

Prüfungsteil A

Prüfling (private Anschrift):

Ausbildungsbetrieb:

Bestätigung über durchgeführte Projektarbeit

diese Bestätigung ist mit der Projektdokumentation einzureichen

Ausbildungsberuf (bitte unbedingt angeben):

Projektbezeichnung:

Projektbeginn: _____ Projektfertigstellung: _____ Zeitaufwand in Std.: _____

Bestätigung der Ausbildungsfirma:

Wir bestätigen, dass der/die Auszubildende das oben bezeichnete Projekt einschließlich der Dokumentation im Zeitraum

vom: _____ bis: _____ selbständig ausgeführt hat.

Projektverantwortliche(r) in der Firma:

Vorname

Name

Telefon

Unterschrift

Ausbildungsverantwortliche(r) in der Firma:

Vorname

Name

Telefon

Unterschrift

Eidesstattliche Erklärung:

Ich versichere, dass ich das Projekt und die dazugehörige Dokumentation selbständig erstellt habe.

Ort und Datum: _____ Unterschrift des Prüflings: _____



Abschlussprüfung November 2023

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Entwicklung eines autonomen Support-Bots

Anwendung zur Teilautomatisierung des Kunden-Supports

Abgabetermin: Köln, den 16.11.2023

Prüfungsbewerber:

Michele Ayadi
Steprathstraße 14
51103 Köln

Ausbildungsbetrieb:

ticket i/O
Im Zollhafen 2-4
50678 Köln



next generation ticketing

Inhaltsverzeichnis

Abbildungsverzeichnis	3
Tabellenverzeichnis	4
Listings	5
Abkürzungsverzeichnis	6
Glossar	7
1 Einleitung	1
1.1 Projektumfeld	1
1.2 Projektziel	1
1.3 Projektbegründung	2
1.4 Projektschnittstellen	2
1.5 Projektabgrenzung	2
2 Projektplanung	3
2.1 Projektphasen	3
2.2 Abweichungen vom Projektantrag	3
2.3 Ressourcenplanung	4
2.4 Entwicklungsprozess	4
3 Analysephase	4
3.1 Ist-Analyse	4
3.2 Wirtschaftlichkeitsanalyse	5
3.2.1 Projektkosten	5
3.2.2 Amortisationsdauer	6
3.2.3 „Make or Buy“-Entscheidung	7
3.3 Anwendungsfälle	7
3.4 Qualitätsanforderungen	7
3.5 Lastenheft/Fachkonzept	7
4 Entwurfsphase	8
4.1 Zielplattform	8
4.2 Architekturdesign	9
4.3 Entwurf der Benutzeroberfläche	9
4.4 Datenmodell	9
4.5 Geschäftslogik	9
4.6 Maßnahmen zur Qualitätssicherung	10
4.7 Pflichtenheft	10

5	Implementierungsphase	10
5.1	Implementierung des Generativen Transformers	10
5.2	Anbindung an die Zendesk Schnittstelle	11
5.3	Implementierung der Geschäftslogik	11
6	Abnahmephase	12
7	Einführungsphase	12
8	Dokumentation	12
9	Fazit	13
9.1	Soll-/Ist-Vergleich	13
9.2	Lessons Learned	13
9.3	Ausblick	14
A	Anhang	1
A.1	Detaillierte Zeitplanung	1
A.2	Lastenheft (Auszug)	2
A.3	Anwendungsfalldiagramm	3
A.4	Klassendiagramm	4
A.5	Pflichtenheft (Auszug)	5
A.6	Prozesskostenrechnung	6
A.7	Amortisationsrechnung	7
A.8	Screenshots des Quellcodes	8
A.9	Testfall SupportAgent Klasse	10

Abbildungsverzeichnis

1	Anwendungsfalldiagramm	3
2	Klassendiagramm	4
3	Detaillierte Prozesskostenrechnung	6
4	Detaillierte Amortisationsrechnung	7
5	Ausschnitt ZendeskService-Klasse	8
6	Ausschnitt SupportAgent-Klasse	9
7	Ein Screenshot eines gelösten Support-Falles	14

Tabellenverzeichnis

1	Zeitplanung	3
2	Kostenaufstellung	6
3	Soll-/Ist-Vergleich	13

Listings

1	Testfall in Python	10
---	------------------------------	----

Abkürzungsverzeichnis

API	Application Programming Interface
GPT	Generative Pre-trained Transformers
LLM	Large Langue Model
GGUF	GPT-Generated Unified Format
GB	Gigabyte
RAM	Random Access Memory
CPU	Central Processing Unit
LLaMA	Large Language Model Meta AI

Glossar

LLaMA Name=LLaMA, Beschreibung=Ein von Meta entwickeltes generatives Sprachmodell, das auf Basis von künstlichen neuronalen Netzwerken natürliche Sprache generiert.

Large Language Model Name=Large Language Model, Beschreibung=Ein generatives Sprachmodell

LangChain Name=LangChain, Beschreibung=Eine Bibliothek um die Kommunikation mit generativen Sprachmodellen aufzureichern

LLaMA.cpp Name=LLaMA.cpp, Beschreibung=Eine Bibliothek um das LLaMA-Modell mit einer 4-Bit Tokenlänge auf einem MacBook auszuführen.

Generative pre-trained transformer Name=Generative pre-trained transformer, Beschreibung=Eine Art von generativen Sprachmodellen, welche vor ihrer Nutzung mit einem riesigen Datensatz trainiert wurden.

Halluzinationen Name=Halluzinationen, Beschreibung=Dazu Fehlinterpretieren oder Dazuerfinden von Text eines generativen Sprachmodells

GPT-Generated Unified Format Name=GPT-Generated Unified Format, Beschreibung=Vereinheitlichtes Format um mit generativen Sprachmodellen zu interagieren

Token Name=Token, Beschreibung=Im Rahmen von generativen Sprachmodellen sind es Zeichenketten, welche von diesen Modellen generiert oder entgegengenommen werden.

Zendesk Name=Zendesk, Beschreibung=Kundendienst Software, welche es ermöglicht Ticket basierte Kundenanliegen zu lösen

Makro Name=Makro, Beschreibung=Eine E-Mail-Vorlage, welche in Zendesk erstellt werden kann, um schneller auf Kundenanliegen zu reagieren.

POST-Request Name=POST-Request, Beschreibung=Eine auf dem HyperText Transfer Protocol basierende Anfrage, welche Daten zu einem Server sendet.

API-Token Name=API-Token, Beschreibung=Ein alphanumerisches Zeichen, welches genutzt wird, um sich bei Anwendungsschnittstelle zu authentifizieren.

1 Einleitung

Diese Projektdokumentation beschreibt den Verlauf meines IHK-Abschlussprojekts, das im Rahmen meiner Ausbildung zum Fachinformatiker für Anwendungsentwicklung durchgeführt wurde. Das Projekt hat zum Ziel, meine während der Ausbildung erworbenen Fähigkeiten zu demonstrieren und meine Erfahrung in einem realistischen Projektszenario zu vertiefen. Die Projektarbeit wurde im Auftrag der ticket i/O GmbH durchgeführt.

Die ticket i/O GmbH ist ein Unternehmen, das sich auf Veranstaltungsticketdienstleistungen spezialisiert hat und als Bindeglied zwischen Ticketkäufern und Veranstaltern fungiert. Mithilfe einer webbasierten Anwendung ermöglicht sie Veranstaltern die eigenständige Verwaltung von Veranstaltungen und Onlineshops im Self-Service. Die Kunden-Support-Abteilung steht den Ticketkäufern bei sämtlichen Fragen im Zusammenhang mit dem Kaufprozess zur Seite. Das Hauptziel des Projekts besteht in der Teilautomatisierung dieser Support-Abteilung.

1.1 Projektumfeld

Im Jahr 2023 verzeichnet unser Unternehmen ein beachtliches Wachstum im Kundenstamm, was zu einem drastischen Anstieg der Kunden-Support-Fälle führt. Diese Support-Anfragen weisen häufig einen eher trivialen Charakter auf und werden mit niedriger Priorität behandelt. Trotz ihrer geringen Priorität beanspruchen diese Fälle aufgrund ihrer hohen Anzahl erheblich Arbeitsaufwand in unserer Kunden-Support-Abteilung. Als Konsequenz gerieten auch einige komplexere Support-Fälle mit höherer Priorität ins Hintertreffen. Das Hauptziel dieses Projekts besteht darin, diese Arbeitsbelastung zu reduzieren. Hierfür wird ein autonomer Bot entwickelt, der in der Lage ist, diese trivialen Support-Fälle zu identifizieren und auf Basis vorgefertigter E-Mail-Vorlagen und Anweisungen zu bearbeiten. Dieser Ansatz zielt darauf ab, die Ressourcen unserer Kunden-Support-Mitarbeiter effizienter zu nutzen und sicherzustellen, dass komplexere Fälle angemessen behandelt werden können.

