server-Side request Forgery (SSRF)

Dies ist ein Risiko, bei dem der Angreifer den Server dazu bringt, Anfragen an unerwartete Ziele zu senden. Es hat eine relativ niedrige Vorkommensrate, aber ein überdurchschnittliches Potenzial für Ausnutzung und Schaden.

2.2.2 Analyse Anforderungen IPA

ANFORDERUNGS NUMMER	BESCHREIBUNG	BEWERTUNG
ANFORDERUNG 1	Integration in das Suppliermodul	Integration sowohl als iframe als auch als Pop-Up ohne Fehler bei allen Testläufen. -> Einbindung als Pop-Up
ANFORDERUNG 2	Interaktivität und Antwortzeit	Antwortzeit unter 5 Sekunden und hohe Interaktivität bei allen Supportanfragen.
ANFORDERUNG 3	Testen der Funktionalität	100% Erfolgsquote bei Supportbeispielen und Eingabevalidierung.
ANFORDERUNG 4	Vollständigkeit der Dokumentation	Vollständige Benutzer- und technische Dokumentation, leicht verständlich und zugänglich.
ANFORDERUNG 5	Benutzerfreundlichkeit	Intuitive Benutzeroberfläche und einfache Bedienung ohne Vorkenntnisse.
ANFORDERUNG 6	Antwortqualität	Korrekte und vollständige Antworten auf alle Anfragen.
ANFORDERUNG 7	Sicherheit und Datenschutz	Der Chat implementiert End-zu-End-Verschlüsselung und folgt streng den OWASP Top 10 Empfehlungen zur Absicherung der Anwendung. Es wird regelmäßig überprüft und bestätigt, dass die Anwendung 90-100% der OWASP-Empfehlungen entspricht. Zusätzlich ist eine klare Datenschutzrichtlinie vorhanden, die Datenschutzbestimmungen vollständig einhält.

2.2.3 Analyse Bewertung IPA

BEWERTUNGSNUMMER	BESCHREIBUNG	BEWERTUNG
B1	Kurzfassung des IPA-Berichtes	1. Die Kurzfassung richtet sich an die fachlich kompetenten Leser. 2. Die Kurzfassung enthält die Punkte: Kurze Ausgangssituation - Umsetzung - Ergebnis. 3. Die Kurzfassung enthält zu jedem dieser genannten Punkte die wesentlichen Aspekte. 4. Die Kurzfassung ist nicht länger als eine A4-Seite Text und enthält keine Grafik.
B2	Führung des Arbeitsjournals	1. Die Darstellung ist übersichtlich, klar und verständlich. 2. Alle Aktivitäten gemäss Zeitplan sowie Überzeiten und ungeplante Arbeiten sind erwähnt. 3. Erfolge und Misserfolge sind erwähnt. 4. Alle beanspruchten Hilfestellungen sind erwähnt und begründet.
B3	Reflexionsfähigkeit	1. Hat im Arbeitsjournal seine Vorgehensweise und das Ergebnis kritisch hinterfragt. 2. Vergleicht mögliche Lösungs-Varianten oder begründet, weshalb es keine Varianten gibt. 3. Zieht im Schlusswort nachvollziehbare Schlüsse aus seiner eigenen Reflexion. 4. Das Schlusswort enthält eine persönliche Bilanz.
B4	Gliederung	1. Der IPA-Bericht ist in eine zu den Themen und Schwerpunkten passende Kapitelstruktur unterteilt. 2. Der IPA-Bericht ist übersichtlich gegliedert und eingesetzte Überschriften sind mit entsprechenden Inhalten gefüllt. 3. Die Reihenfolge der Themen im IPA-Bericht ist aufeinander abgestimmt. 4. Die Gestaltung von Überschriften, Texten und Grafiken erleichtert den

		Lesefluss und behindert ihn nicht.
B5	Prägnanz	Der Text des IPA-Berichtes ist hinsichtlich der Prägnanz bestmöglich gestaltet. Er ist durchgängig oder mit höchstens einer Ausnahme so ausführlich wie für das Verständnis erforderlich und enthält weder Ballast noch unnötige Redundanzen.
B6	Formale Vollständigkeit des IPA-Berichts	1. Der IPA-Bericht ist in Teil 1 (obligatorische Kapitel) und Teil 2 (Projekt-Dokumentation) unterteilt. Ein allfälliger Quellcode ist im Anhang vorhanden; 2. Teil 1 enthält: Projektaufbauorganisation, Zeitplan, Arbeitsjournal; 3. Der IPA-Bericht enthält ein aktuelles Inhaltsverzeichnis; 4. …zu sämtlichen Quellen besteht ein schriftlicher Nachweis, die referenzierten Quellen sind gültig und verlässlich; 5. …auf allen Seiten (optional Titelblatt) eine Kopf- oder Fusszeile mit dem aktuellen Druckdatum und dem Namen des Kandidaten; 6. …ein alphabetisch sortiertes Glossar mit korrekten Erläuterungen der verwendeten Fachbegriffe und Abkürzungen, welche einer aussenstehenden Fachperson unbekannt sein dürften.

B7	Sprachlicher Ausdruck und Stil / Rechtschreibung und Grammatik	1. Die Sprache ist durchgehend klar verständlich (Satzbau, Wortstellungen), in einem flüssigen Stil sowie in vollständigen und ausformulierten Sätzen geschrieben. 2. Fachbegriffe werden korrekt und adressatengerecht eingesetzt. 3. Der IPA-Bericht enthält nur wenige Rechtschreib- oder Grammatikfehler.
B8	Darstellung	1. Die Darstellung enthält eine geeignete Seitennummerierung. 2. Der Seitenumbruch ist sinnvoll oder behindert den Lesefluss nicht. 3. Jede Seite enthält Informationen und nicht nur eine einzelne Textzeile oder Überschrift. 4. Die Darstellung ist zweckmässig und sauber
B9	Grafiken, Bilder, Diagramme und Tabellen	1. Es werden an vernünftigen Stellen Grafiken, Bilder, Diagramme oder Tabellen eingesetzt, um die Inhalte im IPA-Bericht besser darzustellen und den Text verständlicher zu machen; 2. Die Wahl der Darstellungen ist durchgehend geeignet; 3. Die Darstellung ist kontrastreich und optisch gut lesbar (als Referenz dient der Ausdruck auf Format A4); 4. Die Darstellungen sind inhaltlich verständlich; 5. Die Darstellungen sind aussagekräftig; 6. Die Darstellungen sind im Text oder in einer Legende erklärt; 7. Die Darstellungen passen zum Kontext.
B10	Durchführung und Auswertung der Tests	1. Die Testdurchführung basiert auf dem Testkonzept; dies ist entsprechend dokumentiert (inkl. allfälliger Abweichungen davon). 2. Alle Testresultate sind

		korrekt und übersichtlich dokumentiert. 3. Das Testprotokoll beinhaltet Angaben über den Testzeitpunkt, die Testperson sowie allfällige spezifische Informationen. 4. Es wird ein aussagekräftiges Fazit zum Testergebnis (pro Testfall) gezogen und es werden allfällige notwendige Massnahmen/Empfehlungen beschrieben.

2.3 Planen

2.3.1 Front End Mock Up

Da mein Schwerpunkt im Backend liegt, will ich nicht allzu viel Zeit für das Frontend verwenden, daher war der Plan, das Frontend einfach anzupassen mit einem einfachen Chat Feld. Dazu soll dieses neue Feature nicht für die Kunden (Lieferanten) störend sein.

