htlperg
innovativ | kreativ | praxisnah

SYSco SIMS

SYSco Information Management System

DIPLOMARBEIT

Höhere Abteilung für Informatik

01/10/2024 – 04/04/2025

Projektmitglieder: Michael Haider
Tobias Fröschl

Betreuer:in: Dipl.-Ing. Helmut Otto

Eidesstattliche Erklärung

Hiermit versichern wir, Michael Haider und Tobias Fröschl, die vorliegende Arbeit mit dem Titel **„SYSco SIMS (SYSco Information Management System)“** eigenständig, ohne unerlaubte fremde Hilfe und nur unter Verwendung der von uns angegebenen Quellen verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften übernommen wurden, sind als solche kenntlich gemacht.

Bei der Erstellung der Arbeit haben wir generative KI-Tools verwendet, insbesondere ChatGPT, zur Unterstützung bei der Formulierung von Textpassagen, Gliederung und Strukturierung der Inhalte sowie sprachlichen Optimierungen einzelner Abschnitte. Die inhaltliche Verantwortung für die Arbeit liegt vollständig bei uns.

Kandidaten: Michael Haider, Tobias Fröschl

Ort, Datum: Perg, 03.04.2025

Unterschriften:

Michael Haider

Tobias Fröschl

Gendererklärung

Im Sinne der besseren Lesbarkeit werden in dieser Diplomarbeit personenbezogene Bezeichnungen, die sich zugleich auf Frauen und Männer beziehen, generell nur in der im Deutschen üblichen maskulinen Form angeführt. Dies soll jedoch keinesfalls eine Geschlechterdiskriminierung oder eine Verletzung des Gleichheitsgrundsatzes zum Ausdruck bringen.

Danksagung

Wir bedanken uns bei sämtlichen Lehrkräften der HTL Perg und der Firma SYSco, die uns bei der Realisierung unserer Diplomarbeit unterstützt haben.
Ein großer Dank geht an unseren Diplomarbeitsbetreuer Dipl.-Ing. Helmut Otto, der uns besonders bei der Entwicklung unseres Produkts sowie beim Verfassen unserer schriftlichen Arbeit unterstützt hat und immer für Fragen bereit war.
Nicht zu vergessen, natürlich ein großes Dankeschön an unseren Partner SYSco EDV, ohne sie wäre das Projekt nicht zustande gekommen. Egal ob bei Ideen für das Produkt, die wir umgesetzt haben, oder bei der kräftigen Unterstützung beim Realisieren unseres Chatbots + PDF-Manager, sie haben immer hinter uns gestanden und so einen großen Teil zum Erfolg des Produkts beigetragen. Hier zu erwähnen ist unser Betreuer Dominik Bindreiter von SYSco, der in dieser Zeit besonders viel mit uns in Kontakt gestanden ist und sich oft Zeit für Fragen oder Ähnliches genommen hat.

Abstract

Our product is made of two main parts: a chatbot and a PDF manager. The chatbot is designed to answer questions by using information from internal PDF documents. This saves time for users, because they no longer have to search through long manuals or guides to find the information they need.

To manage the documents that the chatbot uses, we developed a PDF manager. This tool allows users to upload, delete, download, and open PDF files. It also includes a search function that helps users find specific documents by typing in keywords. This reduces the effort required to locate relevant files and improves overall efficiency.

We worked closely with SYSco, our client, to choose the right technologies for the development of this product. Through teamwork and regular communication, we created a solution that fits their specific needs. The goal is to make daily work easier for SYSco's employees by giving them quick access to important information.

After a successful test phase, the product will also be available to SYSco's customers. It will help them get answers about SYSco's products quickly and easily, without needing to contact customer support directly.

Another important feature of the product is its flexibility. It is not limited to use by SYSco alone. Because the chatbot gets its answers from the uploaded PDFs, the product can also be used by other companies. They simply need to upload their own documents, and the chatbot will be ready to help.

In summary, our product reduces time and effort in finding information, supports both employees and customers, and can be used by many different companies.

Zusammenfassung

Zusammenfassend besteht unser Produkt aus einem Chatbot und einem PDF-Manager. Der Chatbot hat die Aufgabe, Fragen mithilfe von internen PDFs zu beantworten, um so den Aufwand für das Durchsuchen von Anleitungen von Produkten oder Ähnlichem zu minimieren. Um die PDFs der Firma zu verwalten und dem Chatbot zur Verfügung zu stellen, gibt es den PDF-Manager. Dieser hat verschiedene Funktionen zum Hochladen, Löschen, Herunterladen und Öffnen von PDFs. Zusätzlich bietet er die Funktion, Dokumente mittels Stichwörtern zu suchen, so wird der Aufwand für die Suche nach relevanten PDFs minimiert.
Mit der Firma SYSco, unserem Kunden, wurden Technologien gesucht, welche für so eine Umsetzung notwendig sind. Zusammen im Team und mit SYSco wurde so ein Produkt erstellt, welches Ihren Anforderungen entspricht und so die Arbeit der Mitarbeiter erleichtert. Zusätzlich soll es nach einer Testphase auch den Kunden helfen, wenn sie Fragen zu Produkten des Unternehmens haben.
Ein weiterer wichtiger Punkt ist es, das Produkt möglichst unabhängig von der Firma SYSco zu machen, sodass es theoretisch auch an weitere Unternehmen verkauft werden kann. Dies wurde mit dem PDF-Manager erreicht, da man hier die PDFs hochlädt, die der Chatbot verwenden soll, um die Fragen zu beantworten.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Künstliche Intelligenz ist ein Thema, das in der heutigen Zeit so präsent ist wie nie zuvor. Chatbots sind seit einigen Jahren in einem richtigen Hype. Daraufhin wollte die Firma SYSco auch so einen Chatbot, nur mit dem feinen Unterschied, dass er die Antworten auf die Fragen, die ihm gestellt werden, mit dem Wissen von den firmeninternen PDFs beantwortet. Das bietet den Vorteil, dass man, wenn Fragen der Mitarbeiter oder Kunden zu firmenspezifischen Themen auftreten, eine schnelle Antwort darauf vom Chatbot erhält. Wir als Team haben die passende Ausbildung an der HTL Perg und befassen uns demnach auch mit Künstlicher Intelligenz. Das Projekt ist zustande gekommen durch den Wunsch von SYSco und durch die Begeisterung für Künstliche Intelligenz der Schüler.

1.2 Problemstellung

Die Mitarbeiter von SYSco verbringen oft Stunden mit dem Lesen von umfangreichen Dokumentationen zu Produkten. Egal, ob bei der Fehlersuche in einem Projekt oder bei der Übernahme von einem Produkt, das man nicht selbst programmiert hat, überall dort ist es notwendig, sich in die Dokumentation einzulesen, um zu verstehen, wie das Produkt funktioniert.
Ein weiteres Problem stellt die zeitaufwendige Suche nach relevanten Dokumenten dar. Es kommt vor, dass die Informationen, die man gerade benötigt, in mehreren PDF-Dokumenten verteilt sind, und man so diese Dokumente zuerst mühsam lokalisieren muss, bevor man mit dem Arbeiten/Einlesen beginnen kann.
In beiden Fällen wird der gesamte Arbeitsablauf der Firma verlangsamt, und bei den Kunden führt dies zu längeren Wartezeiten für den Support oder ähnlichen Problemen.

1.3 Ausgangspunkt

1.3.1 Allgemein

Wir freuen uns, mit dem Unternehmen 'SYSco EDV ist Vertrauenssache' zusammenarbeiten zu dürfen. SYSco wollte ein Produkt, das ihren Mitarbeitern und später auch ihren Kunden dabei helfen soll, Fragen, die sie zu ihren Produkten oder Ähnlichem haben, beantwortet. Zusätzlich sollte die Suche nach relevanten Dokumenten massiv verkürzt werden. Dazu wollten sie also einen Chatbot und einen PDF-Manager, die genau das realisieren sollten. Ein vierköpfiges Team hat an diesem Produkt ab dem vierten Jahrgang der HTL-Perg gearbeitet. Hier kommt man auf eine Arbeitszeit von einem Schuljahr, in Stunden ergeben sich so mehrere hundert Arbeitsstunden, die an diesem Projekt gearbeitet wurden. Diese Zeit entspricht aber nicht der reinen Programmierzeit, da die Planung und Weiteres auch Zeit in Anspruch genommen haben. Am Ende des Schuljahres ist ein Produkt zustande gekommen, welches folgende Funktionen aufweisen konnte:

1.3.2 Chatbot

- Eine Benutzeroberfläche, die es dem Benutzer ermöglicht, eine Nachricht an den Chatbot-Server zu senden.
- Zu diesem Zeitpunkt ist nur eine einfache künstliche Intelligenz mittels LLM dahinter. (Beantwortet nur allgemeine Fragen)
- Kommunikation zwischen Client und Server ist vorhanden mittels SignalR.
- Die Antworten werden asynchron übermittelt.
- PDFs werden über eine REST-API an das Chatbot-Backend übermittelt und in der Chat-UI, für Testzwecke, gänzlich ausgegeben.

1.3.3 PDF-Manager

- Ein sehr einfaches Benutzerfenster mit Buttons wie "PDF-Auswählen", "PDF-Hochladen"welche auch die Funktion des Namens ausführen.
- Eine Datenbank, welche den Namen und den Speicherort des PDFs speichern kann.
- Das Programm, das die Operationen von der Benutzeroberfläche ausführt.

- Das Suchen von PDFs mittels Stichwörtern ist möglich, aber noch nicht in der gewünschten Performance.
- Die PDFs werden nicht direkt beim Hochladen dem Chatbot zur Verfügung gestellt, sondern erst ab dem Zeitpunkt, wo der Bot die PDFs benötigt, um die Frage des Benutzers zu beantworten.

Da das Projekt so umfangreich ist, wurde entschieden, dass zwei Schüler an diesem Projekt als Diplomarbeit fortfahren.

1.4 Zielsetzung

Grundsätzlich soll das Projekt erweitert und optimiert werden. Dazu zählen zum Beispiel, dass der Chatbot die Fragen präziser und schneller beantwortet, wobei nur die PDFs als Informationsquelle herangezogen werden sollen. Um mehr Freude beim Arbeiten mit dem PDF-Manager zu haben, soll er optisch ansprechender werden, die Datenbankstruktur erweitert und verbessert, der Code performanter gestaltet werden, mehr Funktionen sollen hinzugefügt und die alten Features sollen optimiert/ausgebaut werden. Neue Funktionen sind zum Beispiel das Herunterladen der PDFs und das Öffnen der Dokumente in einem Browser. Zu den Verbesserungen des Codes gehört die Verbesserung der Performance, die sinnvollere Benennung der Variablen und die Reduzierung von Codeduplikationen. Die Datenbank wird um sinnvolle Meta- und Userdaten ergänzt. Die Benutzeroberfläche von beiden Bestandteilen soll in ein Fenster zusammengefügt werden und mittels Tabs erreichbar sein. Ein weiterer Schritt ist die Implementierung der Views in die SYSco-Shell, welche dafür sorgt, dass das Produkt bei der Firma intern lauffähig und aufrufbar ist.

1.5 Projektumfeld

Der Hauptteil der Programmierarbeit wurde in einem Praktikum bei der Firma SYSco im Sommer 2024 erledigt. Dort hatten wir einen eigenen Arbeitsplatz, wo wir unsere Programmierarbeit erledigen konnten. Zusätzlich standen uns immer Kollegen zur Verfügung, falls Fragen aufgetaucht sind. Meetings fanden meistens vor Ort statt. Wenn ein physisches Treffen nicht möglich oder nicht lohnenswert war, wurde ein Meeting über Microsoft Teams abgehalten. Bei den Meetings präsentierten wir unseren aktuellen Stand und konnten Fragen stellen. Zusätzlich gab es die Möglichkeit, Fragen über einen Teams-Chat zu stellen, wo drei Angestellte von SYSco, Mitglieder der Gruppe, waren, die uns bei unseren Fragen geholfen haben. Die Zeiterfassung

der Arbeitsstunden wurde mittels Clockify und die Dokumentation mit einer Word-Datei gemacht.

1.5.1 Projektteam

Tobias Fröschl ist zuständig für den PDF-Manager.
Michael Haider ist zuständig für den Chatbot.

1.5.2 Betreuung

Dipl.-Ing. Helmut Otto

Dipl.-Ing. Helmut Otto ist seit 2023 wieder Lehrer im Bereich Informatik an der HTL Perg. Er ist Experte in Bereichen wie Programmierung in vielen verschiedenen Programmiersprachen sowie auch in der Artificial-Intelligence-Branche. Er ist außerdem MVP bei Microsoft, also "Most Valuable Professional", im Bereich Cloud- und Datacenter-Management sowie auch Microsoft Azure.

> 'Using Windows since it exists :)' - Dipl.-Ing. Helmut Otto

Ing. Dominik Bindreiter

Ing. Dominik Bindreiter ist der Leiter der Entwicklung in der Firma SYSco. Er beaufsichtigt sämtliche Projekte, die in der Entwicklung programmiert werden. Zusammen mit Herrn Aichinger und Herrn Pernthaler unterstützte er uns während der gesamten Praxis der Diplomarbeit.

1.5.3 Auftraggeber

SYSco EDV ist Vertrauenssache GmbH ist ein IT-Dienstleister aus Schwertberg, Oberösterreich, seit 1990. Das Unternehmen bietet beinahe alles an, von Hardware, Software, Netzwerklösungen bis hin zu Schulungen und Support. Ein besonderer Schwerpunkt liegt auf ERP-Systemen wie Mesonic WinLine für Handel und Industrie sowie VenDoc für das Baugewerbe. Diese Lösungen helfen Unternehmen, ihre Geschäftsprozesse effizient zu steuern. SYSco steht für Vertrauen, Kontinuität und Fairness, weshalb viele Kunden auf eine langfristige Zusammenarbeit setzen. Das Team besteht aus rund 54 IT-Experten. Geführt wird SYSco von Dipl. Ing. Peter Wurm und Ing. Thomas Rafetseder.

Abbildung 1: SYSco Logo

2 Theoretische Grundlagen

2.1 Grundlegende Fachbegriffe

2.1.1 LLM

LLM bedeutet "Large Language Model". Es ist ein spezialisierter Typ von Künstlicher Intelligenz, der darauf trainiert wurde, große Mengen an Text zu verstehen, zu verarbeiten und zu generieren. Diese Modelle nutzen Deep-Learning-Techniken und werden mit umfangreichen Datensätzen trainiert, um Muster und Strukturen der menschlichen Sprache zu erlernen. [1]

2.1.2 KM - Kernel Memory - Microsoft Projekt

Kernel Memory (KM) ist ein von Microsoft entwickeltes KI-gestütztes System zur effizienten Indizierung, Speicherung und Abfrage großer Datenmengen. Es kombiniert Techniken wie RAG, semantische Suche und kontextuelles Gedächtnis, um natürliche Sprachabfragen mit präzisen und kontextbezogenen Antworten zu ermöglichen. KM kann als Web-Service, Docker-Container oder .NET-Bibliothek integriert werden und ist mit Microsoft Copilot sowie dem Semantic Kernel kompatibel. Dadurch erlaubt es eine dynamische und skalierbare Speicherverwaltung für KI-Modelle, die erweiterte Wissensverarbeitung und Antwortgenerierung mit Zitaten und Quellenangaben unterstützt. [2]

2.1.3 RAG - Retrieval Augmented Generation

Retrieval Augmented Generation ist eine Methode in der künstlichen Intelligenz, die große Sprachmodelle (LLMs) mit externen Daten verbindet, um die Genauigkeit und Relevanz von generierten Antworten zu erhöhen. Es werden keine vorhandenen Daten, auf die bereits das LLM trainiert wurde, verwendet, sondern es werden Daten aus externen Quellen, z.B. in Form einer PDF-Datei, eingesetzt. Die gefundenen Informationen aus solchen Dokumenten werden mit dem bestehenden Wissen des Sprachmodells kombiniert, um eine präzise und kontextbezogene Antwort zu erstellen. [3] [4]

2.2 Technologien

2.2.1 C# / .NET

C# ist eine moderne, objektorientierte Programmiersprache, die von Microsoft entwickelt wurde. Sie ist Teil des .NET-Frameworks und wird häufig für die Entwicklung von Anwendungen auf der Windows-Plattform verwendet. C# bietet eine Vielzahl von Funktionen, darunter starke Typisierung, Garbage Collection und Unterstützung für asynchrone Programmierung, was die Entwicklung robuster und effizienter Anwendungen erleichtert. [5]

2.2.2 CUDA

CUDA bedeutet "Compute Unified Device Architecture". Das ist eine von NVIDIA entwickelte parallele Rechenplattform und Programmiermodell, das die Nutzung von GPUs für allgemeine Berechnungen ermöglicht. Mit CUDA können Entwickler rechenintensive Aufgaben parallelisieren, indem sie die massive Parallelität moderner GPUs ausnutzen. Dies führt zu erheblichen Leistungssteigerungen bei Anwendungen wie wissenschaftlichen Berechnungen, maschinellem Lernen und Bildverarbeitung. [6]

2.2.3 Git und GitLab

Git ist ein Open-Source-Tool zur Versionskontrolle, das Entwicklern hilft, den Überblick über Codeänderungen zu behalten. Man kann sich Git wie ein intelligentes Backup-System vorstellen, das jede Änderung speichert und es ermöglicht, zu früheren Versionen zurückzukehren. Besonders in Teams ist das praktisch: Mehrere Entwickler können gleichzeitig an einem Projekt arbeiten, ohne sich gegenseitig in die Quere zu kommen. Git speichert alles dezentral, das heißt, jeder hat eine komplette Kopie des Projekts und ist nicht auf eine zentrale Serververbindung angewiesen. GitLab geht noch einen Schritt weiter: Es ist eine Plattform, die auf Git aufbaut und das Arbeiten im Team noch einfacher macht. Statt nur Änderungen zu verfolgen, bietet GitLab eine zentrale Umgebung, in der Entwickler Code verwalten, testen und direkt bereitstellen können. Es kommt mit praktischen Zusatzfunktionen wie CI/CD, Code-Reviews und Projektmanagement-Tools. GitLab kann entweder als Cloud-Dienst genutzt oder auf eigenen Servern gehostet werden – was besonders für Unternehmen wichtig ist, die volle Kontrolle über ihre Daten behalten wollen. [7] [8]

Abbildung 2: Git

2.3 Entwicklungssysteme

2.3.1 Visual Studio 2022

Visual Studio 2022 ist eine integrierte Entwicklungsumgebung (IDE) von Microsoft, die Werkzeuge für die Entwicklung, das Debuggen und das Testen von Anwendungen bereitstellt. Sie unterstützt eine Vielzahl von Programmiersprachen, darunter C#, C++ und Python. Die neueste Version bietet verbesserte Leistung, erweiterte Debugging-Tools und Unterstützung für moderne Entwicklungsworkflows. [9]

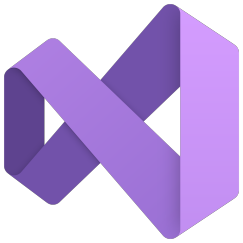

Abbildung 3: Visual Studio 2022

2.4 Bibliotheken/Plug-Ins (NuGets)

2.4.1 LLamaSharp

LLamaSharp ist eine Bibliothek für C# und .NET. Es kann große Sprachmodelle (LLMs) wie LLama oder LLava direkt auf dem eigenen Computer ausführen. Es läuft komplett lokal, also besteht keine Verbindung zu einem externen Dienst. Grundsätzlich kann man mit LlamaSharp einen eigenen intelligenten Assistenten, wie z.B. ChatGPT, selbst erstellen. Diese Bibliothek bietet jedoch noch weitere nützliche Tools, wie z.B. die Erweiterung mit KernelMemory, um Dokumente zu erfassen und zu analysieren. Außerdem ist es möglich, bereits bestehende LLMs feinzutunen, also noch weiter zu perfektionieren. LLamaSharp basiert auf llama.cpp, das das 'offizielle' LLama von META bereitstellt. Es kann sowohl auf der CPU als auch auf der GPU ausgeführt werden, die es benötigt, um die KI-Anwendung auszuführen. Dieses Projekt ist ein aktives Open-Source-Projekt und ist daher noch sehr instabil bei gewissen Versionen. [10] [11]

