

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Natalie Stricker

**Entwurf und Implementierung einer Webanwendung zur
automatisierten Erstellung von TechReports für
Abschlussarbeiten**

Design and Implementation of a Web Application for
Automated Generation of TechReports for Theses

Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Studiengang Medieninformatik

Bachelorarbeit

von

Natalie Stricker

**Entwurf und Implementierung einer Webanwendung zur
automatisierten Erstellung von TechReports für
Abschlussarbeiten**

Design and Implementation of a Web Application for
Automated Generation of TechReports for Theses

Bearbeitungszeitraum: von 05. Juli 2024
bis 04. Dezember 2024

1. Prüfer: Prof. Dr.-Ing. Christoph P. Neumann

2. Prüfer: Prof. Dr. Dieter Meiler

Selbstständigkeitserklärung

Name und Vorname
der Studentin/des Studenten: **Stricker, Natalie**

Studiengang: **Medieninformatik**

Ich bestätige, dass ich die Bachelorarbeit mit dem Titel:

**Entwurf und Implementierung einer Webanwendung zur automatisierten
Erstellung von TechReports für Abschlussarbeiten**

selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine
anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und
sinngemäße Zitate als solche gekennzeichnet habe.

Datum: 27. November 2024

Unterschrift:

Bachelorarbeit Zusammenfassung

Studentin/Student (Name, Vorname):	Stricker, Natalie
Studiengang:	Medieninformatik
Aufgabensteller, Professor:	Prof. Dr.-Ing. Christoph P. Neumann
Durchgeführt in (Hochschule):	OTH Amberg-Weiden
Ausgabedatum: 05. Juli 2024	Abgabedatum: 04. Dezember 2024

Titel:

**Entwurf und Implementierung einer Webanwendung zur automatisierten
Erstellung von TechReports für Abschlussarbeiten**

Zusammenfassung:

Die Verwendung von Large Language Models gewinnt zunehmend an Bedeutung, da im Internet mit großen Datenmengen gearbeitet werden muss. Diese Modelle helfen dem Benutzer, das Wesentliche herauszufiltern und somit Zeit zu sparen. Gerade im akademischen Bereich sehen sich Lehrende und Studierende mit einer wachsenden Menge an wissenschaftlichen Arbeiten konfrontiert, die gelesen und analysiert werden müssen. Diese Arbeit beschäftigt sich daher mit dem Entwurf und der Implementierung einer Webanwendung, die technische Berichte generiert. Dabei werden auch Formatierungen wie IEEEtran berücksichtigt.

Das Ziel dieser Arbeit ist die Entwicklung eines webbasierten Berichtsgenerators namens Reforge, der in der Lage ist, eingereichte Abschlussarbeiten zu analysieren, zusammenzufassen und in spezifischen Formaten auszugeben. Die entwickelte Anwendung soll Studierenden und Lehrenden helfen, den Prozess der Erstellung von technischen Berichten zu automatisieren und zu vereinfachen. Es wird untersucht, welche Erkenntnisse aus der Entwicklung und dem Einsatz dieser Anwendung gewonnen werden können und ob der Einsatz eines solchen Generators sinnvoll ist. Der Fokus liegt dabei auf der Qualität der generierten Berichte und dem Design der Anwendung.

Schlüsselwörter: LLM, KI, React, TypeScript, Web-Entwicklung, API

Title:

**Design and Implementation of a Web Application for Automated Generation of
TechReports for Theses**

Abstract:

The use of large language models is becoming increasingly important as large amounts of data have to be processed on the Internet. These models help the user to filter out the essentials and thus save time. Particularly in the academic field, teachers and students are confronted with a growing volume of scientific papers that need to be read and analysed. This thesis therefore deals with the design and implementation of a web application that generates technical reports. Formatting such as IEEEtran is also taken into account.

The aim of this thesis is to develop a web-based report generator called Reforge, which is able to analyse, summarise and output submitted theses in specific formats. The developed application should help students and teachers to automate and simplify the process of creating technical reports. It will be analysed which insights can be gained from the development and use of this application and whether the use of such a generator makes sense. The focus is on the quality of the generated reports and the design of the application.

Keywords: LLM, KI, React, TypeScript, Web-Entwicklung, API

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	2
1.2	Ziel der Arbeit	2
1.3	Aufbau der Arbeit	3
2	Grundlagen	4
2.1	Theoretische Grundlagen	4
2.1.1	Large-Language-Model Grundlagen	4
2.1.2	Reguläre Ausdrücke	5
2.1.3	Web-Design Grundlagen	7
2.2	Software-Bausteine	8
2.2.1	Typescript	8
2.2.2	React	8
2.2.3	API-Schnittstelle	9
3	Forschungsstand und verwandte Arbeiten	10
3.1	Forschungsstand	10
3.2	Verwandte Webanwendungen	12
4	Konzeption und Entwurf von Reforge	14
4.1	Anforderungsanalyse	14
4.1.1	Funktionale Anforderungen	14
4.1.2	Nicht-Funktionale Anforderungen	16
4.2	Wireframes	17
4.3	Auswahl des LLMs für die Textzusammenfassung	18
4.4	Systemarchitektur	20
4.4.1	Speicherung von Daten	21
4.4.2	Verlauf des Uploads in Reforge	21
5	Implementierung von Reforge	23
5.1	Entwicklung des Backends	23
5.1.1	Kommunikation zwischen Frontend und Backend	23
5.1.2	Verwendete Backend Bibliotheken	25
5.1.3	LaTeX-ZIP-Verarbeitung	26
5.1.4	Spracherkennung mittels Stopwörtern	31

5.1.5	OpenAI und DeepL	33
5.1.6	Export-Formatierungen	37
5.1.7	DOCX-Verarbeitung	37
5.2	Entwicklung des Frontends	37
5.2.1	Verwendete Frontend Bibliotheken	38
5.2.2	Finales Frontend-Design	39
5.2.3	Die Upload-Seite	41
5.2.4	Die Download-Seite	42
6	Evaluation und Ergebnisse	44
6.1	Konfigurationsmanagement und Inbetriebnahme	44
6.2	Kosten der API-Nutzung	46
6.3	Qualität der generierten Berichte	48
6.4	Evaluation der mobilen Ansicht von <i>Reforge</i>	50
6.5	Projektvergleich mit verwandten Webanwendungen	50
6.5.1	Vergleich mit ChatGPT-4	50
6.5.2	Vergleich mit Copilot	53
6.5.3	Vergleich mit Gemini	53
6.5.4	Vergleich mit QuillBot	54
6.6	Lessons Learned	55
7	Projektausblick von Reforge	57
7.1	Verbesserung der Textbereinigungen	57
7.2	Hinzufügen des Literaturverzeichnisses	58
7.3	Hinzufügen einer Datenbank	58
7.4	Berücksichtigung von Bildern	58
7.5	Erweiterung der Parameter im Frontend	59
7.6	Asynchrone Verarbeitungen und Benachrichtigungen	59
7.7	Kostenlimitierung	59
8	Fazit	60
8.1	Zusammenfassung der Arbeit	60
8.2	Erreichung der Ziele	60
	Literaturverzeichnis	62
	Abbildungsverzeichnis	66
	Tabellenverzeichnis	67
A	Bilder	68
A.1	Creditkauf auf Fotor	68
A.2	Reforge LaTeX-Ausgabetest	69
A.3	Reforge DOCX-Ausgabetest	71
A.4	ChatGPT Test 2	73
A.5	ChatGPT Test 3	75
A.6	ChatGPT Test 4	77

B	Codeauszüge	78
B.1	Text-Filterungen	78
B.2	DOCX-Dokument Generierung	80

Abkürzungsverzeichnis

KI Künstliche Intelligenz

GPT Generative Pre-trained Transformer

UML Unified Modelling Language

LLM Large Language Model

DOCX Office Open XML Document

PDF Portable Document Format

UA Universal Accessibility

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

URI Uniform Resource Identifier

URL Uniform Resource Locator

UML Unified Modeling Language

DOM Document Object Model

API Application Programming Interface

CORS Cross-Origin Resource Sharing

Blob Binary Large Object

SPA Single Page Application

UI User Interface

Kapitel 1

Einleitung

Die Künstliche Intelligenz (KI) hat in den letzten Jahren immer mehr an Bedeutung gewonnen und wird auch in Zukunft eine wichtige Rolle spielen. Sie wird in den unterschiedlichsten Lebensbereichen eingesetzt. Von personalisierten Produktempfehlungen in Online-Shops über autonome Fahrzeuge bis hin zur Diagnose von Krankheiten. In Arizona gibt es zum Beispiel einen autonomen Taxi-Service, der ohne einen Menschen hinter dem Lenkrad fährt. Es wird sogar gesagt, dass die Fahrweise der KI sicherer sei als die eines Menschen. [Gibbs, 2017]

Dies wirft bei einigen Menschen Bedenken auf, da Sie fürchten, dass die KI den Menschen in verschiedenen Bereichen wie auch den traditionellen Taxifahrer überflüssig machen. Federspiel hingegen betont in seinem Artikel den Nutzen der Ersetzung von repetitiver Arbeit, da sie dem Menschen dabei hilft, sich auf andere Bereiche zu fokussieren. [Federspiel et al., 2023]

Im Bildungsbereich bietet die KI erhebliche Potenziale. Sie kann Lehrkräfte unterstützen, indem sie individuelles Feedback für Schülerinnen und Schüler generiert und maßgeschneiderte Lernpläne erstellt. Diese können Lehrern als wertvolle Empfehlungen für den Unterricht dienen und den Schülern helfen, gezielt an ihren Schwächen zu arbeiten. [Fraivan and Khasawneh, 2023, S.3 ff.]

Trotzdem ist es wichtig, KI immer als Hilfsmittel zu betrachten. Die Ergebnisse müssen sorgfältig geprüft werden, da sie Fehlinformationen enthalten können und die Antworten nicht immer konsistent sind. Der Einsatz von KI, speziell von Sprachmodellen und Chatbots, kann zu einer potenziellen Abhängigkeit führen. Es besteht die Möglichkeit, dass Forscher, Lehrer und Studenten ihre eigenen kritischen Denkfähigkeiten vernachlässigen. Dadurch können in ihrer Arbeit Fehler und Ungenauigkeiten auftreten. [Fraivan and Khasawneh, 2023, S.10 ff.]

Darüber hinaus besteht das Problem des sogenannten „Bias“ in KI-Systemen. Ein Bias ist eine fehlerhafte Neigung zu bestimmten Faktoren in einer Studie. Da die von Chatbots generierten Antworten auf den ihnen zur Verfügung stehenden Datensätzen basieren, können in diesen Datensätzen Fehler, Verzerrungen oder Biases auftreten. Dies führt dazu, dass die Ergebnisse nicht der Realität entsprechen [Anslinger, 2021,

S.3 ff.]. Ob diese Bias-Verzerrungen auch auftreten, wenn eigene Textdatensätze wie zum Beispiel eine Bachelorarbeit vorgegeben werden, ist noch offen. Es stellt sich auch die Frage, ob die KI, ähnlich wie beim autonomen Fahren, bessere Ergebnisse bei der Textzusammenfassung liefern kann als ein Mensch.

1.1 Motivation

Die zunehmende Verbreitung von KI in verschiedenen Branchen und Anwendungsbereichen weckt nicht nur Interesse, sondern macht es auch notwendig, ihre Möglichkeiten voll auszuschöpfen. Unternehmen, Forschungsinstitute und Bildungseinrichtungen müssen sich mit der Verarbeitung von riesigen Datenmengen auseinandersetzen. Hier kann KI helfen, Daten zu analysieren, Muster zu erkennen und Vorhersagen zu treffen. Um diese Potenziale optimal nutzen zu können, müssen jedoch zunächst Systeme entwickelt werden, die auf die jeweiligen Anforderungen zugeschnitten sind.

Ein besonders relevantes Anwendungsfeld für KI ist die Automatisierung repetitiver und zeitintensiver Aufgaben. In der Wissenschaft kann es aufwändig sein, Texte zu erstellen und zu bearbeiten. Forscher verbringen viel Zeit damit, Texte zu schreiben, zu formatieren und zu lesen. Diese Zeit ist besser investiert, wenn man sie für kreative und innovative Aufgaben nutzt. Es wäre daher sinnvoll, eine Anwendung zu entwickeln, die derartige wiederkehrende Aufgaben übernimmt.

Darüber hinaus können vorhandene Erfahrungen in der Entwicklung von Webanwendungen genutzt und erweitert werden. Vorerfahrungen aus früheren Projekten wie *BattleSprout*, bieten eine Basis [Cyberlytics et al., 2023]. Das dort erworbene Wissen über die Gestaltung von Webanwendungen kann direkt in die Entwicklung neuer Webanwendungen einfließen.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist die Entwicklung eines webbasierten Bericht Generators namens *Reforge*. Dieser soll in der Lage sein, eingereichte Abschlussarbeiten zusammenzufassen und in spezifischen Formaten auszugeben. Die entwickelte Anwendung soll zudem den Studierenden und Dozenten dabei helfen, den Vorgang der Erstellung von technischen Berichten zu automatisieren und zu vereinfachen. Darüber hinaus wird bei den Endprodukten geprüft, inwieweit die originalen Input-Dateien im Vergleich zu den generierten Berichten verändert wurden, um die Genauigkeit der KI-generierten Inhalte zu überprüfen.

Im Rahmen der Bachelorarbeit wird untersucht, welche Erkenntnisse aus der Entwicklung und dem Einsatz dieser Anwendung gewonnen werden können und ob der Einsatz eines solchen Generators sinnvoll ist. Der Fokus liegt dabei auf der Qualität der erzeugten Berichte und der Konzeption der Anwendung.

Folgende Fragen sollen beantwortet werden:

- Welche Architektur ist geeignet für die Entwicklung einer Webanwendung, die die automatische Erstellung von TechReports unterstützt?
- Wie kann ein Large Language Model (LLM) zur automatischen Zusammenfassung in die Webanwendung integriert werden?
- Wie lässt sich die Qualität der automatisch generierten Berichte im Vergleich zu den manuell erstellten Versionen bewerten?

1.3 Aufbau der Arbeit

Diese Arbeit gliedert sich in mehrere Kapitel, die von den theoretischen Grundlagen über den aktuellen Forschungsstand bis hin zur Implementierung und Evaluierung des Projekts *Reforge* führen. Kapitel zwei behandelt die notwendigen theoretischen Grundlagen, die für das Verständnis der weiteren Kapitel erforderlich sind. Daraufhin wird im dritten Kapitel der aktuelle Stand der Forschung zusammengefasst und relevante, verwandte Webanwendungen aufgeführt, die einen Überblick über bestehende Ansätze bieten. Im vierten Kapitel liegt der Fokus auf der Konzeption und dem Entwurf von *Reforge*. Hier werden die grundlegenden Designentscheidungen und die Struktur des Systems erläutert. Das darauffolgende Kapitel fünf beschreibt die konkrete Implementierung der Anwendung, wobei auf technische Details und wichtige Komponenten eingegangen wird. In Kapitel sechs werden die Ergebnisse der Arbeit präsentiert und eine Evaluation der entwickelten Lösung vorgenommen. Kapitel sieben bietet schließlich einen Ausblick auf mögliche Erweiterungen und zukünftige Verbesserungen der Anwendung. Das abschließende Kapitel acht fasst die wesentlichen Erkenntnisse dieser Arbeit zusammen und zieht ein abschließendes Fazit.

Kapitel 2

Grundlagen

In diesem Kapitel werden die benötigten Grundlagen für diese Arbeit erläutert.

2.1 Theoretische Grundlagen

2.1.1 Large-Language-Model Grundlagen

In dieser Arbeit wollen wir den Prozess der Textgenerierung automatisieren. Dabei ist es notwendig zu verstehen, was ein LLM ist und wie dieser funktioniert. Wie der Name bereits sagt, ist ein LLM ein großes Sprachmodell, das Texte verstehen und generieren kann. Zu Beginn erhält ein LLM eine Zufuhr umfangreicherer Textkörper wie Bücher, Artikel und Webseiten. Dabei muss es verschiedene Sprachaufgaben bearbeiten, um sich selbst zu trainieren. Auf diese Weise lernt es, wie Wörter, Ausdrücke und Sätze gebildet werden, damit sie ihren Zusammenhang und ihre Bedeutung nicht verlieren. Nach diesem Prozess ist das LLM in der Lage, auf die Eingaben des Benutzers zu reagieren und ihm relevante Inhalte anzuzeigen, sofern diese in seinem Datensatz enthalten sind. [Kumar et al., 2024, S.1]

Das in dieser Arbeit verwendete LLM stammt von OpenAI. OpenAI ist bekannt für seine Generative Pre-trained Transformer (GPT) und bietet Entwicklern mit einer Application Programming Interface (API) die Möglichkeit, diese Modelle in ihre Anwendungen zu integrieren. Es gibt verschiedene GPT-Modelle, die für die Integration ausgewählt werden können. Je nach gewähltem Modell können sowohl die erzielten Ergebnisse als auch die damit verbundenen Kosten unterschiedlich ausfallen. Die kleineren GPT Modelle sind schneller und kostengünstiger als die größeren Modelle, jedoch sind die kleinen Modelle bei weitem nicht so genau wie die größeren Modelle. In dieser Arbeit wurde das GPT-3.5-Turbo Modell verwendet, welches zu den größeren Modellen gehört. Dieses Modell besitzt die für eine Textzusammenfassung notwendige Performance und hat überschaubare Kosten. [OpenAI, 2024c]

GPT Modelle arbeiten nicht mit Wörtern oder Zeichen als Texteinheiten, sondern mit Tokens, die Zeichen, Wortteile oder auch ganze Wörter sein können. Die API

Nutzungskosten werden auf der Basis von Tokens berechnet. Es gibt Input- und Output-Token, die jeweils unterschiedliche Kosten verursachen. [OpenAI, 2024a]

Ein weiterer wichtiger Punkt bei der Verwendung der GPT Modelle sind die Prompts. Bei einem Prompt handelt es sich um eine textuelle Aufforderung, die an einen LLM übergeben wird, damit dieser weiß, wie es vorzugehen hat. Wie man einen Prompt am besten formuliert und worauf geachtet werden muss, ist Gegenstand des Prompt Engineerings. Es gibt verschiedene Möglichkeiten, die Ausgabe der LLMs zu manipulieren. Wenn beispielsweise die generierte Ausgabe zu lang ist, kann das Modell im Prompt angewiesen werden, nur eine kurze Antwort zu geben. Auch andere Anweisungen wie Formatangaben oder Textkomplexität können im Prompt angegeben werden. Die Prompt-Entwicklung besitzt daher einen entscheidenden Einfluss auf die Ausgabe. [OpenAI, 2024c]

2.1.2 Reguläre Ausdrücke

Eine weitere hilfreiche Technik für das Bearbeiten von Dokumenten sind reguläre Ausdrücke. Sie können einem dabei helfen Muster in Texten zu finden, zu erkennen und gezielt zu verändern [Friedl, 2009, S.1]. In diesem Abschnitt wird zunächst das Grundverständnis für reguläre Ausdrücke vermittelt, indem die zentralen Konzepte und die häufig verwendeten Musterkategorien wie Zeichenklassen, Quantoren, Pattern Modifiers und Wortgrenzen dargestellt und erklärt werden.

Es gibt Literale Zeichen und Metazeichen, die verwendet werden, um bestimmte Muster in Texte zu finden. Sie dienen als kleine Bausteine, die man auf verschiedenster Art und Weise zusammensetzen kann, um die gewünschten Ausdrücke in den Texten zu finden. Zu den Literalzeichen gehören Buchstaben und Zahlen. Metazeichen sind Zeichen, die eine bestimmte Bedeutung besitzen. [Friedl, 2009, S.5]. Im Folgendem werden alle wichtigen Musterkategorien mit ihren Metazeichen aufgelistet.

Bei den Zeichenklassen werden mehrere Zeichen oder Zeichenfolgen zusammengefasst, sodass diese als eine Einheit behandelt werden. Sie können auch festlegen, welche Zeichen an einer bestimmten Stelle zulässig sind [Friedl, 2009, S.9]. In der Tabelle 2.1 werden die Metazeichen für die Gruppierung von Zeichen und ihre Bedeutung aufgelistet.

Metazeichen	Beschreibung
.	irgendein Zeichen
(a b)	a oder b
(...)	Gruppe
(?:...)	Passive (nicht erfassende) Gruppe
[abc]	(a oder b oder c)
[^abc]	Nicht (a oder b oder c)
[a-q]	Kleinbuchstaben von a bis q
[A-Q]	Großbuchstaben von A bis Q
[0-7]	Ziffern von 0 bis 7

Tabelle 2.1: Musterkategorie: Zeichenklassen

Quantoren sind Metazeichen, die einem dabei helfen eine bestimmte Anzahl an Wiederholungen von Textelementen zu finden [Friedl, 2009, S.18 ff.]. Alle wichtigen Metazeichen der Quantoren und ihre Bedeutung werden in der Tabelle 2.2 dargestellt.

Metazeichen	Beschreibung
*	0 oder mehr
+	1 oder mehr
?	0 oder 1
{n}	Genau n Wiederholungen
{n,}	Mindestens n Wiederholungen
{n,m}	Zwischen n und m Wiederholungen

Tabelle 2.2: Musterkategorie: Quantoren

Pattern Modifiers beeinflussen die Verhaltensweise des regulären Ausdrucks, zum Beispiel durch das Ignorieren von Groß- und Kleinschreibung oder das Erkennen von Zeilenumbrüchen [Friedl, 2009, S.47 ff.]. Andere Pattern Modifiers und ihre Benutzung sind in der Tabelle 2.3 aufgelistet.

Metazeichen	Beschreibung
g	Global – Muster im kompletten Text finden
i	Groß- und Kleinschreibung ignorieren
m	Multiline – behandelt den Text als mehrere Zeilen
s	String als einzelne Zeile behandeln
x	Kommentare und Leerzeichen in Mustern zulassen

Tabelle 2.3: Musterkategorie: Pattern Modifiers

Eine weitere nützliche Kategorie bilden die Wortgrenzen, welche spezifische Positionen im Text definieren können, wie zum Beispiel den Anfang oder das Ende einer Zeile. Diese Positionsmarker sind hilfreich, um Muster präzise zu lokalisieren [Friedl, 2009, S.94]. Eine Sammlung von Wortgrenzen sind in der Tabelle 2.4 aufgelistet.

Metazeichen	Beschreibung
<code>^</code>	die Position am Zeilenanfang
<code>\$</code>	die Position am Zeilenende
<code>\A</code>	Anfang der Zeichenkette
<code>\Z</code>	Ende der Zeichenkette
<code>\b</code>	Wortgrenze
<code>\B</code>	Nicht Wortgrenze
<code>\<</code>	die Position am Wortanfang
<code>\></code>	die Position am Wortende

Tabelle 2.4: Musterkategorie: Wortgrenzen

Diese Auflistung an Metazeichen bildet eine gute Grundlage, um zum Beispiel bestimmte Muster in LaTeX-Dokumenten zu finden. Ein Beispiel dazu wäre der Ausdruck `\chapter{ }`. Wenn wir danach suchen wollen, dann geht das mit diesem regulärem Ausdruck `/(?= \\chapter \\)/g`. Um dieses Beispiel besser zu verstehen, wird dieser in Stücke aufgeteilt. Mit `(?=...)` benutzen wir den passiven Lookahead, der nach unseren Muster `\chapter{ }` sucht ohne das Muster selbst zu erfassen [Friedl, 2009, S.62]. Das `\` ist in regulären Ausdrücken als Escape-Zeichen vorhanden und stellt sicher das Zeichen wie `\` und `{` auch gesucht werden können. Das `/g` zum Schluss ist unser Modifikator, der dafür sorgt, dass im kompletten Textkorpus nach Treffern gesucht wird. Wäre der globale Modifikator nicht da, dann würde die Mustersuche nach dem ersten gefundenen Muster aufhören.

2.1.3 Web-Design Grundlagen

Um die Weboberfläche von *Reforge* für den Benutzer ansprechend zu gestalten, müssen die Grundlagen des Webdesigns bekannt sein. Ein Grundgedanke für jede Webseite ist, dass der Benutzer der Webseite so wenig wie möglich nachdenken muss. Das bedeutet, dass die Webseite für den Benutzer klar und verständlich gestaltet sein muss [Krug, 2018]. Da der Benutzer Farben, Typografie und andere visuelle Elemente in der ersten Sekunde wahrnimmt und beurteilt, sollte diese klare Gestaltung bereits auf der ersten Seite der Website erfolgen. Dadurch wird sichergestellt, dass der Nutzer die Seite nicht verlässt und ein bleibender positiver Eindruck entsteht.

Wenn der Nutzer über ein Smartphone auf die Seite zugreifen möchte, muss die Seite auch an kleinere Bildschirme angepasst werden. Das Prinzip der Anpassung an verschiedene Bildschirmgrößen nennt sich Responsive Webdesign. Hierbei wird darauf geachtet, dass sich das Layout der Webseite an alle Bildschirmgrößen anpasst, so dass die Webseite auch auf Smartphones problemlos genutzt werden kann. [Naumann, 2024, S.51 ff]

Eine weitere Möglichkeit, die Website für den Benutzer attraktiver zu gestalten, ist die Verwendung einer visuellen Hierarchie. Mit Hilfe dieser Hierarchie kann die Aufmerksamkeit des Benutzers gezielt auf bestimmte Elemente gelenkt werden, indem beispielsweise die Größe der Typografie verändert wird. So sollten aussagekräftige Inhalte wie Überschriften größer dargestellt werden und eher unwichtige Informationen

kleiner. Ein weiterer Ansatz der visuellen Hierarchie ist die Verwendung von Farben. Mit Farben können Kontraste gesetzt werden, die beispielsweise Schaltflächen von anderen Elementen abheben. Dies signalisiert dem Benutzer, dass mit dieser Schaltfläche eine Aktion möglich ist. Letztlich hilft die visuelle Hierarchie dem Benutzer, die Struktur der Website besser zu verstehen. [Naumann, 2024, S.54 ff.]

Darüber hinaus sollte man bei der Wahl der Farben vorsichtig sein, da diese nicht nur einen Einfluss auf das Aussehen, sondern auch auf die Emotionen der Benutzer haben können. So vermitteln Blau und Grün Vertrauen und Freundlichkeit, während Rot und Gelb eine Dringlichkeit signalisieren. Farben sollten daher strategisch eingesetzt werden, um eine bestimmte Stimmung beim Nutzer zu erzeugen. [Bartel, 2013]

Schließlich sollte der Bereich einer Website, der als Weißraum bezeichnet wird, sinnvoll genutzt werden. Dieser Bereich enthält keine Elemente und dient dazu, den Inhalt einer Seite zu trennen und zu strukturieren. Sie verhindern eine Überfrachtung des Nutzers mit zu vielen Inhalten, wodurch die Website zu unübersichtlich werden könnte [Beaird et al., 2020]. Letztendlich wird gutes Webdesign daran gemessen, wie die Informationen an den Benutzer vermittelt werden.

2.2 Software-Bausteine

Im Folgenden werden die grundlegenden Software-Bausteine beschrieben, die in *Reforge* verwendet werden.

2.2.1 Typescript

In diesem Abschnitt wird erläutert, weshalb TypeScript als Programmiersprache für die *Reforge* Anwendung ausgewählt wurde. TypeScript ist eine von Microsoft entwickelte Programmiersprache, die auf JavaScript basiert. TypeScript besitzt jedoch zusätzliche Eigenschaften, die nicht in JavaScript mit enthalten sind, wie zum Beispiel die Typisierung und die objektorientierte Programmierung. Der TypeScript-Kompilier wandelt den Code in nativen JavaScript-Code um, der von allen gängigen Browsern und JavaScript-Umgebungen ausgeführt werden kann. Diese zusätzlichen Funktionen machen TypeScript viel robuster und unterstützen den Entwickler dabei, Fehler bereits zur Entwicklungszeit zu erkennen. Dadurch wird die Zuverlässigkeit und Stabilität des Codes verbessert und der Code lässt sich langfristig besser warten und erweitern. [Japikse et al., 2020]

2.2.2 React

React ist ein Frontend Framework, welches auf JavaScript basiert. Es wurde von Facebook entwickelt und eignet sich gut für den Aufbau des User Interface (UI), insbesondere für eine Single Page Application (SPA) [Lazuardy and Anggraini, 2022]. Eine SPA bezeichnet eine Webanwendung die nur aus einem Hypertext Markup Language (HTML)-Dokument besteht und Inhalte dynamisch nachlädt [Fink et al., 2014].