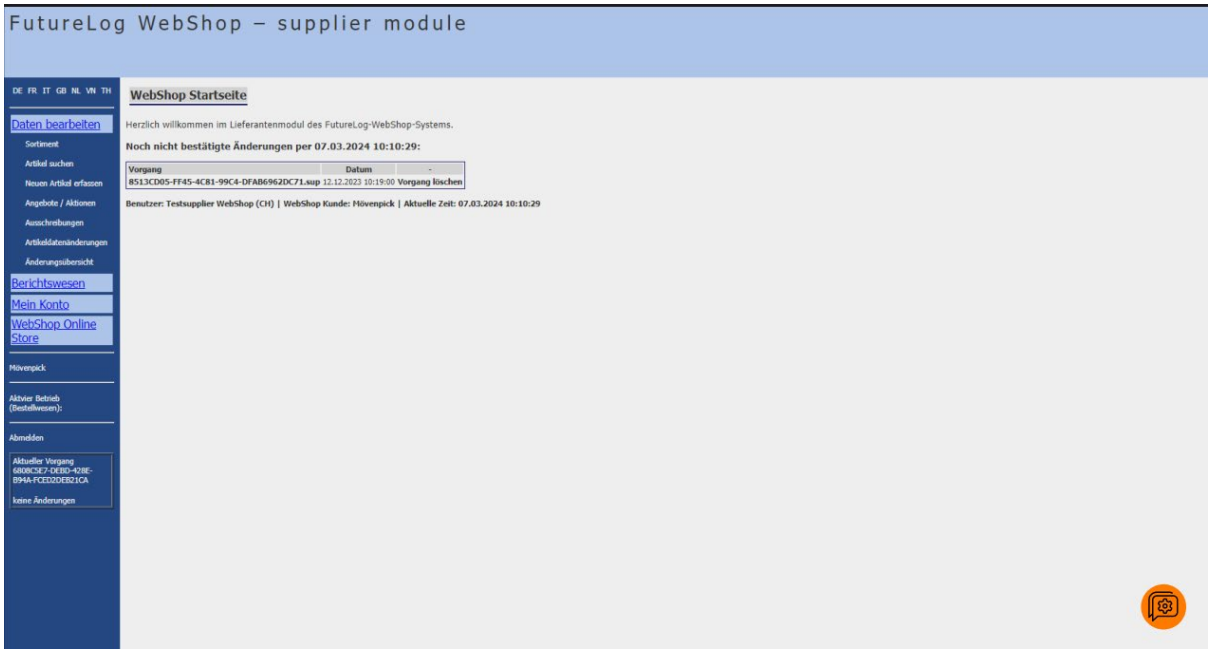

Abbildung 2 Mock Up 1

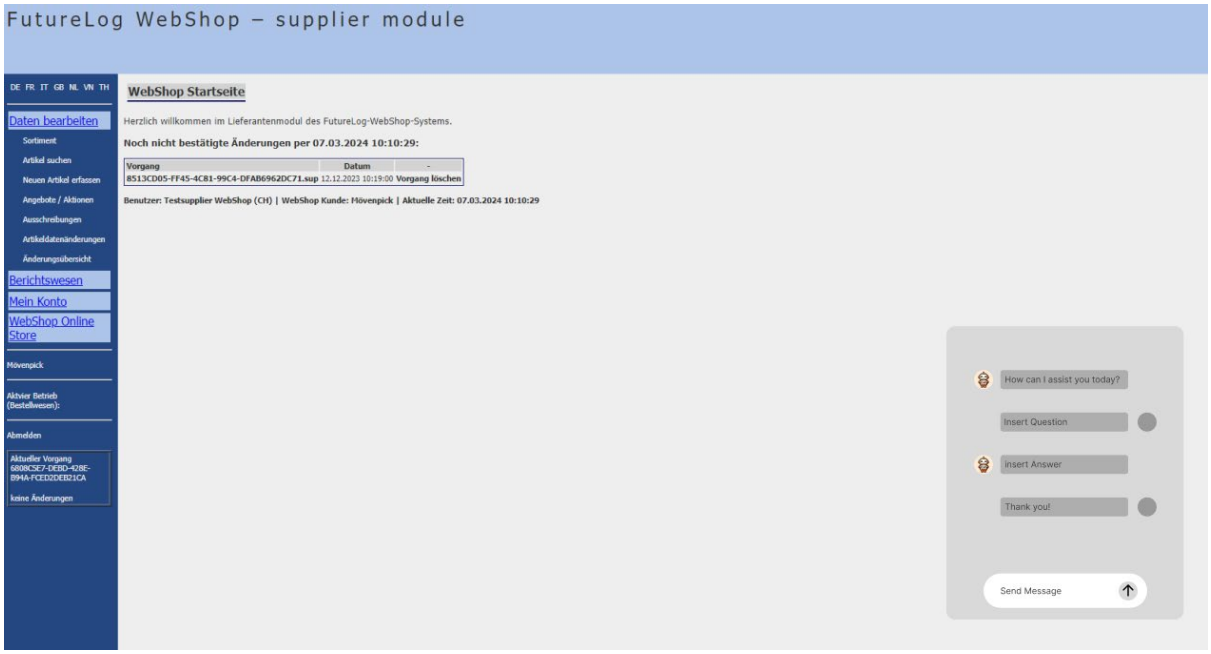

Abbildung 3 Mock Up 2

2.3.2 Programm Planung Diagramme

2.3.2.1 Pap

In diesem Pap [3] möchte ich kurz den Ablauf des Projektes aufzeigen.

Hierbei zeige ich auf, welche Schritte die Applikation tätigt, um auf die Frage des Nutzers zu antworten.

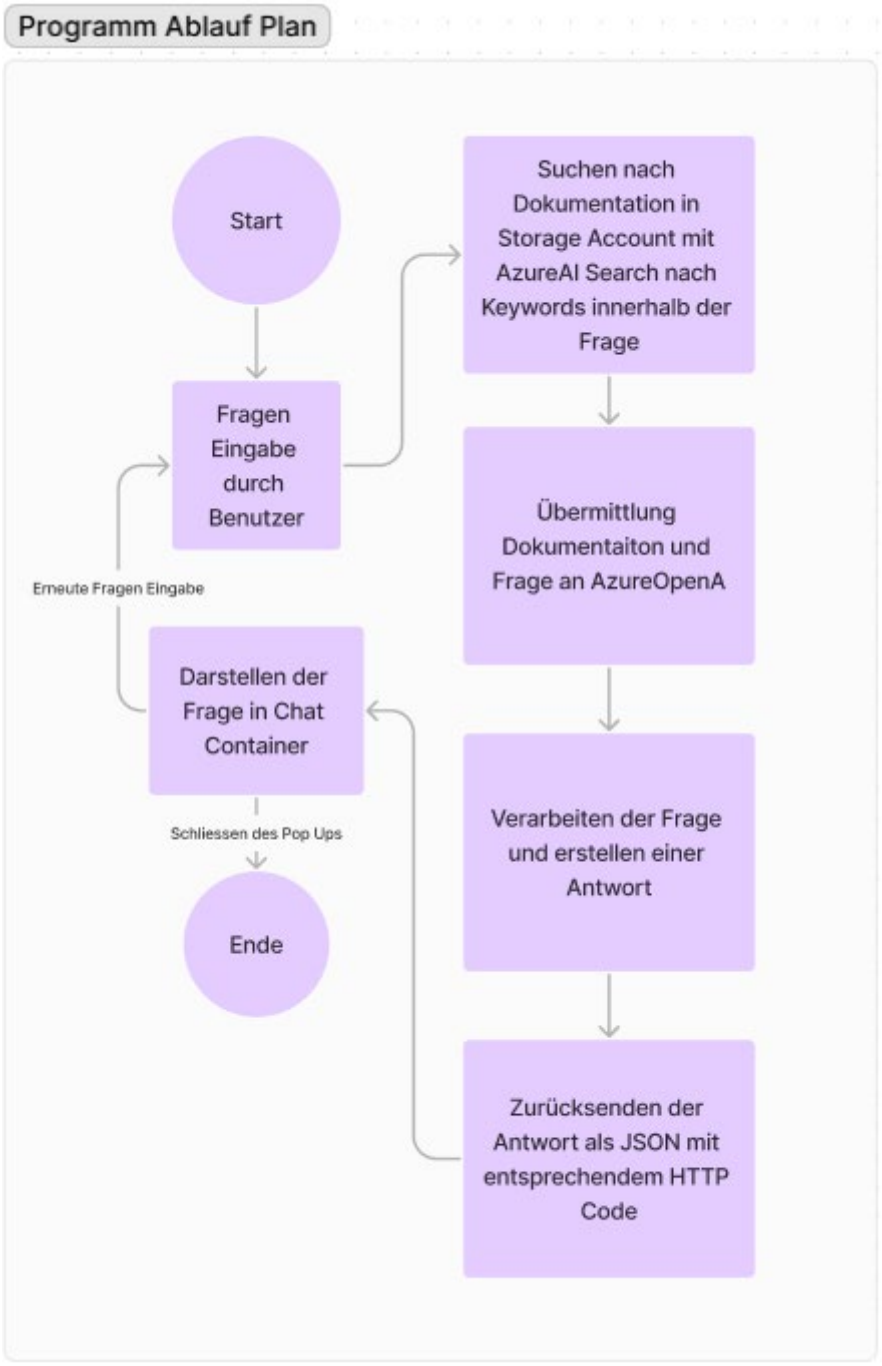

Abbildung 4 PAP

[3] Programm Ablauf Plan

2.3.2.2 Use Case

Die Aufgabe des Chatbots ist nicht sehr breit gefächert. Erhaltene Anfragen von Nutzern sollen beantwortet werden.

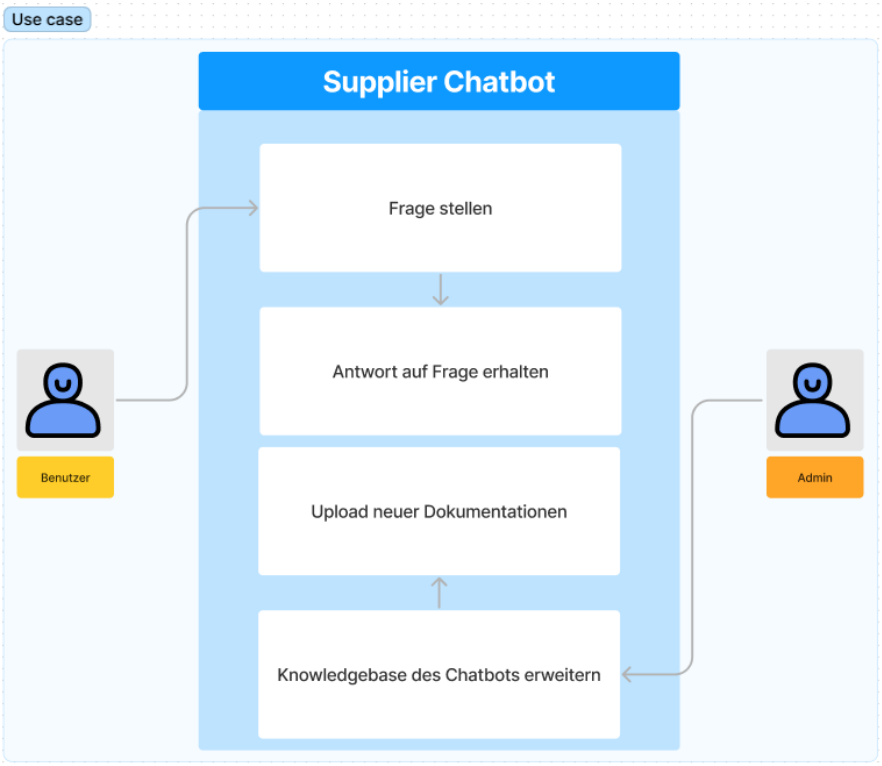

Abbildung 5 Use Case

Jedoch ist es ebenfalls wichtig, den Chatbot mittels wenig Aufwand erweitern zu können. Deshalb kann ein Admin nach Erstellung neuer Dokumentationen / Handbücher diese auf Azure hochladen. Danach kann der Bot neue Fragen beantworten bzw. neue Informationen aus den hochgeladenen Files entnehmen.

Wenn neue Funktionen implementiert werden sollen, kann dies ebenfalls schnell, dank einer aufgeräumten Codebase erledigt werden.

2.3.2.3 Azure Diagramme

2.3.2.3.1 Deployment Diagramm

Mit diesem Diagramm zeige ich auf, wie ich die App auf Azure hosten werde.

Während dem Development schreibe ich meinen Code mittels Jetbrains IDEs[4].

Für Python benutze ich Pycharm, für das Development im Frontend benutze ich Webstorm.

Während des Development teste ich meine API und das Frontend lokal mit Docker Instanzen auf meinem Computer.

Meinen Code und dessen Änderungen werden auf GitHub gespeichert.

Nach Fertigstellung der API und des Frontends erstelle ich Docker Images und pushe diese auf Azure. Diese werden dann mittels 2 Apps gehostet.

