HAW HAMBURG

BACHELORTHESIS
Nico Karsten Lange

Automatisierte Generierung von Projektmanagementdokumenten mittels eines Large Language Models

FACULTY OF COMPUTER SCIENCE AND ENGINEERING
Department of Information and Electrical Engineering

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

HAMBURG UNIVERSITY
OF APPLIED SCIENCES

Hochschule für Angewandte
Wissenschaften Hamburg

Nico Karsten Lange

Automatisierte Generierung von Projektmanagementdokumenten mittels eines Large Language Models

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Elektro- und Informationstechnik*
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Hensel
Zweitgutachter: Prof. Dr. Herster

Eingereicht am: 04. Juni 2025

Nico Karsten Lange

Title of Thesis

Automated generation of project management documents using a large language model

Keywords

AI, LLM, Prompting, Document generation

Abstract
This bachelor thesis deals with the development of an application for the automated generation of requirements. A requirements document is read in from the customer and broken down into individual requirements. These are then written in a separate requirements document. A large language model is used to generate the requirements.

Nico Karsten Lange

Thema der Arbeit

Automatisierte Generierung von Projektmanagementdokumenten mittels eines Large Language Models

Stichworte

KI, LLM, Prompting, Dokumenten Generierung

Kurzzusammenfassung

Diese Bachelorthesis behandelt die Entwicklung einer Anwendung zur automatisierten Generierung von Anforderungen. Es wird ein Anforderungsdokument vom Kunden eingelesen und in einzelne Anforderungen aufgegliedert. Danach werden diese in ein eigenes Anforderungsdokument geschrieben. Für die Generierung der Anforderungen wird ein Large Language Model verwendet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

API **A**pplication **P**rogramming **I**nterface.

DCC **D**ocument **C**lassification **C**ode.

GPT **G**enerative **P**re-trained **T**ransformer.

GQA **G**rouped **Q**uery **A**ttention.

HTTP **H**yper**t**ext **T**ransfer **P**rotocol.

JSON **J**ava**S**cript **O**bject **N**otation.

KI **K**ünstliche **I**ntelligenz.

LLM **L**arge **L**anguage **M**odel.

MINT **M**athematik, **I**nformatik, **N**aturwissenschaft und **T**echnik.

MMLU **M**assive **M**ultitask **L**anguage **U**nderstanding.

NLP **N**atural **L**anguage **P**rocessing.

PDF **P**ortable **D**ocument **F**ormat.

PNG **P**ortable **N**etwork **G**raphics.

REST **Re**presentational **S**tate **T**ransfer.

RMG **R**ail-**M**ounted **G**antry Crane.

RTG **R**ubber-**T**ired **G**antry Crane.

SPS **S**peicher**p**rogrammierbare **S**teuerung.

STEM **S**cience, **T**echnology, **E**ngineering and **M**athematics.

STS **S**hip-**T**o-**S**hore Crane.

SWA **S**liding **W**indow **A**ttention.

Glossar

BERT BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) ist ein vortrainiertes Sprachmodell von Google. Mit der Einführung wurde ein deutlicher Sprung im NLP und der Leistungsfähigkeit von Sprachmodellen erzielt.

Chain of Thought Der **C**hain **o**f **T**hought (COT) ist eine Gedankenkette, welche an ein LLM übergeben werden kann. Dabei wird das strukturierte und logische Vorgehen beschrieben, um eine Aufgabe zu lösen.

CNN Ein CNN (**C**onvolutional **N**eural **N**etwork) ist neuronales Netzwerk, welches vorwiegend zur Audio- und Bildverarbeitung eingesetzt wird..

Polarion Polarion ist ein Anforderungsmanagement Tool von Siemens, in dem die Dokumentation von Projekten verwaltet wird. Dort werden unter anderem Anforderungsdokumente, technische Spezifikationen und Testfälle abgelegt.

1 Einleitung

In diesem Kapitel werden die Ausgangslage, Motivation, das Umfeld, die Zielsetzung und der Aufbau der Arbeit dargestellt.

1.1 Ausgangslage und Motivation

Ein essenzieller Teil in der Projektarbeit ist das Projektmanagement, insbesondere das Anforderungsmanagement. Qualitativ hochwertige Anforderungsdokumente, technische Spezifikationen, sowie Testfälle stellen wichtige Komponenten für den Erfolg eines Projekts dar. Die Erstellung dieser Dokumente ist jedoch meistens mühselig und unbeliebt. Schlechte, unvollständige oder unspezifizierte Anforderungen verursachen immer Kosten am Ende eines Projekts. Ergebnisse werden nicht erreicht und es entsteht eine Unzufriedenheit beim Kunden.
Dem kann mit einem vernünftigen Anforderungsmanagement vorgebeugt werden. Im Anforderungsmanagement gibt es verschiedene Tätigkeiten, wie beispielsweise: [12]

- Erhebung und Identifikation von Anforderungen
- Strukturierte Dokumentation der ermittelten Anforderungen
- Überprüfung und Abstimmung der Anforderungen mit relevanten Stakeholdern
- Validierung der Anforderungen hinsichtlich Zielerreichung und Umsetzbarkeit

Vor allem das strukturierte und einheitliche Dokumentieren von Anforderungen ist entscheidend, damit die Erwartungen von unterschiedlichen Personen an ein Projekt nicht auseinanderlaufen. Die Projekte haben dabei häufig eine Vielzahl von Anforderungen, welche alle manuell durchgegangen und dokumentiert werden müssen. Dieser Prozess soll optimiert, sowie Mitarbeitende entlastet werden.

1.2 Umfeld

Diese Bachelorarbeit wird im Rahmen eines dualen Studiums bei der Siemens AG in Bremen erstellt. Die Bearbeitung erfolgt in der Abteilung RC-DE DI PA SO WFC MC-CRANES, die sich mit der Elektrifizierung und Automatisierung von Kranen, insbesondere in Hafenbereichen, beschäftigt. Zu den typischen Krantypen in Häfen zählen der **S**hip-**T**o-**S**hore Crane (STS), der **R**ubber-**T**ired **G**antry Crane (RTG) und der **R**ail-**M**ounted **G**antry Crane (RMG).

Derzeit werden Anforderungsdokumente, technische Spezifikationen und Testfälle manuell erstellt. Ziel dieser Bachelorarbeit ist, einen Teil dieser Tätigkeiten zu übernehmen und zu erleichtern. Es handelt sich zudem um die erste Anwendung, die ein **L**arge **L**anguage **M**odel (LLM) einsetzt, da solche Modelle in der Abteilung bisher kaum genutzt werden.

1.3 Zielsetzung

Das Ziel dieser Arbeit ist, die Erstellung von Projektmanagementdokumenten, insbesondere von Anforderungsdokumenten, zu vereinfachen. Das heißt, es werden die einzelnen Anforderungen aus einem Anforderungsdokument geschrieben . Zusätzlich soll eine Grundlage für künftige Arbeiten geschaffen werden, um weitere Dokumente, wie technische Spezifikationen und Testfälle, automatisiert zu generieren. Die Generierung soll mit einem LLM realisiert werden, da auch in diesem Bereich Kompetenzen aufgebaut werden sollen. Es soll hier ebenfalls darauf geachtet werden, dass auf den Einsatz von LLMs aufgebaut werden kann, um bei weiteren Projekten und Anwendung generative **K**ünstliche **I**ntelligenz (KI) einzusetzen. Dabei geht es nicht um das Trainieren eines LLMs. Es werden bereits vorhandene Modelle eingesetzt und evaluiert, wie gut diese die Aufgabe lösen.

1.4 Aufbau der Arbeit

Diese Arbeit ist in acht Kapitel gegliedert. Zunächst beschäftigt sich Kapitel 2 mit den Grundlagen dieser Arbeit. Anschließend werden in Kapitel 3 die Anforderungen an diese Arbeit definiert. In Kapitel 4 und Kapitel 5 geht es um das Konzept und die Implementierung, um diese Anforderungen umzusetzen. Inwiefern die Anforderungen am Ende

umgesetzt wurden, wird in Kapitel 6 evaluiert. Zum Schluss folgt ein Fazit mit einem Ausblick auf weitere Arbeiten.

2 Grundlagen

Dieses Kapitel behandelt die fundamentalen Grundlagen, die für die weitere Arbeit von Bedeutung sind. Zunächst wird ein allgemeiner Überblick über maschinelles Lernen gegeben, gefolgt von einer Einführung in die Computerlinguistik. Hierbei wird erläutert, wie natürliche Sprache verarbeitet wird und was unter dem Begriff „Token“ zu verstehen ist. Anschließend wird das Thema LLM behandelt, welche Fähigkeiten sie besitzen und welche davon relevant für diese Arbeit sind, sowie das Thema KI-Halluzination. Im weiteren Verlauf werden die Transformer-Architektur, Aufmerksamkeitsmechanismen, die Bewertung von LLMs und das Prompt Engineering beschrieben. Es folgt ein Abschnitt zu Markdown. Das Format wird zum einen dafür verwendet den Kontext für ein LLM bereitzustellen und zum anderen eine verarbeitbare Antwort zu erhalten. Abschließend wird kurz auf den Architekturstil REST, welcher die Schnittstellen zum Requirement Management Tool bildet, eingegangen.

2.1 Maschinelles Lernen

Das Feld der KI gewinnt heutzutage zunehmend an Bedeutung und Popularität.[1] Große Fortschritte in der KI, wie beispielsweise ChatGPT und BERT, haben die vielfältigen Möglichkeiten und Anwendungsbereiche von KI verdeutlicht. Ein bedeutender Teilbereich der KI ist das maschinelle Lernen, zu dem auch LLMs, auf denen ChatGPT und BERT basieren, gehören.[2] [2]
Maschinelles Lernen beschreibt die Fähigkeit von Computern, aus Daten Vorhersagen oder Entscheidungen zu treffen. Hierbei wird ein Modell mit bereits vorhandenen Daten trainiert, um daraus spezifische Ergebnisse zu generieren. Für die Datenverarbeitung und das Training stehen zahlreiche Modelle zur Verfügung. Die Einsatzmöglichkeiten

[1] https://de.statista.com/statistik/kategorien/kategorie/15/themen/2604/branche/kuenstliche-intelligenz/#overview, Zugriffsdatum: 25.02.2025
[2] https://www.iais.fraunhofer.de/de/geschaeftsfelder/big-data-analytics-and-intelligence/innovation-briefing-generative-ki.html, Zugriffsdatum: 28.05.2025

des maschinellen Lernens sind äußerst vielfältig. In der Medizin wird es beispielsweise zur Diagnoseunterstützung eingesetzt, im Telekommunikationsbereich zur Analyse von Anrufmustern und in den Bereichen Physik, Astronomie und Biologie zur Analyse großer Datenmengen. Wie oben bereits erwähnt, gehören ChatGPT und BERT ebenfalls zum Bereich des maschinellen Lernens. Genauer gesagt fallen sie in den Bereich der generativen KI, welcher sich mit dem Erzeugen von Inhalten beschäftigt. [2]

2.2 Generative KI

Generative KI befasst sich mit Modellen, die Inhalte oder Daten erzeugen, die nicht im Trainingsdatensatz enthalten waren, aber dennoch kohärent sind und die gleiche Struktur aufweisen. Die wichtigsten Merkmale der generativen KI sind die **Datengenerierung**, die **Synthese** und die **Lernverteilung**. Die Datengenerierung bezieht sich auf die Fähigkeit, neue Daten zu erzeugen, die nicht im Trainingsdatensatz vorhanden waren. Dies wird auch als „produktiv“ bezeichnet. Es werden Daten produziert, welche es zuvor nicht gegeben hat. Dies können Texte, Bilder, Audios, Videos und vieles mehr sein. Synthese bedeutet, dass das generative Modell in der Lage ist, mehrere verschiedene Eingaben zu kombinieren, beispielsweise zwei Bilder zu einem zusammenzuführen. Die Lernverteilung beschreibt die Fähigkeit eines generativen KI-Modells, Wahrscheinlichkeitsverteilungen aus den Trainingsdaten zu erlernen, um daraus neue Muster zu generieren.
Generative KI unterscheidet sich vor allem zu anderen KIs in der Hinsicht, dass dabei versucht wird, neue Daten zu synthetisieren und zu generieren, während die meisten traditionellen KI Modelle Daten klassifizieren und analysieren. [4, 17]

2.3 Natural Language Processing

Damit KI schlüssige Texte generieren kann, muss die KI lernen, wie zusammenhängende Sätze geschrieben werden. Diese Verarbeitung von natürlicher Sprache durch Maschinen wird als **N**atural **L**anguage **P**rocessing (NLP) bezeichnet. Dabei wird nicht nur die Semantik eines einzelnen Wortes analysiert, sondern der gesamte Kontext betrachtet. Es geht dementsprechend um die Verarbeitung von ganzen Sätzen, bzw. Texten. [9, 11]

2.3.1 Vorverarbeitung natürlicher Sprache

Um die natürliche Sprache für ein Modell verarbeitbar zu machen, muss sie zunächst vorverarbeitet werden. Dazu wird der Eingabetext vektorisiert. Das heißt, der Text wird in diskrete Sprachelemente, wie Wörter, zerlegt und in Vektoren geschrieben. Dies wird auch Tokenisierung genannt. [9, 11]

Beispiel (vgl. Kapitel 2, Abschnitt 2.1, S. 8, [9])

Eingabe:
'Die Sonne steht hoch am Himmel.'
Tokenisierung:
['Die', 'Sonne', 'steht', 'hoch', 'am', 'Himmel', '.']

Moderne und bekannte LLMs, wie der Chatbot ChatGPT von OpenAI, arbeiten nicht mit einzelnen Wörtern als Token, sondern unterteilen diese in noch kleinere Einheiten. Diese Token können sehr unterschiedlich sein, wie beispielsweise „dar“, „ett“, „kennenlernen“ oder auch einzelne Buchstaben. Der Grund dafür liegt in der enormen Anzahl von Wörtern (im Duden befinden sich 13 Millionen deutsche Wörter). Zudem können dadurch auch rhetorische Stilmittel wie z. B. Neologismen besser erfasst werden. [17]

2.3.2 Berechnung des nächsten Wortes

Im Unterabschnitt 2.3.1 wurde ein Eindruck gegeben, wie natürliche Sprache aufbereitet wird, um es für ein LLM verarbeitbar zu machen. In diesem Abschnitt wird beschrieben, wie aus einer Eingabe eine sprachlich und inhaltlich sinnvolle Ausgabe generiert und berechnet wird. Dafür wird lediglich das grundlegende Prinzip beschrieben und nicht genau im Detail erläutert.
Prinzipiell wird zur Berechnung des nächsten Wortes bzw. für das nächste Token eine Wahrscheinlichkeit für jedes mögliche Token berechnet. Das Token mit der größten Wahrscheinlichkeit wird ausgegeben. Für ein besseres Verständnis wird im Folgenden nicht mehr von Token, sondern von Wörtern gesprochen.
Zur Berechnung des nächsten Wortes werden zunächst alle Eingabewörter eingelesen und nach einer Verarbeitung durch ein mehrschichtiges neuronales Netz in einem internen Format dargestellt. Dies wird auch Kontext genannt. Nachdem jedes Eingabewort verarbeitet wurde und in den Kontext eingeflossen ist, wird für jedes mögliche Wort eine

Wahrscheinlichkeit erstellt. Geht man beispielsweise davon aus, dass das LLM 100.000 Wörter kennt, wird für jedes dieser 100.000 Wörter eine Wahrscheinlichkeit zwischen 0 und 1 erstellt. Die Summe aller 100.000 Wahrscheinlichkeiten ergibt 1. Dementsprechend wird nicht direkt das nächste Wort, sondern nur eine Wahrscheinlichkeit für die einzelnen Wörter berechnet. Zudem müssen die Wahrscheinlichkeiten für die nächsten Wörter immer neu berechnet werden und können nicht für bestimmte oder alle Fälle gespeichert werden. Andernfalls wäre eine Datenmenge enormer Größe notwendig, um die möglichen Fälle abzudecken. Zudem kann das nächste Wort mit der größten Wahrscheinlichkeit von dem am häufigsten verwendeten nächsten Wort abweichen. [17]

2.4 Large Language Model

Das Fraunhofer-Institut für experimentelles Software-Engineering beschreibt LLMs folgendermaßen: „Large Language Models (kurz: LLM und auf Deutsch: Große Sprachmodelle) sind leistungsstarke Modelle, die darauf ausgelegt sind, menschliche Sprache zu verstehen und zu generieren. Sie können Texte analysieren und verstehen, kohärente Antworten generieren und sprachbezogene Aufgaben ausführen“.[3] Moderne LLMs sind mittlerweile ausschließlich neuronale Netze und sind damit ein Teil des maschinellen Lernens. Es gibt zudem LLMs, welche keine Texte, sondern Bilder, Audios oder Ähnliches generieren. [4, 17]

2.4.1 Fähigkeiten von LLMs

Im Folgenden befindet sich ein Überblick über die Fähigkeiten von LLMs. Die Fähigkeiten werden in **nützliche** und **elementare** Fähigkeiten unterteilt. Die **nützlichen Fähigkeiten** sind diejenigen, die von den Anwendenden direkt genutzt werden können. Die **elementaren Fähigkeiten** hingegen sind grundlegende Funktionen, ohne die die nützlichen Fähigkeiten nicht funktionieren würden. Diese elementaren Fähigkeiten sind abstrakter und werden von den Anwendenden nicht direkt genutzt.
In Bezug auf diese Arbeit sind nicht alle Fähigkeiten relevant, der Vollständigkeit halber werden dennoch alle Text- und Bildbezogenen Fähigkeiten aufgelistet. Im Anschluss wird auf die unterstrichenen, vom Verfasser ausgewählten, **nützlichen Fähigkeiten** genauer

[3] https://www.iese.fraunhofer.de/blog/large-language-models-ki-sprachmodelle/, Zugriffsdatum: 26.03.2025

eingegangen. Diese bilden im späteren Verlauf einen großen Anteil an der Nutzung eines LLMs.

Nützliche Fähigkeiten

- Text generieren
- Text umschreiben
- Text übersetzen
- Text analysieren, erklären und interpretieren
- Text zusammenfassen
- Themen und Konzepte erklären
- Vorschläge machen
- Probleme lösen
- Gespräche führen
- Bilder generieren
- Bilder beschreiben

Elementare Fähigkeiten

- Plausible Worte vorhersagen
- Allgemeine Erwartungen von Benutzern erfüllen
- Aufforderungen verstehen
- Langreichweitige Beziehungen berücksichtigen
- Große Kontexte berücksichtigen und Polysemie auflösen
- Ähnlichkeiten erkennen
- Analogien und Beispiele verstehen und generalisieren
- Koreferenzen auflösen, insbesondere Pronomen
- Redewendungen und uneigentliche Rede
- Grammatik und Morphologie

Die **Textgenerierung** ist ein wesentlicher Aspekt von LLMs. Dabei geht es nicht nur um das Generieren von einzelnen Wörtern oder Sätzen, sondern um das Erstellen eines längeren Texts zu einem bestimmten bzw. vorgegebenen Themenbereich.
Beim **Umschreiben von Texten** ist ein Text bereits vorgegeben und das LLM ist in seinen Antwortmöglichkeiten eingeschränkt, da der Inhalt durch den Text gegeben ist. Zu dem Umschreiben gehört auch die sprachliche Korrektur von Texten.
Eine spezielle Form des Umschreibens von Texten ist das **Übersetzen von Texten**. Dabei werden Texte in einer anderen Sprache „umgeschrieben“.
Texte analysieren, erklären und interpretieren umfasst drei sehr ähnliche und sich

überschneidende Fähigkeiten. Das Analysieren von Texten ist dabei die Voraussetzung für das Erklären und Interpretieren von Texten. LLMs analysieren ständig Texte, da sie eine Voraussetzung für fast alle Fähigkeiten ist. Dabei setzen sie die Wörter und ihre Bestandteile in komplexer werdende Beziehungen zueinander. Die Erklärung ist eine Art des Umschreibens in andere, eigene Wörter, um den Text verständlich zu machen.
Um einen **Text zusammenzufassen,** muss das LLM Texte verstehen und die wichtigsten Informationen erkennen können. LLMs werden häufig für das Zusammenfassen von Texten genutzt, daher werden sie in dieser Hinsicht nach trainiert. Zum Schluss gibt es die Fähigkeit **Bilder zu beschreiben**. Das erkennen von Elementen in einem Bild beruht nicht auf der Transformer-Architektur, sondern CNNs. Dieses gibt die Informationen über die Elemente an das LLM weiter, welches anschließend diese in Beziehung zu einander setzt, um das Bild zu beschreiben. [17]

2.4.2 KI-Halluzination

Als KI-Halluzination wird das Generieren von Daten bezeichnet, welche plausibel erscheinen, jedoch frei erfunden sind. Hervorgerufen wird dieses Phänomen durch unvollständige, verzerrte oder anderweitig fehlerhafte Trainingsdaten.[4] Die folgenden Methoden sind Beispiele, wie KI-Halluzination verhindert, bzw. eingeschränkt werden kann: [4]

1. **Feedback**
   Es besteht die Möglichkeit, dem Modell Feedback zu einer Antwort zu geben und dem Modell mitzuteilen, welche Informationen in die Antwort sollen und welche nicht.

2. **Faktencheck**
   Mittels Faktenchecks der Ergebnisse eines Modells können Halluzinationen erkannt und markiert werden.

3. **Prompt Engineering** Das Prompt Engineering kann Halluzination ebenfalls vorbeugen. Wichtig sind dabei klare, spezifische und informative Prompts, welche zu genaueren Ergebnissen führen. Methoden des Prompt Engineerings werden in Abschnitt 2.7 beschrieben.