Ein Kernkonzept von React ist der virtual Document Object Model (DOM). Bei diesem DOM wird eine virtuelle Darstellung einer UI gespeichert, die mit der echten DOM synchronisiert wird. Dabei werden nur die Veränderungen im DOM tree aufgenommen, weshalb die Webseite dynamisch nachlädt. Mit den React Components lassen sich UI Elemente in kleine Stücke zerteilen. Dies erlaubt es Entwicklern eine gute Code Struktur anzulegen [Lazuardy and Anggraini, 2022]. Die React Hooks ermöglichen den Zugriff auf diese Components, wodurch die aktuellen Daten auf dem Bildschirm angepasst werden. [Meta Platforms, Inc., 2024]

2.2.3 API-Schnittstelle

Die Kommunikation zwischen Frontend und Backend erfolgt über eine API Schnittstelle. Eine API dient dazu, Dienste oder Daten einer Anwendung anderen Programmen zur Verfügung zu stellen, ohne dass ein direkter Zugriff auf den zugrundeliegenden Code erforderlich ist. Grundsätzlich kann man sich eine API als ein verbindendes Element vorstellen, das eine Art Vertrag zwischen zwei Anwendungen darstellt. [De and De, 2017, S.1]

Eine häufig verwendete Art von Web-API ist die RESTful API, die das Hypertext Transfer Protocol (HTTP)-Protokoll verwendet, um Daten zwischen einem Client und einem Server zu übertragen. Die Daten werden als Ressourcen angelegt, die durch eine eindeutige Uniform Resource Identifier (URI) identifiziert werden. Jede Ressource repräsentiert eine bestimmte Entität, zum Beispiel `/users` für eine Liste von Benutzern oder `/users/{id}` für einen einzelnen Benutzer. [De and De, 2017, S.29 ff.]

Nach der Beschreibung der Ressource muss angegeben werden, was mit der Ressource gemacht werden soll. Die RESTful API verwendet dazu vier HTTP-Verben, die beschreiben, wie mit der Ressource umgegangen werden soll. Wenn eine neue Ressource erstellt werden soll, wird das **POST** Verb verwendet. Anschließend können die Informationen der Ressource mit dem **GET** Verb abgefragt werden. Wenn man eine bestehende Ressource aktualisieren möchte, kann man dies mit dem **PUT** Verb erledigen. Schließlich kann man die Ressource mit dem **DELETE** Verb löschen. [De and De, 2017, S.37 ff.]

Darüber hinaus müssen die gesendeten Daten sich selbst beschreiben, damit der Client versteht, wie die Anfrage zu verarbeiten ist. Diese zusätzlichen Informationen werden im Header der HTTP-Nachricht beschrieben, wie zum Beispiel der Content-Type, der das Format der Nachricht beschreibt. [De and De, 2017, S.33]

Kapitel 3

Forschungsstand und verwandte Arbeiten

In diesem Abschnitt werden die aktuellen Forschungsstände von Technologien beschrieben, die für das Verständnis und die Umsetzung des Projekts wesentlich sind. Dazu gehören der aktuelle Stand der Textzusammenfassung, LLMs, LaTeX, Office Open XML Document (DOCX) und Web-Entwicklung.

3.1 Forschungsstand

Die **Textzusammenfassung** per Hand erweist sich als sehr mühsam, da die Masse an Texten im Internet stetig steigt. Aus diesem Grund wird versucht, die Textzusammenfassung zu automatisieren. Dabei wurde eine Formel aufgestellt, die die Kompressionsrate τ des Textes berechnet. [Yadav et al., 2022, S.1]

$$\tau = \frac{|\text{Summary}|}{|\text{Source}|} \quad (3.1)$$

Es wurde festgestellt, dass bei einer generierten Zusammenfassung die beste Leistung bei einer Kompressionsrate von 15-30% erzielt wird [Yadav et al., 2022, S.1]. Bei einer Kompressionsrate von weniger als 15% kann es vorkommen, dass für die Verständlichkeit wichtige Informationen fehlen. Der Text wird kompakt, aber möglicherweise unvollständig, und der Kontext geht verloren. Umgekehrt werden bei einer Kompressionsrate von mehr als 30% zu viele Details beibehalten, was dazu führt, dass die Funktion einer kurzen und prägnanten Zusammenfassung nicht mehr erfüllt wird.

Eine Zusammenfassung kann auf zwei Arten erfolgen, die in der Abbildung 3.1 dargestellt sind. Die Extraktive Textzusammenfassung entnimmt Sätze aus dem originalen Textkorpus und setzt diese in der Zusammenfassung zusammen.

Die Abstrakte Zusammenfassung hingegen erstellt aus dem originalen Text neue Sätze und bildet dadurch die Zusammenfassung. [Yadav et al., 2022, S.3]

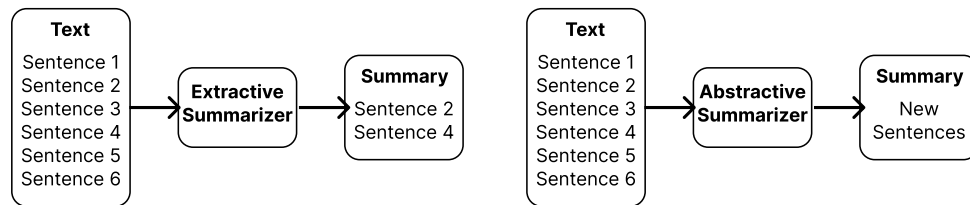


Abbildung 3.1: Extractive and Abstractive text summarizer [Yadav et al., 2022, S.3]

Die LLMs stehen durch das Wachstum des Internets große Informationsmengen zur Verfügung. Sie können kurze Zusammenfassungen von Quelltexten erstellen, ohne den Gesamtzusammenhang zu verlieren. Allerdings muss man bei der Verwendung von LLMs vorsichtig sein und den Output kontrollieren, da zum Beispiel Halluzinationen auftreten können. Unter Halluzinationen versteht man Ausgaberesultate von LLMs, die weder real noch im gelernten Datensatz enthalten sind. [Banerjee et al., 2023, S.1]

In Fisher's LLM Comparison Research zeigt sich, dass LLMs eine Möglichkeit bieten, Zeit zu sparen. Dennoch ist es notwendig, dass jemand die generierten Daten überprüft [Fisher et al., 2024]. Es zeigt sich auch, dass LLMs noch nicht an von Menschen geschriebene Texte heranreichen. Denn die von LLMs generierten Texte sind umfangreich, folgen einer logischen Struktur und wiederholen sich häufig. Im Gegensatz dazu ist menschliches Schreiben vielfältig und unterschiedlich, da jeder Mensch eine andere Art hat, sich auszudrücken. Es wurde jedoch festgestellt, dass sich LLMs ebenso wie menschliche Autoren voneinander unterscheiden [Rosenfeld and Lazebnik, 2024]. Es bleibt abzuwarten, ob LLMs in Zukunft so sprachgewandt werden wie Menschen.

Ein weiterer interessanter Aspekt im Zusammenhang mit der Entwicklung von Prompts für LLMs ist der von Levi und Neumann getestete Jailbreak-Angriff. Diese Attacke zielt darauf ab, den Sicherheitsmechanismus eines Sprachmodells zu umgehen. Dabei werden gezielt unauffällige Wörter oder Trennzeichen in Prompts eingefügt, um unerwünschte oder gefährliche Antworten zu provozieren. Selbst scheinbar harmlose Veränderungen des Prompts können das Verhalten des Modells erheblich beeinflussen [Levi and Neumann, 2024]. Die Sicherheit von LLMs spielt daher heute eine zentrale Rolle, insbesondere bei der Entwicklung von Modellen für die breite Öffentlichkeit.

LaTeX ist ein Textsatzsystem, das vor allem in wissenschaftlichen Kreisen zur Erstellung von Dokumenten verwendet wird. Es bietet erweiterte Funktionen für die Darstellung komplexer mathematischer Formeln und Referenzierungen. Die Verwendung von LaTeX ermöglicht eine präzise Kontrolle über das Layout und die Formatierung, was besonders bei umfangreichen Dokumenten von Vorteil ist.

Aktuelle Entwicklungen in LaTeX zielen darauf ab, die Zugänglichkeit und Wiederverwendbarkeit eines Portable Document Format (PDF) zu verbessern, um

Standards wie PDF/Universal Accessibility (UA) zu erfüllen. Diese Entwicklungen sind bereits seit einigen Jahren im Gange und 2024 wurde ein Prototyp erstellt, der diesen Standards entspricht. [Mittelbach et al., 2024]

DOCX ist eines der am weitesten verbreiteten Formate zur Erstellung und Bearbeitung von Textdokumenten. Es bietet eine breite Palette an Funktionen für Textformatierung, Multimedia-Integration und Layoutgestaltung.

Eine der neuesten Funktionen von DOCX ist die Integration von Copilot. Copilot ist eine KI, die Benutzer beim Entwerfen und Schreiben von Dokumenten unterstützt. Sie bietet Möglichkeiten, Texte in Tabellen umzuwandeln, Texte umzuschreiben und mit dem Benutzer in einem interaktiven Chat zu kommunizieren. Zudem kann Copilot auch Zusammenfassungen generieren, was den Schreibprozess erleichtert. [Microsoft, 2024a]

Die Web-Entwicklung hat in den letzten Jahren bedeutende Fortschritte gemacht und ist zu einem der zentralen Bereiche der Informatik geworden. Insbesondere die Frontend-Entwicklung mit JavaScript-Frameworks wie React, Angular und Vue.js wird immer beliebter. Die Analyse von Vyas zeigt, dass React mit 35,9% der Entwickler derzeit das beliebteste Framework ist. Außerdem zeigt Vyas mit Hilfe von Google Trends, dass React ein stabiles Wachstum aufweist, während Angular leicht rückläufig ist und Vue nur langsam wächst. [Vyas, 2022]

3.2 Verwandte Webanwendungen

In diesem Abschnitt werden verwandte Webanwendungen vorgestellt, welche ähnlich wie *Reforge* mit KI arbeiten und ebenfalls Zusammenfassungen generieren können. Sucht man im Browser nach Summary-Generatoren, so stößt man auf unzählige Seiten und Tools, die Texte zusammenfassen. Eine Handvoll dieser Tools wird näher betrachtet. Anschließend werden die Unterschiede des vorliegenden Projekts erläutert.

ChatGPT ist ein KI-basiertes Sprachmodell von OpenAI, das entwickelt wurde, um auf natürliche Weise mit Benutzern zu interagieren. Es ermöglicht beispielsweise das Schreiben von Texten, das Erlernen neuer Konzepte und das Sammeln von Ideen. In der mobilen Anwendung kann der Benutzer sogar eine Sprachkonversation starten. Außerdem können Antworten mit Links zu relevanten Webquellen abgerufen werden. Darüber hinaus bietet ChatGPT die Möglichkeit, Dateien hochzuladen, um sie zu analysieren oder zusammenzufassen. ChatGPT verfügt also über eine Vielzahl von Funktionen. [OpenAI, 2024b]

Microsoft Copilot ist ein von Microsoft entwickelter KI-Assistent, der sogar in Microsoft 365-Anwendungen integriert ist. Mit Hilfe von generativer KI kann Copilot Inhalte erstellen und Daten analysieren. Er bietet auch eine Chat-Funktion, über die Nutzer Anweisungen an Copilot übermitteln können. [Microsoft, 2024b]

Google Gemini ist eine von Google entworfene Chat-Anwendung mit künstlicher Intelligenz. Es hilft Nutzern bei Aufgaben wie Schreiben, Planen und Lernen. Durch die Eingabe von Textbefehlen können Nutzer direkt mit der KI interagieren

und erhalten passende Antworten. Gemini nutzt Informationen aus verschiedenen Quellen, einschließlich anderer Google-Dienste, um hilfreiche Antworten zu liefern. [Sundar Pichai, 2023]

Quillbot ist ein KI-gestütztes Schreibwerkzeug, das als Chrome-Erweiterung verfügbar ist. Es unterstützt den Benutzer bei der Verbesserung von Texten durch Funktionen wie Paraphrasierung, Grammatikprüfung und Tonerkennung. Darüber hinaus bietet es weitere Funktionen wie die Erstellung von Zusammenfassungen¹. Außerdem bietet die Erweiterung sowohl kostenlose als auch kostenpflichtige Funktionen, um den unterschiedlichen Bedürfnissen der Nutzer gerecht zu werden.

Unterscheidungsmerkmale von *Reforge*

Zusammenfassend unterscheidet sich das vorliegende Projekt von den genannten Lösungen durch folgende Eigenschaften:

- *Reforge* bietet keine Chat-Funktion, sondern nur Parameterfelder, die ausgefüllt werden müssen. Nach der Eingabe wird die Verarbeitung der Eingaben ähnlich wie bei Quillbot über einen Button gestartet.
- Ein wesentlicher Unterschied besteht in der Ausgabe. Der Nutzer kann wählen, ob er die Ausgabe als IEEEtran oder als OTH-Forschungsbericht erhalten möchte. Außerdem kann entschieden werden, ob der Technische Bericht in deutscher oder englischer Sprache erhalten werden soll. Diese Ausgabeparameter sind bei den genannten Werkzeugen nicht vorhanden.
- Das System bietet eine automatisierte Zusammenfassung von LaTeX-ZIP-Archive und DOCX-Dokumenten mit Hilfe von LLMs. Der Benutzer muss nicht wie bei ChatGPT oder Copilot Prompts an den LLM senden.

Dieses Projekt stellt somit eine innovative Lösung zur Generierung von technischen Berichten aus LaTeX- oder DOCX-Dokumenten dar. Der Benutzer ist flexibel und kann die gewünschte Ausgabe anpassen.

¹Quillbot: <https://quillbot.com/summarize>

Kapitel 4

Konzeption und Entwurf von Reforge

4.1 Anforderungsanalyse

Eine Anforderungsanalyse ist ein essenzieller Schritt im Entwicklungsprozess von Softwareprojekten. Sie stellt sicher, dass alle notwendigen Anforderungen erfasst, dokumentiert und verstanden werden, bevor mit der eigentlichen Entwicklung begonnen wird. Im Rahmen dieses Projekts wird das Modell der Anforderungsanalyse nach Kleuker herangezogen. [Kleuker and Kleuker, 2013, S.51 ff.]

4.1.1 Funktionale Anforderungen

Funktionale Anforderungen werden üblicherweise durch die Modalverben „muss“, „soll“ und „wird“ klassifiziert, die den Erfüllungsgrad der jeweiligen Anforderung definieren. Anforderungen von wichtiger Bedeutung für das System werden mit „muss“ beschrieben, da deren Erfüllung für den Projekterfolg unverzichtbar ist. Anforderungen, die als hilfreich oder wünschenswert gelten, jedoch nicht zwingend erforderlich sind, werden mit „soll“ gekennzeichnet. Anforderungen, die als potenzielle Erweiterungen für die Zukunft betrachtet werden, werden mit „wird“ formuliert. Im Folgenden werden alle funktionalen Anforderungen des Projekts *Reforge* nach dieser Klassifizierung beschrieben. [Kleuker and Kleuker, 2013, S.71 ff.]

Des Weiteren werden, ähnlich zu Raus Vorgehensweise, für die einzelnen Anforderungen Akzeptanzkriterien formuliert, um diese präziser zu definieren. Ein Akzeptanzkriterium ist eine spezifische Bedingung, die erfüllt sein muss, damit eine Anforderung als vollständig umgesetzt gilt. [Rau, 2016, S.49]

A1 DOCX- und LaTeX-Input

A1.1 Textextraktion: Das Programm muss in der Lage sein, den Textinhalt aus den hochgeladenen Dokumenten zu extrahieren. Es werden nur Abschlussarbeiten im DOCX-Format oder als LaTeX-ZIP-Ordner entgegengenommen. Die LaTeX-ZIP-Ordner müssen zudem eine Hauptdatei enthalten. Damit wird sichergestellt, dass der Text

für die weitere Bearbeitung zur Verfügung steht. Diese Anforderung gilt als erfüllt, wenn der extrahierte Text den gesamten Inhalt des Dokuments umfasst und keine wesentlichen Teile auslässt.

A2 LLM Textzusammenfassung

Die Anforderungen in „LLM Textzusammenfassung“ umfasst mehrere Funktionen, die sicherstellen, dass Texte von dem LLM verarbeitet und zusammengefasst werden können.

A2.1 Zusammenfassung-Generator: Das Programm muss mithilfe eines LLMs in der Lage sein, eine Zusammenfassung der hochgeladenen Inhalte zu generieren. Das Akzeptanzkriterium dieser Anforderung ist erfüllt, wenn die generierten Zusammenfassungen die Hauptaussagen des Originaldokuments widerspiegeln.

A2.2 Textaufteilung zur Vermeidung von Token-Limits: Das Programm muss in der Lage sein, große Textmengen in kleinere Segmente aufzuteilen und diese nach der Verarbeitung wieder zu einem vollständigen Text zusammenzufügen. Diese Anforderung gilt als akzeptiert, wenn der Prozess der Aufteilung und Zusammenführung die inhaltliche Reihenfolge des ursprünglichen Textes nicht verändert. Außerdem muss die Segmentierung für sowohl DOCX als auch LaTeX funktionieren.

A2.3 API-Schnittstelle für LLM: Das Programm muss in der Lage sein, mit der OpenAI-API zu kommunizieren. Das Akzeptanzkriterium dieser Anforderung ist erfüllt, wenn die Anwendung eine stabile Verbindung zur OpenAI-API aufbauen kann und Anfragen ohne Fehler übermitteln kann.

A3 DOCX- und LaTeX-Output

Die Anwendung bietet zwei Exportfunktionen für die Ausgabe des generierten Textes. Sie bietet die Konvertierung in ein Word-Dokument und in ein LaTeX-Dokument.

A3.1 DOCX-Ausgabe: Die Anwendung muss den generierten Text in das .docx-Format konvertieren können und dabei die Forschungsbericht-Vorlage verwenden. Als Akzeptanzkriterium wird sichergestellt, dass der entstandene Text in Microsoft Word fehlerfrei lesbar und editierbar ist.

A3.2 LaTeX-Ausgabe: Die Anwendung muss den generierten Text zu .tex konvertieren können und dabei die IEEEtran-Vorlage berücksichtigen. Diese Anforderung gilt als akzeptiert, sobald das entstandene .tex-Dokument fehlerfrei kompilierbar ist.

A4 Multi-lingual (EN/DE)

A4.1 Sprachauswahl: Der Benutzer muss auswählen können, in welcher Sprache der Bericht erstellt wird. Diese Auswahl muss unabhängig von der Eingabesprache des Inputs erfolgen. Diese Anforderung gilt als akzeptiert, wenn der Nutzer eine Sprachauswahl tätigen kann.

A4.2 Automatische Übersetzung: Die Anwendung muss eine automatisierte Übersetzungsfunktion für englische und deutsche Versionen des Techreports haben. Sobald die vom Benutzer gewählte Sprache im generierten Endprodukt vorliegt, ist diese Anforderung erfüllt.

A5 Sonstige funktionale Anforderungen

A5.1 Benutzeroberfläche: Die Anwendung muss einen Upload-Bereich für die Dokumente haben. Die Anwendung muss Eingabefelder für die Report-Metadaten wie Titel, Autor und Datum haben. Die Anwendung muss eine Auswahl der Ausgabeformate und der gewünschten Sprache haben. Das Akzeptanzkriterium dieser Anforderung ist erfüllt, wenn alle Oberflächenelemente funktionsfähig sind.

A5.2 Fortschrittsanzeige: Die Anwendung soll den Fortschritt während der Textverarbeitung und des Exports anzeigen. Diese Anforderung gilt als erfüllt, sobald eine Fortschrittsanzeige zu sehen ist.

A5.3 Export-Funktion: Die Anwendung muss die Möglichkeit bieten, die generierten Berichte herunterzuladen. Das Akzeptanzkriterium dieser Anforderung ist erfüllt, wenn ein Dokument erfolgreich lokal gespeichert werden konnte.

A5.4 Fehlerbehandlung: Die Anwendung sollte auf Fehlersituationen wie ungültige Dateiformate oder API-Ausfälle reagieren und dem Benutzer entsprechende Meldungen anzeigen. Diese Anforderung gilt als akzeptiert, wenn der Benutzer in der Lage ist Fehler-Rückmeldungen zu sehen.

4.1.2 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen sind entscheidend dafür, dass ein funktionstüchtiges und benutzerfreundliches Software-Projekt entsteht, das den Qualitätsstandards entspricht. [Kleuker and Kleuker, 2013, S.77]

Zur Klassifikation der nicht-funktionalen Anforderungen wird das FURPS-Modell herangezogen. FURPS steht dabei für Funktionalität, Usability, Reliability, Performance und Supportability. [Jamwal, 2010]

Die Funktionalität umfasst Funktionen des Systems und Sicherheitsvorkehrungen:

- Plattformunabhängigkeit: Die Anwendung sollte auf verschiedenen Betriebssystemen und in allen gängigen Webbrowsern konsistent funktionieren.
- Responsive Design: Die Benutzeroberfläche sollte sich dynamisch an verschiedene Bildschirmgrößen anpassen und sowohl auf Desktops als auch auf mobilen Geräten gut nutzbar sein.

Die Usability fokussiert sich auf die Benutzerfreundlichkeit:

- Intuitive Benutzeroberfläche: Die Anwendung sollte einfach und intuitiv zu bedienen sein. Die wichtigsten Funktionen wie zum Beispiel der Upload sollten leicht zugänglich sein.

- Fehlermeldungen: Die Anwendung sollte verständliche Fehlermeldungen haben, welche dem Benutzer Hinweise geben, wie Fehler behoben werden können.

Die Reliability bezieht sich auf die Zuverlässigkeit des Systems:

- Fehlertoleranz: Die Anwendung sollte in der Lage sein, bei einzelnen Fehlern wie zum Beispiel falsche Benutzereingaben weiterzufunktionieren.
- LLM Verbindungsfehler: Die Anwendung sollte den Benutzer informieren, wenn die Verbindung zur LLM im Backend fehlschlägt.

Die Performance beschreibt Anforderungen an Geschwindigkeit und Effizienz:

- Reaktionszeit: Die Anwendung sollte in der Lage sein, Dokumente schnell zu verarbeiten und Zusammenfassungen in höchstens einer Minute zu generieren.
- Effiziente Verarbeitung: Trotz der Arbeit mit großen Textdokumenten sollte die Anwendung so optimiert sein, dass sie effizient mit Ressourcen wie Speicher umgeht.

Die Supportability definiert die Wartbarkeit und Erweiterbarkeit des Systems:

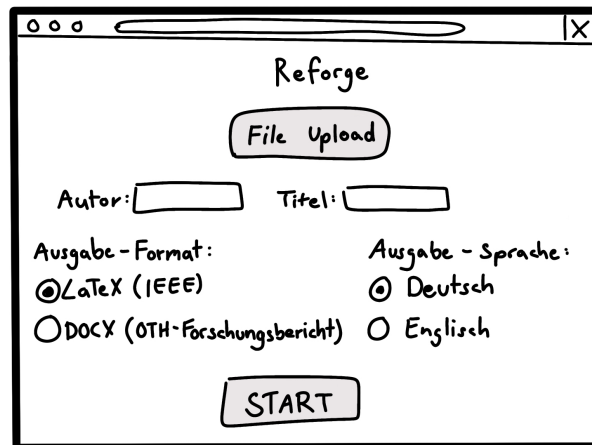
- Zukünftige Formate: Die Architektur der Anwendung sollte so gestaltet sein, dass leicht neue Exportformate oder Vorlagen hinzugefügt werden können, ohne große Änderungen am bestehenden System vorzunehmen.
- API-Erweiterbarkeit: Die Integration neuer APIs, zum Beispiel für Übersetzungen oder andere LLM-APIs, sollte leicht möglich sein.

4.2 Wireframes

Ein wichtiges Mittel des Prototypings ist die Erstellung eines Wireframes. Ein Wireframe ist eine vereinfachte grafische Darstellung der Benutzeroberfläche einer Anwendung. Wireframes werden verwendet, um die grundlegende Struktur und Funktionalität einer Website oder Anwendung vor dem eigentlichen Design und der Implementierung zu visualisieren. Sie zeigen, wie Informationen angeordnet sind, welche Elemente auf der Benutzeroberfläche vorhanden sind und wie die Navigation im System funktioniert. Wireframes helfen, Ideen zu kommunizieren, Feedback von Nutzern oder Stakeholdern einzuholen und sicherzustellen, dass das Layout der Benutzeroberfläche den Anforderungen entspricht. [Preim and Dachzelt, 2015, S.119]

Die Abbildung 4.1 zeigt den Entwurf der Upload-Seite, in der alle wesentlichen Elemente skizziert sind. Zentrales Element der Seite ist der File-Upload-Button, über den die Benutzer ihre Dokumente hochladen können. Direkt darunter befinden sich die Eingabefelder für Autor und Titel, mit denen das generierte Dokument beschriftet wird. Es folgen Radiobuttons zur Auswahl des Ausgabeformats, wobei der Nutzer zwischen LaTeX und DOCX wählen kann. Daneben ist eine Sprachauswahl eingebettet, die ebenfalls über Radiobuttons gesteuert wird und zwischen Deutsch und Englisch unterscheidet. Schließlich gibt es noch einen Start-Button, mit dem der Benutzer die

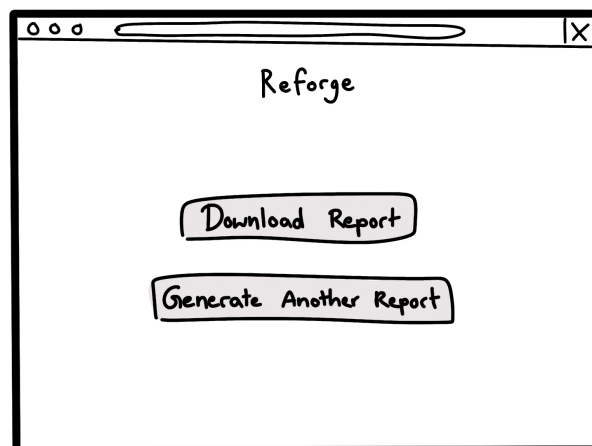
Generierung des Berichts startet. Die Elemente wurden so angeordnet, dass die Seite wie ein Formular aussieht, das von oben nach unten ausgefüllt werden muss.



The wireframe shows a browser window titled 'Reforge'. Inside, there is a 'File Upload' button. Below it are two input fields labeled 'Autor:' and 'Titel:'. Further down are two sections: 'Ausgabe - Format:' with radio buttons for 'LaTeX (IEEE)' (selected) and 'DOCX (OTH-Forschungsbericht)', and 'Ausgabe - Sprache:' with radio buttons for 'Deutsch' (selected) and 'Englisch'. At the bottom is a 'START' button.

Abbildung 4.1: Wireframe für die Upload-Seite von Reforge

Die Abbildung 4.2 zeigt die Hauptfunktionen der Download-Seite, die sich in zwei Buttons zusammenfassen lassen. Der eine Button ist für das Herunterladen der generierten Datei zuständig. Mit dem anderen Button kehrt der Benutzer zur Upload-Seite zurück, falls er sich entscheidet, einen weiteren Bericht zu generieren. Diese beiden Wireframes bilden eine gute Grundlage für die Entwicklung und stellen sicher, dass die Anordnung der Elemente optimal geplant wird.



The wireframe shows a browser window titled 'Reforge'. Inside, there are two buttons: 'Download Report' and 'Generate Another Report'.

Abbildung 4.2: Wireframe für die Download-Seite von Reforge

4.3 Auswahl des LLMs für die Textzusammenfassung

Die Auswahl eines geeigneten LLMs für die Textzusammenfassung ist ein wesentlicher Schritt, um die Qualität der erzeugten Berichte zu gewährleisten. Für dieses Projekt wurde das von OpenAI entwickelte GPT-3.5-turbo ausgewählt, da dieses LLM entscheidende Vorteile gegenüber anderen LLMs bietet.

