

15. NOVEMBER 2023

Projektdokumentation

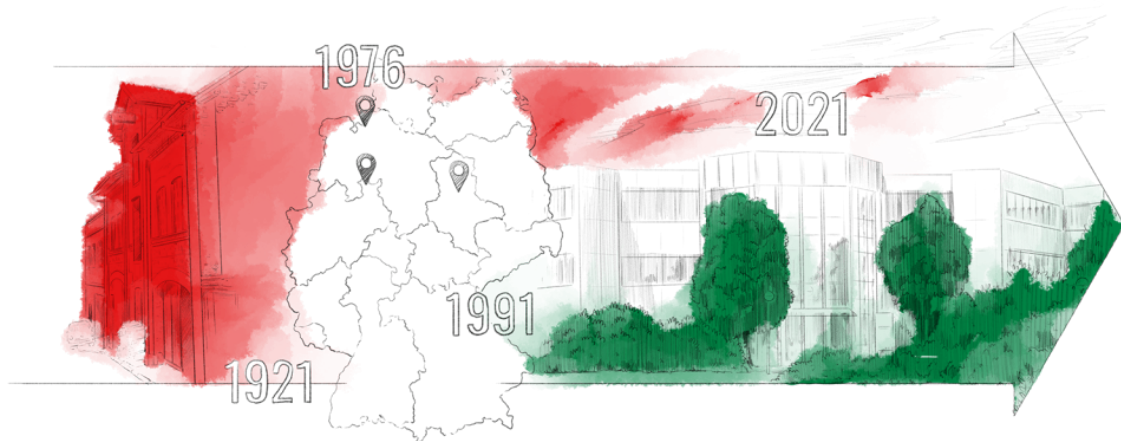
Projektarbeit Winter 2024/25

Fachinformatiker Anwendungsentwicklung

AiConnect

Planung und Entwicklung einer RESTful-Schnittstelle zur Kommunikation mit einer firmenintern KI-Instanz und Bereitstellung einer Entwicklungs- und Laufzeitumgebung

Projektzeitraum: 30.09.2024 – 15.11.2024
Prüfungsnummer: 5130
Projektbetreuer: Dr. Christian Büschking



JAN-HENRIK CAPSIUS
BRÜDER SCHLAU GMBH & CO. KG
Oehrkestraße 1, 32457 Porta Westfalica

Inhaltsverzeichnis

1	Einleitung	1
1.1	Projektbeschreibung	1
1.2	Projektumfeld	1
1.3	Projektziel	1
1.4	Projektbegründung	1
1.5	Projektschnittstellen	2
1.6	Projektabgrenzung	2
2	Projektplanung	3
2.1	Projektphasen	3
2.2	Ressourcenplanung	3
2.3	Entwicklungsprozess	3
3	Analysephase	4
3.1	Ist-Analyse	4
3.2	Wirtschaftlichkeitsanalyse	4
3.2.1	Make or Buy	4
3.2.2	Projektkosten	5
3.2.3	Amortisationsdauer	5
3.3	Nicht-monetäre Vorteile	6
3.4	Lastenheft	6
4	Entwurfsphase	6
4.1	Auswahl Programmiersprachen der Schnittstelle	6
4.1.1	Python	7
4.1.2	Java	7
4.1.3	Schlussfolgerung	7
4.2	Backend	8
4.3	Entwicklungs- und Laufzeitumgebung	8
4.4	Architekturdesign	9
4.4.1	Model	9
4.4.2	Controller	9
4.5	Maßnahmen zur Qualitätssicherung	9
5	Implementierungsphase	10

5.1	Erstellung Docker-Entwicklungs- und Laufzeitumgebung	10
5.2	Erstellung von JUnit-Test.....	10
5.3	Implementierung der Schnittstelle	10
5.3.1	Implementierung der Datenstruktur	10
5.3.2	Implementierung des Controllers und Mappings	11
5.3.3	Implementierung der Geschäftslogik	11
5.4	Abnahme	12
5.5	Dokumentation	12
6	Fazit	13
6.1	Soll-/ Ist-Vergleich.....	13
6.2	Lessons Learned	13
6.3	Ausblick	13
7	Quellenverzeichnis.....	14
Anhang	i
A1	Detaillierte Zeitplanung.....	i
A2	Ressourcenliste.....	ii
A3	Lastenheft (Ausschnitt)	iii
A4	Deploykonfiguration.....	iv
A5	Auszug Docker-Compose Dev	v
A6	JavaToJSONMapperTests.....	v
A6	ContextDTO	vi
A7	ConversationChatcontroller.....	vi
A8	Klassendiagramm.....	vii
A9	Schnittstellendokumentation	viii

Abbildungsverzeichnis

Abbildung I: Detaillierte Zeitplanung.....	i
Abbildung II: Lastenheft (Ausschnitt).....	iii
Abbildung III: Deploykonfiguration.....	iv
Abbildung IV; Auszug Docker-Compose Dev	v
Abbildung V: JavaToJSONMapperTests	v
Abbildung VI: ContextDTO	vi
Abbildung VII:ConversationChatcontroller	vi
Abbildung VIII: Klassendiagramm	vii
Abbildung IX: Swagger-UI Übersicht.....	viii
Abbildung X: Single Chat Controller.....	viii
Abbildung XI: Mock Chat Controller	ix
Abbildung XII: Conversation Chat Controller.....	ix
Abbildung XIII: Server Status Controller.....	x
Abbildung XIV: DTO Schemas	x

Tabellenverzeichnis

Tabelle 1: Grobe Zeitplanung	3
Tabelle 2: Kostenberechnung.....	5
Tabelle 3: Ressourcenliste	ii

Formelverzeichnis

Formel 1: Stundensatz Autor.....	5
Formel 2: Amortisationsdauer	5

1 Einleitung

Die folgende Projektdokumentation schildert den Ablauf des IHK-Abschlussprojektes, welches der Autor im Rahmen seiner Ausbildung zum Fachinformatiker mit der Fachrichtung Anwendungsentwicklung durchgeführt hat. Der Ausbildungsbetrieb ist die Brüder Schlau GmbH & Co. KG, ein deutscher Handelskonzern mit Standort in Porta Westfalica.

Aktuell beschäftigt die Brüder Schlau GmbH & Co. KG über 5.500 Mitarbeiter in mehr als 180 Hammer Fachmärkten und 60 Schlau Handwerkermärkten. Durch die Vertriebslinien Hammer (Einzelhandel) und Schlau (Großhandel) bietet die Brüder Schlau GmbH & Co. KG ein breites Sortiment an Produkten für Boden, Wand, Stoffe, Gardinen und Wohnraumgestaltung so wie handwerkliche Dienstleistungen an.

1.1 Projektbeschreibung

Als Groß- und Einzelhandel und Anbieter von handwerklichen Dienstleistungen erhalten die verschiedenen Supportabteilungen der BS täglich zahlreiche Anfragen von Privatkunden und Fachgeschäften bezüglich der Ware und Dienstleistungen. Die Anfragen werden über ein Kontaktformular automatisiert an die angegebenen Leistungsabteilungen in Form eines Tickets weitergeleitet. Im Anschluss werden die Tickets von dem Team-Lead zugewiesen und manuell von den Mitarbeitern bearbeitet. Die manuelle Erstellung von Antworten zu komplexen Anfragen ist sehr zeitaufwendig, ineffizient und nicht mehr zeitgemäß.

Daher sollen im Zuge der geplanten Digitalisierung des Unternehmens die Mitarbeiter über eine Weboberfläche mit der firmeninternen KI-Instanz kommunizieren können, um den Bearbeitungsprozess der Anfragen mit gleichbleibender Qualität effizienter zu gestalten.

1.2 Projektumfeld

Das Projekt wird im IT-Bereich der Brüder Schlau GmbH & Co. KG durchgeführt. Die IT-Abteilung ist unter anderem für technische Lösungen in unseren Hammer- und Schlau-Märkten zuständig, setzt technische Anforderungen für unsere Online-Präsenzen im Groß- und Einzelhandel um und optimiert sowie automatisiert interne Prozesse. Der Kundensupport übernimmt die Rolle des Auftraggebers und ist für die Festlegung der Anforderungen sowie die Rückmeldungen bezüglich dieser verantwortlich.

1.3 Projektziel

Das Ziel dieses Projekts ist es, den bestehenden Workflow durch den Einsatz von Künstlicher Intelligenz zu optimieren, um den Mitarbeitern die Arbeit zu erleichtern, ohne die Qualität zu

beeinträchtigen. Dabei soll es den Mitarbeitern ermöglicht werden, über eine Weboberfläche mit der firmeninternen KI-Instanz zu interagieren. Dies kann entweder durch einfache Anfragen mit direkten Antworten geschehen oder in Form eines Chatverlaufs, bei dem die künstliche Intelligenz auf vorherige Nachrichten eingeht.

