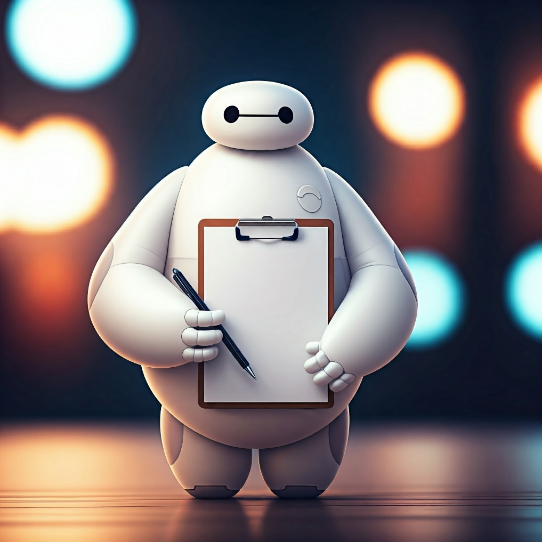
**Web-Tool für verständliche und fehlerfreie Texte**



**Abgabetermin**: 18.12.2024

**Durchführungszeitraum**: 30.10.2024 - 18.12.2024

**Projektentwickler**

Fynn Meyer

Asternweg 17

27367 Sottrum



**Ausbildungsbetrieb**

Xsigns GmbH & Co. KG

Kleekamp 6

27356 Rotenburg

Inhalt

[Einleitung 3](#_Toc184195018)

[Projekt Title 3](#_Toc184195019)

[Kurzbeschreibung 3](#_Toc184195020)

[Projekt Planung 3](#_Toc184195021)

[Einleitung in die Planung 3](#_Toc184195022)

[Ist Soll Zustand 3](#_Toc184195023)

[Zeitplanung 4](#_Toc184195024)

[Vergleich von Alternativen 4](#_Toc184195025)

[Projekt Anforderungen 4](#_Toc184195026)

[Projekt Durchführung 5](#_Toc184195027)

[Dokumentation der Durchführung 5](#_Toc184195028)

[Beschreibung der Durchgeführten Arbeitsschritte 5](#_Toc184195029)

[Technische Details 6](#_Toc184195030)

[Erklärung einzelner verwendeter Packages oder Frameworks 6](#_Toc184195031)

[Herausforderungen und Lösungen 7](#_Toc184195032)

[Abschluss 7](#_Toc184195033)

[Ergebnisbeschreibung 7](#_Toc184195034)

[Funktionstest 8](#_Toc184195035)

[Kundeneinweisung 8](#_Toc184195036)

[Fazit 8](#_Toc184195037)

[Reflektion über den Verlauf des Projektes 8](#_Toc184195038)

[Persönliche Erkenntnis und Lernerfahrung 9](#_Toc184195039)

# Einleitung

## Projekt Title

Web-Tool für verständliche und fehlerfreie Texte

## Kurzbeschreibung

In meinem Projekt entwickle ich ein benutzerfreundliches Webtool mithilfe von Vue 3 und Quasar. Dieses Tool nimmt Texte entgegen und überprüft sie über die Google Gemini API auf Rechtschreib- und Satzbaufehler. Die API korrigiert die Texte automatisch, ohne den persönlichen Schreibstil der Nutzer zu verändern.

Das Tool richtet sich insbesondere an Menschen mit Lese-Rechtschreib-Schwäche, aber auch an Schüler, Studierende und alle, die sich bei Rechtschreibung und Grammatik unsicher fühlen. Ziel ist es, eine einfache Möglichkeit zu schaffen, Texte schnell fehlerfrei und gut lesbar zu machen, ohne dass Nutzer aufwendige Nachbearbeitungen vornehmen müssen.