Ein großer Vorteil ist, dass die LLMs von OpenAI nicht selbst gehostet werden müssen. Dies erleichtert die Integration der LLMs in das System. Zudem verfügt OpenAI über eine gut dokumentierte API, die sich leicht in die bestehende Systemarchitektur integrieren lässt, was die Entwicklungszeit verkürzt. Darüber hinaus zeichnen sich die von OpenAI entwickelten Modelle durch Präzision und ein tiefes Verständnis komplexer Textstrukturen aus. Sie ermöglichen die Erstellung von kontextbezogenen Zusammenfassungen, was insbesondere bei der Verarbeitung wissenschaftlicher Texte eine große Rolle spielt.

OpenAI beschreibt in der Dokumentation, wie ein REST-API-Aufruf gestaltet werden kann. Dieses Beispiel aus der offiziellen OpenAI Dokumentation ist in Listing 1 dargestellt. Hier ist zu sehen, dass im API-Request ein Modell definiert werden muss. OpenAI bietet viele verschiedene LLMs an, die alle unterschiedliche Stärken und Schwächen besitzen. Eine Übersicht aller aktuell von OpenAI angebotenen Modelle ist auf der offiziellen Webseite¹ zu finden. Darunter ist das GPT-4o-mini Model zu finden, welches hier in diesem Beispiel von OpenAI verwendet wird. Dieses Modell zeichnet sich damit aus günstig und schnell für kleinere Aufgaben zu sein.

```
1 import OpenAI from "openai";
2 const openai = new OpenAI();
3
4 const completion = await openai.chat.completions.create({
5   model: "gpt-4o-mini",
6   messages: [
7     { role: "system", content: "You are a helpful assistant." },
8     {
9       role: "user",
10      content: "Write a haiku about recursion in programming.",
11    },
12   ],
13 });
14
15 console.log(completion.choices[0].message);
```

Listing 1: OpenAI REST-API-Example [OpenAI, 2024c]

Als nächstes muss eine Nachricht, der sogenannte Prompt, definiert werden, welche das LLM bei der Bearbeitung der Anfragen unterstützt. Diese Nachricht ist in mehrere Rollen unterteilt. Es gibt die system-Rolle, die dem LLM Anweisungen gibt, was es erledigen und wie es reagieren soll. Die user-Rolle beschreibt, was der Output sein soll. Diese Rolle ist vergleichbar mit dem Benutzer im ChatGPT, der im Chat eingibt, was er als Ergebnis haben möchte. [OpenAI, 2024c]

Im Vergleich dazu sind andere Modelle, wie die von Hugging Face angebotenen LLMs, zwar ebenfalls leistungsfähig, erfordern aber oft eine umfangreiche Feinabstimmung und eine eigene Hosting-Infrastruktur, was den Aufwand für das Projekt erhöht hätte.

¹OpenAI Modell Übersicht: <https://platform.openai.com/docs/models>

Die Entscheidung für OpenAI basierte daher auf einer Abwägung zwischen einfacher Integration, hoher Sprachqualität und möglichst effizienter Entwicklungszeit.

4.4 Systemarchitektur

In diesem Abschnitt wird zunächst die Architektur des Systems erläutert. Anschließend wird auf die Speicherung der Daten eingegangen. Schließlich wird ein Sequenzdiagramm vorgestellt, welches den Upload Ablauf visualisiert.

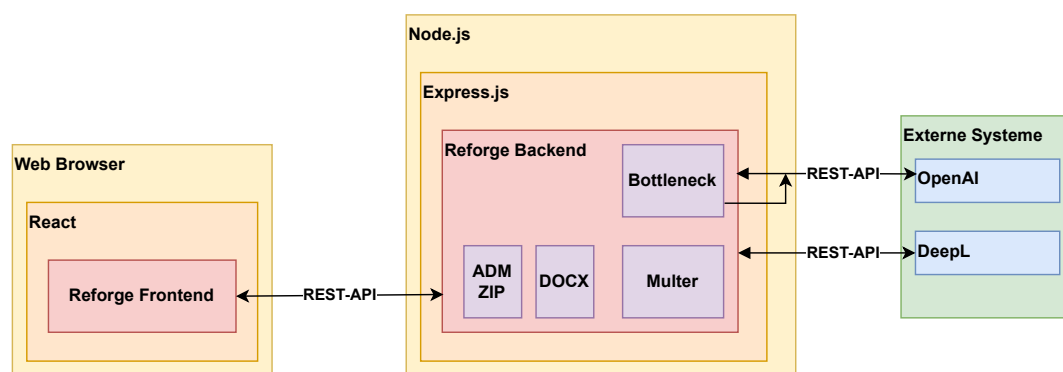


Abbildung 4.3: Systemarchitektur von Reforge

Die Abbildung 4.3 zeigt den Entwurf für die Systemarchitektur der Anwendung Reforge. Diese Architektur setzt auf eine RESTful API, die eine Kommunikation zwischen Frontend, Backend und den externen Systemen ermöglicht. Dabei werden Daten über HTTP-Methoden wie **POST** ausgetauscht.

Das Frontend basiert auf React und läuft im Webbrowser des Benutzers. Es stellt die Benutzerschnittstelle bereit, über die alle Interaktionen erfolgen. Die Benutzereingaben werden über die RESTful API an das Backend übertragen.

Das Backend, ausgeführt mit Node.js und Express.js, übernimmt die serverseitige Logik und verarbeitet die Anfragen, die vom Frontend gesendet werden. Die Verwaltung der Dateispeicherung erfolgt mithilfe von Middleware wie Multer¹, das die Daten temporär im Arbeitsspeicher zwischenspeichert, bevor sie weiterverarbeitet werden. Für das Management und die Verarbeitung von Dateien verwendet das Backend ADM-ZIP² für das Handling von ZIP-Dateien sowie DOCX³ zur Formatierung und Erstellung von Word-Dokumenten. Diese Tools ermöglichen es dem Backend, die hochgeladenen Inhalte im gewünschten Format bereitzustellen.

¹Multer: <https://expressjs.com/en/resources/middleware/multer.html>

²ADM-ZIP: <https://www.npmjs.com/package/adm-zip>

³DOCX: <https://www.npmjs.com/package/docx>

Zur Erweiterung der Funktionalität greift die Anwendung auf externe Systeme zurück. Die OpenAI API⁴ wird für die Verarbeitung und Generierung von Textinhalten genutzt, während die DeepL API⁵ die Übersetzung von Überschriften innerhalb der Anwendung übernimmt. Da OpenAI bei zu vielen Anfragen in kurzer Zeit blockiert, wird Bottleneck⁶ eingesetzt, um die Anfragerate für die OpenAI API gezielt zu drosseln.

4.4.1 Speicherung von Daten

Im Rahmen dieses Projekts wurde bewusst darauf verzichtet, eine Datenbank zu implementieren, da dies nicht Teil der Anforderungen war und den Projektrahmen gesprengt hätte. Stattdessen erfolgt die Speicherung der Daten aktuell nur in Form einer temporären Zwischenspeicherung innerhalb der Anwendung.

Für die Zwischenspeicherung kommt die Middleware Technologie Multer zum Einsatz, die es ermöglicht, hochgeladene Dateien für die Dauer der Verarbeitung zu speichern und anschließend weiterzuleiten. Hierfür wird die **memoryStorage**-Funktion von Multer verwendet, welche die Dateien als Buffer speichert. Bei dieser Methode enthält das Dateifeld ein Buffer-Objekt, welches die gesamte Datei im Arbeitsspeicher hält. Diese Vorgehensweise ist effizient für kleinere Dateien. Der Arbeitsspeicher kann jedoch bei großen oder sehr vielen kleinen Dateien überlastet werden, weshalb die Speicherlast im Auge behalten werden sollte. [OpenJSFoundation, 2024]

4.4.2 Verlauf des Uploads in Reforge

Um die zeitliche Abfolge von Interaktionen zwischen den Komponenten eines Systems darzustellen, eignet sich das Sequenzdiagramm. Es gehört zu den Verhaltensdiagrammen der Unified Modeling Language (UML) und ist eines der vier Interaktionsdiagramme. Sequenzdiagramme zeigen, wie die Kommunikation zwischen verschiedenen Systemkomponenten abläuft, indem sie die gesendeten Nachrichten und deren zeitliche Reihenfolge veranschaulichen [Rumpe and Rumpe, 2004]. Mit Hilfe von einem Sequenzdiagrammen soll nun der Prozess des Uploadens von Dateien erklärt werden.

Abbildung 4.4 zeigt das Sequenzdiagramm des Upload-Prozesses in der Anwendung. Der Ablauf beginnt im Frontend, wo der Benutzer eine Datei hochlädt. Das Backend empfängt die Datei, liest sie und bereitet den Text für die weitere Verarbeitung vor. Anschließend wird der vorbereitete Text an die OpenAI API gesendet, um eine Zusammenfassung zu generieren. Sobald die Zusammenfassung erstellt wurde, kehrt diese zum Backend zurück. Danach wird der Text an die DeepL API gesendet, um die Überschriften zu übersetzen. Die übersetzten und verarbeiteten Daten werden schließlich an das Frontend zurückgesendet, sodass die fertige Datei dem Benutzer zur Verfügung gestellt werden kann.

⁴OpenAI API: <https://platform.openai.com/docs/overview>

⁵DeepL API: <https://www.deepl.com/de/pro-api/>

⁶Bottleneck: <https://www.npmjs.com/package/bottleneck>

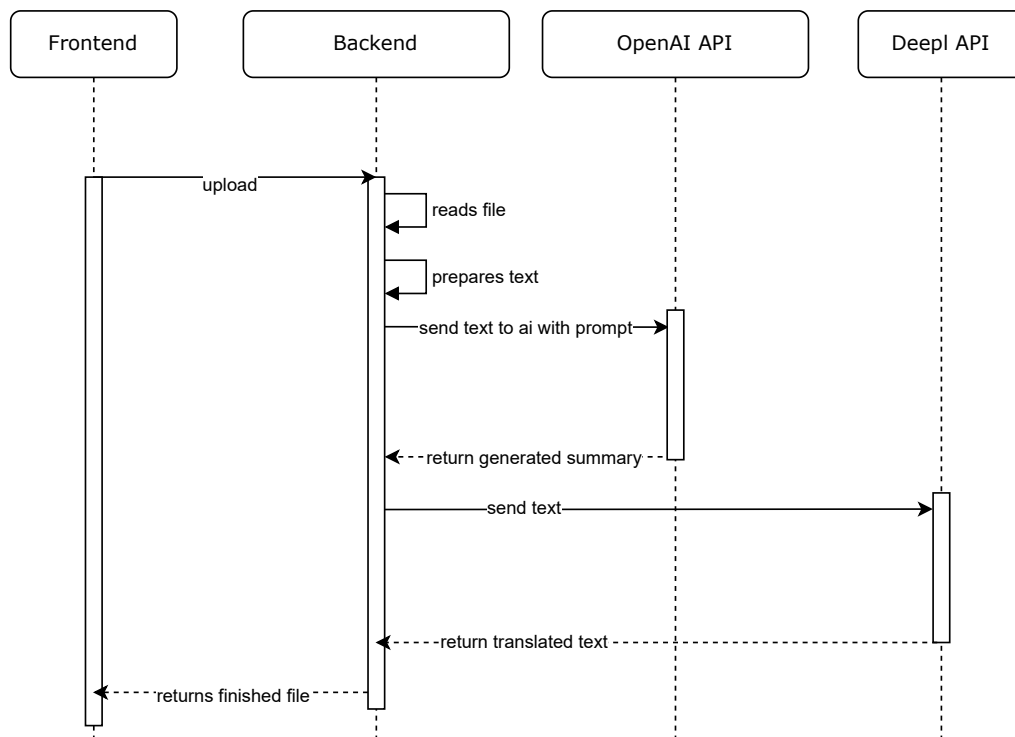


Abbildung 4.4: Sequenzdiagramm des Upload-Prozesses

Kapitel 5

Implementierung von Reforge

5.1 Entwicklung des Backends

Das Backend von *Reforge* ist für die Verarbeitung der hochgeladenen Dateien und die Generierung der technischen Berichte verantwortlich.

5.1.1 Kommunikation zwischen Frontend und Backend

Es wird nun im folgendem die Kommunikation zwischen Frontend und Backend von *Reforge* näher erläutert. Ziel dieser Kommunikation ist der Austausch von Daten und die Bereitstellung eines verarbeiteten Berichts als Datei-Download für den Benutzer. Der Ablauf erfolgt mit einer RESTful API. Er wird erst gestartet, sobald der Nutzer im Frontend zum Beispiel einen LaTeX-ZIP-Ordner einfügt, alle nötigen Parameter ausfüllt und auf den Button **Start the generation** klickt.

Das Listing 2 zeigt den Code für den Aufruf der RESTful API im Frontend. Die URI `http://localhost:5000/api/upload` verweist auf den Endpunkt des Backends, der für das Hochladen von Dateien zuständig ist. Die Anfrage wird als HTTP **POST** gesendet, da Dateien und zusätzliche Parameter wie das gewünschte Format oder die Sprache übermittelt werden. Der Header Content-Type ist auf `multipart/form-data` gesetzt, um dem Server mitzuteilen, dass es sich um einen Datei-Upload handelt. Zusätzlich wird mit `responseType: 'blob'` spezifiziert, dass die Antwort vom Server als Binary Large Object (Blob) erwartet wird, was für den späteren Download des Berichts wichtig ist. Es dient unter anderem dazu, binäre Daten übertragbar zu machen. So ist es möglich, die Dateien später herunterzuladen.

```
1 // API-Aufruf im Frontend
2 const response = await axios.post('http://localhost:5000/api/upload', formData, {
3   headers: {
4     'Content-Type': 'multipart/form-data',
5   },
6   responseType: 'blob',
7 });
```

Listing 2: API POST Aufruf im Frontend

Nachdem das Backend die Anfrage erhalten hat, wird die hochgeladene Datei verarbeitet. Sobald das Backend alle Verarbeitungsschritte zur Erstellung des Berichts abgeschlossen hat, fügt es die neuen Informationen in den Header der Antwort ein. Der Statuscode 200 wird zurückgesendet, um anzuzeigen, dass die Anfrage des Frontends erfolgreich verarbeitet wurde. Gleichzeitig wird der zusammengesetzte Bericht mit dem Statuscode an das Frontend gesendet. Das folgende Listing 3 zeigt, wie der Code dafür aussieht.

```
1 // API-Antwort im Backend
2 res.setHeader('Content-Disposition', `attachment; filename="${title}.tex"`);
3 res.setHeader('Content-Type', 'text/plain');
4 res.status(200).send(combinedReport);
```

Listing 3: Zurücksendung der abgeschlossenen API-Anfrage ans Frontend

Im Frontend wird die Antwort des Backends überprüft, wie in Listing 4 dargestellt. Bei einem Statuscode von 200 wird ein neues Blob-Objekt erzeugt, das die empfangenen Binärdaten (combinedReport) enthält. Mithilfe von `window.URL.createObjectURL` wird eine Uniform Resource Locator (URL) generiert, die es dem Benutzer ermöglicht, die Datei herunterzuladen.

```
1 if (response.status === 200) {
2   const url = window.URL.createObjectURL(new Blob([response.data]));
3   setUploadStatus('File successfully uploaded.');
```

```
4
5   navigate('/download', { state: { downloadUrl: url, title: title, format: format } });
6 } else {
7   setUploadStatus('Error while uploading the file.');
```

```
8 }
```

Listing 4: API Rückgabe Empfang im Frontend

Die URL-Generierung erfolgt dabei in mehreren Schritten. Zunächst wird das vom Backend empfangene Binärdatenobjekt als Blob formatiert. Anschließend wird mit `window.URL.createObjectURL(blob)` eine temporäre URL erstellt, die im Browser als

Verweis auf dieses Blob dient. Diese URL ist nur während der aktuellen Browsersitzung gültig und sieht beispielsweise so aus:

```
blob:http://localhost:3000/3f9a8d7e-2b4c-48e6-a91f-bb23c1def456.
```

Nach der URL-Erstellung wird der Benutzer auf eine neue Seite weitergeleitet, auf der er den Bericht herunterladen kann. Ist hingegen vorher ein Fehler aufgetreten, beispielsweise wenn das Backend einen anderen Statuscode zurückgibt, wird statt der Weiterleitung eine Fehlermeldung angezeigt.

Auf diese Weise wird die gesamte Kommunikation zwischen Frontend und Backend durch einen einzigen **POST**-Request ermöglicht. Das Frontend überträgt alle notwendigen Informationen an das Backend und wartet auf die Antwort. Nach erfolgreicher Verarbeitung stellt das Backend den generierten Bericht als Datei bereit, die im Frontend heruntergeladen werden kann.

5.1.2 Verwendete Backend Bibliotheken

Die Backend-Entwicklung der Anwendung basiert auf einer Reihe von Bibliotheken, die die Verarbeitung von Daten, die Dateiverwaltung und die Bereitstellung der APIs unterstützen. Im Folgenden werden die verwendeten Bibliotheken aus dem Listing 5 und ihre Hauptfunktionen beschrieben.

```
1 "dependencies": {  
2   "adm-zip": "^0.5.15",  
3   "bottleneck": "^2.19.5",  
4   "cors": "^2.8.5",  
5   "docx": "^8.5.0",  
6   "express": "^4.17.1",  
7   "jsdom": "^25.0.1",  
8   "multer": "^1.4.2"  
9 }
```

Listing 5: Backend Bibliotheken

Die Bibliothek ADM-ZIP ermöglicht den einfachen Umgang mit ZIP-Dateien. Sie dient zum Erstellen, Entpacken und Lesen von ZIP-Archiven. Da in Reforge mit LaTeX-ZIP-Dateien gearbeitet wird, ist diese Bibliothek unverzichtbar.

Die Bottleneck-Bibliothek wird für die Verwaltung von Anforderungsraten verwendet. Sie hilft, die Anzahl der gleichzeitigen Anfragen an externe APIs zu kontrollieren, um eine Überlastung zu vermeiden.

Die CORS Bibliothek ermöglicht die Aktivierung von Cross-Origin Resource Sharing (CORS), um den Zugriff von Webanwendungen auf Serverressourcen zu kontrollieren. Dies ist notwendig, damit der Frontend-Client sicher auf die Backend-Ressourcen zugreifen kann.

Die DOCX-Bibliothek wird zur Erzeugung von Word-Dokumenten verwendet. Sie ermöglicht es der Anwendung, Berichte im Word-Format zu erstellen und diese zu formatieren.

Express ist ein minimalistisches Web-Framework für Node.js, das die Entwicklung von Backend-Servern vereinfacht. Es bietet Funktionen zur Erstellung von APIs, die das Rückgrat der serverseitigen Logik bilden. Express stellt sicher, dass die Kommunikation zwischen Client und Server reibungslos verläuft.

Die jsdom-Bibliothek bietet eine JavaScript-Implementierung des DOM. Sie ermöglicht das Parsen und Manipulieren von HTML im Backend, was bei der Verarbeitung und Extraktion von Daten aus DOCX-Dokumenten hilfreich ist.

Multer ist eine Middleware für Datei-Uploads. Sie wird verwendet, um die vom Benutzer hochgeladenen Dateien zur Verarbeitung entgegenzunehmen. Diese Dateien werden mit der Funktion **memoryStorage()** im Arbeitsspeicher zwischengespeichert. Diese Bibliotheken bilden die Grundlage für das Backend von *Reforge*.

5.1.3 LaTeX-ZIP-Verarbeitung

In diesem Abschnitt werden die einzelnen Prozessschritte der LaTeX-ZIP-Verarbeitung erläutert. Im Listing 6 ist zu sehen, wie das Backend zunächst prüft, ob es sich bei der hochgeladenen Datei um eine ZIP-Datei handelt. Abhängig davon wird hier entweder der ZIP-Prozess oder der DOCX-Prozess gestartet.

```
1 if (req.file.originalname.endsWith('.zip')) {  
2   combinedReport = await processZipFile(req.file, mainfile, author, language, format);  
3 } else if (req.file.originalname.endsWith('.docx')) {  
4   combinedReport = await processDocxFile(req.file, author, language, format);  
5 }
```

Listing 6: Entscheidung zwischen der ZIP- oder DOCX-Verarbeitung

LaTeX-Projekte sind häufig in mehrere Kapitel und Abschnitte unterteilt, die in separaten Dateien organisiert sind. Diese Struktur wird durch die Hauptdatei gesteuert, in der die einzelnen Kapiteldateien mittels `\include{}`-Anweisungen eingebunden werden. Um die korrekte Reihenfolge der Kapitel für die Weiterverarbeitung zu haben, muss die Reihenfolge dieser `include`-Anweisungen aus der Hauptdatei entnommen werden. Damit die spätere Zusammenfassung in der richtigen Reihenfolge erfolgt, ist dieser Schritt unerlässlich.

Im Frontend fragt die Anwendung über einen Parameter den Namen der Hauptdatei des LaTeX-Ordners ab. Das Listing 7 zeigt den Code für die Suche nach der Hauptdatei. Sobald also die Funktion **processZipFile** aufgerufen wird, wird zunächst geprüft, ob die angegebene Hauptdatei innerhalb des ZIP-Ordners gefunden werden kann. Wenn die Hauptdatei nicht gefunden werden kann, wird ein Fehler ausgegeben. Ist die Suche

nach der Hauptdatei erfolgreich, wird mit der Funktion **getIncludeOrder** jede Include-Anweisung in einem String gespeichert. Außerdem wird hier die ADM-ZIP Bibliothek verwendet, um die ZIP-Inhalte zu durchsuchen und die Dateien zu entnehmen.

```

1  const zip = new AdmZip(file.buffer);
2  const zipEntries = zip.getEntries();
3  const path = require('path');
4  let includeOrder: string[] = [];
5  let mainFileFound = false;
6
7  // Mainfile im zip finden
8  for (const zipEntry of zipEntries) {
9      const entryName = path.basename(zipEntry.entryName);
10
11     // wenn mainfile gefunden -> hole include reihenfolge
12     if (!zipEntry.isDirectory && entryName === mainfile) {
13         mainFileFound = true;
14         const fileContent = zipEntry.getData().toString('utf-8');
15         if (fileContent.includes('\\include{')) {
16             includeOrder = getIncludeOrder(fileContent);
17             break;
18         }
19     }
20 }
21
22 // error wenn mainfile nicht gefunden
23 if (!mainFileFound) throw new Error(`Main file "${mainfile}" not found in ZIP`);

```

Listing 7: Dieser Codeabschnitt sucht die LaTeX-Hauptdatei im ZIP-Ordner und holt sich die Include-Reihenfolge aus der Hauptdatei

Die getIncludeOrder-Funktion

Der folgende Code in Listing 8 zeigt die Implementierung der Funktion, welche die Include-Reihenfolge ausliest. Die Suche nach dem Inhalt erfolgt mit einem regulären Ausdruck, wobei der Inhalt zwischen den geschweiften Klammern {} des regulären Ausdrucks am wichtigsten ist, da wir diesen Inhalt für die Datei zusammenstellung benötigen. Dieser Part des Musters wird dementsprechend näher erläutert: $([\^}]^+)$

```
1 function getIncludeOrder(mainfile: string): string[] {  
2     const includePattern = /\include\{([~}]+\)\}/g;  
3     let match;  
4     const includeFiles: string[] = [];  
5  
6     //suche nach Übereinstimmungen solange match !== null  
7     while ((match = includePattern.exec(mainfile)) !== null) {  
8         //jedes gefundene include wird in den array gepushed  
9         includeFiles.push(match[1].trim());  
10    }  
11  
12    return includeFiles;  
13 }
```

Listing 8: Funktion zur Extraktion der Include-Reihenfolge

Für ein besseres Verständnis wird das Muster `([~}]+)` in mehreren Teilen erklärt. Die eckigen Klammern `[~}]` definieren eine Negation. Das `~` bedeutet „nicht“, also steht `[~}]` für „kein `}`“. Das Pluszeichen `+` bedeutet, dass mindestens ein oder mehrere Zeichen gefunden werden sollen, die nicht `}` sind. Die runden Klammern `(...)` um `([~}]+)` gruppieren diesen Teil und sorgen dafür, dass der Inhalt in einer sogenannten Capture Group gespeichert wird. Mit diesem Muster können alle Dateinamen innerhalb der Include-Klammer extrahiert werden.

Solange das Muster in der While-Schleife Übereinstimmungen findet, wird diese Übereinstimmung in einem String angelegt. Dazu wird mit `match[1]` auf den gefundenen Inhalt der Capture Group zugegriffen. Zusätzlich wird mit `trim()` sichergestellt, dass keine unerwünschten Leerzeichen enthalten sind.

Kombinieren der Kapiteldateien

Nachdem die Reihenfolge der Kapiteldateien ermittelt wurde, müssen die entsprechenden `.tex`-Dateien aus dem ZIP-Archiv geladen und in der richtigen Reihenfolge zusammengefügt werden. Das folgende Listing 9 zeigt, wie die Dateien aus dem ZIP-Archiv mit der erhaltenen Reihenfolge zusammengefügt werden.

```
1 for (const fileName of includeOrder) {  
2   //hinzufigen von .tex zur filename falls nicht vorhanden  
3   const texFileName = fileName.endsWith('.tex') ? fileName : `${fileName}.tex`;  
4   //finde texFileName im Zip  
5   const zipEntry = zipEntries.find(entry => entry.entryName.endsWith(texFileName));  
6   //prüfung ob datei existiert und kein verzeichnis ist  
7   if (zipEntry && !zipEntry.isDirectory) {  
8     //hole text inhalt  
9     let fileContent = zipEntry.getData().toString('utf-8');  
10    //sammlung von allen datei inhalten in allFileContent  
11    allFileContent += fileContent + '\n';  
12  }  
13 }
```

Listing 9: Kombination der Kapiteldateien

Zunächst wird geprüft, ob der Dateiname die Endung `.tex` enthält. Falls nicht, wird diese angehängt, da in LaTeX-Dateien oft nur der Basisname im Include angegeben ist. Anschließend wird das ZIP-Archiv nach der entsprechenden Datei durchsucht. Wenn die Datei gefunden wird und es sich nicht um ein Verzeichnis handelt, wird ihr Inhalt ausgelesen und in der Variablen `allFileContent` gespeichert. Dieses Verfahren stellt sicher, dass die Kapiteldateien in der richtigen Reihenfolge kombiniert werden, so dass der gesamte Text des Dokuments vollständig rekonstruiert werden kann.

LaTeX-Textfilter und Aufteilung

Um sicherzustellen, dass nur die relevanten Teile eines LaTeX-Dokuments an die OpenAI-Schnittstelle für die Erstellung von Zusammenfassungen übergeben werden, ist es notwendig, den Text vorher zu bereinigen. Außerdem ist die Anzahl der übertragbaren Token begrenzt, so dass überflüssige LaTeX-Befehle, Kommentare und unnötige Leerzeichen aus dem Text entfernt werden müssen. Der Befehl `\chapter` bleibt jedoch erhalten, da dieser für die spätere Aufteilung des Textes wichtig ist. Für die Filterung werden reguläre Ausdrücke verwendet, um LaTeX-Kommandos gezielt auszusortieren beziehungsweise beizubehalten. Ein Codeauszug der LaTeX-Filterfunktion ist im Anhang im Listing 24 zu finden.