1.4 Projektbegründung

Das Projekt zielt darauf ab, die Bearbeitung komplexer Anfragen effizienter zu gestalten. Mithilfe von Künstlicher Intelligenz soll der Zugriff auf das interne Unternehmenswissen erleichtert werden. Informationen können schneller und einfacher abgerufen werden, da Mitarbeiter nicht mehr mühsam zahlreiche Dokumentationsseiten durchsuchen müssen, um die relevanten Daten zu finden. Dies soll nicht nur die Antwortzeiten bei gleichbleibender Qualität verkürzen, sondern auch die Nutzererfahrung der Mitarbeiter verbessern.

1.5 Projektschnittstellen

Da es sich um ein neues und eigenständiges Projekt handelt, gibt es keine Schnittstellen zu anderen firmeninternen Projekten.

Die Benutzeroberfläche soll jedoch in die Confluence-Landschaft des Unternehmens eingebunden werden. Somit steht die Anwendung automatisch jedem Mitarbeiter mit den entsprechenden Berechtigungsrollen zu Verfügung.

Auf kommunikativer Ebene findet der Austausch mit den Team-Leads der einzelnen Kundenservice-Abteilungen statt, welche als Sprachrohr für die Mitarbeiter fungieren und Rückfragen an diese kommunizieren. Zusammen mit den Projektbetreuern und den Auftraggebern werden die Anforderungen formuliert und Fragen, sowie zusätzliche oder zu streichende Anforderungen untereinander geklärt und für zukünftige Erweiterungen festgehalten.

1.6 Projektabgrenzung

Durch den begrenzten Zeitumfang von 80 Stunden, wurden bestimmte Eingrenzungen getroffen und nur die folgenden Punkte sind Teil des Projektes:

- Planung des Aufbaus der Schnittstelle unter Berücksichtigung der möglichen Programmiersprachen
- Erstellung von Unit-Tests
- die Implementierung der Schnittstellenlogik
- das Bereitstellen einer Entwicklungs- und Laufzeitumgebung in Docker-Containern zur Vereinheitlichung
- das Erstellen einer Schnittstellendokumentation mit Swagger

2 Projektplanung

2.1 Projektphasen

Für das gesamte Projekt stehen insgesamt 80 Stunden zur Verfügung. Zur Einhaltung des Zeitrahmens wurde das Projekt in verschiedene Phasen unterteilt, die sich am erweiterten Wasserfallmodell orientieren. Dieses Modell strukturiert das Projekt in klare Phasen und ermöglicht es dennoch, bei Bedarf zu vorherigen Phasen zurückzukehren, falls sich Anforderungen ändern oder Änderungswünsche auftreten. Der grobe Zeitplan ist in *Tabelle 1: Grobe Zeitplanung* dargestellt, während die detaillierte Zeitplanung, die aus kleineren Teilschritten besteht, in *Abbildung I: Detaillierte Zeitplanung Seite i* zu finden ist.

Phase	Zeit in Stunden
Definitionsphase	5
Planungsphase	7
Durchführungsphase	54
Projektabschluss	14
Gesamt	80

Tabelle 1: Grobe Zeitplanung

2.2 Ressourcenplanung

Mittels Ressourcenplanung wurden die für das Projekt „AiConnect“ erforderlichen Hardware-, Software- und Personalmittel bestimmt. Da die notwendige Hardware, sowie alle erforderlichen Anwendungen inklusive Lizenzen für die Entwicklung und Implementierung der RESTful-Schnittstelle bereits vorhanden waren, war eine zusätzliche Beschaffung nicht notwendig.

Die für das Projekt verwendeten Ressourcen sind in **Error! Reference source not found.** Seite **Error! Bookmark not defined.** gelistet.

2.3 Entwicklungsprozess

Für das Projekt „AiConnect“ wurde die agile Softwareentwicklung als Vorgehensmodell gewählt. Diese Vorgehensweise ermöglicht es, flexibel auf veränderte Anforderungen oder Änderungswünsche einzugehen, da sie auf wenig bürokratischen Aufwand setzt und eine enge Zusammenarbeit zwischen Entwicklern und Projektverantwortlichen fördert. Der Entwicklungsprozess erfolgte in iterativen Zyklen, wobei am Ende jeder Iteration ein nutzbares Zwischenprodukt vorliegen sollte. Dies ermöglichte regelmäßige Rücksprachen mit den Projektverantwortlichen, um sicherzustellen, dass die Entwicklung stets den Anforderungen entsprach.

Aufgrund des kleinen Projektumfangs von 80 Stunden fand täglich ein kurzes, 10-minütiges Meeting statt. In diesem präsentierte der Autor den aktuellen Projektfortschritt, erhielt direktes Feedback vom Projektverantwortlichen und besprach die nächsten Aufgaben.

3 Analysephase

3.1 Ist-Analyse

Wie in 1.1 beschrieben, erhalten die Supportabteilungen der Brüder Schlau GmbH & Co. KG täglich zahlreiche Anfragen, die derzeit über ein Kontaktformular in Form von Tickets an die zuständigen Abteilungen weitergeleitet werden. Die Bearbeitung dieser Tickets erfolgt manuell und bringt folgende Herausforderungen mit sich:

- **Zeitaufwendige Bearbeitung:** Die manuelle Bearbeitung der Anfragen ist sehr zeitintensiv. Gerade bei komplexen Anfragen muss viel in der firmeninternen Wissensdatenbank recherchiert werden, um eine Antwort zu erstellen, was die Bearbeitungsdauer verlängert und die Effizienz einschränkt.
- **Erhöhter Kommunikationsaufwand:** Absprachen zwischen Mitarbeitern und Abteilungen zur Klärung der Tickets sind zum Teil notwendig, was zu zusätzlichem bürokratischem Aufwand führt.
- **Qualitätsabweichungen:** Die Qualität der Antworten variiert aufgrund des individuellen Bearbeitungsstils der Mitarbeiter und ist schwer standardisierbar.

3.2 Wirtschaftlichkeitsanalyse

Die folgende Wirtschaftlichkeitsanalyse veranschaulicht die finanziellen Einsparungen und Vorteile, die durch die Lösung der in 3.1 *Ist-Analyse* genannten Herausforderungen erreicht werden können.

3.2.1 Make or Buy

Der Autor hat die Möglichkeit, die Spezifikation zu entwickeln, ohne dabei zusätzliche interne Kosten zu verursachen. Daher wurde die Entscheidung getroffen, die Entwicklung intern durchzuführen, anstatt externe Ressourcen oder Dienstleistungen in Anspruch zu nehmen. Diese Entscheidung basiert auf der Überlegung, dass die interne Entwicklung nicht nur kosteneffizienter ist, sondern auch eine bessere Kontrolle über den Entwicklungsprozess und die Qualität der Spezifikation ermöglicht. Durch die interne Umsetzung kann der Autor zudem sicherstellen, dass alle Anforderungen und Spezifikationen genau eingehalten werden, ohne dass zusätzliche externe Einflüsse oder Risiken entstehen.

3.2.2 Projektkosten

Die Gesamtkosten des Projekts setzen sich aus den Personalkosten des Autors sowie den Kosten der beteiligten Mitarbeiter zusammen. Zudem werden die Nebenkosten für den Arbeitsplatz und die Arbeitsmaterialien berücksichtigt. Diese Informationen sind in *Tabelle 3.2.2-1: Kostenberechnung* aufgeführt. Für den beteiligten technischen Berater und Teamleiter wurde von der Personalabteilung ein durchschnittlicher Stundensatz von 65 € und 50 € angesetzt, Der ungefähre Stundensatz des Autors wurde mit der *Formel 1: Stundensatz Autor* berechnet.

$$\frac{\text{ca. 19.400,00 € Jahresbrutto inkl. Lohnnebenkosten}}{(251d(\text{Arbeitstage 2024 NRW}) - 30d(\text{Urlaub})) \times 7,7h (\text{tägliche Arbeitszeit})} = 11,40\text{€}$$

Formel 1: Stundensatz Autor

Für den Autor fallen 80 Stunden Projektarbeit an. Der zuständige Berater hat etwa 15 Stunden im Rahmen von Meetings, Spezifikationsentwurf, Hilfestellung und Testing in das Projekt investiert. Die Teamleiter der einzelnen Abteilungen haben um die 6 Stunde für das Sammeln von Anforderungen, deren Kommunikation und Rücksprachen aufgewendet. Aus den genannten Stunden und den entsprechenden Stundensätzen ergeben sich die folgenden Kosten:

Mitarbeiter	Stundenanzahl	Stundensatz	Kosten
Autor	80	11,40 €	912,00 €
Technischer Berater	15	65,00 €	975,00 €
Teamleiter Support-Abteilung	8	50,00 €	400,00 €
Gesamt			2.287,00 €

Tabelle 2: Kostenberechnung

3.2.3 Amortisationsdauer

Um festzustellen, ob und wann sich die Anwendung amortisiert haben wird, werden die potenziellen Einsparungen geschätzt. Die verantwortlichen Teamleiter schätzten, dass die zeitlichen Einsparungen in den verschiedenen Supportabteilungen aufgrund der kürzeren Bearbeitungsdauer pro Ticket bei etwa 10 Stunden pro Monat liegen. Bei einem angesetzten Stundensatz von 30 € für die Mitarbeiter in diesen Abteilungen ergibt sich eine Amortisationsdauer von etwa 8 Monaten. Laufende Kosten fallen nicht an, da die ganze Infrastruktur schon besteht. Diese Berechnung basierten auf der *Formel 2: Amortisationsdauer*:

$$\frac{2.287,00 \text{ €}}{10 \text{ Stunden}(\text{Zeitersparnis pro Monat}) \times 30\text{€ Stundensatz}} = 7,62 \approx 8 \text{ Monate}$$

Formel 2: Amortisationsdauer

3.3 Nicht-monetäre Vorteile

Das Projekt zielt in erster Linie darauf ab, die Effizienz im Kundenservice der Brüder Schläu GmbH & Co. KG zu steigern und die Kundenzufriedenheit zu erhöhen. Durch die Entwicklung der RESTful-Schnittstelle AiConnect wird der Prozess der Anfragebearbeitung teilautomatisiert und optimiert. Dies ermöglicht den Mitarbeitern, schneller und präziser auf Anfragen von Privatkunden und Fachgeschäften zu reagieren, was zu einem deutlich angenehmeren Erlebnis für die Kunden und Mitarbeiter führt.