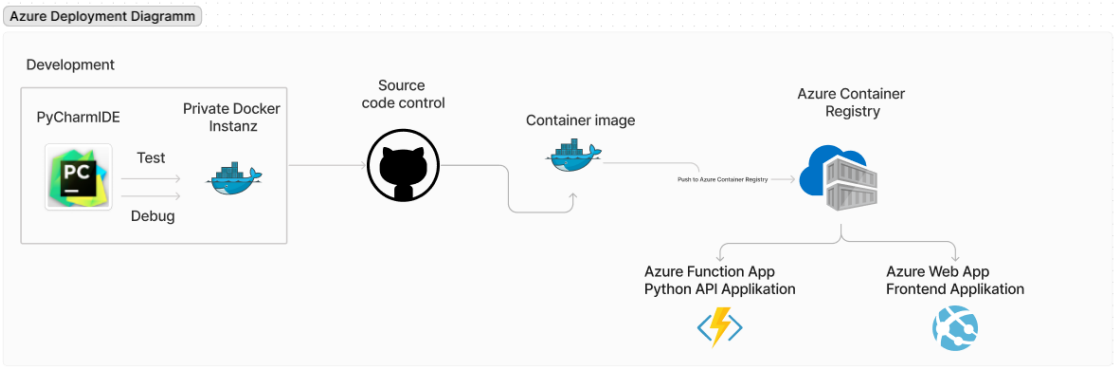

Abbildung 6 Azure Deployment Diagramm

2.3.2.3.2 Hosting und Serving

Hier zeige ich das Hosting des Projekts auf. Alles, was für das Hosting benötigt wird, sind 2 App Services, welche das Frontend und die API zur Verfügung stellen. Diese werden aus Images erstellt, welche im Container Registry gespeichert werden.

Am Ende muss das Frontend nur mit der API kommunizieren. Die API übernimmt den Rest.

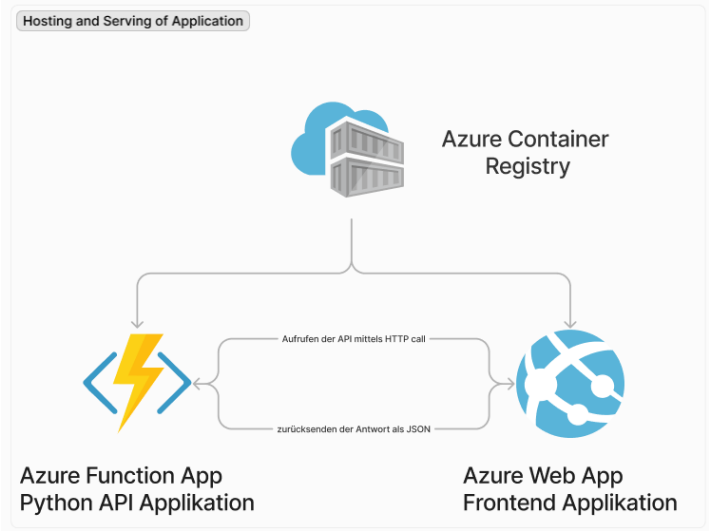

Abbildung 7 Azure Hosting und Serving

[4] Integrated Development Environment

2.3.2.3.3 Azure AI Service Diagramm

Hier zeige ich auf, wie die Azure Services für die Formulierung der Antwort zusammenarbeiten.

Hierbei übernimmt AzureOpenAI die generative Arbeit, also die Erzeugung der Antwort.

Mit AzureAiSearch werden die Dokumentationen / Handbücher nach Keywords durchsucht.

Im Storage Account werden die Dokumentationen gespeichert.

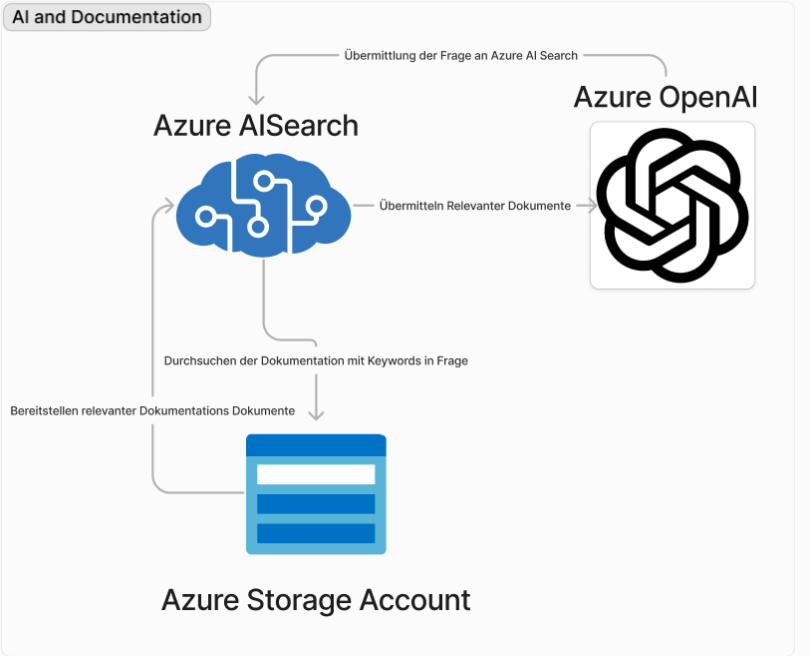

Abbildung 8 Azure AI and Documentation

2.3.3 Sicherung der Daten

Alle Daten bezüglich der API werden in einem Repository auf GitHub gespeichert.

Alle Daten bezüglich des Frontends werde ich ebenfalls in einem separaten Repository auf GitHub hosten.

Das Word File für die Dokumentation der IPA habe ich auf OneDrive, wie auch im API Git Repo gespeichert.

Alle sonstigen Dateien werden auf OneDrive im Ordner IPA gespeichert.

2.3.4 Testfallspezifikation

Testfallnummer	1
Anforderung	Integration in das Suppliermodul
Voraussetzung	Das Frontend wurde als PopUp integriert.
Eingabe	Der Chatbot wird geöffnet.
Ausgabe	Das Chatfenster des Supportbots wird in einem Pop-Up geöffnet.

Testfallnummer	2
Anforderung	Interaktivität und Antwortzeit
Voraussetzung	Die App ist geöffnet.
Eingabe	Text Eingabe: «what does ba mean? »
Ausgabe	«Ba mean Ballen which is Bale in english» Bei einer Antwortzeit unter 5 Sekunden.

Testfallnummer	3
Anforderung	Testen der Funktionalität
Voraussetzung	Die App ist geöffnet.
Eingabe	Text Eingabe: «what does ba mean? » Text Eingabe: «what does ki mean? » Text Eingabe: «how do I add a new offer? »
Ausgabe	Ausgabe 1: «Ba mean Ballen which is Bale in English». Ausgabe 2: «Ki means Kiste which is crate in English». Ausgabe 3: «To add a new offer in FutureLog, click on the product range or search for the item you want to make a promotion for. Then, define the special offer by filling in the discount price per order unit, start and end of the special offer, and shelf life (if available). Save the information with a click on Save data, and your special offer will be published automatically. »

Testfallnummer	4
Anforderung	Vollständigkeit der Dokumentation
Voraussetzung	Durchlesen der Dokumentation
Eingabe	-
Ausgabe	Leichte Verständlichkeit für den Leser

Testfallnummer	5
Anforderung	Benutzerfreundlichkeit
Voraussetzung	Die App ist gestartet.
Eingabe	Analysieren der Benutzeroberfläche
Ausgabe	Die Benutzeroberfläche ist intuitiv gestaltet und leicht verständlich.

Testfallnummer	6
Anforderung	Antwortqualität
Voraussetzung	Die App ist gestartet
Eingabe	Text: «how do i add a new offer? »
Ausgabe	Detaillierte Ausgabe wie: «To add a new offer in FutureLog, click on the product range or search for the item you want to make a promotion for. Then, define the special offer by filling in the discount price per order unit, start and end of the special offer, and shelf life (if available). Save the information with a click on Save data, and your special offer will be published automatically. »

Testfallnummer	7
Anforderung	Sicherheit und Datenschutz
Voraussetzung	Der Quellcode ist fertig geschrieben
Eingabe	Testing der einzelnen Kriterien, sofern diese für diese App zutreffen.
Ausgabe	Alle Tests sind positiv

2.4 Entscheiden

Im Folgenden Abschnitt gehe ich über einige verschiedene Varianten, bezüglich der Umsetzung des Projekts.

Variante 1:

Als Erstes kam mir die Idee das Projekt mittels OpenAI's custom GPTS zu erledigen. Hierbei konnte ich einfach die Dokumentation / Handbuch hochladen und mit benutzerdefinierten Befehlen das GPT so konfigurieren, dass es meinen Vorstellungen entsprach.

Variante 2:

Als zweite Variante hatte ich die Idee, die Daten der Dokumentation in einer Vektor Datenbank zu speichern. Daraufhin wollte ich mittels OpenAI's API ein Basic GPT Model ansprechen und dieses mit der Dokumentation befüllen. Daraufhin sollte das Basic GPT Model mithilfe der Dokumentation den Benutzer unterstützen.

Variante 3:

Als dritte Variante entschied ich mich mittels Azure und deren Dienstleistungen einen Chatbot zu erstellen. Hierbei wird die Speicherung der Dokumentation, wie auch die Formulierung einer Antwort von Azure bereitgestellt.

Kriterien	Gewichtung	Variante1		Variante 2		Variante 3	
		Bewertung	Punkte	Bewertung	Punkte	Bewertung	Punkte
Sicherheit	2	3	6	2	4	3	6
Kosten	1	2	3	3	3	1	1
UpKeep Time	1	3	3	2	2	3	3
Herausforderungen	3	1	3	2	6	3	9
Antwortzeit	2	2	4	1	2	3	6
Komplexität	2	3	6	1	2	2	4
Notizen		1*		2*		3*	
Summe			25		19		29

1*: Open AI bietet zum Zeitpunkt dieser Arbeit (12.03.2024) noch keine API-Einbindung für Custom GPTs an. Somit wäre es nicht möglich mittels eines API-Calls dieses Model direkt anzusprechen. Hierbei müsste auf Assistants gesetzt werden.

2*: Hierbei würde ich eine Pinecone[5] Vector Datenbank benutzen, während meines Testing konnte ich jedoch weder eine Verbindung zu Pinecone herstellen, noch konnte OpenAI die Daten aus der Datenbank bereitstellen. Mit diesem Ansatz hätte die Arbeitszeit nicht gereicht.

3*: Bei Azure sind Komplexität und Kosten zu beachten. Da für diese Arbeit sehr viele verschiedene Azure Dienste benutzt werden, können die Kosten sehr schnell steigen, so dass es irgendwann nicht mehr wirtschaftlich dieen Chatbot zu betreiben. Daher ist eine ausführliche Planung essenziell.

Entscheidung:

Nach sorgfältigem Überlegen entschied ich mich für Variante 3, da die Programmierung mit Azure durch eine gute Dokumentation einfach verständlich und gut umsetzbar ist. Ebenfalls sprach durch die Tatsache, dass die FutureLog AG bereits einen Grossteil ihrer Dienste in Azure hostet alles für die dritte Variante.