Auch nach der Filterung ist der Gesamttext noch zu lang, so dass eine sinnvolle Zerlegung des Dokuments notwendig ist. Damit die KI sinnvolle Zusammenfassungen erstellen kann, ohne den Kontext zu verlieren, wird der Text mit Hilfe der `\chapter`-Kommandos zerlegt. Aus diesem Grund wurden die `\chapter`-Befehle vorher nicht herausgefiltert. Diese Vorgehensweise ermöglicht es, die begrenzte Anzahl von Token der OpenAI einzuhalten und dennoch einen zusammenhängenden Text für die Zusammenfassungen zu erzeugen. Die Kapitelaufteilung ist im Listing 10 in Zeile zwei dargestellt.

```

1 //Zerstückelung des Textes bei Chapter
2 const chapters = allFileContent.split(/(?=\chapter\{}/g);
3
4 //Alle Kapitel durchgehen & Bericht erstellen
5 const reportPromises = chapters.map((chunk) => {
6   const chapternameMatch = chunk.match(/\\chapter\{([~]+)\}/);
7   const chapterTitle = chapternameMatch ? chapternameMatch[1] : 'Unnamed Chapter';
8
9   return LanguageDetection(chunk, language, author, format, 'tex').then((report) => {
10    return `\\section{${chapterTitle}}\n${report}`;
11  });
12 });
13 const reportParts = await Promise.all(reportPromises);
14 let combinedReport = reportParts.join('\n\n');
15
16 combinedReport = removeSpecialChars(combinedReport);

```

Listing 10: Chapter-Mapping für die Textübertragung an OpenAI

Darüber hinaus wird im Listing 10 ein Kapitel-Mapping durchgeführt, damit jedes einzelne Kapitel für die Zusammenfassungsgenerierung übergeben wird. Um sicherzustellen, dass die Kapitelnamen während des Prozesses nicht verloren gehen, werden diese zuvor mit einem regulären Ausdruck gesucht und in `chapterTitle` abgelegt. Ist kein Titel vorhanden, wird der Textabschnitt mit `Unnamed Chapter` betitelt. In der Funktion **Language Detection** findet die Spracherkennung und die Kommunikation mit OpenAI statt. Diese Funktion wird im folgenden Abschnitt erläutert. Anschließend werden alle generierten Berichtsteile wieder in `combinedReport` zusammengeführt.

Es kann vorkommen, dass OpenAI bei der Generierung Sonderzeichen wie `&` oder `%` verwendet. Da LaTeX mit solchen Zeichen Fehler aufwirft, werden diese mit der Funktion **removeSpecialChars** nachträglich entfernt. Zusätzlich entfernt diese Funktion die ursprüngliche Nummerierung der Überschriften. Diese Funktion ist im Listing 25 im Anhang zu finden.

Die LanguageDetection-Funktion

Die von OpenAI generierten Texte wurden nicht immer zuverlässig in der richtigen Sprache zurückgegeben, obwohl im Prompt die gewünschte Sprache angegeben wurde. Um dieses Problem zu lösen, wurde eine Spracherkennung implementiert, welche die Sprache des generierten Textes mit der gewünschten Sprache vergleicht.

Das Listing 11 zeigt den Teil der Funktion **Language Detection**, in dem eine Schleife ausgeführt wird, bis die erkannte Sprache im generierten Bericht mit der erwarteten Sprache übereinstimmt. Im Frontend kann der Benutzer auswählen, ob die Ausgabe in Englisch oder Deutsch erfolgen soll. Diese Auswahl wird als *erwartete Sprache* definiert. Nur wenn die Sprachen übereinstimmen, wird der in **generateTechReport** erzeugte Bericht zurückgegeben. Die Spracherkennung erfolgt über die Funktion **detectLanguageUsingStopWords**, welche die Sprache der generierten Texte anhand

von Stopwörtern überprüft. Wie die Spracherkennung mittels Stopwörtern funktioniert, wird im folgenden Abschnitt erläutert.

```
1 do {  
2   summary = await generateTechReport(content, expectedLanguage, format, docType);  
3  
4   detectedLanguage = detectLanguageUsingStopWords(summary);  
5  
6   if (detectedLanguage.toLowerCase() !== expectedLanguage.toLowerCase()) {  
7     console.log(`Sprache nicht korrekt erkannt. Versuche erneut...`);  
8   }  
9 } while (detectedLanguage.toLowerCase() !== expectedLanguage.toLowerCase());  
10 return summary;  
11 }
```

Listing 11: Do-While-Schleife der LanguageDetection-Funktion

5.1.4 Spracherkennung mittels Stopwörtern

Die Analyse von Stoppwörtern ist eine einfache Methode, um die Sprache eines Textes zu ermitteln. Stoppwörter sind Wörter, die in der natürlichen Sprache als wenig informativ gelten und häufig vorkommen. Sie können jedoch Hinweise auf die Sprache eines Textes geben, da ihre genaue Form von Sprache zu Sprache variiert.

Dunning hat gezeigt, dass statistische Methoden zur Spracherkennung eingesetzt werden können. Die Idee dahinter ist, dass man anhand der Verteilung von Zeichenketten feststellen kann, ob ein Text beispielsweise auf Deutsch oder Englisch verfasst wurde [Dunning, 1994]. Bei diesem Ansatz werden statt Zeichenketten Stoppwörter verwendet, um die Sprachverteilung zu bestimmen.

Warum sind Stopwörter nützlich für die Spracherkennung?

- **Hohe Häufigkeit und Sprachspezifität:** Stopwörter treten in jedem Text häufig auf, ihre Form ist jedoch in jeder Sprache unterschiedlich. Beispielsweise kommen die deutschen Stopwörter „der“, „die“, „das“ oft in deutschen Texten vor, während sie in englischen Texten fehlen. Umgekehrt finden sich englische Stopwörter wie „the“, „and“, „is“ in fast jedem englischen Text.
- **Geringe Bedeutung für den Inhalt:** Da Stoppwörter eher eine grammatische als eine inhaltliche Funktion haben, kommen sie in fast allen Texten vor. Dies ermöglicht es, die Sprache eines Textes zu identifizieren, ohne den Inhalt genauer analysieren zu müssen.
- **Effizienz:** Die Spracherkennung mittels Stopwörtern ist sparsam, da nur eine kleine Gruppe von Wörtern gesucht und deren Häufigkeit gezählt wird. Dies ermöglicht eine schnelle Sprachanalyse.

Begrenzungen dieser Methode

Bei mehrsprachigen Texten kann es schwierig sein, eine klare Sprache zu definieren. Außerdem können kurze Texte zu einem Mangel an Stoppwörtern führen. Ähnliche Sprachen wie Spanisch und Portugiesisch können ebenfalls zu Unterscheidungsschwierigkeiten beitragen. Diese Einschränkungen gelten jedoch nicht für *Reforge*, da die Dateneingabe beispielsweise aus Bachelorarbeiten besteht. Diese sind in der Regel ausreichend lang und nur in einer Sprache verfasst.

Die in *Reforge* verwendeten Stopwörter:

```
1 const englishStopWords = ["the", "as", "by", "of", "is", "and"];  
2 const germanStopWords = ["der", "die", "das", "von", "ist", "und"];
```

Listing 12: Verwendete Stopwörter in *Reforge*

Implementierung der Spracherkennung mittels Stopwörtern

Um die Sprache eines Textes zu bestimmen, wird der Text auf das Vorkommen der in Listing 12 aufgeführten Stopwörter untersucht. Der dazu notwendige Algorithmus ist in Listing 13 dargestellt. Dabei wird für jedes gefundene Stopwort der entsprechende Zähler für die jeweilige Sprache erhöht. Am Ende vergleicht man die Zählerwerte. Ist der deutsche Zähler höher, so handelt es sich um einen deutschsprachigen Text. Ist der englische Zähler höher, liegt ein englischsprachiger Text vor. Sollte es jedoch zu einem Unentschieden kommen, kann die Sprache nicht eindeutig bestimmt werden, und es wird Unknown zurückgegeben. Dies kann darauf hindeuten, dass der Text entweder in keiner der beiden Sprachen verfasst wurde oder eine Mischung aus Deutsch und Englisch enthält.

```
1 function detectLanguageUsingStopWords(text: string): string{
2   const words = text.toLowerCase().split(/\s+/);
3
4   let englishCount = 0;
5   let germanCount = 0;
6
7   words.forEach(word => {
8     if (englishStopWords.includes(word)) {
9       englishCount++;
10    }
11    if (germanStopWords.includes(word)) {
12      germanCount++;
13    }
14  });
15
16  if (englishCount > germanCount) {
17    return "english";
18  } else if (germanCount > englishCount) {
19    return "deutsch";
20  } else {
21    return "Unknown";
22  }
23 }
```

Listing 13: Spracherkennungsverfahren mit Stopwörtern

5.1.5 OpenAI und DeepL

Der OpenAI API Aufruf für die Textgenerierung

Um eine automatische Zusammenfassung für die Dokumente zu erstellen, wird die OpenAI API verwendet. Dafür wird ein API-Aufruf durchgeführt, welcher die Texte an das GPT-3.5-Turbo Modell von OpenAI übermittelt. Dort werden die Texte zusammengefasst und in der entsprechenden Sprache zurückgeliefert.

Die API-Anfrage hierfür ist im Listing 14 dargestellt. Hier wird der Inhalt mit einem **POST** an die URI von OpenAI gesendet. Zum besseren Verständnis der Verarbeitung müssen in diesem Request jedoch noch zusätzliche Informationen angegeben werden. Wie bereits in Kapitel 4.3 erwähnt, muss angegeben werden, welches Modell OpenAI für die Anfrage verwenden soll. Außerdem muss angegeben werden, welche Nachricht an das Model übergeben werden soll. Diese Nachricht ist in zwei Rollen aufgeteilt. Die Rolle des Systems und die Rolle des Benutzers. Für die Benutzerrolle wird der erstellte Prompt eingefügt. Der Prompt enthält den Textinhalt und die Aufforderung, eine kurze Zusammenfassung in der gewählten Sprache zu generieren. Außerdem besteht die Möglichkeit, ein Token-Limit einzugeben, um die Länge der Ausgabe des LLMs zu regulieren. Schließlich wird im Header eine Autorisierung mit dem OpenAI-Schlüssel durchgeführt. Dieser API-Schlüssel muss bei OpenAI generiert werden.

```
1 const response = await limiter.schedule(() => axios.post(  
2   'https://api.openai.com/v1/chat/completions',  
3   {  
4     model: 'gpt-3.5-turbo',  
5     messages: [  
6       { role: 'system', content: 'Summarizer.' },  
7       { role: 'user', content: prompt }  
8     ],  
9     max_tokens: 500,  
10    },  
11    {  
12      headers: {  
13        'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`,  
14        'Content-Type': 'application/json',  
15      },  
16    }  
17  ));
```

Listing 14: OpenAI API-Request

Um die OpenAI-Anfragen zu verlangsamen, wird die Bottleneck Bibliothek verwendet. Mit `limiter.schedule()` wird sichergestellt, dass nur alle 1,5 Sekunden eine Anfrage gesendet wird. Dies verhindert eine Überlastung der API und stellt sicher, dass die Anfragen korrekt verarbeitet werden, ohne dass es zu Ausfällen der API kommt. Die Anzahl der Sekunden ist frei wählbar und wird im Bottleneck-Konstruktor festgelegt. Das Listing 15 zeigt den Konstruktor.

```
1 const limiter = new Bottleneck({  
2   minTime: 1500, // 1 anfrage alle 1.5 sekunden  
3 });
```

Listing 15: Bottleneck Constructor

Prompt Entwicklung

Der Prompt ist eine textuelle Aufforderung, welche der KI gegeben wird, um eine spezifische Aufgabe auszuführen. Die Art und Weise, wie der Prompt formuliert ist, hat einen direkten Einfluss auf die Qualität der Ergebnisse. Je nach Formulierung kann die KI unterschiedlich auf die Eingabe reagieren, was zu variierenden Ergebnissen führen kann. Das bedeutet auch, dass der gleiche Prompt nicht immer zum gleichen Ergebnis führt. Dies ist vor allem bei mehreren Versuchen der Fall, da die KI auch eine gewisse Varianz in ihren Antworten liefert. [White et al., 2023]

Die Aufforderung für Reforge muss daher so gestaltet werden, dass die Wahrscheinlichkeit des gewünschten Ergebnisses maximiert wird. Deshalb muss bei der Entwicklung des Prompts darauf geachtet werden, dass er weder zu allgemein noch zu spezifisch ist.

Ein guter Prompt sollte klar, präzise und zielgerichtet sein. Dabei ist es hilfreich, die erwarteten Ergebnisse im Voraus zu definieren. Beispielsweise können Anweisungen wie „kurz“, „detailliert“, „in einfachen Worten“ oder „technisch“ dazu verwendet werden, um die Relevanz der Ausgabe zu verbessern. [Wang et al., 2023]

Im Listing 16 wird dem LLM von Reforge die Anweisung gegeben, den Inhalt in einer bestimmten Sprache kurz zusammenzufassen. Der Ausdruck „kurz“ signalisiert dem LLM, dass nur die wichtigsten Informationen extrahiert werden sollen. Würde man das Wort „kurz“ weglassen, wäre die Antwort des LLM umfassender.

```
1 let prompt = `  
2   Fasse das kurz in ${language} zusammen:  
3   ${content}  
4 `;
```

Listing 16: Der Prompt von Reforge an das LLM

Die Entwicklung eines effektiven Prompts ist ein iterativer Prozess. Oft ist es notwendig, mehrere Versuche zu starten und die Formulierungen anzupassen, um ein optimales Ergebnis zu erzielen. Dieser Prozess erfordert Geduld und ein Verständnis dafür, wie das LLM auf unterschiedliche Eingaben reagiert. Es wurden verschiedene Prompts für Reforge getestet, wobei der Prompt in Listing 16 die besten Ergebnisse lieferte und daher beibehalten wurde. Die generierte Ausgabe hat mit diesem Prompt die gewünschte Form und Größe.

Grenzen der Token

Ein Token besteht aus Wortteilen und wird vom LLM bei der Verarbeitung verwendet. Ein Token kann aus etwa vier Zeichen bestehen, was im Durchschnitt etwa 3/4 eines englischen Wortes entspricht. Das bedeutet, dass 100 Token etwa 75 Wörtern entsprechen können, wobei dies je nach Sprache und Wortlänge variiert. Die Tokenlänge hängt auch von der Sprache ab, da Sprachen unterschiedlich effizient in Token unterteilt werden können. Beispielsweise sind in Sprachen wie Deutsch oder Spanisch die Wörter oft länger, wodurch sich eine andere Tokenisierung ergibt als im Englischen. Da jedes Modell seine eigene Tokengröße und Tokenisierungsmethode verwendet, ist eine genaue Angabe der Anzahl der Wörter pro Token nicht verallgemeinerbar. Für eine genaue Verarbeitung der Eingaben muss die Tokenisierung immer im Kontext des verwendeten Modells betrachtet werden. [OpenAI, 2024d]

Für die Textverarbeitung in diesem Projekt ist es notwendig, den Text in kleinere Teile zu zerlegen, da eine vollständige wissenschaftliche Arbeit die Token-Grenzen des Modells überschreiten würde. Dabei ist jedoch eine Einschränkung zu beachten. Je nach Größe der zerlegten Textsegmente kann sich die Verarbeitungszeit ändern. Größere Segmente führen zu weniger API-Anfragen, wobei die Verarbeitungszeit pro Anfrage länger ist. Kleinere Segmente hingegen führen mehrere API-Anfragen aus und bieten eine bessere Kontrolle über die Token-Nutzung und reduzieren die Verarbeitungszeit pro Anfrage.

Bei der Generierung der Ausgabe wurde das Token-Limit auf 500 Token gesetzt. Ein Limit von 100 Token führte dazu, dass die Ausgabe der KI häufig abgeschnitten wurde, weshalb dieser Wert erhöht wurde. Sollte es notwendig sein, größere Textmengen abzudecken, kann dieser Wert entsprechend angepasst werden, um umfangreichere Antworten zu ermöglichen.

Korrektur von Kapitelüberschriften anhand der DeepL API

Ein Problem bei der Textverarbeitung bestand darin, dass die Kapitelüberschriften im Vergleich zum restlichen Text nicht in der richtigen Sprache ausgegeben wurden. Außerdem war teilweise noch die alte Kapitelnummerierung vorhanden, welche die Lesbarkeit beeinträchtigte. Das Problem der Kapitelnummerierung wurde mit Hilfe regulärer Ausdrücke aus dem Dokument herausgefiltert. Die Funktion ist im Anhang im Listing 25 zu sehen.

Für die Übersetzung der Kapitelüberschriften wurde die DeepL API verwendet. Ein Vorteil dieser Lösung ist, dass DeepL sowohl kostenlos als auch leistungsstark ist. Die kostenlose Version bietet ein Volumen von 500.000 Zeichen pro Monat, welches für die Übersetzung der Kapitel ausreichend ist. Da die Kapitelüberschriften durchschnittlich circa 30 Zeichen lang sind und jedes Dokument circa sieben Kapitel enthält, ergibt sich eine Zeichenanzahl von circa 210 Zeichen pro Generierungsanfrage. Bei 500.000 Zeichen pro Monat könnten also ca. 2.380 Dokumente übersetzt werden. Dies reicht aus, um die Anzahl der monatlich generierten Berichte abzudecken.

Der Code im Listing 17 zeigt, wie für die Kapitelübersetzung zunächst nach dem Inhalt der Section-Klammern mit einem regulären Ausdruck gefiltert wird. Dieser Inhalt wird zusammen mit der im Interface ausgewählten Sprache an die DeepL API übergeben. Anschließend wird der ursprüngliche Titel durch den neuen übersetzten Titel ersetzt. Nachdem alle Titel übersetzt wurden, wird der Inhalt zurückgegeben.

```
1  const sectionRegex = /\section\{([~]+)\}/g;
2  let match;
3  let translatedContent = content;
4
5  while ((match = sectionRegex.exec(content)) !== null) {
6    const sectionTitle = match[1];
7
8    //Übersetze die Titel mit DeepL
9    const translatedSectionTitle = await translateText(sectionTitle, targetLanguage);
10
11    const originalSection = `\\section${sectionTitle}`;
12    const translatedSection = `\\section${translatedSectionTitle}`;
13
14    translatedContent = translatedContent.replace(originalSection, translatedSection);
15  }
16  return translatedContent;
```

Listing 17: Kapitel-Übersetzungsfunktion

Für den Aufruf der DeepL API müssen bestimmte Parameter übergeben werden. Dazu gehört ähnlich wie bei OpenAI auch eine Authentifizierung, welche über einen von DeepL generierten Schlüssel erfolgt. Außerdem muss der zu übersetzende Text und die Sprache, in die der Text übersetzt werden soll, übergeben werden. Der Aufruf dazu ist in Listing 18 zu sehen.

```
1  const response = await axios.post('https://api-free.deepl.com/v2/translate', null, {  
2    params: {  
3      auth_key: process.env.DEEPL_API_KEY,  
4      text: text,  
5      target_lang: targetLanguage  
6    },  
7    headers: {  
8      'Content-Type': 'application/x-www-form-urlencoded'  
9    }  
10  });
```

Listing 18: DeepL API Aufruf

5.1.6 Export-Formatierungen

Hat sich der Benutzer zuvor in der Benutzeroberfläche für eine LaTeX-Ausgabe entschieden, muss der generierte Inhalt in das entsprechende Format gebracht werden. Dazu wird der gesamte generierte Text mit einem LaTeX-Header, Referenzen und einem LaTeX-Footer versehen. Der Header bringt das Dokument in das IEEEtran-Format. Der Referenzbereich im Dokument wird derzeit nur als Platzhalter eingefügt, welcher bei Bedarf manuell angepasst werden muss.

Wenn der Benutzer sich im Frontend für eine DOCX-Ausgabe entschlossen hat, wird die DOCX-Bibliothek verwendet, um ein DOCX-Dokument zu erzeugen. Der entsprechende Code für die DOCX-Erzeugung ist im Anhang im Listing 26 zu finden.

5.1.7 DOCX-Verarbeitung

Die DOCX-Bearbeitung ist der LaTeX-Bearbeitung sehr ähnlich. Der einzige Unterschied besteht in der Art und Weise, wie der Text aufgeteilt wird. Zu Beginn muss die DOCX-Datei nach HTML konvertiert werden, damit der Text leichter nach den `<h1>`-Tags gefiltert werden kann. Diese `<h1>`-Tags stehen für die einzelnen Kapitel der Arbeit und sind daher ein guter Anhaltspunkt, um das Dokument entsprechend zu unterteilen. Die Tags werden mit regulären Ausdrücken gesucht und anschließend wie in LaTeX verarbeitet.

5.2 Entwicklung des Frontends

Das Frontend von *Reforge* besteht aus zwei zentralen Seiten. Die Upload-Seite, die zugleich die Startseite darstellt, und die Download-Seite.

Upload-Seite: Diese Seite dient als Schnittstelle zur Erfassung aller für die Berichtserstellung erforderlichen Daten. Der Benutzer kann hier die zu verarbeitenden Dateien hochladen und alle erforderlichen Parameter einstellen. Anschließend kann er den Generierungsprozess starten.

Download-Seite: Nach erfolgreicher Generierung des Berichts wird der Nutzer automatisch auf die Download-Seite weitergeleitet. Auf dieser Seite kann der Benutzer den fertigen Bericht herunterladen. Zusätzlich wird eine Option angeboten, zur Upload-Seite zurückzukehren, falls ein neuer Bericht generiert werden soll.

Die Trennung von Funktionen auf zwei separate Seiten folgt dem Prinzip der Aufgabentrennung. Mili et al. beschreiben die Separation of Concerns als ein grundlegendes Konzept zur Beherrschung der Komplexität in der Softwareentwicklung. Sie unterscheiden zwischen essenzieller Separierbarkeit, die auf den Anforderungen basiert, und zufälliger Separierbarkeit, die durch die gewählte Implementierung beeinflusst wird [Mili et al., 2004, S.76]. In diesem Fall handelt es sich um eine zufällige Separierbarkeit, da die Aufteilung des Frontends auf zwei Seiten nicht in den Anforderungen spezifiziert ist.

5.2.1 Verwendete Frontend Bibliotheken

Die Frontend-Entwicklung von *Reforge* basiert auf einer Reihe von Bibliotheken, welche in Listing 19 aufgeführt sind. Im Folgenden werden die verwendeten Bibliotheken und ihre Hauptfunktionen beschrieben.

```
1 "dependencies": {  
2   "@testing-library/jest-dom": "^5.17.0",  
3   "@testing-library/react": "^13.4.0",  
4   "@testing-library/user-event": "^13.5.0",  
5   "react": "^18.3.1",  
6   "react-dom": "^18.3.1",  
7   "react-router-dom": "^6.26.0",  
8   "react-scripts": "5.0.1",  
9   "web-vitals": "^2.1.4"  
10 },
```

Listing 19: Frontend Bibliotheken

Die testing-Bibliotheken dienen zum Testen von React-Anwendungen. Sie helfen dabei, Tests zu schreiben, welche das Verhalten der Anwendung aus Sicht des Endbenutzers überprüfen.

React ist das zentrale Framework für die Benutzeroberfläche. React-dom wird verwendet, um die React-Komponenten im Browser anzuzeigen. Die react-router-dom Bibliothek ermöglicht das Routing innerhalb der Anwendung, sprich die Navigation zwischen den verschiedenen Seiten. Die grundlegenden Skripte für den Aufbau, das Testen und das Deployment der Anwendung werden durch die Bibliothek react-scripts

zur Verfügung gestellt. Zusammen bilden diese Bibliotheken die Grundlage für die Entwicklung der Benutzeroberfläche.

Die Web-vitals Bibliothek wird verwendet, um wichtige Performancemetriken der Webanwendung zu messen. Dazu gehören Metriken wie Ladezeit, Interaktivität und Stabilität des Layouts.

5.2.2 Finales Frontend-Design

Das finale Frontend-Design, welches in Abbildung 5.1 dargestellt ist, baut auf den Grundlagen der zuvor im Kapitel 4.2 entworfenen Wireframes auf. Bei der Weiterentwicklung wurde auf eine klare und minimalistische Darstellung Wert gelegt, um den Benutzer nicht zu überfordern und eine intuitive Bedienung zu ermöglichen.

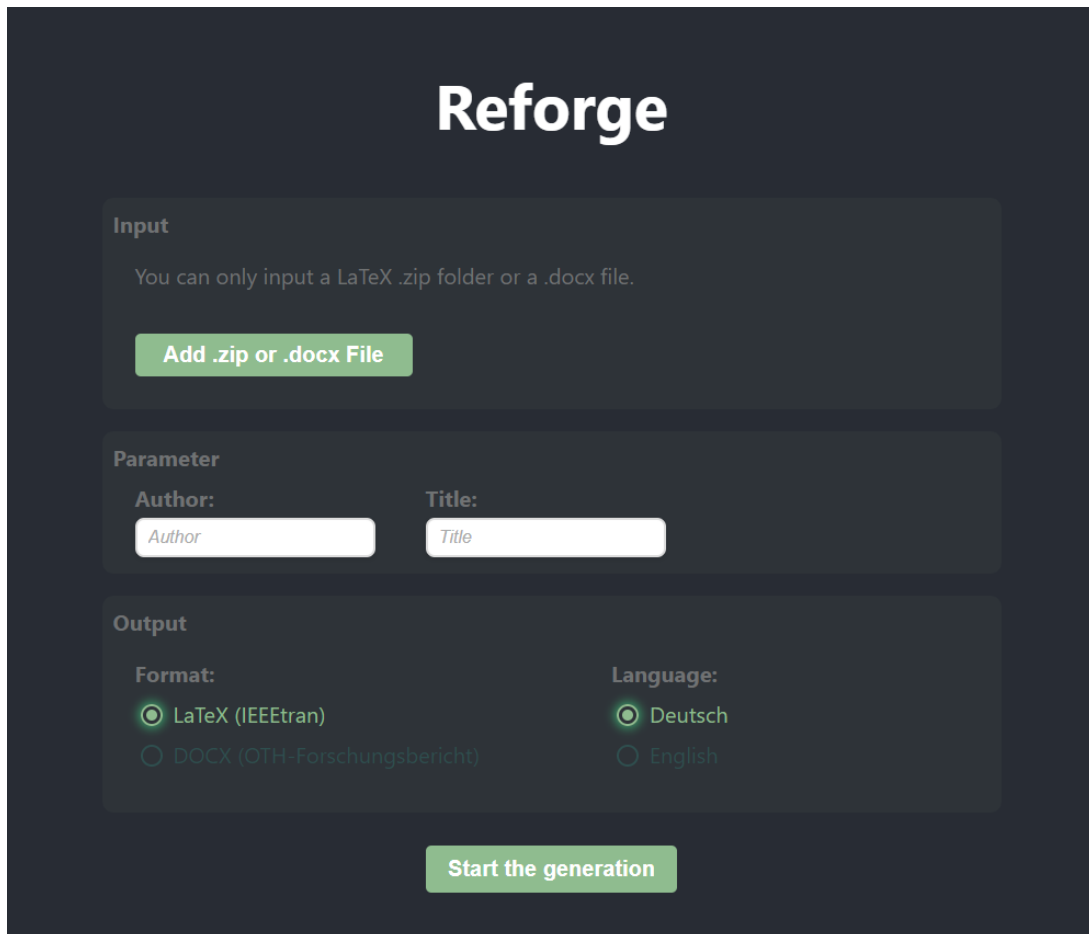
The image shows a web interface for 'Reforge' with a dark background. At the top, the word 'Reforge' is written in a large, white, sans-serif font. Below it, there are three main sections: 'Input', 'Parameter', and 'Output'. The 'Input' section contains a text box with the placeholder 'You can only input a LaTeX .zip folder or a .docx file.' and a green button labeled 'Add .zip or .docx File'. The 'Parameter' section has two input fields: 'Author:' with a placeholder 'Author' and 'Title:' with a placeholder 'Title'. The 'Output' section has two groups of radio buttons. The first group is labeled 'Format:' and has two options: 'LaTeX (IEEEtran)' which is selected with a green circle, and 'DOCX (OTH-Forschungsbericht)'. The second group is labeled 'Language:' and has two options: 'Deutsch' which is selected with a green circle, and 'English'. At the bottom of the form is a green button labeled 'Start the generation'.

Abbildung 5.1: Oberflächendesign der Upload-Seite

Die Benutzeroberfläche der Upload-Seite wurde zur besseren Übersichtlichkeit in drei klar definierte Bereiche unterteilt:

Input-Bereich: In diesem Bereich können Dokumente hochgeladen werden, die als Grundlage für die Berichtserstellung dienen. Es werden nur LaTeX-ZIP-Ordner

oder DOCX-Dateien unterstützt. Diese Einschränkung stellt sicher, dass nur zulässige Dateiformate verarbeitet werden.