Dieses Tool spart Zeit und unterstützt die Nutzer dabei, die Lesbarkeit und Qualität ihrer Texte zu verbessern. Da sie sich nicht mehr um Grammatik und Rechtschreibung kümmern müssen, können sie sich voll auf den eigentlichen Inhalt konzentrieren.

In Zukunft könnte das Tool um weitere Funktionen und Sprachen erweitert werden, um eine größere Zielgruppe anzusprechen. Auch die Entwicklung einer mobilen Version, die Korrekturen direkt auf Smartphones ermöglicht, wäre eine sinnvolle Ergänzung.

# Projekt Planung

## Einleitung in die Planung

In der Schule führen wir ein Projekt durch, bei dem wir ein eigenes Thema auswählen durften. Die Anforderungen bestanden darin, dass das Projekt in Python geschrieben ist und mindestens eine API-Anfrage implementiert wird.

## Ist Soll Zustand

Die detaillierte Ist-Soll-Analyse ist in dem separaten Dokument **„IstSollZustand.docx“** enthalten. Diese Datei gehört zu den offiziellen Projektunterlagen und bietet eine umfassende Darstellung des aktuellen Zustands (Ist-Zustand) sowie der angestrebten Ziele (Soll-Zustand) im Rahmen des Projekts.

## Zeitplanung

**Vorbereitung**

* Ist- und Soll-Analyse: **2 Stunden**
* Projektplanung und Erstellung eines Anwendungsfalldiagramms: **3 Stunden**

**Durchführung**

* Installation und Einrichtung aller Nötigen Programme und Frameworks **3 Stunden**
* Entwicklung der Benutzeroberfläche: **6 Stunden**
* API-Integration und Korrektur-Logik: **10 Stunden**
* Implementierung der Anzeige der Korrekturvorschläge: **4 Stunden**
* Usability-Tests und Feedback-Implementierung: **2 Stunden**

**Abschluss**

* Soll-Ist-Vergleich: **1 Stunden**
* Projektdokumentation: **12 Stunden**
* Letzte Anpassungen und Optimierungen: **3 Stunden**

**Gesamt: 46 Stunden**

## Vergleich von Alternativen

Im Internet gibt es bereits zahlreiche Rechtschreibtools. Allerdings erfordern diese oft, dass der Nutzer jede Korrektur manuell auswählt und übernimmt. Es ist jedoch deutlich einfacher und effizienter, einen vollständig korrigierten Text zurückzubekommen, bei dem alle Fehler automatisch behoben wurden.

Die Google Gemini API stellt eine führende Text-KI dar und eignet sich hervorragend, um Texte zuverlässig und präzise zu korrigieren. Ihre fortschrittlichen Fähigkeiten machen sie zur idealen Wahl für dieses Projekt.

## Projekt Anforderungen

Das Produkt soll es ermöglichen, Texte in einer benutzerfreundlichen Weboberfläche korrigieren zu lassen. Die einfache Bedienung sorgt dafür, dass der Nutzer möglichst wenig falsch machen kann.

Für die Umsetzung wird ein Backend-Server benötigt, der in Python entwickelt wird. Außerdem ist eine API-Anbindung an Google Gemini erforderlich, um die Texte automatisch zu analysieren und zu korrigieren.

# Projekt Durchführung

## Dokumentation der Durchführung

### Beschreibung der Durchgeführten Arbeitsschritte

Zu Beginn des Projekts habe ich mich im Internet informiert, welche KI für mein Vorhaben geeignet ist. Dabei habe ich zwei große Alternativen herausgearbeitet: Google Gemini und ChatGPT. Aufgrund der Kostenfreiheit von Google Gemini habe ich mich für diese Lösung entschieden. Anschließend habe ich die Dokumentation von Google Gemini gelesen, um zu prüfen, ob eine Integration technisch möglich ist.