[5] Diensteleistungsanbieter für Vektor Datenbanken

2.5 Realisieren

2.5.1 Frontend

2.5.1.1 Overview

Für mein Frontend werde ich ein JavaScript Framework namens “Svelte” verwenden. Dieses Frontend wird HTTP Requests an meine Python API schicken und die Antworten in einem Chatfenster anzeigen.

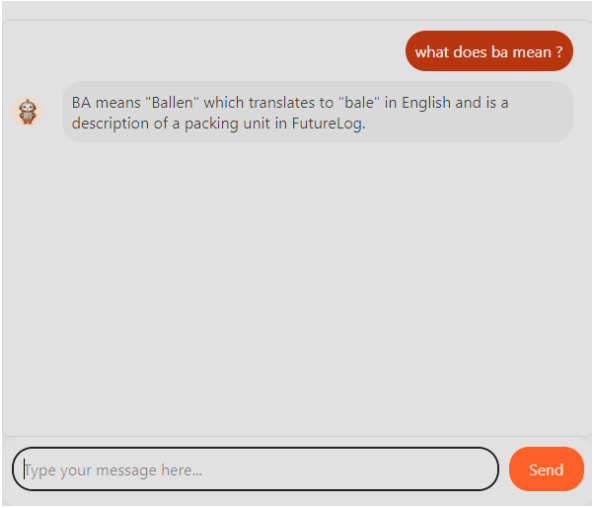

Abbildung 9 Frontend mit Frage

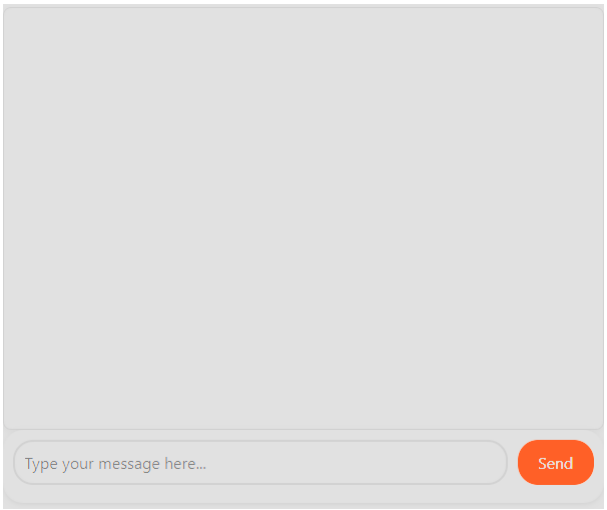

Abbildung 10 Frontend ohne Frage

2.5.1.2 Docker Deployment

In diesem Dockerfile deploye ich die Applikation in 2 Schritten.

Im 1. Schritt nehme ich ein Node[6] Image und kopiere alle Dateien den App Ordner. Ebenfalls werden alle Packages kopiert und installiert. Danach builden (erstellen) wir unsere App.

Im 2. Schritt installiere ich "serve", ein Package, mit dem ich statische Webseiten hosten kann. Danach starte ich das Frontend mittels einer Alpine Linux Dispo[7].

```
FROM node:14 as build-stage

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

FROM node:14-alpine

RUN npm install -g serve

ENV PORT=5000

COPY --from=build-stage /app/public /public

EXPOSE 5000

CMD serve -s public -l tcp://0.0.0.0:${PORT}
```

2.5.1.3 Code Beschreibung

Hier beschreibe ich alle Komponenten, welche mein Frontend braucht, um zu funktionieren.

Komponenten Name	Beschreibung	Funktionen
App.svelte	Main Page für das ChatInterface	sendMessage() class=app
ChatContainer.svelte	Komponente, welche alle Messages in einem Container anzeigt.	class=chat-container
ChatInput.svelte	Entgegennahme von Benutzereingaben mittels Textfeldes und Submit Button	onSend() class=input-area
ChatMessage.svelte	Benutzer Frage und Bot Antwort in Textblasen darstellen.	class=message-container

[6] Node: JavaScript Run Time Environment
[7] Alpine: Eine lightweight Linux Distribution

2.5.1.2.1 App

sendMessage()

Diese Funktion erstellt aus der Frage des Nutzers ein HTTP Request und sendet diesen an die Python Flask API. Dazu wird die Eingabe des Benutzers im Messages Array gespeichert.

Die Antwort wird auf Fehler geprüft und bei einem 200 HTTP-Code in das Messages Array gespeichert.

Da das Messages Array durch Svelte direkt mit dem Chat Container verbunden ist, werden Veränderungen in Real Time aktualisiert und dementsprechend dargestellt.

```
<script>
  import ChatMessage from './ChatMessage.svelte';
  import ChatInput from './ChatInput.svelte';
  import ChatContainer from './ChatContainer.svelte';

  let messages = [];

  async function sendMessage(event) {
    const text = event.detail;
    const userMessage = {text, sender: 'user'};
    messages = [...messages, userMessage];

    try {
      const apiResponse = await fetch('https://fl-supportbot-
api.azurewebsites.net/sendMessageToAzure', {
        method: 'POST',
        headers: {'Content-Type': 'application/json'},
        body: JSON.stringify({Question: text}),
      });

      if (!apiResponse.ok) {
        throw new Error(`HTTP error! Status: ${apiResponse.status}`);
      }

      const response = await apiResponse.json();
      const botReply = response.message || "I didn't understand that.";
      messages = [...messages, {text: botReply, sender: 'bot'}];
    } catch (error) {
      messages = [...messages, {text: 'Sorry, there was an error
processing your request.', sender: 'bot'}];
    }
  }

  $: if (messages.length > 0) {
    setTimeout(() => {
      const chatContainer = document.querySelector('.chat-container');
      chatContainer.scrollTop = chatContainer.scrollHeight;
    }, 0);
  }
</script>
```

app

In dieser Klasse stelle ich das Frontend zusammen. Die Nachrichten werden mit einer Liste in den Chatcontainer geladen. Dazu ordne ich hier das Input Feld unter dem Container an und füge die sendMessage Funktion dem Submit Button zu. Ebenfalls definiere ich hier die App Grösse.

```
<div class="app">
  <ChatContainer>
    {#each messages as message, index (index)}
      <ChatMessage key={index} {message}/>
    {/each}
  </ChatContainer>
  <ChatInput on:sendMessage={sendMessage}/>
</div>

<style>
  .app {
    max-width: 600px;
    margin: auto;
    margin-top: 50px;
  }
</style>
```

2.5.1.2.2 Chat Container

chat-container

Hier definiere ich den Chat Container. In diesem habe ich einen Slot definiert, welcher später von den Chat Messages benutzt wird.

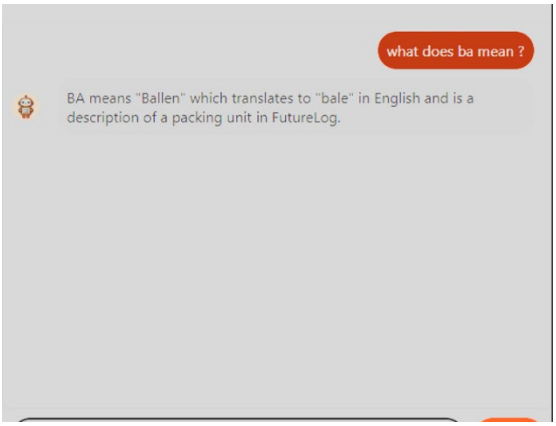

Abbildung 11 chat container

```
<div class="chat-container">
  <slot />
</div>

<style>
  .chat-container {
    border: 1px solid #ccc;
    height: 400px;
    overflow-y: auto;
    padding: 10px;
    border-radius: 8px;
    background-color: #fff;
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  }
</style>
```

2.5.1.2.3 Chat Input

onSend()

Mit dieser Funktion formatiere ich den Input Text und übergebe diesen.

Nachdem eine Nachricht gesendet wurde, wird der Input Text zurückgesetzt.

```
<script>
  import { createEventDispatcher } from 'svelte';
  const dispatch = createEventDispatcher();

  let inputText = '';

  function onSend() {
    dispatch('sendMessage', inputText.trim());
    inputText = '';
  }
</script>
```

input-area
Hier wird die Eingabefläche des Nutzers definiert.

Abbildung 12 Input Field

Texteingaben werde mit einem Textinput entgegengenommen. Eine Nachricht kann der Nutzer entweder durch die Enter Taste oder mit dem Button absenden.

Wenn der Nutzer keine Eingabe gemacht hat, wird ein Platzhalter angezeigt.

```
<div class="input-area">
    <input
      type="text"
      bind:value={inputText}
      on:keyup={(e) => e.key === 'Enter' && onSend()}
      placeholder="Type your message here..."
    />
    <button on:click={onSend}>Send</button>
</div>

<style>
  .input-area {
    display: flex;
    justify-content: space-between;
    padding: 10px;
    background-color: #fff;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);
    border-radius: 20px;
  }
  input[type="text"] {
    flex-grow: 1;
    padding: 10px;
    margin-right: 10px;
    border: 2px solid #ccc;
    border-radius: 20px;
  }
  button {
    padding: 10px 20px;
    border-radius: 20px;
    border: none;
    background-color: #ff5722;
    color: white;
    cursor: pointer;
  }
  button:hover {
    background-color: #e64a19;
  }
</style>
```

2.5.1.2.4 Chat Message

message-container

Hier werden die Textblasen definiert.

Wenn es sich um eine Antwort des Bots handelt, wird diese in einem Grauton mit einem kleinen Avatar angezeigt.

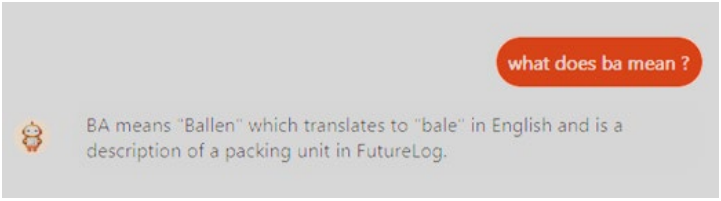

Abbildung 13 chat message

Bei einer Frage des Nutzers wird eine nur eine orangene Textblase angezeigt.