2.4.2 LLamaSharp.KernelMemory

LLamaSharp.KernelMemory ist eine Erweiterung der LLamaSharp-Bibliothek und hilft dabei, Large Language Models direkt in .NET-Anwendungen einzusetzen. Damit kann eine Anwendung Texte durchsuchen, verstehen sowie auch intelligente Antworten daraus geben. Einfach gesagt, ein persönlicher Assistent, der anhand von Dokumenten antwortet, die ihm der Benutzer vorher gegeben hat. KernelMemory speichert die Dokumente so effizient, sodass es jederzeit gezielt nach Antworten suchen kann. Somit sind die Antworten nicht nur willkürlich generiert, sondern werden

speziell auf relevante Informationen der vorhandenen Daten durchsucht, damit die Antwort umso präziser wird. Das Paket wird nur unter .NET 8.0 oder höher unterstützt. [12]

2.4.3 SignalR

SignalR ist eine Open-Source-Bibliothek für ASP.NET-Entwickler. Mithilfe von SignalR ist es möglich, Inhalte in Echtzeit an verbundene Clients zu senden, und das, ohne dass man manuell nach Updates fragen muss. Es werden neben WebSockets noch weitere Transportprotokolle wie Server-Sent Events oder Long Polling, falls WebSockets nicht unterstützt werden, angeboten. Zusätzlich bietet es neben .NET-Clients auch JavaScript-Clients für Webanwendungen sowie Java-Clients für Android-Apps an, wodurch ein breites Feld von Plattformen abgedeckt wird. Durch diese Eigenschaften eignet sich SignalR gut für Anwendungen, bei denen ein dauerhafter und schneller Austausch von Daten stattfinden muss, wie zum Beispiel bei Chatsystemen oder Live-Dashboards. Auf GitHub findet man den Quellcode und die offizielle Dokumentation von SignalR, somit ist das Einsetzen von SignalR und das Suchen von Fehlern deutlich vereinfacht. [13] [14] [15]

2.4.4 Lucene.NET

Lucene.NET ist eine leistungsstarke Open-Source-Suchbibliothek, die ursprünglich für Java entwickelt wurde. Später wurde es aber in die .NET-Welt portiert, um es in .NET-Anwendungen benutzen zu können. Entwickelt wurde es von Doug Cutting, einer bekannten Persönlichkeit aus der Open-Source-Welt und Mitgründer von Apache Hadoop. Heute wird es von der Apache Software Foundation weiterentwickelt. Lucene.NET ist keine eigenständige Anwendung, sondern eine Codebibliothek und Programmierschnittstelle (API), mit der sich bestehende Anwendungen einfach um leistungsfähige Suchfunktionen erweitern lassen. Man kann mit diesem Tool mittels Stichwörtern Texte, nach genau diesen Schlagwörtern, durchsuchen. Wenn etwas gefunden wird, kann man sich zum Beispiel die relevanten Dokumente, in welchen sich die gesuchten Stichwörter befinden, in einem C#-Programm zurückgeben lassen. Zusätzlich hat man die Möglichkeit, nur Dokumente ab einem bestimmten Score, also welche die die Stichwörter mehrmals beinhalten und somit relevanter sind, zurückzugeben. Lucene.NET läuft auf jeder Plattform, die .NET unterstützt, einschließlich ASP.NET, WinForms, WPF und Blazor. Dazu zählen Systeme wie Windows, Mac oder Unix. Es ist also bei Websites, mobilen Apps (Android oder iOS) Desktop-Anwendungen oder auf IoT-Geräten als Suche anwendbar. Um Lucene.NET

Abbildung 4: Lucene.NET

besser zu verstehen, kann man im Quellcode, der auf GitHub zur Verfügung steht, nachsehen, was ein großer Vorteil von Open-Source-Programmen ist. [16] [17]

2.4.5 Microsoft Edge WebView2

WebView2 ist eine Technologie von Microsoft, mit der man Webinhalte direkt in Desktop-Anwendungen einbinden kann. Das bedeutet, dass eine App nicht komplett nativ entwickelt werden muss, sondern moderne Web-Technologien wie HTML, CSS und JavaScript nutzen kann. WebView2 basiert auf Microsoft Edge (Chromium), was bedeutet, dass es eine leistungsstarke und aktuelle Rendering-Engine verwendet – ähnlich wie ein eingebauter Browser in der Anwendung. Entwickler können so moderne Webfunktionen nutzen, ohne ständig zwischen einer Web- und einer Desktop-App wechseln zu müssen. WebView2 eignet sich perfekt für hybride Anwendungen, bei denen ein Teil des Programms als klassische Desktop-Software läuft, während andere Bereiche auf Web-Technologien basieren. Es funktioniert mit Win32 C/C++, .NET und WinUI und ist mit Windows 10 und 11 kompatibel. Microsoft bietet zwei Versionen: Die Evergreen-Version, die sich automatisch aktualisiert, und eine festgelegte Version, die genau definiert bleibt. [18]

2.4.6 EntityFramework

Entity Framework ist ein Open-Source-Objekt-Relationales Mapping (ORM)-Framework, das für .NET-Anwendungen (C#) eingesetzt werden kann. Hinter der Entwicklung steckt Microsoft. Es ermöglicht, mit Datenbanken bei Verwendung von .NET-Objekten zu arbeiten. Es ist in der Lage, Datenzugriffscodes zu erstellen. Somit wird der Aufwand beim Programmieren von Zugriffscodes, wie die Modellklassen in klassischen MVC-Architekturen, verringert. Zudem müssen keine SQL-Abfragen manuell geschrieben werden, sondern es können Datenbankabfragen direkt mit LINQ oder anderen Methoden ausgeführt werden. Dies sorgt dafür, dass die Fehleranfälligkeit bei Datenbankzugriffen und bei Modellklassen stark verringert wird. Seit der Version 6.0 ist Entity Framework als eigenes Paket bereitgestellt, außerhalb des .NET-Frameworks und unter einer Open-Source-Lizenz entwickelt. Die neueste Entwicklung, Entity Framework Core (EF-Core), ist eine leichtgewichtige, plattformübergreifende und erweiterbare Version des ursprünglichen Entity Frameworks. EF Core unterstützt verschiedenste Datenbank-Engines, darunter SQL Server, MySQL, SQLite, PostgreSQL, durch eine Plugin-API. In Visual Studio 2022 wird EF-Core durch verschiedene Werkzeuge unterstützt, welche das Arbeiten mit Datenbanken erleichtern. Zum Beispiel können über die Package Manager Console oder über das EF Core Command Line Interface, Datenbankmigrationen erstellt und verwaltet werden. Die Modellierung erfolgt typischerweise über den Code-First-Ansatz, wobei das Datenbankschema direkt aus C#-

Klassen generiert wird. Alternativ können aus einer bestehenden Datenbank die entsprechenden Modellklassen generiert werden, das wird Database-First-Ansatz genannt. [19] [20] [21]

2.5 Hardware

2.5.1 Windows Server 2022

Windows Server 2022 ist das aktuelle Server-Betriebssystem von Microsoft und wurde entwickelt, um stabil, sicher und leistungsfähig zu sein. Es baut auf den Stärken von Windows Server 2019 auf, bringt aber viele Verbesserungen mit – vor allem in den Bereichen Sicherheit, Cloud-Integration und Virtualisierung. Ein Fokus liegt auf erweiterten Sicherheitsfunktionen. Windows Server 2022 unterstützt TLS 1.3 für verschlüsselte Verbindungen, bringt eine bessere SMB-Verschlüsselung mit und hat spezielle Sicherheitsfunktionen für moderne Hardware, um Angriffe zu erschweren. Wer also sensible Daten verarbeitet, bekommt hier ein besonders sicheres System. Auch für Unternehmen, die teils lokal und teils in der Cloud arbeiten, ist Windows Server 2022 perfekt. Die Azure-Integration wurde verbessert, sodass sich Cloud-Dienste einfacher nutzen und verwalten lassen. Zudem läuft das System extrem stabil und bringt Optimierungen für Container und Virtualisierung, was moderne Software-Entwicklung erleichtert. [22]

2.6 Datenhaltung

2.6.1 Docker

Docker ist eine Open-Source-Software, die benutzt werden kann, um Anwendungen isoliert in Container-Virtualisierungen laufen zu lassen. Es werden von einer Anwendung alle Codebereiche und Abhängigkeiten in ein Image gegeben, welches dann später von der Docker-Software in Form eines Docker-Containers ausgeführt wird. Das hat den riesigen Vorteil, dass kein ganzes Betriebssystem virtualisiert wird, sondern wirklich nur die eine Anwendung, was den Bedarf von Ressourcen gering hält. Trotzdem läuft die Anwendung isoliert vom Host-Betriebssystem. Ein weiterer Vorteil von Docker ist es, dass die Anwendung in jedem System in einem Container gestartet werden kann, das auch Docker unterstützt. Dazu zählen Systeme wie Windows, Linux oder sogar macOS. Standardmäßig verwendet Docker aber Linux-Container, was bedeutet, wenn man diesen in

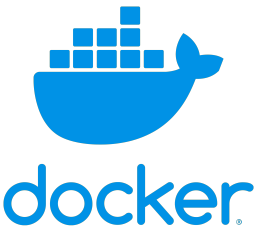

Abbildung 5: Docker

einem anderen Betriebssystem startet, läuft im Hintergrund ein Linux-Betriebssystem, also eine leichtgewichtige Umgebung, mithilfe von Hyper-V oder WSL2. Man kann aber auch Windows-Container starten, diese laufen auf dem Windows-Kernel, also demselben wie das Betriebssystem selbst, was bedeutet, dass kein Linux im Hintergrund läuft. Zusätzlich bietet Docker Tools an wie Docker-Compose, mit denen man Multi-Container-Anwendungen definieren und orchestrieren kann.

[23] [24]

2.6.2 MSSQL

MSSQL, ausgeschrieben Microsoft SQL Server, ist ein relationales Datenbankmanagementsystem (RDBMS), das von Microsoft entwickelt wurde. Es wird verwendet für die Speicherung, Verwaltung und Verarbeitung von Daten. Die Daten werden in Tabellen, die in Beziehungen zueinander stehen können, gespeichert. MSSQL verwendet die Programmiersprache SQL (Structured Query Language). Microsoft hat diese Sprache für sich erweitert und nennt sie Transact-SQL (T-SQL). Am Grundprinzip ändert sich nichts, jedoch ist die Sprache an sich leicht abgeändert. Das System unterstützt Funktionen zur Datenanalyse, Berichterstellung, Sicherheitsverwaltung und Transaktionsverarbeitung. MSSQL ist in verschiedenen Editionen verfügbar, je nach Einsatzzweck. Meistens wird SQL in professionellen IT-Umgebungen eingesetzt, da es hohe Stabilität, Sicherheit und Skalierbarkeit bietet. MSSQL ist für Windows sowie auch für Linux verfügbar. [25] [26]

2.6.3 SQL Server 22 Express

SQL Server 2022 Express ist eine kostenlose Version von Microsofts Datenbankmanagementsystem SQL Server 2022. Es eignet sich ideal für die Entwicklung von Desktop-, Web- oder kleineren Serveranwendungen. Es bietet eine sehr zuverlässige und leistungsfähige Datenverwaltung. Da die Express-Version kostenfrei ist, gibt es ein paar Einschränkungen wie z.B. die maximale Datenbankgröße von 10 GigaByte. Also ist es für kleine Projekte und Anwendungen gut einsetzbar, um so auch Kosten zu sparen. [27] [28]

2.6.4 SQL Server Management Studio

SQL Server Management Studio (SSMS) ist das zentrale Tool von Microsoft, um SQL-Server-Datenbanken zu verwalten. Es stellt eine grafische Oberfläche zur Verfügung, mit der man Datenbanken einfach erstellen, bearbeiten und abfragen kann. Das Ganze funktioniert in dem

Objekt-Explorer, der alle Datenbanken, Tabellen und auch weitere wichtige Elemente in einer Baumstruktur anzeigt. So wird sich viel Arbeit mit SQL-Befehlen erspart. Es bietet jedoch ebenso einen integrierten Editor, in dem SQL-Befehle eingegeben werden können. [29]

2.6.5 Swagger

Swagger ist ein Open-Source-Framework, welches für die automatische Dokumentation und für die Visualisierung von REST-APIs dient. So ermöglicht es Swagger den Entwicklern, ihre API-Endpunkte einfach zu testen, ohne dafür selbst einen Code zu schreiben, der darauf zugreift. Dafür wird eine Swagger-UI in einem Browser aufgerufen, welche es ermöglicht, die einzelnen API-Endpunkte aufzurufen und die Rückgabe einzusehen. In ASP.NET Core wird Swagger mit dem Swashbuckle.AspNetCore-Paket integriert und über UseSwagger() aktiviert. Durch die Nutzung von Swagger wird also die API-Entwicklung effizienter und deutlich erleichtert und wartbarer, was es zu einem sehr nützlichen Tool macht. [30] [31]

Abbildung 6: Swagger

2.7 Sonstige Software

2.7.1 Internet Information Service - IIS

Der Internet Information Service ist Microsofts Webserver für Windows Server. Er sorgt dafür, dass Webseiten, Web-Apps und APIs sicher und schnell im Internet oder in einem internen Netzwerk laufen. Ob kleine private Webseiten oder große Unternehmensplattformen – mit IIS lassen sich Inhalte zuverlässig bereitstellen. Ein großer Vorteil von IIS ist, dass man ihn an die eigenen Bedürfnisse anpassen kann. Man entscheidet selbst, welche Funktionen aktiv sind und welche nicht. Dadurch bleibt der Server schlank und arbeitet effizient. Wer Wert auf Sicherheit legt, kann verschiedene Schutzmaßnahmen aktivieren. Wer eine schnelle Website braucht, kann Einstellungen optimieren, damit alles flüssig läuft. Außerdem unterstützt IIS neben Microsofts eigener Technologie ASP.NET auch PHP, Node.js und andere Programmiersprachen. Die Verwaltung von IIS ist ebenfalls einfach. Wer es bequem mag, nutzt die grafische Oberfläche, in der man alles per Mausklick steuern kann. Wer lieber direkt arbeitet, kann den Server über die Kommandozeile oder PowerShell verwalten. Damit ist IIS für Anfänger genauso geeignet wie für Profis. [32]

2.7.2 Microsoft Teams

Microsoft Teams ist eine Nachrichten-App von Microsoft, die sich speziell auf Organisationen spezialisiert, die Zusammenarbeit und Kommunikation in Echtzeit erfordern. Diese wird mit einfachen Chats zwischen Personen sowie Gruppen, die man anlegen kann, realisiert. Zusätzlich bietet Teams die Möglichkeit, Besprechungen abzuhalten, Dateien und Apps freizugeben, was es zu einem guten Allrounder für Gruppen macht, die Kommunikation via Chat oder Besprechungen benötigen. [33]

2.7.3 Clockify

Abbildung 7: Clockify

Clockify ist eine kostenlose Software, die Zeiterfassungen ermöglicht. Das kann man zum einen mit einem Klick auf Start, der die Zeiterfassung beginnen lässt, und später wieder mit einem Klick auf Stop, realisieren. Oder man kann die Zeit auch im Nachhinein manuell eintragen, indem man die Zeit durch das Eintragen von Start- und Endzeitpunkt erfasst. Die Zeiten werden in einer Liste gespeichert, wo man sieht, an welchem Projekt man gearbeitet hat, und eine Beschreibung, die man beim Starten der Zeit eingeben kann. Man hat die Möglichkeit, verschiedene Projekte anzulegen oder an welchen teilzunehmen, die später bei der Zeiterfassung auszuwählen sind, um immer nachverfolgen zu können, wer wann an welchem Projekt gearbeitet hat. Zusätzlich dient es zur Zusammenfassung und Veranschaulichung der Zeiten von einem Projekt und deren Teilnehmern. [34]

3 Implementierung

3.1 SYSco-Shell

3.1.1 Allgemein

Die SYSco Shell ist das interne Framework der Firma, in der sämtliche Applikationen, die eigens entwickelt wurden, enthalten sind. Jede Applikation wird als 'Modul' bezeichnet. Dieses Framework wird ebenso an die Kunden übergeben, damit diese auf die Module zugreifen können und die Software mitnutzen. Eines dieser Module stellt nun in Zukunft den Chatbot und den PDF-Manager dar.

3.1.2 Integration

Die SYSco Shell arbeitet mit DevExpress MVVM. Durch diese Struktur werden unsere Module mittels ViewModel in die Shell importiert.

Im nachfolgenden Code wird das ViewModel eines Moduls beschrieben:

Listing 1: ViewModel-Example von Chatbot

```
1  public class Chat_ViewModel : IModulViewModel
2  {
3      public string Caption { get; set; } = "ChatBot";
4      public SvgImage Image { get; set; } =
           SvgImage.FromResources("scoSIMS.Images.Chat.svg", typeof(Chat_ViewModel).Assembly);
5      public bool LayoutIsCustomization { get; set; }
6      public User User { get; set; }
7      public Module Module { get; set; }
8
9      public Chat_ViewModel() : this(new User() { Username = "testUser" }) { }
10     public Chat_ViewModel(User user)
11     {
12         User = user;
13         Messenger.Default.Send(new UserMessage(user));
14     }
15
16     public void LoadLayout(string layout) { }
17     public string SaveLayout() {    return "";  }
18     public static Chat_ViewModel Create(Dictionary<int, object> parameters)
19     {
20         Module module = parameters.GetModuleParam<Module>(ModuleParameter.Module)!;
21         List<User> users = parameters.GetModuleParam<List<User>>(ModuleParameter.Users)!;
22         User user = parameters.GetModuleParam<User>(ModuleParameter.User)!;
23
24         return ViewModelSource.Create(() => new Chat_ViewModel(user));
25     }
26 }
```

3.1.3 Module hochladen

Für jedes Modul muss ein ViewModel bestehen, das definiert, wie das Modul in der Shell dargestellt wird. Nun gibt es eine Datei, die bei jedem SYSco-Projekt besteht, und zwar die 'ModuleInfo.cs'. Diese wird vom SYSco-GitLab automatisch erkannt und verarbeitet deren Inhalt. Hier werden dann noch spezifische Details der Module angegeben. Ebenso wird im ModuleInfo die View und das dazugehörige ViewModel mitgegeben.
Die ModuleInfo.cs sieht folgendermaßen aus:

Listing 2: ModuleInfo für die Shell

```
public sealed class ModuleInfo : IModuleInfo {
    public string Sentry => ".......";

    public Type? Permissions => null;

    public IEnumerable<DisplayViews> DisplayViews => [
        new() {
            Caption = "ChatBot",
            Image = SvgImage.FromResources("scoSIMS.Images.Chat.svg",
                typeof(Chat_ViewModel).Assembly),
            ModelName = nameof(SIMSchatViewModel),
            ViewGroup = "SIMS",
            ViewName = nameof(SIMSchatView)
        },
        new () {
            Caption = "PDF-Manager",
            Image = SvgImage.FromResources("scoSIMS.Images.PDF.svg",
                typeof(PDF_List_ViewModel).Assembly),
            ModelName = nameof(SIMSpdfViewModel),
            ViewGroup = "SIMS",
            ViewName = nameof(SIMSpdfView)
        }
    ];
}
```

3.2 SignalR

3.2.1 Kommunikation zwischen ChatUI und Chatbot-Server

Der Chatbot verwendet SignalR zur Kommunikation für Nachrichten. In der Benutzeroberfläche wird eine Nachricht eingegeben und diese wird dann über SignalR an den Server versendet. Der Client definiert einen 'HubConnectionBuilder()'. Hier wird z.B. die URL vom Server hinterlegt, mit dem sich der Client verbinden soll.
Das Frontend ist in HTML und JavaScript programmiert.
Somit sieht die Implementierung folgendermaßen aus:

Listing 3: Verbindungsparameter festlegen

```
const connection = new signalR.HubConnectionBuilder()
    .withUrl("http://localhost:5233/chatHub", {
        withCredentials: true
    })
    .build();
```

SignalR ist deshalb gut geeignet, weil die Nachrichten asynchron übermittelt werden können. Die Antwort vom Server kann also Token für Token (Wort für Wort) an den Client zurückgesendet werden.

Ursprünglicher Lösungsansatz

Der Benutzer gibt eine Nachricht in das Textfeld ein und verschickt sie an den Server. Der Server erhält die Nachricht und generiert mithilfe des LLMs eine passende Antwort darauf. Während der Server die Antwort generiert, versendet er gleichzeitig jedes Wort, das er generiert, an den Client zurück. Somit muss der Benutzer nicht darauf warten, bis das LLM fertig generiert hat und dann erst versendet, sondern kann in Echtzeit Wort für Wort die Antwort mitverfolgen.