Nach dieser Analyse habe ich Überlegungen angestellt, wie ich das Projekt umsetzen kann. Da der Backend-Server in Python programmiert werden sollte, habe ich mich kurz informiert, wie man grafische Oberflächen und Webserver in Python erstellt. Aufgrund der begrenzten Zeit entschied ich mich für die Implementierung eines Webservers mit Python und die Nutzung des Frameworks Quasar. Dieses Framework bietet vorgefertigte Komponenten, die die Entwicklung erheblich vereinfachen. Darüber hinaus habe ich geprüft, welche weiteren Tools wie Node.js oder npm erforderlich sind, damit das Projekt reibungslos funktioniert.

Nach diesen Vorbereitungen habe ich den Ist-Soll-Zustand dokumentiert, um darzustellen, welche Anforderungen aktuell erfüllt sind und welche Ziele ich mit dem Projekt erreichen möchte.

Nach Abschluss meiner Recherche habe ich ein Use-Case-Diagramm erstellt. Dieses half mir dabei, die geplante Umsetzung meines Programms zu visualisieren und während der Entwicklung als Orientierung zu dienen.

Im nächsten Schritt begann ich mit der Implementierung eines einfachen Webservers, der Anfragen an Google Gemini sendet und die Antworten entgegennimmt. Es war wichtig, die API-Anfragen im Backend zu verarbeiten, da hierfür ein API-Key erforderlich ist. Dieser sollte nicht im Frontend sichtbar sein, da er sonst über die Entwicklertools im Browser ausgelesen werden könnte. Für Python gibt es ein spezielles Paket von Google, das die Nutzung der Google-Gemini-API erleichtert und den Abruf der Ergebnisse vereinfacht.

Nachdem die API-Integration erfolgreich war, habe ich den Webserver so angepasst, dass er bei Aufruf der entsprechenden URL die generierte index.html ausgibt. Zusätzlich habe ich den öffentlichen Ordner des Projekts freigegeben, damit Bilder und das Favicon geladen werden können und diese Anfragen nicht vom Server blockiert werden.

Mit einer funktionierenden Weboberfläche auf dem Python-Server begann ich, das Design der Seite zu erstellen. Dabei nutzte ich Quasar-Komponenten wie Loader, Buttons und Eingabefelder. Nach Abschluss des Designs implementierte ich die Verbindung zwischen der Weboberfläche und dem Server, um Anfragen an Google Gemini zu senden und die Antworten auf der Webseite auszugeben.

In einem weiteren Schritt habe ich den optimalen Prompt getestet, um qualitativ hochwertige Antworten von der KI zu erhalten. Zudem habe ich verschiedene Modelle ausprobiert, da diese unterschiedliche Ergebnisse liefern. Meine Wahl fiel schließlich auf **Gemini 1.5 Pro**, da es das aktuell am besten entwickelte Modell von Google ist und insgesamt die besten Antworten liefert.

### Technische Details

Für das Projekt wird Python in der Version 3.12 benötigt. Das Projekt wurde mit dieser Version entwickelt. Als Package kommt **google-ai-generativelanguage** in der Version 0.6.10 zum Einsatz. Zusätzlich werden **Flask** und **flask\_cors** in jeweils der neuesten Version verwendet.

Für das Frontend ist eine Node.js-Installation in der Version 20 erforderlich, ebenso wie die Nutzung von **npm**.

### Erklärung einzelner verwendeter Packages oder Frameworks

#### Quasar

#### Quasar ist ein leistungsstarkes Framework, das Entwicklern hilft, ansprechende und benutzerfreundliche Oberflächen (UIs) zu erstellen. Es stellt eine Vielzahl von Komponenten zur Verfügung, mit denen eine ansprechende und funktionale UI schnell und einfach umgesetzt werden kann. Quasar erleichtert das Design responsiver Layouts und bietet eine riesige Bibliothek vorgefertigter Komponenten, die die Entwicklungszeit erheblich verkürzen. Das Framework basiert auf **Vue.js**, was bedeutet, dass es die Vorteile und die Flexibilität von Vue nutzt, um moderne, skalierbare Webanwendungen zu erstellen.