Parameter-Bereich: Hier werden Parameter wie Autor, Titel und im Falle eines ZIP-Uploads die LaTeX-Hauptdatei eingegeben. Diese Informationen werden für die personalisierte Generierung des Berichts benötigt.

Output-Bereich: In diesem Bereich kann der Benutzer das gewünschte Ausgabeformat auswählen. Er kann zwischen einer LaTeX- oder einer DOCX-Ausgabe entscheiden. Außerdem kann hier die Sprache des Berichts festgelegt werden, wobei der Benutzer nur zwischen Englisch und Deutsch wählen kann.

Die Schaltfläche **Start the generation** am unteren Rand der Benutzeroberfläche ermöglicht es dem Benutzer, den Prozess der Berichtsgenerierung zu beginnen. Durch die farbliche Hervorhebung ist die Schaltfläche leicht erkennbar und intuitiv zu bedienen.

Die Abbildung 5.2 zeigt das Design der Downloadseite. Seit der Erstellung des Wireframes wurden, abgesehen von der Farbgestaltung, keine Änderungen vorgenommen. Lediglich zwei Buttons wurden vorgesehen. Mit **Download Report** wird der Bericht heruntergeladen und mit **Generate Another Report** gelangt der Benutzer zurück zur Upload-Seite, falls er einen weiteren Bericht generieren möchte.

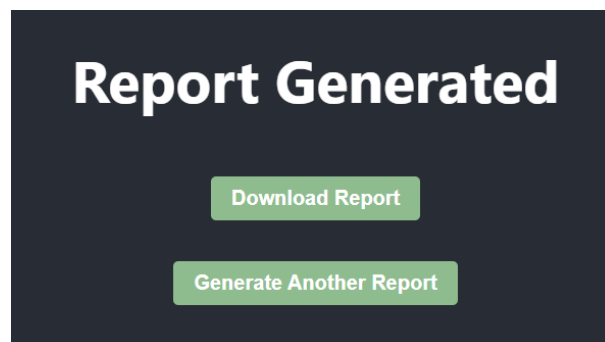


Abbildung 5.2: Oberflächendesign der Download-Seite

Insgesamt wurde bei der Oberflächenentwicklung auf folgendes geachtet:

Konsistenz: Es wurde darauf hingearbeitet, dass die Oberfläche einheitlich ist. Das bedeutet, dass zum Beispiel anklickbare Elemente wie Schaltflächen immer gleich aussehen. So versteht der Benutzer, dass es sich um eine Schaltfläche handelt. Würde man einer Schaltfläche immer wieder eine andere Farbe oder Form geben, wäre dies für den Endnutzer verwirrend. Die Konsistenz hilft, die Bedienung vorhersehbar zu machen.

Feedback und Sichtbarkeit: Ein weiterer Punkt ist, dem Benutzer ein angemessenes Feedback zu geben, insbesondere wenn er eine falsche Eingabe macht. Wenn der Benutzer beispielsweise vergisst, eine Datei hochzuladen, wird ein Hinweis angezeigt. In diesem Fall werden die Hinweise unterhalb des Start-Buttons angezeigt. Ein Beispiel für eine solche Fehlermeldung ist in Abbildung 5.3 dargestellt. Damit soll der Benutzer bei der Bedienung der Anwendung unterstützt und geführt

werden. Sichtbarkeit bedeutet auch, dass der Status des Systems für den Benutzer jederzeit klar sein sollte. Dies wird ebenfalls, wie bei der Fehlermeldung, durch eine Textmeldung an gleicher Stelle erreicht.

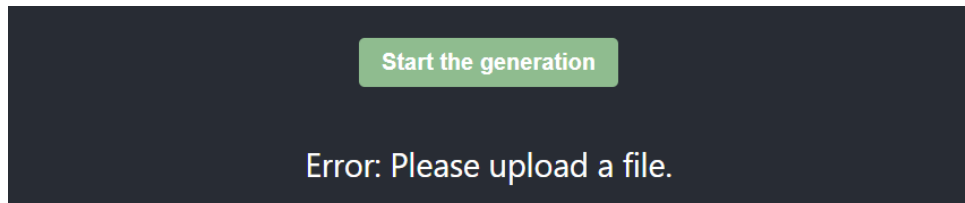


Abbildung 5.3: Fehleranzeige auf der Upload-Seite

Einfachheit und Klarheit: Die Benutzeroberfläche ist so einfach wie möglich gestaltet. Unnötige Informationen werden vermieden, um den Benutzer nicht zu überfordern. Die Verwendung von klaren Elementen und minimalem Text lenkt die Aufmerksamkeit auf die wesentlichen Funktionen.

Visuelle Hierarchie: Durch den gezielten Einsatz von Farben, Größen und Abständen wurde eine klare visuelle Hierarchie geschaffen. Wichtige Informationen oder Aktionen sind hervorgehoben. Dies erleichtert die Orientierung und unterstützt den Benutzer. Auf diese Weise erkennt der Benutzer, was auf der Seite zu erledigen ist.

Das endgültige Design der Benutzeroberfläche bietet eine klare Struktur, in der die Aufgaben des Benutzers leicht zu verstehen sind. Die gewählten Farben sorgen für eine angenehme visuelle Erfahrung und heben die wichtigsten Aktionen hervor.

5.2.3 Die Upload-Seite

Die Upload-Seite ist, wie bereits erwähnt, dafür verantwortlich, alle für die Berichtsgenerierung notwendigen Informationen an das Backend zu übermitteln. Im Abschnitt 5.1.1 wurde bereits beschrieben, wie die Kommunikation zwischen Backend und Frontend abläuft. Es wird daher nur auf die zusätzliche Eingabe eingegangen.

Wenn der Benutzer eine ZIP-Datei hochlädt, erscheint ein zusätzliches Eingabefeld. In diesem zusätzlichen Eingabefeld muss der Benutzer den Namen der Hauptdatei des LaTeX-Ordners angeben. Dies wird im Backend dafür benötigt, um die LaTeX-Inhalte in die richtige Reihenfolge zu bringen. Die Abbildung 5.4 zeigt, wie das zusätzliche Eingabefeld für die LaTeX-ZIP-Datei aussieht. Wird ein DOCX-Dokument hochgeladen, gibt es dieses Eingabefeld für die Hauptdatei nicht, da DOCX es nicht benötigt.

A dark-themed form with the title "Parameter" at the top left. It contains three input fields: "Author:" with the placeholder text "Author", "Title:" with the placeholder text "Title", and "Main .tex file:" with the placeholder text "Master.tex".

Abbildung 5.4: Das zusätzliche Eingabefeld für die LaTeX-Hauptdatei

5.2.4 Die Download-Seite

Die Download-Seite muss nur die Funktion des Datei-Downloads erfüllen. Die benötigten Daten werden mithilfe des State-Parameters an diese Seite übergeben. Dieser speichert die Zustandsinformationen der aktuellen Seite und ermöglicht deren Übergabe an die Zielseite. Der Wechsel zwischen den Seiten erfolgt über den React Router, der hierfür die Navigate-Funktion nutzt.

Im Listing 20 wird der Vorgang der Datenübergabe dargestellt. Dabei wird der React Router verwendet, um die Download-Seite anzusteuern. Der Navigate-Befehl enthält den Zielpfad `'/download'` und den State-Parameter, der die erforderlichen Informationen wie die Download-URL, den Dateititel und das gewünschte Format enthält. Diese Informationen werden anschließend auf der Download-Seite genutzt, um den Download-Prozess auszuführen.

```
1 navigate('/download', { state: { downloadUrl: url, title: title, format: format } });
```

Listing 20: State Übergabe von der Upload-Seite and die Download-Seite

Der Status der Anwendung wird auf der Download-Seite mit dem `useLocation`-Hook des React Routers ausgelesen. Die im State enthaltenen Informationen werden über `location.state` extrahiert. Da das Projekt mit TypeScript umgesetzt wird, müssen die Typen des State-Objekts eindeutig definiert werden. Dies geschieht über das Interface `LocationState`, welches die Struktur und die Typen der State-Eigenschaften eindeutig beschreibt. Die Implementierung ist im Listing 21 dargestellt.

```
1 import { useLocation } from 'react-router-dom';
2
3 interface LocationState {
4   downloadUrl: string;
5   title: string;
6   format: string;
7 }
8
9 const DownloadPage: React.FC = () => {
10   const location = useLocation();
11   const { downloadUrl, title, format } = location.state as LocationState;
```

Listing 21: UseLocation-Hook auf der Download-Seite in *Reforge*

Um die übergebenen Daten für den Download-Prozess zu verwenden, wird eine Funktion **handleDownload** implementiert. Diese erzeugt ein temporäres Link-Element, welches die Datei anhand der übergebenen URL herunterlädt. Der Dateiname setzt sich dabei aus dem Titel und dem Format der Datei zusammen. Nach dem Klick auf das erzeugte Linkelement wird dieses wieder entfernt, um keine unnötigen Elemente im DOM zu hinterlassen. Listing 22 zeigt, wie diese Funktion realisiert wird. Diese Methode stellt eine Möglichkeit dar, Dateien im Browser herunterzuladen.

```
1  const handleDownload = () => {  
2    const link = document.createElement('a');  
3    link.href = downloadUrl;  
4    link.setAttribute('download', `${title}.${format}`);  
5    document.body.appendChild(link);  
6    link.click();  
7    link.remove();  
8  };
```

Listing 22: HandleDownload-Funktion auf der Download-Seite

Kapitel 6

Evaluation und Ergebnisse

In diesem Kapitel werden die Evaluation und die Ergebnisse von *Reforge* vorgestellt. Es beginnt mit einer Beschreibung des Konfigurationsmanagements und der Inbetriebnahme von *Reforge*. Anschließend werden die Kosten für die Nutzung der API analysiert. Die Qualität der generierten Berichte und die mobile Nutzbarkeit werden bewertet. Außerdem wird ein Vergleich der Anwendung *Reforge* mit bestehenden Tools durchgeführt. Abschließend werden die wichtigsten Erkenntnisse und Erfahrungen aus dem Entwicklungsprozess zusammengefasst.

6.1 Konfigurationsmanagement und Inbetriebnahme

Um *Reforge* lokal auszuführen, sind einige Vorbereitungen erforderlich. In diesem Abschnitt wird der Prozess der API-Schlüsselgenerierung für OpenAI und DeepL sowie deren Integration in die Anwendung beschrieben.

Was ist eine `.env`-Datei?

Eine `.env`-Datei ist eine Umgebungsdatei. Sie dient dazu, sensible Konfigurationsdaten wie API-Schlüssel, Datenbankzugangsdaten oder andere Umgebungsvariablen sicher zu speichern. Die Anwendung liest die `.env`-Datei beim Start aus, so dass diese Variablen nicht direkt im Code stehen müssen.

Die API-Schlüsselgenerierung

Für die Nutzung der OpenAI- und DeepL-APIs müssen zunächst API-Schlüssel generiert werden. Der Prozess beginnt mit der Erstellung eines Benutzerkontos bei den jeweiligen Plattformen.

Für OpenAI wird die Website <https://platform.openai.com> besucht, um sich anzumelden oder zu registrieren. Danach wird ein Dashboard sichtbar, über das eine Navigation zu den API Keys möglich ist. Nachdem eine Zahlungsmethode hinterlegt wurde, kann dort über **Create new secret key** ein neuer Schlüssel generiert werden. Es

ist wichtig, den Schlüssel direkt zu kopieren und sicher zu speichern, da er nur einmal angezeigt wird.

In ähnlicher Weise erfolgt die Schlüsselgenerierung für DeepL. Es wird die Webseite <https://www.deepl.com/pro> aufgerufen, um dort einen Account zu erstellen. Anschließend muss auch hier eine Zahlungsmethode ausgewählt werden, womit der Zugriff auf die API ermöglicht wird. Im Bereich des Benutzerkontos gibt es einen API-Bereich, in dem ein Schlüssel generiert werden kann. Auch hier sollte der Schlüssel sicher gespeichert werden, da er nur einmal erscheint und für die spätere Konfiguration der Anwendung benötigt wird.

Integration der API-Schlüssel in *Reforge*

Die *Reforge*-Anwendung ist öffentlich auf GitHub verfügbar und kann lokal betrieben werden. Es ist über diesen Link erreichbar: <https://github.com/cyberlytics/reforge>

Das Repository muss zunächst in ein lokales Verzeichnis geklont werden. Im Verzeichnis `sys-src/backend` befindet sich die Datei `env-example`, die als Vorlage für die Konfiguration der API-Schlüssel dient. Um die Anwendung betriebsbereit zu bekommen, müssen die in der Datei enthaltenen Platzhalter durch die zuvor generierten API-Schlüssel ersetzt werden.

Der Platzhalter **dein-openai-api-schluessel** wird durch den API-Schlüssel von OpenAI ersetzt, während **dein-deepl-api-schluessel** mit dem DeepL-Schlüssel überschrieben wird. Anschließend wird die Datei von `env-example` in `.env` umbenannt, sodass sie von der Anwendung korrekt eingelesen werden kann. Die resultierende `.env`-Datei sollte dann wie in Listing 23 aussehen. Dabei handelt es sich hier um Beispiel-Keys, die nicht funktionsfähig sind und lediglich der Veranschaulichung dienen.

```
1 OPENAI_API_KEY=sk-12345abcdefghijklmnopqrstuvwxyz12345
2 DEEPL_API_KEY=12345678-90ab-cdef-1234-567890abcdef:fx
```

Listing 23: `.env`-Datei im `sys-src/backend`-Ordner

Inbetriebnahme der Anwendung

Nachdem die API-Schlüssel erfolgreich konfiguriert wurden, müssen die notwendigen Abhängigkeiten installiert werden. Dazu muss der Befehl **npm install** in einem Terminal in mehreren Verzeichnissen ausgeführt werden. Zuerst im Wurzelverzeichnis, dann im Verzeichnis `sys-src/backend` und schließlich im Verzeichnis `sys-src/frontend`. Dieser Schritt stellt sicher, dass alle für die Anwendung benötigten Pakete heruntergeladen und installiert werden.

Nach der Installation der Abhängigkeiten kann die Anwendung gestartet werden. Dazu wird im Terminal in das Verzeichnis `sys-src/backend` gewechselt und der Backend-Server mit dem Befehl **npm start** gestartet. Anschließend wird in das Verzeichnis `sys-src/frontend` navigiert und das Frontend ebenfalls mit **npm start** ausge-

führt. Nach erfolgreichem Start der Anwendung öffnet sich automatisch die *Reforge*-Oberfläche im Browser und steht für die Generierung von technischen Berichten zur Verfügung.

Hinweise zu möglichen Problemen

Bei der Nutzung der *Reforge*-Anwendung können unter Umständen Probleme auftreten, die berücksichtigt werden sollten. Zum einen besteht die Möglichkeit, dass die APIs von OpenAI oder DeepL temporär ausfallen. Solche Ausfälle sind selten, sind aber während der Entwicklungszeit von *Reforge* einmal aufgetreten. In solchen Fällen bleibt nichts anderes übrig, als auf die Behebung der Störung durch die jeweiligen Anbieter zu warten.

Andererseits ist zu beachten, dass die Nutzung der OpenAI-API mit Kosten verbunden ist. Diese richten sich nach der Anzahl des verarbeiteten Tokens und variieren je nach Nutzungshäufigkeit und Größe der Anfragen. Eine Aufschlüsselung der entstehenden Kosten ist im folgenden Kapitel 6.2 beschrieben.

6.2 Kosten der API-Nutzung

Für die Umsetzung der Anwendung wurden die OpenAI und DeepL APIs genutzt. Die Nutzung dieser APIs sind mit unterschiedlichen Kosten verbunden, die hier näher erläutert werden.

Die OpenAI API bietet verschiedene Preismodelle an, welche auf der Anzahl der verwendeten Tokens basieren. Die Kosten richten sich nach der Anzahl der Ein- und Ausgabe-Token, wobei pro 1.000 Token ein bestimmter Betrag berechnet wird. Für dieses Projekt wurde das GPT-3.5-turbo Modell verwendet und zum Zeitpunkt der Entwicklung betrugen die Kosten für GPT-3.5-turbo ca. 0,002 US-Dollar pro 1.000 Token. Die tatsächlichen Kosten können je nach Umfang der Anfragen variieren, da längere Dokumente und komplexere Anfragen zu einer höheren Anzahl von Token führen.

Die Abbildung 6.1 zeigt ein Diagramm, welches die Kosten der OpenAI API für den Monat August im Jahr 2024 darstellt. Dabei handelt es sich um eine Webseite, auf die zugegriffen werden kann, sobald eine OpenAI API verwendet wird. Die Balken, die in der Abbildung zu sehen sind, sind während der Entwicklung von *Reforge* entstanden, als die Anwendung mit Testdaten ausprobiert wurde. Diese Daten können jedoch für eine vorausschauende Kostenanalyse genutzt werden.

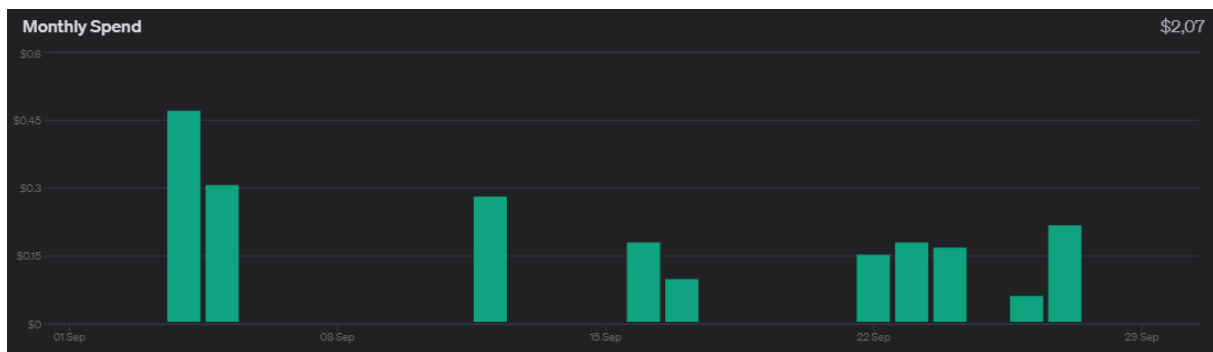


Abbildung 6.1: OpenAI Kosten von August

Ein weiteres nützliches Diagramm ist in Abbildung 6.2 dargestellt, welches die Input Tokens und Output Tokens in Relation zu den Kosten zeigt. Aus dem Diagramm geht hervor, dass es sich um Zusammenfassungen handelt, da die Balken für die Output Tokens einen geringeren Ausschlag als die Input Tokens haben.

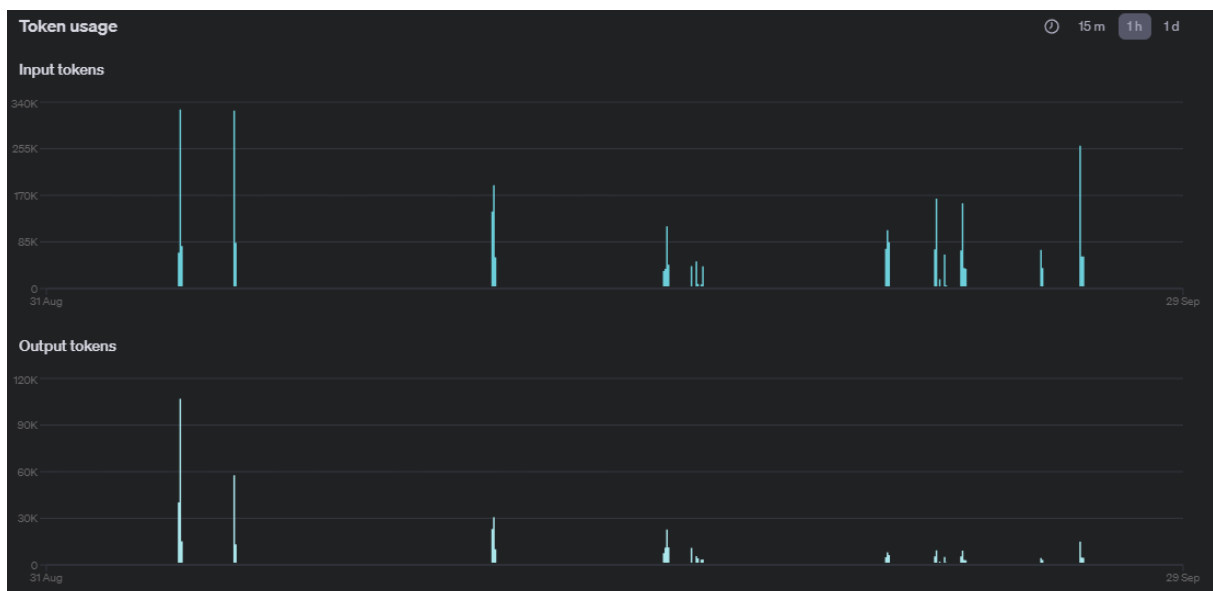


Abbildung 6.2: OpenAI Token Usage von August

Für den gesamten Entwicklungszeitraum beliefen sich die Kosten für OpenAI auf insgesamt 2,68 US-Dollar. Im August betrugen die Kosten 0,41 US-Dollar, im September 2,07 US-Dollar und im Oktober 0,20 US-Dollar. Insgesamt wurden 118 Berichte generiert, was einem Durchschnitt von ca. 0,023 US-Dollar pro Bericht entspricht. Der Tokenverbrauch zeigt, dass größere Berichte mehr Token und damit höhere Kosten verursachen, während kürzere Dokumente weniger Ressourcen beanspruchen.

Für die DeepL API wurde die kostenlose Version verwendet. Diese Version erlaubt eine begrenzte Anzahl an Übersetzungen pro Monat, bietet jedoch eine ausreichende Qualität für die Anforderungen dieses Projekts. Die kostenlose Version ist ideal für kleinere Projekte, da keine direkten Kosten entstehen und dennoch hochwertige

Übersetzungen zwischen Deutsch und Englisch möglich sind. Bei einem größeren Umfang oder einer intensiveren Nutzung könnte jedoch eine kostenpflichtige Version erforderlich werden, die zusätzliche Funktionen und ein höheres Übersetzungslimit bietet.

Die Kosten der verwendeten APIs sind insgesamt überschaubar, insbesondere da die kostenlose Version von DeepL genutzt werden kann. Die OpenAI API hingegen verursacht Kosten, die jedoch im Rahmen bleiben und durch die Qualität der generierten Zusammenfassungen gerechtfertigt werden können. Für zukünftige Projekte könnte gegebenenfalls die Nutzung eines anderen OpenAI Modells sinnvoll sein, um das Kosten-Nutzen-Verhältnis weiter zu optimieren.

6.3 Qualität der generierten Berichte

Um die Qualität der erstellten Berichte zu bewerten, wurde ein Bericht auf der Grundlage dieser Arbeit mit *Reforge* produziert. Für die LaTeX-Ausgabe wurde der Bericht in englischer Sprache generiert. Das Ergebnis dieser Generierung ist im Anhang in den Abbildungen A.2 und A.3 dargestellt. Zusätzlich wurde eine Generierung für die DOCX-Ausgabe durchgeführt. Die Ergebnisse dieser Ausgabe sind ebenfalls im Anhang unter den Abbildungen A.4 und A.5 zu finden.

Wie Chang et al. in ihrer Evaluation von LLMs beschreiben, neigen LLMs in komplexen Kontexten zu Verwechslungen oder Fehlern. Sie stoßen bei der Erkennung semantischer Ähnlichkeiten an ihre Grenzen und zeigen Schwächen in ressourcenarmen Kontexten [Chang et al., 2024]. Ob diese Einschränkungen auch hier zutreffen, wurde überprüft. Es zeigte sich, dass Schwächen in ressourcenarmen Kontexten erkennbar sind. Betrachtet man die Abschnitte Bilder und Codeauszüge in den Abbildungen A.3 und A.5, so fällt ein Mangel an Inhalt auf. Dies ist darauf zurückzuführen, dass in diesen Abschnitten kein Text vorhanden ist. Die *Reforge*-Anwendung verarbeitet derzeit keine Bilder und Codeausschnitte, wodurch dieser ressourcenarme Kontext entsteht.

Um die Kompressionsrate τ von Yadav zu berechnen, wurde die Anzahl der Wörter berücksichtigt [Yadav et al., 2022]. Diese Arbeit enthält ungefähr 24.150 Wörter. Die Anzahl der Wörter in der generierten DOCX-Zusammenfassung A.4 beträgt ungefähr 970 Wörter. Mit diesen Werten kann nun die Kompressionsrate berechnet werden.

$$0,04017 = \frac{|970|}{|24.150|} \quad (6.1)$$

Die Kompressionsrate beträgt somit gerundet 4% und liegt damit weit unter den gewünschten 15-30%, die Yadav als gut erachtet. Dies lässt den Schluss zu, dass entweder *Reforge* zu viel Inhalt entfernt oder GPT-3.5-turbo zu viel Information auslässt.

Es werden nun die gestellten Fragen aus der Einleitung 1 betrachtet. Zunächst wird untersucht, ob in den generierten Inhalten Biases auftreten. Es zeigt sich, dass das

GPT-3.5-turbo-Modell keine Biases in den generierten Inhalten aufweist, da es sich ausschließlich auf den Inhalt des hochgeladenen Dokuments stützt.

Die zweite Frage beschäftigt sich damit, ob *Reforge* mithilfe von KI bessere Ergebnisse erzielt als ein Mensch. Hier wird deutlich, dass *Reforge* in der Lage ist, eine solide Grundlage zu schaffen. Diese Ergebnisse müssen jedoch von einem Menschen überarbeitet und geprüft werden, sodass *Reforge* die Arbeit des Menschen nicht vollständig ersetzt, sondern lediglich unterstützt.

Abschließend wurde die Frage betrachtet, inwieweit sich die Input-Datei im Vergleich zur Output-Datei verändert. Es konnte festgestellt werden, dass die grobe Inhaltsstruktur erhalten bleibt, während Details teilweise ausgelassen werden. Die weitere Bewertung des erstellten Berichts erfolgte anhand folgender Kriterien.

- **Inhaltliche Genauigkeit:** Die erstellten Zusammenfassungen wurden auf ihre inhaltliche Richtigkeit überprüft. Dabei wurde geprüft, ob die wesentlichen Informationen des Originaldokuments korrekt in die Zusammenfassung übernommen wurden. Die Ergebnisse zeigen, dass das LLM in der Lage ist, die wichtigsten Inhalte zusammenzufassen. Da es sich jedoch um eine komprimierte Fassung handelt, gehen Informationen hinsichtlich der Erläuterung der Inhalte verloren.
- **Kohärenz und Lesbarkeit:** Die Berichte sind im Allgemeinen gut lesbar und logisch aufgebaut. Die Verwendung von LLM trägt dazu bei, dass die Texte eine natürliche Sprache aufweisen. Dennoch gibt es vereinzelte Fälle, in denen Sätze etwas zu komplex formuliert sind, was die Lesbarkeit beeinträchtigen könnte.
- **Formatierung und Layout:** Ein weiterer wichtiger Aspekt der Evaluierung ist die Formatierung und das Layout der Berichte. Die Anwendung ist in der Lage, vorgegebene Vorlagen wie LaTeX oder DOCX korrekt umzusetzen, so dass die resultierenden Berichte den Anforderungen der jeweiligen Vorlage entsprechen.
- **Übersetzungsqualität:** Bei der Evaluierung der Übersetzungsfunktion wurde die Qualität der Übersetzung des generierten LaTeX-Berichts zwischen Deutsch und Englisch untersucht. Das GPT-3.5-turbo-Modell erwies sich als zuverlässig. Die Übersetzungen sind im Allgemeinen korrekt und behalten die ursprüngliche Bedeutung des Inhalts bei. Die Titelübersetzungen von DeepL sind ebenfalls richtig.
- **Geschwindigkeit:** Die Erstellung der technischen Berichte dauerte immer weniger als eine Minute und ist somit ein schneller Prozess. Der Zeitaufwand für die Generierung ist im Vergleich zu manuellen Zusammenfassungen minimal.

Insgesamt wurde die Qualität der erstellten Berichte als gut bewertet. Die Anwendung erzeugt verständliche technische Berichte und ermöglicht eine korrekte Formatierung. Sie bietet eine gute Vorlage, die manuell ergänzt werden kann.