Listing 4: Kommunikation Token-by-Token

```
// Antwort per tokens versenden
await foreach (string token in _modelService.GetAnswerFromBot(msgUser.Content))
{
    msgAssistant.Content += token;
    await Clients.Caller.SendAsync("ReceiveMessage", token);
}

// Methodenaufruf -> GetAnswerFromBot()
public async IAsyncEnumerable<string> GetAnswerFromBot(string message)
{
    var msg = new LLama.Common.ChatHistory.Message(LLama.Common.AuthorRole.User, message);
    // Generate Answer
    await foreach (string text in Session.ChatAsync(msg, _inferenceParams))
    {
        yield return text;
    }
}
```

Problem

KernelMemory stellt für die Generierung einer Antwort aus einer PDF-Datei nur eine 'AskAsync()'-Methode zur Verfügung. Diese Methode hat den Nachteil, dass am Server auf die Antwort gewartet werden muss, bis sie fertig generiert wird. Somit muss der Benutzer ebenfalls auf die Antwort warten und hat keine Echtzeitanzeige der Wörter.

Alternative Lösung

Durch das Problem mit KernelMemory wurde eine andere Lösung verwendet. Es wird weiterhin SignalR verwendet, jedoch mit dem Unterschied, dass auf die Nachricht gewartet wird und diese dann mit einem einzigen Send-Befehl die gesamte Antwort an den Benutzer übermittelt wird. Diese Variante hat den Nachteil, dass es immer zu ewigen Ladezeiten in der Benutzeroberfläche kommt. Hier können Wartezeiten von mehreren Minuten entstehen.

Nun die aktuelle Implementierung, wie diese Methode mit KernelMemory umgesetzt wurde:

Listing 5: Kommunikation als ganze Nachricht

```
// Antwort versenden
MemoryAnswer answer;
answer = await _kmService.GenerateAnswer(msgUser);
await Clients.Caller.SendAsync("ReceiveMessage", answer.Result);

// Methodenaufruf -> GenerateAnswer()
public async Task<MemoryAnswer> GenerateAnswer(string question)
{
    MemoryAnswer answer = await _memory.AskAsync(question);
    return answer;
}
```

3.2.2 PDF-Austausch zwischen PDF-Server und Chatbot-Server

Im PDF-Manager werden die Dokumente, die hochgeladen werden, mittels SignalR-Verbindung an den Chatbot geschickt. Hierzu wird in der 'UploadFile'-Klasse, welche beim Aufrufen den Namen des Dokuments als Parameter übergeben bekommt, zuerst einmal eine Verbindung zu einem SignalR-Hub definiert. Die URL zeigt hierbei auf den laufenden SignalR-Server beim Chatbot:

Listing 6: Verbindung aufbauen

```
var connection = new HubConnectionBuilder()
    .WithUrl("http://172.16.100.145:9090/chatHub")
    .Build();

await connection.StartAsync();
```

Mit dem await-Befehl unten wird eine asynchrone Verbindung aufgebaut, die immer darauf wartet, dass die Verbindung erfolgreich ist, bevor der Code weiterläuft.
Ist die Verbindung erfolgreich aufgebaut, geht es weiter mit dem Senden der Daten. Diese werden in kleinen Datenschnipseln (Chunks), in diesem Fall in 500 KB großen Chunks, an den Chatbot gesendet. Um dem Empfänger klarzumachen, wann die Übertragung beendet ist und somit alle Chunks übergeben wurden, wird der letzte Chunk mithilfe von 'isLastChunk' markiert:

Listing 7: Senden der Datei in Chunks

```
var chunkSize = 500 * 1024; // 500KB
var totalChunks = (int)Math.Ceiling((double)fileBytes.Length / chunkSize);

for (int i = 0; i < totalChunks; i++)
{
    byte[] chunk = fileBytes.Skip(i * chunkSize).Take(chunkSize).ToArray();
    bool isLastChunk = (i == totalChunks - 1);
```

In der Schleife geht es weiter damit, dass beim Empfänger die SignalR-Hub-Methode aufgerufen wird, die die Chunks annimmt:

Listing 8: Methode beim Empfänger aufrufen

```

await connection.InvokeAsync("UploadFileChunk", fileName, chunk, isLastChunk);
```

Hier werden dann die Parameter Filename, das aktuelle Chunk und das 'isLastChunk' übergeben.

Schlussendlich wird die Verbindung mit dem 'connection.StopAsync'()-Befehl, geschlossen und fertig ist die Übertragung mit SignalR.

3.3 LLamaSharp - Integration von LLMs

LLamaSharp ist die Hauptkomponente für die Integration von LLMs in den Chatbot. Es wird nicht nur dafür genutzt, um das Sprachmodell einzubinden, sondern dient gleichzeitig als Schnittstelle für KernelMemory. Ohne LLamaSharp wäre es nicht möglich, das KernelMemory-Plugin zu implementieren, da das KM in diesem Kontext eine Erweiterung von LLamaSharp ist.

3.3.1 Auswahl des Backends für die LLM-Erweiterung

LLamaSharp bietet verschiedene Möglichkeiten, wie das LLM im Hintergrund ausgeführt wird:

- CPU - das Modell wird über den Prozessor berechnet
- OpenGL - das Modell nutzt die Grafikkarte
- CUDA - das Modell nutzt speziell Nvidia-GPUs die für CUDA optimiert sind

Da das Programm auf einem SYSco-Server in einer virtuellen Umgebung läuft, wird CPU als Backend verwendet. Der Grund dafür ist, dass es in dieser Umgebung keinen Grafikkarten-Support gibt. Dadurch wird sichergestellt, dass das LLM trotzdem genutzt werden kann, auch wenn es nicht mit einer GPU berechnet wird.

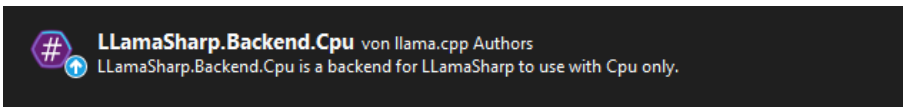

Abbildung 8: Verwendetes NuGet für das CPU-Backend

3.3.2 Optimierung der CPU-Nutzung

Das LLM läuft auf dem Prozessor. Dafür müssen einige Optimierungen vorgenommen werden, damit es möglichst schnell arbeitet. Dazu gehören:

- Mehrere CPU-Kerne nutzen
  Dadurch arbeitet das LLM schneller, da die Berechnung auf mehrere Kerne verteilt werden.
- Arbeitsspeicher
  Die CPU nutzt den Arbeitsspeicher. Somit muss der RAM auch genügend Kapazität zur Verfügung stellen.
- Modellgröße
  Die Modellgröße ist in diesem Fall abhängig von der Arbeitsspeicher-Kapazität. Somit kann nur ein Modell verwendet werden, dass auch im Rahmen der RAM-Kapazität bleibt.

3.4 KernelMemory - Analyse/Speichern von Dokumenten

KernelMemory wird zusammen mit LLamaSharp genutzt, um PDFs zu analysieren und die Inhalte in einer MSSQL-Datenbank auf dem SYSco-Server zu speichern. Dabei werden komplette PDF-Dateien an KernelMemory übergeben, das daraus Vektordaten erstellt. Das bedeutet, dass alle Inhalte eines Dokuments erfasst und in eine Form gebracht werden, die später durchsucht werden kann.

Ein wichtiger Bestandteil von KernelMemory ist RAG. Diese Komponente ist für das Finden passender Informationen zuständig. Wenn eine Anfrage gestellt wird, durchsucht RAG die gespeicherten Daten und stellt die passenden Inhalte zusammen. Das LLM von LLamaSharp nutzt diese Daten dann, um eine Antwort zu generieren. Dadurch bestehen die Antworten sowohl aus den Fakten aus den PDFs als auch aus dem allgemeinen Wissen des KI-Modells.

3.4.1 Analysierung

Die Analyse von PDFs läuft in mehreren Schritten ab. Zuerst wird das Dokument in verschiedene Teile zerlegt. Dazu gehören der Haupttext, Metadaten wie Titel und Erstellungsdatum sowie weitere Strukturen wie Tabellen oder Aufzählungen. KernelMemory verarbeitet all diese Daten und erstellt daraus Vektoren, um in Zukunft damit besser und schneller zu arbeiten.

Während der Analyse erkennt dieses System wichtige Begriffe oder Zusammenhänge. Dadurch ist es leichter, ähnliche Inhalte zu finden. Unnötige Informationen werden automatisch herausgefiltert, sodass nur relevante Daten erfasst werden. Um Antworten besser verfassen zu können, ist dieser Schritt notwendig, damit nur sinnvolle Antworten generiert werden.

3.4.2 Speicherung

Die analysierten Daten werden in einer MSSQL-Datenbank auf dem SYSco-Server gespeichert. Es werden die Vektordaten sowie auch andere Informationen wie der Dokumententitel oder das Datum der Analyse abgelegt. Dadurch soll erreicht werden, dass das spätere Auffinden von Informationen beschleunigt wird.

KernelMemory verwendet spezielle Suchmethoden, die das Beantworten von Fragen immens verbessern. Wenn ein Benutzer eine Frage stellt, sucht das System in den gespeicherten Daten nach den passenden Informationen und generiert daraus mithilfe des LLMs eine verständliche und vollständige Antwort.

3.4.3 Aufbau im Code

Im folgenden Code ist die genaue Implementierung zu sehen, die zeigt, wie ein KernelMemory aufgebaut wird und welche Komponenten für das Erstellen benötigt werden. Es wird bei jedem Start ein neues Element erstellt, welches sich sämtliche Vektor-Daten aus der Datenbank neu lädt.

Listing 9: Builden des KernelMemory

```
public void CreateMemory()
{
    var sqlServerConfig = new SqlServerConfig
    {
        ConnectionString = _configuration["Variables:ConnectionString_KM"]
    };
    InferenceParams infParams = new()
    {
        AntiPrompts = ["\n"],
        Temperature = 0.5f,
        MaxTokens = 250
    };
    LLamaSharpConfig lsConfig = new(_modelPath) { DefaultInferenceParams = infParams };
    SearchClientConfig searchClientConfig = new()
    {
        MaxMatchesCount = 3,
        AnswerTokens = 250,
    };
    TextPartitioningOptions parseOptions = new()
    {
        MaxTokensPerParagraph = 300,    // Anzahl der Tokens pro Paragraph
        MaxTokensPerLine = 100,         // Tokens pro Zeile
        OverlappingTokens = 30          // Muss kleiner sein als MaxTokensPerParagraph
    };
    _memory = new KernelMemoryBuilder()
        .WithSqlServerMemoryDb(sqlServerConfig)
        .WithLLamaSharpDefaults(lsConfig)
        .WithSearchClientConfig(searchClientConfig)
        .With(parseOptions)
        .Build();
}
```

Aufgrund von Versionskonflikten zwischen LLamaSharp, KernelMemory und der SqlServerMemoryDb (siehe oben Codesnippet) kommt es sehr oft zu Komplikationen bei der Ausführung des Programms. Somit wird als Alternative keine MSSQL-Datenbank, sondern ein Local-Storage

verwendet. Der Unterschied besteht darin, dass sämtliche Vektor-Daten der PDFs nicht in einer Datenbank in Tabellen gespeichert werden, sondern lokal auf dem Server in einem definierten Verzeichnis.

3.5 Suchalgorithmus

Im PDF-Manager hat man die Funktion, PDFs mittels Stichwörtern zu suchen. Wie das genau funktioniert, wird hier beschrieben. Hinter dem Ganzen steckt die Technologie Lucene.NET, die so eingebunden wurde, dass sie dem PDF-Manager die gewünschte Fähigkeit verleiht, die PDFs zu durchsuchen.

3.5.1 Architektur der Suchfunktion

Folgende Komponenten sind Teil der Suchfunktion des PDF-Managers:

- Textextraktion aus PDF-Datei:
  Lucene.NET bietet keine Möglichkeit den Text aus PDFs selbst zu extrahieren. Daher wurde iTextSharp zur Extraktion verwendet. iTextSharp ist eine Open-Source-.NET-Bibliothek welche das Erstellen, Verarbeiten und auch Extrahieren des Inhalts von PDF-Dokumenten ermöglicht.
- Erstellung des Suchindex:
  Die Inhalte, die extrahiert wurden, werden in einem Index gespeichert, welcher von Lucene.NET verwaltet wird. Dieser Index ermöglicht es, eine schnelle Suche nach Stichwörtern zu gewährleisten.
- Durchführung von Suchabfragen:
  Das Stichwort, welches vom Benutzer eingegeben wurde, wird mit dem Index abgeglichen. Lucene.NET bewertet die Relevanz von den Treffern und gibt die Ergebnisse zurück.

3.5.2 Implementierung der Suchfunktion

In der Klasse 'SearchAlgorithm' erfolgt die Implementierung der Suchfunktion. Hier die nötigen Schritte, die in dieser Klasse gemacht wurden, um die Suchfunktion lauffähig zu machen:

Initialisierung von Lucene.NET

Zur Textverarbeitung wird ein StandardAnalyzer verwendet und ein RAMDirectory, das den Index temporär im Arbeitsspeicher hält.

Listing 10: Initialisierung von Lucene.NET

```
1
2  private const LuceneVersion version = LuceneVersion.LUCENE_48;
3  private readonly StandardAnalyzer analyzer;
4  private readonly RAMDirectory directory;
5
6  public SearchAlgorithm()
7  {
8      analyzer = new StandardAnalyzer(version);
9      directory = new RAMDirectory();
10 }
```

Text aus PDF extrahieren

Wie bei der Architektur bereits erwähnt, muss der Text, bevor er in den Index aufgenommen werden kann, extrahiert werden. Das passiert beim Hochladen des PDFs in die Datenbank, da hier der Klartext des Dokuments gespeichert wird. Dafür wird in der 'DatabaseSQL'-Klasse eine Methode 'ExtractTextFromPdf' aufgerufen, bevor das PDF in die Datenbank gespeichert wird. Diese Methode sieht wie folgt aus:

Listing 11: PDF extrahieren

```
1
2  public string ExtractTextFromPdf(string pdfFilePath)
3  {
4      StringBuilder stringBuilder = new StringBuilder();
5
6      using (PdfReader reader = new PdfReader(pdfFilePath))
7      {
8          for (int i = 1; i <= reader.NumberOfPages; i++)
9          {
10             ITextExtractionStrategy strategy = new SimpleTextExtractionStrategy();
11             string text = PdfTextExtractor.GetTextFromPage(reader, i, strategy);
12             text = Encoding.UTF8.GetString(Encoding.Convert(Encoding.Default,
                   Encoding.UTF8, Encoding.Default.GetBytes(text)));
13             stringBuilder.Append(text);
14         }
15     }
16     return stringBuilder.ToString();
17 }
```

Mit diesem Code wird die PDF-Datei seitenweise eingelesen und mithilfe der iTextSharp-Bibliothek extrahiert. Dabei wird der Text zusätzlich von der Standardkodierung in UTF-8 umgewandelt, um Probleme mit Sonderzeichen zu vermeiden. Das Ergebnis der Methode ist ein String mit dem gesamten PDF-Inhalt.

Indizierung der PDF-Daten

Der extrahierte Text wird zusammen mit seinem Dateipfad und der PDF-ID, welche in der Datenbank gespeichert wird, in den Lucene.NET-Index aufgenommen. Dazu gibt es folgende

Methode, die für jedes PDF ein Lucene-Dokument erstellt, welches folgende Felder enthält:

Identifikationsnummer des PDFs

PdfPath: Speicherort der Datei

Content: Extrahierter Text

Listing 12: Indizierung der PDF-Daten

```
1
2  public void AddPdfsToIndex(List<PDF> pdfs)
3  {
4      using (var writer = new IndexWriter(directory, new IndexWriterConfig(version,
           analyzer)))
5      {
6          foreach (var pdf in pdfs)
7          {
8              var document = new Document();
9              document.Add(new StoredField("id", pdf.ID));
10             document.Add(new TextField("PdfPath", pdf.PDF_Path, Field.Store.YES));
11             document.Add(new TextField("Content", pdf.Content, Field.Store.YES));
12
13             writer.AddDocument(document);
14         }
15         writer.Commit();
16     }
17 }
```

Diese Informationen ermöglichen später eine gezielte Suche nach PDF-Dateien.

Durchführung der Suche

Die Methode 'SearchPdfs()' ist für das Durchsuchen des Indexes nach den eingegebenen Stichwörtern verantwortlich:

Listing 13: Durchführung der Suche

```
1
2  public List<PDF> SearchPdfs(string keyword, List<PDF> pdfs)
3  {
4      AddPdfsToIndex(pdfs); // Erst die PDFs in den Index aufnehmen
5
6      using (var directoryReader = DirectoryReader.Open(directory))
7      {
8          var indexSearcher = new IndexSearcher(directoryReader);
9          string[] fields = { "PdfPath", "Content" };
10         var queryParser = new MultiFieldQueryParser(version, fields, analyzer);
11
12         string escapedKeyword = QueryParser.Escape(keyword);
13         var query = queryParser.Parse(escapedKeyword);
14         var hits = indexSearcher.Search(query, 100).ScoreDocs;
15         List<PDF> myPdfs = new List<PDF>();
16
17         foreach (var hit in hits)
18         {
19             if (hit.Score > 0.001) // Nur relevante Ergebnisse zurueckgeben
20             {
21                 var document = indexSearcher.Doc(hit.Doc);
22                 int id = int.Parse(document.Get("id"));
23                 string pdfName = document.Get("PdfPath");
24                 string text = document.Get("Content");
25                 myPdfs.Add(new PDF(id, pdfName, text));
26             }
27         }
28         return myPdfs;
29     }
30 }
```

Diese Methode führt folgende Schritte aus:

- Erstellt eine Suchanfrage (queryParser.Parse(escapedKeyword)).
- Durchsucht die Felder PdfPath und Content nach Übereinstimmungen.
- Bewertet die Treffer anhand eines Relevanz-Scores.
- Gibt die Ergebnisse als Liste von PDF-Objekten zurück.

Der 'hit.Score' ist, wie der Name schon vermuten lässt, der Wert der Übereinstimmung von dem eingegebenen Wort und dem durchsuchten PDF. Ist also das Wort öfter in dem PDF enthalten, steigt der Score. Nur wenn das PDF den angegebenen Score übertrifft, wird es auch dem Benutzer angezeigt.

3.5.3 Vorteile der Lucene.NET-Integration

Hier einige Punkte, welche Vorteile von der Benutzung von Lucene.NET darstellen:

- Schnelle Suchabfragen durch Indexierung und optimierte Algorithmen.
- Effiziente Speicherverwaltung durch den Einsatz eines RAM-basierten Index.

Dank der Verwendung von Lucene.NET hat der PDF-Manager die Möglichkeit, eine performante und qualitativ hochwertige Ausgabe von PDFs für die gesuchten Stichwörter zu liefern.

3.6 Datenbank

Um im Projekt die benötigte Datenbank zur Verfügung zu stellen, ist die Auswahl auf einen Docker-Container, in welchen eine MSSQL-Datenbank läuft, gefallen. Gründe für diese Entscheidung sind die ressourcenschonende Art von Docker-Containern und die effiziente Möglichkeit, diese Datenbank isoliert und unabhängig vom Betriebssystem laufen zu lassen. Dazu wurde zuerst Docker-Desktop installiert und ein Container erstellt, in welchen das MSSQL-Image geladen wurde. So besteht jetzt die Möglichkeit, mit einem einfachen Klick auf den Startbutton des Containers die Datenbank zum Laufen zu bringen. Die Verbindung zur Datenbank erfolgt über einen SQL-Client wie zum Beispiel mit SQL Server Management Studio. So wurde eine gute, ressourcenschonende und sehr flexible Lösung geschaffen, eine Datenbank bereitzustellen, die man ganz einfach verwenden kann.

3.7 Datenbankstruktur KernelMemory

Für die Speicherung der Vektordaten von KernelMemory kommt das Plugin 'SqlServerMemoryDb' zum Einsatz. Dieses Plugin ist eine Erweiterung von KM und ermöglicht eine vollständige Speicherung der Daten in einer MSSQL-Datenbank.