3.4 Lastenheft

Die Analysephase basiert auf einem Lastenheft, das vom Auftraggeber erstellt wurde und in *Abbildung II: Lastenheft (Ausschnitt)* auf Seite iii finden ist. Dieses Lastenheft fungiert als Leitfaden für den Entwickler während der Entwurfsphase. Die Anforderungen an das Produkt aus der Perspektive des Auftraggebers können dem Lastenheft entnommen werden.

4 Entwurfsphase

In der Entwurfsphase wird zunächst die Auswahl der Programmiersprache für die RESTful-Schnittstelle getroffen. Hierbei stehen zwei Programmiersprachen zur Wahl, die hinsichtlich ihrer Eignung für die geplante Anwendung bewertet werden. Anschließend erfolgt die Beschreibung der Zielplattformen sowie des Architekturdesigns des Projekts. Darüber hinaus werden der Entwicklungsablauf, die verschiedenen Entwürfe, Maßnahmen zur Qualitätssicherung und der gesamte Findungsprozess dokumentiert.

4.1 Auswahl Programmiersprachen der Schnittstelle

Die Wahl der Programmiersprache für die Entwicklung der RESTful-Schnittstelle ist ein entscheidender Faktor für den Erfolg des Projekts „AiConnect“. In diesem Abschnitt werden die beiden in Betracht gezogenen Programmiersprachen, Python und Java, hinsichtlich ihrer Vorteile und Eignung für die spezifischen Anforderungen des Projekts verglichen.

4.1.1 Python

Python ist bekannt für seine einfache Syntax und Lesbarkeit und für die Implementierung von Machine Learning-Algorithmen macht. Die Vorteile von Python in diesem Kontext sind:

- **Reiches Ökosystem:** Python verfügt über eine Vielzahl von Bibliotheken und Frameworks (wie Flask oder Django) für die Erstellung von RESTful-Schnittstellen bietet.
- **Integrierbar in die KI-Instanz:** Da die KI-Instanz auf Python basiert, lässt sich die Schnittstelle leicht integrieren.
- **Einfachheit:** Die Lesbarkeit von Python-Code ermöglicht es neuen Entwicklern, schneller produktiv zu werden, was insbesondere bei der Entwicklung neuer Features von Vorteil ist.

4.1.2 Java

Java ist bekannt für seine Plattformunabhängigkeit, da es einmal geschrieben werden kann und auf jeder Plattform ausgeführt werden kann, die eine Java Virtual Machine (JVM) unterstützt. Zudem wird es häufig für die Entwicklung von Unternehmensanwendungen so wie großen Systemen verwendet. Die Argumente für Java sind:

- **Stabilität und Performance:** Java ist bekannt für seine hohe Performance und Stabilität, insbesondere in Unternehmensanwendungen. Der Einsatz von Java kann daher dazu beitragen, eine robuste und skalierbare Schnittstelle zu entwickeln.
- **Verbreitung im Unternehmen:** Da Java bereits die Hauptprogrammiersprache im Unternehmen ist, verfügen die Entwickler über umfangreiche Erfahrungen und Kenntnisse. Dies reduziert die Komplexität und Risiko des Projektes im gegebenen Zeitraum zu scheitern.
- **Starke Typisierung:** Die statische Typisierung von Java reduziert zur Laufzeit mögliche Fehler und führt zu robusterem Code. Dies ist besonders wichtig für eine Schnittstelle, die als zentrale Anlaufstelle für Anfragen fungiert.
- **Erweiterbarkeit:** Java bietet zahlreiche Frameworks wie Spring Boot, die die Entwicklung von RESTful-Schnittstellen unterstützen und die Modularität sowie Wartbarkeit des Codes fördern.

4.1.3 Schlussfolgerung

Obwohl Python viele Vorteile in der schnellen Entwicklung und im Zugriff auf KI-Frameworks bietet, sind die spezifischen Anforderungen der Brüder Schläu GmbH und Co. KG sowie die vorhandenen Ressourcen und Kenntnisse im Unternehmen entscheidend für die Auswahl der Programmiersprache. Die Stabilität, Performance und Integration in die bestehende Systemlandschaft von Java machen es zur idealen Wahl für die Entwicklung der Schnittstelle. Zudem möchten wir die gesamte Applikation möglichst nah an der Micro-Service-Architektur anlehnen.

4.2 Backend

Das Backend besteht aus einem Spring Boot Projekt. Spring Boot ist ein Framework zur Entwicklung von eigenständigen, produktionsbereiten Anwendungen, die auf dem Spring Framework basieren. Es wurde von Pivotal entwickelt und ist darauf ausgelegt, den Einstieg in die Entwicklung mit Spring so einfach wie möglich zu gestalten. Das Framework ermöglicht es Entwicklern, Anwendungen schnell und unkompliziert aufzusetzen, ohne dass umfangreiche XML-Konfigurationen erforderlich sind. Stattdessen setzt Spring Boot auf „Convention over Configuration“ und bietet sinnvolle Voreinstellungen für die wichtigsten Features und Integrationen.

Die grundlegende Programmiersprache von Spring Boot ist Java, was die Plattformunabhängigkeit sichert.

Hauptsächlich wird es für die Entwicklung von Web- und Microservices-Architekturen verwendet. Seine Stärke liegt in der nahtlosen Integration mit Spring-Ökosystem-Komponenten, die Entwicklern Zugang zu einer Vielzahl von Funktionen und Modulen wie Spring Data, Spring Security oder Spring Web bietet.

Zusätzlich unterstützt Spring Boot die Java Enterprise Edition (Java EE), was die Entwicklung moderner Webanwendungen mit transaktionsbasierten Ausführungen vereinfacht. Die Wichtigkeit wird im Ausblick klar.

Anwendungen, die mit Spring Boot entwickelt werden, benötigen keinen separaten Webserver, da ein eingebetteter Webserver wie Apache Tomcat oder Jetty standardmäßig integriert ist. Dies ermöglicht das direkte Ausführen der Anwendung und vereinfacht den Deployment-Prozess.

4.3 Entwicklungs- und Laufzeitumgebung

Als Entwicklungs- und Laufzeitumgebung für das Deployment wird Docker verwendet. Docker ist eine Open-Source-Containerplattform, die entwickelt wurde, um Anwendungen in Containern auszuführen und zu verwalten. Container isolieren Anwendungen und alle ihre Abhängigkeiten in leichtgewichtige, portable Pakete, die auf jedem Betriebssystem mit Docker-Unterstützung konsistent laufen können.

Die Hauptvorteile von Docker liegen in der Portabilität und Skalierbarkeit von Anwendungen. Docker-Container können auf verschiedenen Betriebssystemen und in verschiedenen Umgebungen (lokal, in der Cloud oder in hybriden Umgebungen) konsistent und unabhängig voneinander ausgeführt werden. Das Container-Image-Konzept von Docker ermöglicht es Entwicklern, genau definierte und getestete Umgebungen mit allen erforderlichen Bibliotheken, Abhängigkeiten und Einstellungen bereitzustellen.

4.4 Architekturdesign

Die gesamte Anwendung richtet sich nach dem MVC-Konzept. Das Konzept sieht die Strukturierung der Anwendung in die drei Hauptkomponenten Model, View und Controller vor. Jede dieser Komponenten hat eine spezifische Rolle und Verantwortung, was zu einer klaren Trennung der Anliegen führt und die Wartbarkeit und Erweiterbarkeit der Anwendung verbessert.

Da das Frontend nicht zum Umfang dieses Projekts gehört, wird die View-Komponente im Folgenden nicht behandelt.