#### Vue

Vue.js ist ein JavaScript-Framework, das den Aufbau von Benutzeroberflächen erheblich erleichtert. Es ermöglicht Entwicklern, sogenannte **Vue-Komponenten** zu erstellen, die später im Build-Prozess zu einer vollständigen HTML-Seite zusammengefügt werden.

Ein großer Vorteil von Vue.js ist die Unterstützung nützlicher Code-Features, wie zum Beispiel:

* **v-if**: Ermöglicht das dynamische Ein- oder Ausblenden von HTML-Elementen basierend auf Bedingungen.
* **v-for**: Erlaubt das einfache Erstellen von Schleifen im HTML, um z. B. Listen von Elementen dynamisch zu generieren.

Eine zentrale Rolle in Vue.js spielen **Referenzen** und **Reaktivität**. Mithilfe dieser Mechanismen können Entwickler flexibel auf Änderungen im Datenmodell reagieren. Dadurch ist es möglich, ohne großen Aufwand Komponenten ein- oder auszublenden, Benutzeraktionen zu verarbeiten oder die Benutzeroberfläche dynamisch zu aktualisieren.

Zusätzlich bietet Vue.js eine klare Struktur für die Entwicklung moderner Webanwendungen. Durch die Modularität der Komponenten bleibt der Code übersichtlich und der Entwicklungsprozess wird effizienter. Für komplexere Projekte stellt Vue erweiterbare Tools wie den **Vue Router** (zur Navigation in Single-Page-Anwendungen) und **Vuex** (für Statusmanagement) bereit.

.

#### Node.js

Node.js basiert auf der V8-Engine von Google Chrome und zeichnet sich durch seine Geschwindigkeit und Flexibilität aus. Es unterstützt **asynchrone Programmierung**, wodurch mehrere Prozesse parallel ausgeführt werden können, ohne den Server zu blockieren. Dies macht es ideal für Anwendungen, die auf hohe Leistung und Skalierbarkeit ausgelegt sind, wie beispielsweise Webserver oder APIs.

Ein weiteres Merkmal von Node.js ist, dass es nicht nur für serverseitige Anwendungen verwendet werden kann, sondern auch zur Erstellung von **Entwicklungswerkzeugen** und **Build-Systemen**. Diese Vielseitigkeit macht es für Entwickler zu einem äußerst wertvollen Werkzeug.

#### NPM Paketmanager

npm ist der Paketmanager von Node.js und eine zentrale Komponente der Node.js-Umgebung. Es bietet Zugriff auf eine riesige Bibliothek von Modulen und Tools, die von der Entwickler-Community bereitgestellt werden. Mit npm können Entwickler schnell und einfach Abhängigkeiten für ihre Projekte installieren und verwalten.

Ein wichtiger Vorteil von npm ist die Möglichkeit, eigene Pakete zu erstellen und zu veröffentlichen. Dadurch wird Code wiederverwendbar und kann mit anderen Entwicklern geteilt werden. Darüber hinaus ermöglicht npm das Ausführen von Skripten für häufig genutzte Aufgaben im Entwicklungsprozess.

### Herausforderungen und Lösungen

Eine der ersten Herausforderungen war, dass der Python-Interpreter defekt war und ich es nicht geschafft habe, ihn richtig zu installieren oder zu reparieren. Ich versuchte, den Fehler zu beheben, indem ich nur den Interpreter aktualisierte, aber das funktionierte nicht. Daher entschloss ich mich, die gesamte IDE neu zu installieren, was das Problem schließlich löste.

Ein weiteres Problem trat auf, als mein Frontend nicht auf den **Public-Ordner** zugreifen konnte, wodurch Images und andere Assets nicht geladen wurden. Da Vue alle Dateien in den Public-Ordner packt, blockierte der Webserver die Anfragen. Um dieses Problem zu lösen, musste ich in **Flask** zusätzliche Konfigurationen vornehmen, um die Ordner tatsächlich als öffentlich zugänglich zu machen.