Ebenfalls werden die Textblasen, wie von bekannten Chats (WhatsApp, Teams etc.) rechts und links angeordnet.

```
<script>
  export let message;
</script>

<div class="message-container {message.sender}">
  {#if message.sender === 'bot'}
    <div class="avatar"></div>
  {/if}
  <div class="message-text">{message.text}</div>
</div>
<style>
  .message-container.bot {
    flex-direction: row;
  }
  .message-container.user {
    flex-direction: row-reverse;
  }
  .message-container.user .message-text {
    background-color: #a8300c;
    color: white;
  }
  .avatar {
    min-width: 30px;
    height: 30px;
    border-radius: 50%;
    background-image: url('/bot_avatar.png');
    background-size: cover;
    background-position: center;
    margin-right: 10px;
  }
</style>
```

2.5.1.4 Einbindung zur API

HTTP Request to API

Mit diesem Stück Code wird die Python API aufgerufen. Hierbei wird der Content als JSON definiert.

Danach wird die Frage des Benutzers im Feld Question an die API geschickt.

Am Ende wird geprüft, ob der Request erfolgreich war.

```
try {
  const apiResponse = await fetch('https://fl-supportbot-
api.azurewebsites.net/sendMessageToAzure', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({Question: text}),
  });

  if (!apiResponse.ok) {
    throw new Error(`HTTP error! Status: ${apiResponse.status}`);
  }
```

2.5.2 Python API

2.5.2.1 Übersicht

Das Ziel meiner API ist es, eine Schnittstelle zwischen dem Frontend und dem Azure-Backend bereitzustellen.

Hierbei sollen beide Seiten unabhängig voneinander funktionieren. Die einzige Kommunikation zwischen dem AI-Teil und dem Frontend soll die Python API sein.

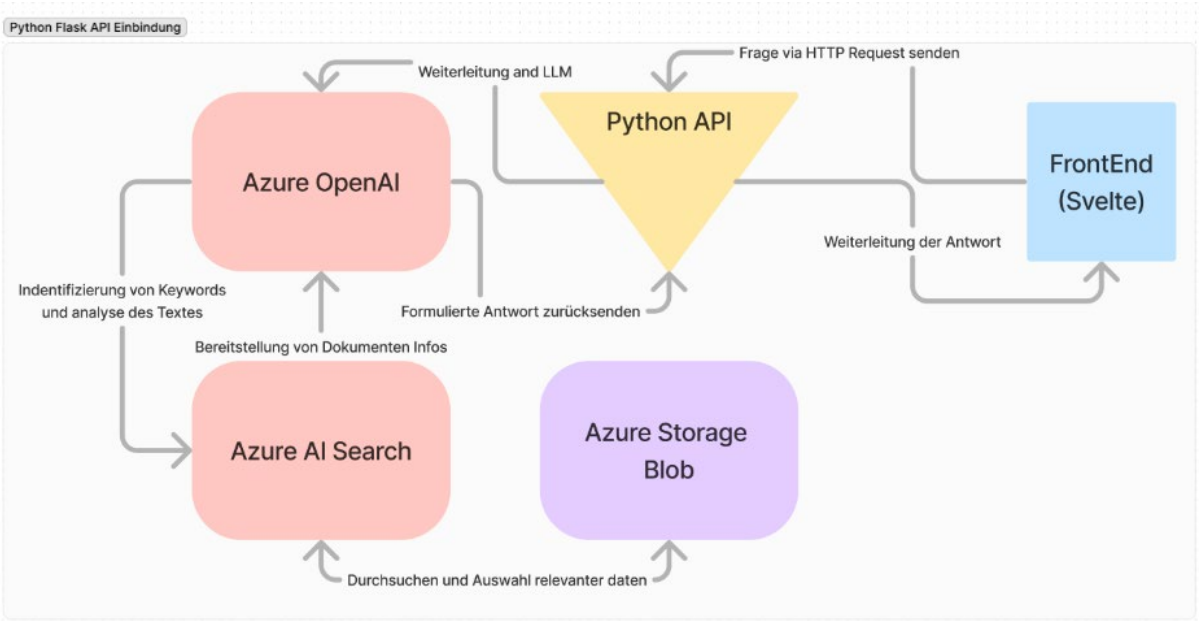

Abbildung 14 Python API Einbindung

2.5.2.1.1 Ordner und Datei Struktur

```
.Support_Chatbot
├── ManageGPTRquest *enthält alle Project Dateien
│   ├──.env *Speicherung aller Sensiblen Schlüssel
│   ├──app.py *API Programm Datei
│   ├──Dockerfile *Build Informationen für Deployment
│   └──requirements.txt *alle packages mit entsprechender Version
├── venv *Virutelle Programmier umgebung für Python
├── .gitignore *enthält alle Dateien, welche nicht ins Repository
sollen
├── API_TEST.http *Test http request
└── README.md *kurze Projekt Beschreibung und Keypoints
```

2.5.2.1.2 Datei Formatierung und Standard

Alle meine Dateien folgen, den Best Practice Regeln der jeweiligen Programmiersprache, in diesem Fall Python und Docker.

2.5.2.2 ENV-File

Während der Entwicklung habe ich alle sensiblen Daten, wie interne Endpoints oder auch API Keys innerhalb eines Environment Files gespeichert.

Dieses wird nicht auf GitHub gepusht. Für das Deployment speichere ich diese Daten mit Azure ebenfalls als Umgebungs Variablen.

2.5.2.3 Code Description

Imports

```python
import os
from dotenv import load_dotenv
from flask import Flask, request, render_template, jsonify
from flask_cors import CORS
from langchain_community.chat_models import AzureChatOpenAI
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from azure.search.documents import SearchClient
from azure.core.credentials import AzureKeyCredential
```

Erstellen der Flask app

Hier erstelle ich die Flask app und aktivere CORS[8].

```python
app = Flask(__name__)
CORS(app)
```

[8] CORS steht für "Cross-Origin Ressource Sharing"

Route und Variablen Definition

Definition der Route und Art von Requests, welche diese API annehmen kann. Ebenfalls werden die Umgebungsvariablen geladen, wie auch Variablen angelegt.

Zum Schluss wird noch die Frage des Nutzers aus einem JSON extrahiert.

```
@app.route('/sendMessageToAzure', methods=['POST'])
def sendAndReceiveMessage():
    question = ''
    response_content = ''
    load_dotenv()

    if request.method == 'POST':
        data = request.get_json()
        question = data["Question"]
```

AzureChatOpenAI Model erstellen

In der Model Variablen definiere ich das Model, welches ich in Azure erstellt habe.

Alle sensiblen Daten werden in Umgebungsvariablen gespeichert.

```
model = AzureChatOpenAI(
    openai_api_type="azure",
    azure_endpoint=os.getenv("AZURE_OPENAI_ENDPOINT"),
    openai_api_version='2023-03-15-preview',
    deployment_name=os.getenv("AZURE_OPENAI_DEPLOYMENT_NAME"),
    api_key=os.getenv("AZURE_OPENAI_KEY")
)
```

AzureAISearch Service erstellen

Als Search Client wird hier der Azure AI Search Service definiert, den ich ebenfalls in Azure erstellt habe.

```
search_service_name = os.getenv("SEARCH_SERVICE_NAME")
search_index_name = os.getenv("SEARCH_INDEX_NAME")
search_api_key = os.getenv("SEARCH_API_KEY")

search_client = SearchClient(

endpoint=f"https://{search_service_name}.search.windows.net/",
    index_name=search_index_name,
    credential=AzureKeyCredential(search_api_key)
)
```

search_database()

Die obenstehende Funktion, verwendet. Nachdem die Resultate aus der Datenbank (Azure Storage Account) zurückkommen, werden sie in einem Array gespeichert und zurückgegeben.

```
def search_database(query):
    results = search_client.search(search_text=query)
    return [result for result in results]
```

format_search_results()

Überprüfung der Resultate. Wenn keine Resultate gefunden wurden, wird eine entsprechende Meldung zurückgegeben.

Die Top Drei Resultate werden formatiert und zurückgegeben.

```
def format_search_results(results):
    if not results:
        return "I couldn't find any information on that topic."
    formatted_results = "Here's what I found: "
    for result in results[:3]:
        formatted_results += f"\n- {result['content']}"
    return formatted_results
```

azure_search_component()

Mit dieser Komponente wird die Datenbank abgefragt und die formatierten Resultate werden zurückgegeben.

```
def azure_search_component(query):
    return format_search_results(search_database(query))
```

Durchführung der AzureOpenAI Abfrage

Mit diesem Code wird der Chatbot mit entsprechenden Befehlen aufgerufen, ihm wird die Frage und Dokumentation übergeben. Am Ende wird die Antwort des Bots an das Frontend retourniert.

```
        prompt_answer = f"""You are a helpful assistant that can
answer questions related to the webapp FutureLog.
        Answer the question based on the given documentation
below. Your answer must be concise. You can also answer without
the Documentation if the Information isn't found in your files.
Please just tell the user so.

        Documentation: {{search_results}}

        Question: {{question}}
        Answer: """

        chain_answer = LLMChain(llm=model,
prompt=PromptTemplate.from_template(prompt_answer))
        answer = chain_answer.run({
            'search_results': azure_search_component(question),
            'question': question
        })

        return jsonify({"message": answer}), 200
    else:
        return jsonify({"message": "bad"}), 401
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

2.5.2.4 Requirements

Alle Packages, welche ich benutze, werden in einem “Requirements.txt” gespeichert. Dieses speichert alle Package Versionen und ermöglicht somit Konsistenz während der Entwicklung, wie auch nach dem Deployment.