4.4.1 Model

Das Model repräsentiert die Daten und die Geschäftslogik der Anwendung. Es ist verantwortlich für die Verwaltung der eingehenden, ausgehenden und für die Geschäftslogik relevanten Datenpakete, die Verarbeitung der Geschäftslogik und die Kommunikation mit einer Datenbank oder wie in diesem Fall mit der KI-Instanz.

4.4.2 Controller

Die Hauptaufgabe der Controller besteht darin, API-Endpunkte bereitzustellen, über die das Frontend mit dem Backend kommunizieren kann. Derzeit verwalten die Controller die POST-Methode, die notwendig ist, um Daten vom Frontend zu empfangen. Zusätzlich sind sie dafür verantwortlich, die Geschäftslogik zu starten und die Antwort in geeigneter Form wieder an das Frontend zurückzugeben.

4.5 Maßnahmen zur Qualitätssicherung

Um die Qualität sicherzustellen, ist eine Testphase geplant. In dieser Phase erstellt der technische Berater verschiedene Testfälle, anhand derer er die korrekte Funktionalität des Systems prüft und bestätigt. Entdeckt der Berater dabei Fehler, werden diese anschließend behoben.

Zusätzlich werden für die Logik der Service-Klassen JUnit-Tests entwickelt.

5 Implementierungsphase

5.1 Erstellung Docker-Entwicklungs- und Laufzeitumgebung

Um eine standardisierte und portable Entwicklungsumgebung für das Spring Boot-Projekt bereitzustellen, wurde eine Docker-Umgebung eingerichtet. Diese ermöglicht eine reibungslose Bereitstellung und Versionierung der Anwendung, eine unkomplizierte Kollaboration im Entwicklerteam sowie Deployment.

Hierfür wurden Dockerfiles für die Entwicklungs -und Laufzeitumgebung erstellt, die alle benötigten Konfigurationen und Abhängigkeiten definieren. Der Aufbau der Deploykonfiguration ist in *Abbildung III: Deploykonfiguration Seite iv* zu finden.

Zudem wurde das Docker-Compose-Tool genutzt, um Container wie das Backend selbst sowie zusätzliche Dienste wie das Frontend und die KI-Instanz gemeinsam starten und verwalten zu können.

Die Einrichtung der Docker-Compose-Konfiguration erfolgte durch die Implementierung der im Docker-Ökosystem integrierten YAML-Dateien. Diese beinhalten spezifische Informationen zur Anwendung, wie den Port-Mapping und die Verlinkung zwischen Spring Boot-Container, Frontend und der KI-Instanz. Diese Konfigurationsdetails sind in *Abbildung IV; Auszug Docker-Compose Dev v* zu sehen

5.2 Erstellung von JUnit-Test

In diesem Projekt wird der testgetriebenen Entwicklungsansatz (TDD) verfolgt. Dabei liegt der Schwerpunkt auf dem Verfassen von Tests, bevor der dazugehörige Code geschrieben wird. Das Hauptziel ist es, die Softwarequalität von Beginn an zu garantieren, da die Tests jederzeit den Code abdecken. Ein Auszug von den Tests ist in der *Abbildung V: JavaToJSONMapperTests* auf *Seite v* dargestellt.

5.3 Implementierung der Schnittstelle

Im Folgenden wir die Implementierung der einen Schnittstellenkomponenten beschrieben.

5.3.1 Implementierung der Datenstruktur

Um die Kommunikation zwischen Frontend, Backend und der KI-Instanz zu ermöglichen werden die Datenpakete, wie in *4.4.1 Model* beschrieben, als statische Datenstrukturen abgebildet. Im Java-Umfeld werden die Strukturen Records genannt. Records eignen sich besonders gut für diesen Zweck, da sie unveränderlich(immutable) sind und somit nur gelesen,

aber nicht verändert werden können. Dies gewährleistet eine hohe Sicherheit vor Datenmanipulation und bietet gleichzeitig Vorhersehbarkeit der Datensätze beim Testen.

In der *Abbildung VI: ContextDTO* auf *Seite vi* ist stellvertretend für das Projekt der ContextDTO zu sehen. Dieses Datenpaket ist dafür zuständig Nachrichten an die KI-Instanz weiterzuleiten.

5.3.2 Implementierung des Controllers und Mappings

In einer Spring Boot-Anwendung fungieren Controller wie der in *Abbildung VII:ConversationChatcontroller* (s. *Seite vi*) als API-Endpoints, um den Zugriff vom Frontend auf das Backend zu ermöglichen und um den Prozess der Geschäftslogik anzustoßen.

Hierfür wird die Klasse mit mehreren Annotationen versehen um diese als REST-Controller zu definieren damit HTTP-Anfragen verarbeitet werden können und die Basis-URL der Endpoints zu definieren. Anschließend werden die Routen der einzelnen CRUD-Operationen definiert und bei Aufruf die Geschäftslogik angestoßen und das Ergebnis zurückgegeben.

5.3.3 Implementierung der Geschäftslogik

Die Grundlage der Geschäftslogik bildet das in *Abbildung VIII: Klassendiagramm* auf *Seite vii* dargestellte Klassendiagramm. Das Interface *IAiContext* dient als Einstiegspunkt, über den die Geschäftslogik mithilfe der Controller gestartet wird. Es wird zwischen drei verschiedenen Chatmöglichkeiten unterschieden:

- **MockChat:** Diese Klasse enthält keine relevante Geschäftslogik und dient lediglich dazu, die Frontend-Entwicklung mit vordefinierten Rückgabewerten zu erleichtern.
- **AiSingleChat:** Diese Klasse wird verwendet, um eine einzelne Frage an die KI-Instanz zu senden. Nach Abschluss der Operation wird der Kontext geleert, um die Bearbeitungsdauer der Frage möglichst kurz zu halten.
- **AiConversationChat:** Diese Klasse nutzt den gegebenen Kontext vollständig aus und speichert die Fragen und Antworten, um auf den Verlauf reagieren zu können.

Die Methode *askAi* bildet das Grundgerüst der Geschäftslogik und besteht aus folgenden Schritten:

1. Die Anfrage wird entgegengenommen und dem Kontext hinzugefügt. Dieser wird von der *Klasse AiContext* verwaltet, die das Hinzufügen, Löschen und Zusammenfassen des Kontextes übernimmt.
2. Mithilfe der *Klasse JavaToJSONMapper* wird der Kontext in ein String-Objekt umgewandelt, das ein JSON-Objekt widerspiegelt. Dies ist wichtig, da die KI-Instanz ihre Anfragen im JSON-Format entgegennimmt.

3. Die Klasse *AiRequester* erstellt mithilfe des Kontext-Strings und weiteren Parametern eine Anfrage.
4. Die erstellte Anfrage wird mithilfe der Methode *callAPI* und der Klasse *AiConnector* an die KI-Instanz gesendet und der Antwort-String wird an die Chatklasse weitergeleitet.
5. Die Antwort wird mithilfe der Klasse *AiResponseMapper* in Java-Datenklassen konvertiert.
6. Die Antwort wird aus der Datenklasse extrahiert und an den Controller zurückgeschickt.

Wichtig ist, dass die Mapper sowie der *AiConnector* als Singletons implementiert wurden, da nur ein Mapper benötigt wird und unnötig viele Verbindungen zur KI-Instanz vermieden werden sollen.

5.4 Abnahme

Nach der abschließenden Code-Review, die der technische Berater gemeinsam mit dem Autor durchgeführt hat und einem umfassenden Funktionstest des Systems wurde die Integration der Schnittstelle in das Gesamtprojekt freigegeben.

5.5 Dokumentation

Diese Projektdokumentation behandelt die einzelnen Phasen und Schritte, die im Verlauf des Projekts durchlaufen wurden und ist selbst ein Ergebnis dieses Prozesses. Viele der Anhänge entstanden bereits während der Projektphasen, da sie integraler Bestandteil der jeweiligen Abschnitte waren.

Ein weiteres Projektergebnis ist die Schnittstellendokumentation mittels Swagger. Sie unterstützt das Frontend-Team dabei, die korrekten Parameter an das Backend zu übermitteln und die gewünschten Attribute als Antwort zu erhalten. Die Schnittstellendokumentation befindet sich im Anhang auf Seite x *Schnittstellendokumentation*.

6 Fazit

6.1 Soll-/ Ist-Vergleich

Wenn der Autor das Projekt Revue passieren lässt, ist festzustellen, dass das Projekt in seiner Gesamtheit umgesetzt wurde. Der vorgegebene Zeitplan von 80 Stunden wurde eingehalten. Die Zeiten konnten durchweg eingehalten werden und wurden nicht verschoben.

6.2 Lessons Learned

Durch die vielfältigen Planungs- und Entwicklungstätigkeiten bot das Projekt eine hervorragende Möglichkeit, die im Rahmen der Ausbildung erlernten Methoden zur Softwareplanung und -entwicklung praxisnah zu vertiefen. Dabei wurde nochmals deutlich, wie wichtig die regelmäßige Kommunikation mit den Stakeholdern ist, da ein zufriedenstellendes Endprodukt ohne deren Einbindung kaum erreichbar ist.