[4] https://cloud.google.com/discover/what-are-ai-hallucinations?hl=de, Zugriffsdatum: 25.03.2025

Die Punkte 1 und 3 beschreiben Methoden zur Reduzierung von Halluzinationen während der Nutzung eines LLMs. Punkt 2 bezieht sich auf das Identifizieren von Halluzinationen und fehlerhaften Ergebnissen, die Methode dient nicht dazu das Auftreten von Halluzinationen zu verhindern. Trotz dieser Methoden zur Reduzierung und Erkennung von Halluzinationen können diese nicht vollständig verhindert oder erkannt werden. Deshalb muss bewusst und kritisch mit KI-generierten Inhalten umgegangen werden. [4]

2.5 Transformer-Architektur

Die Transformer-Architektur ist ein Eckpfeiler bei den sequence-to-sequence Aufgaben. Transformer transformieren eine Eingabesequenz in eine Ausgabesequenz. Die elementaren Schichten der Transformer-Architektur sind in Abbildung 2.1 dargestellt. Diese Elemente werden in den folgenden Abschnitten genauer beschrieben. Transformer sind ein zentrales Element moderner LLMs und bestehen aus zwei Hauptkomponenten: Dem **Encoder** und dem **Decoder**. [10, 20]

2.5.1 Encoder

Der Encoder hat die Aufgabe, die Eingangssequenz zu verarbeiten und die komprimierten Informationen in einem Kontext dem Decoder zur Verfügung zu stellen. Dabei wird nicht nur die Information über die Bedeutung jedes einzelnen Wortes weitergegeben. Es werden zusätzlich auch die Informationen der vorherigen Worte berücksichtigt. Der Encoder besteht dabei aus mehreren identischen Schichten. Diese haben jeweils zwei Hauptelemente, welche in Abbildung 2.1 zu sehen sind:

- **Multi-Head Attention:** Dieses Element des Encoders erlaubt es dem Modell, mit jedem Attention-Head unterschiedliche Bereiche der Eingabedaten zu berücksichtigen und somit verschiedene Aspekte der Daten zu erkennen.
- **Feed-Forward Neural Network:** Ein neuronales Netzwerk, das die Attention-Vektoren verarbeitet, nichtlineare Transformationen durchführt und diese für die nachfolgende Encoder-Schicht sowie die Decoder-Schicht zugänglich macht.
- **Add&Norm:** Die Schicht Add&Norm normiert die Ausgabe einer Schicht und macht diese für die nächste Schicht verfügbar. Sie befindet sich sowohl im Encoder, als auch Decoder hinter jedem Hauptelement. [10, 20]

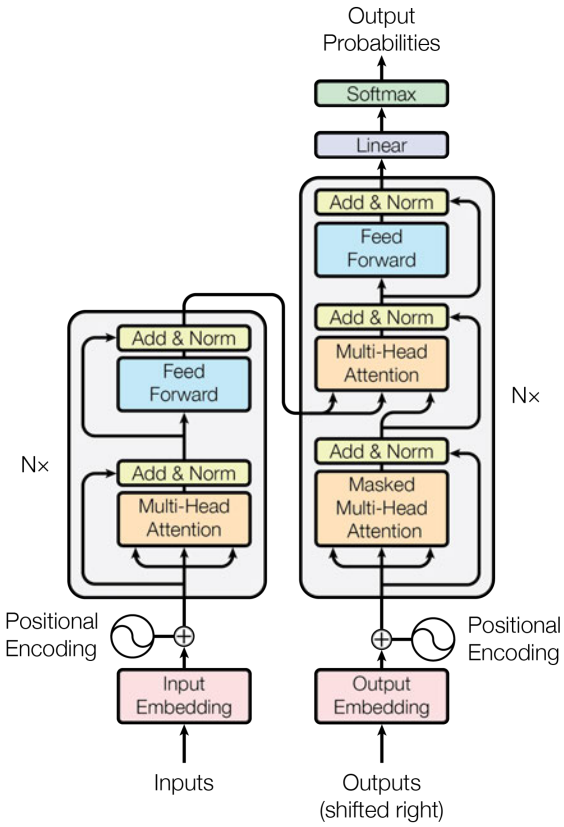

Abbildung 2.1: Der Transformer - Modellarchitektur (vgl. Kapitel 3, S.3, [20])

2.5.2 Decoder

Der Decoder erzeugt die Ausgabesequenz aus dem übergebenen Kontext des Encoders. Dabei wird ähnlich wie beim Encoder nicht nur der übergebende Kontext vom Encoder betrachtet, sondern auch die zuvor generierten Token. Diese haben einen Einfluss auf das nächste Token. Auch der Decoder besteht aus mehreren identischen Schichten, wobei jede Schicht drei Hauptelemente besitzt:

- **Masked Multi-Head Attention:** Diese Schicht funktioniert ähnlich wie die Multi-Head Attention, jedoch wird durch einen Masking-Mechanismus sichergestellt, dass die Generierung eines Wortes nicht von zukünftigen Worten abhängt.
- **Encoder-Decoder Attention:** Die Encoder-Decoder Attention sorgt für den Fokus auf die relevanten Eingabesequenzen durch den übergebenen Kontext des Encoders.

- **Feed-Forward Neural Network:** Es handelt sich um die gleiche Schicht wie beim Encoder.
- **Add&Norm:** Wie bereits in Unterabschnitt 2.5.1 beschrieben, ist diese Schicht identisch zu der im Encoder. [10, 20]

2.6 Aufmerksamkeitsmechanismen

Mit Hilfe von Aufmerksamkeitsmechanismen kann der Fokus von LLMs auf die relevanten Eingabeinformationen gelenkt werden. Dadurch lässt sich die Geschwindigkeit von LLMs erhöhen, oder es kann ein längerer Kontext betrachtet werden. In Abschnitt 2.5 wurde die Multi-Head-Attention kurz eingeführt. In diesem Abschnitt werden mit, **G**rouped **Q**uery **A**ttention (GQA) und **S**liding **W**indow **A**ttention (SWA), zwei weitere Aufmerksamkeitsmechanismen beschrieben.

2.6.1 Grouped Query Attention

Die GQA unterteilt mehrere Abfragen (*Query*) in einzelne Gruppen. Jeder Gruppe wird dabei ein Schlüssel (*Key*) und ein Wert (*Value*) zugewiesen. Somit wird nicht mehr jeder Abfrage ein Schlüssel und ein Wert zugewiesen, wie beispielsweise bei der Multi-Head Attention. Dies reduziert die Anzahl der Berechnungen. Das LLM kann mit der GQA schneller eine Antwort generieren, ohne dass diese groß an Qualität verliert. [1]

2.6.2 Sliding Window Attention

Mit der SWA können lange Sequenzen eines LLMs verarbeitet werden. Dabei wird die Aufmerksamkeit auf bestimmte Token innerhalb einer definierten Fenstergröße w gelegt. Das Fenster (*Window*), mit einer definierten Größe, wird auf die Eingabesequenz, sowie über die Transformer-Schichten (*Layer*), gelegt. Die Token können sich nur innerhalb dieses Fensters betrachten und ignorieren alle Token außerhalb des Fensters. Das Fenster liegt $\frac{1}{2}w$ Token links und rechts neben dem betrachteten Token. [3, 6]

2.7 Prompt Engineering

Das Prompt Engineering bezeichnet das Schreiben von zielgerichteten Anweisungen für ein LLM, um eine konsistente Antwort zu erhalten, die den gewünschten Anforderungen entspricht. [13, 16, 18]

2.7.1 Kontextbezogene Prompts

Damit das LLM eine Aufgabe korrekt lösen oder eine Frage richtig beantworten kann, kann es notwendig sein, dem LLM einen Kontext, bezogen auf die Aufgabe, zu übergeben. In dem Kontext werden Informationen übergeben, die erforderlich sind, um die Aufforderung nach den Wünschen der Nutzenden zu lösen. [18]

Beispiel

Prompt: „Isa hat am 30. September Geburtstag. An welchem Wochentag hat Isa dieses Jahr Geburtstag?“

Ohne den Kontext zum Geburtsdatum von Isa kann die nachfolgende Frage nicht vom LLM gelöst werden.

2.7.2 Zero-shot und Few-shot

Es gibt zwei Methoden, Aufgaben an ein LLM zu stellen. Einmal wird dem LLM eine Aufgabe/ Frage optional mit einem Kontext gestellt, wobei jedoch keine Beispiele übergeben werden. Die andere Methode ist es eine Aufgabe/ Frage mit Beispiel zu liefern.

Zero-shot Das Zero-shot Prompting beschreibt Prompts, die ohne Beispiele erstellt werden. Das LLM muss die Aufgabe nur anhand seines Wissens und des überlieferten Kontexts lösen.
Few-shot Im Gegensatz zum Zero-shot Prompting werden beim Few-shot Prompting dem LLM Beispiele, neben der Aufgabe, übergeben. [13, 16]

2.7.3 Structured Output

Es ist möglich, LLMs immer in einem gleichen Format antworten zu lassen. Dies wird Structured Output genannt. Dabei kann das Antwortformat vom Nutzenden festgelegt werden. Ein Structured Output hat die Vorteile, dass keine falsch formatierten Antworten überprüft werden müssen, die Prompts nicht wesentlich erweitert werden, um ein einheitliches Format zu erhalten und [14]

2.7.4 Fine-Tuning

Mit dem Fine-Tuning lassen sich die Antworten von LLMs in Eigenschaften Kreativität, Kohärenz und Länge kontrollieren. Nachkommend werden die Funktionen „Temperature“, „Top_p“ und „Max Tokens“ kurz beschrieben.

Temperature Die Temperatur eines LLMs beschreibt die Zufälligkeit der Antwort. Bei höheren Werten, ab 0,8, ist die Antwort kreativer und vielfältiger. Dies hat den Nachteil, dass die Antwort auch zufälliger wird und bei einer zu hohen Temperatur die Sinnhaftigkeit verliert. Kleine Temperaturen (z. B. 0,2) führen zu zielgerichteten und mehr deterministischen Antworten. Die Antwort wird vorhersehbarer. Für die Temperatur sind Werte zwischen 0 und 2 möglich.

Top_p Neben der Temperatur kann die Zufälligkeit der Antwort auch durch die Funktion „Top_p“ gesteuert werden. Diese Technik wählt nicht das wahrscheinlichste nächste Token, sondern berücksichtigt eine Gruppe der wahrscheinlichsten Tokens. Möglich sind Werte zwischen 0 und 1. Bei höheren Werten wird die Antwort, ähnlich wie bei der Temperatur, zufälliger. Im Gegensatz dazu führt die Wahl kleinerer Werte zu einer deterministischeren Antwort.

Max Tokens Neben der Temperatur und dem Top_p lässt sich mit der Funktion „Max Tokens“ die Antwortlänge einstellen. Dadurch können kurze Antworten erzwungen werden. [18]

2.7.5 Rollen eines Prompts

Für Modelle, die mit der OpenAI API kompatibel sind, existieren drei unterschiedliche Nachrichtenrollen.[5]

- **System:** Dies sind Nachrichten, die Regeln und Vorgehensweisen für die Antwort vergeben.
- **User:** Dies sind die Nachrichten vom Benutzer. Das können Fragen oder Aufgaben sein.
- **Assistant:** Assistant Nachrichten sind die Antworten des LLMs.

2.8 Bewertung von LLMs

Aufgrund des großen Umfangs von LLMs und des Mangels an menschlichen Präferenzen in vorhandenen Benchmarks, ist es schwer, chatfähige LLMs zu bewerten. [22] Es gibt eine Reihe von Benchmarks, um die Leistungsfähigkeit von LLMs zu bewerten, bevor diese genutzt werden. Dabei enthalten diese Benchmarks Beispieldaten, verschiedene Fragen und Aufgaben, ein Bewertungssystem sowie ein Punktesystem.[6] Im Folgenden werden die geläufigen Benchmarks MT-Bench, **M**assive **M**ultitask **L**anguage **U**nderstanding (MMLU) und HellaSwag betrachtet, da diese vom Verfasser dieser Arbeit als relevant für die späteren Aufgaben angesehen werden.

2.8.1 MT-Bench

Die geläufigen Benchmarks zur Bewertung von LLMs vernachlässigen größtenteils die menschlichen Präferenzen. Dabei werden LLMs hauptsächlich für die Interaktion zwischen Mensch und KI eingesetzt. Der im Jahr 2023 veröffentlichte Benchmark MT-Bench bewertet LLMs im Hinblick auf diese menschlichen Präferenzen.
Bei dem MT-Bench werden dem LLM 80 hochqualitative mehrteilige Fragen gestellt. Zudem werden mehrstufige Konversationen und die Fähigkeit zur Befolgung von Anweisungen getestet. Um die gängigen Anwendungsfälle von Chatbots abzudecken, werden

[5] https://platform.openai.com/docs/guides/text?api-mode=chat, Zugriffsdatum: 30.05.2025
[6] https://www.ibm.com/think/topics/llm-benchmarks, Zugriffsdatum: 21.05.2025

User Prompts in acht Kategorien aufgeteilt: Schreiben, Rollenspiel, Extraktion, logisches Denken, Mathematik, Codierung, Wissen I (MINT) und Wissen II (Geistes-/ Sozialwissenschaften). Für jede Kategorie werden zehn mehrteilige Fragen gestellt.

Im Anhang (Abbildung A.3, A.4) ist dargestellt, wie verschiedene LLMs bei dem Benchmark abgeschnitten haben. Abbildung A.3 zeigt dabei die Ergebnisse in den acht verschiedenen Kategorien, während Abbildung A.4 vergleicht, wie gut einzelne Modelle in den verschiedenen Benchmarks MMLU, TruthfulQA und MT-Bench abschneiden.

2.8.2 MMLU

Der MMLU Benchmark wurde 2021 veröffentlicht und ist ein umfangreicher Test mit 57 Aufgaben aus verschiedenen Wissensbereichen und Schwierigkeitsstufen. Dazu zählen unter anderem die Bereiche Geistes-, Sozial- und Naturwissenschaften. Er enthält über 15.000 Multiple-Choice-Fragen, die von Studierenden aus frei verfügbaren Online-Quellen gesammelt wurden. Der Test dient dazu, die Fähigkeit von LLMs im Hinblick auf ihre Vielseitigkeit, den Umfang ihres globalen Wissens und der Entwicklung anspruchsvoller Problemlösungsstrategien zu bewerten. Während unspezialisierte Personen etwa 34,5 % Genauigkeit erreichen, liegt die geschätzte Leistung von spezialisierten Personen bei rund 89,8 %. Für medizinische Prüfungen liegt die Genauigkeit für spezialisierte Personen beispielsweise bei 87 %. Der Benchmark gilt als anspruchsvolles und langfristig relevantes Ziel für die Entwicklung leistungsfähiger Sprachmodelle. [8]

Im Anhang (Abbildung A.5) sind die Ergebnisse verschiedener LLMs für den MMLU Benchmark dargestellt. Dabei sind die Ergebnisse auf die vier verschiedenen Themen Geisteswissenschaften, Sozialwissenschaft, STEM und „Andere“, sowie der Durchschnitt aller Themen, abgebildet. Da der Benchmark aus dem Jahr 2021 stammt, sind Modelle wie DeepSeek R1 oder GPT-4 nicht enthalten.

2.8.3 HellaSwag

Der HellaSwag Benchmark wurde im Jahr 2019 veröffentlicht und stellt eine verbesserte Version von SWAG dar. Er wurde gezielt entwickelt, um stilistische Verzerrungen zu vermeiden und die tatsächliche Fähigkeit zum Schlussfolgern mit Alltagswissen zu testen.

ActivityNet Captions Zur Bewertung wurde der Datensatz ActivityNet Captions integriert. Dieser beinhaltet zeitliche Beschreibungen und Aktivitätsbezeichnungen für jede Beschriftung. Dadurch soll die Generalisierungsfähigkeit getestet werden.

WikiHow Testfeld Neben ActivityNet Captions wird auch das Testfeld WikiHow betrachtet. Es beinhaltet 80.000 Kontext- und Folgeabsätze, zu Themen wie „Wie man eine Origami-Eule bastelt“ oder „Wie man einen Banküberfall überlebt“. Die Folgeabsätze sind zwei Sätze lang, da sie für LLMs herausfordernd, aber für Menschen gut lösbar sind.

HellaSwag enthält spezielle Zero-shot Sätze, um zu prüfen, ob Modelle auf neue, unbekannte Themenbereiche verallgemeinern können. Das Thema Zero-shot wird im Folgenden Abschnitt 2.7 erläutert. Für jeden Validierungs- oder Testsatz gibt es zwei Teilmengen:

- 5k In-domain Beispiele, die aus bekannten Kategorien vom Training stammen
- 5k Zero-shot Beispiele, aus zufällig ausgewählten, zurückgehaltenen Kategorien [21]

Abbildung A.6 aus dem Anhang zeigt die Ergebnisse verschiedener Modelle, sowie vom Menschen. Die Ergebnisse sind aufgeteilt, in ein generelles Ergebnis über den gesamten Benchmark, die der In-Domain und Zero-shot Teilmenge sowie den Datensätzen ActivityNet und WikiHow. Es lässt sich erkennen, dass der Mensch, in allen Kategorien, wesentlich besser abschneidet als die LLMs.

2.9 Markdown

Um dem LLM eine strukturierte Antwort zu übergeben und eine ebenso strukturierte Antwort zurück zu bekommen, Markdown verwendet werden. Markdown ist ein Textformat, welches das Ziel hat, möglichst einfach lesbar und schreibbar, zu sein. Dementsprechend ist die Syntax nah an das Zieldesign angelehnt und Listen sehen beispielsweise im Markdownformat bereits aus wie Listen. Jedoch hat es im Vergleich zu z. B. HTML wesentlich weniger Umfang und Möglichkeiten, Text darzustellen. Dies ist für diese Arbeit auch nicht notwendig, da Markdown lediglich zur Darstellung von **P**ortable **D**ocument **F**ormat (PDF) Dokumenten und zur Formulierung der Prompts genutzt werden soll.

Für diese beiden Einsatzmöglichkeiten müssen Überschriften, Tabellen, Listen, horizontale Linien, sowie Absätze dargestellt werden können, was mit Markdown möglich ist. Überschriften werden mit ein bis sechs Rauten („#“) dargestellt, wobei die eine Raute für die erste Ebene steht und sechs Rauten für die sechste. Tabellen haben kein explizites Format. Die Linien werden mit Hilfe der Symbole „|“ und „-“ dargestellt und der Text passend dazwischen geschrieben. Unsortierte Listen können mit den Symbolen „*“, „+“ und „-“ dargestellt werden, während bei sortierten Listen die Nummer davor steht, z. B. „1.“ oder „2.“.[7]

Beispiel

```
# Überschrift
## Unterabschnitt
* Liste
* Liste
* Liste
## Sortierte Liste
1. Liste
2. Liste
3. Liste
## Horizontale Linie
---
```

2.10 Representational State Transfer

Representational **S**tate **T**ransfer (REST) ist ein Architekturstil für den Aufbau webbasierter APIs. Es basiert dabei auf sechs Prinzipien:

- **Einheitlichkeit der Schnittstelle:** Die Einheitlichkeit der Schnittstelle vereinfacht die Systemarichtektur und verbessert die Transparenz der Datenflüsse. Bei einer HTTP basierten REST API werden beispielsweise die standardmäßigen HTTP-Methoden (GET, POST, PUT, DELETE etc.) verwendet.
- **Client und Server:** Durch die Trennung von Client und Server können sich beide Komponenten unabhängig voneinander entwickeln.

[7] https://daringfireball.net/projects/markdown/, Zugriffsdatum: 02.05.2025

- **Zustandslosigkeit:** Mit der Zustandslosigkeit wird beschrieben, dass bei einer Anfrage alle notwendigen Informationen enthalten sein müssen. Es werden keine Informationen von vorherigen Anfragen gespeichert.
- **Zwischenspeicherung:** Bei einer Antwort an den Client ist die Information zur Zwischenspeicherung enthalten. Diese berechtigt den Client, bei Zustimmung, die Antwort für einen bestimmten Zeitraum zwischenzuspeichern, um auf ähnliche Anfragen zu reagieren.
- **Mehrschichtiges System:** Das mehrschichtige System sorgt für einen Aufbau, bei dem jede Schicht nur Informationen aus der unmittelbar nächsten Schicht erhält, mit welcher sie interagiert.
- **Code auf Anfrage:** Durch den Code auf Anfrage können Clients Code vom Server herunterladen und ausführen. Dadurch verringert sich die Anzahl der implementierten Funktionen des Clients.[8]

[8] https://restfulapi.net/, Zugriffsdatum: 24.05.2025

3 Anforderungsanalyse

Die Anforderungsanalyse ist essenziell für den Erfolg eines Projekts. Sie dokumentiert die Anforderungen und das Projektziel und stellt sicher, dass diese klar verstanden werden. Zunächst wird dafür der Systemkontext, für einen ersten Überblick über das System, beschrieben. Anschließend werden die wichtigsten Stakeholder betrachtet, um die Erwartungen der interessierten Personen an das Projekt bestmöglich zu erfüllen. Im nächsten Schritt werden die Anwendungsfälle betrachtet, aus denen sich später die konkreten Anforderungen bilden. Durch eine gründliche Anforderungsanalyse wird sichergestellt, dass die Bedürfnisse der Interessengruppen verstanden werden.

3.1 Systemkontext

Für ein besseres Verständnis des Themas der Arbeit, wird in diesem Abschnitt die Umgebung des Systems beschrieben. Dabei wird auf einzelne Komponenten der Umgebung und deren Bezug auf das System eingegangen. Des Weiteren wird definiert, auf welche Elemente Einfluss genommen werden kann und welche lediglich verwendet werden, jedoch nicht beeinflussbar sind. Dabei wird über Schnittstellen die Umgebung mit dem System verknüpft. Abbildung 3.1 zeigt die wesentlichen Elemente des Systemkontexts, zur Erstellung von Dokumenten für das Projektmanagement.

Die Anwendung soll ein Anforderungsdokument von Siemens, technische Spezifikationen und Testfälle, automatisch generieren. Dazu wird ein Anforderungsdokument vom Kunden, aus einem externen Anforderungsmanagement Tool entnommen. Mit Hilfe eines LLM wird aus dem Anforderungsdokument vom Kunden ein Anforderungsdokument von Siemens generiert, in dem die einzelnen Anforderungen in einer Liste aufgeführt werden. Dieses Dokument wird anschließend wieder im Anforderungsmanagement Tool abgelegt und kann später von Mitarbeitenden der Abteilung kontrolliert und korrigiert werden. Zudem werden durch die Mitarbeitenden die einzelnen Anforderungen kategorisiert.

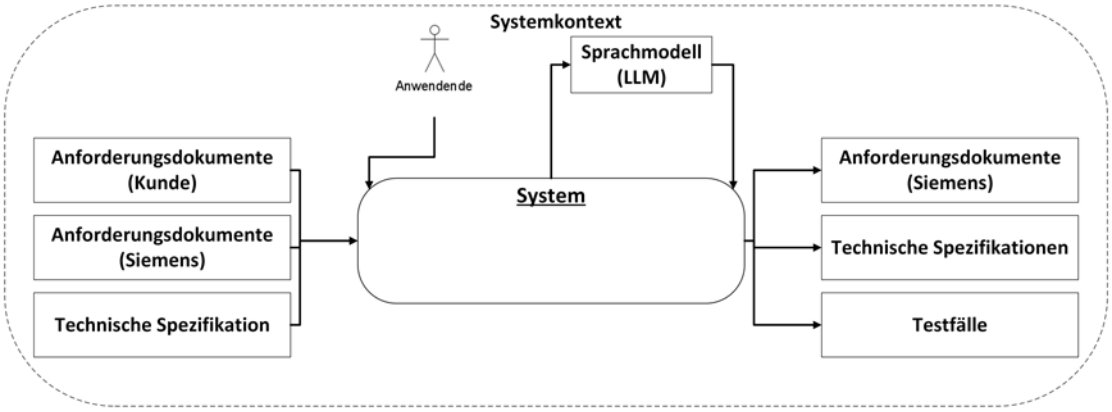

Abbildung 3.1: Systemkontext

Aus dem korrigierten und kategorisierten Anforderungsdokument von Siemens können, durch das LLM, technische Spezifikationen für die einzelnen Kategorien generiert werden. Die erstellten technischen Spezifikationen werden wieder im Anforderungsmanagement Tool abgelegt und können anschließend von Mitarbeitenden korrigiert und angepasst werden.
Zuletzt können aus einer technischen Spezifikation Testfälle erstellt werden. Diese werden ebenfalls mit Hilfe des LLMs generiert und anschließend im Anforderungsmanagement Tool abgelegt.

3.2 Stakeholder

Für das Projekt sind neben der technischen Umsetzung auch die Stakeholder ein wichtiger Faktor für das Gelingen des Projekts. Stakeholder sind Personen oder Personengruppen, welche ein Interesse an dem Projekt haben. Im Folgenden werden die wichtigsten Stakeholder kurz beschrieben.

3.2.1 Auftraggeber

Als Auftraggeber fungiert die Siemens AG, insbesondere die Abteilung RC-DE DI PA SO WFC MC-CRANES. Die Software wird für diese Abteilung entwickelt, um Arbeitszeit beim Erstellen der einzelnen Testfälle und beim Analysieren der Kundenanforderungen zu sparen. Sie haben daher ein besonderes Interesse am Erfolg der Arbeit.

3.2.2 Software-Entwickler

Der Software-Entwickler, der diese Bachelorarbeit verfasst, ist ein zentraler Stakeholder. Sein Ziel ist, eine funktionsfähige Lösung zu entwickeln und zu präsentieren. Dabei liegt sein Interesse nicht nur auf der Realisierung einer funktionierenden Lösung, welche den Hauptteil der Arbeit darstellt, sondern auch auf dem Erwerb neuer Fähigkeiten in Bezug auf künstlicher Intelligenz und dem Nutzen von LLMs.