3.7.1 'SqlServerMemoryDb'-Plugin

Um die Vektordaten zu speichern, verwendet KernelMemory das Plugin 'SqlServerMemoryDb'. Dieses NuGet-Paket generiert automatisch die gesamte Datenbankstruktur in MSSQL. Dabei wird sichergestellt, dass sämtliche Daten, also Vektoren, Metadaten und Weiteres, in einer strukturierten Form abgelegt werden.
Als Grundlage für weitere Erklärungen folgt hier eine grafische Darstellung der Datenstruktur:

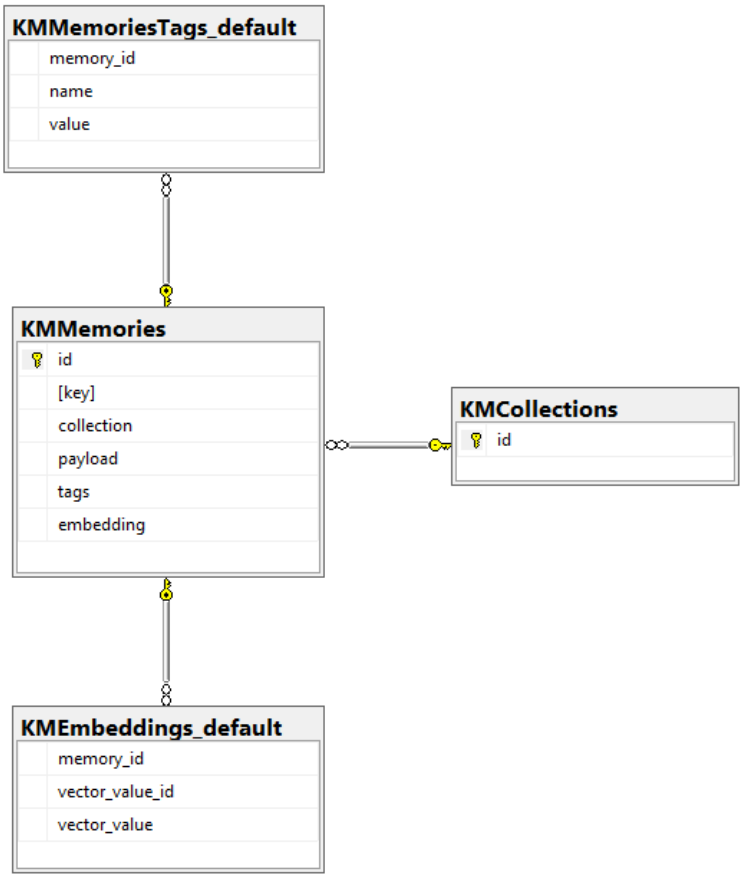

Abbildung 9: Datenbank Diagramm von KernelMemory Speicherung

3.7.2 KMMemories

Die Tabelle 'KMMemories' speichert die Hauptdaten innerhalb von KM. Sie besteht aus folgenden Feldern:

- ID (Primärschlüssel):
  Ist die eindeutige Erkennung der gespeicherten Memories.
- Key (Optional):
  Ein optionaler Schlüsselwert zur internen Identifikation
- Collection (Fremdschlüssel):
  Referenziert die KMCollection, in der das Memory gespeichert ist.
- Payload:
  Speichert Informationen oder Rohdaten, die mit dem Memory zu tun haben.
- Tags:
  Enthält Metadaten für weitere Kategorisierung
- Embedding:
  Hier wird speziell der Vektor gespeichert, von der aus KernelMemory die Informationen generiert.

3.7.3 KMCollections

Diese Tabelle ist dafür da, dass Memories in Sammlungen gespeichert werden. Es gibt in dieser Tabelle nur die Spalte ID. Der Standardwert für ID ist 'default' die sämtliche Memories beinhaltet.

3.7.4 KMMemoriesTags_default

Es ist speziell dazu da, um eine separate Metadaten-Tabelle für das jeweilige Memory bereitzustellen.
Es beinhaltet:

- Memory_ID (Fremdschlüssel):
  Stellt die Verknüpfung von einem Memory zu einem Tag sicher.
- Name:
  Speichert den Namen des Tags (z.B. ___file_type).

- Value:

  Speichert den Wert des Tags (z.B. application/pdf -> zugehörig zum Tag 'filetype').

	memory_id	name	value
1	DD74F606-A7A7-4E23-909E-3A1F4D134A51	__document_id	711afc2203544da88e26cbc4248602cc2024110702402732...
2	DD74F606-A7A7-4E23-909E-3A1F4D134A51	__file_id	dade274e33924ba8b230b339f2e9e7a9
3	DD74F606-A7A7-4E23-909E-3A1F4D134A51	__file_part	bb20a722b42a4a19a7bb291aa9b6e656
4	DD74F606-A7A7-4E23-909E-3A1F4D134A51	__file_type	application/pdf
5	DD74F606-A7A7-4E23-909E-3A1F4D134A51	__part_n	0
6	DD74F606-A7A7-4E23-909E-3A1F4D134A51	__sect_n	0

Abbildung 10: Beispiel-Eintrag für ein PDF-Tag

3.7.5 KMEmbeddings_default

Die KMEmbeddings-Tabelle speichert die Vektordaten, die für die semantische Suche in KernelMemory verwendet werden.
Sie besteht aus folgenden Spalten:

- Memory_ID (Fremdschlüssel):

  Verknüpft einen Vektor mit einem Memory.

- Vector_Value_ID:

  Speicherung der ID von einem Wert.

- Vector_Value:

  Der eigentliche numerische Wert eines Vektors.

	memory_id	vector_value_id	vector_value
1	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3223	0,011265863
2	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3222	-0,0053338083
3	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3221	-0,004733075
4	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3220	-0,026235182
5	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3219	-0,007257969
6	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3218	0,01719519
7	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3217	-0,0049402164
8	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3216	-0,016336428
9	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3215	-0,010144866
10	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3214	-0,007790631
11	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3213	-0,0034201487
12	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3212	0,013212583
13	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3211	0,00069918507
14	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3210	0,0019388073
15	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3209	0,006352048
16	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3208	0,0019965249
17	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3207	0,03183092
18	DD74F606-A7A7-4E23-909E-3A1F4D134A51	3206	0,0060003838

Abbildung 11: Vektoren in der Datenbank

3.7.6 Beziehungen zwischen den Tabellen

Jede Tabelle steht mit einem bestimmten Fremdschlüssel in Beziehung zueinander.

KMMemories -> KMCollections:
Jedes Memory gehört genau zu einer Sammlung. Eine Sammlung hat mehrere Memories. Der Fremdschlüssel ist hier 'collection'.

KMMemoriesTags_default -> KMMemories:
Jedem Memory können mehrere Tags zugewiesen werden. So entsteht eine 1:N-Beziehung zwischen KMMemories und KMMemoriesTags_default.

KMEmbeddings_default -> KMMemories:
Jeder Vektor ist einem spezifischen Memory zugeteilt. Speziell diese Verknüpfung ermöglicht die Nutzung von Vektorsuche in KernelMemory.

3.8 Datenbankstruktur der Chat-Speicherung

Nach einer Unterhaltung mit dem Chatbot wird der Chat grundsätzlich nicht gespeichert. Deshalb kommt eine neue Datenbank ins Spiel, die jede Antwort, jede Unterhaltung sowie auch jeden Benutzer speichert, um später darauf wieder zugreifen zu können. Im nachfolgenden Datenbank-Diagramm wird die Struktur der Datenbank gezeigt. Dieses Diagramm ist notwendig, um die weitere Erklärung der Tabellen besser zu verstehen:

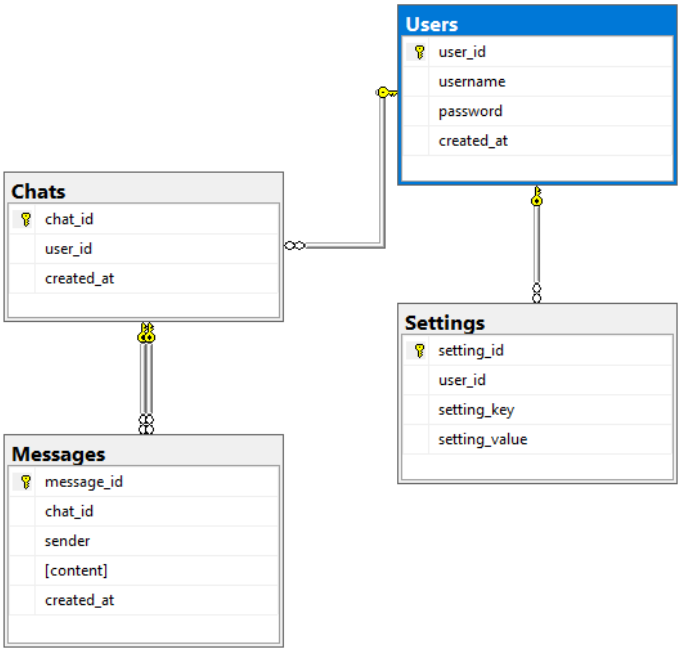

Abbildung 12: Datenbank Diagramm von Chat-Speicherung

3.8.1 Settings

Die Settings-Tabelle dient zur Speicherung individueller Benutzereinstellungen.

Spalte	Typ	Beschreibung
setting_id	INT (Primärschlüssel)	Eindeutige ID für die Einstellung
user_id	INT (Fremdschlüssel)	Verweist auf den Benutzer, dem die Einstellung gehört
setting_key	VARCHAR	Schlüssel der Einstellung z.B. ein Theme
setting_value	TEXT	Wert des Einstellungsschlüssels z.B. von Theme 'dark'

Tabelle 1: Tabellarische Ansicht von Settings

Die Verbindung zur Users-Tabelle stellt sicher, dass jede Einstellung einem bestimmten Benutzer zugeordnet werden kann.

3.8.2 Users

Die Users-Tabelle verwaltet alle registrierten Benutzer. Jeder User hat eine eindeutige ID sowie eigene Zugangsdaten.

Spalte	Typ	Beschreibung
user_id	INT (Primärschlüssel)	Eindeutige ID für den User
username	VARCHAR	Benutzername des Users
password	VARCHAR	Passwort des Users
created_at	TIMESTAMP	Datum der Registrierung

Tabelle 2: Tabellarische Ansicht von Users

Diese Tabelle ist das Herzstück der Datenbank. Sie steht mit fast allen anderen Tabellen in Verbindung.

3.8.3 Chats

Die Chats-Tabelle speichert die einzelnen Chat-Sitzungen. Ein Benutzer kann mehrere Chats haben, wobei jede Unterhaltung eine eindeutige ID erhält.

Spalte	Typ	Beschreibung
chat_id	INT (Primärschlüssel)	Eindeutige ID des Chats
user_id	INT (Fremdschlüssel)	Veweist auf den Benutzer, der den Chat führt
created_at	TIMESTAMP	Zeitpunkt der Erstellung des Chats

Tabelle 3: Tabellarische Ansicht von Chats

Dies ermöglicht es, einzelne Konversationen voneinander zu trennen und gezielt abzurufen.

3.8.4 Messages

Die Messages-Tabelle speichert sämtliche Nachrichten innerhalb eines Chats. Jede Nachricht, egal ob vom Benutzer oder vom Chatbot, gehört zu einer bestimmten Unterhaltung.

Spalte	Typ	Beschreibung
message_id	INT (Primärschlüssel)	Eindeutige ID der Nachricht
chat_id	INT (Fremdschlüssel)	Veweist auf den zugehörigen Chat
sender	VARCHAR	Identifiziert den Absender der Nachricht (User/Bot)
content	TEXT	Der Inhalt der Nachricht
created_at	TIMESTAMP	Zeitpunkt der Erstellung der Nachricht

Tabelle 4: Tabellarische Ansicht von Messages

Durch die Verbindung zur Chats-Tabelle kann somit jede Nachricht einem Chat zugeordnet werden.

3.8.5 Beziehung zu den Tabellen

Die Beziehung zwischen den Tabellen ist folgendermaßen aufgebaut:

- Users -> Chats (1:N):
  Ein User kann mehrere Chats enthalten.
- Chats -> Messages (1:N):
  Ein Chat kann mehrere Nachrichten enthalten.
- Settings -> Users (1:N):
  Ein Setting können mehrere Benutzer haben.

3.9 Datenbankstruktur PDF-Manager

Die Datenbankstruktur basiert auf einer relationalen Datenbank und besteht aus mehreren Tabellen, die die verschiedenen Entitäten des Produkts repräsentieren.

3.9.1 PDF-Manager

Um eine effiziente Datenhaltung der wichtigsten Informationen sicherzustellen, wurde die Struktur der Daten, die vom Projekt übernommen wurde, grundlegend geändert. Es werden sinnvolle Metadaten, die wichtigsten Informationen zu den PDFs und Benutzerinformationen gespeichert. Die PDFs selbst werden in einem lokalen Dateisystem gespeichert, um für den Anfang bessere Performance beim Zugriff zu erhalten und um die Datenbank anfangs nicht unnötig aufzublähen und somit Tests besser durchführen zu können. Zur Veranschaulichung und zur Grundlage für weitere Erklärungen ist hier eine grafische Darstellung der Datenbankstruktur:

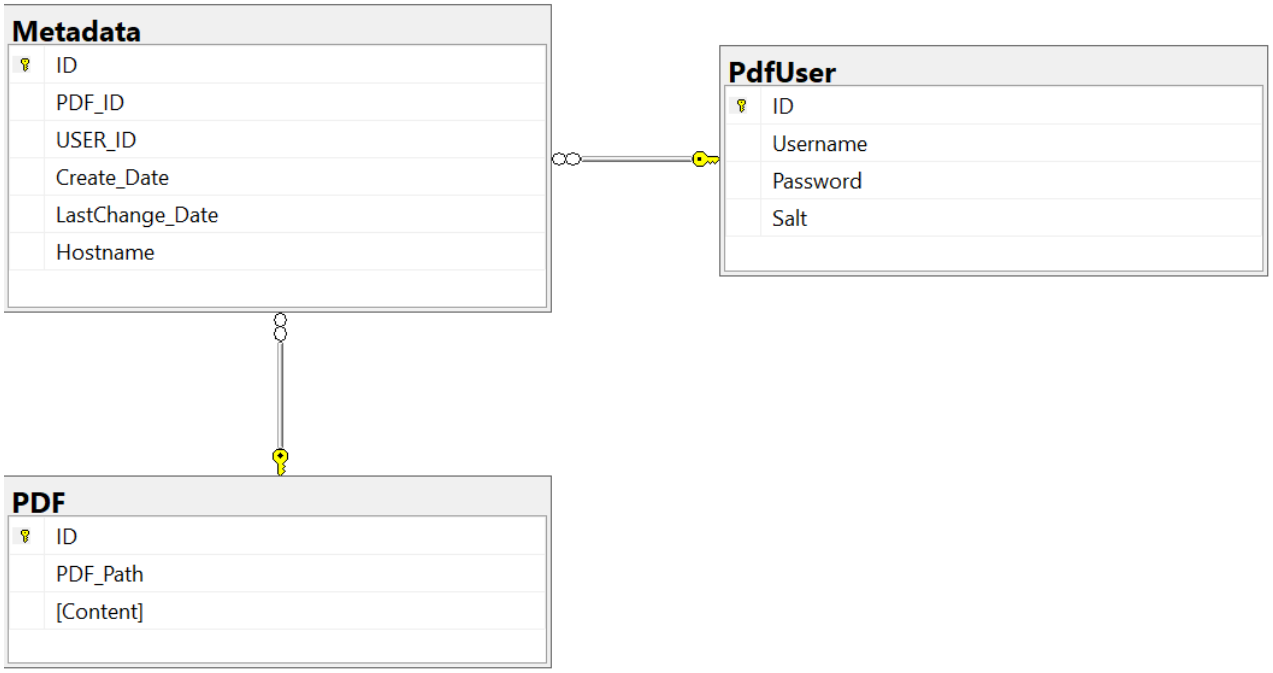

Abbildung 13: Datenbank Diagramm von PDF-Manager

3.9.2 PDF

Die Tabelle PDF speichert Informationen zu den hochgeladenen PDF-Dateien. Sie besteht aus folgenden Feldern:

- ID (Primärschlüssel): Dient zur eindeutigen Kennung der PDFs
- PDF_Path: Speichert den Speicherort der Dateien in dem Dateisystem
- Content: Hier können extrahierte Textinhalte der PDFs gespeichert werden.

3.9.3 Metadata

Diese Tabelle speichert sinnvolle Metadaten zu den PDFs. Sie besteht aus folgenden Feldern:

- ID (Primärschlüssel): Dient zur eindeutigen Kennung der Metadaten
- PDF_ID (Fremdschlüssel auf PDF.ID): Stellt die Verknüpfung zwischen der PDF-Datei und der dazugehörigen Metadatentabelle sicher.
- USER_ID (Fremdschlüssel auf PdfUser.ID): Speichert den Benutzer, der das PDF hochgeladen hat.
- Create_Date: Speichert das Datum, an dem das PDF in den PDF-Manager hochgeladen wurde.

- LastChange_Date: Speichert das Datum an dem das PDF als Letztes geändert wurde, derzeit noch nicht relevant weil man das PDF nicht ändern kann (für die Zukunft).
- Hostname: Hier wird der Hostname des Computers, mit welchen man das PDF hochlädt, gespeichert.

3.9.4 PdfUser

Diese Tabelle speichert Benutzerinformationen. So wird die Erweiterbarkeit des Produkts erleichtert, da schon vorgesehen ist, dass später Benutzer für den PDF-Manager und/oder für den Chatbot folgen könnten.

- ID (Primärschlüssel): Dient zur eindeutigen Kennung der User
- Username: Hier wird der Name des Benutzers abgelegt.
- Password: Enthält das Passwort für den Benutzer.
- Salt: Speichert den Salt-Wert zu jedem Passwort, was die Sicherheit der Passwortverschlüsselung erhöht.

3.9.5 Beziehungen zwischen den Tabellen

- Metadata –> PDF: Jeder Metadatensatz gehört genau zu einer PDF-Datei. So wie eine PDF-Datei nur einen Metadatensatz hat. Die Beziehung wird durch den Fremdschlüssel PDF_ID hergestellt.
- Metadata –> PdfUser: In den Metadaten wird die USER_ID erfasst, um festzuhalten, welcher Benutzer, welche PDF-Datei erstellt oder geändert hat.

3.10 Datenhaltung

3.10.1 Chatbot

Die Datenhaltung für den Chatbot basiert auf einer relationalen Datenbank. Auf diese Datenbank wird mittels ADO.NET zugegriffen, sprich mit manuell geschriebenen SQL-Befehlen. Anders als beim PDF-Manager setzt der Chatbot nicht auf das ORM-Framework 'Entity Framework', sondern auf einen manuell gespeicherten Connection-String. Dies ermöglicht flexible Kontrolle über eigene SQL-Befehle.

Aufbau der Datenhaltung

Die Speicherung wird von einer eigenen Klasse namens 'DbService' übernommen. Diese Klasse beinhaltet sämtliche Funktionen, welche mit der Datenspeicherung der Chats zu tun haben. Außerdem implementiert 'DbService' das Interface 'IDbService', welches folgende Methoden bietet:

Methode	Beschreibung
SaveMessagesToDatabase()	Speichert eine Nachricht vom Chat
CreateNewChat()	Erstellt einen neuen Chat in der Datenbank
LoadSessionsFromDatabase()	Lädt alle Unterhaltungen von einem Benutzer
LoadMessagesFromDatabase()	Lädt alle Nachrichten von einer Unterhaltung
DeleteSessionFromDatabase()	Löscht eine Unterhaltung mit allen Nachrichten
DeleteChatsWithNoMessages()	Löscht alle leeren Chats (Bereinigung)

Tabelle 5: Methoden von IDbService

Verbindung zur Datenbank

Die Klasse 'DbService' wird über Dependency Injection mit der Konfiguration initialisiert. Der Connection-String wird direkt aus der appsettings.json-Datei geladen.

Listing 14: Connection-String aus appsettings.json laden

```
1  private readonly IConfiguration _configuration;
2  private readonly string _connectionString;
3
4  public DbService(IConfiguration configuration)
5  {
6      _configuration = configuration;
7      _connectionString = _configuration["Variables:ConnectionString"];
8  }
```

Speicherung von Nachrichten

Die Methode 'SaveMessageToDatabase()' speichert eine einzelne Nachricht in der Tabelle 'Messages'. Dazu wird eine SQL-Verbindung über den geladenen Connection-String geöffnet. Danach

schreibt ein Insert-Befehl die Nachricht in die Datenbank.