```
Flask==3.0.2
openai==1.13.3
Werkzeug==3.0.1
python-dotenv==1.0.1
azure-core==1.30.1
azure-ai-ml==1.13.0
azure-search-documents==11.6.0b2
langchain-community==0.0.28
openai==1.13.3
Flask-Cors==4.0.0
langchain==0.1.12
```

2.5.2.4 Docker Deployment

```
# Use an official Python runtime as the parent image
FROM python:3.12.2

# Set the working directory in the container to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Run app.py when the container launches
CMD ["flask", "run", "--host=0.0.0.0"]
```

Die Python API wird mittels Docker deployed, hierbei baue ich mit einem Dockerfile zuerst ein Image. Dieses Image beinhaltet alle Befehle, welche benötigt werden, um die App nachher zu bauen und zu starten.

Hierbei nehme ich zuerst eine normale Python 3.12.2 Version. Danach werden alle Files kopiert und alle Bibliotheken installiert.

Wenn das File erstellt wurde, konnte ich es in mein Azure Container Registry hochladen und von dort aus in einer “Function app” hosten.

2.5.2.5 API-Informationen

2.5.2.5.1

Name	Parameter	Endpoint	Beschreibung
/sendMessageToAzure	Question (String)	https://fl-supportbot-api.azurewebsites.net	Schickt die Frage eines Kunden an Azure und sendet eine Antwort zurück

2.5.2.5.2 HTTP Request

```
POST /sendMessageToAzure HTTP/1.1
Host: localhost:5000
Content-Type: application/json
Content-Length: 120

{
    "Question": "you question here"
}
```

2.5.2.5.3 JSON-Antwort

bei Statuscode 200

```
{
    "message": "message_inhalt"
}
```

oder bei Statuscode 401

```
{
    "message": "bad"
}
```

2.5.3 Azure

Um meine Arbeit mit Azure zu starten, habe ich zuerst eine neue Ressourcengruppe erstellen lassen. In dieser Ressourcengruppe werde ich alle Azure Dienste, welche ich für dieses Projekt benötige, registrieren.

2.5.3.1 Azure Dienste

Azure Service Name	Beschreibung
Azure OpenAI	Benutzen eines LLM, in diesem Fall bereitgestellt von OpenAI. Analysieren der Frage und Erstellen einer Antwort.
Azure Function App	In der Function App hoste ich meine Python API als Docker Image. Mit dieser Function app entstehen nur Kosten, wenn diese Function ausgeführt wird. Daher entstehen keine laufenden Kosten, nur im Falle einer Verwendung.
Azure Web App	In der Azure Web App wird die Svelte Frontend Applikation gehostet.
Azure Container Registry	Im Container Registry werden alle Docker Images gespeichert, welche für dieses Projekt notwendig sind.
Azure AI Search	Der Azure AI Search Dienst druchsucht mit Keywords in der Frage das Document Storage nach relevanten Dokumentationen.
Azure Storage Account *	Im Azure Storage Account werden sämtliche Informationen, Dokumentation und Handbücher des Suppliermoduls gespeichert.

2.5.3.1.1 Azure OpenAI

Erstellung eines Bot Models

Im Azure OpenAI Studio habe ich ein GPT 3.5 Model erstellt.

Dieses werde ich nachher benötigen, um den Benutzerinput zu verstehen und zu verarbeiten, sowie eine Antwort zu formulieren und zurückzusenden.

Dieser Bot könnte jedoch ebenfalls direkt zu einer Web App deployed und so, als eigene Webseite verwendet werden.

Hier bin ich im Einzelnen, wie folgt vorgegangen.

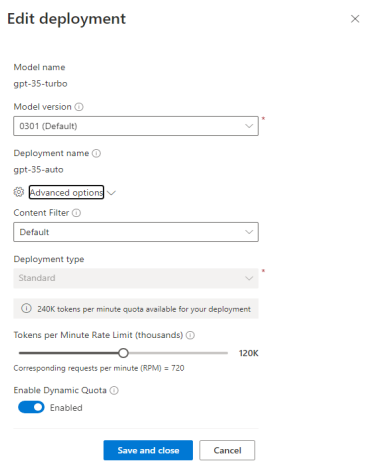

Abbildung 16 Erstellen eines AzureOpenAI Bots

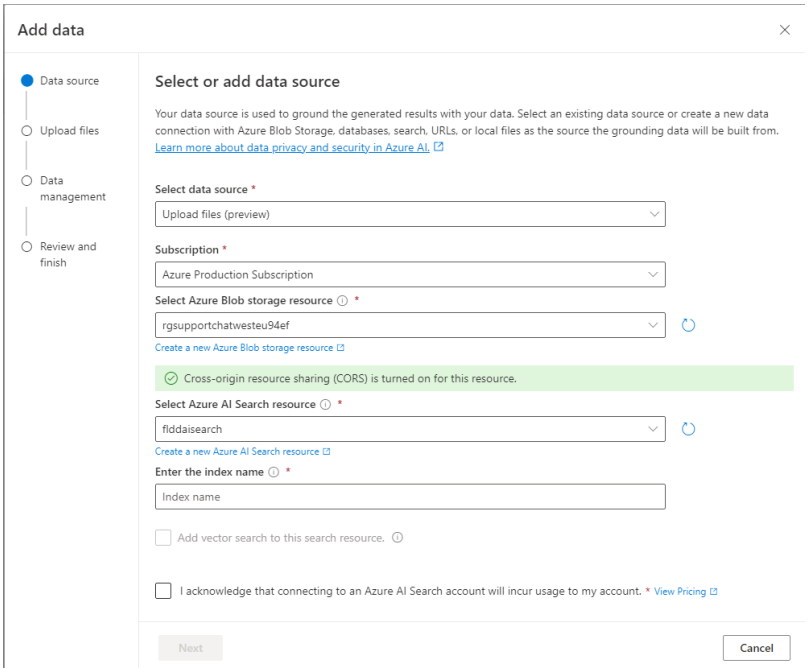

Abbildung 15 Verknüpfen von Dokumentation und Bot

Ich habe mit einem UI den Azure Search AI und Storage Dienst verbunden. Der Search Dienst benutzt die Dokumentationen / Handbücher, welche im Storage gespeichert werden.

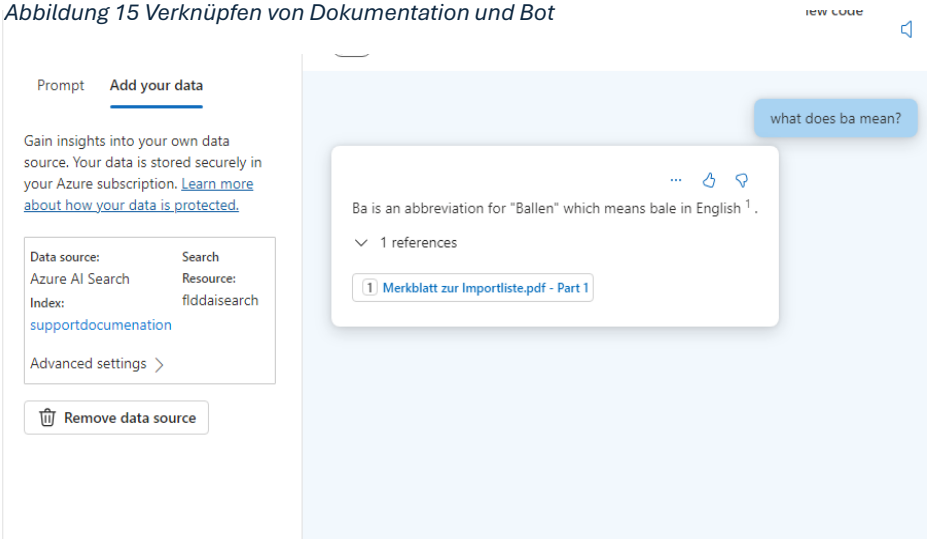

Abbildung 17 1. Test des Bots

Dann habe ich den Bot getestet. Nach ein paar Sekunden fand der Bot die Antwort in der Dokumentation und konnte so die Frage beantworten.

2.5.3.1.2 Azure Function App

Hier habe ich ein Screenshot der erstellten Function App.

Als Option habe ich ausgewählt, dass es sich bei dieser Applikation um eine Container App handelt

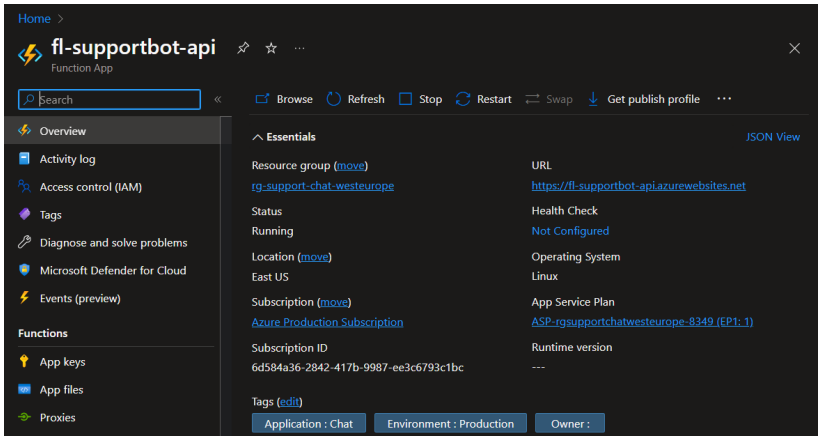

Abbildung 18 Azure Function App Einstellungen

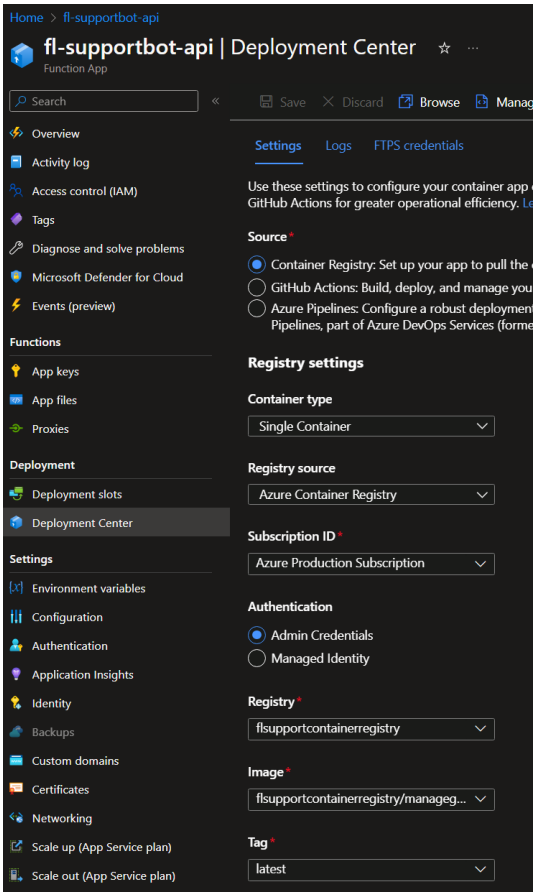

Abbildung 19 Azure Function App Deployment

Im Deployment verweise ich hier auf das API Docker Image, welches ich vorher ins Azure Container Registry gepushed habe.

2.5.3.1.3 Azure Webapp

Dann habe ich das Frontend als Azure Web App gehostet. Diese wird 24/7 online und erreichbar sein.

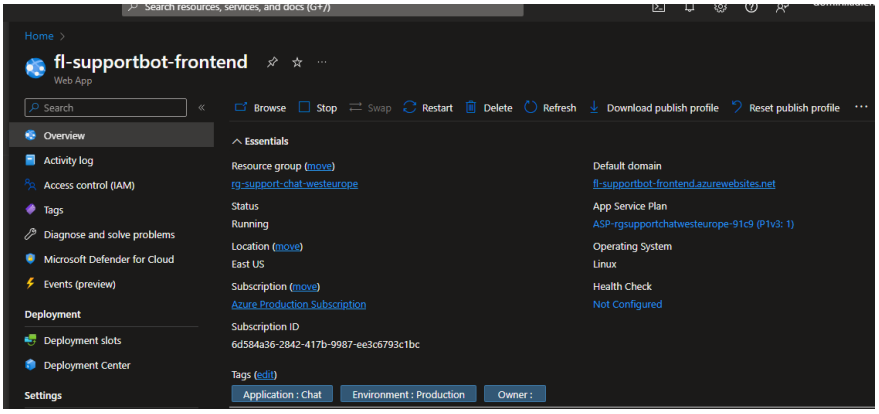

Abbildung 20 Azure Web App Einstellung

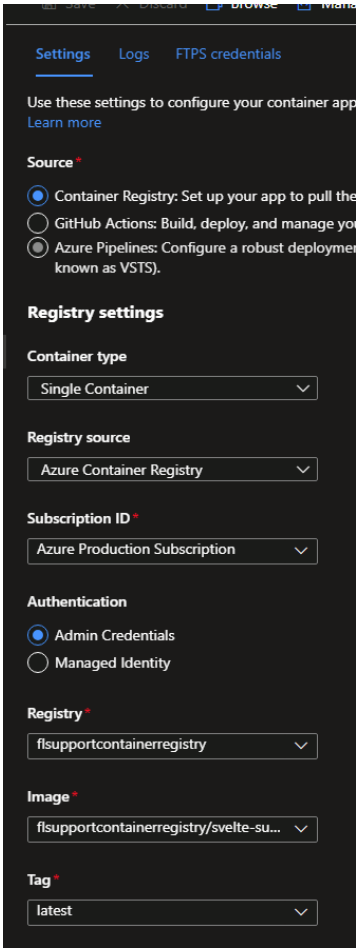

Abbildung 21 Azure Web App Deployment

Wie auch bei der Azure Function App definiere ich hier das Image, welches die Webapp benutzen soll, um

2.5.3.1.4 Azure Container Registry

Erstellung der Azure Container Registry.

In diesem Registry befinden sie meine 2 Images (die obersten 2), wie auch die meines Mitpraktikanten.

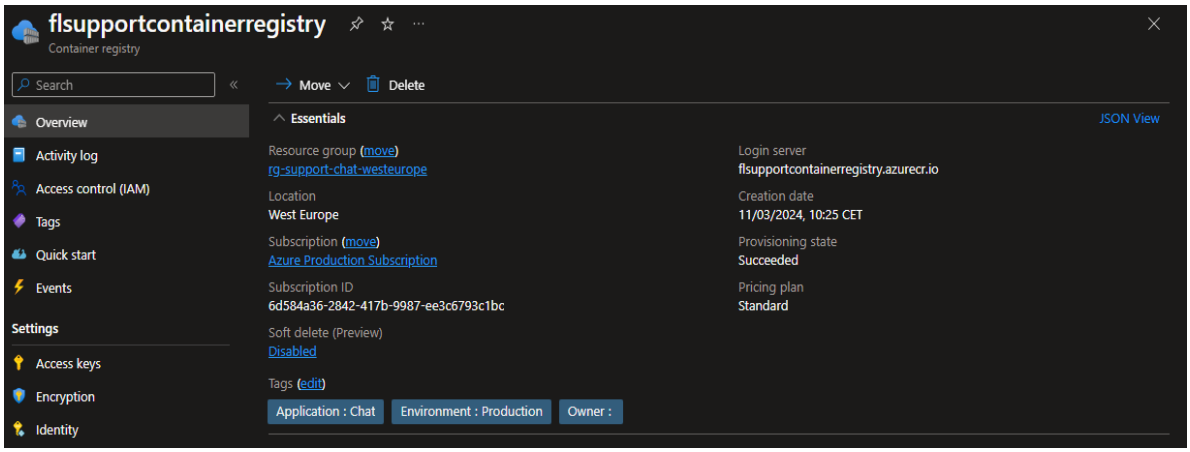

Abbildung 22 Azure Container Registry Einstellungen

Die Images werden mit verschiedenen Tags gespeichert. Diese Tags können nun verwendet werden um eine Produktions-, wie auch eine Testumgebung zu automatisieren.

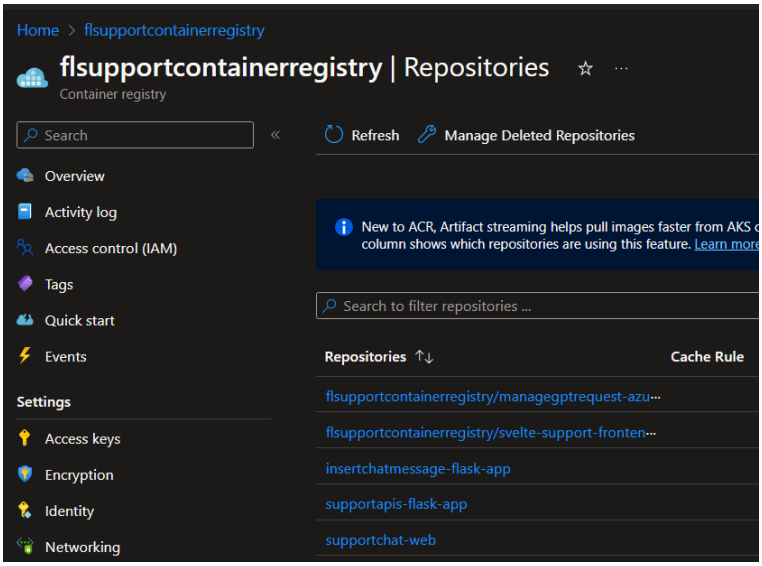

Abbildung 23 Container Registry Repositories

2.5.3.1.5 Azure AI Search Service

Als letzten Azure Dienst wird, hier der Azure AI Search Service definiert.

Mit diesem wird das Storage mit dem Index «supportdocumentation» durchsucht.

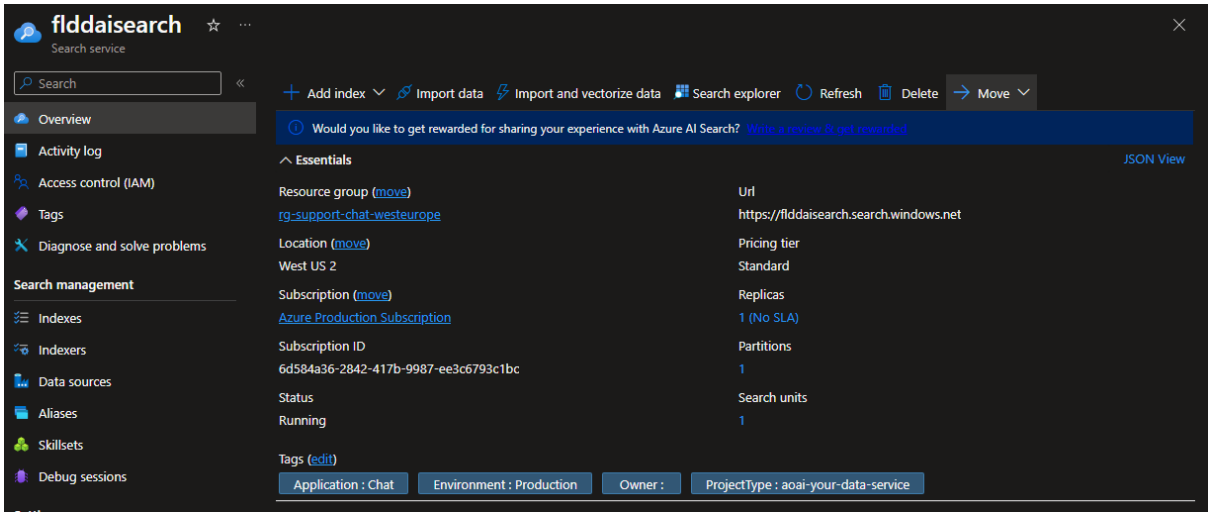

Abbildung 24 Azure AI Search Einstellungen

In diesem Index wird die gesamte Supplier Dokumentation gespeichert.

Diese PDFs werden in einzelne Artikel unterteilt, welche anschliessen als JSON gespeichert werden.

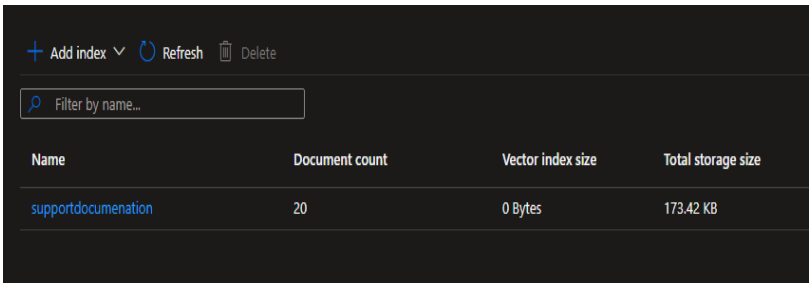

Abbildung 25 Azure AI Search Index

2.5.4 Einbindung in Suppliermodul

2.5.4.1 Navigation.asp

In der seitlichen Nav Bar wird hier ein Bild definiert, welches nach einem Klick das oben beschriebene PopUp öffnet.