Der projektbegleitende Einsatz des erweiterten Wasserfall-Modells und der testgetriebenen Entwicklung entsprach den Vorstellungen des Autors und er beabsichtigt, dieses Vorgehen auch in zukünftigen Projekten beizubehalten. Sowohl der Autor als auch sein Ausbilder bewerten das Projekt als vollen Erfolg. Es bringt nicht nur der Unternehmensgruppe einen Mehrwert, sondern war auch für die persönliche und fachliche Weiterentwicklung des Autors von großer Bedeutung.

6.3 Ausblick

Wie in *1.6 Projektabgrenzung* erwähnt, ist die Schnittstelle nur ein Bestandteil des gesamten Projekts. Künftig soll das Frontend in Abstimmung mit den Team-Leads der Support- und Service-Abteilungen gestaltet und umgesetzt sowie in die Confluence-Umgebung integriert werden. Geplante Zusatzfunktionen umfassen unter anderem die Speicherung von Konversationen in einer Datenbank, um diese später erneut abrufen zu können.

7 Quellenverzeichnis

HTTP: https://de.wikipedia.org/wiki/Hypertext_Transfer_Protocol (Abgerufen am 20.05.2022)

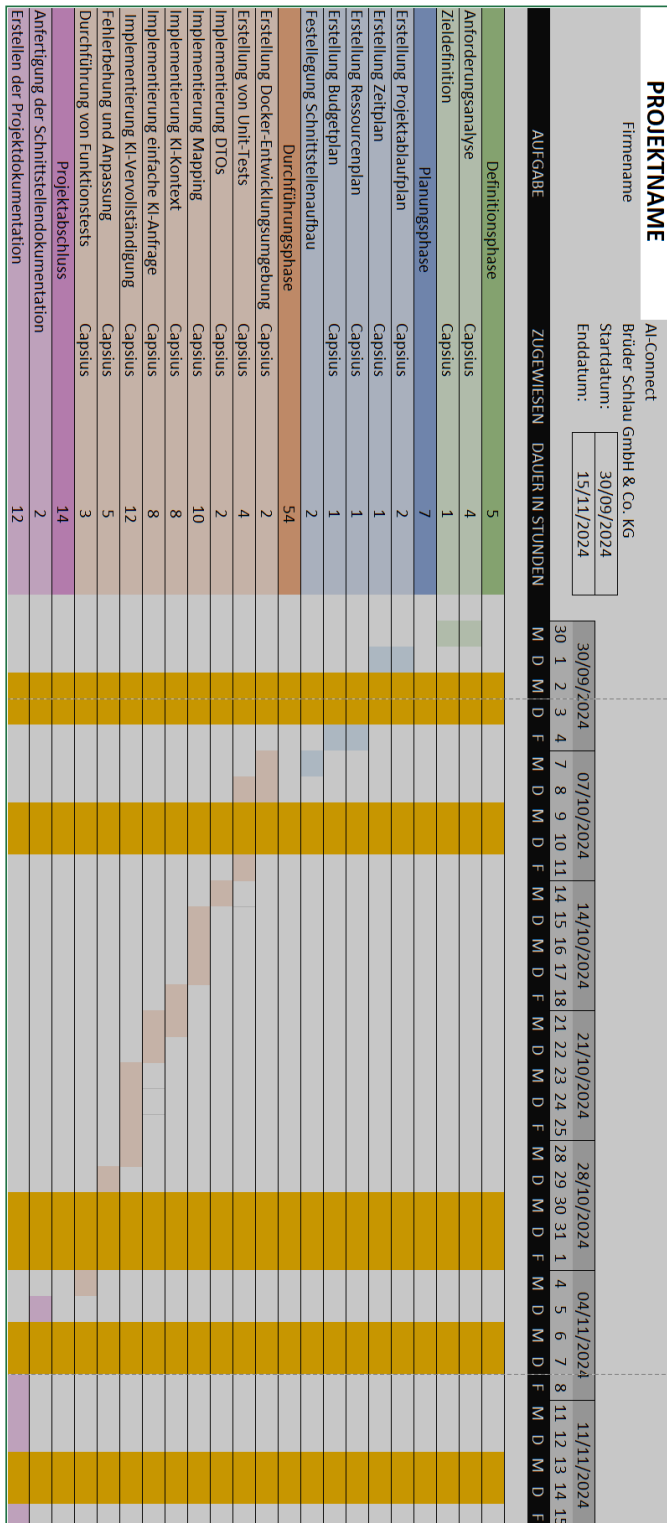
URL: https://de.wikipedia.org/wiki/Uniform_Resource_Locator (Abgerufen am 20.05.2022)

Erweitertes Wasserfallmodell: <https://de-academic.com/dic.nsf/dewiki/407012> (Abgerufen am 20.05.2022)

MVC: https://de.wikipedia.org/wiki/Model_View_Controller (Abgerufen am 20.05.2022)

Amortisation: <https://sevdesk.de/lexikon/amortisationsrechnung> (Abgerufen am 20.05.2022)

A1 Detaillierte Zeitplanung



A2 Ressourcenliste

Ressourcengruppe	Ressource
Hardware	Docking-Station
	Zwei 34" Bildschirme
	Macbook Pro
Software	Betriebssystem: MacOS 15 Sequioa
	Entwicklungsumgebung: Visual Studio Code
	Interne Dokumentation: Confluence
	Diagrammmodellierung: <ul style="list-style-type: none">• draw.io• Excel
	Versionsverwaltung: Git
	Textverarbeitung: Word 365
	Ticketsystem und Projektmanagement: Jira
	Interne Kommunikation Microsoft Teams und Outlook
	Screenshots: macOS Screenshot-Funktion
Container	Docker
Programmiersprache	Java 17
Framework	Spring Boot 3.3.5
	OkHttp3 4.12.0
	Lombok 1.18.34
Personal	Autor: Planung, Durchführung und Dokumentation des Projekts
	Berater: Meetings, Spezifikationsentwurf, Hilfestellung und Testing
	Teamleiter: Sprachrohr zwischen den Abteilungen, Anforderungsdefinition

Tabelle 3: Ressourcenliste

A3 Lastenheft (Ausschnitt)

2. Produkteinsatz

Das Produkt soll als Teil einer großen Anwendung im Rahmen der Digitalisierung des Unternehmens in den Support-Abteilungen eingesetzt werden.

Benutzergruppen:

- Mitarbeiter der Support-Abteilungen

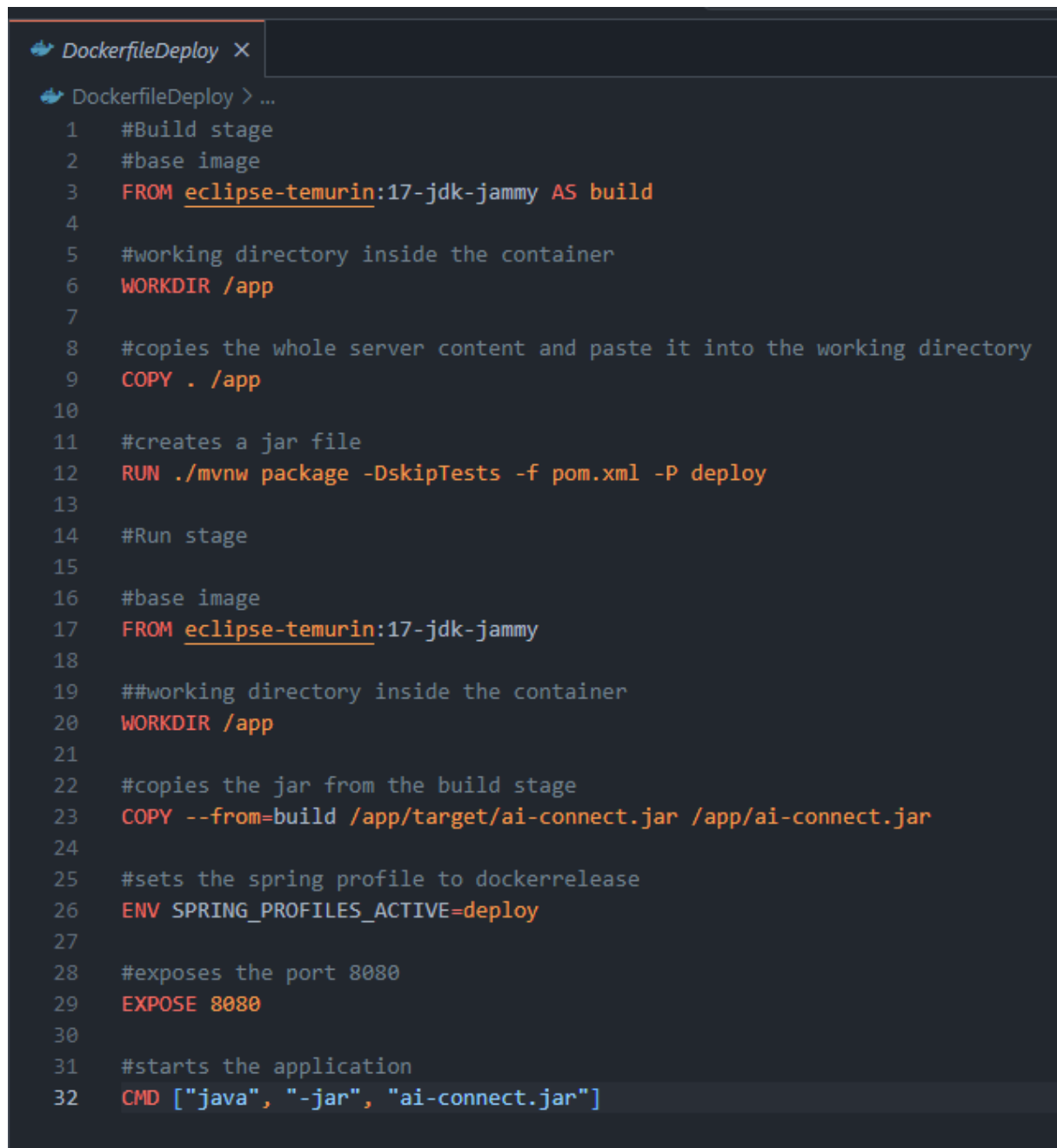
3. Produktfunktionen

Pflichtfunktionen

- Erstellen einer einzelnen Nachricht ohne Kontext
- Erstellen eines Nachrichtenverlaufs mit Kontext

Abbildung II: Lastenheft (Ausschnitt)

A4 Deploykonfiguration



```
DockerfileDeploy X
DockerfileDeploy > ...
1  #Build stage
2  #base image
3  FROM eclipse-temurin:17-jdk-jammy AS build
4
5  #working directory inside the container
6  WORKDIR /app
7
8  #copies the whole server content and paste it into the working directory
9  COPY . /app
10
11 #creates a jar file
12 RUN ./mvnw package -DskipTests -f pom.xml -P deploy
13
14 #Run stage
15
16 #base image
17 FROM eclipse-temurin:17-jdk-jammy
18
19 ##working directory inside the container
20 WORKDIR /app
21
22 #copies the jar from the build stage
23 COPY --from=build /app/target/ai-connect.jar /app/ai-connect.jar
24
25 #sets the spring profile to dockerrelease
26 ENV SPRING_PROFILES_ACTIVE=deploy
27
28 #exposes the port 8080
29 EXPOSE 8080
30
31 #starts the application
32 CMD ["java", "-jar", "ai-connect.jar"]
```

Abbildung III: Deploykonfiguration

A5 Auszug Docker-Compose Dev

```
ai-connect:
  container_name: ai-connect
  build:
    context: .
    dockerfile: DockerfileDev
  ports:
    - "8080:8080"
  volumes:
    - ./ai-connect/main/java/com/bs/ai-connect:/usr/src/app
    - ./ai-connect/main/resources:/usr/src/app/resources
  networks:
    - frontend-aiconnect
    - aiconnect-aiinstance
```

Abbildung IV; Auszug Docker-Compose Dev

A6 JavaToJSONMapperTests

```
@SpringBootTest
public class JavaToJSONMapperTests {

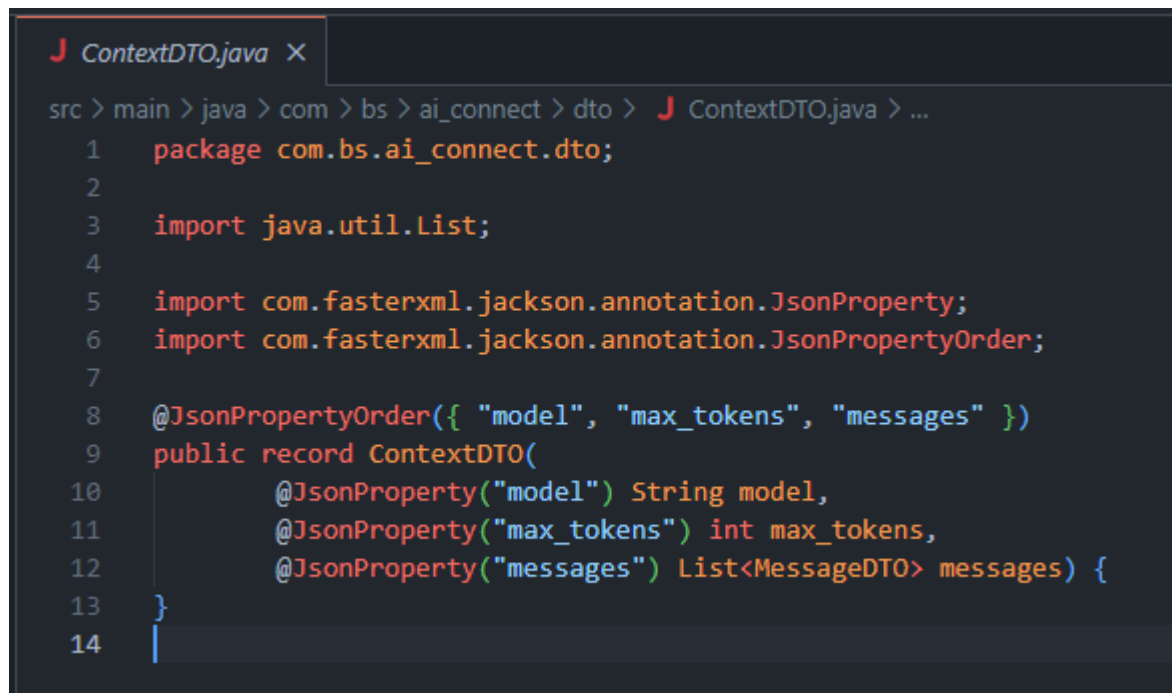
    @Test
    public void testGetInstance(){
        assertNotNull(JavaToJSONMapper.getInstance());
    }

    @Test
    public void testGetInstance_Singleton(){
        assertEquals(JavaToJSONMapper.getInstance(), JavaToJSONMapper.getInstance());
    }

    @Test
    public void testMapResponse(){
        List<MessageDTO> messages = List.of(new MessageDTO(role:"system",content:"Hello from AI"));
        String comparable = "{\"model\":\"gpt-3.5-turbo\",\"max_tokens\":1024,\"messages\":[{\"role\":\"system\",\"content\":\"Hello from AI\"}]}";
        ContextDTO testContext = new ContextDTO(model:"gpt-3.5-turbo",max_tokens:1024, messages);
        String contextToJSON;
        try {
            contextToJSON = JavaToJSONMapper.mapJSON(testContext);
            assertEquals(comparable,contextToJSON);
        } catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }
}
```

Abbildung V: JavaToJSONMapperTests


A6 ContextDTO



```
J ContextDTO.java X
src > main > java > com > bs > ai_connect > dto > J ContextDTO.java > ...
1  package com.bs.ai_connect.dto;
2
3  import java.util.List;
4
5  import com.fasterxml.jackson.annotation.JsonProperty;
6  import com.fasterxml.jackson.annotation.JsonPropertyOrder;
7
8  @JsonPropertyOrder({ "model", "max_tokens", "messages" })
9  public record ContextDTO(
10      @JsonProperty("model") String model,
11      @JsonProperty("max_tokens") int max_tokens,
12      @JsonProperty("messages") List<MessageDTO> messages) {
13  }
14
```

Abbildung VI: ContextDTO

A7 ConversationChatcontroller



```
@RestController
@RequestMapping("/api/conversation-chat")
public class ConversationChatController {
    private IAICompletion aiCompletion;

    public ConversationChatController(@Qualifier("aiConversationChat") IAICompletion aiCompletion){
        this.aiCompletion = aiCompletion;
    }

    @PostMapping("/")
    public String postQuestion(@RequestBody QuestionDTO question) {
        String answer = aiCompletion.askAI(question);
        return answer;
    }
}
```