1.2 Projektziel

Das Hauptziel dieses Projekts besteht darin, eine benutzerfreundliche Anwendung zu entwickeln, die auf einem lokalen Rechner eines erfahrenen Kundensupport-Mitarbeiters ausgeführt wird. Diese Anwendung soll in der Lage sein, Support-Fälle effizient, autonom und präzise zu bearbeiten. Die spezifischen Funktionalitäten der Anwendung umfassen:

- Die Klassifizierung von Support-Fällen basierend auf festgelegten Kriterien.
- Die automatische Zuordnung der Klassifizierung zur richtigen E-Mail-Vorlage, um die Antwort auf den Support-Fall vorzubereiten.
- Die Auswahl und Anwendung einer ausgewählten E-Mail-Vorlage auf den Support-Fall.

- Die Einreichung des Support-Tickets über die Anwendungsschnittstelle der von unserem Kundensupport verwendeten Helpdesk-Software.

Ein Arbeitszyklus der Anwendung umfasst die genaue Klassifizierung, effiziente Verwendung der E-Mail-Vorlagen und die zügige Einreichung eines Support-Tickets. Dieser Vorgang kann beliebig oft wiederholt werden.

1.3 Projektbegründung

Die Einführung der Anwendung bietet dem Unternehmen die Möglichkeit, einen zeitunabhängigen und effizienten Kunden-Support zu gewährleisten. Dies führt zu einer Entlastung des Kundensupport-Teams, da die Anwendung eigenständig Support-Tickets analysiert, kategorisiert und geeignete Lösungen vorschlägt. Die Automatisierung von wiederkehrenden und standardisierten Anfragen ermöglicht es den Support-Mitarbeitern, sich verstärkt auf komplexe Support-Fälle zu konzentrieren.

1.4 Projektschnittstellen

Unsere Anwendung wird mit den folgenden Schnittstellen interagieren:

- Externe Schnittstelle - Zendesk: Die Anwendung wird die externe Zendesk-Anwendungsschnittstelle nutzen. Zendesk ist die von unseren Kunden-Support genutzte Helpdesk-Software. Diese Schnittstelle ermöglicht es unserer Anwendung, direkt mit den Support-Fällen zu interagieren, sie zu klassifizieren, E-Mail-Vorlagen zuzuweisen und Support-Tickets über die Zendesk-Schnittstelle einzureichen. Dies gewährleistet eine reibungslose Integration mit unserem bestehenden Kundensupport-System.
- Lokales GPT -Modell: Um die natürliche Sprache der Support-Fälle zu klassifizieren, wird unsere Anwendung ein lokales GPT -Modell nutzen. Dieses Modell spielt eine zentrale Rolle bei der Identifizierung der richtigen Klassifizierung und E-Mail-Vorlagen für die Support-Anfragen.

Es bestehen keine internen Schnittstellen zu anderen Anwendungen oder Systemen.

1.5 Projektabgrenzung

Aufgrund der begrenzten Zeitspanne von 70 Stunden Entwicklungszeit mussten folgende Einschränkungen und Abgrenzungen für dieses Projekt berücksichtigt werden:

- Keine Benutzeroberfläche: Ein Benutzeroberfläche bietet für die Umsetzung des Projekts keinen erheblichen Mehrwert. Dies bedeutet, dass die Anwendung in erster Linie auf automatische Prozesse und Schnittstellen angewiesen ist. Benutzerinteraktion wird in dieser Version nicht unterstützt.

- Nicht in der Cloud gehostet: Die Anwendung wird vorübergehend lokal auf einem Rechner eines Kundensupport-Mitarbeiters ausgeführt, um die Projektziele zu erreichen.

2 Projektplanung

2.1 Projektphasen

Die Umsetzung dieses Projekts erfolgte innerhalb eines effektiven Zeitraums von 70 Stunden. Um die Projektumsetzung optimal zu organisieren und den Zeitrahmen besser zu gliedern, habe ich mich für die Phaseneinteilung des erweiterten Wasserfallmodells entschieden. Diese Methode ermöglichte es mir, die Zeiteinteilung präziser zu gestalten, indem ich Teilaufgaben den verschiedenen Phasen des Projekts zugeordnet habe. Die Phaseneinteilung half dabei, das Projekt in klare Abschnitte zu unterteilen und die Fortschritte zu verfolgen.

Zeitplanung Ausschnitt der groben Zeiteinschätzung.

Projektphase	Geplante Zeit
Anforderungen analysieren	10 h
Entwurf erstellen	10 h
Entwurf implementieren	35 h
Tests durchführen	8 h
Inbetriebnahme	2 h
Erstellen der Dokumentaion	10 h
Pufferzeit	5 h
Gesamt	80 h

Tabelle 1: Zeitplanung

Eine ausführliche Zeitplanung findet sich im Anhang A.1: Detaillierte Zeitplanung auf Seite 1.

2.2 Abweichungen vom Projektantrag

- Aufgrund von Ressourcen- und Zeitmangel musste auf ein Fine-Tuning des Modells verzichtet werden.
- Durch eine präzise Beschreibung der Klassen, Funktion, Variablen .etc wurde auf Kommentare im Quellcode weitgehen verzichtet.
- Da die Anwendung als eine Application Bundle bereit gestellt wird, wurde auf eine Benutzerdokumentation verzichtet, da die Anwendung als ausführbares Programm verfügbar ist.

2.3 Ressourcenplanung

Die Ressourcenplanung umfasst alle direkten und indirekten Ressourcen. Dies beinhaltet das Personal, Hardware, Software und die von unserem Arbeitgeber bereitgestellte Bürofläche. Um die Kosten zu minimieren, wurde hauptsächlich auf Open-Source-Software zurückgegriffen, wodurch Lizenzgebühren eingespart werden können.

2.4 Entwicklungsprozess

Die Entwicklung dieses Projekts folgte dem erweiterten Wasserfallmodell. Die Wahl dieses Modells ergab sich aus der Tatsache, dass kein großes Entwicklerteam parallel an verschiedenen Teilaufgaben arbeitete und der Zeitrahmen des Projekts eine schlanke Methodologie begünstigte. Die Entscheidung für das erweiterte Wasserfallmodell ermöglichte mir die Flexibilität, bei Bedarf Rücksprünge in vorherige Phasen einzuräumen und neue Erkenntnisse in die bereits durchlaufenen Phasen einfließen zu lassen.

3 Analysephase

3.1 Ist-Analyse

Der Ist-Zustand, wie bereits im Abschnitt 1.1 “Projektumfeld” beschrieben, zeigt, dass im Kunden-Support vermehrt triviale und repetitive Kundenanfragen eingehen. Diese Anfragen bestehen oft aus dem erneuten Verschicken von E-Mails mit gekauften Tickets, dem Löschen von personenbezogenen Daten und der Umpersonalisierung von ticketbezogenen Informationen. Der Kunden-Support bedient sich dabei der Webanwendung Zendesk, welche die Nutzung von Makros ermöglicht. Makros sind vorgefertigte E-Mail-Vorlagen, die bereits den passenden Text und gegebenenfalls Links enthalten.

Die aktuelle Lage ergibt folgende Probleme:

- Manuelle Verknüpfung von Support-Ticket und Makro: Ein Support-Mitarbeiter muss den Inhalt eines Support-Tickets stets manuell mit dem passenden Makro verknüpfen, was zeitaufwändig und fehleranfällig ist.
- Mangelnde Klassifizierung und automatische Anwendung: Zendesk bietet keine Möglichkeit, den Inhalt zu klassifizieren und das Makro eigenständig anzuwenden, was die Effizienz einschränkt.
- Beeinträchtigung der Effizienz: Aufgrund der Menge dieser Anfragen werden andere Fälle vernachlässigt, hinzu kommt die zeitliche Begrenzung eines Mitarbeiters. Dies beeinflusst die Gesamteffizienz des Kunden-Supports und kann zu unzufriedenen Kunden führen.

3.2 Wirtschaftlichkeitsanalyse

Die Teilautomatisierung der in Abschnitt 3.1 “Ist-Analyse” beschriebenen Problemstellung bietet die Möglichkeit, die Wirtschaftlichkeit dieses Vorhabens zu validieren. Um dies zu erreichen, wird eine Prozesskostenrechnung durchgeführt, um die ungefähren Kosten der Bearbeitung eines Support-Tickets zu ermitteln.