3.2.3 Anwendende

Weitere Stakeholder sind die Personen, welche die Software später aktiv nutzen. Ihr Interesse besteht nicht nur darin, dass die Anwendung funktioniert, sondern auch darin, dass sie die geforderten Aufgaben zuverlässig erfüllt und dabei einfach sowie intuitiv zu bedienen ist. Zudem soll die Anwendung übersichtlich gestaltet sein und dem Anwendenden die Arbeit zur Erstellung von Projektmanagementdokumenten erleichtern.

3.2.4 Personen für die Weiterentwicklung

Die Ergebnisse dieser Arbeit sollen auch für die zukünftige Weiterentwicklung innerhalb der Abteilung nutzbar sein. Vor allem ist die Kommunikation mit einem LLM für künftige Projekte besonders interessant. Daher ist die Wiederverwendbarkeit der Schnittstelle zum LLM und eine umfassende Dokumentation erforderlich. Die Arbeit soll es ermöglichen, die Anwendung fertigzustellen, zu erweitern oder ähnliche Anwendungen zu erstellen, welche auf Basis eines großen Sprachmodells Aufgaben ausführen. In der Abteilung geht es hauptsächlich um die SPS-Programmierung. Hochsprachen kommen nicht alltäglich vor. Dennoch wird dabei vorwiegend die Entwicklungsumgebung VisualStudio in Verbindung mit der Programmiersprache C# verwendet.

3.3 Anwendungsfälle

In Abbildung 3.2 ist das Anwendungsfalldiagramm für die Anforderungsmanagement-Anwendung abgebildet. Dort werden die einzelnen Funktionalitäten der Anwendung übersichtlich dargestellt. Unten werden die Anwendungsfälle (AF) kurz beschrieben.

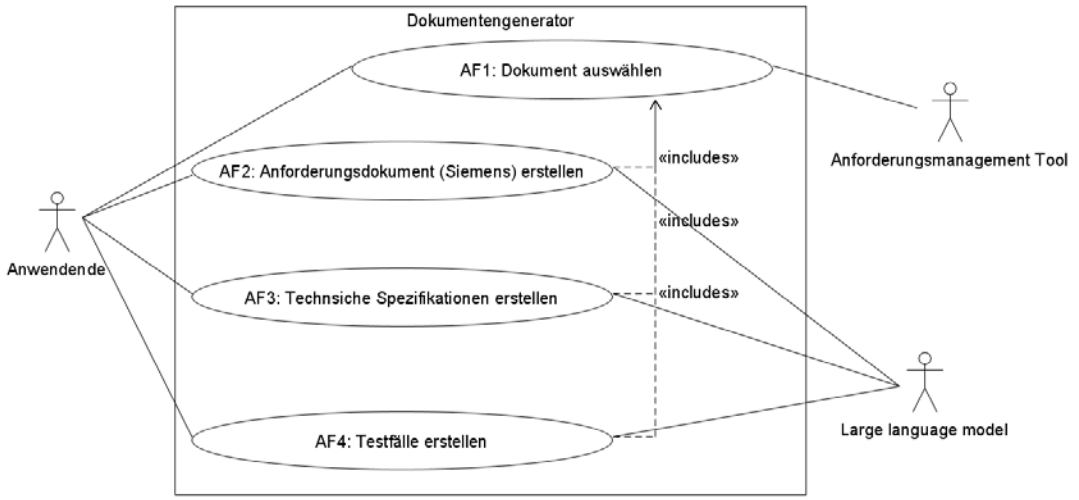

Abbildung 3.2: Anwendungsfalldiagramm für die Anforderungsmanagement Anwendung

AF1: Dokument auswählen Es wird ein Dokument aus dem Anforderungsmanagement Tool ausgewählt.

AF2: Anforderungsdokument (Siemens) erstellen Aus dem ausgewählten Dokument wird mit Hilfe eines LLM ein Anforderungsdokument von Siemens erstellt.

AF3: Technische Spezifikationen erstellen Aus dem ausgewählten Dokument werden mit Hilfe eines LLM technische Spezifikationen erstellt.

AF4: Testfälle erstellen Aus dem ausgewählten Dokument werden mit Hilfe eines LLM Testfälle erstellt.

3.4 Anforderungen

Bei den Anforderungen wird zwischen funktionalen und nicht funktionalen Anforderungen unterschieden. **Funktionale Anforderungen (F)** sind die Anforderungen, welche die Funktionalität der Anwendung beschreiben. Die **nicht funktionalen Anforderungen (NF)** sind hingegen Anforderungen, welche nicht die Funktion der Anwendung

beschreiben, sondern die Eigenschaften der Anwendung und wie die Funktionen umgesetzt werden. Die Priorisierung wird in eckigen Klammern dargestellt **([...])** und in zwei Arten unterteilt. **Muss** steht für die Anforderungen, ohne die die Funktionalität später nicht gegeben ist. Diese Anforderungen sind daher essenziell für den Erfolg dieser Arbeit. **Kann** steht für die Anforderungen, welche zum Projekt dazu gehören, das Projekt jedoch auch ohne diese funktioniert und ihren Zweck erfüllt.

Da die zu erstellenden Dokumente zum Großteil verschiedene Anforderungen haben, werden die Anforderungen im Folgenden in vier Unterkapiteln (Dokumentengenerator, Anforderungsdokumente, technische Spezifikationen und Testfälle) aufgeteilt.

3.4.1 Dokumentengenerator

Die Anforderungen für den **Dokumentengenerator (DG)** befassen sich mit den Anforderungen für die Umgebung der Anwendung. Hier wird festgehalten, was die Anwendung selbst für Anforderungen erfüllen muss.

DG-F1 [muss]: Aus dem Anforderungsdokument des Kunden wird ein Anforderungsdokument von Siemens generiert.
Diese Anforderung wird in Unterabschnitt 3.4.2 genauer beschrieben.

DG-F2 [kann]: Das Anforderungsdokument des Kunden wird aus dem Anforderungsmanagement Tool ausgewählt.

DG-F3 [muss]: Aus dem Anforderungsdokumenten von Siemens werden technische Spezifikationen generiert.
Diese Anforderung wird in Unterabschnitt 3.4.3 genauer beschrieben.

DG-F4 [kann]: Das Anforderungsdokument von Siemens wird aus dem Anforderungsmanagement Tool ausgewählt.

DG-F5 [muss]: Aus einer technischen Spezifikation werden Testfälle generiert.
Diese Anforderung wird in Unterabschnitt 3.4.4 genauer beschrieben.

DG-F6 [kann]: Die technische Spezifikation wird aus dem Anforderungsmanagement Tool ausgewählt.

DG-F7 [muss]: Die Anwendung besitzt eine grafische Benutzeroberfläche.

DG-F8 [muss]: Es ist möglich den zu generierenden Dokumententypen in der grafischen Benutzeroberfläche auszuwählen. Es gibt drei Dokumententypen: Anforderungsdokument, technische Spezifikation, Testfälle.

DG-F9 [kann]: Die grafische Benutzeroberfläche besitzt eine Auswahl, ob mit dem Anforderungsmanagement Tool oder lokal gearbeitet wird.

DG-F10 [muss]: Die Projektnummer wird von den Anwendenden eingegeben.

DG-F11 [muss]: Es muss ein lokales Dokument ausgewählt werden, aus dem das Projektmanagementdokument generiert wird.
Dieser Fall tritt nur ein, wenn der Nutzende zuvor ausgewählt hat lokal zu arbeiten.

DG-F12 [muss]: Die Nutzenden können ein Zielverzeichnis, für das generierte Dokument auswählen. Wird dieses nicht ausgewählt, wird das Dokument auf dem Desktop abgelegt.
Dieser Fall tritt nur ein, wenn der Nutzende zuvor ausgewählt hat lokal zu arbeiten.

DG-F13 [kann]: Die Testfälle werden im richtigen Projekt im Verzeichnis „Test cases“ im Anforderungsmanagement Tool abgelegt.
Dazu muss der Nutzende zuvor die Auswahl für das Anforderungsmanagement Tool getroffen haben (vgl. Anforderung DG-F9).

DG-F14 [kann]: Die technsichen Spezifikationen werden im richtigen Projekt im Verzeichnis „Technical functions“ im Anforderungsmanagement Tool abgelegt.
Dazu muss der Nutzende zuvor die Auswahl für das Anforderungsmanagement Tool getroffen haben (vgl. Anforderung DG-F9).

DG-F15 [kann]: Die technsichen Spezifikationen werden im richtigen Projekt im Verzeichnis „System Requirements“ im Anforderungsmanagement Tool abgelegt.
Dazu muss der Nutzende zuvor die Auswahl für das Anforderungsmanagement Tool getroffen haben (vgl. Anforderung DG-F9).

DG-NF1 [kann]: Alle generierten Dokumente sowie die Anwendung selber sind in Englisch (amerikanisch).
Da die Kunden weltweit verteilt sind und überwiegend außerhalb von Deutschland sitzen, ist die Kommunikations- und Dokumentationssprache Englisch (amerikanisch). Zudem sprechen nicht alle Personen der Abteilung deutsch, weshalb Englisch (amerikanisch) als Sprache notwendig ist.

DG-NF2 [muss]: Die Implementierung und Nutzung der Anwendung darf keine zusätzlichen Kosten verursachen. Ausgenommen hiervon sind lediglich die Lizenzkosten für Visual Studio, die während der Implementierungsphase anfallen dürfen.

DG-NF3 [muss]: Die Anwendung soll auf einem Windows-Betriebssystem mit Windows 10 oder höher kompatibel sein.

DG-NF4 [muss]: Die Dokumente werden mit Hilfe eines LLMs erstellt.
Da ein Ziel der Arbeit darin besteht, dass Kompetenzen im Bereich KI und speziell dabei in LLMs erworben werden, muss ein LLM auch eine wesentliche und sinnvolle Funktion der Anwendung übernehmen.

DG-NF5 [kann]: Die übergebenen Informationen an das LLM müssen vertraulich behandelt werden.
Für die Umsetzung des Projektes dieser Arbeit spielt die Vertraulichkeit keine Rolle, da die Anforderungsdokumente vom Kunden, welche als Kontext genutzt werden sollen öffentlich sind. Diese Anforderung bezieht sich, auf zukünftige Projekte, die auf Basis dieser Arbeit realisiert werden.

3.4.2 Anforderungsdokument

In diesem Abschnitt werden die Anforderungen für die Generierung eines **Anforderungsdokuments (AD)** von Siemens aufgelistet. Dabei wird festgehalten, welche Informationen in dem Anforderungsdokument enthalten sein müssen und wie dieses aufgebaut ist. Der Inhalt des Anforderungsdokuments wird anhand eines Beispielprojekts festgelegt. Die Anforderungen AD-F1 bis AD-F9 sowie die nicht funktionalen Anforderungen sind für jedes Projekt gleich.

AD-F1 [muss]: Die Anforderungen werden aus einem Kundenanforderungsdokument generiert.
Das Kundenanforderungsdokument ist in Form eines PDF Dokuments.

AD-F2 [muss]: Das Dokument hat eine Übersicht, welche die Projektnummer, die Dokumentenbezeichnung, das Erstellungsdatum, die Dokumentennummer und die Version beinhaltet.

AD-F3 [muss]: In der Übersicht gibt es eine Tabelle für das Versionsmanagement, mit den Spalten: Version, Änderungsdatum, Autor, Änderungen und Bemerkungen

AD-F4 [muss]: Die Anforderungen werden eindeutig nummeriert. Die Nummern werden in einer Spalte, mit der Überschrift „No.“ eingetragen.
Die Nummerierung hat die Form: Anf1, Anf2, Anf3, etc.

AD-F5 [muss]: Für das Kapitel der Anforderungen gibt es eine Spalte „Chapter“. Dort werden die jeweiligen Kapitel des Kundenanforderungsdokuments eingetragen.

AD-F6 [muss]: Die Anforderungen haben eine kurze und messbare Beschreibung, welche in der Spalte „Description“ stehen.

AD-F7 [muss]: Es gibt eine leere Spalte für die Kategorisierung, mit der Überschrift „Category“ und eine Spalte für Kommentare, mit der Überschrift „Notes“.

AD-F8 [kann]: Die einzelnen Anforderungen werden auf ihre Umsetzbarkeit überprüft und farblich markiert.
Dies dient dazu, dass beim späteren Korrigieren des Dokuments durch einen Mitarbeitenden die Arbeit vereinfacht wird.

AD-F9 [muss]: Der Dateiname wird aus der Projektnummer, dem DCC, der Dokumentennummer, dem Status, der Version und der Dokumentenbezeichnung zusammengesetzt. Der Name für das Anforderungsdokument lautet: Projektnummer + „-EC411-W01-RequirementList“.
Es gibt für die Dokumentennamen eine Vorschrift der Abteilung, von der ein Auszug im Anhang (Abschnitt A.1) zu finden ist.

AD-F10 [muss]: Es soll kein neuer Inhalt generiert werden. Die Anforderungen werden anhand der Informationen des Anforderungsdokument vom Kunden erstellt.

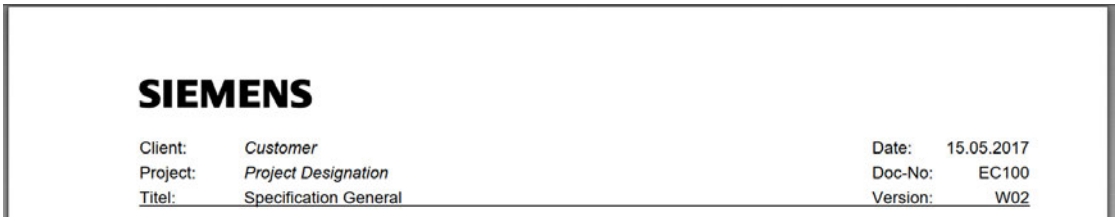

Abbildung 3.3: Aufbau der Kopfzeile der technischen Spezifikation

Abbildung 3.4: Aufbau der Fußzeile der technischen Spezifikation

AD-NF1 [kann]: Das Dokument ist in Form einer Excel-Datei.

3.4.3 Technische Spezifikationen

Der Abschnitt **Technische Spezifikationen (TS)** listet die Anforderungen an die Dokumente für die technischen Spezifikationen. Dabei wird sowohl auf das Dateiformat, das Design des Dokuments, als auch den Inhalt anhand von Beispieldokumenten eingegangen.

TS-F1 [muss]: Die technische Spezifikation wird aus einem Anforderungsdokument generiert.
Das Anforderungsdokument ist eine .xlsx-Datei

TS-F2 [muss]: Das Dokument hat eine Kopfzeile mit den folgenden Informationen: Kunde, Projekt, Dokumentenbezeichnung/ Titel, Erstellungsdatum, Dokumentennummer, Version. Zudem ist der Schriftzug „SIEMENS“ abgebildet.
Die Aufbau der Kopfzeile kann aus Abbildung 3.3 entnommen werden.

TS-F3 [muss]: Das Dokument hat eine Fußzeile mit den folgenden Informationen: Seitenzahl, Dateiname mit Dateiformat, Copyright-Hinweis
Die Aufbau der Fußzeile kann aus Abbildung 3.4 entnommen werden.

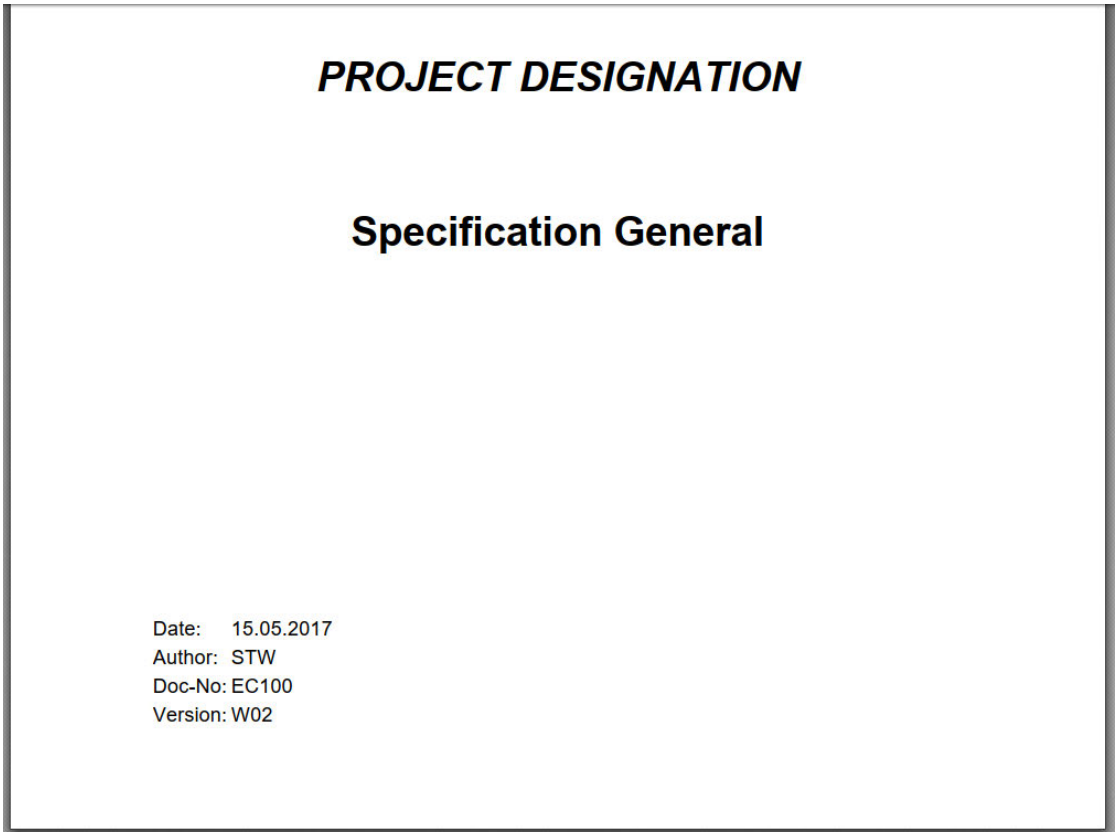

Abbildung 3.5: Aufbau der Titelseite der technischen Spezifikation

TS-F4 [muss]: Die Titelseite enthält im linken unteren Bereich die Informationen: Erstellungsdatum, Autor, Dokumentennummer und Version.
Das Design der Titelseite kann aus Abbildung 3.5 entnommen werden.

TS-F5 [muss]: Die Titelseite enthält keine Fußzeile.

TS-F6 [muss]: Die Titelseite enthält als Titel die Projektbezeichnung und sowie die Dokumentenbezeichnung

TS-F7 [muss]: Es gibt ein automatisches Inhaltsverzeichnis, welches die einzelnen Abschnitten des Dokuments beinhaltet.

TS-F8 [kann]: Es gibt ein Tabellenverzeichnis, in dem alle, in der technischen Spezifikation, vorkommenden Tabellen aufgelistet sind.

Version Management

Rev.	Date	Written by:	Adapted sections	Remarks
W01	18.04.2017	STW		First version
W02	22.05.2017	STW	2,	

Abbildung 3.6: Tabelle für das Versionsmanagement der technischen Spezifikation

Document Status

Action	Name, Signature and Date
Written / Changed by	
Checked by	
Released by	

Abbildung 3.7: Tabelle für den Dokumentenstatus der technischen Spezifikation

TS-F9 [kann]: Es gibt ein Abbildungsverzeichnis, in dem alle, in der technischen Spezifikation, vorkommenden Abbildungen aufgelistet sind.

TS-F10 [muss]: Für das Versionsmanagement gibt es eine Tabelle mit den Spalten: Version, Änderungsdatum, Autor, Änderungen und Bemerkungen.
Eine Vorlage für das Aussehen des Versionsmanagement ist in Abbildung 3.6 dargestellt.

TS-F11 [muss]: Für den Dokumentenstatus gibt es eine Tabelle mit den Spalten: Aktion und Name, Unterschrift und Datum. In der Spalte „Aktion“ gibt es drei Felder mit dem Inhalt Autor, Kontrolle und Veröffentlichung.
In Abbildung 3.7 ist eine Designvorlage für den Dokumentenstatus abgebildet.

TS-F12 [muss]: Der Dateiname wird aus der Projektnummer, dem DCC, der Dokumentennummer, dem Status, der Version und der Dokumentenbezeichnung zusammengesetzt. Der Name für das Anforderungsdokument lautet: Projektnummer + „-EC412-W01-TechnicalSpecification“.
Es gibt für die Dokumentennamen eine Vorschrift der Abteilung. Ein Ausschnitt davon ist im Anhang zu finden.

TS-NF1 [kann]: Das Dokument ist in Form einer Word-Datei

3.4.4 Testfälle

Im Folgenden werden die Anforderungen zum Erstellen von **Testfällen (TF)** beschrieben. Dabei werden zunächst die Anforderungen an den Aufbau eines Testfalls aufgeführt und anschließend, welche Informationen die Testfälle beinhalten müssen.

TS-F1 [muss]: Die Testfälle werden aus einer technischen Sepzifikation generiert.
Die technische Sepzifikation ist eine .docx-Datei.

TF-F2 [muss]: Das Dokument hat eine Übersicht, in der das Projekt, die Testart/ der Testname, sowie die Dokumentenbezeichnung beinhaltet.

TF-F3 [muss]: Die Testfälle müssen eine eindeutige ID („ID“) haben.

TF-F4 [muss]: Die Testfälle haben eine Spalte „Section“ für die Gruppierung des Testfalls.

TF-F5 [muss]: Die Testfälle werden in der Spalte „Description“ eindeutig beschrieben.

TF-F6 [muss]: Für das Testdatum wird die Spalte „Test date“ angelegt.

TF-F7 [muss]: Es gibt eine Spalte „Tester“ für den Namen der testenden Person.

TF-F8 [muss]: Es wird das zu erwartende Ergebnis („Expected result“) definiert.

TF-F9 [muss]: Zum Eintragen des Testergebnisses gibt es ein Feld „Test result“.

TF-F10 [muss]: Es können Kommentare („Notes“) für einen S7-Programmtest und den CMS Test eingetragen werden.

TF-F11 [muss]: Es werden sowohl Negativ- als auch Positivtests erstellt.

TF-F12 [muss]: Der Dateiname wird aus der Projektnummer, dem DCC, der Dokumentennummer, dem Status, der Version und der Dokumentenbezeichnung zusammengesetzt. Der Name für das Anforderungsdokument lautet: Projektnummer + „-WT1901-W01-Textcases“.
Es gibt für die Dokumentennamen eine Vorschrift der Abteilung. Ein Ausschnitt davon ist im Anhang zu finden.

TF-NF1 [kann]: Die Dokumente sind in Form einer Excel-Datei

3.5 Hinweis

Aufgrund der Vollständigkeit, sowie eines besseren Überblicks über den Umfang der gesamten Anwendung, sind in Abschnitt 3.4 alle Anforderungen an die Anwendung aufgelistet. Da der Umfang und die Dauer einer Bachelorarbeit beschränkt sind, können nicht alle Anforderungen erfüllt werden. Der Schwerpunkt liegt aus diesem Grund vor allem auf die Anforderungen aus Unterabschnitt 3.4.1 und 3.4.2. Dies sind die Kernpunkte der Anwendung, auf die später bei einer Weiterentwicklung aufgebaut werden können. Für die spätere Weiterentwicklung wird kurz auf die Anforderungen aus Unterabschnitt 3.4.3 und 3.4.4 im Konzept (Abschnitt 4.5) eingegangen.
Zudem können die Anforderungen zum Anforderungsmanagement Tool Polarion nur bedingt erfüllt werden, da lediglich ein Testzugang für die Anwendung vorliegt. Dies wird in Abschnitt 5.4 genauer erläutert.