6.4 Evaluation der mobilen Ansicht von *Reforge*

Die mobile Ansicht der *Reforge*-Anwendung wurde im Browser getestet, um ihre Nutzbarkeit auf kleineren Bildschirmen zu bewerten. Obwohl sich die Webseite für mobile Geräte automatisch skaliert, treten vereinzelt kleinere Layout-Probleme auf. Diese beeinträchtigen jedoch nicht die Funktionalität der Anwendung, sodass die Seite weiterhin vollständig nutzbar bleibt. Die Darstellung der mobilen Version ist in Abbildung 6.3 zu sehen. *Reforge* erfüllt somit die Anforderungen an Responsive Design.

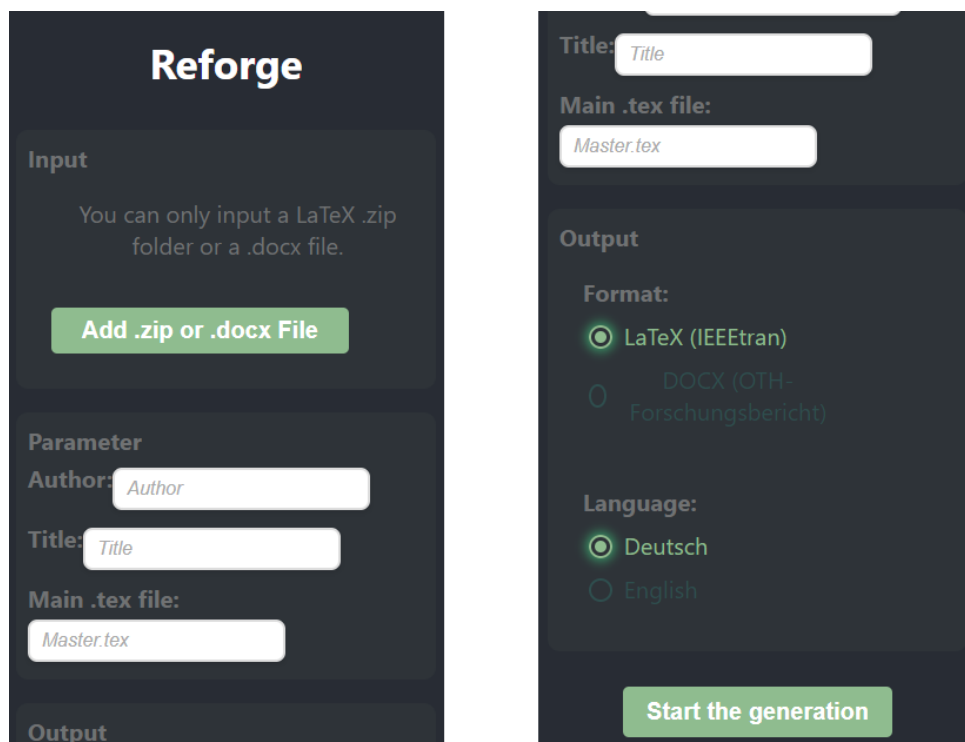


Abbildung 6.3: Die mobile Ansicht der Upload-Seite von *Reforge*

6.5 Projektvergleich mit verwandten Webanwendungen

Um herauszufinden, ob die bestehenden Werkzeuge besser sind als *Reforge*, wird in diesem Abschnitt ein Vergleich mit den verwandten Webanwendungen durchgeführt. Dies soll helfen, den Nutzen von *Reforge* besser einschätzen zu können.

6.5.1 Vergleich mit ChatGPT-4

Für diesen Vergleich wird versucht, die mit *Reforge* möglichen Ausgaben mit ChatGPT zu erzeugen. Dazu wird ChatGPT als Eingabe entweder ein DOCX-Dokument oder ein LaTeX-ZIP-Ordner übergeben. Zusätzlich wird ein Prompt angegeben, der ChatGPT anweist, einen technischen Bericht zu erzeugen. Außerdem wird das Ausgabeformat angegeben, in dem der Bericht erzeugt werden soll. Ähnlich wie bei *Reforge* wird für die

LaTeX-Ausgabe IEEEtran und für das DOCX-Dokument der OTH-Forschungsbericht angefordert. Um jede Kombination einmal zu testen, werden vier Tests benötigt. Die Ergebnisse der einzelnen Tests sind unten aufgeführt. Als LaTeX-ZIP-Testdokument für die Generierung wurde die Bachelorarbeit von Hoffmann verwendet [Hoffmann, 2022]. Die Bachelorarbeit von Schotter wurde als Testdokument für die DOCX-Generierung verwendet [Schotter, 2022].

Test 1 - LaTeX-ZIP-Ordner zu LaTeX

Für diesen Vergleich wurde ChatGPT gebeten, aus dem LaTeX-ZIP-Archiv eine LaTeX-Ausgabe zu erzeugen. In der Abbildung 6.4 ist der übergebene Test-Prompt zu sehen. Mit diesem Prompt wurde versucht, ChatGPT dazu zu bringen, aus der LaTeX-ZIP-Datei einen technischen Bericht im IEEEtran-Format zu erstellen.

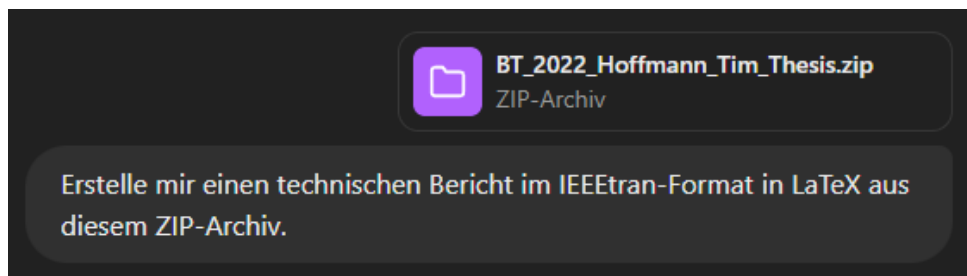


Abbildung 6.4: ZIP-Test-Prompt an ChatGPT

ChatGPT war jedoch nach mehrmaliger Aufforderung nicht in der Lage, das ZIP-Archiv zu analysieren und das gewünschte Ergebnis zu liefern. Es wurde lediglich auf den Inhalt der LaTeX-ZIP-Datei eingegangen und erwähnt, dass die Inhaltsanalyse in Arbeit sei. Die Antwort auf den Prompt ist in der Abbildung 6.5 dargestellt.

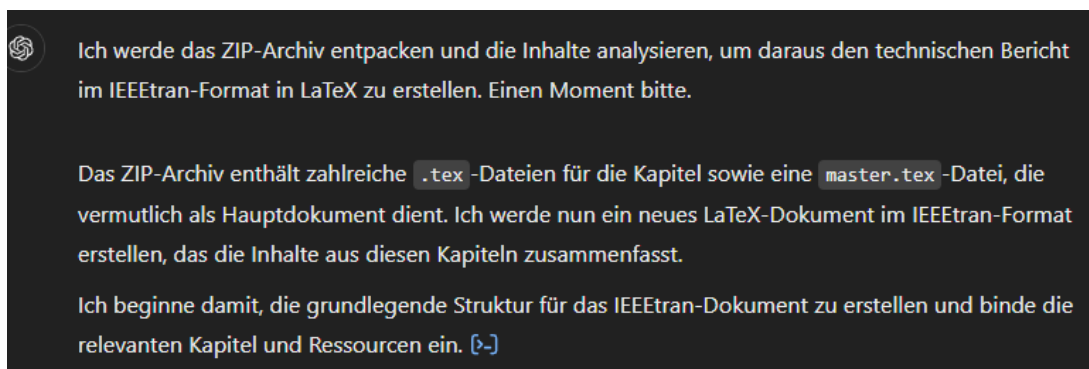


Abbildung 6.5: ZIP-Test-Antwort von ChatGPT

Test 2 - DOCX zu LaTeX

In einem weiteren Test wurde nun das DOCX-Dokument an ChatGPT übergeben. Dabei war ChatGPT in der Lage, eine LaTeX-Ausgabe zu erzeugen, die auch das IEEEtran-Format darstellt. Ein auffallender Nachteil ist jedoch der fehlende Download-Button. Mit ChatGPT muss der erzeugte Text manuell kopiert und in eine LaTeX-Datei

konvertiert werden, was für den Benutzer zusätzliche Umstände mit sich bringt. Die ChatGPT Antwort ist in der Abbildung A.6 im Anhang dargestellt.

Darüber hinaus gibt es kleinere Fehler in der Ausgabe, wie zum Beispiel die Vermischung von deutscher und englischer Sprache und das Fehlen des Inhalts des Reference-Abschnitts. Dieser Abschnitt müsste bei Bedarf manuell eingefügt werden. Außerdem wurde eine Beispiel-E-Mail eingefügt, die ebenfalls im Nachhinein angepasst werden müsste. Das generierte Ergebnis von ChatGPT ist im Anhang in der Abbildung A.7 zu sehen.

Test 3 - LaTeX-ZIP-Ordner zu DOCX

Es wurde versucht, eine DOCX-Ausgabe aus dem LaTeX-ZIP-Ordner zu erhalten. Dazu wurde ChatGPT angewiesen, einen technischen Bericht für DOCX zu erzeugen. Bei diesem Test erzeugte ChatGPT einen Download-Link, um das erstellte DOCX-Dokument herunterzuladen. Die ChatGPT Antwort auf diesen Test ist in der Abbildung A.8 im Anhang dargestellt.

Bei näherer Betrachtung wies das von ChatGPT erstellte Dokument zahlreiche Fehler auf, wie die Abbildung A.9 im Anhang zeigt. Die Kapitel sind nicht benannt, sondern durchnummeriert. Außerdem fehlt der obere Rahmen mit Titel, Autor und Studiengang. Ein weiterer Fehler ist, dass LaTeX-Befehle wie Chapter, Section, Cite, Label und Begin im Text vorkommen. Schließlich fehlt der Inhalt des Referenzteils.

Test 4 - DOCX zu DOCX

Im vierten Test wird probiert, aus dem DOCX-Dokument eine DOCX Zusammenfassung zu erzeugen. Es wird festgestellt, dass der obere Rahmen wie Autor, Studiengang, Hochschule und Ort fehlt. Außerdem gibt es auch hier keine Referenzsektion. Ein weiteres Problem ist, dass die Ausgabe manuell in eine DOCX-Datei eingefügt werden muss. Wie die ChatGPT-Ausgabe aussieht, ist im Anhang unter A.10 zu finden.

Schlussfolgerung

Zusammengefasst ist nur Test zwei auf einem ähnlichen Niveau wie die Ausgabe in *Reforge*. Allerdings muss der Benutzer im Dokument von Test zwei die Sprache, die Referenzen und die Daten wie E-Mail überprüfen oder ändern. Alle anderen Tests weisen zu viele Mängel auf, die durch den Anwender korrigiert werden müssten. Andererseits ist anzumerken, dass durch eine Änderung des Prompts möglicherweise bessere Ergebnisse erzielt werden könnten.

Im Vergleich zu ChatGPT bietet *Reforge* eine konsistente Ausgabe in Bezug auf Layout und Export. Außerdem muss sich der Benutzer nicht um einen eigenen Prompt bemühen. Er muss lediglich seine Dokumente hochladen und die gewünschten Parameter auswählen. Darüber hinaus achtet *Reforge* auf die Textsprache, so dass es nicht vorkommen kann, dass im Endprodukt eine Mischung aus Englisch und Deutsch erscheint.

6.5.2 Vergleich mit Copilot

Bei diesem Vergleich wurde zunächst versucht, ein DOCX-Dokument in Copilot hochzuladen. Dies wurde jedoch von Copilot mit der Meldung abgelehnt, dass nur Bilder hochgeladen werden können. Diese Mitteilung ist in Abbildung 6.6 dargestellt.

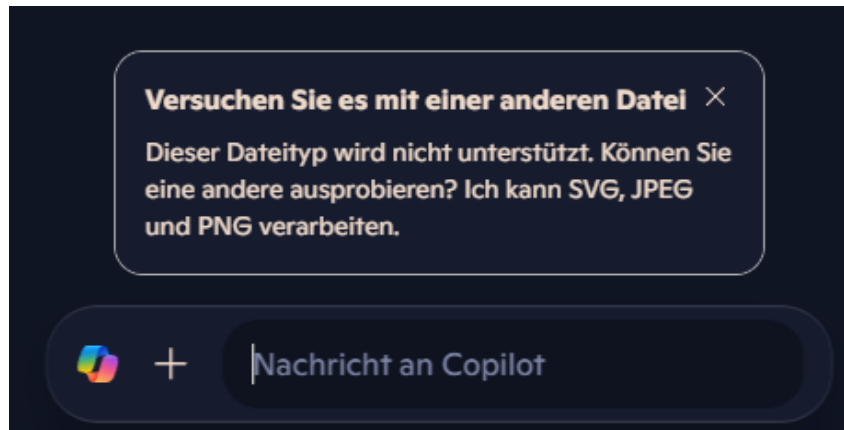


Abbildung 6.6: Copilot Datei Einschränkungen

Da ein direktes Hochladen der Datei nicht möglich war, wurde alternativ der Inhalt des DOCX-Dokuments kopiert und eingefügt. Dabei stellte sich heraus, dass Copilot ein Zeichenlimit von 10.240 Zeichen vorgibt. Dies ist problematisch, da wissenschaftliche Arbeiten in der Regel deutlich umfangreicher sind. Diese Beschränkung ist in Abbildung 6.7 dargestellt. Um Copilot dennoch als Werkzeug für die Berichterstellung nutzen zu können, wäre es notwendig, den Text manuell in kleinere Abschnitte zu unterteilen und diese einzeln zu bearbeiten. Dies würde jedoch einen erheblichen Mehraufwand für den Anwender bedeuten.

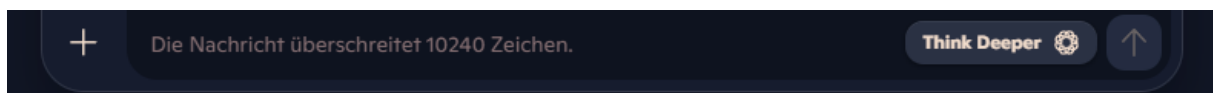


Abbildung 6.7: Copilot Textlängen Limitierung

Abschließend lässt sich festhalten, dass *Reforge* gegenüber Copilot den Vorteil bietet, dass Dokumente direkt hochgeladen werden können und die Inhalte automatisch in bearbeitbare Abschnitte unterteilt werden. Zudem muss der Benutzer bei *Reforge* keine eigenen Prompts erstellen, da diese bereits vordefiniert sind. Dies vereinfacht die Anwendung und reduziert den Arbeitsaufwand.

6.5.3 Vergleich mit Gemini

Als drittes Vergleichstool wurde Gemini ausgewählt. Ähnlich wie bei Copilot ist das Hochladen von Dateien in Gemini eingeschränkt, wie in Abbildung 6.6 dargestellt. Außerdem muss der Benutzer bei Gemini zusätzlich einen entsprechenden Prompt für die Verarbeitung angeben.

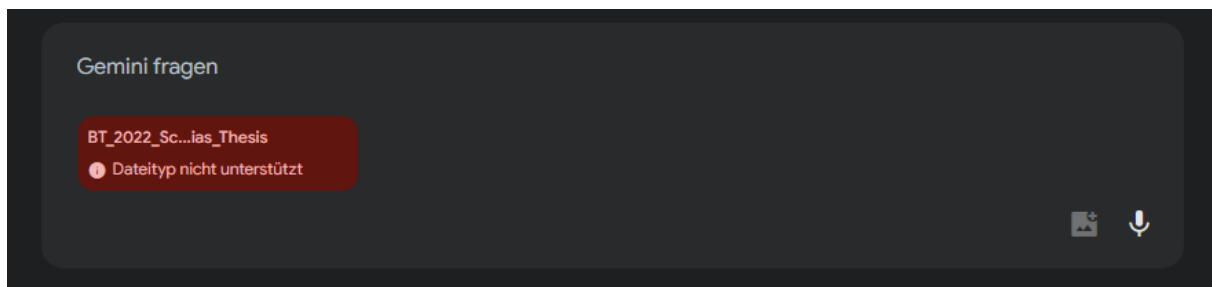


Abbildung 6.8: Gemini Datei Einschränkungen

Stattdessen wurde erneut versucht, den Textinhalt manuell in den Chat einzufügen. Auch hier gibt es jedoch eine Zeichenbegrenzung, ab der das Einfügen weiterer Inhalte nicht mehr möglich ist. Gemini zeigt allerdings keine genauen Informationen über die Zeichenbegrenzung an, was die Arbeit mit längeren Texten erschwert. Um den gesamten Inhalt eines Dokuments verarbeiten zu können, wäre es daher notwendig, den Text manuell in kleinere Abschnitte zu unterteilen.

Im Vergleich zu Gemini zeigt sich, dass der direkte Datei-Upload von *Reforge* den Benutzern viele Arbeitsschritte erspart. Der Textinhalt wird in *Reforge* automatisch in bearbeitbare Abschnitte unterteilt und es sind keine zusätzlichen Eingabeaufforderungen erforderlich.

6.5.4 Vergleich mit QuillBot

Zuletzt wurde *Reforge* mit QuillBot verglichen, einem Tool, das wie *Reforge* keine Chatfunktion bietet. Es dient lediglich zum Zusammenfassen von Texten und unterstützt daher nicht das direkte Hochladen von Dokumenten. Aus diesem Grund muss der Inhalt einer wissenschaftlichen Arbeit für die Zusammenfassung manuell kopiert und eingefügt werden. Es wurde jedoch festgestellt, dass QuillBot eine Begrenzung der Textlänge besitzt, die in Abbildung 6.9 dargestellt ist. Dieses Limit kann durch das Anlegen eines Accounts oder den Erwerb einer Premium-Mitgliedschaft erhöht werden. Diese Abbildung zeigt auch, dass die Zusammenfassung durch die Auswahl von Schlüsselwörtern personalisiert werden kann.

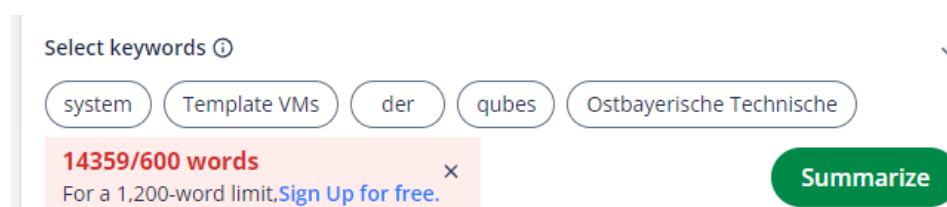


Abbildung 6.9: Quillbot Textlängen Limitierung

Weitere Einstellungen, wie beispielsweise die Länge der Zusammenfassung, können ebenfalls in QuillBot vorgenommen werden. Eine Übersicht dieser Einstellungsmöglichkeiten ist in Abbildung 6.10 dargestellt.

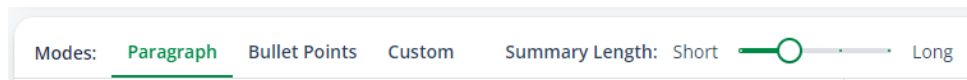


Abbildung 6.10: Quillbot Einstellugen für die Zusammenfassung

Zusammenfassend lässt sich festhalten, dass QuillBot aufgrund der Beschränkung der Textlänge nicht mit *Reforge* mithalten kann. Zudem fehlt die Möglichkeit, Dokumente direkt hochzuladen, was eine manuelle Eingabe des Textinhalts erfordert. Andererseits bieten die zusätzlichen Zusammenfassungsoptionen von Quillbot interessante Ansätze. Beispielsweise könnte die Anpassung der Zusammenfassungslänge für zukünftige Versionen von *Reforge* in Betracht gezogen werden.

6.6 Lessons Learned

Die aus Projekten gewonnenen Erkenntnisse werden als Lessons Learned bezeichnet. Patton betrachtet Lessons Learned als lokales, erfahrungsbasiertes Wissen, das oft unspezifisch und ungeprüft ist. Er diskutiert, dass Lessons Learned nicht nur als Methode der Programmevaluierung verwendet werden sollten, sondern auch als Wissensquelle für eine effektive Programmgestaltung [Patton, 2001]. Im Folgenden wird daher auf die Erkenntnisse eingegangen, die zur Verbesserung zukünftiger Programmkonzeptionen genutzt werden können:

- **Technische Herausforderungen und Lösungsansätze:** Die Integration der OpenAI API erwies sich als technisch anspruchsvoll. Dies gilt insbesondere für die Optimierung der Token-Nutzung, um die Kosten gering zu halten. Es wurde gelernt, dass eine sorgfältige Planung der Abfragen und die Aufteilung langer Texte entscheidend ist, um die Tokenbeschränkungen einzuhalten.
- **Bedeutung der Benutzerfreundlichkeit:** Ein weiterer wichtiger Aspekt war die Gestaltung der Benutzeroberfläche. Die Gestaltungsprinzipien erwiesen sich als hilfreich, um die Anwendung auch für Nutzer ohne tiefere technische Kenntnisse zugänglich zu machen. Es wurde deutlich, dass eine klare und einfache Benutzerführung den Umgang mit der Anwendung erleichtert.
- **Kostenmanagement und API-Nutzung:** Die monatlichen Kosten für die OpenAI API waren überschaubar. Das Projekt hat jedoch gezeigt, dass eine genaue Kostenüberwachung und eine Beschränkung der Anfragen notwendig sind, um im vorgegebenen Budget zu bleiben. Die Nutzung der kostenlosen Version von DeepL war dabei eine sinnvolle Entscheidung, um die Kosten gering zu halten.
- **Zukunftspotenzial:** Während der Entwicklung wurde deutlich, dass die Anwendung durch die Integration weiterer Funktionen noch verbessert werden kann. Diese Erweiterungen bieten Potenzial für zukünftige Projekte und könnten die Anwendungsbreite und den Nutzen von *Reforge* weiter ausbauen. In Kapitel 7 werden mögliche Projekterweiterungen erläutert.
- **Zeitplanung:** Eine gute Zeitplanung ist für die Entwicklung von Anwendungen unerlässlich. Daher ist es wichtig, im Voraus zu überlegen, wie viel Zeit für

welche Funktionen aufgewendet werden soll.

Die gewonnenen Erkenntnisse bieten wertvolle Einblicke, die in zukünftigen Projekten genutzt werden können.

Kapitel 7

Projektausblick von Reforge

In diesem Kapitel werden mögliche Erweiterungen und Verbesserungen für *Reforge* vorgestellt.

7.1 Verbesserung der Textbereinigungen

Ein wichtiger Aspekt bei der Bearbeitung von LaTeX-Dokumenten ist die Textbereinigung. Der derzeitige Ansatz filtert bestimmte Teile der LaTeX-Dateien heraus, um nur den wesentlichen Inhalt zu erhalten. Dies ist notwendig, um die Anzahl der Token für die nachfolgenden Verarbeitungsschritte nicht zu groß werden zu lassen. Allerdings werden dabei auch Codeabschnitte und andere potenziell wertvolle Inhalte herausgefiltert.

In Zukunft könnte eine verbesserte Textbereinigung entwickelt werden, die in der Lage ist, Codeabschnitte in LaTeX korrekt zu erkennen und entsprechend zu behandeln. Damit wäre sichergestellt, dass diese Inhalte nicht verloren gehen und gegebenenfalls in die Zusammenfassung aufgenommen werden können. Insbesondere für wissenschaftliche Arbeiten, die viele mathematische Formeln oder programmiertechnische Elemente enthalten, wäre dies von Vorteil, da so die Zusammenhänge im Text besser erhalten bleiben.

Darüber hinaus sollte die Textbereinigung flexibler gestaltet werden, so dass weitere, bisher ausgeschlossene Inhalte einbezogen werden können. Dies könnte zum Beispiel die Berücksichtigung von Kommentaren umfassen, die zwar derzeit als unwichtig erachtet werden, jedoch für die Interpretation des Textes von Bedeutung sein könnten.

Die Verbesserung der Textbereinigung würde dazu beitragen, eine genauere und vollständigere Zusammenfassung des Originaldokuments zu erstellen. Dies würde auch die Qualität der Berichte verbessern.

7.2 Hinzufügen des Literaturverzeichnisses

Ein weiterer Bereich, der in Zukunft verbessert werden sollte, ist das Literaturverzeichnis. Momentan muss das Literaturverzeichnis manuell hinzugefügt werden, da es nicht automatisch in die generierte Ausgabe übernommen wird. In einer zukünftigen Version der Anwendung sollte das Literaturverzeichnis korrekt erkannt und in das Ausgabeformat integriert werden. Eine automatische Übernahme des Literaturverzeichnisses würde den Generierungsprozess für den Benutzer vereinfachen.

7.3 Hinzufügen einer Datenbank

Ein möglicher Ansatz zur Verbesserung wäre die Integration einer Datenbank zur Speicherung der Daten. Derzeit werden die Daten nur temporär verarbeitet und nicht dauerhaft gespeichert. Eine Datenbank würde die langfristige Speicherung von Dateien, Metadaten, Bearbeitungsstatus und Benutzerinformationen ermöglichen. Wie Saake et al. betonen, ermöglichen Datenbanksysteme nicht nur die effiziente Speicherung und den Zugriff auf große Datenmengen, sondern gewährleisten auch die Datenunabhängigkeit und die Kontrolle von Mehrbenutzerzugriffen [Saake et al., 2019]. Darüber hinaus wird in ihrem Buch die Implementierung einer Datenbank detailliert beschrieben, welche als wertvolle Grundlage für die Realisierung dienen kann.

Außerdem hätte die Einführung einer Datenbank mehrere Vorteile. Zum einen könnte der Benutzer jederzeit auf frühere Berichte und Bearbeitungen zugreifen, was die Wiederverwendbarkeit der Ergebnisse begünstigt. Zum anderen könnte eine Datenbank die Verwaltung und Nachvollziehbarkeit der Benutzeraktionen unterstützen, dies wäre insbesondere bei einer größeren Anzahl von Benutzern wichtig. Die Wahl der Datenbanktechnologie wie MongoDB, SQL oder NoSQL würde von den spezifischen Anforderungen des Projekts abhängen.

Die Entwicklung eines konzeptionellen Datenmodells kann ebenfalls Gegenstand künftiger Arbeiten sein. Die Planung eines Informatikprojekts sollte gemäß Vetter systematisch von einer groben zur detaillierten Beschreibung erfolgen. Dabei ist es entscheidend, zunächst eine abstrakte Übersicht des Systems zu erstellen, bevor man sich den spezifischen Details widmet. Ein konzeptionelles Datenmodell bildet dabei die Grundlage für diese Vorgehensweise, da es eine klare Struktur und Orientierung bietet. [Vetter, 2013, S.15 ff.]

7.4 Berücksichtigung von Bildern

Ein weiterer Schritt zur Verbesserung der Anwendung könnte die Berücksichtigung von Bildern bei der Verarbeitung sein. Derzeit werden Diagramme, Abbildungen und andere visuelle Inhalte, die oft wichtige Informationen enthalten, herausgefiltert und fehlen somit in der generierten Ausgabe. Dies ist insbesondere für wissenschaftliche Arbeiten von Bedeutung, die häufig auf visuelle Darstellungen zur Veranschaulichung von komplexen Zusammenhängen angewiesen sind.

7.5 Erweiterung der Parameter im Frontend

Eine weitere Verbesserung könnte darin bestehen, dem Frontend mehr Parameter hinzuzufügen, um die Flexibilität der Benutzeroberfläche zu erhöhen. Beispielsweise könnte der Benutzer angeben, welche spezifischen Abschnitte des Dokuments in den Bericht aufgenommen werden sollen. Auf diese Weise könnte der Benutzer den technischen Bericht besser an seine Bedürfnisse anpassen, indem er zum Beispiel nur bestimmte Abschnitte wie die Einleitung, das Fazit oder ausgewählte Kapitel einbezieht. Diese Erweiterung würde dem Benutzer mehr Kontrolle über die generierten Berichte geben.