Das Hauptziel dieser Wirtschaftlichkeitsanalyse besteht darin, die Amortisationsdauer des Projekts zu bestimmen. Dies bedeutet, dass wir die erwarteten Projektkosten aus Abschnitt 3.3 “Projektkosten” mit den eingesparten Kosten durch die Teilautomatisierung gegenüberstellen werden.

Die Prozesskostenrechnung ermöglicht es, die finanziellen Auswirkungen der Implementierung der Lösung zu bewerten und festzustellen, ob das Projekt in einem angemessenen Zeitrahmen Rendite erzielt.

3.2.1 Projektkosten

Projektkostenrechnung Die Kosten für das Projekt setzen sich aus verschiedenen Komponenten zusammen:

Personalkosten:

- Als Auszubildender im dritten Lehrjahr beträgt mein Bruttogehalt 1145€ pro Monat.
- Für die anderen Mitarbeiter wird pauschal ein Stundenlohn von 20€ angenommen.

Ressourcenkosten:

- Die allgemeinen Ressourcenkosten, einschließlich der Bürofläche und der Hardware, werden mit einem ungefähren Stundensatz von 15€ veranschlagt.

$$8 \text{ h/Tag} \cdot 220 \text{ Tage/Jahr} = 1760 \text{ h/Jahr} \quad (1)$$

$$1250 \text{ €/Monat} \cdot 12 \text{ Monate/Jahr} = 13745,6 \text{ €/Jahr} \quad (2)$$

$$\frac{13745,6 \text{ €/Jahr}}{1760 \text{ h/Jahr}} \approx 7,81 \text{ €/h} \quad (3)$$

Mit den angegebenen Komponenten und einer Projektdurchführungszeit von 70 Stunden belaufen sich die Projektkosten auf insgesamt 2086,70 €. In der unteren Tabelle 2 befindet sich eine Veranschaulichung der einzelnen Kostenträger.

Vorgang	Mitarbeiter	Zeit	Stundensatz	Ressourcen	Gesamt
Entwicklungskosten	1x Auszubildender	70 h	7,81 €	15,00 €	1596,70 €
Fachgespräch	2x Support-Mitarbeiter	5 h	20,00 €	15 €	350,00 €
Abnahme	2x Support-Mitarbeiter	2 h	20,00 €	15 €	140,00 €
					2086,70 €

Tabelle 2: Kostenaufstellung

3.2.2 Amortisationsdauer

Um die Amortisationsdauer zu bestimmen wurde mithilfe einer Prozesskostenrechnung der ungefähre Preis der Bearbeitung eines Tickets berechnet.

Die Komponenten der Berechnung bestehen aus:

Eintrittshäufigkeit einer Aktivität:

- Support-Ticket kategorisieren $\approx 10 \%$
- Lösung recherchieren $\approx 20 \%$
- Kundenkorrespondenz $\approx 30 \%$

Mitarbeiter Effizienz:

- Jahresarbeitslohn $\approx 32.000,00 \text{ €}$
- Jahresarbeitsminuten $\approx 110\text{min}$

Prozessmenge und neutrale Kosten:

- Prozessmenge ≈ 38.000 Support-Tickets pro Jahr
- Leistungsmengenneutrale Kosten $\approx 15\%$

Verrechnet man diese Komponenten ergibt sich ein ungefähre Preis von 2,06 € pro Support-Ticket. Eine detaillierte Rechnung befindet sich im Anhang A.6: Prozesskostenrechnung auf dieser Seite.

Entnimmt man dem Anhang A.7: Amortisationsrechnung auf der nächsten Seite die ungefähren laufenden Betriebskosten des Projekts von 0,40 € und der geschätzten Arbeitsleistung von 10 Support-Tickets pro Tag kommt man auf eine Amortisationsdauer von 103 Tage was ungefähr 3,3 Monaten entspricht. Die genaue Aufstellung kann dem genannten Anhang entnommen werden.

3.2.3 „Make or Buy“-Entscheidung

Die Entscheidung, keine alternative Software in Betracht zu ziehen und stattdessen die Entwicklung mit Generative Pre-trained Transformers (GPT)-Modellen fortzusetzen, basierte auf mehreren Faktoren. Die Sensibilität von personenbezogenen Daten und die Notwendigkeit, strenge Datenschutz- und Sicherheitsrichtlinien einzuhalten, erforderten eine Lösung, die diesen Anforderungen gerecht wird. Die Integration der entwickelten Lösung mit Zendesk und die Sicherstellung der Einhaltung dieser Richtlinien waren von entscheidender Bedeutung. Es gab keine vorhandene Software, die sowohl die erforderlichen Datenschutzstandards erfüllte als auch nahtlos mit Zendesk integriert werden konnte und gleichzeitig einen niedrigeren wirtschaftlichen Aufwand mit sich brachte.

3.3 Anwendungsfälle

Das im Anhang A.3: Anwendungsfalldiagramm auf Seite 3 bildet die zentralen Anwendungsfälle ab, die Interaktionen zwischen Benutzern, Fremdsystemen und der Anwendung veranschaulichen. Diese Anwendungsfälle werden während der Implementierung in die einzelnen Features und Funktionen der Anwendung aufgeschlüsselt.

3.4 Qualitätsanforderungen

Um die Qualität der Anwendung zu gewährleisten und Fehlerquellen zu minimieren, wurden bestimmte Schlüsselwörter festgelegt, die von der Anwendung zur Klassifizierung von Support-Tickets verwendet werden. Ein besonders wichtiger Aspekt dieser Qualitätsanforderungen besteht darin, dass die Anwendung in der Lage sein muss, Halluzinationen oder fehlerhafte Klassifizierungen des GPT-Modells zu verhindern.

3.5 Lastenheft/Fachkonzept

In Zusammenarbeit mit den Stakeholder haben sich im Anhang A.2: Lastenheft (Auszug) auf Seite 2 die sich darin befindlichen Anforderungen an die Anwendung herauskristallisiert.

4 Entwurfsphase

4.1 Zielplattform

Bei Wahl der Zielplattform gab es einige Auswahlkriterien zu beachten, um die Implementierung zu erleichtern. Diese teilen sich in folgende Punkte auf.

Schlankes und leistungsstarkes GPT-Modell:

- Die gewählte Zielplattform erforderte ein GPT -Modell, das in der Lage ist, mit begrenzten Ressourcen in angemessener Zeit qualitativ hochwertige Antworten zu generieren.

Freie kommerzielle Nutzung des GPT-Modells:

- Die Möglichkeit, das GPT-Modell kommerziell frei zu nutzen, ist ein wichtiger Faktor, um Lizenzbeschränkungen zu vermeiden.

Kompatibilität mit einem lokalen GPT-Modell:

- Die reibungslose Integration mit einem lokalen GPT-Modell ist von entscheidender Bedeutung, um die Anforderungen mit dem fachgerechten Umgang mit personenbezogenen Daten, des Projekts zu erfüllen.

Kompatibilität mit der genutzten LangChain-Bibliothek:

- Da die Kernlogik des Projekts hauptsächlich auf dem Datenaustausch zwischen dem lokalen GPT -Modell und der Zendesk-Application Programming Interface (API) basiert, war die Kompatibilität mit der LangChain-Bibliothek von großer Relevanz, da diese Bibliothek die Daten anreichert.

Erstellung eines ausführbaren Programms für macOS:

- Die Zielplattform sollte die Möglichkeit bieten, ein ausführbares Programm für macOS zu generieren, um sicherzustellen, dass die Anwendung in der gewünschten Umgebung ausgeführt werden kann.

Basierend auf diesen Kriterien wurde die Entscheidung für das 7B Large Language Model Meta AI (LLaMA)-2 Modell von Meta als Zielplattform getroffen. Dieses Modell zeichnet sich durch seine kompakte Größe von 13 Gigabyte (GB) aus und kann auf einem Rechner mit lediglich einer Central Processing Unit (CPU) und 16 GB Random Access Memory (RAM) effizient betrieben werden. Die Auswahl der Programmiersprache wurde stark von der Kompatibilität mit dem GPT-Modell beeinflusst. Python wurde als die geeignete Programmiersprache gewählt, da sie eine einfache Anbindung an

die LangChain-Bibliothek ermöglicht, die wiederum die Verwaltung des GPT-Modells erleichtert. Darüber hinaus bietet Python einen einfachen Weg, um plattformunabhängige ausführbare Programme zu generieren.