4 Konzept und Design

In diesem Kapitel werden das Konzept und das Design der Anwendung detailliert beschrieben. Zunächst erfolgt ein Vergleich verschiedener Sprachmodelle zur Generierung der Projektmanagementdokumente. Im Anschluss daran wird eine geeignete Programmiersprache für die Implementierung der Anwendung ausgewählt. Danach wird die Kommunikation mit dem LLM erläutert und wie diese strukturiert sein soll. Dabei wird auf das Senden einer Nachricht von chatbasierten und visuellen LLMs, sowie das Empfangen der Antwort eingegangen. Weiterhin wird das Einlesen und Verarbeiten von PDFs erläutert, um den Kontext zur Erstellung von Anforderungsdokumenten generieren zu können. Es werden verschiedene Methoden dargestellt und ihre Vorteile sowie Nachteile erläutert. Anschließend wird auf die Erstellung der einzelnen Dokumente eingegangen. Zum einen wird dabei der erforderliche Kontext für das LLM betrachtet und zum anderen, wie die Informationen verarbeitet und das Dokument geschrieben wird. Darauf folgt die Kommunikation mit dem Anforderungsmanagement Tool „Polarion“. Von dort werden die Dokumente zur Generierung der Projektmanagementdokumente entnommen und die generierten Dokumente wieder abgelegt. Zum Schluss wird auf die grafische Oberfläche der Anwendung Bezug genommen.

4.1 Eingesetzte Sprachmodelle

Die interne Siemensplattform *code.siemens* bietet Zugriff auf verschiedene LLMs. Andere Plattformen, wie ChatGPT, speichern Informationen häufig zwischen oder verwenden sie zum Trainieren von Modellen. Da die Kommunikation mit dem LLM auch für andere Projekte genutzt werden kann, müssen die übergebenen Informationen an das LLM vertraulich behandelt werden (vgl. Anforderung DG-NF5). Daher wird die unternehmensinterne interne Plattform *code.siemens* verwendet (vgl. Anforderung DG-NF2, Anforderung DG-NF4).
Es gibt mehrere Modelle, welche für die Anwendung genutzt werden können. Dabei sind

vor allem die drei Modelle, **Mistral 7b Instruct**, **Deepseek R1 Distill Qwen 7b** und **Qwen3 30B A3B** zu betrachten. Diese können frei verwendet werden und sind für den Produktionseinsatz vorgesehen.
Neben den drei genannten Modellen gibt es noch weitere Modelle, welche genutzt werden können. Diese befinden sich jedoch im Alpha-Status und sind für die Entwicklung und Forschung vorgesehen. Daher werden diese Modelle auch häufiger gegen neuere Modelle ersetzt und sind damit nicht Vorwärtskompatibel. Es handelt sich dabei um die Modelle **Llama 3.1 8B Instruct**, **Mistral Nemo Instruct 2407** und **Qwen2.5 Coder 7B Instruct**. Es werden nur chatfähige Modelle einbezogen, da dies notwendig für das Erstellen der Dokumente ist. Zum Extrahieren des Textes einer PDF-Datei wird jedoch das **Pixtral 12B 2409** betrachtet, welches ein visuelles Modell darstellt und für die Bildanalyse ausgelegt ist. [15]

4.1.1 Mistral 7b Instruct

Das Modell Mistral 7B ist ein Modell aus September 2023 mit 7,3 Milliarden Parametern. Es benutzt die GQA für schnellere Schlussfolgerungen und die SWA, um besser mit längeren Sequenzen umzugehen. Die Fenstergröße beträgt 4k und die Sequenzlänge 16k Token. Mistra 7B Instruct ist eine Abwandlung von Mistral 7B und für eine Chatinteraktion ausgelegt. Im MT-Bench schneidet das Modell besser ab, als andere 7B Modelle, wie Llama-2-7b-chat. Zudem hat Mistral 7B Instruct eine bessere Performance bei dem Benchmark, als manche 13B Modelle, wie Llama-2-13b-chat oder WizardLM-13B-v1.1. Insgesamt erreicht das Modell Mirstral 7B Instruct bei dem MT Bench einen Wert von $6,84 \pm 0,065$. Im MMLU Benchmark wird eine Genauigkeit von 60,1 % und im HellaSwag Benchmark eine Genauigkeit von 81,3 %.[1,2]

4.1.2 Deepseek R1 Distill Qwen 7b

DeepSeek R1 ist ein Modell von der Firma DeepSeek aus dem Jahr 2025. Es besitzt 7,62 Milliarden Parameter und die Fähigkeit zur Selbstüberprüfung, Reflexion und Denkketten (Chain of Thought) zu bilden. Die maximale Kontextlänge beträgt 32.768 Token.

[1] https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2, Zugriffsdatum: 06.03.2025
[2] https://mistral.ai/news/announcing-mistral-7b, Zugriffsdatum: 06.03.2025

DeepSeek ist für die Sprachen Englisch und Chinesisch optimiert, wodurch es vorkommen kann, dass die Antwort und Schlussfolgerung auf Englisch ist, obwohl die Anfrage in einer anderen Sprache gestellt wurde. Neben der Antwort auf die Anfrage, liefert DeepSeek R1 einen Gedankengang (Chain of Thought). Dort wird die Vorgehensweise zur Absolvierung der Anfrage beschrieben. Gekennzeichnet wird der Chain of Thought durch den XML-Tag <think> . Das 32B Modell erreicht im MMLU Benchmark eine Genauigkeit von 87,4 %. Es ist anzunehmen, dass das 7B Modell schlechter im MMLU Benchmark abschneidet.[3] [5]

4.1.3 Qwen2.5 Coder 7B Instruct

Das neueste Modell der CodeQwen Code-spezifischen LLMs ist der Qwen2.5-Coder. Das Modell Qwen2.5 Coder 7B Instruct hat 7,62 Milliarden Parameter und 28 Schichten. Die Kontextlänge beträgt 131.072 Token. Es ist speziell für das Schreiben und Analysieren von Code ausgelegt und daher vermutlich für den Zweck dieser Arbeit nicht optimal geeignet. Wie in Unterabschnitt 2.4.1 bereits beschrieben, ist es notwendig, Text zu generieren, zusammenzufassen, zu analysieren und ggf. zu übersetzen. Code wird in keiner Form verarbeitet. Dennoch wird das Modell, wie die anderen genannten Modelle, für die Anwendung getestet. Es verwendet die GQA mit 28 Köpfen für schnellere Antworten.[4,5] Im MMLU Benchmark erreicht das Modell Qwen2.5 Coder 7B Instruct eine Genauigkeit von 67,6 %.[6]

4.1.4 Llama 3.1 8B Instruct

Auch die Firma Meta hat mit Llama ein LLM auf den Markt gebracht. Das Modell Llama 3.1 8B Instruct hat 8,03 Milliarden Parameter und ist im Juli 2024 erschienen. Es ist ausschließlich für Texteingaben entwickelt und für den mehrsprachigen Dialog optimiert. Die Kontextlänge beträgt 128k Token. Das Modell verwendet GQA für eine

[3] https://huggingface.co/deepseek-ai/DeepSeek-R1-Distill-Qwen-7B, Zugriffsdatum: 10.04.2025
[4] https://huggingface.co/Qwen/Qwen2.5-Coder-7B, Zugriffsdatum: 27.05.2025
[5] https://qwenlm.github.io/blog/qwen2.5-coder-family/, Zugriffsdatum: 28.05.2025
[6] https://llm-stats.com/models/compare/qwen-2.5-14b-instruct-vs-qwen-2.5-coder-7b-instruct, Zugriffsdatum: 03.06.2025

verbesserte Skalierbarkeit der Inferenz. Es erreicht eine Genauigkeit von 68,5 % im MMLU Benchmark.[7] [8]

4.1.5 Mistral Nemo Instruct 2407

Mistral Nemo Instruct 2407 ist ebenfalls im Juli 2024 erschienen, besitzt 12,2 Milliarden Parameter und wurde in Zusammenarbeit mit NVIDIA entwickelt. Die Kontextlänge beträgt 128k Token und das Modell hat 40 Schichten. Als Aufmerksamkeitsmechanismus wird eine GQA verwendet. Es erreicht im HellaSwag Benchmark eine Genauigkeit von 83.5% und 68.0 % im MMLU Benchmark.[9,10]

4.1.6 Qwen3 30B A3B

Das Modell Qwen3 30B A3B ist ein Modell aus der neuesten Qwen Reihe. Es besitzt 30,5 Milliarden Parameter und 48 Schichten. Die Kontextlänge beträgt 128K Token und verwendet die GQA. Im MMLU Benchmark wird eine Genauigkeit von 81.38 %. Zudem liefert das Modell, wie DeepSeek R1, bei der Antwort den Chain of Thought mit. Dieser wird ebenfalls durch „<think></think>“ markiert. Für dieses Modell wurden keine Genauen Ergebnisse für den MMLU, HellaSwag oder MT-Bench Benchmark gefunden. Jedoch performen die Modelle der Qwen3 Reihe genauso gut, wie die der Qwen2.5 Reihe mit mehr Parametern. Qwen3 30B lässt sich demnach Qwen2.5 72B verleichen, welches im MMLU Benchmark eine Genauigkeit von 86,1 % erreicht hat.[11,12] [19]

4.1.7 Pixtral 12B 2409

Das Pixtral Modell ist im Gegensatz zu den anderen Modellen nicht zur generellen Textverarbeitung gedacht, sondern für das Verarbeiten von Bildern. Es wurde im September 2024 veröffentlicht und hat 12,4 Milliarden Parameter. Die Kontextlänge beträgt 128k Token. Pixtral 12B unterstützt variable Bildgrößen und Seitenverhältnisse, sodass diese

[7] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct, Zugriffsdatum: 10.04.2025
[8] https://ai.meta.com/blog/meta-llama-3-1/, Zugriffsdatum: 04.04.2025
[9] https://mistral.ai/news/mistral-nemo, Zugriffsdatum: 04.04.2025
[10] https://huggingface.co/mistralai/Mistral-Nemo-Instruct-2407, Zugriffsdatum: 04.04.2025
[11] https://huggingface.co/Qwen/Qwen3-30B-A3B, Zugriffsdatum: 27.05.2025
[12] https://qwenlm.github.io/blog/qwen2.5-max/, Zugriffsdatum: 03.06.2025

nicht zuvor normiert werden müssen. Beim MMLU Benchmark wird eine Genauigkeit von 69,2 % erreicht und beim MT-Bench einen Wert von 7,68. Zudem wurde ein multimodaler MT-Bench Benchmark entwickelt, welcher das Modell auf die Verarbeitung von mehreren verschiedenen Datentypen (Bild und Text gleichzeitig) bewertet. Dort erreicht Pixtral 12B einen Wert von 6,05.[13,14]

4.2 Programmiersprache

Für die Entwicklung der Anwendung muss zunächst eine Programmiersprache ausgewählt werden. Hierzu werden verschiedene Programmiersprachen im Hinblick auf die folgenden Aspekte betrachtet, welche vom Verfasser als relevant erachtet werden:

1. **Entwicklung einer grafischen Benutzeroberfläche**
   (Anforderung DG-F7)
   Für die Entwicklung einer grafischen Benutzeroberfläche können z. B. die Programmiersprachen JavaScript[15], Python[16], Java[17] und C#[18] genutzt werden. Es gibt noch viele weitere Programmiersprachen, mit denen die Entwicklung einer grafischen Benutzeroberfläche möglich ist.

2. **Schreiben von Microsoft Excel und Word-Dateien**
   (Anforderung AD-NF1, TS-NF1, TF-NF1)
   Das Schreiben von Microsoft Excel und Word-Dateien ist mit verschiedenen Programmiersprachen möglich. Beispiel sind Python[19,20], VBA[21], C#[22], JavaScript[23] und Java[24]. VBA wird dabei vor allem für die Automatisierung von Office-Dateien genutzt, beispielsweise in Form von Makros.

[13] https://mistral.ai/news/pixtral-12b, Zugriffsdatum: 27.05.2025
[14] https://huggingface.co/mistralai/Pixtral-12B-2409, Zugriffsdatum: 27.05.2025
[15] https://docs.nodegui.org/, Zugriffsdatum: 12.05.2025
[16] https://wiki.python.org/moin/GuiProgramming, Zugriffsdatum: 09.04.2025
[17] https://spring.io/, Zugriffsdatum: 12.05.2025
[18] https://dotnet.microsoft.com/en-us/apps/desktop, Zugriffsdatum: 09.04.2025
[19] https://www.python-excel.org/, Zugriffsdatum: 09.05.2025
[20] https://python-docx.readthedocs.io/en/latest/, Zugriffsdatum: 09.05.2025
[21] https://learn.microsoft.com/en-us/office/vba/library-reference/concepts/getting-started-with-vba-in-office, Zugriffsdatum: 09.05.2025
[22] https://learn.microsoft.com/de-de/dotnet/csharp/advanced-topics/interop/how-to-access-office-interop-objects, Zugriffsdatum: 09.05.2025
[23] https://learn.microsoft.com/en-us/office/dev/add-ins/overview/office-add-ins, Zugriffsdatum: 09.05.2025
[24] https://poi.apache.org/, Zugriffsdatum: 09.05.2025

3. **Kommunikation mit dem LLM**
   (Anforderung DG-NF4, DG-NF5)
   Die Kommunikation mit dem LLM kann über ein API mit **H**yper**t**ext **T**ransfer **P**rotocol (HTTP)[25] oder eine OpenAI Bibliothek realisiert werden.[26] Dafür eignen sich JavaScript, Python, C#, Java und Go.[27] Eine HTTP-Verbindung kann mit vielen Programmiersprachen realisiert werden.

4. **Kommunikation mit Polarion**
   (Anforderung DG-F2, DG-F4, DG-F6, DG- F13, DG-F14, DG-F15)
   Mit einer REST API kann die Kommunikation mit Polarion implementiert werden. Besonders eignen sich dafür die Programmiersprachen Java, JavaScript und C#, da die Schnittstelle für diese Programmiersprachen dokumentiert ist.[28]

5. **Verwendete Programmiersprachen in der Abteilung**
   (Unterabschnitt 3.2.4)
   Da das Kerngeschäft die Elektrifizierung und Automatisierung von Krananlagen ist und dabei vorwiegend die SPS-Programmierung notwendig ist, überwiegen die Sprachen FUP und SCL. Des Weiteren werden noch C# und VBA verwendet, für Makros oder Anwendungen verwendet.

Anhand der aufgeführten Kriterien zur Programmiersprache ist zu erkennen, dass sich mehrere Programmiersprachen, wie Java, JavaScript, Python oder C#, zur Erstellung der Anwendung eignen. Da in der Abteilung bereits C# verwendet wird und in Zukunft weitere Projekte, basierend auf der Anwendung, implementiert werden sollen, wird sich für C# zur Entwicklung der Anwendung entschieden.

4.3 Kommunikation mit dem LLM

Für die Kommunikation mit dem LLM, bzw. den Modellen gibt es verschiedene Möglichkeiten. Generell wird Kommunikation über HTTP realisiert. Dabei werden im Request die URL, der API-Key und die Nachricht übergeben. Der Aufbau der Nachricht wird

[25] https://developer.internal.siemens.com/code-siemens-com/llm/overview.html, Zugriffsdatum: 09.05.2025
[26] https://code.siemens.io/ai/, Zugriffsdatum: 09.05.2025
[27] https://platform.openai.com/docs/libraries, Zugriffsdatum: 09.05.2025
[28] https://testdrive.polarion.com/polarion/sdk/doc/rest/index.html, Zugriffsdatum: 06.03.2025

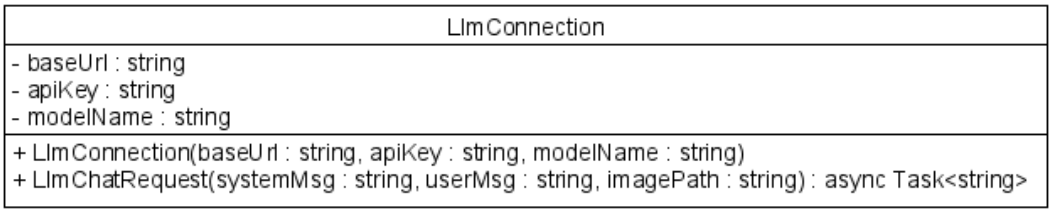

Abbildung 4.1: Klassendiagramm der Klasse **LlmConnection**

in Unterabschnitt 4.3.1 näher beschrieben. Es gibt die Option, die Kommunikation über HTTP selbst zu implementieren oder ein vorhandenes Paket zur Kommunikation zu nutzen, welches auf der OpenAI **A**pplication **P**rogramming **I**nterface (API) basiert. In diesem Fall fällt die Entscheidung auf das Paket *OpenAI-DotNet*. Das *OpenAI*-Paket wird nicht genutzt, da es dort nicht möglich ist, die URL für die Verbindung zu ändern.[29,30] Für die Kommunikation wird eine Klasse **LlmConnection** implementiert (siehe Abbildung 4.1). Diese beinhaltet die Attribute für die Hauptadresse (baseUrl), den API Schlüssel (apiKey) und den Modellnamen (modelName). Zudem besitzt die Klasse noch die Methode **LlmChatRequest()** zur Kommunikation. Dabei soll es möglich sein, Chatnachrichten, sowie Bilder zu senden.

4.3.1 Aufbau der Nachrichten

Die Nachrichten sind im JSON-Format (**J**ava**S**cript **O**bject **N**otation). In Listing 4.1 sind die notwendigen Parameter für die Kommunikation aufgelistet. Dabei wird zunächst das Modell („**model**“) festgelegt, welches genutzt wird. Die verwendeten Modelle dieser Arbeit sind bereits in Unterabschnitt 4.1.1 bis Unterabschnitt 4.1.6 aufgeführt. Nach der Auswahl des Modells werden die Nachrichten („**messages**“) geschrieben. Dabei gibt es drei Kategorien der Nachrichten, die durch „**role**“ festgelegt werden: „**system**“, „**user**“ und „**assistant**“. Im „**content**“ steht der jeweilige Prompt für den Nachrichtentyp. Über „**temperature**“ kann die Kreativität und Zufälligkeit der Antwort eingestellt werden. Eine niedrige Temperatur sorgt für eine kohärente und konsistente Antwort, während eine hohe Temperatur für eine kreative Antwort sorgt. Durch „**stream**“ wird die Antwort

[29] https://www.nuget.org/packages/OpenAI/2.1.0#show-readme-container, Zugriffsdatum: 06.05.2025
[30] https://platform.openai.com/docs/api-reference/introduction, Zugriffsdatum: 30.05.2025

in Echtzeit während der Generierung in Chunks übermittelt und die Antwort baut sich nach und nach auf.

```
{
    "model": "mistral-7b-instruct",
    "messages": [
        {
            "role": "system",
            "content": "You are a helpful AI assistant."
        },
        {
            "role": "user",
            "content": "Hello, how are you?"
        }
    ],
    "temperature": 0,
    "stream": true
}
```

Listing 4.1: Aufbau der Nachricht für Chat LLMs im JSON-Format

Die Struktur der Nachricht für visuelle Modelle, wie etwa Pixtral 12B, unterscheidet sich geringfügig (vgl. Listing 4.2), da es sich, wie in Unterabschnitt 4.1.7 erläutert, um ein Bildverarbeitungsmodell handelt. Dabei wird der Inhalt „**content**“ in verschiedene Kategorien aufgeteilt. Zunächst wird die Kategorie „**type**“ selbst festgelegt, in der beschrieben wird, ob es sich um eine Textanweisung oder um ein Bild handelt. Bei einer Textanweisung, wird in „**text**“ die Anweisung beschrieben, welche Aktion das Modell, bezogen auf das Bild, durchgeführt wird. Zum Bereitstellen des Bilds wird über „**image_url**“ und „**url**“ die URL des zu verarbeitenden Bildes übergeben. Anstelle einer URL ist es ebenfalls möglich, das Bild als einen Base64 String zu übergeben. Die Parameter **„model“**, **„messages“**, **„role“**, **„temperature“** und **„stream“** sind die gleichen wie in Listing 4.1.

Bei der Antwort des LLMs wird lediglich der Bereich „**choices**“ genutzt, da dort die Nachricht des LLM enthalten ist. Dies ist in Listing 4.3 abgebildet. Im Bereich „**choices**“ befinden sich: „**finish_reason**“ „**index**“ und „**message**“. „**Message**“ beinhaltet „**content**“, „**role**“, „**tool_calls**“ und „**function_call**“, wobei „**content**“ die Nachricht des LLMs enthält.

```
{
    "model": "pixtral-12b-2409",
    messages = [
    {
        "role": "user",
        "content": [{"type": "text", "text": prompt}, {"type":
           "image_url", "image_url": {"url": image_url}}]
    },
],
    "temperature": 0,
    "stream": true
}
```

Listing 4.2: Aufbau der Nachricht für visuelle LLMs im JSON-Format

4.3.2 Prompting

Die Prompts werden spezifisch für die unterschiedlichen Dokumente erstellt. Jeder Fall beinhaltet dabei einen System Prompt sowie einen User Prompt. Für die Pompts wird die Few-shot Methode angewandt und die Antwort als Structured Output vorgegeben. Der Systemprompt wird in einer Textdatei im Markdown und XML-Format geschrieben. Dabei gibt es die folgenden drei Kategorien:

- Identity
- Instructions
- Examples

Die Abschnitte werden als Markdown dargestellt und geschrieben. Nur der Abschnitt „Examples“ ist im XML-Format, um zu zeigen wie auf eine Nachricht reagiert werden soll. Dazu gibt es ein Element für die Benutzeranfrage („<user_query>“) und eins für die Antwort des LLMs („<assistant_response>“). Der Aufbau ist beispielhaft in Listing 4.4 dargestellt.

4.4 Einlesen und Verarbeiten einer PDF-Datei

Um die Informationen aus den Anforderungsdokumenten des Kunden, welche in Form einer PDF-Datei vorliegen, zu extrahieren, muss es möglich sein, den Inhalt dieser PDF-

```
"choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": " Hello! I'm just an AI, so I don't have
           feelings, but I'm here and ready to assist you. How
           can I help you today?",
        "role": "assistant",
        "tool_calls": null,
        "function_call": null
      }
    }
  ]
```

Listing 4.3: Aufbau der Antwort des LLM im JSON-Format

Datei einzulesen (Anforderung AD-F1). Zudem muss der Inhalt sinnvoll verarbeitet werden, um dem LLM eine vernünftige und eindeutige Nachricht übermitteln zu können. Ein großes Problem stellt dabei das Einlesen von Grafiken und Tabellen, die in einer PDF-Datei vorkommen, dar. Dieses Problem wird später genauer erläutert.

4.4.1 PdfSharp-Bibliothek

Eine Möglichkeit, zum Einlesen von PDF-Dateien, bietet die *PdfSharp*-Bibliothek in C#. Diese ist eine frei verwendbare Open-Source-Bibliothek[31] zum Lesen, Erstellen und Bearbeiten von PDF-Dokumenten. Das Extrahieren von Text aus diesen Dokumenten ist dabei zwar möglich, es gibt jedoch keine explizite Funktion dafür. Damit gestaltet sich das Extrahieren von Text mit der *PdfSharp*-Bibliothek aufwendig und diese Funktion soll nur einen kleinen Teil der Arbeit darstellen, weshalb dazu die *PdfSharp*-Bibliothek nicht weiter verwendet wird.
Das Extrahieren von Bildern aus einem PDF Dokument ist dahingegen intuitiv und ohne großen Aufwand verbunden. Dennoch muss vorher selbstständig überprüft werden, ob es sich bei einem Objekt um ein Bild handelt.