```
Navigation.asp

    <%
    If Session("UserID")<>"" Then
       Response.Write "<HR>"
       Response.Write "<p><a id='titlelink' href=#
onclick=window.open(" & chr(34) & "GetSupport.asp?SupplierID=" &
Session("UserID") & "&SessionID=" & Session("SID") & chr(34) & "," &
chr(34) & "Chat" & chr(34) & "," & chr(34) &
"width=600,height=600,top=100,toolbar=0,status=0,location=0,menubar=0,s
crollbars=1,resizable=0" & chr(34) & "><img border=0 height=80
title='Get Support' src=" & BildPfad("system/fl_support_buddy.png") &
"></a></p>"
    End If
    %>

GetSupport.asp
```

2.5.4.1 GetSupport.asp

Mit dem 2. “Response.Write”, rufe ich das Chatbot Frontend in einem Pop-Up auf.

Mit diesem Skript kann der Nutzer entscheiden, ob er mit einem Mitarbeiter schreiben, oder ob er den Chatbot verwenden will.

Wenn der Nutzer sich entschieden hat, wird das Pop-Up mit dem entsprechenden Inhalt angezeigt.

Wichtig anzumerken ist, dass die App nicht als Iframe implementiert werden konnte, da die Server Einstellungen des Suppliermoduls, keine Links auf externen Domains in iframes zulassen.