7.6 Asynchrone Verarbeitungen und Benachrichtigungen

Eine sinnvolle Erweiterung der Anwendung wäre die Option, den fertig generierten Bericht per E-Mail an den Benutzer zu senden. Diese Funktionalität würde es den Nutzern ermöglichen, die Webseite zu verlassen, anstatt auf die Fertigstellung des Berichts zu warten. Besonders bei längeren Verarbeitungszeiten könnte dies die Benutzererfahrung verbessern, da der fertige Bericht automatisch zugestellt wird, sobald die Generierung abgeschlossen ist.

Eine weitere Möglichkeit der asynchronen Verarbeitung wäre im Kontext einer Datenbank. In diesem Fall könnten die generierten Berichte für jeden Benutzer auf dem Benutzerkonto in der Datenbank gespeichert werden. Auf diese Weise muss der Benutzer auch nicht auf seinen Bericht warten, sondern kann den generierten Bericht später einfach in seinem Account abrufen.

7.7 Kostenlimitierung

Wenn die Anwendung öffentlich zugänglich gemacht wird, sollte im Backend eine Beschränkung implementiert werden, um die Nutzung zu kontrollieren. Diese Beschränkung soll verhindern, dass Nutzer unbegrenzt Dokumente generieren, da dies die Kosten für den Betreiber der Website erheblich erhöhen könnte. Eine solche Maßnahme ist wichtig, damit die Betriebskosten überschaubar bleiben und die Anwendung langfristig nachhaltig betrieben werden kann.

Eine Möglichkeit wäre, die Anzahl der Berichte, die pro Tag erstellt werden können, auf zwei zu begrenzen. Wenn der Benutzer jedoch mehr als zwei Berichte erstellen möchte, könnte eine Art Kreditsystem eingeführt werden. Mit diesem System könnte der Benutzer mehr Versuche für die Generierung erwerben. Dieses Kreditsystem wird beispielsweise auf Fotor.com verwendet, einer Website für die Erstellung von KI-Bildern. Die Abbildung A.1 im Anhang zeigt, wie die Option zum Kauf von Credits auf dieser Webseite aussieht. Darüber hinaus bietet Fotor die Alternative, täglich Credits zu erhalten, so dass der Kauf von Credits nicht zwingend erforderlich ist [Fotor, 2024]. Ein ähnliches System könnte für *Reforge* durchaus sinnvoll sein, um die Kosten überschaubar zu halten und eventuell einen kleinen Gewinn zu erzielen.

Kapitel 8

Fazit

Im Fazit wird die Arbeit abschließend zusammengefasst und die Erreichung der zuvor definierten Ziele reflektiert.

8.1 Zusammenfassung der Arbeit

Im Rahmen dieser Bachelorarbeit wurde die Webanwendung *Reforge* entwickelt und erfolgreich implementiert. Sie ist in der Lage, eingereichte Dokumente zusammenzufassen und in spezifischen Formaten wie IEEEtran oder dem OTH-Forschungsbericht zur Verfügung zu stellen. Besonderes Augenmerk wurde auf die Integration moderner LLMs zur Textzusammenfassung und -generierung sowie auf die Unterstützung mehrsprachiger Ausgaben in Deutsch und Englisch gelegt.

Die entwickelte Anwendung bietet Studierenden und Lehrenden eine Plattform zur effizienten Erstellung technischer Berichte. Die Implementierung umfasst sowohl die Verarbeitung von LaTeX-ZIP-Dateien als auch von DOCX-Dokumenten. Zudem ermöglicht die Integration von APIs, wie OpenAI und DeepL, die Textzusammenfassung und Übersetzung.

Die Ergebnisse der Arbeit zeigen, dass der Einsatz von *Reforge* den Zeit- und Arbeitsaufwand reduzieren kann. Das Tool bietet daher eine sinnvolle Anwendungsmöglichkeit und wird dementsprechend als nützlich bewertet. Gleichzeitig wurden die Grenzen der Automatisierung aufgezeigt, wie die Abhängigkeit von Token-Limits und die Notwendigkeit menschlicher Validierung. Diese Arbeit leistet somit einen Beitrag auf dem Gebiet der webbasierten Textzusammenfassung und der Entwicklung von KI-Werkzeugen für den akademischen Gebrauch.

8.2 Erreichung der Ziele

In diesem Abschnitt wird dargestellt, inwieweit die gesteckten Ziele und funktionalen Anforderungen des Projekts erreicht wurden. Dabei wird insbesondere auf die in Kapitel 4.1.1 definierten Anforderungen eingegangen und überprüft, ob diese vollständig

umgesetzt wurden. Im Kapitel 1.2 wurden drei Kernfragen gestellt, auf die nun Bezug genommen wird.

Die für die Webanwendung *Reforge* gewählte Architektur erfüllt alle Anforderungen an das System und ermöglicht die Erstellung von TechReports. In dieser Arbeit wurde gezeigt, wie ein LLM zur automatischen Textzusammenfassung erfolgreich integriert werden kann. Außerdem wurde die Qualität der generierten Berichte untersucht. Diese entsprechen den Anforderungen und bieten eine Grundlage für manuell erstellte Berichte, sowohl hinsichtlich der Struktur als auch der inhaltlichen Genauigkeit.

Alle in A1 und A2 definierten Anforderungen wurden erfüllt. Die Anwendung ist in der Lage, den gesamten Text der Dokumente zu erfassen und für die Generierung in Segmente zu zerlegen. Diese Segmente werden anschließend fehlerfrei wieder zusammengesetzt. Darüber hinaus funktioniert die Kommunikation mit der OpenAI API Schnittstelle. Die generierten Berichte spiegeln zudem den Inhalt des Originaltextes wider.

Die Anforderungen aus A3 DOCX- und LaTeX-Output sind ebenfalls vollständig implementiert. Die generierten Berichte werden korrekt im Word- oder LaTeX-Format ausgegeben, wobei spezifische Vorlagen wie IEEEtran oder der OTH-Forschungsbericht berücksichtigt werden.

Die in A4 Multi-lingual definierten Anforderungen werden ebenfalls erfüllt. Der Benutzer kann im Webinterface die gewünschte Sprache für den Bericht auswählen. Die Anwendung generiert den Bericht daraufhin entweder in Deutsch oder Englisch.

Alle übrigen Anforderungen der A5 sind ebenfalls in *Reforge* berücksichtigt. Die Benutzeroberfläche verfügt über alle geforderten Elemente und ist funktionsfähig. Sowohl der aktuelle Fortschritt als auch Fehlermeldungen der Anwendung werden dem Benutzer angezeigt. Darüber hinaus wurde auch die Funktion zum Herunterladen von Berichten implementiert.

Zusammenfassend kann festgestellt werden, dass alle gesteckten Ziele und definierten Anforderungen erreicht wurden. Die Anwendung ist nicht nur funktionsfähig, sondern ermöglicht auch die Zusammenfassung der Dokumente.

Literaturverzeichnis

- [Anslinger, 2021] Anslinger, J. (2021). Faire ki-(wie) geht das?
- [Banerjee et al., 2023] Banerjee, D., Singh, P., Avadhanam, A., and Srivastava, S. (2023). Benchmarking llm powered chatbots: Methods and metrics.
- [Bartel, 2013] Bartel, S. (2013). *Farben im Webdesign: Symbolik, Farbpsychologie, Gestaltung*. Springer-Verlag.
- [Beaird et al., 2020] Beaird, J., Walker, A., and George, J. (2020). *The principles of beautiful web design*. SitePoint Pty Ltd.
- [Chang et al., 2024] Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P. S., Yang, Q., and Xie, X. (2024). A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3).
- [Cyberlytics et al., 2023] Cyberlytics, Montell, B., Tassilo, K., Rebecca, K., Fabian, K., Illia, P., Fabian, S., and Natalie, S. (2023). Battlesprout - web application project. <https://github.com/cyberlytics/BattleSprout>. Zugegriffen: 17. November 2024.
- [De and De, 2017] De, B. and De, B. (2017). *API management*. Springer.
- [Dunning, 1994] Dunning, T. (1994). *Statistical identification of language*. Computing Research Laboratory, New Mexico State University Las Cruces.
- [Federspiel et al., 2023] Federspiel, F., Mitchell, R., Asokan, A., Umana, C., and McCoy, D. (2023). Threats by artificial intelligence to human health and human existence. *BMJ Global Health*, 8(5).
- [Fink et al., 2014] Fink, G., Flatow, I., Fink, G., and Flatow, I. (2014). Introducing single page applications. *Pro single page application development: Using backbone.js and asp.net*, pages 3–13.
- [Fisher et al., 2024] Fisher, W. M., Wagner, N., and Garrison, K. (2024). Large language model (llm) comparison research. Technical report, Acquisition Research Program.
- [Fotor, 2024] Fotor (2024). Fotor – online photo editing and graphic design tool. <https://www.fotor.com>. Zugegriffen: 30. Oktober 2024.

- [Fraiwan and Khasawneh, 2023] Fraiwan, M. and Khasawneh, N. (2023). A review of chatgpt applications in education, marketing, software engineering, and healthcare: Benefits, drawbacks, and research directions.
- [Friedl, 2009] Friedl, J. E. (2009). *Reguläre Ausdrücke*. O'Reilly Germany.
- [Gibbs, 2017] Gibbs, S. (2017). Google sibling waymo launches fully autonomous ride-hailing service. *The Guardian*, 7.
- [Hoffmann, 2022] Hoffmann, T. (2022). *Entwurf und Implementierung einer Progressiven Web App am Beispiel der technischen Transition einer Cloud/Lösung für webbasierte Datenbearbeitung und Reporting*. Bachelorarbeit, Ostbayerische Technische Hochschule Amberg/Weiden. In Kooperation mit der INSIGMA IT Engineering GmbH.
- [Jamwal, 2010] Jamwal, D. (2010). Analysis of software quality models for organizations. *International Journal of Latest Trends in Computing*, 1(2):19–23.
- [Japikse et al., 2020] Japikse, P., Grossnicklaus, K., and Dewey, B. (2020). *Introduction to TypeScript*, pages 413–468. Apress, Berkeley, CA.
- [Kleuker and Kleuker, 2013] Kleuker, S. and Kleuker, S. (2013). Anforderungsanalyse. *Grundkurs Software-Engineering mit UML: Der pragmatische Weg zu erfolgreichen Softwareprojekten*, pages 55–92.
- [Krug, 2018] Krug, S. (2018). *Don't make me think!: Web & Mobile Usability: Das intuitive Web*. MITP-Verlags GmbH & Co. KG.
- [Kumar et al., 2024] Kumar, A., Murthy, S. V., Singh, S., and Ragupathy, S. (2024). The ethics of interaction: Mitigating security threats in llms. *arXiv preprint arXiv:2401.12273*.
- [Lazuardy and Anggraini, 2022] Lazuardy, M. F. S. and Anggraini, D. (2022). Modern front end web architectures with react.js and next.js. *Research Journal of Advanced Engineering and Science*, 7(1):132–141.
- [Levi and Neumann, 2024] Levi, P. and Neumann, C. P. (2024). Vocabulary attack to hijack large language model applications.
- [Meta Platforms, Inc., 2024] Meta Platforms, Inc. (2024). React - a javascript library for building user interfaces. <https://react.dev>. Zugegriffen: 30. Oktober 2024.
- [Microsoft, 2024a] Microsoft (2024a). Microsoft 365 copilot für word. <https://copilot.cloud.microsoft/de-de/copilot-word#ID0EBBD=Erstellen>. Zugegriffen: 29. Oktober 2024.
- [Microsoft, 2024b] Microsoft (2024b). Microsoft copilot overview. <https://www.microsoft.com/de-de/microsoft-copilot/organizations>. Zugegriffen: 29. Oktober 2024.
- [Mili et al., 2004] Mili, H., Elkharraz, A., and Mcheick, H. (2004). Understanding separation of concerns. *Early aspects: aspect-oriented requirements engineering and architecture design*, pages 75–84.

- [Mittelbach et al., 2024] Mittelbach, F., Fischer, U., Carlisle, D., and Wright, J. (2024). Automatically producing accessible and reusable pdfs with latex. In *Proceedings of the ACM Symposium on Document Engineering 2024 (DocEng '24)*, page 4, New York, NY, USA. ACM.
- [Naumann, 2024] Naumann, J. (2024). *Setzen Sie auf gutes Webdesign*, pages 51–78. Springer Fachmedien Wiesbaden, Wiesbaden.
- [OpenAI, 2024a] OpenAI (2024a). Api pricing. <https://openai.com/api/pricing/>. Zugegriffen: 31. Oktober 2024.
- [OpenAI, 2024b] OpenAI (2024b). Chatgpt overview. <https://openai.com/chatgpt/overview/>. Zugegriffen: 31. Oktober 2024.
- [OpenAI, 2024c] OpenAI (2024c). Quickstart guide for text generation. <https://platform.openai.com/docs/guides/text-generation/quickstart>. Zugegriffen: 31. Oktober 2024.
- [OpenAI, 2024d] OpenAI (2024d). What are tokens and how to count them? <https://help.openai.com/en/articles/4936856>. Zugegriffen: 30. Oktober 2024.
- [OpenJSFoundation, 2024] OpenJSFoundation (2024). Multer - node.js middleware for handling multipart/form-data. <https://expressjs.com/en/resources/middleware/multer.html>. Zugegriffen: 04. November 2024.
- [Patton, 2001] Patton, M. Q. (2001). Evaluation, knowledge management, best practices, and high quality lessons learned. *The American journal of evaluation*, 22(3):329–336.
- [Preim and Dachzelt, 2015] Preim, B. and Dachzelt, R. (2015). *Interaktive Systeme: Band 2: User Interface Engineering, 3D-Interaktion, Natural User Interfaces*. Springer-Verlag.
- [Rau, 2016] Rau, K.-H. (2016). *Agile objektorientierte Software-Entwicklung*, volume 1, 3.3. Springer.
- [Rosenfeld and Lazebnik, 2024] Rosenfeld, A. and Lazebnik, T. (2024). Whose llm is it anyway? linguistic comparison and llm attribution for gpt-3.5, gpt-4 and bard. *arXiv preprint arXiv:2402.14533*.
- [Rumpe and Rumpe, 2004] Rumpe, B. and Rumpe, B. (2004). Sequenzdiagramme. *Modellierung mit UML: Sprache, Konzepte und Methodik*, pages 241–260.
- [Saake et al., 2019] Saake, G., Sattler, K.-U., and Heuer, A. (2019). *Datenbanken. Implementierungstechniken*. MITP-Verlags GmbH & Co. KG.
- [Schotter, 2022] Schotter, T. (2022). *Evaluation von Qubes OS für den Einsatz in Klein- und Mittelständischen SW/Entwicklungsunternehmen*. Bachelorarbeit, Ostbayerische Technische Hochschule Amberg/Weiden.
- [Sundar Pichai, 2023] Sundar Pichai, D. H. (2023). Introducing gemini, googles next-generation ai. <https://blog.google/technology/ai/google-gemini-ai>. Zugegriffen: 29. Oktober 2024.

- [Vetter, 2013] Vetter, M. (2013). *Aufbau betrieblicher Informationssysteme: mittels konzeptioneller Datenmodellierung*. Springer-Verlag.
- [Vyas, 2022] Vyas, R. (2022). Comparative analysis on front-end frameworks for web applications. *International Journal for Research in Applied Science and Engineering Technology*, 10(7):298–307.
- [Wang et al., 2023] Wang, J., Liu, Z., Zhao, L., Wu, Z., Ma, C., Yu, S., Dai, H., Yang, Q., Liu, Y., Zhang, S., et al. (2023). Review of large vision models and visual prompt engineering. *Meta-Radiology*, page 100047.
- [White et al., 2023] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt.
- [Yadav et al., 2022] Yadav, D., Desai, J., and Yadav, A. K. (2022). Automatic text summarization methods: A comprehensive review. <https://arxiv.org/abs/2204.01849>.

Abbildungsverzeichnis

3.1	Extractive and Abstractive text summarizer [Yadav et al., 2022, S.3] . . .	11
4.1	Wireframe für die Upload-Seite von Reforge	18
4.2	Wireframe für die Download-Seite von Reforge	18
4.3	Systemarchitektur von Reforge	20
4.4	Sequenzdiagramm des Upload-Prozesses	22
5.1	Oberflächendesign der Upload-Seite	39
5.2	Oberflächendesign der Download-Seite	40
5.3	Fehleranzeige auf der Upload-Seite	41
5.4	Das zusätzliche Eingabefeld für die LaTeX-Hauptdatei	41
6.1	OpenAI Kosten von August	47
6.2	OpenAI Token Usage von August	47
6.3	Die mobile Ansicht der Upload-Seite von <i>Reforge</i>	50
6.4	ZIP-Test-Prompt an ChatGPT	51
6.5	ZIP-Test-Antwort von ChatGPT	51
6.6	Copilot Datei Einschränkungen	53
6.7	Copilot Textlängen Limitierung	53
6.8	Gemini Datei Einschränkungen	54
6.9	Quillbot Textlängen Limitierung	54
6.10	Quillbot Einstellugen für die Zusammenfassung	55
A.1	Pop-up Fenster von Fotor.com für den Creditkauf	68
A.2	Reforge LaTeX-Ausgabetest auf Basis dieser Bachelorarbeit, Seite eins .	69
A.3	Reforge LaTeX-Ausgabetest auf Basis dieser Bachelorarbeit, Seite zwei .	70
A.4	Reforge DOCX-Ausgabetest auf Basis dieser Bachelorarbeit, Seite eins .	71
A.5	Reforge DOCX-Ausgabetest auf Basis dieser Bachelorarbeit, Seite zwei .	72
A.6	ChatGPT Test 2: Prompt; Testdokument [Schotter, 2022]	73
A.7	ChatGPT Test 2: Ergebnis; Testdokument [Schotter, 2022]	74
A.8	ChatGPT Test 3: Prompt; Testdokument [Hoffmann, 2022]	75
A.9	ChatGPT Test 3: Ergebnis; Testdokument [Hoffmann, 2022]	76
A.10	ChatGPT Test 4: Prompt; Testdokument [Schotter, 2022]	77

Tabellenverzeichnis

2.1	Musterkategorie: Zeichenklassen	6
2.2	Musterkategorie: Quantoren	6
2.3	Musterkategorie: Pattern Modifiers	6
2.4	Musterkategorie: Wortgrenzen	7

Anhang A

Bilder

A.1 Creditkauf auf Fotor

Erwerben Sie Kreditpunkte,
schalten Sie Ihre grenzenlose
Kreativität frei! 🌿

Monatlich ☒ Jährlich

☒ 200 Kreditpunkte/Monat × 12
€3.33/monat €39.99/jahr

☐ 500 Kreditpunkte/Monat × 12
€6.66/monat €79.99/jahr

☐ 1000 Kreditpunkte/Monat × 12
€12.08/monat €144.99/jahr

Hast du einen Aktionscode?

Weiter

EUR



Abbildung A.1: Pop-up Fenster von Fotor.com für den Creditkauf

A.2 Reforge LaTeX-Ausgabetest

Reforge

Natalie Stricker

*Fakultät Elektrotechnik, Medien und Informatik
Ostbayerische Hochschule Amberg-Weiden
Amberg, Deutschland*

I. INTRODUCTION

The chapter discusses the increasing significance of artificial intelligence in various domains, such as personalized product recommendations, autonomous vehicles, and disease diagnosis. It explores the benefits and concerns, including the potential impact on traditional roles like taxi drivers. The text stresses the importance of AI as a supportive tool that must be critically evaluated due to risks like misinformation and biases. Additionally, it highlights the automation potential for repetitive tasks, especially in academia and technical fields. The focus is on developing a web-based report generator named "Reforge" to streamline the process of analyzing and summarizing technical reports. The chapter outlines the structure of the thesis, covering theoretical foundations, research overview, design concepts, implementation details, results, future enhancements, and concluding remarks.

II. BASICS

Chapter "Fundamentals" explains the necessary basics for automating text generation in this work. The focus is on understanding what a model is and how it functions. A model is a large language model capable of understanding and generating texts. Models are trained on large text datasets like books, articles, and websites to learn words, expressions, and sentence structures to maintain coherence and relevance. The work uses an OpenAI model known for its capabilities in integrating with applications. Different model sizes have varying levels of accuracy and costs, with the work using the -3.5-Turbo model for its balance of performance and cost-effectiveness. The text units for models are tokens, and costs are based on token usage. Prompts guide model outputs, prompting for specific responses such as length or format. Regular expressions are also discussed for pattern recognition in text manipulation. The section also covers web design basics for user-friendly interfaces, including responsive design and visual hierarchy principles. TypeScript is chosen as the programming language for its additional features like typing and object-oriented programming, enhancing code reliability and maintenance. React, a frontend framework, aids in building single-page applications dynamically loading content. Communication between frontend and backend is facilitated by APIs, with RESTful APIs using HTTP protocols for resource management through CRUD operations. The chapter emphasizes the importance of self-descriptive data for proper handling of API requests.

III. STATE OF RESEARCH AND RELATED WORK

This section provides an overview of current research on technologies essential for understanding and implementing the project, including text summarization, LaTeX, and web development. It discusses different types of summarization methods such as extractive and abstractive, highlighting the challenges and capabilities of each. The potential of AI in generating summaries is explored, noting the need for human verification due to issues like hallucinations in the output. The text also touches on the security concerns related to AI models like prompts for AI text generation. LaTeX is described as a powerful typesetting system widely used in academic circles for document creation. Lastly, the advancements in web development, particularly with JavaScript frameworks like React, are discussed, along with related web applications that generate summaries similar to the project at hand. The project aims to offer an innovative solution for creating technical reports from LaTeX or other document formats, providing flexibility for users in customizing outputs.

IV. CONCEPTION AND DESIGN OF REFORGE

The text discusses the essential step of requirement analysis in software project development using Kleuker's model. Functional requirements are classified with modal verbs like "must," "should," and "will." Acceptance criteria are defined for each requirement to ensure proper implementation. The project involves functions for text extraction, summarization, and conversion, with specific criteria for each task. The application utilizes OpenAI models for text summarization due to their precision and ease of integration. The system architecture includes a frontend, backend, and external systems like OpenAI and DeepL for text processing and translation. Data storage is temporary, managed by middleware like Multer. The upload process is illustrated through a sequence diagram showing interactions between components.

V. IMPLEMENTATION OF REFORGE

The backend of Reforge is responsible for processing uploaded files and generating technical reports. The communication between the frontend and backend is detailed, involving RESTful API calls for data exchange and report download. The frontend triggers the backend process by uploading a LaTeX-ZIP folder, filling in parameters, and clicking a button. The backend processes the file, updates the response header, and sends the report back to the frontend. Important libraries used in backend development include ADM-ZIP for ZIP file

Abbildung A.2: Reforge LaTeX-Ausgabetest auf Basis dieser Bachelorarbeit, Seite eins

handling, Bottleneck for rate limiting, CORS for enabling cross-origin resource sharing, and others for file upload, Word document manipulation, and more. The backend processes LaTeX-Zip files by detecting the main file, extracting chapter order, and generating a report for download. Language detection is done to ensure the report language matches user selection. OpenAI is used for text summarization. The frontend consists of upload and download pages, following Separation of Concerns principle. Various libraries like React, React Router, and Jest are used for frontend development. The design focuses on simplicity, clear information hierarchy, and usability. The upload page allows users to input additional details for LaTeX files, while the download page facilitates file retrieval based on passed state parameters. The process involves React Router's navigation and state management. The download function creates a temporary link for file download based on specified URL, title, and format. Overall, the communication and interaction between frontend and backend facilitate seamless report generation and download for Reforge users.

VI. EVALUATION AND RESULTS

The chapter "Evaluation and Results" discusses the evaluation and results of a project focusing on configuration management, API usage costs, report quality, mobile usability, and comparison with existing tools. It also covers the process of generating keys for OpenAI and DeepL, along with integrating them into the application. The process of setting up and running the application locally is detailed, involving key generation and configuration settings. The costs and quality of the OpenAI and DeepL services are analyzed, along with a comparison with other tools like Chat, Copilot, Gemini, and QuillBot. Lessons learned from the project are summarized as valuable insights for future endeavors.

VII. PROJECT OUTLOOK FROM REFORGE

In this chapter on "Reforge Project Outlook," potential expansions and improvements are discussed, focusing on text cleanup for LaTeX documents. Suggestions include enhancing text cleanup to preserve code sections, improving handling of content like comments, automating integration of the bibliography, implementing a database for data storage and user actions tracking, considering inclusion of images in processing, adding user-defined parameters to the frontend, enabling email delivery of generated reports, and managing document generation limits through a credit system to control costs. These improvements aim to enhance user control, efficiency, and sustainability of the application.

VIII. CONCLUSION

The conclusion summarizes the work and reflects on achieving the defined goals. A web application was developed and successfully implemented in this bachelor thesis. It can summarize submitted documents and provide them in specific formats like IEEEtran or the OTH research report. Modern techniques for text summarization and generation, as well as

multilingual support in German and English, were integrated. The application offers a platform for efficient creation of technical reports for students and teachers. It processes LaTeX-ZIP files and Word documents, and integrates tools like OpenAI and DeepL for text summarization and translation. The study shows that such a tool can reduce time and effort, but also highlights limitations like token limits and the need for human validation. The work contributes to text summarization and academic tool development. It confirms meeting project goals and functional requirements, with attention to architecture, text summarization integration, and report quality. All defined requirements have been fulfilled, including capturing document text, generating segments error-free, and supporting various output formats. The application also meets multilingual requirements and has a functional user interface. In conclusion, all goals and requirements were achieved, making the application operational for document processing and summarization.

IX. PICTURES

Summary: The chapter "Images" is discussed or focused on.

X. CODE EXTRACTS

Summary: The chapter is about code excerpts.

XI. BIBLIOGRAPHY

This report was generated from the uploaded file using OpenAI. All sources must be added manually before publication. Name of the file: BT2024StrickerNatalieThesis.zip

Abbildung A.3: Reforge LaTeX-Ausgabetest auf Basis dieser Bachelorarbeit, Seite zwei

A.3 Reforge DOCX-Ausgabetest

Reforge

Natalie Stricker
Fakultät Elektrotechnik, Medien und Informatik
Ostbayerische Technische Hochschule Amberg-Weiden
Amberg, Deutschland

Einleitung

Die Einleitung behandelt die zunehmende Bedeutung von KI in verschiedenen Lebensbereichen, wie autonome Fahrzeuge und Bildung. Es wird diskutiert, wie KI unterstützend eingesetzt werden kann, aber auch potenzielle Probleme wie Bias und Abhängigkeit werden angesprochen. Die Arbeit zielt darauf ab, einen webbasierten Berichtsgenerator namens zu entwickeln, um Abschlussarbeiten zu analysieren und zusammenzufassen. Die Arbeit ist in Kapitel unterteilt, die von theoretischen Grundlagen über Forschungsstand bis zur Implementierung und Evaluierung führen.

Grundlagen

In dem Kapitel "Grundlagen" geht es um die automatisierte Textgenerierung und die Verwendung eines Sprachmodells, insbesondere des -3.5-Turbo Modells von OpenAI. Es wird erklärt, wie das Modell trainiert wird und wie es auf Benutzereingaben reagiert. Des Weiteren werden reguläre Ausdrücke und deren Anwendung zur Mustererkennung in Texten erläutert. Im Hinblick auf Webdesign wird über die Gestaltung von Webseiten für eine benutzerfreundliche Erfahrung gesprochen, einschließlich responsivem Webdesign und visueller Hierarchie. Außerdem wird die Wahl von TypeScript für die Programmierung und von React als Frontend-Framework erklärt, sowie die Verwendung von APIs und RESTful Web-Services zur Kommunikation zwischen Frontend und Backend.