Listing 15: Methode zur Speicherung von einer Nachricht

```
using (SqlConnection connection = new SqlConnection(_connectionString))
{
    await connection.OpenAsync();

    string insertQuery = "INSERT INTO Messages (chat_id, sender, content, created_at)
        VALUES (@chatId, @sender, @content, @createdAt);";
    SqlCommand command = new SqlCommand(insertQuery, connection);
    command.Parameters.AddWithValue("@chatId", chatId);
    command.Parameters.AddWithValue("@sender", message.Sender.ToString());
    command.Parameters.AddWithValue("@content", message.Content);
    command.Parameters.AddWithValue("@createdAt", message.CreatedAt);
    await command.ExecuteNonQueryAsync();
    Console.WriteLine($"Message erfolgreich gespeichert: {message.Content}");
}
```

Erstellung eines neuen Chats

Mit der Methode 'CreateNewChat()' wird ein neuer Chat für einen Benutzer erstellt.

Listing 16: Methode zur Erstellung eines neuen Chats

```
using (SqlConnection connection = new SqlConnection(_connectionString))
{
    connection.Open();
    // Chat erstellen
    string insertChatQuery = "INSERT INTO Chats (user_id, created_at) VALUES (@userId,
        @createdAt); SELECT SCOPE_IDENTITY();";
    SqlCommand insertChatCommand = new SqlCommand(insertChatQuery, connection);
    insertChatCommand.Parameters.AddWithValue("@userId", userId);
    insertChatCommand.Parameters.AddWithValue("@createdAt", DateTime.Now);
    // ChatID returnen
    return Convert.ToInt32(insertChatCommand.ExecuteScalar());
}
```

Sollte der Benutzer in der Datenbank noch nicht enthalten sein, so wird automatisch in der Tabelle 'Users' ein neuer Benutzer erstellt. Dabei wird der 'IDENTITY_INSERT' verwendet, um eine manuelle 'user_id' setzen zu können, da in der SYSco-Shell bereits jeder eine fixe ID besitzt.

Listing 17: Überprüfung ob der Benutzer bereits vorhanden ist und Verwendung von IDENTITY_INSERT

```
string checkUserQuery = "SELECT COUNT(*) FROM Users WHERE user_id = @userId";
SqlCommand checkUserCommand = new SqlCommand(checkUserQuery, connection);
checkUserCommand.Parameters.AddWithValue("@userId", userId);
int userCount = (int)checkUserCommand.ExecuteScalar();
if (userCount == 0)
{
    // Setze IDENTITY_INSERT auf ON
    string setIdentityInsertOn = "SET IDENTITY_INSERT Users ON";
    SqlCommand setIdentityInsertOnCommand = new SqlCommand(setIdentityInsertOn,
        connection);
    setIdentityInsertOnCommand.ExecuteNonQuery();
}
```

Laden von sämtlichen Unterhaltungen eines Benutzers

Über 'LoadSessionsFromDatabase()' werden alle gespeicherten Chats von einem Benutzer über die 'user_id' geladen. Speziell für die Benutzeroberfläche vom Chatbot wird für jeden Chat die erste Nachricht übergeben, die dann in der linken Spalte unter 'Gespeicherte Chats' erscheint. Dieses Feature ist dafür gedacht, dass die Auflistung der vergangenen Chats der Struktur von OpenAI ChatGPT ähnelt.

Listing 18: Laden von erster Nachricht aus Datenbank

```
while (reader.Read())
{
    int chatId = Convert.ToInt32(reader["chat_id"]);
    string firstMessage = reader["first_message"] != DBNull.Value ?
        reader["first_message"].ToString() : "No messages yet";
    sessions.Add(new SavedChats(chatId, firstMessage));
}
```

Wie man am Code-Beispiel erkennen kann, werden nur die ChatIDs und die erste Nachricht in ein eigens definiertes Modell gespeichert und danach in eine 'sessions'-Liste. Somit zeigt die Benutzeroberfläche auch nur die erste Nachricht an. Mithilfe der ChatID kann der Benutzer dann denjenigen Chat von der Datenbank laden, den er benötigt.

Laden der Nachrichten

Wenn der Benutzer nun auf denjenigen Chat klickt, den er braucht, wird die 'LoadMessagesFromDatabase()'-Methode aufgerufen. Mithilfe der ChatID kann so in der Datenbank geprüft werden, welche Nachrichten zu dieser ID gehören.

Listing 19: Laden von sämtlichen Nachrichten aus einer Session

```
string selectQuery = "SELECT sender, content, created_at FROM Messages WHERE chat_id =
    @chatId;";
```

Löschen von Chats

Die Methode 'DeleteSessionFromDatabase()' wird dann gebraucht, wenn der Benutzer einen bestimmten Chat löschen will. Das heißt, es werden in der Datenbank zuerst sämtliche Nach-

richten, die auf den Chat verweisen, gelöscht, und danach wird der Chat selbst gelöscht.

Listing 20: Löschen aller Nachrichten eines Chats + Chat selbst

```
// First, delete all messages related to the chat
string deleteMessagesQuery = "DELETE FROM Messages WHERE chat_id = @chatId";
using (SqlCommand deleteMessagesCommand = new SqlCommand(deleteMessagesQuery, connection,
    transaction))
{
    deleteMessagesCommand.Parameters.AddWithValue("@chatId", chatId);
    await deleteMessagesCommand.ExecuteNonQueryAsync();
}

// Then, delete the chat itself
string deleteChatQuery = "DELETE FROM Chats WHERE chat_id = @chatId";
using (SqlCommand deleteChatCommand = new SqlCommand(deleteChatQuery, connection,
    transaction))
{
    deleteChatCommand.Parameters.AddWithValue("@chatId", chatId);
    await deleteChatCommand.ExecuteNonQueryAsync();
}
```

3.10.2 PDF-Manager

Um im Backend mit den PDFs, Metadaten und den Usern arbeiten zu können, bei Übergaben oder Sonstigem, wurde für jedes oben genannte Objekt eine Modellklasse erstellt, die genau die Informationen hält, die auch in der Datenbank gespeichert werden. Hier als Beispiel die PDF-Modellklasse:

Listing 21: PDF-Modellklasse

```
public class PDF
{
    public int ID { get; set; }
    public string? PDF_Path { get; set; }
    public string? Content { get; set; }

    public PDF(string pdfPath, string text)
    {
        PDF_Path = pdfPath;
        Content = text;
    }

    public PDF(int id, string pdfPath, string text)
    {
        ID = id;
        PDF_Path = pdfPath;
        Content = text;
    }

    public PDF()
    {

    }
}
```

Zusätzlich zu den 3 Standard-Modellklassen gibt es eine 'PdfSendable'-Modellklasse, welche alle Daten enthält, die bei der Übergabe von PDFs zwischen Backend und Frontend benötigt werden. Diese Klasse enthält folgende Werte:

- String PdfName
- String Base64String

- Metadata Metadata

Um eine Verbindung vom Code zur Datenbank zu schaffen, wurde Entity Framework in Kombination mit einer Kontext-Klasse verwendet. Die 'PdfContext-Klasse' ist der DbContext von Entity Framework Core in diesem Projekt. Diese Klasse hält die drei verschiedenen Tabellen und ihre Attribute, um sie in der Datenbank speichern zu können.
Mit DbSet<> werden die Entitäten definiert, also die Tabellen für den Datenbankzugriff:

Listing 22: Entitäten definieren

```
// DbSets fuer die Entitaeten
public virtual DbSet<PdfUser> PdfUsers { get; set; }
public virtual DbSet<PDF> PDFs { get; set; }
public virtual DbSet<Metadata> Metadata { get; set; }
```

Datenbankschema definieren

Mit den 'OnModelCreating()'-Methoden wird das Datenbankschema konfiguriert, sprich, wie die Tabellen und Spalten später in der Datenbank aussehen. Dafür wurde folgender Code für alle drei Tabellen verfasst:

Listing 23: Datenschema konfigurieren

```
modelBuilder.Entity<Metadata>(entity =>
{
    entity.HasKey(e => e.ID);
    entity.ToTable("Metadata");
    entity.Property(e => e.ID).HasColumnName("ID");
    entity.Property(e => e.PDF_ID).HasColumnName("PDF_ID");
    entity.Property(e => e.USER_ID).HasColumnName("USER_ID");
    entity.Property(e => e.Create_Date).HasColumnName("Create_Date");
    entity.Property(e => e.LastChange_Date).HasColumnName("LastChange_Date");
    entity.Property(e => e.Hostname)
        .IsRequired()
        .HasColumnName("Hostname");
```

Fremdschlüsselbeziehungen

Um die Tabellen untereinander zu verknüpfen, wurden Fremdschlüssel definiert. In Entity Framework verwendet man dafür die 'OnModelCreating()'-Methode, welche in der DbContext-Klasse programmiert wurde:

Listing 24: Beziehungen

```
modelBuilder.Entity<Metadata>(entity =>
{
    entity.HasKey(e => e.ID);
    entity.ToTable("Metadata");
    entity.Property(e => e.PDF_ID).HasColumnName("PDF_ID");
    entity.Property(e => e.USER_ID).HasColumnName("USER_ID");

    entity.HasOne<PdfUser>()
        .WithMany()
        .HasForeignKey(e => e.USER_ID)
        .OnDelete(DeleteBehavior.Cascade);

    entity.HasOne<PDF>()
        .WithMany()
        .HasForeignKey(e => e.PDF_ID)
        .OnDelete(DeleteBehavior.Cascade);
});
```

Mit diesem Code wird sichergestellt, dass ein Benutzer mehrere PDFs besitzen kann, dafür wird eine 1:N-Beziehung verwendet. Weiters wird festgelegt, dass beim Löschen eines PDFs oder Benutzers die zugehörigen Metadaten auch automatisch entfernt werden.

Datenbankservice

Eine Klasse namens 'DatabaseSql' ist eine Datenbankservice-Klasse, welche für die Verwaltung von PDF-Dokumenten dient. Sie bietet die Möglichkeit, PDFs und ihre Metadaten hinzuzufügen, abzurufen, zu aktualisieren und zu löschen. Von diesen Funktionen machen sich die restlichen Klassen des Projekts Gebrauch, die auf Benutzereingaben reagieren, zum Beispiel wenn ein Benutzer ein neues PDF in der GUI hochladen, öffnen oder löschen will. Ebenfalls wird hier die SignalR-Verbindung für die Übergabe der PDFs an den Chatbot, wenn eines in den PDF-Manager hochgeladen wird, realisiert. Generell läuft das Hochladen eines Dokuments in die Datenbank wie folgt im Code ab:

Listing 25: Pfad festlegen

```
string pdfFilePath = System.IO.Path.Combine(PdfDirectory, pdfSendable.PdfName);
```

Hier wird zuerst der Pfad, wo das PDF abgelegt wird, festgelegt. Denn die PDFs selbst werden in diesem Projekt in das lokale Dateisystem gespeichert. Dann folgt eine Prüfung, ob das PDF bereits existiert, um keine doppelte Speicherung vornehmen zu können.

Listing 26: Base64 String in Datei umwandeln

```
byte[] pdfByte = Convert.FromBase64String(pdfSendable.Base64String);
File.WriteAllBytes(pdfFilePath, pdfByte);
```

Der Base64-String, der für die Übergabe von Frontend zum Backend benutzt wurde, wird in diesem Code wieder in eine Datei umgewandelt. Darunter wird dann das PDF in das Dateisystem gespeichert. Jetzt ist alles soweit fertig, um die Speicherung in die Datenbank vorzunehmen:

Listing 27: PDF in Datenbank hochladen

```
PDF pdf = new PDF(pdfFilePath, ExtractTextFromPdf(pdfFilePath));
this.pdfContext.PDFs.Add(pdf);
await this.pdfContext.SaveChangesAsync();
```

Mit diesem Code wird der Pfad des PDFs mit dem extrahierten Text von dem Dokument in die Datenbank gespeichert. Das Gleiche passiert auch noch mit den Metadaten von einem PDF und dann ist das Hochladen auch schon erledigt.

Schnittstelle zum Frontend

Dass das Frontend vom PDF-Manager die Möglichkeit hat, Dokumente vom Backend abzurufen oder welche zu speichern, bietet der Server eine Schnittstelle, die dies zulässt. Hierzu wurde eine Klasse namens 'PdfController' verwendet, welche einen REST-API-Controller darstellt, der verschiedene Endpunkte für den Zugriff auf die in der Datenbank gespeicherten PDFs oder eine Schnittstelle für das Hochladen bereitstellt. Der Controller benutzt dafür die 'DatabaseSql'-Klasse, welche die benötigten Funktionen bietet. Hierzu wurde die Klasse als API-Controller markiert und über die URL + /PDF erreichbar gemacht:

Listing 28: API-Controller

```
[ApiController]
[Route("[controller]")]
public class PdfController : ControllerBase
```

Um auf die PDFs zugreifen zu können, wurden Funktionen für die API-Endpunkte programmiert und diese dann auf bestimmte Pfade der API gesetzt. Das sieht für das Abrufen aller PDFs so aus:

Listing 29: Alle PDFs abrufen

```
[HttpGet("getAllPdfs")]
public async Task<ActionResult<List<PDF>>> GetAllPdfs()
```

Oben wird der Pfad angegeben und in der zweiten Zeile werden alle PDFs aus der Datenbank abgerufen und als Liste zurückgegeben.
Alle anderen Funktionen haben genau den gleichen Aufbau, rufen jedoch immer andere Funktionen auf, die mit der Datenbank kommunizieren und haben natürlich immer eine andere URL, womit man sie aufrufen kann. Zum Beispiel sieht man im folgenden Code die Schnittstelle für das Hochladen der PDFs:

Listing 30: PDF hochladen

```
[HttpPost("Add")]
public async Task<ActionResult<PdfSendable>> AddPdf([FromBody] PdfSendable pdfSendable)
```

Hier wird ein JSON-Objekt als PdfSendable, ein Objekt, welches die wichtigsten Werte der PDFs hält, vom Frontend erwartet. Dieses Objekt wird dann mit der 'AddToDatabase'-Methode, die

sich in der 'DatabaseSql'-Klasse befindet, in die Datenbank gespeichert und bei Erfolg bekommt man einen '200-OK'-Statuscode zurück.

Swagger

In dem Projekt (PDF-Manager) wird Swagger verwendet, um die REST-API zu testen und zu dokumentieren. Dadurch ist es möglich, beim Starten des Backends auch ohne Frontend auf alle Funktionen der REST-API zuzugreifen, und somit wird das Testen der Funktionalität des Backends erleichtert. Um Swagger zu implementieren, sind einige Schritte notwendig, angefangen mit der 'Program.cs'-Klasse, wo man Swagger hinzufügen muss:

Listing 31: Swagger hinzufügen

```
1 // Swagger/OpenAPI hinzufuegen
2 builder.Services.AddEndpointsApiExplorer();
3 builder.Services.AddSwaggerGen();
```

Danach wird Swagger als Middleware in der API aktiviert und eine Weiterleitung zu Swagger angelegt, sodass der Benutzer des Backends beim Starten auf die Swagger-UI weitergeleitet wird. Wenn man den Server startet, wird diese Swagger-UI im Browser geöffnet, was so aussieht:

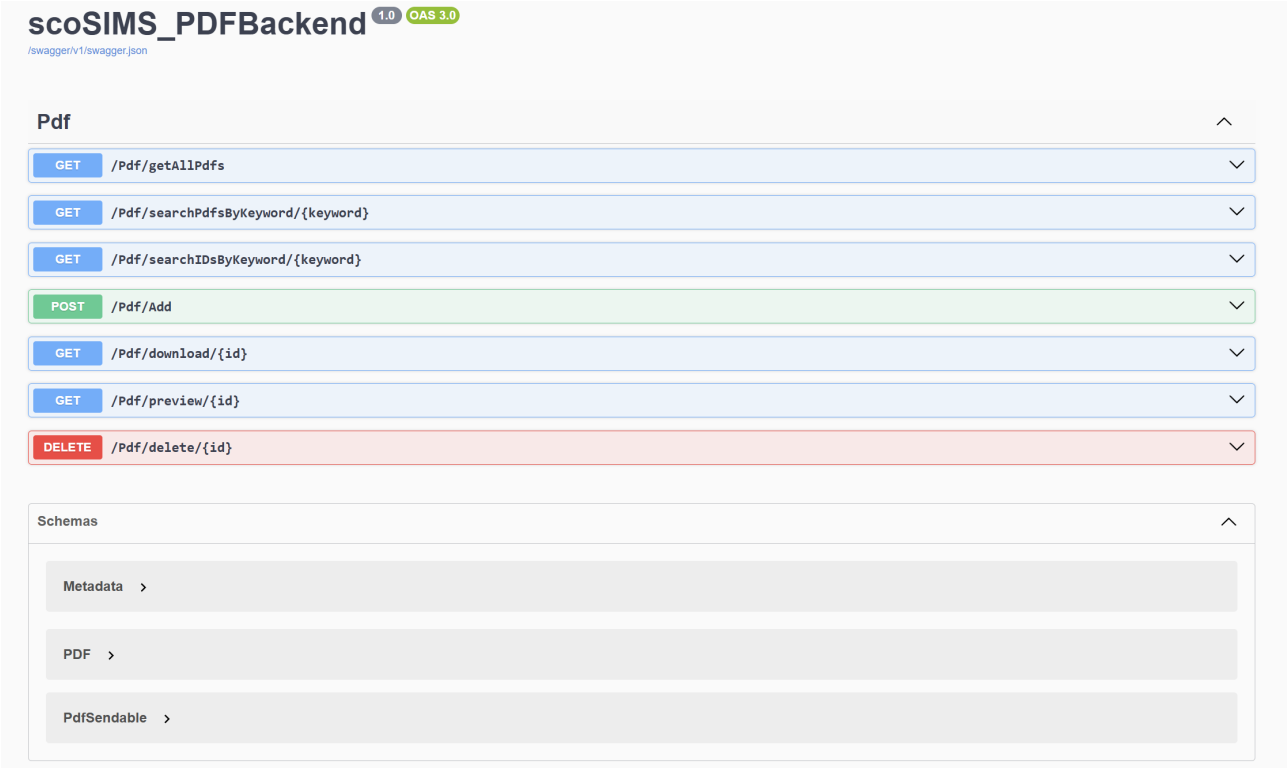

Abbildung 14: SwaggerUI

Hier sind alle Funktionen aufgelistet und darunter das Schema der Datenbank.
Wenn man jetzt eine Methode aufklappt, kann man die Funktion ausführen und darunter sieht man dann die Ausgabe:

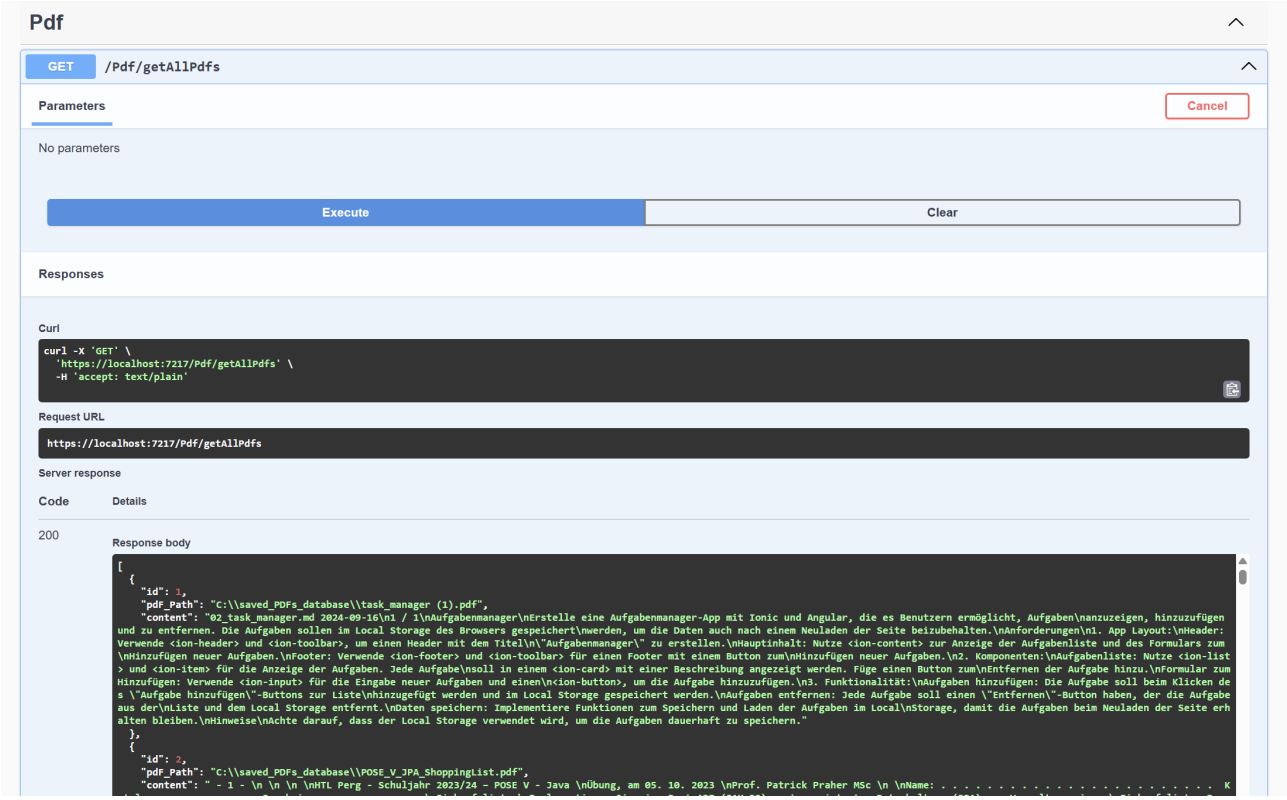

Abbildung 15: SwaggerUITryItOut

3.11 Benutzeroberfläche - UI

3.11.1 Chatbot

Das Frontend für den Chatbot wurde rein in HTML, JavaScript sowie auch CSS programmiert. Diese drei Komponenten werden in einer WebView2 als Browser-Ansicht angezeigt. WebView2 wird wiederum in eine WPF-Anwendung implementiert.