```
GetSupport.asp

<%@ Language=VBScript %>
<!-- #include virtual="/webshoplibrary/funclibwebshop.asp" -->
<!-- #include virtual="/webshoplibrary/funclibwebshopinit.asp" -->
<%
UserCheck
AuthorityCheck "S",Session("UserLevel")

Set db=CreateObject("ADODB.Connection")


db.Open DBPfad()

KopfWS "Please choose how we can help you"

Response.Write "<p><a class=link
href=https://flahcchatfrontend.azurewebsites.net/supportchat?UserID" &
Session("UserID") & "&SessionID=" & Session("SID") & ">Chat with our
Support Team</a></p>"

Response.Write "<p><a class=link href=https://fl-supportbot-
frontend.azurewebsites.net/>Chat with our Chat Bot</a></p>"

FussPopUp

db.Close
%>
```

2.6 Kontrollieren

2.6.1 Erfüllung der Aufgabenstellung überprüfen

Ich konnte alle Anforderungen erfüllen.

Der Bot verhält sich bis jetzt noch ein wenig starr, jedoch kann dies mit weiteren Dokumentationen / Handbüchern, wie auch dem Feintuning der Befehle behoben werden.

Jedoch kann ich dies nun in dieser kurzen Zeit, welche mir bis zur Abgabe noch bleibt, leider nicht mehr verbessern.

2.6.2 Erfüllung der Bewertungskriterien überprüfen

Die meisten Bewertungskriterien wurden nach meiner Einschätzung erfüllt.

Bewertungskriterium Nr.	Wer?	Datum	Resultat	Bemerkung
B1	DD	14.03.2024	Okay	-
B2	DD	14.03.2024	Okay	-
B3	DD	14.03.2024	Okay	-
B4	DD	14.03.2024	Okay	-
B5	DD	14.03.2024	Okay	-
B6	DD	14.03.2024	Okay	-
B7	DD	14.03.2024	Okay	-
B8	DD	14.03.2024	Okay	-
B9	DD	14.03.2024	Okay	-
B10	DD	14.03.2024	Okay	-

2.6.3 Testprotokoll

Testnummer	Wer?	Datum	Resultat	Bemerkung
Anforderung 1	DD	14.03.2024	Okay	-
Anforderung 2	DD	14.03.2024	Okay	-
Anforderung 3	DD	14.03.2024	Okay	-
Anforderung 4	DD	15.03.2024	Okay	-
Anforderung 5	DD	15.03.2024	Okay	-
Anforderung 6	DD	15.03.2024	Okay	-
Anforderung 7	DD	15.03.2024	Okay	-

2.6.4 Testbericht Fazit

Ich konnte alle gewünschten Features implementieren. Dazu wird es einfach sein, in Zukunft weiter an diesem Bot zu arbeiten. Neue Dokumentationen können hochgeladen werden, und für Konsistenz könnten Nutzer Eingaben, wie auch Ausgaben in verschiedene Sprachen übersetzt werden.

2.7 Auswertung

2.7.1 Reflexion über die Arbeit

Diese Arbeit war für mich eine hervorragende Gelegenheit, mein Wissen in den Bereichen Flask und Azure nicht nur zu vertiefen, sondern auch mein Spektrum an Fähigkeiten zu erweitern und vorhandene Kompetenzen auszubauen. Zunächst hatte ich Bedenken, dass die Projektzeitplanung vielleicht zu ambitioniert angesetzt war, doch diese Sorgen waren unbegründet. Die Aufgaben ließen sich gut innerhalb der vorgesehenen 10 Arbeitstage bewältigen, was meine Fähigkeiten in Zeitmanagement und Prioritätensetzung signifikant verbesserte.

Die Unterstützung durch das Team von New Ocean war in diesem Prozess von unschätzbarem Wert. Ihre prompten und klaren Rückmeldungen ermöglichten es mir, aufkommende Fragen zügig zu klären und Hindernisse effizient zu bewältigen. Dies unterstreicht erneut, wie wichtig eine gute Kommunikation und Kooperation in Projektumgebungen ist.

Ein wesentlicher Teil meiner Arbeitszeit war der Dokumentation gewidmet. Was anfangs als Herausforderung erschien, entpuppte sich als äußerst wertvoll. Die intensive Beschäftigung mit der Dokumentation verfeinerte meine Fähigkeiten in der klaren und strukturierten Kommunikation. Sie half mir dabei, komplexe technische Sachverhalte verständlich aufzubereiten – ein unerlässlicher Aspekt für zukünftige Projekte und die Zusammenarbeit mit verschiedenen Interessengruppen.

Durch diese Erfahrung wurde mir bewusst, wie entscheidend es ist, über den reinen Programmier- und Entwicklungsaspekt hinauszublicken. Projektmanagement, effektive Kommunikation und Teamarbeit sind ebenso wichtige Bestandteile für den Erfolg eines Projekts. Diese Erkenntnisse sind für meine zukünftige berufliche Laufbahn von großer Bedeutung, da sie es mir ermöglichen, Projekte ganzheitlich zu betrachten und effizienter zu gestalten.

Die intensive Auseinandersetzung mit Flask und Azure hat zudem meine technische Kompetenz erheblich erweitert. Durch das praktische Anwenden dieser Technologien konnte ich ein tiefgreifendes Verständnis für ihre Funktionsweisen und Anwendungsbereiche entwickeln. Diese Kenntnisse sind für meine zukünftige Arbeit in der Softwareentwicklung und im Cloud Computing von unschätzbarem Wert.

Insgesamt hat mich diese Arbeit sowohl beruflich als auch persönlich bereichert. Sie hat verdeutlicht, dass kontinuierliches Lernen und Anpassungsfähigkeit in der dynamischen Welt der Technologie Schlüssel zum Erfolg sind. Ich freue mich darauf, diese Erkenntnisse in zukünftigen Projekten anzuwenden und sowohl meine technischen Fertigkeiten als auch meine sozialen Kompetenzen kontinuierlich weiterzuentwickeln.

2.7.2 Schlusswort

Zum Abschluss dieses Projekts blicke ich mit einem gewissen Stolz auf das Erreichte zurück. Es war eine Zeit, die nicht nur durch intensives Lernen geprägt war, sondern auch durch erhebliches Wachstum im Bereich des Projektmanagements. Trotz einiger Herausforderungen, vor allem bei der umfangreichen Dokumentation, habe ich bedeutende Erfahrungen sammeln können. Die Länge der Dokumentation, die zeitweise fast überwältigend war, hat die Bedeutung der detaillierten Aufzeichnung jeder Projektphase unterstrichen. Diese Genauigkeit ermöglichte es mir, ein tieferes Verständnis für die verschiedenen Aspekte des Projekts zu entwickeln und wird bei der Planung und Umsetzung zukünftiger Projekte von großem Nutzen sein.

Ich habe erkannt, dass der Schlüssel zum Erfolg eines Projekts nicht allein in der technischen Umsetzung liegt, sondern auch in der Fähigkeit, diese technischen Details klar und verständlich zu dokumentieren. Diese Fertigkeit ist insbesondere wertvoll, wenn es darum geht, Wissen an Teammitglieder zu vermitteln oder als Grundlage für nachfolgende Projekte zu dienen.

Zudem habe ich die Wichtigkeit von Anpassungsfähigkeit und Kreativität bei der Problemlösung erkannt. Kein Projekt verläuft exakt nach Plan, und die Fähigkeit, flexibel auf veränderte Umstände zu reagieren und kreative Lösungen für unvorhergesehene Herausforderungen zu finden, ist essenziell.

Des Weiteren habe ich die Wichtigkeit von Zielorientierung und Prioritätensetzung erkannt. Bei einem Projekt dieser Größenordnung ist es leicht, sich in Details zu verlieren. Deshalb war es entscheidend, immer das große Ganze im Blick zu behalten und sich auf die Hauptziele zu konzentrieren. Diese Fokussierung hat mir geholfen, effizienter zu arbeiten und letztendlich ein erfolgreiches Ergebnis zu erzielen.

Die Erfahrung, die ich durch dieses Projekt gewonnen habe, hat mich nicht nur beruflich, sondern auch persönlich wachsen lassen. Ich habe gelernt, Verantwortung zu übernehmen, flexibel zu bleiben und mich stets neuen Herausforderungen zu stellen. Dieses Wachstum ist für mich von unschätzbarem Wert und ich bin zuversichtlich, dass ich diese Erkenntnisse in zukünftigen Projekten gewinnbringend einsetzen kann.

Abschließend möchte ich sagen, dass dieses Projekt eine bedeutende Etappe in meiner beruflichen Laufbahn darstellt. Die dabei gewonnenen Kenntnisse und Fähigkeiten sind ein solides Fundament für meine zukünftige Entwicklung und ich bin gespannt, wo mich dieser Weg noch hinführen wird. Die Erfahrungen aus diesem Projekt werden mir als Leitfaden dienen, um auch zukünftige Herausforderungen erfolgreich zu meistern und meine beruflichen Ziele zu erreichen

2.8 Quellenverzeichnis

Microsoft. (19. 10 2023). *azure-sdk-overview*. Von learn.microsoft.com: https://learn.microsoft.com/en-us/azure/developer/python/sdk/azure-sdk-overview abgerufen

OWASP. (2021). *OWASP Top Ten*. Von Owasp: https://owasp.org/www-project-top-ten/ abgerufen

2.8.1 Personen

- FutureLog IT Team Switzerland
  - Frank Dierberger (Berufsbildner & Verwantwortliche Fachkraft)
- FutureLog IT Team DACH (unterstützung bei Fragen zu Azure)
  - Andriy Sierov
  - Roman Burdiuzha
- New Ocean Vietnam (unterstützung bei Fragen zu Python API)
  - Bao Ho
  - Anh Tran
  - Shawn Dao
- PkORG
  - Adrian Woerz (Hauptexperte)
  - Estefania Otero (Nebenexpertin)

2.8.2 Dokumente

- Kriterienkatalog Standardkriterien Dierberger Dominik Frank.pdf
- Aufgabenstellung Dierberger Dominik Frank.pdf

2.8.3 Abbildungsverzeichnis

2.8.4 Stichwortverzeichnis

2.9 Anhang

2.9.1 Benutzerhandbuch

DominikDierberger-IPA-Benutzerhandbuch.docx

2.9.2 Technische Dokumentation (readme.md)

readme.md Dateien innerhalb beider .zip ordner

2.9.3 Python Quell Code

DominikDierberger_IPA_Support_Chatbot.zip

2.9.4 Svelte Quell Code

DominikDierberger-IPA-svelte-support-frontend.zip

2.9.5 Zeitplan

DominikDierberger-IPA-Zeitplan.xlsx