Forschungsstand und verwandte Arbeiten

Der Abschnitt beschreibt den aktuellen Stand der Technologien, die für das Verständnis und die Umsetzung des Projekts wichtig sind, wie Textzusammenfassung, LaTeX und Web-Entwicklung. Es wird erwähnt, dass Textzusammenfassung bei einer Kompressionsrate von 15-30% am besten funktioniert und extraktive sowie abstrakte Zusammenfassungsmethoden verwendet werden können. Es wird diskutiert, dass automatisierte Zusammenfassungen leistungsfähig sind, aber sorgfältig überprüft werden müssen, um Halluzinationen zu vermeiden. Es wird auch darauf hingewiesen, dass automatisierte Texte noch nicht mit menschlichem Schreiben mithalten können. Die Sicherheit von Textgenerierungssystemen wie s ist wichtig, da diese anfällig für Manipulationen sind. LaTeX wird als leistungsstarkes Textsatzsystem für wissenschaftliche Dokumente erwähnt, das präzise Kontrolle über das Layout und die Formatierung bietet. Zudem wird auf die Integration von Copilot in erwähnt, um den Schreibprozess zu unterstützen. Die Bedeutung der Frontend-Entwicklung mit JavaScript-Frameworks wie React wird betont. Schließlich wird das vorliegende Projekt als innovative Lösung zur Generierung technischer Berichte aus LaTeX- oder -Dokumenten dargestellt.

Konzeption und Entwurf von Reforge

Die Anforderungsanalyse für das Softwareprojekt Reforge erfolgt nach dem Modell von Kleuker. Funktionale Anforderungen werden in "muss", "soll" und "wird" klassifiziert, wobei "muss" essentiell für den Projekterfolg ist. Akzeptanzkriterien werden definiert, um Anforderungen genauer zu bestimmen. Es gibt Anforderungen zur Textextraktion, inneren Struktur, Zusammenfassungsgenerator, Textaufteilung, Schnittstelle für OpenAI, Exportfunktionen, Sprachauswahl, Benutzeroberfläche und Fehlerbehandlung. Nicht-funktionale Anforderungen werden nach dem FURPS-Modell kategorisiert. Ein Wireframe und die Wahl des OpenAI-Modells sind wichtige Schritte. Die Systemarchitektur basiert auf RESTful Kommunikation und externen Systemen wie OpenAI und DeepL. Eine Datenbank wird bewusst nicht verwendet, und die Speicherung erfolgt temporary. Ein Sequenzdiagramm visualisiert den Datei-Upload-Prozess.

Implementierung von Reforge

Das Backend von Reforge verarbeitet hochgeladene Dateien und generiert technische Berichte. Die Kommunikation zwischen Frontend und Backend für den Datenaustausch und den Berichtsdownload erfolgt über einen RESTful API-Request. Der Ablauf beginnt, wenn der Nutzer im Frontend einen LaTeX-ZIP-Ordner hochlädt und auf "Start the generation" klickt. Der Code im Frontend ruft die RESTful API auf, um Dateien

Abbildung A.4: Reforge DOCX-Ausgabetest auf Basis dieser Bachelorarbeit, Seite eins

hochzuladen. Das Backend verarbeitet die Datei und fügt die Informationen in die Antwort ein, die dann an das Frontend gesendet wird. Nach erfolgreicher Verarbeitung erhält der Benutzer einen Link zum Herunterladen des Berichts. Die Backend-Entwicklung basiert auf wichtigen Bibliotheken wie ADM-ZIP für ZIP-Dateien, Bottleneck für Anforderungsratekontrolle, und Express für Backend-Entwicklung. Weitere Bibliotheken werden für Dateiverwaltung, Word-Dokumenterstellung und Dateiuploads verwendet. Die LaTeX-Zip-Verarbeitung beinhaltet die Suche nach Hauptdateien, die Extraktion von Teiltexten und die Bereinigung für die AI-Zusammenfassung. Die AI-Generierung erfolgt über OpenAI, mit Stopwörtern zur Spracherkennung. Zudem wird DeepL für Kapitelübersetzungen genutzt. In der Frontend-Entwicklung werden Bibliotheken für React-Tests, Rendering und Routing verwendet. Die Benutzeroberfläche ist klar strukturiert, mit verschiedenen Seiten für Upload und Download, und bietet eine einfache, benutzerfreundliche Nutzung. Der Dateidownload-Prozess wird durch Datenübergabe und React-Hooks realisiert, um die Datei herunterzuladen.

Evaluation und Ergebnisse

Das Kapitel "Evaluation und Ergebnisse" beschreibt die Bewertung des Konfigurationsmanagements, die Kostenanalyse der API-Nutzung, die Qualität der Berichte, den Vergleich mit anderen Tools und wichtige Erkenntnisse aus dem Entwicklungsprozess. Es werden Anleitungen zur Schlüsselgenerierung für OpenAI und DeepL gegeben. Die Anwendung kann lokal betrieben werden und erfordert die Konfiguration von Umgebungsvariablen. Es wird auch ein Vergleich mit anderen Tools wie Chat, Copilot, Gemini und QuillBot durchgeführt, wobei Reforge als benutzerfreundlicher und konsistenter hervorgeht. Verschiedene Kostenmodelle und Funktionalitäten der genutzten APIs werden aufgezeigt, sowie Lessons Learned aus dem Projekt präsentiert für zukünftige Projekte.

Projektausblick von Reforge

In dem Kapitel "Projektausblick von Reforge" werden mögliche Erweiterungen und Verbesserungen für LaTeX-Dokumente vorgestellt. Es wird diskutiert, wie die Textbereinigung optimiert werden kann, um wichtige Codeabschnitte nicht zu filtern. Eine bessere Textbereinigung könnte auch Kommentare einbeziehen und das Literaturverzeichnis automatisch in die Ausgabe integrieren. Die Einführung einer Datenbank zur langfristigen Speicherung von Daten wird vorgeschlagen. Weitere Verbesserungen könnten die Berücksichtigung von Bildern, mehr Flexibilität im Frontend und die Möglichkeit des Berichtsversands per E-Mail sein. Es wird auch darauf hingewiesen, dass bei einer öffentlichen Nutzung der Anwendung Beschränkungen implementiert werden müssen, um Betriebskosten zu kontrollieren.

Fazit

Im Fazit der Bachelorarbeit wird die Entwicklung einer Webanwendung zur automatischen Textzusammenfassung und -generierung in verschiedenen Formaten wie IEEEtran oder dem OTH-Forschungsbericht zusammengefasst. Die Anwendung unterstützt mehrsprachige Ausgaben in Deutsch und Englisch und bietet eine Plattform für Studierende und Lehrende zur schnellen Erstellung technischer Berichte. Sie integriert moderne Sprachmodelle wie OpenAI und DeepL zur Textzusammenfassung und Übersetzung. Die Ergebnisse zeigen, dass die Nutzung des Generators den Zeit- und Arbeitsaufwand verringern kann, jedoch auch menschliche Validierung erfordert. Alle definierten Ziele und Anforderungen des Projekts wurden erfüllt, wodurch die Anwendung effizient und funktionsfähig ist.

Bilder

In dem Abschnitt "Bilder" wird über das Thema Bilder gesprochen.

Codeauszüge

Der Abschnitt "Codeauszüge" in einem Text oder Dokument wird thematisiert.

Literaturverzeichnis

Dieser Bericht wurde aus der hochgeladenen Datei mit OpenAI generiert.
Alle Quellen müssen vor einer Veröffentlichung noch manuell ergänzt werden.
Name der Datei: BT2024StrickerNatalieThesis.zip

Abbildung A.5: Reforge DOCX-Ausgabetest auf Basis dieser Bachelorarbeit, Seite zwei

A.4 ChatGPT Test 2

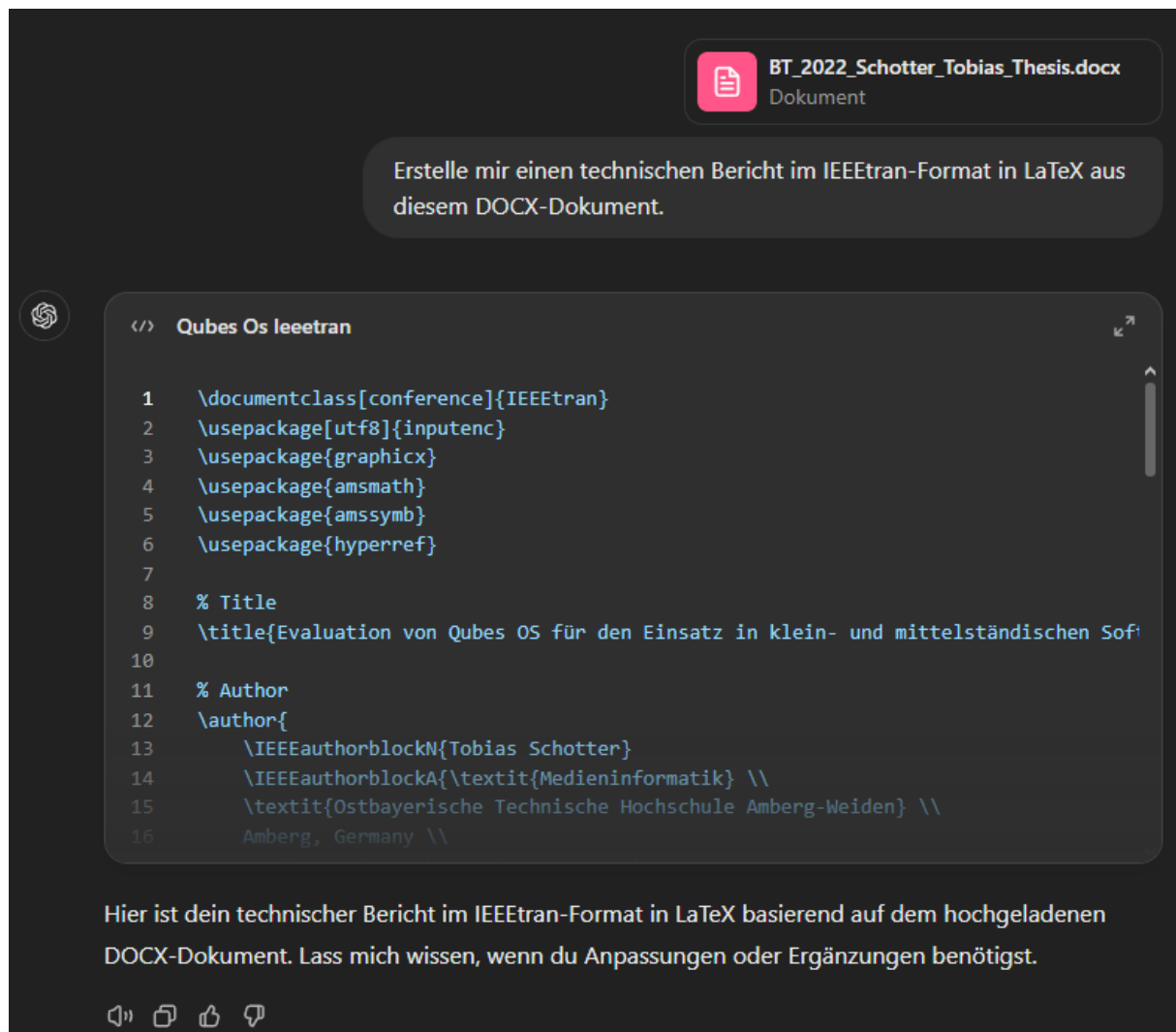


Abbildung A.6: ChatGPT Test 2: Prompt; Testdokument [Schotter, 2022]

Evaluation von Qubes OS für den Einsatz in klein- und mittelständischen Software-Entwicklungsunternehmen

Tobias Schotter
Medieninformatik
Ostbayerische Technische Hochschule Amberg-Weiden
Amberg, Germany
Email: tobias.schotter@example.com

Abstract—Hardly any company in Germany is spared from an increasing number of cyber-attacks. Cybercrime offences now cause damage of around 220 billion euros per year in Germany alone. The Qubes OS operating system offers an approach to protect computers further than conventional IT security measures by isolating processes, applications, and hardware through compartmentalization, for example in the form of virtual machines. Qubes follows the practical approach of limiting and containing damage to separate valuable data from risky activities.

In the context of this thesis, concepts are evaluated on how small and medium-sized software development companies can integrate Qubes OS into their everyday use. Typical usage scenarios of software engineers are examined and compared with a native Windows 10 operating system. Based on the evaluations, the necessary changes in the everyday cycle of a software developer are presented and compared with the known "best practices".

I. EINLEITUNG

Heutige IT-Systeme sind einer Vielzahl von Angriffen ausgesetzt, sowohl Firmencomputer als auch private Systeme sind betroffen. Anti-Viren-Software und Firewalls sind typische Gegenmaßnahmen, jedoch verschaffen sich Angreifer immer wieder Zugang zu Unternehmenssystemen und verursachen erhebliche Schäden. Allein in Deutschland belaufen sich die geschätzten Schäden durch Cyberkriminalität auf circa 220 Milliarden Euro pro Jahr [?].

Qubes OS stellt ein Open-Source-Betriebssystem dar, das durch die Isolation von Prozessen, Anwendungen und Hardware ein hohes Maß an Sicherheit bietet. Anwendungen werden in separaten und teilweise temporären virtuellen Maschinen (VMs) ausgeführt, wodurch die Angriffsfläche verringert wird. Diese Arbeit untersucht, wie Qubes OS in klein- und mittelständischen Software-Entwicklungsunternehmen sinnvoll eingesetzt werden kann.

II. METHODIK

Die Methodik dieser Arbeit besteht aus einer Evaluation der Nutzung von Qubes OS in einem realistischen Szenario von kleinen und mittleren Softwareunternehmen. Dazu werden typische Nutzungsszenarien für Software-Ingenieure untersucht

und mit der Nutzung eines nativen Windows 10 Betriebssystems verglichen. Es wird eine Gegenüberstellung von Sicherheitsmechanismen herkömmlicher IT-Sicherheitsmaßnahmen und den spezifischen Mechanismen von Qubes OS vorgenommen.

III. INFORMATIONSSICHERHEIT

A. Relevanz von Informationssicherheit

Die meistgenutzten Betriebssysteme wie Windows und macOS sind zwar beliebt aufgrund ihrer Benutzerfreundlichkeit, weisen jedoch erhebliche Sicherheitsprobleme auf. Malware kann über E-Mails oder Websites in Systeme eindringen und dabei sensible Informationen gefährden. Es ist daher wichtig, alternative Sicherheitslösungen wie Qubes OS zu betrachten, das durch die Isolation von Prozessen eine hohe Sicherheit bietet.

B. Sicherheitsmechanismen von Qubes OS

Qubes OS verfolgt den Ansatz der Kompartimentierung, bei dem verschiedene Tätigkeiten in isolierten virtuellen Maschinen, den sogenannten Qubes, getrennt werden. Dies bedeutet, dass eine kompromittierte VM keine anderen VMs beeinträchtigen kann. Beispielsweise können Weblinks in sogenannten DisposableVMs, die sich nach Beendigung selbst zerstören, geöffnet werden, wodurch die Gefahr einer Infektion des Gesamtsystems minimiert wird.

IV. ANALYSE VON NUTZUNGSSZENARIEN

A. Administrative Nutzungsszenarien

Die Installation von Qubes OS erfordert spezifische Hardwareanforderungen und die Aktivierung von IOMMU-basierter Virtualisierung. Software kann über TemplateVMs installiert und auf mehrere AppVMs verteilt werden, wodurch eine effiziente Administration ermöglicht wird. Ein weiteres Nutzungsszenario ist die Installation einer Windows-VM innerhalb von Qubes OS, um Windows-spezifische Anwendungen auszuführen.

Abbildung A.7: ChatGPT Test 2: Ergebnis; Testdokument [Schotter, 2022]

A.5 ChatGPT Test 3

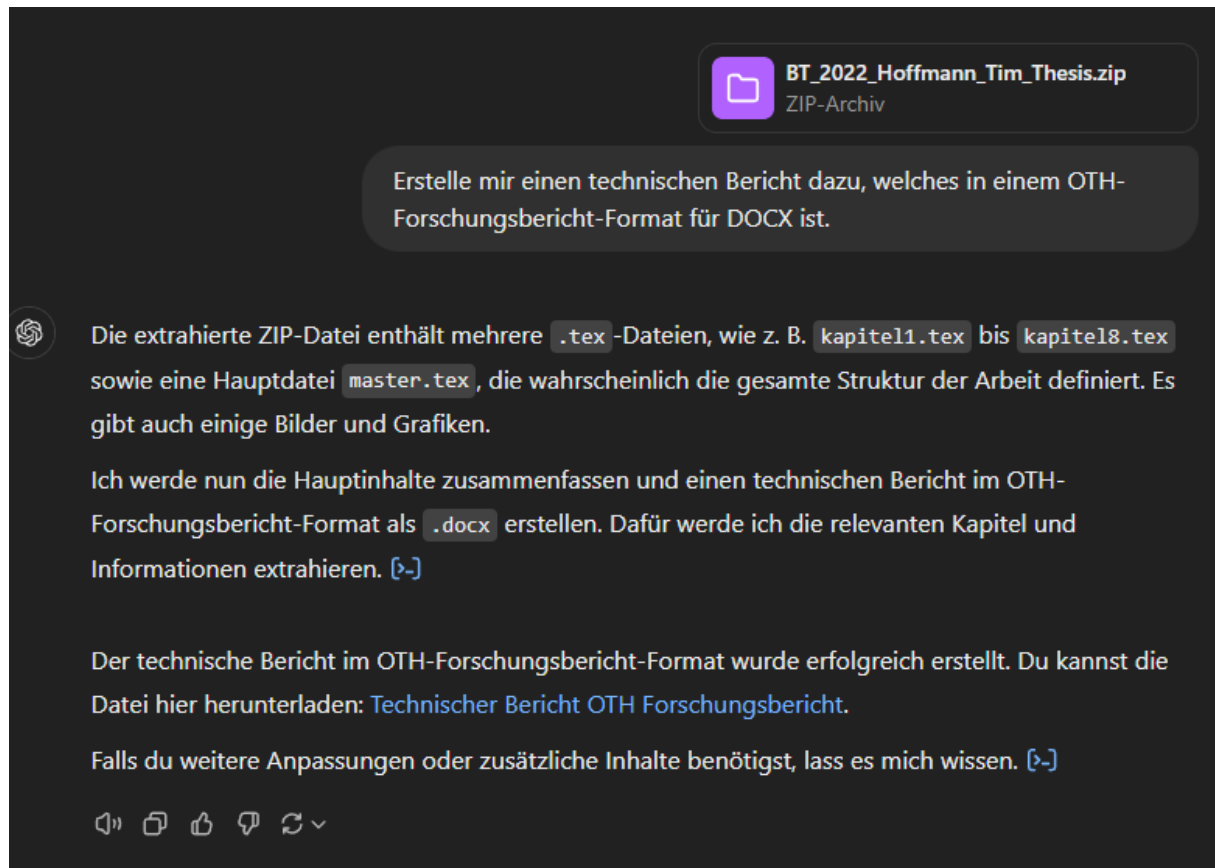


Abbildung A.8: ChatGPT Test 3: Prompt; Testdokument [Hoffmann, 2022]

Technischer Bericht

Kapitel1

\chapter{Einleitung}

Dieses Kapitel dient der thematischen Einführung in die Arbeit. Es werden die Ziele der Arbeit definiert und das methodische Vorgehen festgelegt.

\section{Thematische Einführung}

Die Bachelorarbeit wird in Zusammenarbeit mit dem Unternehmen INSIGMA IT Engineering GmbH verfasst. Das Unternehmen hat seine Zentrale in Frechen, westlich von Köln. Die Abteilung für Softwareentwicklung ist in den Geschäftsfeldern Automotive, Cloud Services, Krankenversicherungen und MIA® tätig.

Die Bachelorarbeit ist im Geschäftsfeld MIA® angesiedelt. MIA® bezeichnet ein von der INSIGMA entwickeltes Konzept, um individuelle Anforderungen auf Basis der zugrunde liegenden Daten und Prozesse schnell und unkompliziert mit Hilfe eines technischen Frameworks webbasiert umzusetzen. Es besteht aus kundenspezifischen und allgemein verwendbaren Modulen wie zum Beispiel Projektmanagement, Arbeitszeiterfassung und CMS-Funktionalitäten. Die Seiten werden bei jedem Aufruf unter Berücksichtigung eing

Kapitel2

\chapter{Grundlagen}

In diesem Kapitel werden zunächst die wichtigsten Grundlagen und Technologien beschrieben, welche in dieser Arbeit zum Einsatz kommen werden. Des Weiteren werden die zugrunde liegenden Rahmenbedingungen erläutert. Dazu werden im ersten Schritt die Grundlagen von Web Apps, Native Apps und Progressive Web Apps erläutert. Im Anschluss wird das MVC-Pattern beschrieben, nach welchem die Cloud-Lösung MIA® entwickelt wurde. Abschließend wird auf MIA® und die technischen Merkmale eingegangen.

\section{Applikation}

Der Begriff App steht für Applikation, ist eine Anwendungssoftware, welche für einen bestimmten Zweck programmiert wurde und sich durch eine Reihe an Funktionen auszeichnet. Da es sich bei einer PWA um eine Webapp mit Funktionen einer nativen App handelt, wird im Folgenden auf die drei Apptypen eingegangen und ihre Merkmale und Eigenschaften beschrieben.

\subsection{Native Applikation}

Unter einer nativen Applikation oder auch nativen Anwendung wird ein Program

Abbildung A.9: ChatGPT Test 3: Ergebnis; Testdokument [Hoffmann, 2022]

A.6 ChatGPT Test 4

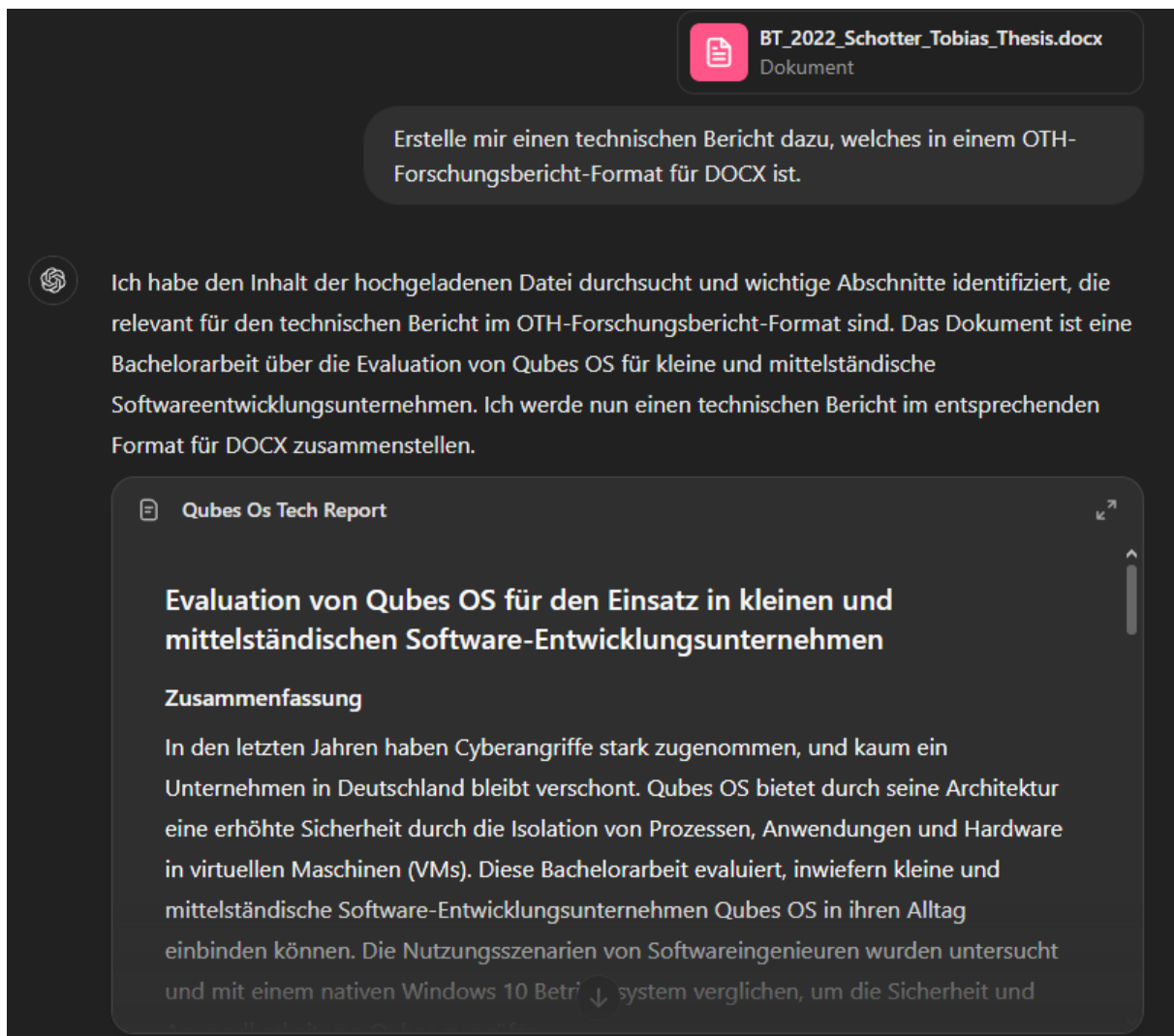


Abbildung A.10: ChatGPT Test 4: Prompt; Testdokument [Schotter, 2022]

Anhang B

Codeauszüge

B.1 Text-Filterungen

```

1  export function cleanLatexText(latexContent: string): string {
2      // lösche alles vor \chapter{
3      const chapterStartIndex = latexContent.search(/\chapter{/);
4      if (chapterStartIndex !== -1) {
5          latexContent = latexContent.substring(chapterStartIndex);
6      }
7
8      // Regex für commands
9      const commandRegex = /\[a-zA-Z]+\{[^}]*\}|\[a-zA-Z]+\[[^\]]*\]|\[a-zA-Z]+\;/g;
10     // Regex für kommentare
11     const commentRegex = /%.*$/gm;
12     // Regex für begin & end
13     const beginRegex = /\begin\{[^}]*\}\[s\S]*?\end\{[^}]*\}/g;
14     // Regex für whitespace
15     const whitespaceRegex = /\s+/g;
16
17     let filteredContent = latexContent
18         // Entfernt Kommentare
19         .replace(commentRegex, '')
20         // Entfernt Begin/End
21         .replace(beginRegex, '')
22         // Entfernt Commands außer Chapter
23         .replace(commandRegex, (match) => {
24             return match.startsWith('\chapter{') ? match : '';
25         })
26         // Entfernt whitespace
27         .replace(whitespaceRegex, ' ')
28         .trim();
29
30     return filteredContent;
31 }

```

Listing 24: Funktion für die LaTeX-Text-Filterung


```
1 export function removeSpecialChars(section: string): string {  
2   // entfernt '1.' '2.' etc. am anfang von einer section und special chars  
3   return section.replace(/\\section\\{\\d+\\.s*/g, '\\section{').replace(/[#%_~$]/g, '');  
4 }
```

Listing 25: Funktion für die SpecialChar-Filterung

B.2 DOCX-Dokument Generierung

```
1  async function PutDocxStyle(title: string, author: string, content: string) {
2  const doc = new Document({
3  sections: [
4  {
5  properties: {},
6  children: [
7  // Titel
8  new Paragraph({
9  text: title,
10 heading: HeadingLevel.TITLE,
11 alignment: AlignmentType.CENTER,
12 }),
13 // Leerzeile
14 new Paragraph({ text: "" }),
15 // Autor
16 new Paragraph({
17 alignment: AlignmentType.CENTER,
18 children: [
19   new TextRun(author),
20   new TextRun({ text: 'Fakultät Elektrotechnik, Medien und Informatik', break: 1 }),
21   new TextRun({ text: 'Ostbayerische Technische Hochschule Amberg-Weiden', break: 1 }),
22   new TextRun({ text: 'Amberg, Deutschland', break: 1 }),
23 ],
24 }),
25 // Leerzeile
26 new Paragraph({ text: "" }),
27 // Content
28 ...content.split('\n').map((sectionText) => {
29   if (sectionText.startsWith('\\section')) {
30     return new Paragraph({
31       text: sectionText.replace('\\section', '').replace(/[{}]/g, ''),
32       heading: HeadingLevel.HEADING_1,
33     });
34   } else {
35     return new Paragraph({
36       text: sectionText,
37     });
38   }
39 },
40 ],
41 },
42 ],
43 });
44
45 const buffer = await Packer.toBuffer(doc);
46 return buffer;
47 }
```

Listing 26: Generierung eines DOCX-Dokuments für die Ausgabe