[31] https://www.pdfsharp.net/Licensing.ashx, Zugriffsdatum: 11.03.2025

```
# Identity

    ...

# Instructions

    *...
    *...

# Examples

    <user_query>
        ...
    </user_query>

    <assistant_response>
        ...
    </assistant_response>
```

Listing 4.4: Aufbau eines system Prompts

4.4.2 PdfPig-Bibliothek

Eine bessere Möglichkeit bietet die *PdfPig*-Bibliothek in C#. Dies ist eine unter der Apache-Lizenz 2.0[32] verwendbare Bibliothek zum Lesen, Erstellen und Bearbeiten von PDF-Dokumenten. Mit *PdfPig* können die einzelnen Buchstaben, Zeichen, Wörter und Bilder einer PDF extrahiert werden. Mit den Methoden **GetWords()** und **GetImage()** der Klasse **Page** lassen sich der Text und die Bilder einer PDF extrahieren.
Eine besondere Herausforderung ist das Auslesen von Tabellen. Diese werden nicht erkannt und der Text wird, wie der restliche Blocktext, in dem verwendeten Format dargestellt. Das erschwert das Lesen und richtige Interpretieren des Textes, nachdem dieser extrahiert wurde. Hinzu kommt, dass leere Felder einer Tabelle gar nicht dargestellt werden. Um die Tabellen sinnvoll darzustellen, muss ein Algorithmus implementiert werden, der Tabellen erkennt und vernünftig in einen **String** formatiert.

Tabellen extrahieren: Um Tabellen aus einer PDF-Datei zu extrahieren, werden zunächst alle Linien einer Seite herausgefiltert, wobei Buchstaben und Zeichen nicht dazu

[32] https://licenses.nuget.org/Apache-2.0, Zugriffsdatum: 11.03.2025

zählen. Es handelt sich dabei um Linien, welche in Form eines Rechtecks auftauchen, wenn diese auch als eine Geometrieform deklariert sind. Dies kann mit der Methode **GetBoundingRectangle()** realisiert werden. Danach können diese in horizontale und vertikale Linien unterteilt werden.
Nachdem der Text, sowie die horizontalen und vertikalen Linien extrahiert wurden, kann eine Tabelle erstellt werden. Dazu ist ein Programmablaufplan in Abbildung 4.2 abgebildet. Zunächst wird überprüft, ob alle Zeilen der Tabelle durchlaufen wurden. Ist dies nicht der Fall, wird die Position der ersten Zeile der Tabelle ermittelt. Danach wird eine horizontale Linie zum String hinzugefügt. Um nun den Inhalt der einzelnen Zellen der Zeile zu extrahieren, wird überprüft, ob alle Spalten durchlaufen wurden. Wurden nicht alle Spalten durchlaufen, wird eine vertikale Linie und der Text der Zelle zum String hinzugefügt. Nachdem alle Spalten durchlaufen wurden, wird noch einmal eine vertikale Linie zum String hinzugefügt und die Schleife beginnt von vorn. Sobald jede Zeile der Tabelle durchlaufen wurde, wird noch eine horizontale Linie zum String hinzugefügt und die Methode beendet.

4.4.3 Bildererkennung

Statt des direkten Lesens einer PDF-Datei, kann diese auch zuerst in ein Bild konvertiert werden, um anschließend das Bild zu lesen. Dazu kann das LLM Pixtral 12B 2409 verwendet werden, welches für visuelle Aufgaben ausgelegt ist.

PDF zu PNG: Um die PDF-Datei als Bild zu verarbeiten, muss diese wie oben genannt erst in ein Bild konvertiert werden. Dazu wird das Paket *Freeware.Pdf2Png* verwendet. Dies konvertiert, wie der Name impliziert, PDFs in einzelne Bilder (ein Bild pro Seite). Die Bilder werden als **P**ortable **N**etwork **G**raphics (PNG) zurückgegeben.

Extrahieren des Textes mit Pixtral 12B: Nachdem die PDF-Datei mit *Freeware.Pdf2Png* in ein Bild konvertiert wurde, ist es möglich, dieses mit Pixtral 12B 2409 zu analysieren. Die Kommunikation mit dem Modell wird bereits in Abschnitt 4.3 beschrieben. Um dem Modell ein Bild zu senden, muss dieses zuvor in einen Base64 String konvertiert werden. Mit Base64 lassen sich 8-Bit-Binärdateien, wie Programme, .zip-Datein und Bilder in eine Zeichenfolge aus ASCII-Zeichen konvertieren.[33] Dazu kann mit

[33] https://www.base64decode.org/de/, Zugriffsdatum: 17.04.2025

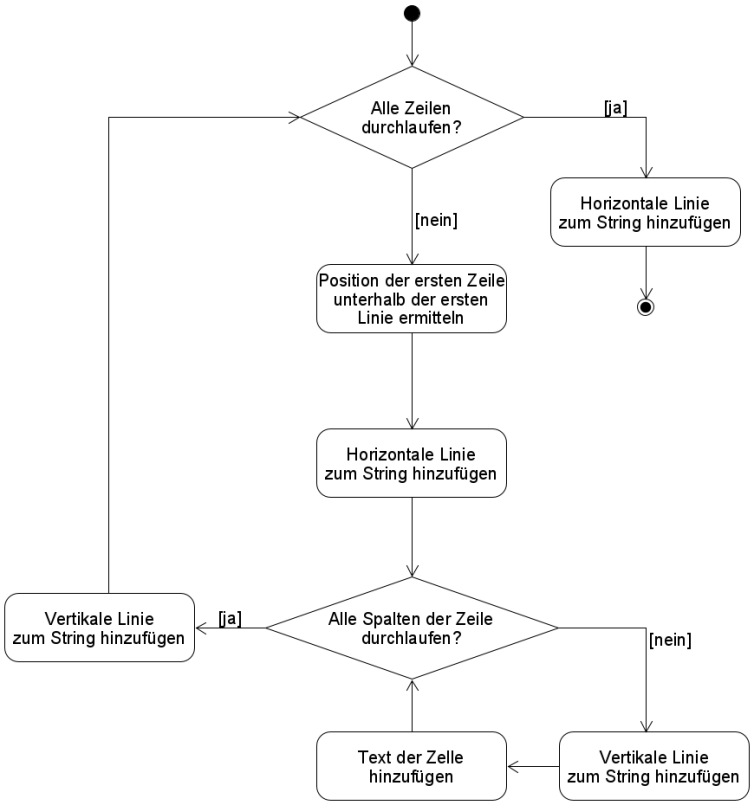

Abbildung 4.2: Ablaufplan zum Erstellen einer Tabelle

der Methode **ReadAllBytes()** der Klasse **File** eine Datei in einen Array von Bytes geschrieben werden. Dieser Array wird anschließend mit der Methode **ToBase64String()** der Klasse **Convert** in einen Base64 String konvertiert. Beide Klassen (**File** und **Convert**) befinden sich im *System.IO* Namespace.
Der Base64 String kann nun an das Pixtral-Modell, mit einer Aufgabe oder Anweisung, gesendet werden. Das Modell soll den Text des Bildes extrahieren und als Markdown wiedergeben. Dabei soll das Design unverändert bleiben.

4.5 Erstellen der Dokumente

Die einzelnen Dokumente werden durch das LLM generiert (Anforderung DG-NF4). Dazu muss ein geeigneter Kontext für das Modell bereitgestellt werden. Für alle Dokumente soll der Kontext als Markdown übergeben werden und auch das Modell soll das erstellte

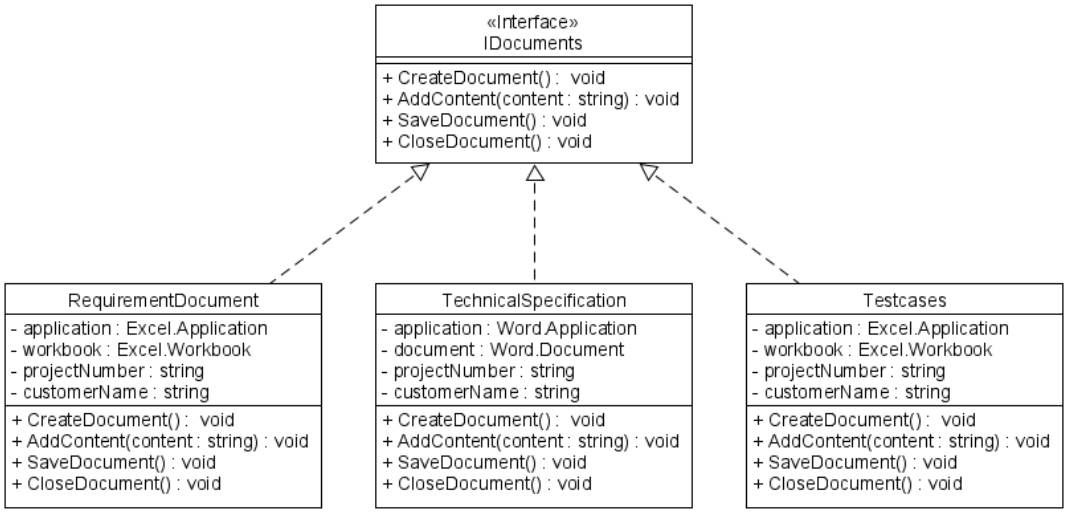

Abbildung 4.3: Klassendiagramm des Interface **Documents** mit den Unterklassen **RequirementDocument**, **TechnicalSpecification** und **Testcases**

Dokument als Markdown zurückgeben. Zudem ist der Kontext für das LLM sowie die Antwort auf Englisch (Anforderung DG-NF1). Mit den Informationen des LLMs werden anschließend die Dokumente erstellt. Da diese Microsoft Office-Dateien sind, wird dafür das *Microsoft.Office.Interop.Excel* und *Microsoft.Office.Interop.Word* Assembly genutzt (Anforderung AD-NF1, Anforderung TS-NF1, Anforderung TF-NF1).
Für die Erstellung der Dokumente wird ein Interface **Documents** mit den Methoden **CreateDocument()**, **AddContent()**, **SaveDocument()** und **CloseDocument()** implementiert. Diese sind in Abbildung 4.3 abgebildet. Die Methoden werden bei den entsprechenden Dokumenten in den Unterabschnitten 4.5.1, 4.5.2 und 4.5.3 genauer beschrieben.

4.5.1 Anforderungsdokument

Die Übersichtseite des Anforderungsdokuments wird unabhängig von der Antwort des LLMs erstellt. Dies wird mit der Methode **CreateDocument()** der Klasse **RequirementDocument** realisiert (Anforderung AD-F2, Anforderung AD-F3, Anforderung AD-F4). In der gleichen Methode wird auch die zweite Seite („Requirements“) erstellt und die Tabellen Überschriften „No.“, „Description“, „Category“ und „Notes“ (Anforderung AD-F5, Anforderung AD-F6, Anforderung AD-F7).

Anforderungen: Um geeignete Anforderungen zu generieren muss zuerst ermittelt werden, was gute Anforderungen ausmachen. Die generierten Anforderungen sollen die folgenden Regeln einhalten:

Regeln

- Nur eine Anforderung pro Satz
- Kurze Sätze bilden
- Das Subjekt muss eindeutig sein
- Das Passiv vermeiden
- Anforderungen beschreiben keinen Lösungsweg
- Anforderungen sind messbar

Neben den sechs Regeln sollen die Anforderungen zusätzlich immer aus einem Subjekt, Prädikat und Objekt bestehen und durch die Wörter „muss“, „soll“ und „kann“ wird die Priorität beschrieben. [7]

Kontext: Für das Generieren der Anforderung wird ein LLM verwendet. Das Modell soll die Antwort in einem vorgegebenen Format wiedergeben. Zuerst kommt die Nummer der Anforderung „Anf1“, „Anf2“, „Anf3“ usw. (Anforderung AD-F5). Danach kommt ein Leerzeichen und es folgt die Anforderung (Anforderung Ad-F6). Endet die Anforderung, wird ein Zeilenumbruch „\n“ hinzugefügt. Für den System Prompt wird die Few-shot Methode genutzt. Es werden dementsprechend Beispiele übergeben, um eine bessere Antwort zu erhalten.
Da die Kontextlänge der Modelle begrenzt ist und die Anforderungsdokumente vom Kunden umfangreich sein können, müssen diese für den Kontext aufgeteilt werden. Eine Möglichkeit ist immer, nur einzelne Seiten, bzw. eine bestimmte Anzahl von Seiten zu übergeben. Dabei kann es jedoch vorkommen, dass Sätze, Wörter oder Abschnitte unterbrochen werden und somit die Informationen verloren gehen. Demzufolge wird das Dokument auf die einzelnen Kapitel aufgeteilt. Zudem werden Seiten, die keine relevanten Informationen für die einzelnen Anforderungen enthalten, ignoriert. Dies bezieht sich auf die Titelseite und das Inhalts-, Tabellen- und Abbildungsverzeichnis.

Erstellen: Um die Antwort des LLMs korrekt in das Anforderungsdokument einzufügen, wird die Methode **AddContent()** implementiert. Diese bekommt die Antwort übergeben und fügt diese in die erstellte Exceldatei ein. Die Antwort hat, wie zuvor bereits beschrieben, immer eine feste Struktur. Der Programmablauf der Methode **AddContent()** ist in Abbildung 4.4 dargestellt. Zunächst wird die Anzahl der Anforderungen

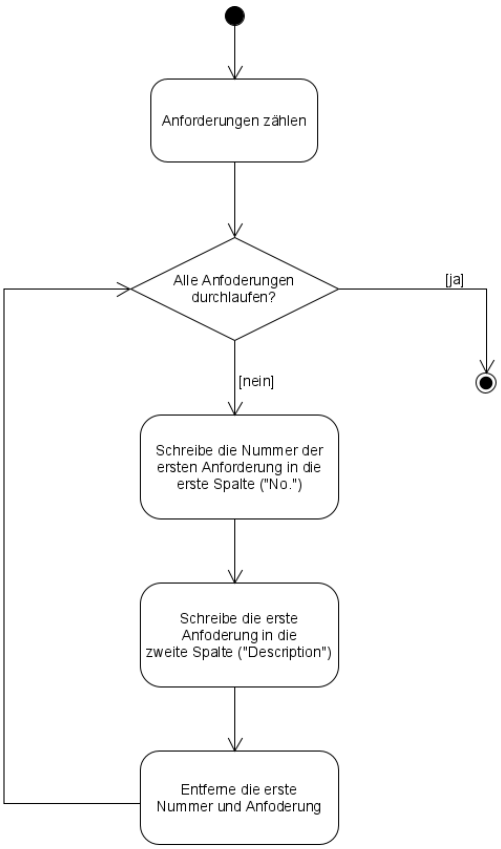

Abbildung 4.4: Programmablaufplan der Methode **AddContent()** aus der Klasse **RequirementDocument**

gezählt. Dazu wird ein regulärer Ausdruck genutzt, der den **String** nach den Anforderungsnummern durchsucht. Daraufhin wird über eine for-Schleife jede Anforderung durchlaufen. Dabei wird zuerst die erste Nummer abgeschnitten und in die Spalte „No.“ eingefügt. Anschließend wird die Anforderung selbst abgeschnitten und in die Spalte „Description“ eingefügt(Anforderung AD-F5, Anforderung AD-F6).

Mit der Methode **SaveDocument()** wird das Anforderungsdokument gespeichert. Für das Speichern muss ein Dateiname, nach den Richtlinien aus Abschnitt A.1, erstellt werden. Dieser besteht aus der Projektnummer, dem **D**ocument **C**lassification **C**ode (DCC), der Dokumentennummer, dem Status, der Version, sowie dem Dokumentennamen. Die Projektnummer kann bei verschiedenen Dokumenten unterschiedlich sein. Die restlichen Parameter sind jedoch immer gleich. Der DCC ist „EC“, was für technische Spezifikations- und Anforderungsdokumente steht. Die Dokumentennummer wird auf „411“ festgelegt.

Der Status beträgt „W“, für in Arbeit und die Version ist „01“. Der Dokumentenname lautet „RequirementList“ (Anforderung AD-F9).

4.5.2 Technische Spezifikation

Die technische Spezifikation wird aus dem Anforderungsdokument generiert (Anforderung DG-F3, TS-F1) und als Word-Datei bereitgestellt (Anforderung TS-NF1). Ähnlich zu Unterabschnitt 4.5.1 wird die Kopf- und Fußzeile (Anforderung TS-F2, Anforderung TS-F3), die Titelseite (Anforderung TS-F4, Anforderung TS-F5, Anforderung TS-F6), das Versionsmanagement (Anforderung TS-F10) und der Dokumentenstatus (Anforderung TS-F11) unabhängig von der Antwort des LLMs erstellt. Dies wird ebenfalls in der Methode **CreateDocument()** der Klasse **TechnicalSpecification** implementiert.

Kontext: Für die technische Spezifikation wird ein Anforderungsdokument übergeben. Die technische Spezifikation soll dabei nur für eine Kategorie („Category“) erstellt werden, welche zuvor vom Nutzenden ausgewählt wird. Dazu werden als Kontext für das LLM lediglich die Anforderungen der vorgegebenen Kategorie übergeben, sowie die Abschnitte, in denen diese aufgelistet waren (Anforderung). Auch in diesem Fall soll die Few-shot Methode genutzt werden, um die Qualität der Antwort des LLMs durch Beispiele zu verbessern.

Erstellen: Das Erstellen der technischen Spezifikationen ist wesentlich aufwendiger als das Anforderungsdokument oder die Testfälle, da in diesem Fall die Worddatei vernünftig formatiert werden muss. Das bedeutet, es wird ein automatisches Inhaltsverzeichnis, ein Titelblatt und Kapitel und Abschnitte mit Überschriften benötigt. Die technische Spezifikation ist als Fließtext geschrieben, welcher vom Markdown zu Word konvertiert werden muss.

Auch hier ist die Methode **SaveDocument()** zum Speichern des Dokuments. Lediglich der Dateiname sowie das Dateiformat ändern sich im Gegensatz zum Anforderungsdokument. Die Dokumentenbezeichnung lautet in diesem Fall „TechnicalSpecification“ und das Dateiformat ist „.docx“. Der Rest bleibt identisch zum Anforderungsdokument.

4.5.3 Testfälle

Die Testfälle werden aus den technischen Spezifikationen generiert (Anforderung DG-F5, TF-F1). Am Ende werden sie, wie das Anforderungsdokument, als Excel-Datei gespeichert (Anforderung TF-NF1). Wie bei den beiden vorherigen Dokumenten aus Unterabschnitt 4.5.1 und 4.5.2 werden vier Methoden zur Erstellung des Testfalldokuments implementiert: **CreateDocument()**, **AddContent()**, **SaveDocument()** und **CloseDocument()**.
CreateDocument() erstellt das Dokument unabhängig von der Antwort des LLMs. Dazu wird einmal eine Übersicht in einem separaten Worksheet erstellt (Anforderung TF-F2). Anschließend werden auf der zweiten Seite die Spaltenüberschriften „ID“, „section“, „Description“, „Test date“, „Tester“, „Expected result“, „Test result“ und „Notes“ erstellt (Anforderung TF-F3, TF-F4, TF-F5, TF-F6, TF-F7, TF-F8, TF-F9, TF-F10).

Kontext: Die Testfälle sollen in einem ähnlichen Format, wie die Anforderungen übergeben werden. Zuerst wird die ID des Testfalls fortlaufend geschrieben, danach kommt die Testbeschreibung und getrennt durch zwei Et-Zeichen („&&“) das erwartete Testergebnis. Jeder Testfall steht in einer eigenen Reihe. Es soll ebenfalls die Few-shot Methode für das Prompting genutzt werden, um die Beschreibung mit Beispielen zu verbessern. Um die Kontextlänge nicht zu überschreiten, sollen, wie bei dem Anforderungsdokument, einzelne, aber in sich abgeschlossene, Abschnitte an das LLM übergeben werden. Die Antworten werden dann aneinander gehängt. Werden aus einem Abschnitt keine Testfälle generiert, soll vordefinierte Antwort übergeben werden.

Erstellen: Das Erstellen des Dokuments ist bei den Testfällen, im Gegensatz zu der technischen Spezifikation, ähnlich zum Anforderungsdokument. Mit der Methode **AddContent()** wird die LLM Antwort in die entsprechenden Zellen geschrieben. Auch der Programmablauf ähnelt sich zu dem des Anforderungsdokuments. Die Änderungen sind, dass es sich um Testfälle statt Anforderungen handelt und dementsprechend andere Spalten beschrieben werden. Es handelt sich um die Spalten „ID“, „Description“ und „Expected result“ (Anforderung TF-F3, TF-F5, TF-F8).
Für die Testfälle ist die Methode **SaveDocument()** wieder ähnlich zum Anforderungsdokument. Die Dokumentenbezeichnung ändert sich zu „Testcases“ und der DCC zu „WT“, was für Logbücher und Prüfprotokolle steht. Somit ändert sich auch der Dateiname, der Rest ist jedoch identisch mit dem Anforderungsdokument.

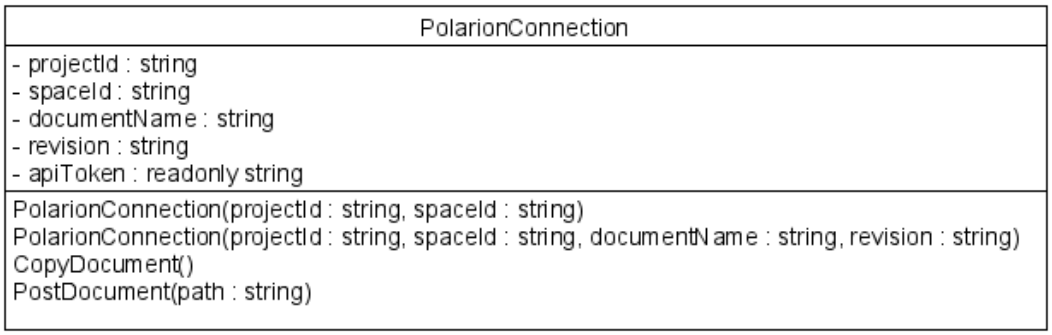

Abbildung 4.5: Klassendiagramm der Klasse **PolarionConnection**

4.6 Anforderungsmanagement Tool (Polarion)

Polarion ist eine Anforderungsmanagement Anwendung von Siemens und soll für das Verwalten von Projekten genutzt werden. Dort werden alle Dokumente, welche im Laufe eines Projekts anfallen, abgelegt. Dazu zählen ebenfalls die Dokumente (Anforderungen, technische Spezifikation, Testfälle), die durch diese Arbeit generiert werden sollen. Damit die Nutzenden die Dokumente nicht selbstständig ablegen müssen, soll dies über die Anwendung automatisiert werden. Polarion verfügt über eine REST API, mit welcher sich die Dokumente aus einem Projekt verwalten lassen. Dazu wird das Paket *RestSharp* verwendet.
Zur Kommunikation mit Polarion wird die Klasse **PolarionConnection** implementiert, welche Methoden zur Auswahl und Speichern von Dokumenten enthält. Des Weiteren enthält die Klasse vier Attribute (**projectId**, **spaceId**, **documentName**, **revision**), zur Ermittlung des Speicherorts. Das Klassendiagramm der Klasse **PolarionConnection** ist in Abbildung 4.5 abgebildet.