Windows Presentation Foundation (WPF)

WPF wird grundlegend für das Anzeigen eines Fensters verwendet. Es stellt die reine Anzeigekomponente dar, in der die eigentliche Anwendung implementiert ist. Dieser Teil hat somit keine weitere Aufgabe. Die Projektstruktur sieht dann wie folgt aus:

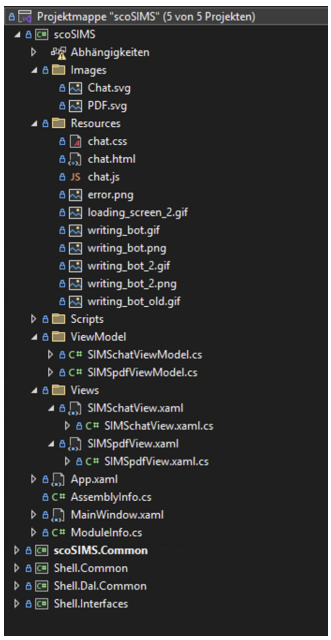

Abbildung 16: Projektstruktur von dem WPF-Frontend

Microsoft Edge WebView2

Das WebView2-Modul stellt hier eine Schnittstelle zwischen WPF und HTML dar. Durch diese Technologie ist es möglich, die Webanwendung, die in HTML und JS geschrieben ist, in einem WPF-Fenster unter Windows anzuzeigen. Wie man im nachstehenden Code-Snippet erkennen kann, ist hierfür nur eine einzige Zeile Code nötig, um WebView2 zu implementieren:

Listing 32: WebView2 Implementierung

```
<UserControl x:Class="scoSIMS.Views.SIMSchatView"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:wv="clr-namespace:Microsoft.Web.WebView2.Wpf;assembly=Microsoft
                       .Web.WebView2.Wpf"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <wv:WebView2 x:Name="MyWebView" VerticalAlignment="Stretch"
            HorizontalAlignment="Stretch"/>
    </Grid>
</UserControl>
```

Damit die Dateien HTML, JavaScript und CSS ebenso implementiert werden, müssen in der Logik von dem xaml-Frontend noch die Elemente mithilfe der WebView2InitializationCompleted() Methode initialisiert werden. Der JavaScript- und CSS-Inhalt wird als String direkt in den HTML-Code injiziert. Danach wird die HTML-Datei an das WebView2 übergeben. Somit ist in der HTML-Datei automatisch als 'script' der JavaScript-Code eingefügt sowie auch bei 'style' der CSS-Code. Im Prinzip wird der bestehende Code vom HTML einfach durch einen neuen ersetzt, indem danach das JS und CSS enthalten sind.

Web Applikation

Die Web-Applikation beinhaltet 3 Elemente: HTML, JavaScript und CSS. HTML steht für die View, die der Benutzer sieht. Das JavaScript ist die gesamte Logik hinter der View. Das CSS beinhaltet das gesamte Design von der View (HTML).

HTML

In der View wird Folgendes implementiert:
Für die Eingabe eines Prompts ist ein Eingabefeld mit einem 'Send'-Button eingepflegt. Hier wird die Nachricht für den KernelMemory eingegeben und versendet.
Über dem Eingabefeld werden dann die Messages angezeigt. Grundsätzlich aber nur die Nachrichten, die von einer Sitzung stammen.
Um auf ältere geschriebene Chats zurückzugreifen, gibt es auf der linken Seite der View eine Seitenleiste mit einer Liste der letzten Chats, die man ebenfalls aufrufen kann, um sie durchzulesen. Bei einem alten Chat weiterschreiben ist nicht möglich, dafür wird auch das Eingabefeld blockiert, um das zu vermeiden.

JavaScript

Das JavaScript beschreibt nun sämtliche Funktionen hinter der View. Es wird hauptsächlich für die Kommunikation verwendet, die Nachrichten von der View ins Backend bringt und umgekehrt.
Nun eine genaue Beschreibung der Implementierung:
Die Hauptaufgabe von dieser Komponente ist die Datenübermittlung mithilfe von SignalR. Das heißt, dieser Teil stellt den Client für die SignalR-Verbindung zum Backend dar. Somit verschickt der Client die vom User eingegebene Nachricht an das Backend. Wenn das Backend nun eine passende Antwort generiert hat, sendet das Backend eine Nachricht zurück, die der Client an die View weitergibt, um sie anzuzeigen.
Weiters werden ebenso Funktionen ausgeführt, die ebenfalls das SignalR-Protokoll verwenden. Wenn ein neuer Chat erstellt wird, wird automatisch ein Befehl an das Backend versendet, das daraufhin einen neuen Eintrag für einen neuen Chat erstellt.

3.11.2 PDF-Manager

Die Benutzeroberfläche des PDF-Managers wurde mit Windows Presentation Foundation, kurz gesagt WPF, entwickelt und stellt eine moderne Gestaltung für den Benutzer dar, die es ihm ermöglicht, PDFs einfach und effizient zu verwalten und mittels Stichwörtern nach ihnen zu

suchen. Die Funktionen, welche die Oberfläche bietet, wurden wie folgt in dem UserInterface dargestellt und funktionsfähig gemacht:

XAML

Die Buttons werden in einer 'SIMSpdfView.xaml'-Klasse dargestellt, wo jeder Button DevExpress verwendet und wie folgt programmiert wurde:

Listing 33: StrukturButtons

```
1  <dx:SimpleButton Grid.Row="2" Grid.Column="2" Content="PDF hochladen" MinWidth="200"
       Height="30"
2                   Click="uploadButton_Click" Margin="10" VerticalAlignment="Top"
3                   Style="{StaticResource BlueButtonStyle}"/>
```

Angezeigt werden sie in einem einheitlichen Design:

Abbildung 17: Buttons

Das beim Klicken auf einen Button auch etwas passiert, wird bei 'Click=' eine Funktion der Code-Behind-Klasse (SIMSpdfView.xaml.cs) aufgerufen, die die angegebene Funktion ausführt. Weiters besteht die Möglichkeit, dass der Benutzer ein PDF auswählt, welches er hochladen möchte, indem er die Möglichkeit bekommt, im Explorer ein PDF auszuwählen oder eine Datei in den Drag-and-Drop-Bereich zu ziehen.

Abbildung 18: Upload-PDF

Dieses Feld reagiert, wenn der Benutzer ein PDF in den Bereich zieht, indem es im Code-Behind eine Methode aufruft, die das PDF als ausgewählt markiert. Beim Klicken auf den Upload-Button wird das ausgewählte PDF mithilfe des Code-Behind, der eine Verbindung zum Backend aufbaut, hochgeladen.
Um die aktuell gespeicherten PDFs auflisten zu lassen, gibt es den 'ALLE PDFS ANZEIGEN'-Button, der vom Code-Behind alle, derzeit gespeicherten PDFs, bekommt und diese dann in einer Liste anzeigt. Dafür wird eine FlowDocument-Tabelle erstellt, die jedes PDF als eigene Zeile in einem einheitlichen Design darstellt. Zusätzlich befinden sich bei jeder Zeile die 'Öffnen',

'Herunterladen' und 'Löschen'-Knöpfe, die mithilfe der Code-Behind-Klasse die gewünschten Funktionen bieten. Um diese Liste wieder verschwinden zu lassen, erscheint beim Anzeigen oben ein "X", welches beim Klicken die Tabelle entfernt.

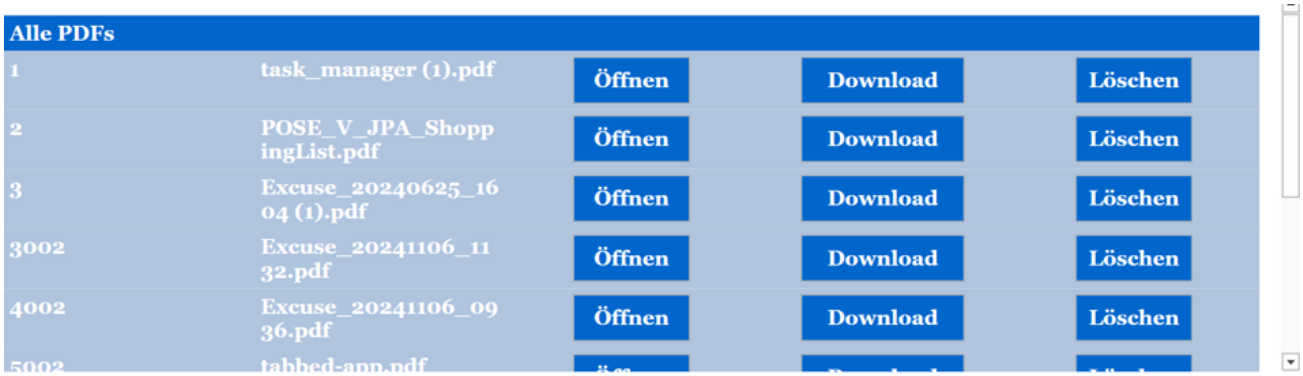

Abbildung 19: PDF-Liste

Die Suchfunktion nach einem PDF mittels Stichwörtern stellt sich aus einem Eingabefeld, wo man die Stichwörter eingeben kann, und einem Button, der die Stichwörter abschickt, zusammen. Beim Abschicken werden die Wörter dem Code-Behind übergeben, der nach einer Rückgabe vom Backend die passenden PDFs weiterleitet, welche dann in der Liste angezeigt werden.

Abbildung 20: PDF-Suche

Code-Behind (XAML.cs)/ Verbindung zum Server

Die Verbindung zwischen dem Code-Behind vom Frontend und dem Server, welcher für die Datenhaltung da ist, wird über eine REST-API hergestellt. Sprich, der Server stellt die Methoden, die das Frontend braucht, als API-Schnittstellen zur Verfügung und so kann das Frontend auf diese REST-API zugreifen und zum Beispiel alle PDFs abfragen.

Bei der 'uploadButtonKlick'-Methode, die beim Hochladen eines PDFs aufgerufen wird, wird das PDF in einen Base64-String konvertiert, dann wird ein 'PdfSendable'-Objekt erstellt, welches alle Informationen zum Speichern enthält, die Metadaten werden befüllt und schließlich wird das Objekt der REST-API übergeben:

Listing 34: Informationen in Objekt + Uebergabe an Rest

```
try
{
    var pdfBytes = File.ReadAllBytes(uploadPdf);
    var base64String = Convert.ToBase64String(pdfBytes);

    PdfSendable pdfSendable = new PdfSendable
    {
        PdfName = Path.GetFileName(uploadPdf),
        Base64String = base64String,
        Metadata = new Metadata
        {
            Create_Date = DateTime.Now,
            LastChange_Date = DateTime.Now,
            USER_ID = 4,
            Hostname = Environment.MachineName
        }
    };
    ...
}
//Uebergabe an die REST-API
RESTService restService = new RESTService(new RESTSettings());
RESTResult<string> result = await restService.PostAsync<string>(url + "Add", pdfSendable);
```

Danach wird noch geprüft, ob der Post-Request erfolgreich war und zeigt bei Erfolg eine Erfolgsmeldung an. Anmerkung: Die User-ID ist hier auf 4 eingestellt, aus dem einfachen Grund, dass der Wert nicht leer sein soll, aber es derzeit noch keine richtigen Benutzer gibt.

Bei der Auswahl von den PDFs hat man 2 Möglichkeiten. Bei der Auswahl über den Explorer wird dieser logischerweise geöffnet. Wird dann eine Datei ausgewählt, werden Informationen wie der Pfad und der Dateiname als Attribute gespeichert, um diesen dann der UI weiterzugeben. Das Öffnen des Explorers und das Anwenden eines Filters, sodass nur PDFs ausgewählt werden können, sieht wie folgt aus:

Listing 35: Explorer öffnen

```
OpenFileDialog openFileDialog = new OpenFileDialog();
openFileDialog.Filter = "PDF files (*.pdf)|*.pdf";
openFileDialog.InitialDirectory = @"c:\temp\";
```

Wählt man die andere Variante mit dem DragAndDrop-Feld, wird eine andere Methode aufgerufen. Hier wird geprüft, ob das DragAndDrop-Event wirklich eine Datei ist, und dann folgt wieder die Speicherung des Pfades und des Namens, die in der UI angezeigt werden.
Bei der Suche nach PDFs mit Stichwörtern wird die REST-API nach PDFs mit den Stichwörtern gefragt, welche mitgegeben werden. Zurück kommt eine Liste mit relevanten PDFs, die dann

wieder an die UI gegeben wird, die diese dann anzeigt:

Listing 36: Suche nach PDFs

```
private async void searchButton_Click(object sender, RoutedEventArgs e)
{
    ShowLoadingText();
    try
    {
        string keyword = searchBox.Text;
        pdfs = await storage.GetNecessaryPdfList(url + "searchPdfsByKeyword/" + keyword);
        DisplayPdfs(pdfs, Ergebnis f r:  + keyword);
        searchBox.Text = string.Empty;
    }
    catch (Exception ex)
    {
        fdocViewer.Document = null;
        DXMessageBox.Show($"Verbindung zum Server fehlgeschlagen: {ex.Message}");
    }
    HideLoadingText();
}
```

Beim Klicken auf den 'Öffnen'-Button von einem PDF wird eine Vorschau der Datei im Browser geöffnet. Dazu wird eine URL zusammengestellt, die dann im Browser aufgerufen wird. Das Öffnen des Standard-Browsers geschieht mit der 'Process.Start()'-Methode, in welcher 'UseShellExecute' auf 'true' gesetzt wird:

Listing 37: Vorschau oeffnen

```
try
{
    // Erstellt die URL zur Vorschau des PDFs auf dem Server
    string previewUrl = url + "preview/" + id;

    // Oeffnet die URL im Standard-Webbrowser
    Process.Start(new ProcessStartInfo
    {
        FileName = previewUrl, // Die URL, die geoeffnet werden soll
        UseShellExecute = true // Legt fest, dass ein externer Prozess (der Browser)
            gestartet wird
    });
}
```

Bei dem Button 'Herunterladen' wird das gewünschte PDF von der REST-API abgefragt und dann lokal abgespeichert. Hierzu öffnet sich der Explorer, um einen Speicherort auswählen zu können.

Listing 38: Herunterladen

```
 try
{
    string downloadUrl = url + "download/" + pdf.ID;

    using (HttpClient client = new HttpClient())
    {
        HttpResponseMessage response = await client.GetAsync(downloadUrl);

        if (response.IsSuccessStatusCode)
        {
            byte[] pdfBytes = await response.Content.ReadAsByteArrayAsync();

            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.Filter = "PDF files (*.pdf)|*.pdf";
            saveFileDialog.FileName = Path.GetFileName(pdf.PDF_Path);

            if (saveFileDialog.ShowDialog() == true)
            {
                File.WriteAllBytes(saveFileDialog.FileName, pdfBytes);
                DXMessageBox.Show("PDF erfolgreich heruntergeladen.");
            }
        }
        ...
    }
}
```

Genauer erklärt erfolgt das Herunterladen asynchron über die REST-API, sodass die Anwendung reaktionsfähig bleibt. Mithilfe des HttpClient wird ein effizienter Abruf der Datei als Binärdaten ermöglicht. Um es zu ermöglichen, dass der Benutzer den Speicherort für die Datei frei wählen kann, wird ein SaveFileDialog genutzt, der den Windows-Explorer aufruft.

Beim Löschen-Button wird ein Bestätigungsdialog für den Benutzer angezeigt, ob er wirklich mit dem Löschen fortfahren möchte. Wenn er diesen bestätigt, wird eine DELETE-Anfrage mit dem übergebenen PDF an die REST-API gestellt. Diese wird aus dem System entfernt und anschließend wird die Liste aller PDFs in der GUI aktualisiert, um das gelöschte PDF nicht mehr anzuzeigen.

4 Planung und Realisierung

4.1 Generelle Planung

4.1.1 Vorstellungen

Die generelle Planung für das Produkt fand nicht, wie üblicherweise, beim Start der Diplomarbeit statt, sondern bereits ein Jahr davor. Warum das so ist, lässt sich einfach erklären. Die Diplomarbeit setzt auf ein Projekt auf, welches schon geplant wurde. Sprich, die Vorstellung des Kunden, wie das Produkt auszusehen hat und wie es funktionieren soll, wurde bereits beim Projekt kommuniziert. Das bedeutet, der Grundstein der Planung wurde bereits gelegt und es stand nur noch die genauere Planung der Erweiterungen, die das Produkt während der Diplomarbeit bekommen soll, an. Dazu wurde beim Start der Diplomarbeit ein großes Meeting mit dem Auftraggeber gehalten, wo über die Erweiterungen, Verbesserungen sowie Änderungen diskutiert wurde. Das Produkt sollte, die erste Version, die beim Projekt herausgekommen ist, benutzerfreundlicher machen, mehr Funktionen bieten und die alten Funktionen performanter und stabiler machen, sodass am Ende ein Produkt dabei herauskommt, welches den Benutzern eine stabile und benutzerfreundliche Version des PDF-Managers mit vielen Funktionen und einem Chatbot, der gute Antworten mithilfe der PDFs generiert, bietet.

4.1.2 Umsetzung planen

Um eine Umsetzung des Produkts zustande zu bekommen, mit welcher der Auftraggeber zufriedengestellt ist, wurde eine ausführliche Dokumentation des Meetings verfasst. Zum einen, um zu erfahren, wie die genaue Vorstellung ist, damit Unklarheiten besser/schneller beseitigt werden können und vor allem, um einen Plan erstellen zu können, wie wir vorgehen müssen, um zum Ziel zu kommen. Um zu wissen, wie wir vorgehen müssen, sodass wir uns gegenseitig nicht ausbremsen, da der Chatbot die PDF von dem PDF-Manager braucht, um die Antworten zu generieren, war der Fokus beim Entwickeln des PDF-Managers klar darauf gesetzt, schnell eine Version bereitzustellen, die die PDFs richtig speichern kann und mit einer SignalR-Verbindung diese PDFs dem Chatbot zur Verfügung stellt. Die anderen Funktionen wurden dabei in den Hintergrund gestellt, um die Entwicklung des Bots nicht einzuschränken. Um einen möglichst

geregelten Ablauf gestalten zu können, hat sich jede Person Meilensteine zusammengestellt, die später miteinander nach Sinnhaftigkeit und Richtigkeit überprüft wurden. So hatten wir immer einen Überblick, ob wir im Zeitplan liegen, welche Schritte nach den aktuellen folgen und was das Ziel ist.