4.2 Architekturdesign

Es wird entschieden eine objektorientierte Architektur zu verwenden, um die Anwendung in sinnvolle Entitäten zu strukturieren. Dies ermöglicht eine bessere Wartbarkeit und Erweiterbarkeit der einzelnen Funktionen, da sie mit geringeren Abhängigkeiten voneinander entwickelt werden können. Eine weitere wesentliche Entscheidung bestand darin, die LangChain-Bibliothek zu verwenden. Diese Bibliothek erfüllt mehrere wichtige Funktionen in der Architektur. Zum einen ermöglicht sie die Anreicherung der Eingabe für das LLaMA-2 Modell. Darüber hinaus bietet sie die Möglichkeit zur Verwaltung verschiedener Modelle. Die Einbindung der LangChain Bibliothek spielt eine entscheidende Rolle, da sie dazu beiträgt, den Inhalt der Support-Tickets effizienter zu verarbeiten. Eine ausführliche Auflistung der Klassen und deren Zusammenwirken finden Sie im Anhang A.4: Klassendiagramm auf Seite 4

4.3 Entwurf der Benutzeroberfläche

Für diese Anwendung wurde auf eine Benutzeroberfläche verzichtet.

4.4 Datenmodell

Die Anwendung verwendet keine direkte Datenbankanbindung. Da der Kundensupport bereits die Zendesk-Software für alle relevanten Anfragen verwendet, wird diese für das Überwachen und Persistieren der erforderlichen Daten genutzt. Die Anwendung greift auf die Daten in Zendesk zu, ohne separate Datenbanken zu verwenden.

4.5 Geschäftslogik

Die Geschäftslogik der Anwendung besteht aus zwei zustandslosen Entitäten, die jeweils als Wrapper für die Zendesk-Schnittstelle und dem LLaMA-2-Modell fungieren. Diese Logik teilt sich in den Datenaustausch mit der Zendesk-API und die Anreicherung von Texten mit dem LLaMA-2-Modell auf. Der Prozess beginnt damit, Support-Tickets von der Zendesk-API abzurufen und die ausgewählten Eigenschaften in Support-Ticket-Entitäten umzuwandeln. Als nächster Schritt erfolgt die Klassifizierung der Tickets, um das passende Makro aus Zendesk anzuwenden. Dieses Makro wird in einem POST-Request an die Zendesk-API gesendet.

4.6 Maßnahmen zur Qualitätssicherung

Um die Qualität der Anwendung zu gewährleisten und einen reibungslosen Betrieb sicherzustellen, wurden Unit-Tests und Integrationstests implementiert. Diese Tests ermöglichen die Überprüfung und Sicherstellung der Kernlogik sowohl in ihren einzelnen Komponenten als auch als Gesamtsystem ordnungsgemäß funktioniert. Dies trägt zur besseren Wartbarkeit und Erweiterbarkeit der Anwendung bei.

4.7 Pflichtenheft

Der Anhang A.5: Pflichtenheft (Auszug) auf Seite 5 umfasst alle technischen Kriterien, die in der Anwendung umgesetzt werden müssen. Es entstand aufgrund der Arbeitsergebnisse der Entwurfsphase. Dieses Dokument ist von entscheidender Bedeutung für die präzise Planung der Implementierungsphase, da es Aufgaben definiert, die zeitlich gut einschätzbar sind.

5 Implementierungsphase

5.1 Implementierung des Generativen Transformers

Die Implementierung des Generativen Transformers erfolgte unter Verwendung des Klassendiagramms aus Kapitel 4.2: Architekturdesign als Vorlage. Der erste Schritt bestand darin, die lokale Konfiguration des LLaMA-2 Modells durchzuführen. Um diese Konfiguration zu starten, musste das Modell zunächst heruntergeladen werden. Hierfür wurde das Repository von Meta geklont und nach der Registrierung auf der Webseite wurde ein Freischaltungslink erhalten. Dieser Link wurde als Parameter für ein Bash-Skript verwendet, um das gewünschte Modell auszuwählen und herunterzuladen.

Die folgenden Schritte umfassten:

- Die Konvertierung des Modells in das GPT-Generated Unified Format (GGUF)-Format.
- Die Quantifizierung des Modells.

Mithilfe der LLaMA.cpp-Bibliothek war es möglich, das Modell mithilfe der `convert.py`-Funktion in das Zielformat GGUF zu konvertieren. Dieses Format ist ein generalisiertes Format für Large Language Model (LLM)s und bietet den Vorteil, dass ein LLM nur mit einer CPU betrieben werden kann. Aufgrund begrenzter Hardware-Ressourcen mussten die Tokens des LLaMA-2-Modells gekürzt werden. Tokens sind Wortkombinationen, die mit einer bestimmten Wahrscheinlichkeit aneinandergereiht werden. Standardmäßig haben Tokens eine Länge von 16 Bit. Mit der `quantize`-Funktion von LLaMA.cpp wurden diese auf 4 Bits gekürzt, um eine schnellere Verarbeitung der Eingabe zu ermöglichen, jedoch auf Kosten der Genauigkeit der Ausgabe.

5.2 Anbindung an die Zendesk Schnittstelle

In diesem Schritt wurde eine Verbindung zur Zendesk-Anwendungsschnittstelle hergestellt. Es gab verschiedene Möglichkeiten zur Authentifizierung. Anstelle von Benutzername und Passwort wurde sich für die Verwendung eines API-Tokens entschieden. Die Wahl des API-Tokens erschien in diesem Fall sicherer, da dieser Token ausschließlich für diese Anwendung gilt, somit wird der Branchenstandard für Machine to Machine Kommunikation gewährleistet. Durch die Kombination von Benutzername und API-Token ist es möglich, im Namen des Nutzers mit der Anwendungsschnittstelle zu interagieren. Dies ermöglicht die Überwachung der Support-Bots, da bei jeder Anfrage an die API die erforderlichen Informationen mitgesendet werden.

Ein Screenshot der implementierten Klasse befindet sich im Anhang A.8: Screenshots des Quellcodes auf Seite 8.

5.3 Implementierung der Geschäftslogik

Die Geschäftslogik konzentriert sich hauptsächlich auf die Steuerung des Datenflusses zwischen der Zendesk-Anwendungsschnittstelle und dem LLaMA-2-Modell. Ein Anwendungszyklus beginnt mit der Abfrage der Zendesk-Schnittstelle, die im ZendeskService durchgeführt wird, um die Daten eines Support-Tickets abzurufen.

Diese Daten werden anschließend in eine SupportTicket-Klasse umgewandelt.

Der Inhalt der Support-Ticket-Klassen wird mithilfe von LangChain in eine Vorlagen-Eingabe eingebettet. Diese Eingabe fordert das LLaMA-Modell auf, als Support-Mitarbeiter zu agieren und auf bestimmte Schlüsselwörter zu achten. Zum Beispiel, wenn das Ticket die Beschreibung "E-Mail verloren, Neue Tickets" enthält. Wird nun das Schlüsselwort **RESEND_TICKET** erwartet.

Diese Logik wird in der `classify_tickets-Methode` der SupportAgent-Klasse umgesetzt.

Nach der Klassifizierung des Inhalts des Support-Tickets wird das `classification-Attribut` der SupportTicket-Klasse auf das entsprechende Schlüsselwort gesetzt. In einem weiteren Schritt wird dieses Attribut ausgelesen, um das passende Makro von der Zendesk-Anwendungsschnittstelle aufzurufen. Die Antwort des Aufrufs wird dann in die Payload der `reply_to_customer-Methode` der ZendeskService-Klasse eingebettet, um eine Antwort für das Support-Ticket zu generieren. Dabei wird das `ticket_id-Attribut` verwendet, um die richtige Antwort dem entsprechenden Ticket zuzuordnen.

Auschnitt Ein Ausschnitt der SupportAgent-Klasse befindet sich im Anhang A.8: Screenshots des Quellcodes auf Seite 8.

6 Abnahmephase

Testprotokoll In der Abnahmephase werden Integrationstests mithilfe des Python unittest-Pakets durchgeführt. Die externen Schnittstellen der Klassen werden gemockt, wodurch die Antworten der Zendesk-Schnittstelle und des LLaMA-2-Modells nachgebildet werden, um die Korrektheit der internen Funktionen zu validieren. Darüber hinaus wird die Anwendung manuell getestet, indem sie realistische Test-Support-Fälle bearbeitet.

Offizielle Abnahme Bei den manuellen Tests stellt sich heraus, dass die Anwendung noch nicht präzise genug ist, um für den produktiven Einsatz freigegeben zu werden. Dies ist hauptsächlich auf die begrenzte Entwicklungszeit und die eingeschränkten Hardware-Ressourcen zurückzuführen. Um die erforderliche Präzision sicherzustellen, wäre es notwendig, das LLaMA-2-Modell speziell für diesen Anwendungsfall zu trainieren. Aus diesem Grund konnte die Anwendung noch nicht in Produktion genommen werden.