4.6.1 Auswahl der Dokumente

Mit der Methode **CopyDocument()** wird ein ausgewähltes Dokument kopiert, um anschließend daraus ein anderes Dokument mit dem LLM zu generieren. Das Dokument wird vom Nutzenden ausgewählt, gleichzeitig wird die Projektnummer gespeichert, um diese bei der Erstellung der Dokumente sowie für das Speichern zu verwenden. Für das Wiedergeben von Dokumenten aus Polarion gibt es bereits eine Vorlage von Polarion,

welche in Listing 4.5 dargestellt ist.[34] In der URL für den REST-Client stehen dabei die Informationen Projektnummer (projectId), Ordner (spaceId), Dokumentenname (documentName) und Version (revision). Die Version ist dabei optional. Um auf den Standardordner zuzugreifen, wird „_default“ für den Ordner verwendet.

```
var client = new RestClient("https://example.com/polarion/rest/v1/projects
    /{projectId}/spaces/{spaceId}/documents/{documentName}/actions/copy?
    revision={revision}");
client.Timeout = -1;
var request = new RestRequest();
request.Method = Method.POST;
request.AddHeader("Content-Type", "application/json");
request.AddHeader("Accept", "application/json");
request.AddHeader("Authorization", "Bearer {personal_access_token}");
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Listing 4.5: Methode **CopyDocument()** zum Kopieren eines Dokuments aus Polarion

4.6.2 Speichern der Dokumente

Mit **patchDocument** können Dokumente aktualisiert werden. Zudem gibt es die Funktion, Dokumente mit PDF-Attachments zu versehen. Das Ablegen, bzw. Hochladen von lokalen Dokumenten, mit der REST API, ist nicht möglich.[35,36] Die Vorlage für das Aktualisieren von Dokumenten ist in Listing 4.6 dargestellt. Dazu wird der Projektname (projectId), der Ordner (spaceId) und der Dokumentenname (documentName) benötigt. Die Aktion (workflowAction) wird im **J**ava**S**cript **O**bject **N**otation (JSON)-Format übergeben. Das Schema dazu ist in Listing 4.7 abgebildet. Um nun die generierten Informationen in ein entsprechendes Dokument zu schreiben, muss dieses bereits existieren.

[34] https://testdrive.polarion.com/polarion/sdk/doc/rest/index.html#api-Documents-getDocument, Zugriffsdatum: 08.05.2025
[35] https://testdrive.polarion.com/polarion/sdk/doc/rest/index.html#api-Documents-postDocuments, Zugriffsdatum: 08.05.2025
[36] https://gsit-polarion-008.siemens.net/polarion/sdk/doc/rest/index.html#api-DocumentAttachments-patchDocumentAttachment, Zugriffsdatum: 13.05.2025

```
var client = new RestClient("https://example.com/polarion/rest/v1/projects
    /{projectId}/spaces/{spaceId}/documents/{documentName}?workflowAction
    ={workflowAction}");
client.Timeout = -1;
var request = new RestRequest();
request.Method = Method.PATCH;
request.AddHeader("Content-Type", "application/json");
request.AddHeader("Accept", "application/json");
request.AddHeader("Authorization", "Bearer {personal_access_token}");
IRestResponse response = client.Execute(request);
Console.WriteLine(response.Content);
```

Listing 4.6: Aktualisierung von Dokumenten in Polarion (**patchDocument**)

4.7 Grafische Benutzeroberfläche

Zur Bedienung der Anwendung wird eine grafische Benutzeroberfläche implementiert (Anforderung DG-F7). Mit der Plattform .NET Core 9.0 gibt es die Möglichkeiten, die Benutzeroberfläche mit *Windows Presentation Foundation*, *Windows Forms* und *.NET MAUI* zu implementieren. Für die Auswahl eines geeigneten Benutzeroberflächenframeworks werden die folgenden Punkte betrachtet:

- Eingabefelder für die Eingabe eines Dokumentenverzeichnisses, Zielverzeichnisses und der Projektnummer (Anforderung DG-F10, Anforderung DG-F11, Anforderung DG-F12).
- Steuerfelder in Form von Button zur Auswahl des zu generierenden Dokumententyps (Anforderung DG-F8).
- Das Benutzeroberflächenframeworks kann unter Windows ausgeführt werden (Anforderung DG-NF3).

Die oben genannten Punkte können alle mit den drei verschiedenen Benutzeroberflächenframeworks realisiert werden. *Windows Presentation Foundation* kann nur unter Windows ausgeführt werden und *Windows Forms* ist für die Entwicklung von Windows-Anwendungen ausgelegt.[37,38,39]

[37] https://learn.microsoft.com/de-de/dotnet/maui/what-is-maui?view=net-maui-9.0, Zugriffsdatum: 14.05.2025
[38] https://learn.microsoft.com/de-de/dotnet/desktop/winforms/overview/?view=netdesktop-9.0, Zugriffsdatum: 14.05.2025
[39] https://learn.microsoft.com/de-de/dotnet/desktop/wpf/overview/, Zugriffsdatum: 14.05.2025

Die Möglichkeit, die Anwendung auf mehreren Plattformen ausführen zu können, ist momentan nicht notwendig, mit Blick auf die Weiterentwicklungsmöglichkeiten jedoch eine nützliche Funktion. Aus diesem Grund wird das Benutzeroberflächenframeworks *.NET MAUI* verwendet.

Der Aufbau der grafischen Benutzeroberfläche ist in Abbildung 4.6 dargestellt. Es gibt ein Feld zur Eingabe der Projektnummer, sowie zur Auswahl des Dokumentenpfads. Zudem ist es möglich, das Zielverzeichnis auszuwählen (Anforderung DG-10, Anforderung DG-11, Anforderung DG-12). Mit den drei Buttons „Create requirement document“, „Create technical specification“ und „Create test cases“ lassen sich die einzelnen Dokumententypen erstellen (Anforderung DG-F8). Es ist jedoch notwendig, zuvor die Projektnummer sowie ein Dokument auszuwählen. Das Zielverzeichnis muss nicht ausgewählt werden, in dem Fall wird das erstellte Dokument auf dem Desktop abgelegt.

```
"data": {
  "type": "documents",
  "id": "MyProjectId/MySpaceId/MyDocumentId",
  "attributes": {
    "autoSuspect": true,
    "homePageContent": {
      "type": "text/html",
      "value": "My text value"
    },
    "outlineNumbering": {
      "prefix": "ABC"
    },
    "renderingLayouts": [
      {
        "label": "My label",
        "layouter": "paragraph",
        "properties": [
          {
            "key": "fieldsAtStart",
            "value": "id"
          }
        ],
        "type": "task"
      }
    ],
    "status": "draft",
    "title": "Title",
    "type": "req_specification",
    "usesOutlineNumbering": true
  }
}
```

Listing 4.7: JSON-Schema zum Übertragen der Aktion für die Aktualisierung von Dokumenten in Polarion

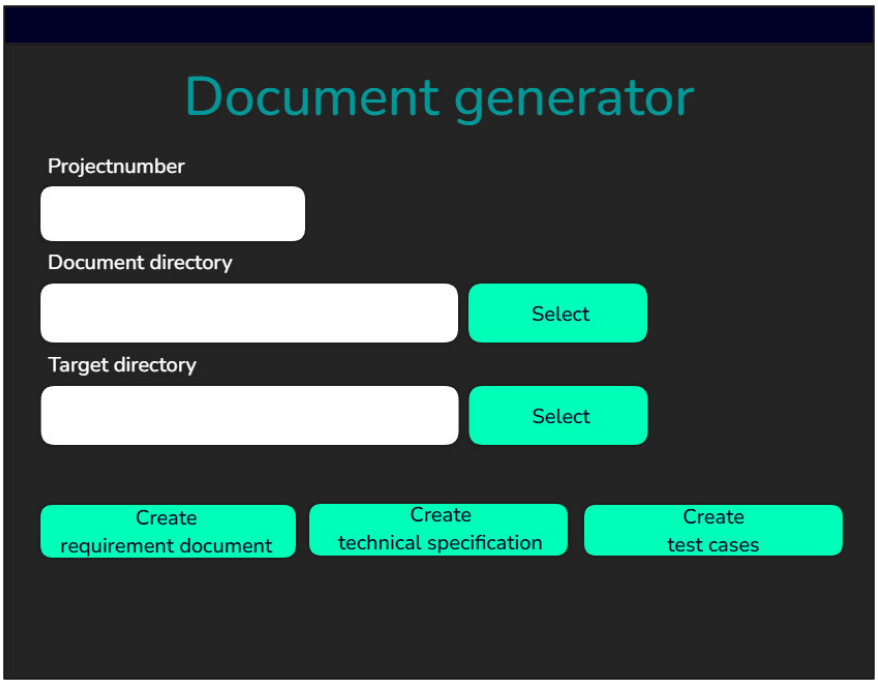

Abbildung 4.6: Design der grafischen Benutzeroberfläche

5 Implementierung

Beim Beschreiben der Implementierung wird zunächst auf die Kommunikation mit dem LLM eingegangen. Es wird erläutert, welche Probleme auftreten und wie diese gelöst werden. Danach wird das Einlesen und Verarbeiten einer PDF-Datei beschrieben und wie dieses, im Gegensatz zum Konzept, implementiert wurde. Darauf folgt das Erstellen der einzelnen Dokumente (Anforderungsdokument, technische Spezifikation, Testfälle) und das Ablegen der Dokumente in das Anforderungsmanagement Tool Polarion. Zum Schluss wird auf die grafische Benutzeroberfläche der Anwendung eingegangen.

5.1 Kommunikation mit dem LLM

Um auch Fine-Tuning an dem LLM vorzunehmen, wurde die Klasse **LlmConnection** nach dem Klassendiagramm aus Abbildung 5.1 implementiert. Nun ist es möglich, auch die Temperatur des LLMs anzupassen. Zudem wurden zwei Methoden für eine LLM Antwort erstellt (**LlmChatRequest()**, **LlmImageRequest()**). Somit wurden Chat-Anfragen und visuelle-Anfragen aufgeteilt. Zudem werden zwei private Methoden (**GetAIClient()**, **LlmStreamResponse()**) implementiert, da der Client sowie die Antwort für beide Anfragen gleich sind. Eine weitere Änderung ist, dass die Antwort als Stream zurückgegeben wird. Darauf wird am Ende dieses Abschnitts genauer eingegangen.

Bei der Methode **LlmChatRequest()**, wird die Nachricht nach dem Schema aus Listing 4.1 aufgebaut. Die Methode **LlmImageRequest()** konvertiert das Bild zunächst in einen Base64 String. Die Nachricht wird anschließend nach dem Schema aus Listing 4.2 erstellt. Dazu muss der Inhalt **„content“** noch einmal extra erstellt werden, da es in dem Fall zwei Contenttypen **„Text, ImageUrl“** gibt. Erst dann kann die Nachricht erstellt und übermittelt werden. Zudem wird das Modell immer auf das Pixtral 12B gesetzt und das übergebene Modell ignoriert. Pixtral 12B ist extra für visuelle Aufgaben ausgelegt

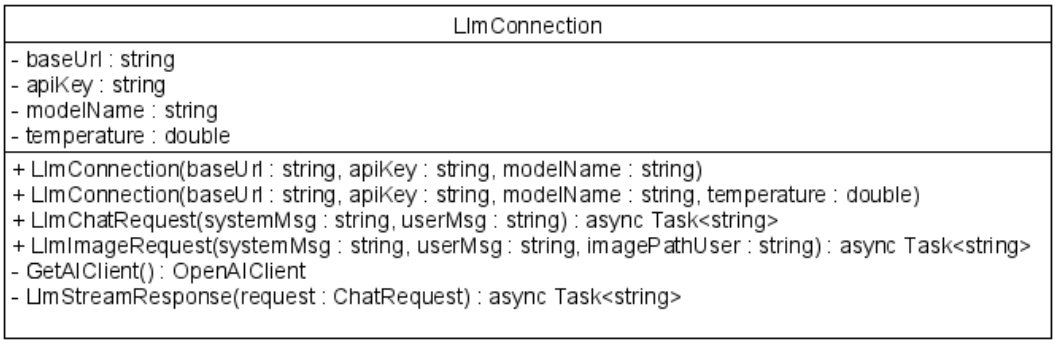

Abbildung 5.1: Klassendiagramm der Klasse **LlmConnection**

und entwickelt worden, weshalb das Modell am besten für die Aufgabe geeignet ist. Die Methode ist in Listing 5.1 dargestellt.

Bei der Kommunikation mit den Chatmodellen Deepseek R1, sowie dem Modell Pixtral 12B treten Probleme bei der Antwort auf. Die drei Modelle benötigen für die Antwort zu lange, weshalb es bei den drei Modellen zu Timeout Fehlern (408 Request Timeout) kommt. Aus diesem Grund wird an das Objekt „client“ vom Typ **OpenAIClient**, neben den Einstellung und dem Authentifizierungsschlüssel, noch ein Http-Client (**HttpClient**) übergeben, bei dem der Timeout auf fünf Minuten hochgesetzt wird. Mit dieser Änderung kann das Modell Deepseek R1 genutzt werden, ohne einen Timeout-Fehler zu generieren. Das Modell Pixtral 12B geniert jedoch weiterhin einen Timeout Fehler (408 Request Timeout). Zudem wird der Fehler bereits ausgeworfen, bevor die fünf Minuten vorbei sind. Um auch das Pixtral Modell nutzen zu können wird die Antwort des LLMs als Stream zurückgegeben. Die Methode **LlmStreamResponse()** gibt die Antwort zum Ende vollständig, als **String**, zurück.

5.2 Einlesen und Verarbeiten einer PDF

Das Einlesen einer PDF-Datei mit dem Paket *PdfPig* oder über *Pixtral 12B* verursacht einige Probleme. Zum Auslesen des Textes eines PDFs wird die Methode **GetText** implementiert. Diese durchläuft jede Seite der PDF-Datei und schreibt den Text Zeile für Zeile und Wort für Wort in einen **String**. Mit *PdfPig* lässt der Text aus einer PDF-Datei, ohne großen Aufwand, komplett und vollständig extrahieren. Jedoch behält dieser nicht sein ursprüngliches Format, da der Text Zeile für Zeile ausgelesen und übergeben wird.

```
// Image request
else
{
    // Convert image to Base64 string
    var imageBytes = File.ReadAllBytes(imagePath);
    var base64Image = Convert.ToBase64String(imageBytes);

    // Create list of content for the messages
    var contentList = new List<Content>
    {
        new Content(ContentType.Text, userMsg),
        new Content(ContentType.ImageUrl, $"{
            base64Image}")
    };

    // Create list of messages
    var imageMessageList = new List<Message>
    {
        new Message(OpenAI.Role.User, contentList)
    };

    // Chat request
    request = new ChatRequest(
        messages: imageMessageList,
        model: "pixtral-12b-2409"
    );
}
```

Listing 5.1: Impelmentierung eines Bild-Request aus der Methode **LlmChatRequest()** zur Kommunikation mit einem LLM

Formen, wie Linien, werden nicht übergeben, weshalb sich die Darstellung von Tabellen problematisch darstellt. In Unterabschnitt 4.4.2 wird bereits eine Methode zum Extrahieren von Tabellen mit *PdfPig* aufgezeigt. Diese ist jedoch recht aufwendig und liest die Tabellen aus, ohne sie an die korrekte Stelle des Textes einzufügen. Dazu müsste eine weitere Methode implementiert, bzw. die Methode **GetText()** dahingehend geändert werden.
Wenn der Text einer PDF-Datei nun statt mit *PdfPig* mit Hilfe des Modells *Pixtral 12B 2407* extrahiert werden soll, tauchen zwei andere Probleme auf. Zum einen ist das Bild, welches aus einer PDF-Seite erstellt wird, zu groß, wenn diese selbst ein Bild beinhaltet. Aus diesem Grund muss die Auflösung auf 200 dpi begrenzt werden, wenn eine PDF-Seite in ein PNG konvertiert wird. Zum anderen werden Zeichen falsch interpretiert. Aus einem „T“ wird beispielsweise ein „I“ oder das große „I“ wird mit dem kleinen „ℓ“ vertauscht.

Was auffällt ist, dass beide Konzepte zusammen sowohl den Text korrekt wiedergeben können, als auch richtig darstellen können, beispielsweise in Tabellenform. Des Weiteren kann die PDF-Datei, durch die Verarbeitung mit dem *Pixtral 12B* Modell, direkt als Markdown wiedergegeben werden, was die Weiterverarbeitung für den Kontext zur Erstellung der Projektmanagementdokumente wesentlich erleichtert. Aus den genannten Gründen wird zunächst der Text der PDF-Datei mit dem Paket *PdfPig* extrahiert. Anschließend verarbeitet das *Pixtral 12B* Modell jede Seite einzeln, in dem der zuvor extrahierte Text der Seite als Kontext, sowie ein Bild der Seite an das Modell übergeben wird. Die Seite soll anschließend als Markdown wiedergegeben werden. Die Bereitstellung des Inhalts mit einem LLM hat den Vorteil, dass dieser direkt für den Kontext weiterverarbeitet werden kann und kein Algorithmus zum Filtern der einzelnen Kapitel und Abschnitte notwendig ist. Für die Kombination der beiden Funktionen wird die Klasse **DocImageToString** erstellt. Diese beinhaltet die Methode zum **GetPdfPagesAsString()** Laden des PDF-Inhalts. Dort wird der Text der PDF-Datei zuerst mit *PdfPig* ausgelesen. Anschließend wird die PDF in Bilder konvertiert. Danach wird Seitenweise das Bild zusammen mit dem Text an das LLM übergeben, um daraus die Seite als Markdown darzustellen.

5.3 Erstellen der Dokumente

Das Erstellen wird nach dem Konzept aus Abschnitt 4.5 implementiert. Dazu enthält jede Klasse zur Generierung der Dokumente die Methoden **CreateDocument()**, **AddContent()**, **SaveDocument()** und **CloseDocument()**.

5.3.1 Anforderungsdokument

Mit der Methode **CreateDocument()** wird das Anforderungsdokument entsprechnend den Anforderungen AD-F2, AD-F3, AD-F4, AD-F5, AD-F6 und AD-F7 formatiert. Dazu wird lediglich eine Exceldatei erstellt und die Zellen werden mit den Informationen beschrieben.
Die Methode **AddContent** zum Hinzufügen der LLM-Antwort in das Anforderungsdokument ist in Listing 5.2 dargestellt. Die Anforderungen werden vom LLM (unabhängig vom Modell) mit dem Aufbau aus Unterabschnitt 4.5.1 wiedergegeben. Das Aufschlüsseln der Antwort wird mit regulären Ausdrücken realisiert. Zunächst wird mit der Methode

ParseLLMResponse() aus der Klasse **LLMResponseParser** der Gedankengang des LLMs ausgeschnitten. Die beiden Modelle *Deepseek R1 Distill Qwen 7B* und *Qwen3 30B A3B* liefern ihre Antwort mit dem Gedankengang. Anschließend wird überprüft, ob die Antwort überhaupt eine Anforderung enthält. Ist dies nicht der Fall übergibt das LLM die Antwort „$empty$“. Die generierten Anforderungen werden danach aufgeteilt und in den Array *splitContent* geschrieben. Daraufhin werden sie einzeln in die jeweiligen Zellen des Excel Worksheets geschrieben (Anforderung AD-F4, AD-F5, AD-F6).

5.3.2 Technische Spezifikation und Testfälle

Die Generierung der technischen Spezifikationen und Tesfälle wird aus zeitlichen Gründen nicht implementiert. Es werden lediglich die Klassen **TechnicalSpecification** und **Testcases** gemäß Abbildung 4.3 aus Abschnitt 4.5, sowie die Methoden **CreateDocument()**, **AddContent()**, deklariert. Die beiden Methoden **SaveDocument()** und **CloseDocument()** wurden bereits implementiert, da sich diese Methoden untereinander sehr ähneln. Nur der Dateipfad ändert sich, sowie die Dateiendung von „.xlsx“ zu „.docx“ bei der technische Spezifikation.

5.4 Anforderungsmanagement Tool (Polarion)

Polarion wird noch nicht offiziell in der Abteilung genutzt. Es gibt einen Testzugang mit dem überprüft wird, ob die Anwendung nützlich für die Abteilung ist. Aus diesem Grund können Dokumente nur aus den Testprojekt entnommen und nur in die Testprojekte abgelegt werden. Dies schränkt dabei Funktionen der Anwendung, wie die korrekte Namensgebung über die Projektnummer und das Entnehmen aus und Ablegenden in das zugehörige Verzeichnis, ein. Bei der Implementierung der Schnittstelle zu Polarion wird daher nur auf die generelle Funktionalität eingegangen, Dokumente aus Polarion zu entnehmen und dort abzulegen.

5.4.1 Kopieren der Dokumente

Für die Kommunikation mit Polarion, muss zunächst der Server gewechset werden. Der Testzugang wurde für den *008* Server eingerichtet, welcher die REST API nicht vollumfänglich unterstützt. Daher musste erst ein Zugang für den *006* eingerichtet werden.

Bei *008* und *006* handelt es sich lediglich um die Serverbezeichnungen und sind für die Generelle Funktionalität nicht notwendig.
Zum Kopieren von Dokumenten wurde anschließend ein Live Dokument „TestDoc“ im Standardraum *Default Space* („_default“) angelegt, um die Funktionalität der Methode **CopyDocument()** zu testen. Zudem muss ein API-Token, für die Authentifizierung, generiert werden. Die Methode wurde nach Listing 4.5 implementiert. Die URL für das Kopieren von Dokumenten ist in Listing 5.3 dargestellt.

5.4.2 Speichern der Dokumente

Das Ablegen der Dokumente in Polarion ist, wie in Unterabschnitt 4.6.2 bereits erwähnt, nicht möglich. Es können lediglich LiveDocs angelegt und aktualisiert werden. Die Implementierung zur Bearbeitung von LiveDocs ist mit großem Zeitaufwand verbunden. Aufgrund des zeitlichen Rahmens und des Umfangs der Arbeit wird diese Funktion nicht weiter implementiert.

5.5 Grafische Benutzeroberfläche

Das Design der grafischen Benutzeroberfläche wird nach Abbildung 4.6 aus Abschnitt 4.7 implementiert (Anforderung DG-F7). Es gibt jeweils ein Eingabefeld für die Projektnummer, den Dateipfad, sowie das Zielverzeichnis (Anforderung DG-F10, DG-F11, DG-F12). Zudem ist es möglich, den Dateipfad und das Zielverzeichnis über den File Explorer auszuwählen. Dabei sind nur „.pdf“, „.docx“ und „.xlsx“ Dateien zugelassen. Es gibt drei Schaltflächen zum Generieren der einzelnen Dokumente (Anforderung DG-F8). Dazu muss zuvor eine Datei ausgewählt werden, aus der das Dokument generiert werden soll. Zudem muss es die entsprechende Dateiendung haben (Anforderung AD-F1, TS-F1, TF-F1). Das endgültige Design der Benutzeroberfläche ist in Abbildung 5.2 abgebildet.

Das Design der Benutzeroberfläche ist zwar identisch zum Konzept aus Abbildung 4.6 geblieben, jedoch wird statt des Benutzeroberflächenframeworks .NET Maui 9.0 die Version 8.0 genutzt. Dies hat den Grund, dass einerseits die Neuerungen von .NET Maui 9.0

gegenüber 8.0, wie das Einführen einer Titelleiste für Windows, in dieser Arbeit nicht notwendig sind.[1] Das Framework .NET Maui 8.0 bietet gegenüber 9.0 ein Langzeitsupport an, welches im Hinblick auf die Weiterentwicklungsmöglichkeiten einen Vorteil bietet. Zudem gibt es einen bekannten Bug bei .NET Maui 9.0 bei der Nutzung der Klasse **FilePicker**, zum Auswählen einer Datei über den Windows-Explorer.[2]

[1] https://learn.microsoft.com/de-de/dotnet/maui/whats-new/dotnet-9?view=net-maui-9.0, Zugriffsdatum: 20.05.2025
[2] https://github.com/dotnet/maui/issues/27552, Zugriffsdatum: 20.05.2025