4.1.3 Dokumentation

Neben den Meilensteinen haben Dokumentationen, die von den Meetings und von unserem Code als Word oder Kommentaren über den Code gemacht wurden, unserer Produktivität enorm geholfen. Denn dadurch hatten wir immer einen Stützpunkt, der geholfen hat, das Produkt genau nach den Vorstellungen des Auftraggebers und so effektiv wie möglich umzusetzen. Zusätzlich wurde durch die Dokumentation vom Code die Fehlersuche erleichtert. Egal, wo ein Problem aufgetreten ist, in der Dokumentation war der Aufbau der Codeteile beschrieben, wodurch man die Fehlerursache viel leichter herausfinden konnte.

4.1.4 Ursprüngliche Planung

In der ursprünglichen Planung hatte Lucene.NET noch einen ganz anderen Zweck, der in der fertigen Version entfallen ist. Und zwar sollte der Chatbot und der PDF-Manager anders zusammenarbeiten. In der finalen Version übergibt der PDF-Manager die PDFs, die hochgeladen werden, direkt dem Chatbot, der diese in das KernelMemory hineinlädt, um so Antworten auf die Fragen mittels der Dokumente generieren zu können. Aber ursprünglich sollte es wie folgt ablaufen:

- Der Chatbot bekommt eine Frage und übergibt diese dem PDF-Manager.
- Dieser durchsucht alle Dokumente nach relevanten PDFs die zu den Wörtern der Frage passen.
- Der PDF-Manager übergibt dem Chatbot die PDFs wo die Relevanz hoch genug ist, um für die Beantwortung der Frage wichtig sein zu können.
- Der Chatbot beantwortet die Frage des Benutzers mithilfe der PDFs die er vom PDF-Manager übergeben bekommen hat.

Probleme dabei waren die Performance des Chatbots, da dieser die erhaltenen PDFs erst nach der Fragestellung des Benutzers einlesen konnte. Die finale Version vom Chatbot bietet zwar die Möglichkeit, selbst die relevanten PDFs für eine Frage zu finden, aber das Einlesen dauert einige Zeit, wodurch es so zu massiven Verzögerungen bei der Beantwortung von Fragen gekommen ist.

Deswegen war die Lösung, die Dokumente schon beim Hochladen dem Chatbot zu übergeben, sodass dieser die PDFs gleich einlesen kann und somit für Antworten schneller bereit ist.

4.1.5 Laufende Meetings

Meetings mit dem Auftraggeber während der Entwicklung waren nicht schwer umzusetzen, da wir nur wenige Meter von unserer Ansprechperson, unseren Arbeitsplatz beim Praktikum, hatten. Dadurch waren wir nicht an fixe Meetings gebunden, sondern konnten, immer wenn es einen neuen Prototypen von beiden Produkten gab, die die neuen Funktionen lauffähig zeigen konnten, mit unserer Ansprechperson ein Meeting vereinbaren, welches meistens noch am selben Tag stattgefunden hat, ausmachen. Bei diesen Meetings, welche 1-2 mal pro Woche stattfanden, wurde der aktuelle Stand präsentiert, Fragen gestellt, Verbesserungsvorschläge aufgenommen und über das weitere Vorgehen diskutiert.

4.2 Probleme

4.2.1 Performance

Die gesamte Applikation läuft auf einer Windows Server 2022 VM auf einem SYSco-Server. Für den Chatbot benötigt man spezielle Hardware für die Generierung bzw. der RAG-Funktion. Mit dem Server kommt nur eine reine CPU-Lösung zum Einsatz. Eine reine CPU zur Verarbeitung von KI ist eine eher nicht empfehlenswerte Option, da eine CPU, die nicht dafür benötigte Leistung mit sich bringt.
Im folgenden Bild werden Ergebnisse aus Testeinheiten gezeigt:

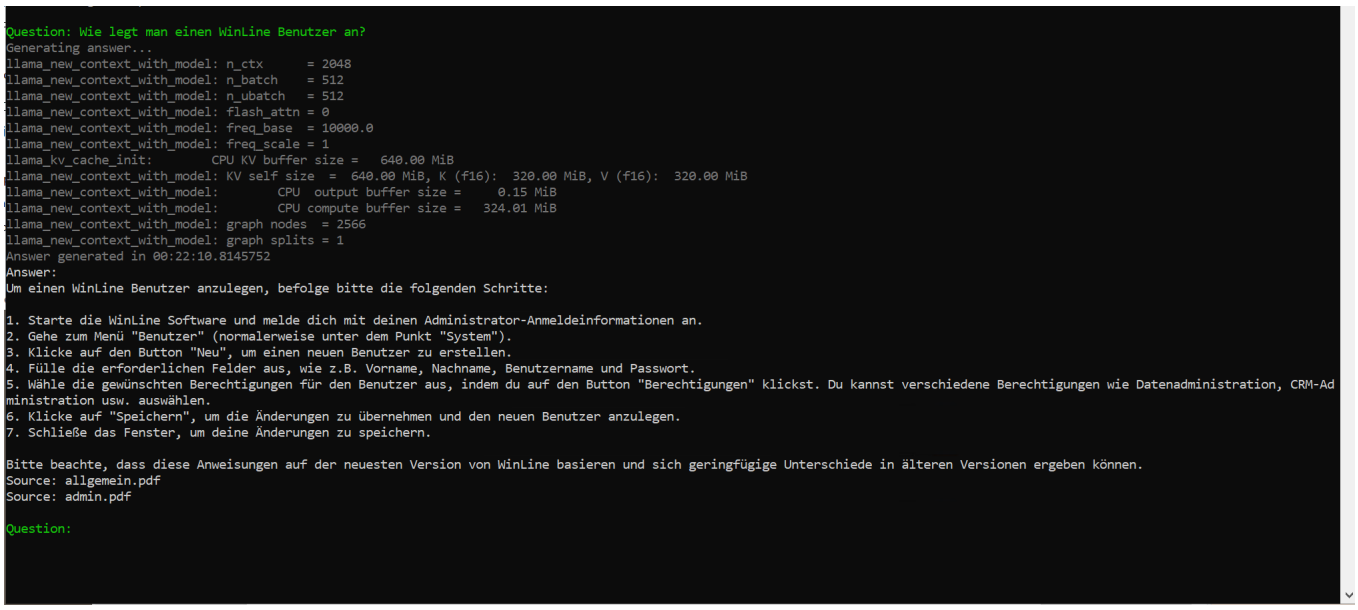

Abbildung 21: Ergebnis von Frage aus einer PDF

In dem Bild wird gezeigt, dass die Antwortzeit in diesem Fall bei 22 Minuten und 10 Sekunden liegt. Viel zu lange für eine KI, die Informationen aus einem PDF suchen soll. Somit besteht das Problem mit der Performance.

4.2.2 Testen der Applikation

Das Testen der Applikation war eher schwierig, da die Performance sehr gering war. Um ein PDF-Dokument in den KernelMemory zu laden, dauerte es bei den Tests ungefähr 2 Wochen. Die Entwicklung der Software hat sich somit sehr in die Länge gezogen, wenn die ständigen Ladezeiten eines PDFs beachtet werden.

4.2.3 8B vs 70B Modell

Grundsätzlich wurde hauptsächlich mit einem 8B-Modell getestet und gearbeitet. Der Grund dafür war die geringe Leistung, die das Modell benötigt. Da der SYSco-Server eher wenig Performance hergibt, war dieses geringe Modell sehr von Vorteil. Der riesige Nachteil von einem 8B-Modell ist, dass die Qualität sehr stark daran leidet. Aus diesem Grund wurde auf ein 70B-Modell hochgestuft. Im Vergleich:

- 8B: Verbraucht ca. 4-8GB Arbeitsspeicher und performt relativ schnell.
- 70B: Verbraucht ca. 40-70GB Arbeitsspeicher und läuft sehr langsam bei schlechter Hardware.

4.2.4 LLamaSharp-Version

Ein großes Problem während der Entwicklung der Software war, die verschiedenen Versionen von LLamaSharp. Die Programmierer dieses Pakets veröffentlichten ständig neue Versionen. Das Grundprinzip von LLamaSharp, dass mithilfe eines Modells willkürliche Antworten aus Allgemeinwissen generiert werden, funktionierte. Jedoch litt darunter die Kompatibilität mit LLamaSharp.KernelMemory. Die letzte Version, die mit KernelMemory noch verfügbar war, ist '0.13.0'. Dies brachte weitere Probleme mit sich, z.B. dass 'SqlServerMemoryDb' mit dieser Version von LLamaSharp nicht mit dem CPU-Backend kompatibel ist.

4.2.5 SqlServerMemoryDb - Fehler mit CPU-Backend

Der Plan ist, dass sämtliche Vektor-Daten von den PDFs in KernelMemory in einer MSSQL-Datenbank gespeichert werden. Sobald das Programm ausgeführt wird, erscheint ein Fehler, der darauf verweist, dass die Versionen untereinander 'zu alt' sind. Die Versionen können aber nicht aktualisiert werden, da bei neueren Versionen das gesamte KernelMemory zusammenbricht (siehe Problem mit LLamaSharp-Version).

4.2.6 Übergabe ohne Dokumentation - 500 Error

Der übergebene PDF-Manager war leider schlecht bis gar nicht dokumentiert, was einige Probleme mit sich brachte. Zum einen gab es bei dem Versuch, das Backend mit IIS zu hosten, sodass das Backend immer aufrufbar ist, einen Fehler. Und auch lokal hat sich die übergebene Version aus unerklärlichen Gründen nicht ohne Fehler ausführen lassen. Genauer gesagt hat man einen 500-Error bekommen, ohne genauere Informationen, warum dieser aufgetreten ist. Bei der Suche nach dem Fehler ist die schlechte Qualität des Codes aufgefallen, in dem sich Codeduplikationen und weitere unsaubere Programmierweisen befanden. Das, in Kombination damit, dass keine Codedokumentation vorhanden war, hat die Suche sehr erschwert. Nach einer längeren Suche ohne Erfolg wurde entschieden, den Code neu zu programmieren, sodass keine unnötig lange Zeit mit der Fehlersuche verloren ging und zusätzlich das Projekt sauberer programmiert werden konnte. Somit wurde ein neues Projekt in Visual Studio 2022 erstellt, in dem das Projekt neu aufgebaut wurde. Der Code wurde an einigen Stellen eins zu eins übernommen, aber bei anderen wurde der Code komplett neu verfasst, um eine sauberere, objektorientierte Programmierweise sicherzustellen. So wurde eine Basis geschaffen, womit sich das Projekt ohne Fehler ausführen und mit IIS hosten ließ und auf dem man die neuen Funktionen und das neue Design aufbauen konnte.

4.2.7 Metadaten sinnlos befüllt

Die Metadaten, die zu den PDFs gespeichert werden, waren bei der Übergabe mit sinnlosen Werten befüllt, die nicht relevant für ein PDF waren. Zudem hatten die Metadaten keine Verbindung zu den PDFs, sodass man sie keinem PDF zuordnen konnte, was natürlich in keiner Sichtweise logisch war. Um die Probleme zu beheben, wurde mit dem Auftraggeber gesprochen, um auf sinnvolle Metadaten für die Dokumente zu kommen. Dabei hat man sich auf ein Erstelldatum, ein Datum, wann es zuletzt bearbeitet wurde, und einen Hostnamen, um erkennen zu können, wer das PDF hochgeladen hat, geeinigt. Und nicht zu vergessen, wurde eine PDF-ID gespeichert, um die Metadaten auch einem PDF zuordnen zu können und sie so den Sinn von Metadaten erfüllen. Somit waren die Daten zwar sinnvoll befüllt, jedoch mussten viele Teile im Code und in der Datenbank geändert werden, sodass sie dem neuen Schema gerecht werden konnten.

4.3 Lösungen

4.3.1 Performance Probleme

Eine Lösung für das Problem mit der Performance wäre, dass Azure AI Search verwendet wird. Das ist ein AI-Service von Microsoft Azure, der speziell RAG mit Vektor-Suche beinhaltet. Dieser Service ist genau für diese Aufgabe ausgelegt. Die Kosten sind eher hoch und deshalb noch nicht im Einsatz. Die Umstellung wird eventuell in Zukunft in der Projektarbeit durchgeführt.

5 Ergebnis

Diese Arbeit ist trotz vieler Umstände erfolgreich gelungen. Nachfolgend sind die Ergebnisse des Chatbots sowie des PDF-Managers aufgelistet.

5.1 Shell

In der Shell werden Chatbot und PDF-Manager als Modul implementiert. Das Modul hat den Projektnamen 'scoSIMS'. Dieses Modul kann vom SYSco-GitLab-Server ganz einfach heruntergeladen werden, wie im folgenden Bild ersichtlich:

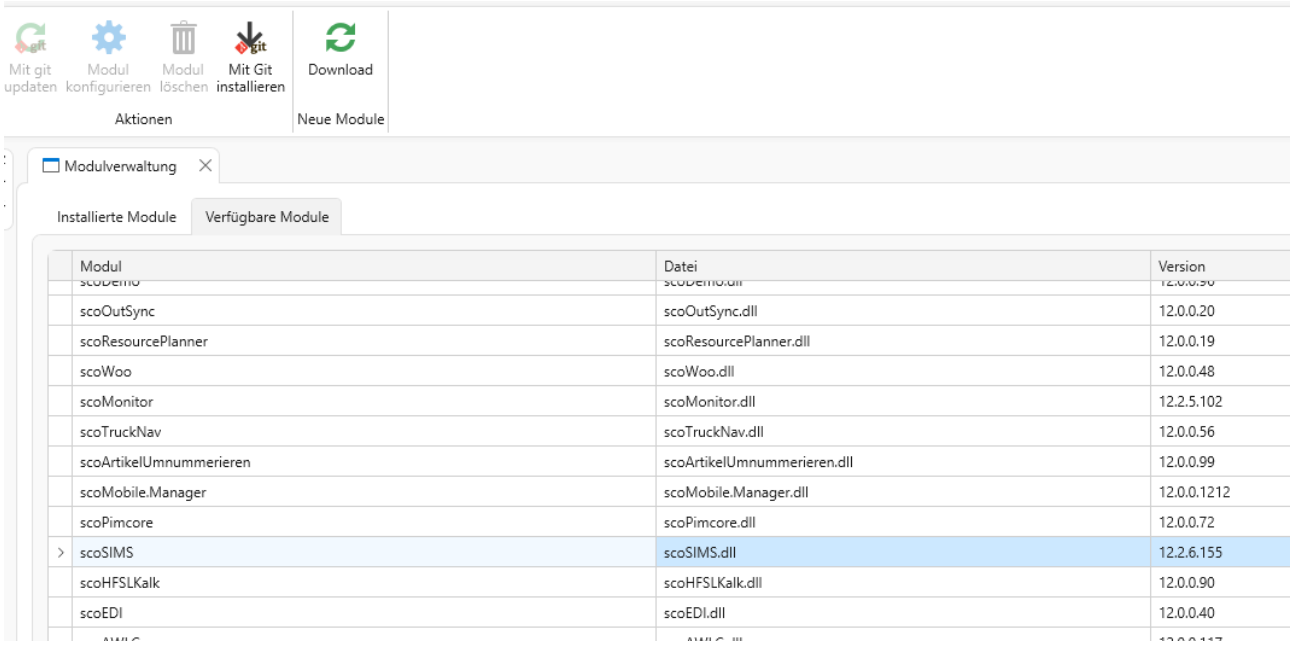

Abbildung 22: Modul 'scoSIMS' installieren in der Shell

Nach der Installation des Moduls wird in der Navigationsbar ein neuer Reiter erstellt, in dem die einzelnen Komponenten, also Chatbot und PDF-Manager, angezeigt werden. Das Ergebnis in der SYSco-Shell sieht dann folgendermaßen aus:

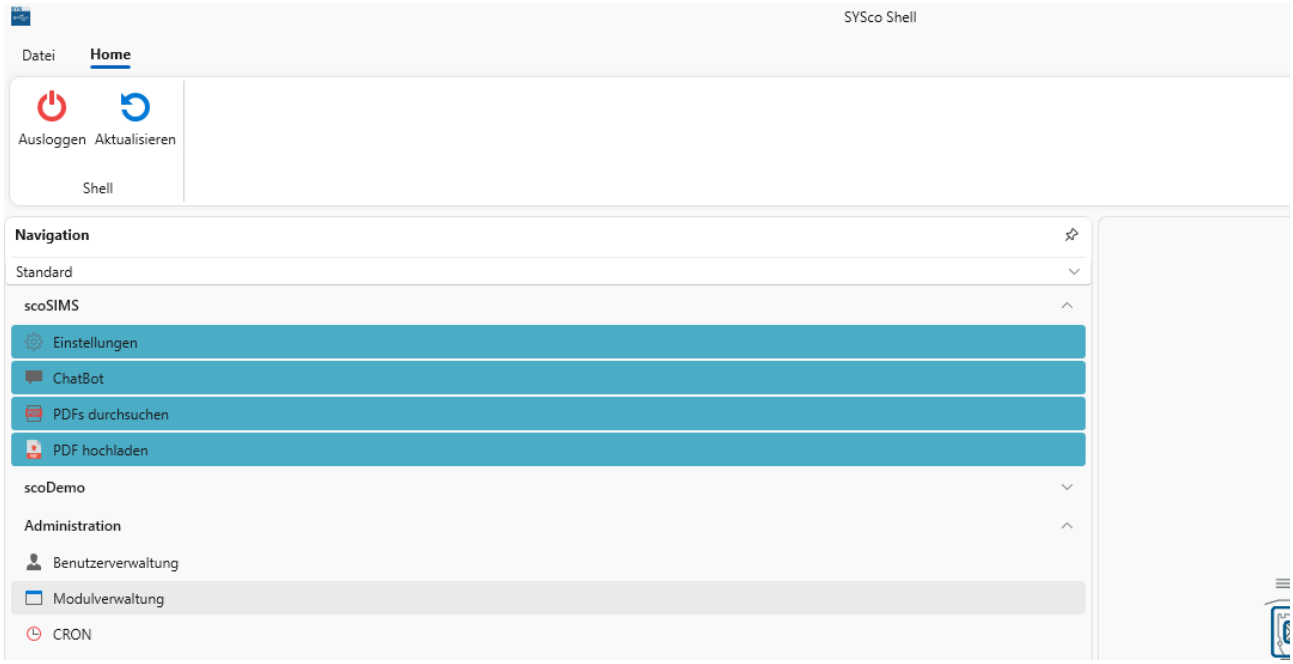

Abbildung 23: Das Modul 'scoSIMS' in der Navigation

5.2 Chatbot

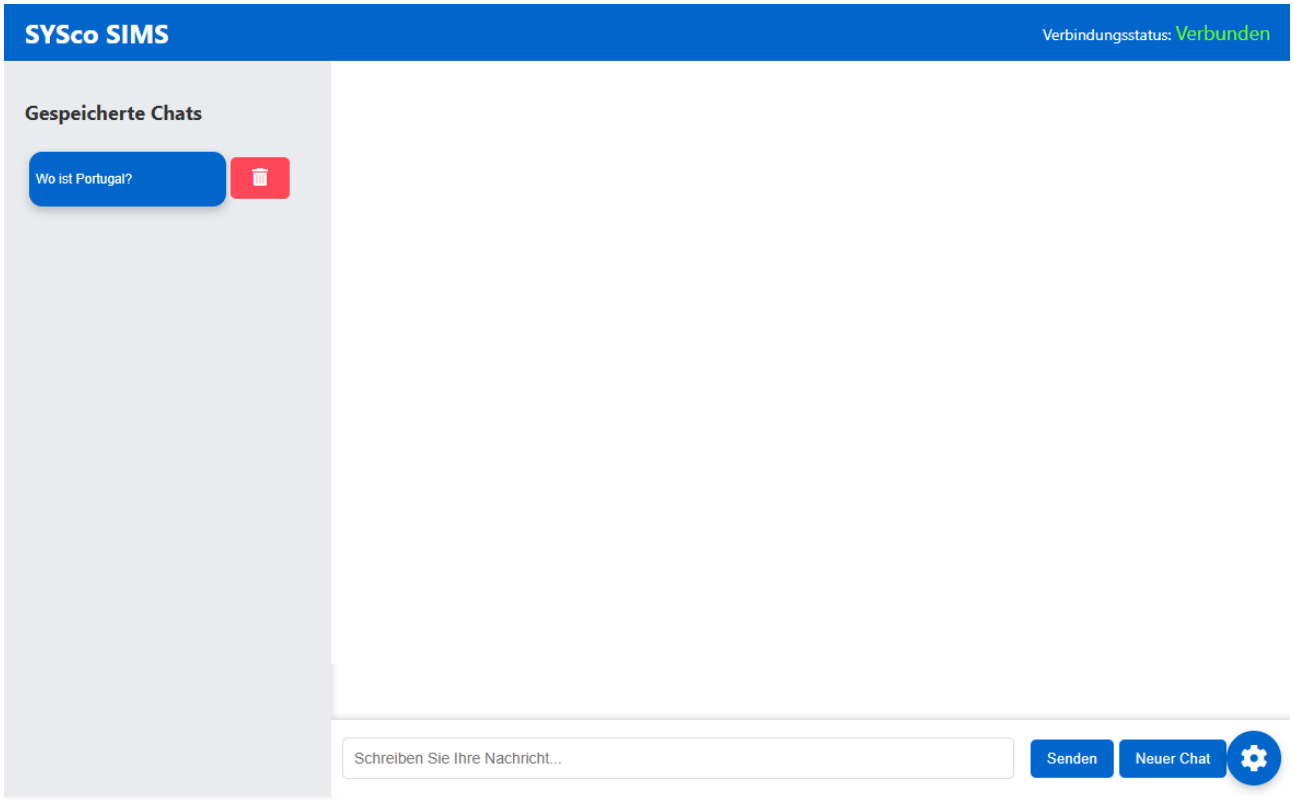

Abbildung 24: Ergebnis Chatbot

Der Chatbot ist trotz großer Anstrengung und viel Arbeit gelungen. Trotz des Problems mit der Performance kann man behaupten, dass der Chatbot funktioniert, auch wenn die Analyse sowie das Antworten eine lange Zeit in Anspruch nehmen. Die Kommunikation mit dem Server

funktioniert einwandfrei sowie auch das Erstellen von neuen Chats mit dem Button rechts unten. Die geschriebenen Chats werden anhand der UserID in eine Datenbank gespeichert und werden links in einer Liste von 'Gespeicherten Chats' auch angezeigt. Somit kann man alte Unterhaltungen mit dem Chatbot jederzeit abrufen, aber auch wieder löschen. Rechts oben wird der aktuelle Verbindungsstatus angezeigt, ob die Benutzeroberfläche mit dem Server verbunden ist.