Beispiel Ein Auszug eines Unit Tests befindet sich im Anhang A.9: Testfall SupportAgent Klasse auf Seite 10. Dort ist auch der Aufruf des Tests auf der Konsole des Webservers zu sehen.

7 Einführungsphase

Da in unserem Unternehmen ausschließlich MacOS-Systeme verwendet werden, musste die Anwendung in ein ausführbares Applikationsbundle umgewandelt werden. Dieser Vorgang wurde manuell mithilfe des PyInstaller-Pakets durchgeführt. Um das LLaMA-2-Modell einzubinden, mussten externe Abhängigkeiten referenziert werden. Anschließend wurden mit PyInstaller alle Anwendungsdateien zu einer einzigen Datei gebündelt, die in ein ausführbares Programm umgewandelt wurde. Da die Anwendung aus nur einer ausführbaren Datei besteht, war keine Schulung erforderlich.

8 Dokumentation

Die Anwendung wird als ausführbares Programm bereitgestellt, und die Bearbeitung der Tickets erfolgt im Hintergrund. Es war eine Anforderung, dass das Verhalten der Anwendung mithilfe der bereits genutzten Helpdesk-Software nachvollziehbar sein soll, um eine einfachere Nutzung zu gewährleisten. Aus diesem Grund wurde auf eine Nutzerdokumentation verzichtet.

9 Fazit

9.1 Soll-/Ist-Vergleich

Das Projektziel wurde nicht vollständig erreicht. Die Distribution über eine ausführbare Anwendung brachte einige Komplikationen mit sich. Der Datenaustausch zwischen Anwendung und LLaMA-2 Model ist bei der Erstellung trotz referenzierter Daten eines Application Bundles verloren gegangen. Weitere Recherche zu betreiben, wie sich dieses Problem im konkreten Anwendungsfall lösen lässt, war aufgrund der zeitlichen Begrenzung dieses Projekts nicht möglich. Die Anwendung konnte ebenfalls nicht in Produktion gehen, da die Genauigkeit des LLaMA-2 Model nicht messbar war.

Soll-/Ist-Vergleich (verkürzt) Die Zeitplanung konnte bis auf wenige Ausnahmen eingehalten werden.

Phase	Geplant	Tatsächlich	Differenz
Analysephase	10 h	10 h	
Entwurfsphase	10 h	10 h	
Implementierungsphase	35 h	37 h	+2 h
Tests	8 h	4 h	-4 h
Inbetriebnahme	2 h	5 h	+3 h
Erstellen der Dokumentation	10 h	14 h	+4 h
Pufferzeit	5 h	0 h	-5 h
Gesamt	80 h	80 h	

Tabelle 3: Soll-/Ist-Vergleich

9.2 Lessons Learned

Recherche und Planung: Die Bedeutung einer gründlichen Recherche und Planung wurde deutlich. Dies gilt besonders, wenn mit neuen Technologien gearbeitet wird, die nicht im täglichen Einsatz des Unternehmens sind. Zukünftige Projekte könnten von einer umfassenderen Vorrecherche profitieren.

Neue Technologien: Der Umgang mit neuen Technologien, wie den Generativen Transformern, kann Herausforderungen mit sich bringen. Es ist wichtig, genügend Zeit für das Erlernen und Verstehen dieser Technologien einzuplanen.

Verwendete Programmiersprache: Der Einsatz von Python, auch wenn nicht aktiv von unseren Unternehmen verwendet, zeigt, dass die Verwendung einer breit akzeptierten und unterstützten Programmiersprache Vorteile bringt. Zukünftige Projekte könnten davon profitieren, bereits im Unternehmen etablierte Sprachen zu bevorzugen.

Frühzeitige Einbindung von Informationen: Informationen über neue Technologien sollten frühzeitig in die Planung einbezogen werden. Dies ermöglicht es, besser auf mögliche Hindernisse vorbereitet zu sein.

9.3 Ausblick

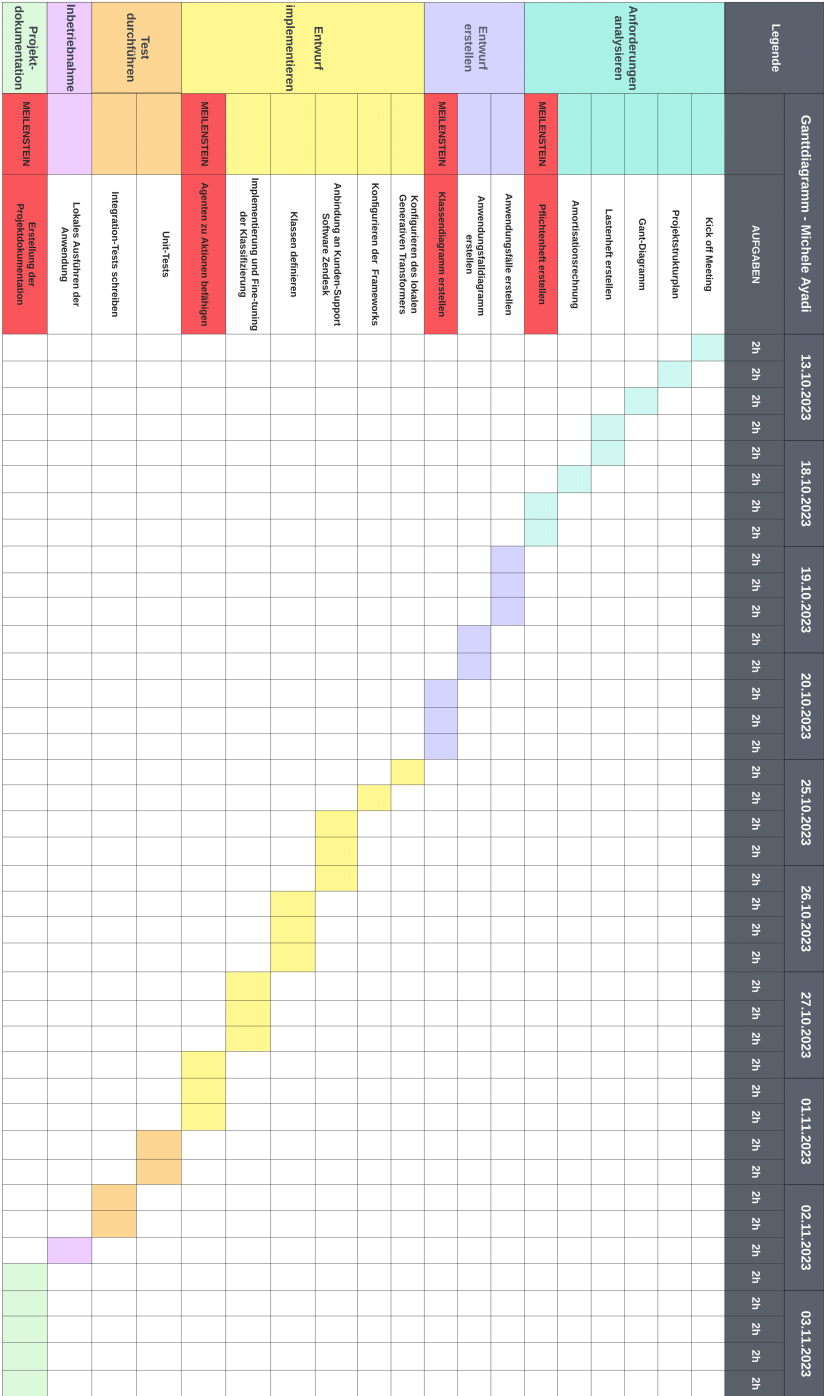
Der Ausblick auf die Weiterentwicklung der Anwendung zeigt vielversprechende Potenziale. Mit einer Erweiterung der Anwendungsfälle und einer verbesserten Präzision bei der Generierung von Antworten könnte das Projekt einen erheblichen Mehrwert für das Unternehmen bieten und somit in den operativen Betrieb übergehen.

Ein vielversprechender Anwendungsfall könnte darin bestehen, Echtzeitdaten in die Anwendung zu integrieren, insbesondere in Bezug auf die Verwaltungsoberfläche des Unternehmens. In der Veranstaltungsbranche, die oft von schnellen Veränderungen und Unvorhersehbarkeiten geprägt ist, kann die Fähigkeit, auf Echtzeitereignisse zu reagieren, entscheidend für den Erfolg sein. Sollte die Anwendung in der Lage ist, diese Echtzeitereignisse präzise zu klassifizieren, eröffnet sich ein breites Spektrum an lösbaren Support-Fällen.