# Abschluss

## Ergebnisbeschreibung

Ich habe einen Webserver entwickelt, der über **Flask** eine generierte **index.html** bereitstellt. Diese Seite habe ich zuvor mithilfe des Frameworks **Quasar** erstellt, auf dem die Weboberfläche für mein Programm basiert. Auf der Seite kann der Nutzer seinen Text in ein Textfeld eingeben und dann auf einen Button klicken. Der Text wird daraufhin geprüft und an mein Backend gesendet. Dort wird er an **Google Gemini** weitergeleitet, und die Antwort wird zurück an das Frontend übermittelt. Im Frontend wird die Antwort grafisch dargestellt, und dem Nutzer wird die Möglichkeit geboten, den korrigierten Text mit einem Klick zu kopieren.

## Funktionstest

Ich habe die Software auf verschiedenen Geräten getestet und dabei auch absichtlich fehlerhafte Texte eingegeben, um mögliche Fehler zu provozieren. Während der Tests traten einige Probleme auf, die ich dann behoben habe. Ein Beispiel dafür war, dass teilweise die Antworten von **Google Gemini** zu fehlerhaften Darstellungen führten, die ich entsprechend korrigiert habe. Diese Tests haben geholfen, die Stabilität und Genauigkeit der Anwendung zu verbessern.

## Kundeneinweisung

Für die Installation wird Python Version 3.12 sowie die aktuelle Version von Node.js und npm benötigt. Mit npm wird Quasar entweder global oder nur im Projekt installiert, um das Projekt bauen zu können. Zuerst müssen jedoch die Abhängigkeiten installiert werden. Dies geschieht im Root-Frontend-Ordner mit dem Befehl: „npm run install“

Anschließend kann das Projekt gebaut werden, indem der folgende Befehl ausgeführt wird: „npm run build“. Wenn der Build-Prozess erfolgreich abgeschlossen wurde, kann der Python-Server gestartet werden. Dazu wird die **index.py** ausgeführt.

# Fazit

## Reflektion über den Verlauf des Projektes

Die Entwicklung des Projektes verlief insgesamt gut, da ich die festgelegten Ziele innerhalb der vorgesehenen Zeit erreichen konnte. Ich habe viele nützliche Informationen im Internet gefunden, beispielsweise auf **Stack Overflow**, die mir geholfen haben, den Backend-Server zu gestalten und verschiedene Probleme zu lösen.

Das Entwickeln des Frontends war ebenfalls recht einfach, da mir das Framework Quasar viel Design- und Logik-Arbeit abgenommen hat. Dadurch konnte ich die Arbeit am Frontend schnell und ohne größere Probleme abschließen.

Bei der Verbindung von Frontend und Backend traten jedoch einige Probleme auf. Diese entstanden vor allem durch meine noch begrenzte Erfahrung mit Backend-Servern in Python. Es gab einige Fehlkonfigurationen, die ich aber dank Foren und Künstlicher Intelligenz nach etwa einer Stunde beheben konnte.

Ein weiterer Rückschlag war die Vorbereitung auf das Projekt. Erst während der Entwicklung stellte ich fest, dass die ursprünglich ausgewählte **Python Version 3.13** nicht mit dem **Google Gemini**-Package kompatibel war. Das war ärgerlich, da ich bei der Planung darauf hätte achten müssen, als ich die Version festlegte.

## Persönliche Erkenntnis und Lernerfahrung

Persönlich habe ich gelernt, dass ich bei der Auswahl von Versionen für Packages oder Frameworks in meinen Projekten sorgfältiger sein muss. Es kann unnötig viel Zeit kosten, wenn man während der Entwicklung auf Probleme stößt, die einfach dadurch entstehen, dass man eine falsche Version ausgewählt hat.