5.3 PDF-Manager

Abbildung 25: Ergebnis PDF-Manager

Der PDF-Manager liefert eine gute Arbeit ab, wenn es darum geht, PDFs zu verwalten. Egal, ob das Hochladen, Öffnen, Löschen der PDFs, alles funktioniert einwandfrei und ohne große Verzögerungen. Nach den ganzen Grundfunktionen gibt es natürlich noch etwas Besonderes, und zwar die Suche nach PDFs mittels Stichwörtern, die mit Lucene.Net ermöglicht wurde. Diese bietet die Möglichkeit, die PDFs in nur wenigen Sekunden nach den relevanten PDFs zu durchsuchen und zurückzugeben. Die Datenbank, die für den Benutzer zwar nicht sichtbar ist, spielt hier auch eine große Rolle und wurde so umgesetzt, dass die PDFs mit den wichtigsten Metadaten abgespeichert werden. Zusätzlich wurde noch auf den Ausbau des Produkts geachtet,

indem die Datenbank auch die Möglichkeit bietet, Benutzer mit einem Passwort zu speichern, um das Produkt auch in Richtung Benutzer weiterentwickeln zu können.

6 Resümee

6.1 Chatbot

Anfangs sah das Vorhaben relativ einfach aus. Nach der Zeit wurde aber klar, dass es doch komplexer wird als gedacht. Das Interesse an Artificial Intelligence wurde von Tag zu Tag größer während der Programmierung an dieser Arbeit. Der schwierigste Teil daran war, dass man in das Thema hineinkommen muss, um zu verstehen, was man macht. Nach wochenlanger Recherche war nun der harte Part des Projekts geschafft. Nun musste das gelernte Wissen umgesetzt werden. Schnell stößt man aber an die ersten Probleme bei der Umsetzung, da reines Theorie-Wissen für so ein Projekt nicht ausreichend ist. Dadurch, dass LLamaSharp ein neu entwickeltes OpenSource-Projekt ist, befinden sich noch eine Menge Bugs im Quellcode. Dadurch stößt man relativ schnell an die Grenzen der Möglichkeiten, wie z.B. dass die neueste Version von LLamaSharp nicht mit LLamaSharp.KernelMemory funktioniert, sondern dauerhaft Exceptions geworfen werden. Deshalb musste auch in der Arbeit auf eine alte Version zurückgegriffen werden, die noch problemlos läuft. Dies und viele weitere ähnliche Probleme kamen uns immer wieder unter. Trotz der Tatsache, dass der Weg hart und steinig war, meisterten wir die Aufgabe, abgesehen von dem Problem mit der Performance, das nicht von uns abhängig ist.
Durch das Praktikum bei der Firma SYSco standen wir in engem Kontakt und hatten somit jederzeit die Möglichkeit, Fragen zu stellen und Hilfe zu bekommen. Auch wenn das Unternehmen noch eher wenig Erfahrung im Bereich mit AI hat, haben sie uns trotz dieser Tatsache bei fast allen Problemen helfen können, und das schätzen wir sehr.
Unser Team bestand aus Tobias Fröschl und mir - und das war auch gut so, denn unsere Einschätzung ist, dass man zu zweit am besten harmoniert. In diesem Sinne haben Tobias und ich uns gut abgestimmt, auf welche Bereiche wir uns spezialisieren und wie wir unsere beiden Projektteile am besten kombinieren. So entstand unsere Arbeit zur Gänze.

6.2 PDF-Manager

Das Entwickeln war ein langer Weg, wo es Höhen, aber auch genauso manche Tiefen und Rückschläge gab. Aber grundsätzlich hat man mit diesem Projekt einen guten Einblick in die Welt von künstlicher Intelligenz erhalten auch wenn es beim PDF-Manager nicht so präsent war wie beim Chatbot, dennoch hat man viel neues gelernt. Einer der größten Hürden war mit Sicherheit der Start. Ein Produkt zu übernehmen, welches jemand anderes entwickelt hat und wo es keine Dokumentation zum Code gibt, ist eine große Herausforderung. Zum einen das man versteht wie der Code funktioniert und zum anderen das Fehlersuchen ohne Dokumentation, dass ebenso eine Challenge war. Aber als der Start gelungen war, ist man mit vollem Elan an die Sache herangegangen und hat in nur wenigen Wochen ein gutes Produkt auf die Beine gestellt was sich zeigen lässt.

Aber nicht nur die ganzen Technologien, die wir uns angeeignet haben, sind positiv hervorzuheben, sondern das Bekommen von praktischer Erfahrung in der realen Welt. Mit einem richtigen Auftraggeber zu arbeiten und ein Produkt, welches später hoffentlich in der Firma eingesetzt wird oder sogar an andere verkauft wird, zu entwickeln war ein großer Ansporn und erfüllt mit Stolz unseren zukünftigen Weg.

Beim Team war nichts auszusetzen. Es war immer ein gutes Zusammenspiel, was mit Sicherheit auch dem zuzuschreiben ist, dass das Team aus nur zwei Personen bestand. Das Entwickeln in der kleinen Gruppe war sehr produktiv und angenehm.

Glossar

API Application Programming Interface (Programmierschnittstelle): Eine API ist eine Schnittstelle, über die Softwarekomponenten oder verschiedene Systeme miteinander kommunizieren können, ohne zu wissen, wie der andere intern funktioniert.

CD Continuous Deployment/Delivery: Ein Vorgang, bei dem neuer Code automatisch veröffentlicht oder für die Veröffentlichung vorbereitet wird.

CI Continuous Integration: Ein Prozess, bei dem neuer Code automatisch getestet wird, sobald er hochgeladen wird.

CPU Central Processing Unit: Ist die zentrale Recheneinheit eines Computers, welche die meisten Rechenarbeiten übernimmt

CUDA Compute Unified Device Architecture: Eine Technik von NVIDIA, mit der Programme besonders schnelle Berechnungen auf der Grafikkarte ausführen können.

GPU Graphics Processing Unit: Ein spezieller Prozessor, der speziell für Grafiken und schnelle Berechnungen verwendet wird.

GUI Graphical User Interface: Eine GUI ist die visuelle Oberfläche, über die ein Benutzer mit einem Programm interagieren kann

Hyper-V Hyper-V ist eine Virtualisierungstechnologie von Microsoft, die es ermöglicht, komplette virtuelle Maschinen mit eigenem Betriebssystem auf einem Windows-System auszuführen.

IoT Internet of things: Ein Netzwerk aus physischen Geräten, welche über das Netzwerk kommunizieren

KM Kernel Memory: Ein Projekt von Microsoft, das Dokumente speichert und verwaltet, damit LLMs besser auf das Wissen dieser Dokumente zugreifen und Fragen beantworten können.

LLM Large Language Model: Ein KI-Modell, das viele Informationen gespeichert hat und dadurch menschenähnliche Sprache verstehen und generieren kann.

MSSQL Microsoft SQL Server: Ein Programm von Microsoft, mit dem man Daten in eine Datenebank speichern, verwalten und abfragen kann.

NuGet Paketmanager von Visual Studio 2022

RAG Retrieval Augmented Generation: Eine Methode, bei der passende Informationen mittels LLM gesucht werden und darauf eine passende Antwort generiert wird.

UI User Interface: Das User Interface ist der Teil es Programms welchen die Benutzer sehen und bedienen können.

WSL2 (Windows Subsystem for Linux 2) ist eine Kompatibilitätsschicht für die Ausführung von Linux-Distributionen direkt unter Windows, die auf einer leichtgewichtigen virtuellen Maschine basiert und eine echte Linux-Kernel-Umgebung bietet.

Literaturverzeichnis

[1] Sean Michael Kerner, „What are large language models (LLMs)?" letzter Zugriff am 19.02.2025. Online verfügbar: https://www.techtarget.com/whatis/definition/large-language-model-LLM

[2] Microsoft, „Kernel Memory," letzter Zugriff am 19.02.2025. Online verfügbar: https://microsoft.github.io/kernel-memory/

[3] IBM, „Retrieval-Augmented Generation," letzter Zugriff am 21.02.2025. Online verfügbar: https://learn.microsoft.com/de-de/sql/sql-server/what-is-sql-server?view=sql-server-ver16

[4] WikiPedia, „What is retrieval-augmented generation?" letzter Zugriff am 21.02.2025. Online verfügbar: https://learn.microsoft.com/de-de/sql/sql-server/what-is-sql-server?view=sql-server-ver16

[5] ?, „C vs. .NET: Die Unterschiede und Vorteile," letzter Zugriff am 19.02.2025. Online verfügbar: https://ankhlabs.de/glossar/c-vs-net-die-unterschiede-und-vorteile/

[6] Wikipedia, „CUDA," letzter Zugriff am 19.02.2025. Online verfügbar: https://en.wikipedia.org/wiki/CUDA

[7] arocom, „Git," letzter Zugriff am 10.02.2025. Online verfügbar: https://www.arocom.de/fachbegriffe/webentwicklung/git

[8] ——, „Gitlab," letzter Zugriff am 10.02.2025. Online verfügbar: https://www.arocom.de/fachbegriffe/webentwicklung/was-ist-gitlab

[9] Microsoft, „Was ist Visual Studio?" letzter Zugriff am 23.07.2024. Online verfügbar: https://learn.microsoft.com/de-de/visualstudio/get-started/visual-studio-ide?view=vs-2022

[10] GitHub, „LLamaSharp," letzter Zugriff am 10.02.2025. Online verfügbar: https://github.com/SciSharp/LLamaSharp

[11] ——, „LLamaSharp," letzter Zugriff am 10.02.2025. Online verfügbar: https://scisharp.github.io/LLamaSharp/

[12] ——, „LLamaSharp," letzter Zugriff am 19.02.2025. Online verfügbar: https://github.com/SciSharp/LLamaSharp

[13] Microsoft, „Introduction to SignalR," letzter Zugriff am 19.02.2025. Online verfügbar: https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr

[14] GitHub, „ASP.NET SignalR," letzter Zugriff am 19.02.2025. Online verfügbar: https://github.com/SignalR/SignalR

[15] Mircosoft, „Real-time ASP.NET with SignalR," letzter Zugriff am 19.02.2025. Online verfügbar: https://dotnet.microsoft.com/en-us/apps/aspnet/signalr?

[16] Lucene.net, „Lucene.NET Documentation," letzter Zugriff am 18.01.2025. Online verfügbar: https://lucenenet.apache.org/docs.html

[17] Github, „Welcome to Apache Lucene.NET," letzter Zugriff am 23.03.2025. Online verfügbar: https://github.com/apache/lucenenet

[18] Microsoft, „Einführung in Microsoft Edge WebView2," letzter Zugriff am 19.01.2025. Online verfügbar: https://learn.microsoft.com/de-de/microsoft-edge/webview2/

[19] ——, „Entity Framework documentation hub," letzter Zugriff am 16.02.2025. Online verfügbar: https://learn.microsoft.com/en-us/ef/

[20] GitHub, „Entity Framework Core," letzter Zugriff am 19.02.2025. Online verfügbar: https://github.com/dotnet/efcore

[21] Microsoft, „Entity Framework Tools in Visual Studio," letzter Zugriff am 19.02.2025. Online verfügbar: https://learn.microsoft.com/en-us/visualstudio/data-tools/entity-data-model-tools-in-visual-studio?view=vs-2022

[22] ——, „What's new in Windows Server 2022," letzter Zugriff am 19.02.2025. Online verfügbar: https://learn.microsoft.com/en-us/windows-server/get-started/whats-new-in-windows-server-2022

[23] IONOS Redaktion, „Docker – die revolutionäre Container-Technologie," letzter Zugriff am 29.01.2025. Online verfügbar: https://www.ionos.de/digitalguide/server/knowhow/was-ist-docker/

[24] Docker Docs, „What is Docker?" letzter Zugriff am 29.01.2025. Online verfügbar: https://docs.docker.com/get-started/docker-overview/?

[25] ComputerWeekly, „Microsoft SQL Server," letzter Zugriff am 05.03.2025. Online verfügbar: https://www.computerweekly.com/de/definition/Microsoft-SQL-Server?utm_source=chatgpt.com

[26] Microsoft, „Microsoft SQL Server," letzter Zugriff am 05.03.2025. Online verfügbar: https://learn.microsoft.com/de-de/sql/sql-server/what-is-sql-server?view=sql-server-ver16

[27] ——, „Microsoft® SQL Server® 2022 Express," letzter Zugriff am 21.02.2025. Online verfügbar: https://www.microsoft.com/en-us/download/details.aspx?id=104781

[28] ——, „SQL Server 2022 Express: Alles, was Sie wissen müssen," letzter Zugriff am 21.02.2025. Online verfügbar: https://softtrader.de/microsoft-sql-server-2022-express/

[29] rwestMSFT, „What is SQL Server Management Studio (SSMS)?" letzter Zugriff am 21.02.2025. Online verfügbar: https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms

[30] SwaggerIO, „API Development forEveryone," letzter Zugriff am 26.03.2025. Online verfügbar: https://swagger.io/

[31] julietrome, „Was ist Swagger?" letzter Zugriff am 26.03.2025. Online verfügbar: https://julietrome.de/swagger/

[32] Microsoft, „IIS," letzter Zugriff am 19.02.2025. Online verfügbar: https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms

[33] ——, „Einführung in Microsoft Teams," letzter Zugriff am 19.03.2025. Online verfügbar: https://support.microsoft.com/de-de/office/einfuehrung-in-microsoft-teams-59b4cf2f-84ef-4a41-860a-37d3b9af09d3

[34] Clockify, „Einführung in Clockify," letzter Zugriff am 19.03.2025. Online verfügbar: https://clockify.me/help/de/getting-started/clockify_basics/introduction-to-clockify

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Anhang

A Aufgabenverteilung

In diesem Anhang wird festgelegt, welcher Schüler dieser Arbeit welches inhaltliche Thema verfasst hat.

A.1 Haider

Geschriebene Themen von Michael Haider:

- 1 Einleitung
  - 1.3 Ausgangspunkt
    - 1.3.2 Chatbot
  - 1.5 Projektumfeld
    - 1.5.2 Betreuung
    - 1.5.3 Auftraggeber
- 2 Theoretische Grundlagen
  - 2.1 Grundlegende Fachbegriffe *(Gesamtes Kapitel)*
  - 2.2 Technologien *(Gesamtes Kapitel*
  - 2.3 Entwicklungssysteme *(Gesamtes Kapitel)*
  - 2.4 Bibliotheken/Plug-Ins (NuGets)
    - 2.4.1 LLamaSharp
    - 2.4.2 LLamaSharp.KernelMemory
    - 2.4.5 Microsoft Edge WebView2
  - 2.5 Hardware *(Gesamtes Kapitel)*
  - 2.6 Datenhaltung
    - 2.6.2 MSSQL
    - 2.6.3 SQL Server 22 Express
    - 2.6.4 SQL Server Management Studio

- 3 Implementierung
  - 3.1 SYSco-Shell *(Gesamtes Kapitel)*
  - 3.2 SignalR
    - 3.2.1 Kommunikation zwischen ChatUI und Chatbot-Server
  - 3.3 LLamaSharp - Integration von LLMs *(Gesamtes Kapitel)*
  - 3.4 KernelMemory - Analyse/Speichern von Dokumenten *(Gesamtes Kapitel)*
  - 3.7 Datenbankstruktur KernelMemory *(Gesamtes Kapitel)*
  - 3.8 Datenbankstruktur der Chat-Speicherung *(Gesamtes Kapitel)*
  - 3.10 Datenhaltung
    - 3.10.1 Chatbot
  - 3.11 Benutzeroberfläche - UI
    - 3.11.1 Chatbot
- 4 Planung und Realisierung
  - 4.2 Probleme
    - 4.2.1 Performance
    - 4.2.2 Testen der Applikation
    - 4.2.3 8B vs 70B Modell
    - 4.2.4 LLamaSharp-Version
    - 4.2.5 SqlServerMemoryDb - Fehler mit CPU-Backend
  - 4.3 Lösungen *(Gesamtes Kapitel)*
- 5 Ergebnis
  - Text von '5 Ergebnis'
  - 5.1 Shell
  - 5.2 Chatbot
- 6 Resümee
  - 6.1 Chatbot

A.2 Fröschl

Geschriebene Themen von Tobias Fröschl:

- 1 Einleitung
  - 1.1 Motivation (gesamtes Kapitel)
  - 1.2 Problemstellung (gesamtes Kapitel)
  - 1.3 Ausgangspunkt
    - 1.3.1 Allgemein
    - 1.3.3 PDF-Manager
  - 1.4 Zielsetzung (gesamtes Kapitel)
  - 1.5 Projektumfeld
    - Text von '1.5 Projektumfeld'
- 2 Theoretische Grundlagen
  - 2.4
    - 2.4.3 SignalR
    - 2.4.4 Lucene.NET
    - 2.4.6 EntityFramework
  - 2.6 Datenhaltung
    - 2.6.1 Docker
    - 2.6.5 Swagger
  - 2.7 Sonstige Software (gesamtes Kapitel)
- 3 Implementierung
  - 3.2
    - 3.2.2 PDF-Austausch zwischen PDF-Server und Chatbot-Server
  - 3.5 Datenhaltung (gesamtes Kapitel)
  - 3.6 Datenbank (gesamtes Kapitel)
  - 3.9 Datenbankstruktur PDF-Manager (gesamtes Kapitel)
  - 3.10 Datenhaltung
    - 3.10.2 PDF-Manager
  - 3.11 Benutzeroberfläche - UI
    - 3.11.2 PDF-Manager
- 4 Planung und Realisierung
  - 4.1 (gesamtes Kapitel)
  - 4.2 Probleme
    - 4.2.6 Übergabe ohne Dokumentation - 500 Error

        * 4.2.7 Metadaten sinnlos befüllt
- 5 Ergebnis
    - 5.3 PDF-Manager (gesamtes Kapitel)
- 6 Resümee
    - 6.2 PDF-Manager (gesamtes Kapitel)