Die kontinuierliche Verbesserung der Anwendung im Hinblick auf neue Anforderungen und Erkenntnisse könnte dazu beitragen, dass die Lösung nicht nur den aktuellen Bedürfnissen gerecht wird, sondern auch zukünftige Herausforderungen erfolgreich bewältigen kann.

A Anhang

A.1 Detaillierte Zeitplanung



A.2 Lastenheft (Auszug)

Es folgt ein Auszug aus dem Lastenheft mit Fokus auf die Anforderungen:

Die Anwendung muss folgende Anforderungen erfüllen:

1. Ticket-Einstufung und Kompetenzbewertung:
 - 1.1. Die Anwendung muss Support-Tickets eigenständig analysieren und korrekt in Kategorien oder Schwierigkeitsgrade einstufen können.
 - 1.2. Sie sollte die eigene Kompetenz bei der Lösung von Support-Tickets bewerten und sicherstellen, dass sie nur Tickets bearbeitet, für die sie ausreichend qualifiziert ist.
2. Zugriff auf E-Mail-Vorlagen:
 - 2.1. Die Anwendung sollte in der Lage sein, auf eine Vielzahl von vordefinierten E-Mail-Vorlagen für den Kunden-Support zuzugreifen und diese bei Bedarf in die Kommunikation einzubinden.
3. Verhaltenstransparenz und Export:
 - 3.1. Die Anwendung sollte in der Lage sein, ihre Aktionen und Entscheidungen in einem exportierbaren Format bereitzustellen, um die Nachvollziehbarkeit und Analyse des Bot-Verhaltens zu ermöglichen.
4. Bedienbarkeit
 - 4.1. Die Anwendung sollte als ausführbares Programm bereitgestellt werden.

A.3 Anwendungsfalldiagramm

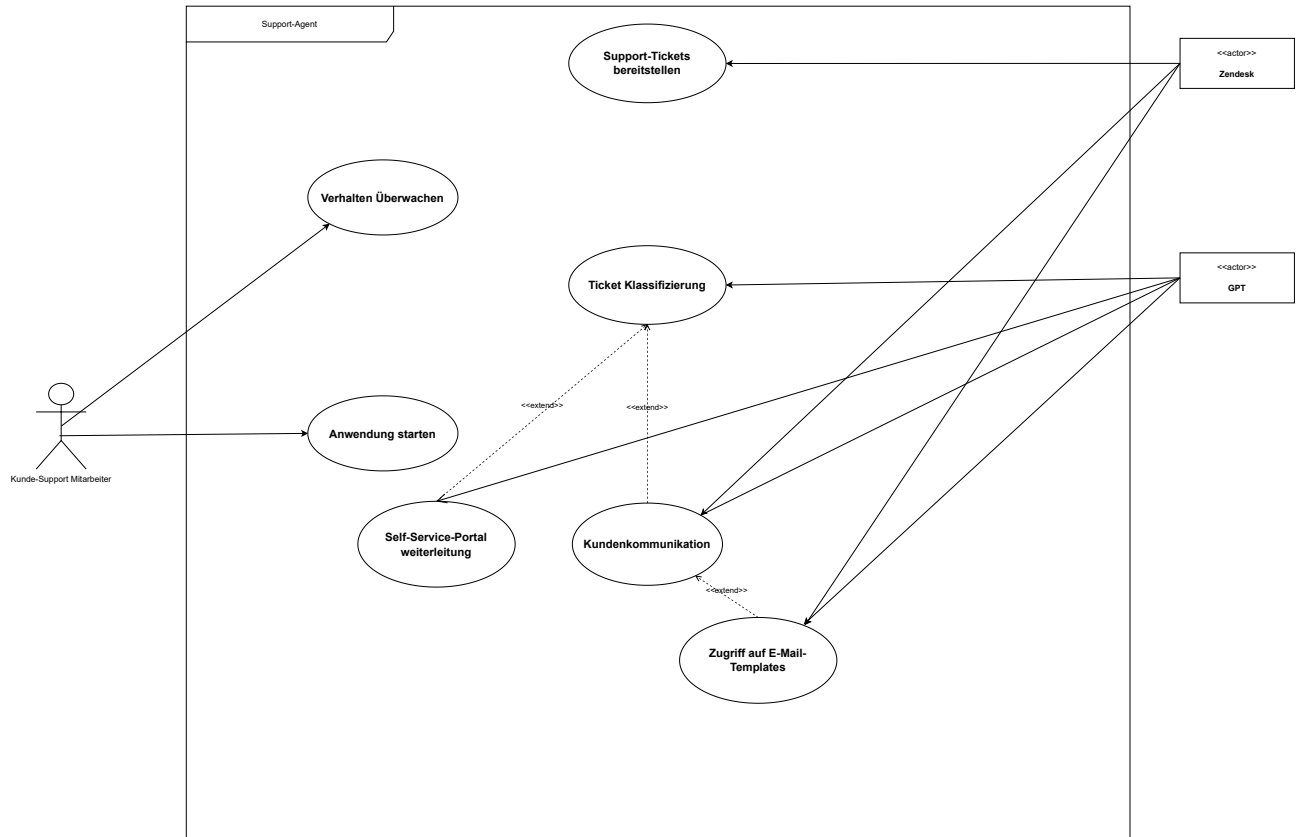


Abbildung 1: Anwendungsfalldiagramm

A.4 Klassendiagramm

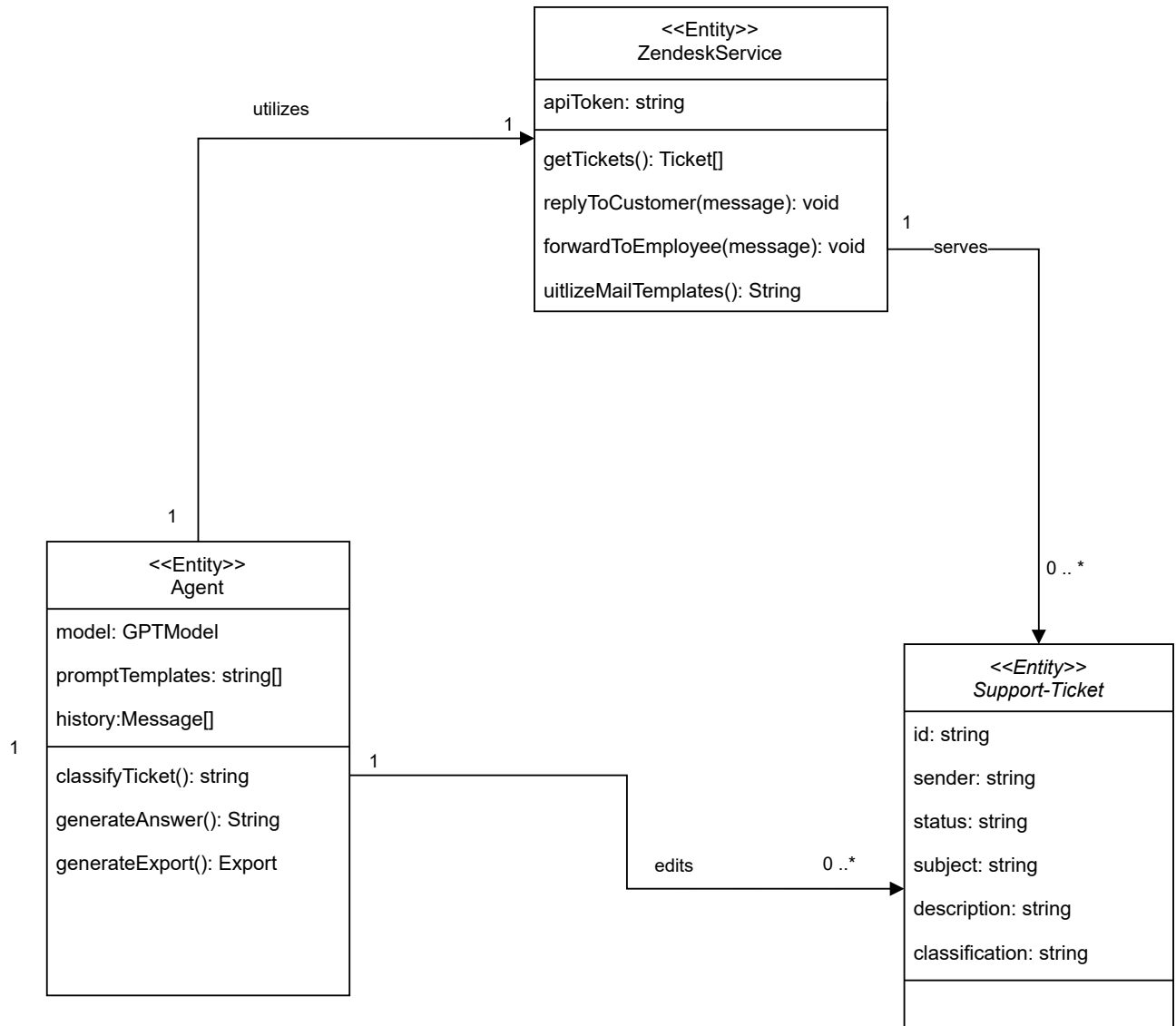


Abbildung 2: Klassendiagramm

A.5 Pflichtenheft (Auszug)

Zielbestimmung

Das vorliegende Pflichtenheft stellt die spezifischen Anforderungen an die Anwendung "SupportBot" dar. Diese Anforderungen dienen als Grundlage für die Implementierung und Entwicklung der Anwendung.