Abbildung VII: ConversationChatcontroller

A8 Klassendiagramm

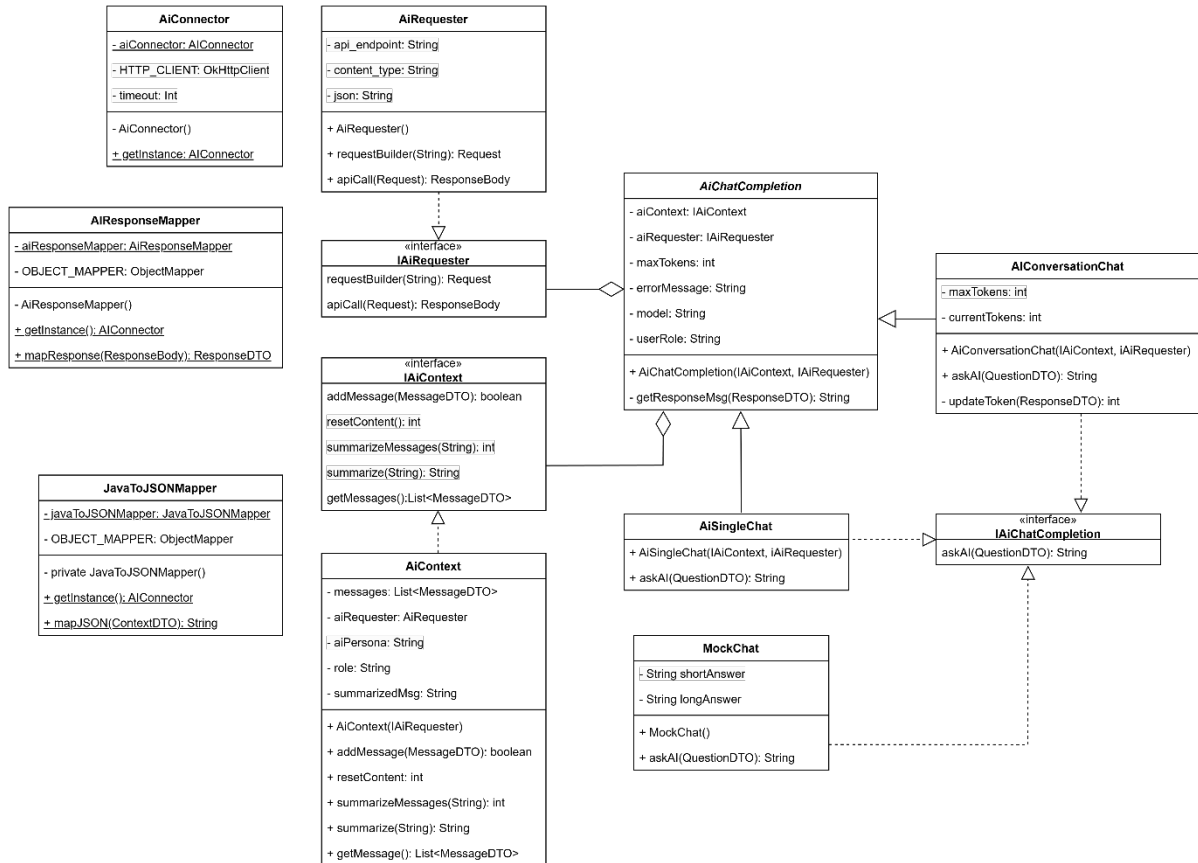


Abbildung VIII: Klassendiagramm

A9 Schnittstellendokumentation

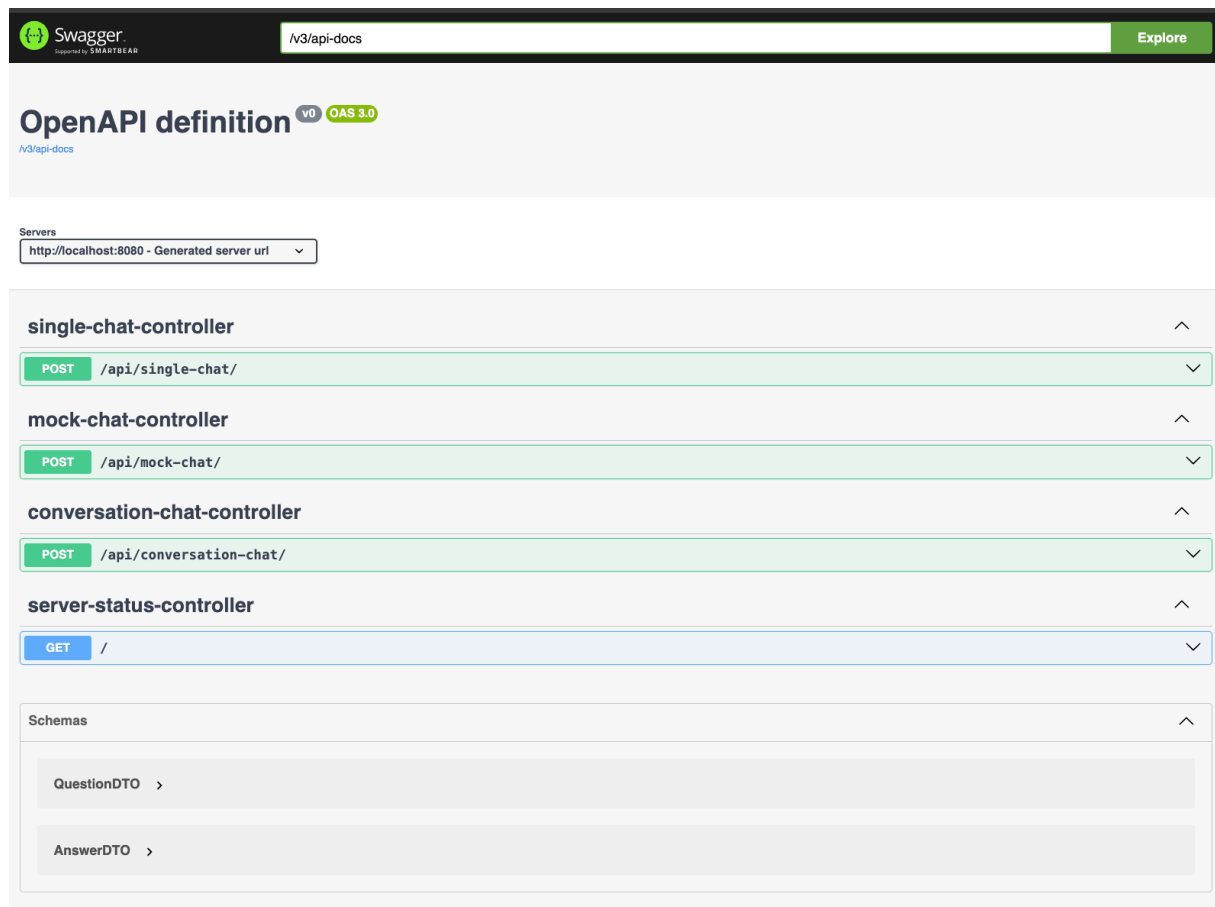


Abbildung IX: Swagger-UI Übersicht

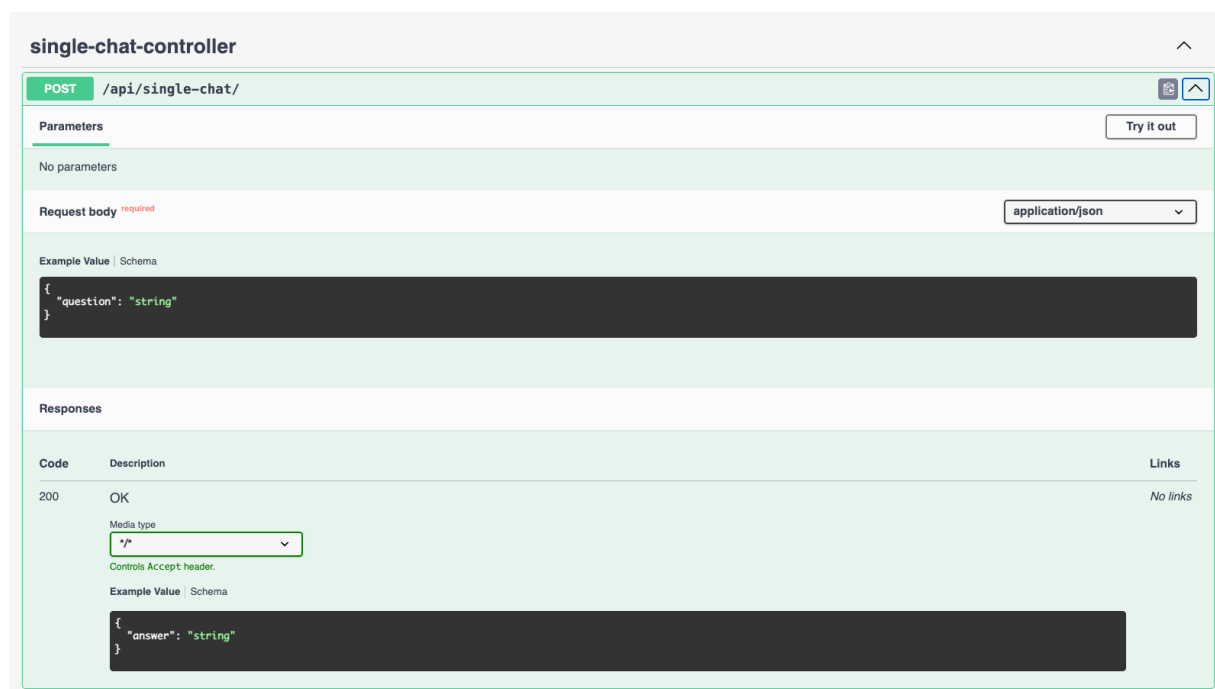


Abbildung X: Single Chat Controller

AiConnect

Planung und Entwicklung einer RESTful-Schnittstelle zur Kommunikation mit einer firmenintern KI-Instanz und Bereitstellung einer Entwicklungs- und Laufzeitumgebung