```
1  public void AddContent(List<string> contentList)
2  {
3      // Load the second worksheet
4      _Excel.Worksheet worksheet = workbook.Worksheets[2];
5      var count = 1;
6
7      // Running through the individual LLM answers
8      foreach (var content in contentList)
9      {
10         // Remove the chain of thoughts
11         var parsedContent = LLMResponseParser.ParseLLMResponse(content,
               "</think>");
12
13         // Check whether the answer contains requirements
14         if (!Regex.IsMatch(parsedContent, @"\$empty\$"))
15         {
16             // Split requirements
17             var splitContent = Regex.Split(parsedContent, @"^Anf[\d]{1,}",
                   RegexOptions.Multiline);
18
19             // Write requirements to the file
20             for (int i = 0; i < splitContent.Length; i++)
21             {
22                 if (!String.IsNullOrEmpty(splitContent[i]))
23                 {
24                     // Write requirement number
25                     worksheet.Cells[count + 1, 1] = "Anf" + count;
26
27                     // Delete space at the beginning if the element
                           conatins one
28                     if (Regex.IsMatch(splitContent[i], @"^ "))
29                     {
30                         splitContent[i] = splitContent[i][1..];
31                     }
32
33                     // Add the chapter/ section to the table
34                     worksheet.Cells[count + 1, 3] = Regex.Match(
                           splitContent[i], @"^([A-Za-z][\d\.]{1,}|[\d
                           \.]{1,})");
35                     // Add the requirement to the table
36                     worksheet.Cells[count + 1, 4] = splitContent[i][
                           splitContent[i].IndexOf(' ')..];
37
38                     count++;
39                 }
40             }
41         }
42     }
43 }
```

Listing 5.2: Impelmentierung der Methode **AddContent()** aus der Klasse **RequirementDocument** zum Einfügen der LLM Antwort in die Excel-Datei

```
$"https://gsit-polarion-006.siemens.net/polarion/rest/v1/
   projects/{projectId}/spaces/{spaceId}/documents/{
   documentName}/actions/copy?revision={revision}"
```

Listing 5.3: URL für die Kommunikation mit Polarion über eine REST-API

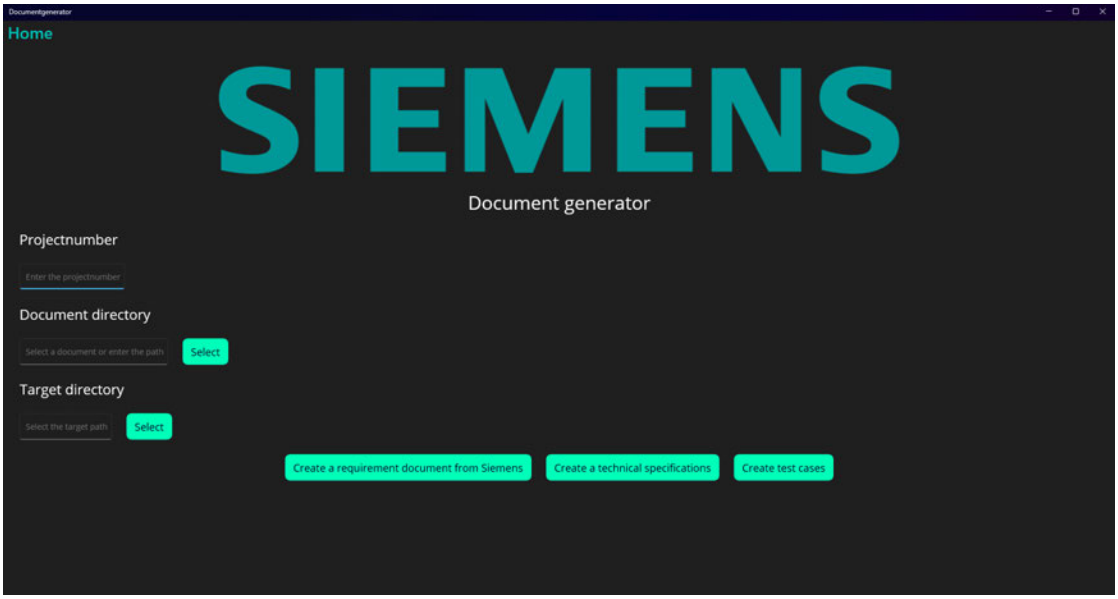

Abbildung 5.2: Design der grafischen Benutzeroberfläche

6 Evaluation

Nachdem die Implementation der Anwendung beschrieben wurde, wird diese evaluiert. Es wird geprüft, welche Anforderungen aus Kapitel 3 erfüllt wurden und ob die Anwendung erfolgreich implementiert wurde. Zudem wird auf die Qualität der generierten Anforderungen eingegangen.

6.1 Dokumentengenerator

Zunächst werden die Anforderungen an den Dokumentengenerator während der Nutzung der Anwendung überprüft. Die Ergebnisse der einzelnen Anforderungen sind im Folgenden aufgeführt.

DG-F1 [muss]: Aus dem Anforderungsdokument des Kunden wird ein Anforderungsdokument von Siemens generiert.

Erfüllt: Aus Anforderungen vom Kunden werden mit Hilfe von LLMs eigene Anforderungen generiert und in eine Excel-Datei geschrieben (vgl. Unterabschnitt 5.3.1). Auf die Anforderungen selbst wird in Unterabschnitt 6.3.2 genauer eingegangen.

DG-F3 [muss]: Aus dem Anforderungsdokumenten von Siemens werden technische Spezifikationen generiert.

Nicht erfüllt: Aufgrund von Zeit- und Umfangsbeschränkungen konzentriert sich diese Arbeit hauptsächlich auf die Erstellung von Anforderungsdokumenten. (vgl. Abschnitt 3.5).

Für die Generierung von technischen Spezifikationen wurde lediglich ein grobes Konzept in Unterabschnitt 4.5.2 erstellt.

DG-F5 [muss]: Aus einer technischen Spezifikation werden Testfälle generiert.

Nicht erfüllt: Wie bereits zuvor, für Anforderung DG-F3 erläutert, lag das Generieren von Anforderungsdokumenten im Vordergrund (vgl. Abschnitt 3.5). Das Generieren von Testfällen wurde nicht implementiert, es wurde nur ein Konzept in Unterabschnitt 4.5.3 entwickelt.

DG-F7 [muss]: Die Anwendung besitzt eine grafische Benutzeroberfläche.

Erfüllt: Für die Anwendung wird eine grafische Benutzeroberfläche bereitgestellt. (vgl. Abschnitt 5.5)

DG-F8 [muss]: Es ist möglich den zu generierenden Dokumententypen in der grafischen Benutzeroberfläche auszuwählen. Es gibt drei Dokumententypen: Anforderungsdokument, technische Spezifikation, Testfälle.

Erfüllt: Auf der grafischen Benutzeroberfläche können, die zu generierenden Dokumententypen, über drei Schaltflächen ausgewählt werden. Dabei wird überprüft, ob eine Datei mit dem korrekten Typ ausgewählt wurde, andernfalls erscheint eine Fehlermeldung und die Dokumentengenerierung wird nicht durchgeführt.

DG-F9 [kann]: Die grafische Benutzeroberfläche besitzt eine Auswahl, ob mit dem Anforderungsmanagement Tool oder lokal gearbeitet wird.

Nicht erfüllt: Da es für Polarion nur einen Testzugang gibt und die Plattform nicht sinngemäß genutzt werden kann, gibt es keine Möglichkeit auszuwählen, ob mit Polarion

oder lokal gearbeitet wird. Es können ausschließlich lokale Dokumente ausgewählt und generierte Dokumente lokal gespeichert werden.

DG-F10 [muss]: Die Projektnummer wird von den Anwendenden eingegeben.

Erfüllt: Ein Eingabefeld zur Eingabe der Projektnummer ist vorhanden. Wird keine Projektnummer eingegeben, wird eine Fehlermeldung ausgeworfen.

DG-F11 [muss]: Es muss ein lokales Dokument ausgewählt werden, aus dem das Projektmanagementdokument generiert wird.

Erfüllt: Ohne die Auswahl eines lokalen Dokuments kann kein Dokument generiert werden und es taucht eine Mitteilung auf, dass ein Dokument auszuwählen ist.

DG-F12 [muss]: Die Nutzenden können ein Zielverzeichnis, für das generierte Dokument auswählen. Wird dieses nicht ausgewählt, wird das Dokument auf dem Desktop abgelegt.

Erfüllt: Die Eingabe des Zielverzeichnisses ist möglich und das Dokument wird in dem Zielverzeichnis gespeichert. Wurde kein Zielverzeichnis ausgewählt, wird es auf dem Desktop gespeichert.

Anforderung DG-F2, DG-F4, DG-F6, DG-F13, DG-F14 und DG-F15

Nicht erfüllt: Die Plattform Polarion wird aktuell nicht von der Abteilung genutzt, sondern es existiert nur ein Testzugang, welcher von mehreren Abteilungen genutzt wird. Dementsprechend werden die Projekte nicht über Polarion verwaltet und es ist nicht möglich, die generierten Dokumente in dem zugehörigen Projekt in Polarion abzulegen. Zudem ist es nicht möglich, die Dokumente aus Polarion auszuwählen.

In der Tabelle 6.1 ist das Ergebnis der nicht funktionalen Anforderungen für den Dokumentengenerator abgebildet. In Bezug auf Anforderung DG-NF3 ist es, neben der

Ausführung der Anwendung auf einem Windows-Betriebssystem, auch möglich, die Anwendung auf weiteren Plattformen auszuführen.

Tabelle 6.1: Evaluation der nicht funktionalen Anforderungen des Dokumentengenerators

Anforderung	Ergebnis
DG-NF1	Erfüllt
DG-NF2	Erfüllt
DG-NF3	Erfüllt
DG-NF4	Erfüllt
DG-NF5	Erfüllt

6.2 Polarion

Bei der Evaluierung der Anforderungen des Dokumentengenerators wurde gezeigt, dass die Anforderungen zum Thema Polarion nicht erfüllt wurden. In Abschnitt 4.6 aus dem Konzept und Abschnitt 5.4 aus der Implementierung wurde ebenfalls darauf eingegangen, dass die Anforderungen an Polarion in der Form nicht umsetzbar sind. Da das Tool jedoch in Zukunft eingesetzt werden soll, wurde dennoch eine Testmethode (**CopyDocument()**) implementiert, um die generelle Funktionsweise zu überprüfen.
Beim Ausführen der Methode wird die folgende Fehlermeldung aus Listing 6.1 ausgeworfen:

```
1 "status":"403","title":"Forbidden","detail":"Sorry, you do not
    have the necessary permissions to perform this operation.
    Please contact your Administrator if you need additional
    permissions."
```

Listing 6.1: Fehlermeldung beim Ausführen der Methode **CopyDocument()**

Es liegen demnach nicht die benötigten Berechtigungen vor, um die Anfrage durchzuführen. Aus diesem Grund kann die Methode nicht weiter getestet werden und es muss auf eine Erteilung der Rechte gewartet werden.

6.3 Anforderungsdokument

In diesem Abschnitt erläutert, welche Anforderungen im Hinblick auf das Anforderungsdokument erfüllt wurden. Zudem wird untersucht, welche Qualität die genierten Anforderungen haben.

6.3.1 Anforderungen

Beim Auswerten der Anforderungen wird nur auf das Modell *Qwen3 30B A3B* Bezug genommen. Dies funktioniert am zuverlässigsten und ist beständig (*stable*) auf der *code.siemens* Plattform. Auf die generelle Qualität aller Modelle, in Bezug auf die Generierung von Anforderungen, wird in Unterabschnitt 6.3.2 eingegangen.

AD-F1 [muss]: Die Anforderungen werden aus einem Kundenanforderungsdokument generiert.

Erfüllt: Die Anforderungen werden aus dem Anforderungsdokument des Kunden generiert (vgl. Abschnitt 5.2). Mit dem Modell *Pixtral 12B* wird das Dokument zudem als Markdown bereitgestellt, um einen geeigneten Kontext für das Generieren der Anforderungen bereitzustellen.

AD-F2 [muss]: Das Dokument hat eine Übersicht, welche die Projektnummer, die Dokumentenbezeichnung, das Erstellungsdatum, die Dokumentennummer und die Version beinhaltet.

Erfüllt: Das Anforderungsdokument besitzt eine Tabelle „Overview“. Die erste Zelle beinhaltet die Informationen zu dem Kundennamen, dem Projektnamen, der Dokumentenbezeichnung, dem Erstellungsdatum, der Dokumentennummer und der Version. Diese wird ohne Hilfe des LLMs erstellt.

AD-F3 [muss]: In der Übersicht gibt es eine Tabelle für das Versionsmanagement,

mit den Spalten: Version, Änderungsdatum, Autor, Änderungen und Bemerkungen

---

Erfüllt: Auf derselben Seite wird, zwei Zeilen unter der Übersicht, eine Versionstabelle angelegt. Dort sind Spalten mit den Überschriften „Version“, „Date of change“, „Author“, „Changes“ und „Notes“ vorhanden. Zudem wir die erste Zeile mit den entsprechenden Informationen gefüllt. Für den Autor wird „Automatically generated“ eingetragen.

AD-F4 [muss]: Die Anforderungen werden eindeutig nummeriert. Die Nummern werden in einer Spalte, mit der Überschrift „No.“ eingetragen.

Erfüllt: Durch den Structed Output gib tdas LLM die Anforderungen in dem vorgegebenen Format zurück. Dabei hat jede Anforderung eine eindeutige Nummer in der Form „Anf1“, „Anf2“, „Anf3“, etc.

AD-F5 [muss]: Für das Kapitel der Anforderungen gibt es eine Spalte „Chapter“. Dort werden die jeweiligen Kapitel des Kundenanforderungsdokuments eingetragen.

Erfüllt: Das LLM gibt die Kapitel, aus welchen die Anforderungen entnommen wurden, zurück. Diese werden anschließend in die Spalte „Chapter“ des Anforderungsdokuments eingetragen (vgl. Unterabschnitt 5.3.1).

AD-F6 [muss]: Die Anforderungen haben eine kurze und messbare Beschreibung, welche in der Spalte „Description“ stehen.

Erfüllt: Die Anforderungen werden in der vierten Spalte „description“ beschrieben (vgl. Unterabschnitt 5.3.1). Die Qualität der Anforderung, mit den verschiedenen Modellen, wird in Unterabschnitt 6.3.2 beurteilt.

AD-F7 [muss]: Es gibt eine leere Spalte für die Kategorisierung, mit der Überschrift „Category“ und eine Spalte für Kommentare, mit der Überschrift „Notes“.

Erfüllt: Die beiden Spalten „Category“ und „Notes“ werden unabhängig vom LLM erstellt (vgl. Unterabschnitt 5.3.1).

AD-F8 [kann]: Die einzelnen Anforderungen werden auf ihre Umsetzbarkeit überprüft und farblich markiert.

Nicht erfüllt: Eine Überprüfung der Machbarkeit der einzelnen Anforderungen wurde nicht implementiert. Die Anforderungen werden lediglich im gewünschten Format aufgelistet und in die Excel-Datei geschrieben.

AD-F9 [muss]: Der Dateiname wird aus der Projektnummer, dem DCC, der Dokumentennummer, dem Status, der Version und der Dokumentenbezeichnung zusammengesetzt. Der Name für das Anforderungsdokument lautet: Projektnummer + „-EC411-W01-RequirementList“.

Erfüllt: Der Dateiname wird entsprechend der Vorgabe aus der Projektnummer, dem DCC, der Dokumentennummer, dem Status, der Version und der Dokumentenbezeichnung zusammengesetzt (vgl. Unterabschnitt 5.3.1). Dazu wurde die Methode **SaveDocument()** implementiert.

AD-F10 [muss]: Es soll kein neuer Inhalt generiert werden. Die Anforderungen werden anhand der Informationen des Anforderungsdokument vom Kunden erstellt.

Erfüllt: Durch das Prompting und Übergeben von spezifischen und kohärenten Anweisungen werden Halluzinationen vorgebeugt. Zusätzlich wird die Temperatur leicht verringert, um die Kreativität des LLMs einzuschränken (Unterabschnitt 4.3.2, 5.3.1).

AD-NF1 [kann]: Das Dokument ist in Form einer Excel-Datei.

Erfüllt: Das Anforderungsdokument ist eine Exceldatei. Dies wird mit Hilfe des Interops *Microsoft.Office.Interop.Excel* realisiert (vgl. Unterabschnitt 5.3.1).

6.3.2 Qualität der Anforderungen

Bevor die Qualität der Anforderungen betrachtet wird und die Textmodelle miteinander verglichen werden, wird auf das Pixtral 12B Modell eingegangen.

Pixtral 12B 2409: Das Pixtral 12B liefert zusammen mit dem über *PdfPig* ausgelesenen Text einer PDF-Datei, eine in vielen Punkten gute Antwort. Drei ausgewählten Antworten können aus dem Anhang (Unterabschnitt A.5.1, A.5.2, A.5.3) entnommen werden. Der Text wird bei den Antworten korrekt wiedergegeben und auch Tabellen, Listen und Aufzählungen werden berücksichtigt. Bei den Tabellen kann es zu Darstellungsfehlern kommen, was in Unterabschnitt A.5.2 zu erkennen ist. Ein weiteres Problem liegt bei der Erkennung und Markierung von Kapiteln und Abschnitten. Diese werden nicht korrekt im Markdown Format angegeben, wodurch die Qualität der Antwort gemindert wird.

Testumgebung: Zur Evaluation werden Auszüge von Dokumenten aus drei verschiedenen Projekten verwendet. Dabei handelt es sich bei zwei Projekten um die Implementierung von STS und bei dem dritten Projekt um RMGs. Die Dokumente haben verschiedene Strukturen, um möglichst viele Fälle abzudecken. Unterabschnitt A.4.2 ist in Tabellenform verfasst worden, während Unterabschnitt A.4.3 bereits in einzelne Anforderungen gegliedert ist. Unterabschnitt A.4.1 ist im Fließtext geschrieben, bei dem die einzelnen Anforderungen nicht direkt ersichtlich sind. Die Antworten der LLMs sind ebenfalls aus dem Anhang (Abschnitt A.5) zu entnehmen.
Es wird jedem Modell der gleiche Prompt, mit demselben Kontext (Anhang Unterabschnitt A.3.1) übergeben. Zudem liefern die Modelle Deepseek R1 und Qwen3 einen Gedankengang (Chain of Thought), welcher jedoch nicht berücksichtigt wird.

Vergleich der Textmodelle: In Tabelle 6.2, 6.3 und 6.4 werden die einzelnen Modelle auf die Qualität der generierten Anforderungen getestet. Dabei wird auf auf die **Regeln** aus Unterabschnitt 4.5.1 zurückgegriffen. Die *Bildung des Passivs* wird nicht in den Tabellen 6.2, 6.3 und 6.4 bewertet, sondern es wird darauf später separat eingegangen.

Zum Vergleich der Modelle werden verschiedene Dokumente getestet, welche verschiedene Formen haben. Einmal sind die Anforderungen als Tabelle aufgeführt (Tabelle 6.2), danach stehen sie im Fließtext (Tabelle 6.3) und zum Schluss sind die Anforderungen bereits einzeln aufgeteilt (Tabelle 6.4). Dabei wird erst die absolute Anzahl der generierten Anforderungen eines Modells aufgezeigt. Danach wird die Anzahl der erfüllten Regeln relativ zur absoluten Anzahl dargestellt. Wichtig zu erwähnen ist, dass nur der Aufbau der Anforderungen an sich bewertet wird. Ergänzungen, wie die Antwort als XML wiederzugeben, werden nicht berücksichtigt. Auch der Inhalt der Anforderungen wird in den Tabellen nicht betrachtet.
Das gewählte Format wird von den getesteten Modellen je nach Quelle unterschiedlich gut eingehalten wird. Die Entnahme aus dem Fließtext schneidet hierbei am besten ab. Am schlechtesten wird das Format bei einem Anforderungsdokument in Tabellenform eingehalten. Es wird bei allen Anforderungsdokumenten von keinem Modell ein Lösungsweg angegeben. Dies hat den Grund, dass die Anforderungsdokumente keine Lösungswege enthalten. Es werden jedoch auch keine eigenen Lösungswege von den Modellen generiert. Die teilweise schlechte Messbarkeit der Anforderungen hängt mit der ersten Regel „eine Anforderung pro Satz“ zusammen. Stehen keine oder mehrere Anforderungen in einem Satz, kann diese auch nicht gemessen werden. Zudem lassen sich Anforderungen, welche nur stichpunktartig geschrieben sind oder bereits im Anforderungsdokument nicht messbar sind, ebenfalls nicht messen.

Vermeiden des Passivs: Ähnlich wie beim Lösungsweg oder der Messbarkeit ist auch das Passiv stark abhängig vom Anforderungsdokument. Werden die Anforderungen dort bereits im Passiv formuliert, werden diese vom LLM nicht ins Aktiv umgeschrieben. Andersherum werden Sätze im Aktiv nicht ins Passiv umgeschrieben.

Bewertung der gesamten Antwort: Die jeweiligen Antworten der Modelle sind im Anhang unter Unterabschnitt A.5.4 bis Unterabschnitt A.5.21 zu entnehmen. Wie die Anforderungen selbst aufgebaut sind, wird bereits in den drei Tabellen evaluiert.
Beim Betrachten der Struktur der Antworten stechen die beiden Modellen *Mistral 7B Instruct* und *Mistral Nemo Instruct 2407* im negativen Sinne heraus. Die Anforderungen werden häufig in einen XML-Block geschrieben und teilweise ist der Gedankengang mit aufgelistet. Neben den Anforderungen wird auch die Nutzeranfrage, in manchen Fällen, vor der eigentlichen Antwort wiederholt. Zusätzlich werden die Kapitelnummern teilweise inkorrekt dargestellt, was in Unterabschnitt A.5.4 und A.5.8 zu sehen ist. Dies macht das

Tabelle 6.2: Vergleich der Modelle beim Generieren von Anforderungen aus einem Kunden Anforderungsdokument in Tabellenform

	Mistral 7B Instruct	Deepseek R1 Distill Qwen 7B	Qwen2.5 Coder 7B Instruct	Llama 3.1 8B Instruct	Mistral Nemo Instruct 2407	Qwen3 30B A3B
Anzahl der generierten Anforderungen	68	34	32	34	46	29
Eine Anforderung pro Satz	100,00 %	91,18 %	53,13 %	100,00 %	100,00 %	100,00 %
Kurze Sätze	95,59 %	91,18 %	50,00 %	100,00 %	100,00 %	96,55 %
Eindeutiges Subjekt	98,53 %	88,24 %	50,00 %	97,06 %	97,83 %	96,55 %
Kein Lösungsweg	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %
Messbar	100,00 %	91,18 %	53,13 %	100,00 %	100,00 %	100,00 %
Einhalten des vorgegebenen Formats	25,00 %	100,00 %	50,00 %	0,00 %	63,04 %	48,28 %

Verarbeiten der Antwort aufwendiger. Neben den formalen Mängeln kommt es bei den beiden Modellen verstärkt zu Halluzinationen, vor allem wenn die Nutzeranfrage keine Anforderungen enthält.

Die Modelle *Deepseek R1 Distill Qwen 7B* und *Llama 3.1 8B Intrsuct* performen insgesamt besser. Es kommt hier ebenfalls vor, dass das Format nicht eingehalten wird, dies bezieht sich meist auf die Anforderungen selbst. Halluzinationen, Gedankengänge oder andere Formfehler treten nicht auf. Jedoch fehlen beim DeepSeek R1 Modell teilweise die Anforderungen selbst (vgl. Unterabschnitt A.5.17) oder es kommt zu Fehlern bei der Information zum Kapitel (vgl. Unterabschnitt A.5.5). Llama 3.1 lässt die Informationen zum Kapitel in Unterabschnitt A.5.13 komplett weg. Auch zu Beginn in Unterabschnitt A.5.7 fehlen diese Angaben.

Das Modell *Qwen2.5 Coder 7B Instruct* stellt ebenfalls die Kapitelnummer teilweise

Tabelle 6.3: Vergleich der Modelle beim Generieren von Anforderungen aus einem Kunden Anforderungsdokument im Fließtext

	Mistral 7B Instruct	Deepseek R1 Distill Qwen 7B	Qwen2.5 Coder 7B Instruct	Llama 3.1 8B Instruct	Mistral Nemo Instruct 2407	Qwen3 30B A3B
Anzahl der Anforderungen	33	33	18	20	51	17
Eine Anforderung pro Satz	57,58 %	54,55 %	88,89 %	95,00 %	62,75 %	88,24 %
Kurze Sätze	51,52 %	54,55 %	88,89 %	95,00 %	74,51 %	88,24 %
Eindeutiges Subjekt	57,58 %	54,55 %	100,00 %	100,00 %	62,75 %	100,00 %
Kein Lösungsweg	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %
Messbar	57,58 %	54,55 %	100,00 %	100,00 %	62,75 %	100,00 %
Einhalten des vorgegebenen Formats	100,00 %	96,97 %	88,89 %	100,00 %	100,00 %	76,47 %

inkorrekt dar (vgl. Unterabschnitt A.5.12). Bei derselben Antwort werden die Anforderungen zudem teilweise stichpunktartig und nicht in vollständigen Sätzen geschrieben. In Unterabschnitt A.5.6 und A.5.18 werden Aufzählungen zu einer Anforderung kombiniert, anstatt diese in einzelne Anforderungen aufzuteilen. Dabei treten keine Halluzinationen, Gedankengänge oder andere Formfehler auf.
Qwen3 30B A3B performt besser als das Qwen2.5 Modell. Es kommt auch hier teilweise zu Fehlern bei der Kapitelnummer (vgl. Unterabschnitt A.5.15 und Listen werden in einer Anforderung kombiniert (vgl. Unterabschnitt A.5.21).

Zusammenfassung: Bei der Evaluation der Qualität der Anforderungen fällt auf, dass sich die generierten Anforderungen an das Anforderungsdokument halten. Dies ist in gewisser Weise gewollt, da das LLM keinen Inhalt generieren soll und auch die Bedeutung

Tabelle 6.4: Vergleich der Modelle beim Generieren von Anforderungen aus einem Kunden Anforderungsdokument in Anforderungsform

	Mistral 7B Instruct	Deepseek R1 Distill Qwen 7B	Qwen2.5 Coder 7B Instruct	Llama 3.1 8B Instruct	Mistral Nemo Instruct 2407	Qwen3 30B A3B
Anzahl der Anforderungen	30	28	25	38	26	32
Eine Anforderung pro Satz	96,67 %	64,29 %	96,00 %	97,37 %	96,15 %	96,88 %
Kurze Sätze	96,67 %	64,29 %	96,00 %	97,37 %	96,15 %	96,88 %
Eindeutiges Subjekt	96,67 %	64,29 %	96,00 %	100,00 %	100,00 %	96,88 %
Kein Lösungsweg	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %	100,00 %
Messbar	100,00 %	67,86 %	100,00 %	97,37 %	100,00 %	96,88 %
Einhalten des vorgegebenen Formats	96,67 %	32,14 %	96,00 %	100,00 %	96,15 %	96,88 %

der Anforderungen nicht verändern soll. Dies führt jedoch dazu, dass bereits qualitativ schlechte Anforderungen aus dem Dokument schlecht bleiben, während bereist qualitativ gute Anforderungen gut bleiben.
Es stellt sich die Frage, ob ein LLM zum Generieren der Anforderungen benötigt wird, wenn diese stark den ursprünglichen ähneln. Es hat sich gezeigt, dass die Modelle lange Anforderungen in mehrere Sätze oder auf mehrere Anforderungen aufteilen. Dies erhöht die Lesbarkeit und Verständlichkeit der Anforderungen. Zudem werden Anforderungen aus einem Fleißtext in eine Listenform überführt, was ebenfalls die Lesbarkeit und Verständlichkeit erhöht.

6.4 Technische Spezifikationen und Testfälle

Wie bereits in Abschnitt 3.5 und 6.1 beschrieben, ist er Schwerpunkt dieser Arbeit die Generierung von Anforderungsdokumenten, sowie der Vergleich verschiedener LLMs. Es wurde lediglich in Unterabschnitt 4.5.2 und 4.5.3 ein grobes Konzept zur Generierung der technischen Spezifikationen und Testfälle erstellt.
Alle Anforderungen aus Unterabschnitt 3.4.3 und 3.4.4 sind **nicht erfüllt**.

7 Fazit und Ausblick

Das Ziel dieser Arbeit bestand darin, eine Anwendung zu entwickeln, die Personen beim Lesen von Kundenanforderungsdokumenten unterstützt und die Erstellung eigener Anforderungsdokumente erleichtert. Zudem sollten Kenntnisse im Umgang mit LLMs erworben werden. Hierzu wurden verschiedene Modelle verwendet, die aus den Kundenanforderungsdokumenten eigene Anforderungsdokumente generieren. Es wurde eine Anwendung konzipiert, die Kundenanforderungsdokumente einliest und die darin enthaltenen Informationen als Kontext für ein LLM bereitstellt. Das LLM formatiert diese Informationen anschließend als einzelne Anforderungen, versehen mit einer Nummer, dem Kapitel, aus dem sie entnommen wurden, und einer Beschreibung. Dabei zeigte sich, dass die verschiedenen Modelle die gestellte Aufgabe gut erfüllen. Besonders wichtig war dabei ein vollständiger und konsistenter Kontext. Herausstechend waren die Modelle *Llama 3.1 Instruct* und *Qwen3 30B A3B*. Auch wenn sie nicht immer perfekte Antworten lieferten, erzielten sie dennoch die besten Ergebnisse für alle Anforderungstypen und produzierten ein verarbeitbares Format, das letztlich in das Anforderungsdokument integriert werden konnte. Im Gegensatz dazu neigten die Modelle Mistral 7B Instruct und Mistral Nemo 2407 vermehrt zu Halluzinationen und hielten das allgemeine Format nicht ein.

Neben der automatischen Generierung der Anforderungsdokumente wurde ebenfalls eine Schnittstelle zu den LLMs implementiert, die in zukünftigen Projekten wiederverwendet werden kann. Beispielsweise besteht bereits die Idee, SPS-Variablen automatisiert aus einem Stromlaufplan mithilfe eines LLMs zu generieren. Hierfür kann die implementierte Schnittstelle genutzt werden.
Auch das Anforderungsmanagement-Tool Polarion könnte zukünftig eingebunden werden. Das Konzept zum Kopieren von Dokumenten sowie das Prinzip, die Antwort in Polarion hochzuladen, wurden kurz erläutert. Hierfür ist jedoch noch weiterführende Arbeit erforderlich, da die Funktionen bislang nicht getestet werden konnten. Zudem verwendet Polarion andere Dokumententypen, was insbesondere die Erstellung und das Hochladen

von Dokumenten herausfordernd gestaltet.
Zusätzlich kann die Generierung zweier weiterer Dokumentarten realisiert werden: der technischen Spezifikationen und der Testfälle. Diese Erweiterung konnte ebenfalls aufgrund zeitlicher Beschränkungen und des begrenzten Umfangs der vorliegenden Arbeit nicht umgesetzt werden. Dennoch sind die Struktur und das Konzept für einen geeigneten Kontext sowie die Klassen und Methoden zur Erstellung dieser Dokumente bereits dokumentiert. Dies bietet eine solide Grundlage, auf der zukünftige Entwicklungen aufbauen können, um die Implementierung der beiden zusätzlichen Funktionen zu erleichtern.

Abschließend wird die eigentliche Arbeit reflektiert. Sie bietet erste Grundlagen zur Generierung von Anforderungsdokumenten, jedoch bestehen noch Verbesserungspotenziale. Die Qualität der Antworten der LLMs könnte beispielsweise durch die Übergabe eines individuellen Kontexts für jedes Modell verbessert werden. So kann auf spezifische Schwachstellen der einzelnen Modelle eingegangen werden. Des Weiteren könnte das Fine-Tuning optimiert werden, indem die Antworten hinsichtlich verschiedener Temperaturen evaluiert, die maximalen Token begrenzt und der Top_p-Wert angepasst wird. Obwohl in der Arbeit kurz auf die Temperatur eingegangen wurde, geschah dies nicht im Detail. Es hat sich jedoch gezeigt, dass bei Temperaturen unter 1 die Antworten qualitativ hochwertiger sind. Für das Extrahieren des Textes einer PDF-Datei mit dem Modell *Pixtral 12B 2409* führt eine Temperatur nahe 0 zu einer verbesserten Antwort. Eine genaue Analyse der Temperatur könnte die Antworten der LLMs ebenfalls verbessern.

Zusammenfassend kann festgehalten werden, dass die Arbeit einen ersten Schritt in Richtung der automatisierten Generierung von Projektmanagementdokumenten darstellt. Sie bietet eine deutliche Zeitersparnis bei der Erstellung von Anforderungsdokumenten, insbesondere, wenn diese als Fließtext vorliegen. Auch die Kompetenz im Bereich LLMs wurde erweitert, um in Zukunft weitere Projekte mithilfe von LLMs realisieren zu können. Zudem liefert die Arbeit selbst eine Grundlage, auf der Projekte unter Verwendung von LLMs aufgebaut werden können.

Literaturverzeichnis

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, et al. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *EMNLP 2023*, 2023. URL https://arxiv.org/abs/2305.13245. Zugriffsdatum: 23.05.2025.

[2] Ethem Alpaydin. *Maschinelles Lernen 2*. De Gruyter, Berlin, 2019.

[3] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. 2020. URL https://arxiv.org/abs/2004.05150. Zugriffsdatum: 24.05.2025.

[4] Oswald Campesato. *Large Language Models : An Introduction*. De Gruyter, Dulles, VA, 2024.

[5] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. 2025. URL https://github.com/deepseek-ai/DeepSeek-R1/blob/main/DeepSeek_R1.pdf. Zugriffsdatum: 27.05.2025.

[6] Zichuan Fu, Wentao Song, Yejing Wang, et al. Sliding window attention training for efficient large language models. 2025. URL https://arxiv.org/abs/2502.18845. Zugriffsdatum: 23.05.2025.

[7] Marcus Grande. *100 Minuten für Anforderungsmanagement: Kompaktes Wissen nicht nur für Projektleiter und Entwickler*. Springer Vieweg, Wiesbaden, 2014.

[8] Dan Hendrycks, Collin Burns, Steven Basart, et al. Measuring massive multitask language understanding. *ICLR 2021*, 2021. URL https://arxiv.org/abs/2009.03300. Zugriffsdatum: 21.05.2025.

[9] Jochen Hirschle. *Deep Natural Language Processing Einstieg in Word Embedding, Sequence-to-Sequence-Modelle und Transformer mit Python*. Carl Hanser Verlag, München, 2022.

[10] Uday Kamath, Kevin Keenan, Garrett Somers, and Sarah Sorenson. *Large Language Models: a Deep Dive : Bridging Theory and Practice*. Springer, Cham, 2024.

[11] Jon Krohn, Grant Beyleveld, Aglaé Bassens, and Kathrin Lichtenberg. *Deep Learning illustriert Deep Learning illustriert: eine anschauliche Einführung in Machine Vision, Natural Language Processing und Bilderzeugung für Programmierer und Datenanalysten*. dpunkt.verlag, München, 2020.

[12] Thomas Niebisch and Jens Kawelke. *Anforderungsmanagement in sieben Tagen: Requirements Engineering im Zeitalter der KI*. Springer Gabler, Berlin, Heidelberg, 2024.

[13] OpenAI. Openai api reference: Text generation and prompting, 2024. URL https://platform.openai.com/docs/guides/text?api-mode=responses. Zugriffsdatum: 18.04.2025.

[14] OpenAI. Openai api reference: Structured outputs, 2024. URL https://platform.openai.com/docs/guides/structured-outputs?api-mode=responses. Zugriffsdatum: 22.05.2025.

[15] Siemens. Models, 2024. URL https://code.siemens.io/ai/models/. Zugriffsdatum: 25.03.2024.

[16] Bhawna Singh. *Building Applications with Large Language Models: Techniques, Implementation, and Applications*. APress, New York, 2024.

[17] Hans-Peter Stricker. *Sprachmodelle verstehen: Chatbots und generative künstliche Intelligenz im Zusammenhang*. Springer, Berlin, 2024.

[18] Mehrzad Tabatabaian. *Prompt engineering using ChatGPT : crafting effective interactions and building GPT apps*. De Gruyter, Boston, Massachusetts, 2024.

[19] Qwen Team. Qwen3 technical report. 2025. URL https://arxiv.org/abs/2505.09388. Zugriffsdatum: 27.05.2025.

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. *NIPS 2017*, 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. Zugriffsdatum: 31.03.2025.

[21] Rowan Zellers, Ari Holtzman, Yonatan Bisk, et al. Hellaswag: Can a machine really finish your sentence? *ACL 2019*, 2019. URL https://arxiv.org/abs/1905.07830. Zugriffsdatum: 22.05.2025.

[22] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *NeurIPS 2023*, 2023. URL https://arxiv.org/abs/2306.05685. Zugriffsdatum: 14.05.2025.

A Anhang

A.1 Dokumentenbezeichnung

A.1.1 Document Classification Code

Extract from IEC 61355-1

Dokument (DE)	DCC	document (EN)
Dokumente für technische Anforderungen und Auslegung	**E..**	**Technical requirement and dimensioning documents**
Normen und Richtlinien	EB	Standards and regulations
techn. Spezifikations-/Anforderungsdokumente	EC	Technical specification / requirement documents
Dimensionierung / Berechnungen	ED	Dimensioning documents

Abbildung A.1: Vorgaben zum Festlegen des DCCs

A.1.2 Dateinamen

Description	DCC	Doc. Nr	Status	Version	Project Nr	Document Name
General Requirement List - Generated automatically	EC	411	W	01	24HB-MC-02-001	24HB-MC-02-001-EC411-W01-RequirementList

Abbildung A.2: Vorgaben zum Dateinamen für Dokumente

A.2 Benchmarks

A.2.1 MT-Bench

Model	Writing	Roleplay	Reasoning	Math	Coding	Extraction	STEM	Humanities
GPT-4	61.2%	67.9%	49.3%	66.1%	56.3%	66.2%	76.6%	72.2%
GPT-3.5	50.9%	60.6%	32.6%	63.8%	55.0%	48.8%	52.8%	53.8%
Vicuna-13B	39.7%	39.2%	20.1%	18.0%	36.9%	29.2%	47.0%	47.5%
LLaMA-13B	15.1%	15.1%	7.8%	7.5%	2.1%	9.3%	6.8%	10.1%

Abbildung A.3: Kategorisierte Genauigkeit verschiedener Modelle im MT-Bench Benchmark [22]

Model	#Training Token	MMLU (5-shot)	TruthfulQA (0-shot)	MT-Bench Score (GPT-4)
LLaMA-7B	1T	35.2	0.22	2.74
LLaMA-13B	1T	47.0	0.26	2.61
Alpaca-7B	4.4M	40.1	0.26	4.54
Alpaca-13B	4.4M	48.1	0.30	4.53
Vicuna-7B (selected)	4.8M	37.3	0.32	5.95
Vicuna-7B (single)	184M	44.1	0.30	6.04
Vicuna-7B (all)	370M	47.1	0.32	6.00
Vicuna-13B (all)	370M	**52.1**	**0.35**	**6.39**
GPT-3.5	-	70.0	-	7.94
GPT-4	-	**86.4**	-	**8.99**

Abbildung A.4: Ergebnisse verschiedener Modelle im MT-Bench Benchmark [22]

A.2.2 MMLU

Model	Humanities	Social Science	STEM	Other	Average
Random Baseline	25.0	25.0	25.0	25.0	25.0
RoBERTa	27.9	28.8	27.0	27.7	27.9
ALBERT	27.2	25.7	27.7	27.9	27.1
GPT-2	32.8	33.3	30.2	33.1	32.4
UnifiedQA	45.6	56.6	40.2	54.6	48.9
GPT-3 Small (few-shot)	24.4	30.9	26.0	24.1	25.9
GPT-3 Medium (few-shot)	26.1	21.6	25.6	25.5	24.9
GPT-3 Large (few-shot)	27.1	25.6	24.3	26.5	26.0
GPT-3 X-Large (few-shot)	40.8	50.4	36.7	48.8	43.9

Abbildung A.5: Kategorisierte Genauigkeit verschiedener Modelle im MMLU Benchmark [8]

A.2.3 HellaSwag

	Overall		In-Domain		Zero-Shot		ActivityNet		WikiHow	
Model	Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
Split Size→	10K	10K	5K	5K	5K	5K	3.2K	3.5K	6.8K	6.5K
Chance					25.0					
fastText	30.9	31.6	33.8	32.9	28.0	30.2	27.7	28.4	32.4	33.3
LSTM+GloVe	31.9	31.7	34.3	32.9	29.5	30.4	34.3	33.8	30.7	30.5
LSTM+ELMo	31.7	31.4	33.2	32.8	30.4	30.0	33.8	33.3	30.8	30.4
LSTM+BERT-Base	35.9	36.2	38.7	38.2	33.2	34.1	40.5	40.5	33.7	33.8
ESIM+ELMo	33.6	33.3	35.7	34.2	31.5	32.3	37.7	36.6	31.6	31.5
OpenAI GPT	41.9	41.7	45.3	44.0	38.6	39.3	46.4	43.8	39.8	40.5
BERT-Base	39.5	40.5	42.9	42.8	36.1	38.3	48.9	45.7	34.9	37.7
BERT-Large	**46.7**	**47.3**	**50.2**	**49.7**	**43.3**	**45.0**	**54.7**	**51.7**	**42.9**	**45.0**
Human	95.7	95.6	95.6	95.6	95.8	95.7	94.0	94.0	96.5	96.5

Abbildung A.6: Ergebnisse verschiedener Modelle im HellaSwag Benchmark [21]

A.3 Prompts

A.3.1 System Prompt zur Generierung der Anforderungen

CD Pfad: „/Anhang/Prompts/RequirementDocSystemMsg.md“

A.3.2 User Prompt zum Extrahieren des Texts eines PDFs

CD Pfad: „/Anhang/Prompts/ImageContentUser.md“

A.4 Anforderungsdokumente

A.4.1 Anforderungsdokument Auszug - Fließtext

CD Pfad: „/Anhang/Anforderungsdokumente/Anforderungsdokument_Fließtext_Auszug_geschwärzt.pdf“

A.4.2 Anforderungsdokument Auszug - Tabellenform

CD Pfad: „/Anhang/Anforderungsdokumente/Anforderungsdokument_Tabelle_Auszug_-geschwärzt.pdf“

A.4.3 Anforderungsdokument Auszug - Anforderungsliste

CD Pfad: „/Anhang/Anforderungsdokumente/Anforderungsdokument_Anforderungen_-Auszug_geschwärzt.pdf“

A.5 LLM Antworten

A.5.1 Anforderungsdokument Fließtext - Pixtral 12B 2409

CD Pfad: „/Anhang/LLMAntworten/Pixtral/Fließtext_PdfContent.txt“

A.5.2 Anforderungsdokument Tabellenform - Pixtral 12B 2409

CD Pfad: „/Anhang/LLMAntworten/Pixtral/Tabelle_PdfContent.txt“

A.5.3 Anforderungsdokument Anforderungsliste - Pixtral 12B 2409

CD Pfad: „/Anhang/LLMAntworten/Pixtral/Anforderungsliste_PdfContent.txt“

A.5.4 Anforderungsdokument Fließtext - Mistral 7B Instruct

CD Pfad: „/Anhang/LLMAntworten/Mistral/Fließtext_MistralResponse.txt“

A.5.5 Anforderungsdokument Fließtext - Deepseek R1 Distill Qwen 7B

CD Pfad: „/Anhang/LLMAntworten/DeepSeek/Fließtext_DeepSeekResponse.txt“

A.5.6 Anforderungsdokument Fließtext - Qwen2.5 Coder 7B Instruct

CD Pfad: „/Anhang/LLMAntworten/Qwen25/Fließtext_Qwen25Response.txt“

A.5.7 Anforderungsdokument Fließtext - Llama 3.1 8B Instruct

CD Pfad: „/Anhang/LLMAntworten/Llama/Fließtext_LlamaResponse.txt“

A.5.8 Anforderungsdokument Fließtext - Mistral Nemo Instruct 2407

CD Pfad: „/Anhang/LLMAntworten/MistralNemo/Fließtext_MistralNemoResponse.txt“

A.5.9 Anforderungsdokument Fließtext - Qwen3 30B A3B

CD Pfad: „/Anhang/LLMAntworten/Qwen3/Fließtext_Qwen3Response.txt“

A.5.10 Anforderungsdokument Tabellenform - Mistral 7B Instruct

CD Pfad: „/Anhang/LLMAntworten/Mistral/Tabelle_MistralResponse.txt“

A.5.11 Anforderungsdokument Tabellenform - Deepseek R1 Distill Qwen 7B

CD Pfad: „/Anhang/LLMAntworten/DeepSeek/Tabelle_DeepSeekResponse.txt“

A.5.12 Anforderungsdokument Tabellenform - Qwen2.5 Coder 7B Instruct

CD Pfad: „/Anhang/LLMAntworten/Qwen25/Tabelle_Qwen25Response.txt“

A.5.13 Anforderungsdokument Tabellenform - Llama 3.1 8B Instruct

CD Pfad: „/Anhang/LLMAntworten/Llama/Tabelle_LlamaResponse.txt“

A.5.14 Anforderungsdokument Tabellenform - Mistral Nemo Instruct 2407

CD Pfad: „/Anhang/LLMAntworten/MistralNemo/Tabelle_MistralNemoResponse.txt“

A.5.15 Anforderungsdokument Tabellenform - Qwen3 30B A3B

CD Pfad: „/Anhang/LLMAntworten/Qwen3/Tabelle_Qwen3Response.txt“

A.5.16 Anforderungsdokument Anforderungsliste - Mistral 7B Instruct

CD Pfad: „/Anhang/LLMAntworten/Mistral/Anforderungsliste_MistralResponse.txt“

A.5.17 Anforderungsdokument Anforderungsliste - Deepseek R1 Distill Qwen 7B

CD Pfad: „/Anhang/LLMAntworten/DeepSeek/Anforderungsliste_DeepSeekResponse.txt“

A.5.18 Anforderungsdokument Anforderungsliste - Qwen2.5 Coder 7B Instruct

CD Pfad: „/Anhang/LLMAntworten/Qwen25/Anforderungsliste_Qwen25Response.txt“

A.5.19 Anforderungsdokument Anforderungsliste - Llama 3.1 8B Instruct

CD Pfad: „/Anhang/LLMAntworten/Llama/Anforderungsliste_LlamaResponse.txt“

A.5.20 Anforderungsdokument Anforderungsliste - Mistral Nemo Instruct 2407

CD Pfad: „/Anhang/LLMAntworten/MistralNemo/Anforderungsliste_MistralNemoResponse.txt“

A.5.21 Anforderungsdokument Anforderungsliste - Qwen3 30B A3B

CD Pfad: „/Anhang/LLMAntworten/Qwen3/Anforderungsliste_Qwen3Response.txt“

A.6 Generierte Anforderungsdokumente

A.6.1 Generiertes Anforderungsdokument aus Fließtext

CD Pfad: „Anhang/LLMAntworten/Qwen3/Fließtext-EC411-W01-RequirementList.xlsx“

A.6.2 Generiertes Anforderungsdokument aus Tabelle

CD Pfad: „Anhang/LLMAntworten/Qwen3/Tabelle-EC411-W01-RequirementList.xlsx“

A.6.3 Generiertes Anforderungsdokument aus Anforderungsliste

CD Pfad: „Anhang/LLMAntworten/Qwen3/Anforderungsliste-EC411-W01-RequirementList.xlsx“

A.7 Quellcode der Anwendung

CD Pfad: „/Documentgenerator“

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort	Datum	Unterschrift im Original