1. Musskriterien

1.1. Ticket-Einstufung und Kompetenzbewertung

- Die Anwendung muss in der Lage sein, Support-Tickets eigenständig zu analysieren und korrekt in bestimmte Kategorien oder Schwierigkeitsgrade einzustufen. Dies dient dazu, die Priorität und Bearbeitungshierarchie festzulegen.
- Die Anwendung sollte die eigene Kompetenz bei der Lösung von Support-Tickets bewerten und sicherstellen, dass sie nur Tickets bearbeitet, für die sie ausreichend qualifiziert ist. Dies gewährleistet eine effiziente Ticketzuweisung.

1.2. Zugriff auf E-Mail-Vorlagen

- Die Anwendung sollte in der Lage sein, auf eine Vielzahl von vordefinierten E-Mail-Vorlagen für den Kunden-Support zuzugreifen. Bei Bedarf kann sie diese Vorlagen in die Kommunikation mit den Kunden einbinden. Dies ermöglicht eine konsistente und effektive Kommunikation.

1.3. Verhaltenstransparenz und Export

- Die Anwendung sollte die Fähigkeit besitzen, ihre Aktionen und Entscheidungen in einem exportierbaren Format bereitzustellen. Dies dient der Nachvollziehbarkeit und Analyse des Bot-Verhaltens. Das exportierbare Format sollte für Analysezwecke geeignet sein.

1.4. Bedienbarkeit

- Die Anwendung wird als ausführbares Programm bereitgestellt. Dies ermöglicht die einfache Installation und Nutzung auf den Rechnern der Kundensupport-Mitarbeiter.

A.6 Prozesskostenrechnung

Eintrittshäufigkeit einer Aktivität:

- Support-Ticket kategorisieren $\approx 10\%$
- Lösung recherchieren $\approx 20\%$.
- Kunden Korrespondenz $\approx 30\%$.

Leistungsmengenneutrale Kosten $\approx 15\%$

Mitarbeiter Effizienz:

- Jahresarbeitslohn $\approx 32.000,00\text{€}$
- Jahresarbeitsminuten $\approx 110.400\text{min}$

Prozesskosten-- Support-Ticket							
Aktivität	Prozessmenge	Zeit in Min.	Zeit gesamt	Personenjahre	Prozesskosten (Imi)	Prozesskosten (Imn)	Prozesskosten (Imi+Imn)
Kundenanliegen aufnehmen	38.000	3	114.000min	1,03	32.960,00€	4.944,00€	37.904,00€
Support-Ticket identifizieren	38.000	0,5	19.000min	0,17	5.440,00€	816,00€	6256,00€
Support-Ticket kategorisieren	3800	0,5	1.900min	0,017	544,00€	81,60€	625,60€
Lösung recherchieren	8600	4	34.400min	0,31	9.920,00€	1.488,00€	11.408,00€
Kunden Korrespondenz	11400	6	68.400min	0,61	19.520,00€	2.928,00€	22.448,00€
Gesamt					68.384,00€	10.257,60€	78.641,60€

Kosten pro Support-Ticket Bearbeitung =
Gesamtprozesskosten / Kostentreiber

78.641,60€ / 38.000 Support-Tickets = 2,06€ pro Support-Ticket

Abbildung 3: Detaillierte Prozesskostenrechnung

A.7 Amortisationsrechnung

Amortisationsrechnung:

Support-Agent:

- Betriebskosten $\approx 0,40\text{€}/\text{Tag}$
- Gelöste Support-Tickets/Tag ≈ 10
- Ersparnisse = $20,60\text{€}/\text{Tag}$

$$f(x) = 0.4x + 2086$$

$$g(x) = 20.60x + 0$$

$$\begin{array}{rcl} 0.4x + 2086 = 20.6x & | & -0.4x \\ 2086 = 20.2x & | & : 20.2 \end{array}$$

$$\mathbf{X = 103}$$

Das Projekt hätte sich somit nach 103 Tagen erfolgreich laufenden Betrieb amortisiert.

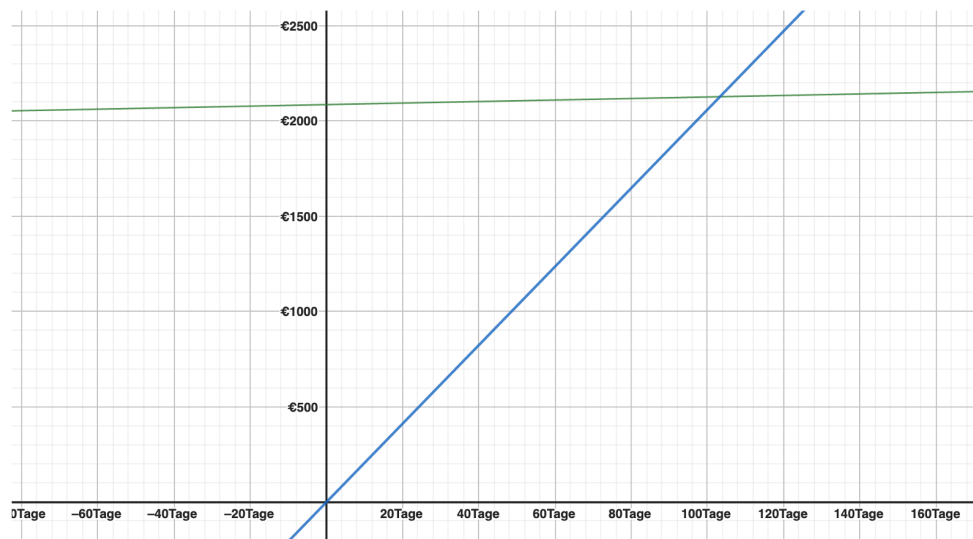


Abbildung 4: Detaillierte Amortisationsrechnung

A.8 Screenshots des Quellcodes

```
9  class ZendeskService:
10  def __init__(self):
11      load_dotenv(find_dotenv())
12      self.base_url = "https://ticketio.zendesk.com/api/v2/"
13  def headers = {
14      "Content-Type": "application/json",
15  }
16  self.auth = (f'{os.environ.get("USERNAME")}/token', os.environ.get("API_TOKEN"))
17
18  def get_tickets(self, count: int):
19  try:
20      params = {
21          "query": f"type:ticket group:1. Level Customer Support status:open count:{count}",
22          "sort_by": "created_at",
23          "sort_order": "asc",
24      }
25      response = requests.get(
26          self.base_url + "tickets",
27          params=params,
28          auth=self.auth,
29          headers=self.headers,
30      )
31
32      support_tickets = self.__generate_support_tickets(json.loads(response.text))
33      return support_tickets
34
35  except Exception as e:
36      print(e)
37      raise
38
39  def reply_to_customer(self, ticket_id, payload):
40  response = requests.put(
41      self.base_url + f"tickets/{ticket_id}",
42      auth=self.auth,
43      headers=self.headers,
44      json=payload,
45  )
46  return response
47
```

Abbildung 5: Ausschnitt ZendeskService-Klasse

```

11 class SupportAgent:
12     def __init__(self):
13         # Initialize the SupportAgent with default parameters
14         self.llm = Llama(
15             model_path="/Users/michele/Documents/Arbeit/Projektarbeit/support-agent/llama.cpp/models/7B/ggml-model-q4_1.g
16             temperature=0.6,
17             max_tokens=2000,
18             n_ctx=2048,
19             top_p=0.8,
20             callback_manager=CallbackManager([StreamingStdOutCallbackHandler()]),
21             verbose=True,
22             repeat_penalty=0.8,
23         )
24         self.zendesk_service = ZendeskService()
25         self.history = []
26         self.template = """
27         <s>[INST]<<SYS>>
28         Du bist Supportio, ein qualifizierter Kunden-Support Mitarbeiter.
29         Deine Aufgabe ist es,Support-Tickets zu klassifizieren.
30         Du antwortest immer mit einem Wort.
31         Analysiere das Support-Ticket
32         Schritt für Schritt, um eine präzise Antwort zu geben.
33
34         Antworte mit:
35
36         RESEND_TICKET
37         wenn das Support-Ticket die folgenden Worte oder ähnliche Sätze enthält:
38
39         "Neu Senden"
40         "Neue Tickets"
41         "Ich finde meine Tickets nicht"
42         "Wo finde ich meine Tickets"
43         "Keine E-Mail erhalten"
44         "Ich komme nicht mehr an die Bestätigungs-Mail mit den Tickets"
45         "Email fach geleert"
46         "Email wiederherstellen"
47
48         DELETE_ACCOUNT
49         wenn das Support-Ticket die folgenden Worte oder ähnliche Sätze enthält:
50
51         "Account löschen"
52         "Ich möchte meinen Account löschen"