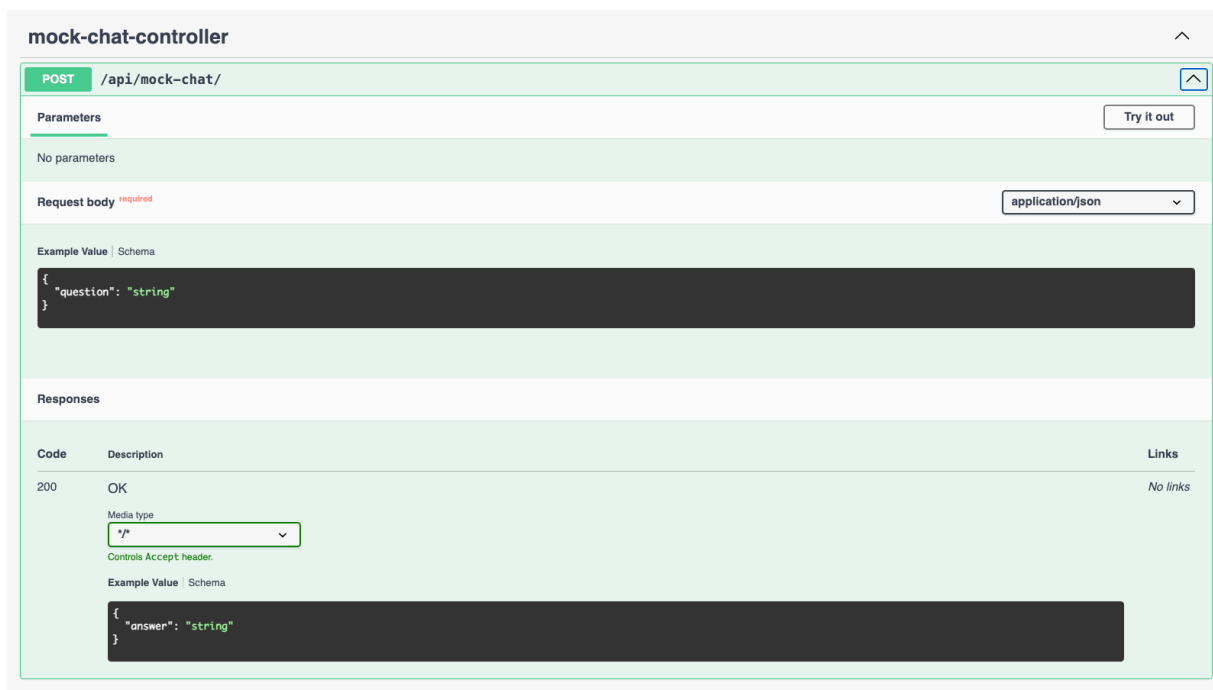


Abbildung XI: Mock Chat Controller

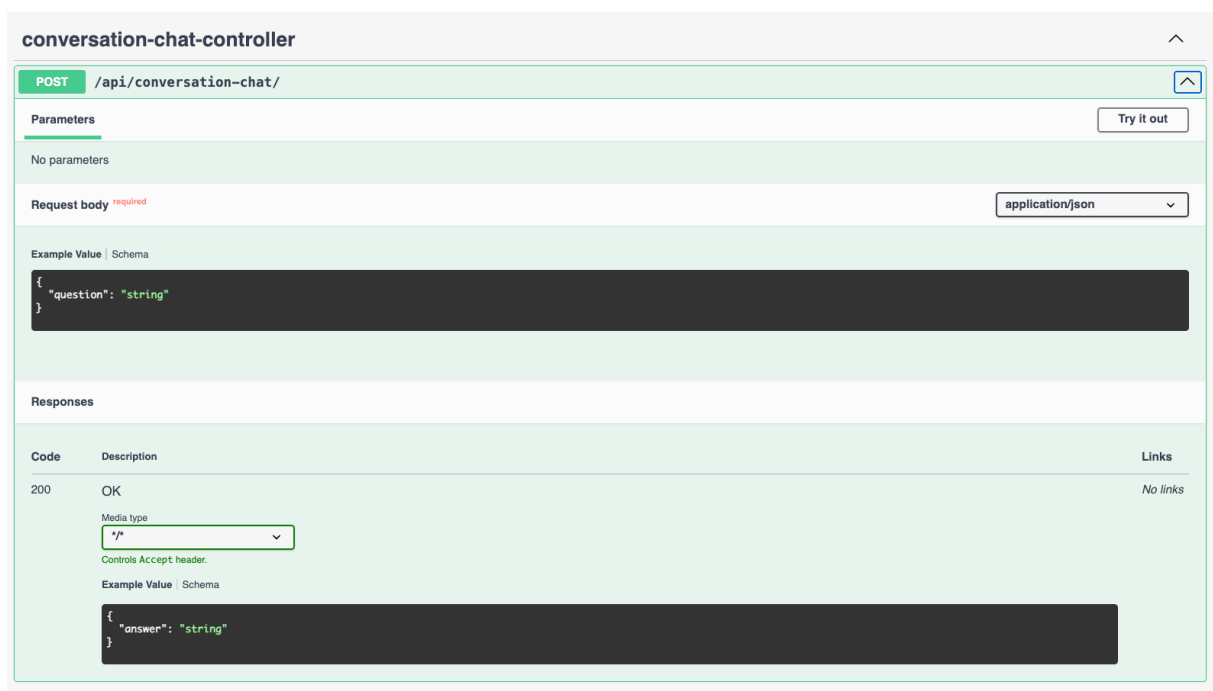


Abbildung XII: Conversation Chat Controller

AiConnect

Planung und Entwicklung einer RESTful-Schnittstelle zur Kommunikation mit einer firmenintern KI-Instanz und Bereitstellung einer Entwicklungs- und Laufzeitumgebung

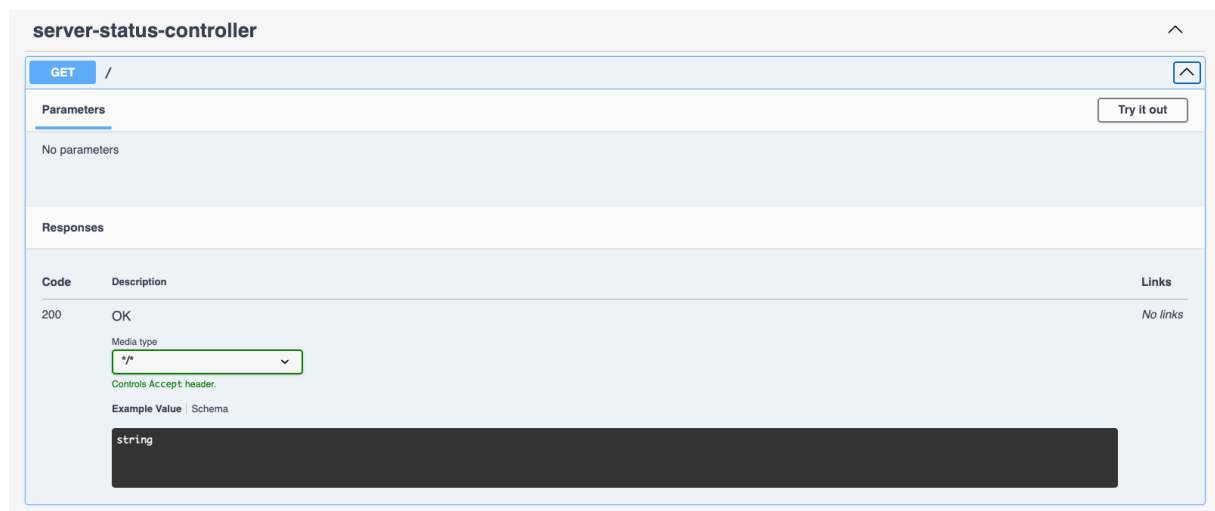


Abbildung XIII: Server Status Controller

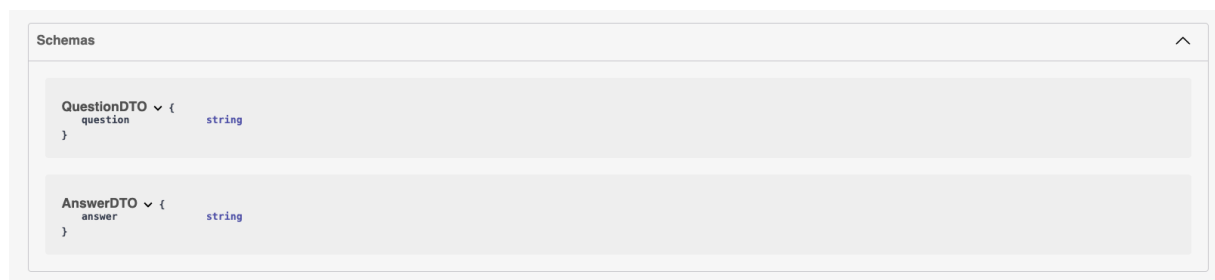


Abbildung XIV: DTO Schemas