```

Abbildung 6: Ausschnitt SupportAgent-Klasse

A.9 Testfall SupportAgent Klasse

```
1 import json
2 import unittest
3 from unittest.mock import Mock, patch
4 from supportagent.support_agent import SupportAgent
5 from supportagent.support_ticket import SupportTicket
6
7
8 class TestSupportAgent(unittest.TestCase):
9     def setUp(self):
10         # Create a SupportAgent instance for testing
11         self.support_agent = SupportAgent()
12
13     def test_classify_tickets ( self ):
14         # Create a list of mock SupportTicket objects
15         support_tickets = [
16             SupportTicket(
17                 ticket_id=1,
18                 customer_email="test@example.com",
19                 status="open",
20                 subject="Test subject",
21                 description="Test description",
22                 classification =None,
23             ),
24             SupportTicket(
25                 ticket_id=2,
26                 customer_email="test2@example.com",
27                 status="open",
28                 subject="Test subject 2",
29                 description="Test description 2",
30                 classification =None,
31             ),
32         ]
33
34         # Mock the Llama class's response
35         llama_output = {"choices": [{"text": "RESEND_TICKET"}]}
36         self.support_agent.llm = Mock(return_value=llama_output)
37
38         # Call the classify_tickets method
39         classified_tickets = self.support_agent._SupportAgent__classify_tickets(
40             support_tickets
41         )
42
43         # Ensure that the classification is set correctly
44         self.assertEqual( classified_tickets [0]. classification , "RESEND_TICKET")
```

```

45     self.assertEqual( classified_tickets [1]. classification , "RESEND_TICKET")
46
47     def test_generate_response_body(self):
48         # Create a mock macro template with valid JSON format
49         macro_template = '{"result": {"ticket": {"comment": {"html_body": "Test body", "public": false}}}}'
50
51         # Call the __generate_response_body method with the mocked template
52         with patch("support_agent.json.loads", return_value=json.loads(macro_template)):
53             response_body = self.support_agent._SupportAgent__generate_response_body(
54                 macro_template
55             )
56
57         # Ensure that the response body is generated correctly
58         expected_body = {
59             "result": {
60                 "ticket": {"comment": {"html_body": "Test body", "public": False}}
61             }
62         }
63         self.assertEqual(response_body, expected_body)
64
65     def test_map_classification(self):
66         # Create a mock SupportTicket with a classification
67         support_ticket = SupportTicket(
68             ticket_id=1,
69             customer_email="test@example.com",
70             status="open",
71             subject="Test subject",
72             description="Test description",
73             classification = "RESEND_TICKET",
74         )
75
76         # Mock the ZendeskService methods
77         self.support_agent.zendesk_service.utilize_mail_template = Mock()
78         self.support_agent.zendesk_service.reply_to_customer = Mock()
79
80         # Mock the behavior of __generate_response_body to return a valid JSON string
81         with patch(
82             "support_agent.SupportAgent._SupportAgent__generate_response_body",
83             return_value='{"result": {"ticket": {"comment": {"html_body": "Test body", "public": false}}}}',
84         ):
85             # Call the __map_classification method
86             self.support_agent._SupportAgent__map_classification(support_ticket)
87
88         # Ensure that the ZendeskService methods were called correctly
89         self.support_agent.zendesk_service.utilize_mail_template.assert_called_with(
90             macro_id=8140353174289
91         )
92         # Ensure that the payload is a JSON string in the expected call
93         self.support_agent.zendesk_service.reply_to_customer.assert_called_with(
94             ticket_id=support_ticket.ticket_id,

```

```

95     payload={'result': {"ticket": {"comment": {"html_body": "Test body", "public": false}}}},
96 )
97
98 def test_generate_export(self):
99     # Create a list of mock interaction data
100     interactions = [
101         {
102             "1": {
103                 "support_ticket": {
104                     "ticket_id": 120052,
105                     "customer_email": "test@example.com",
106                     "status": "open",
107                     "subject": "Test subject",
108                     "description": "Test description",
109                     "classification": "RESEND_TICKET",
110                 },
111                 "llama_output": {
112                     "id": "cmpl-c4cdeb89-3fac-4a54-a3f2-2873a9aa4b7a",
113                     "object": "text_completion",
114                     "created": 1696518220,
115                     "model": "/Users/michele/Documents/Arbeit/Projektarbeit/support-agent/llama.cpp/models
116                        /7B/ggml-model-q4_1.gguf",
117                     "choices": [
118                         {
119                             "text": " ",
120                             "index": 0,
121                             "logprobs": "null",
122                             "finish_reason": "length",
123                         }
124                     ],
125                     "usage": {
126                         "prompt_tokens": 684,
127                         "completion_tokens": 128,
128                         "total_tokens": 812,
129                     },
130                 },
131             },
132             {
133                 "2": {
134                     "support_ticket": {
135                         "ticket_id": 120052,
136                         "customer_email": "test@example.com",
137                         "status": "open",
138                         "subject": "Test subject2",
139                         "classification": "RESEND_TICKET",
140                     },
141                     "llama_output": {
142                         "id": "cmpl-c4cdeb89-3fac-4a54-a3f2-2873a9aa4b7a",
143                         "object": "text_completion",


```

```

144         "created": 1696518220,
145         "model": "/Users/michele/Documents/Arbeit/Projektarbeit/support-agent/llama.cpp/models
           /7B/ggml-model-q4_1.gguf",
146         "choices": [
147             {
148                 "text": "",
149                 "index": 0,
150                 "logprobs": "null",
151                 "finish_reason": "length",
152             }
153         ],
154         "usage": {
155             "prompt_tokens": 684,
156             "completion_tokens": 128,
157             "total_tokens": 812,
158         },
159     },
160 }
161 },
162 ]
163
164 # Set the history attribute with mock data
165 self.support_agent.history = interactions
166
167 # Call the __generate_export method
168 self.support_agent._SupportAgent__generate_export()
169
170 # Ensure that the export file is created and written
171 with open("/test/test_history.json", "r") as outfile :
172     exported_data = json.load(outfile)
173
174     self.assertEqual(exported_data, interactions)
175
176
177 if __name__ == "__main__":
178     unittest.main()

```

Listing 1: Testfall in Python




Donnerstag 15:14

An: ticket i/O [Mehr anzeigen](#)

Guten Tag,

ich würde gerne von meinen Recht gebrauch machen und rechtmäßig meine Daten löschen lassen.
Wie kann ich meine Daten in ihrem Unternehmen löschen lassen

LG



Donnerstag 16:05

Hallo Michele,

vielen Dank für deine Nachricht!

Du kannst deinen Account in unserem Ticketkäuferportal auf <https://my.ticket.io/> ganz einfach selbst löschen.

Gehe dazu oben rechts auf deinen Namen und wähle dann den Punkt "Mein Konto" aus. Dort kannst du dann ganz unten auf den roten Knopf klicken, um dein Konto dauerhaft zu löschen.

Bitte beachte aber:

Sollte für die vorhandenen Daten eine gesetzliche Aufbewahrungspflicht bestehen, bedeutet die Löschung des persönlichen Kontos nicht, dass sämtliche Daten aus unserem System dauerhaft gelöscht werden.

Die Löschung erfolgt automatisch nach Ablauf der gesetzlichen Aufbewahrungsfristen. Eine Weitergabe personenbezogener Daten von Testtermin-Buchungen an unbeteiligte Dritte ist grundsätzlich ausgeschlossen.

Wenn du also beispielsweise über dieselbe Mailadresse wieder einen Testtermin buchst, wirst du auch die Daten zu vergangenen Buchungen in deinem persönlichen Bereich wiederfinden, sofern sie der gesetzlichen Aufbewahrungspflicht unterliegen.

Abbildung 7: Ein Screenshot eines gelösten Support-Falles