

Projekt Dokumentation

Texteditor

Texteditor mit GUI



BSIT23d

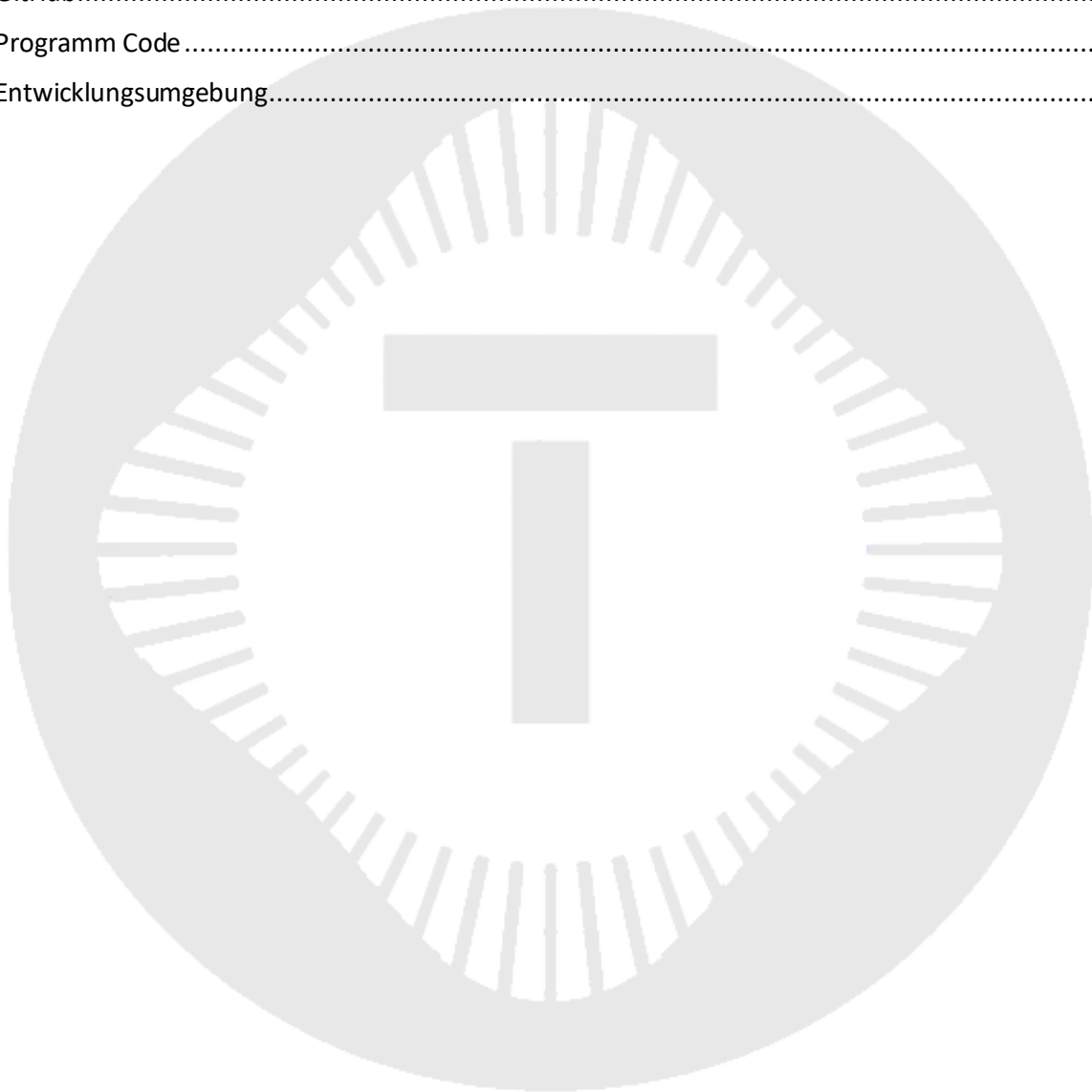
LF05

Verantwortliche:	Luca Rohmann
	Robin Buchholz
	Falk-André Siegel

Inhalt

Einleitung	4
Projektbeschreibung	5
Pflichtenheft.....	6
Pflichtenheft stand 11.04.2024	6
Lastenheft	7
Lastenheft stand 11.04.2024	7
Lastenheft stand 15.04.2024	7
Lastenheft stand 29.04.2024	7
Wochen Berichte	8
Wochen Bericht 1 (11.04.2024 bis zum 14.04.2024)	8
Wochen Bericht 2 (15.04.2024 bis zum 21.04.2024)	8
Wochen Bericht 3 (22.04.2024 bis zum 28.04.2024)	10
Wochen Bericht 4 (29.04.2024 bis zum 05.05.2024)	11
Wochen Bericht 5 (06.05.2024 bis zum 12.05.2024)	13
Wochen Bericht 6 (13.05.2024 bis zum 19.05.2024)	13
Wochen Bericht 7 (20.05.2024 bis zum 26.05.2024)	14
Wochen Bericht 8 (27.05.2024 bis zum 02.06.2024)	15
Wochen Bericht 9 (03.06.2024 bis zum 05.06.2024)	16
Code Dokumentation	18
Swing-Editor	18
„Öffnen“-Funktion	18
Wörter-„Zähler“	20
Spracheinstellungen.....	21
FX-Editor.....	23
Erstellung des Fensters per Java-FX.....	23
Erstellung der Speichern- / Öffnungsfunktion	24
Erstellung der „Neu“ Funktion	25
Ladebildschirm	25
Erstellung der Spracheinstellungen	28
Erstellung des Help-Menü	28
API Nutzung.....	30
„Werbe-“ Bilder.....	31
Programm Start.....	31

Zusammenfassung und Fazit	32
Anlagenverzeichnis	33
Logo	33
GUI.....	33
Miro Board	34
GitHub.....	34
Programm Code	34
Entwicklungsumgebung.....	34



Einleitung

Diese Dokumentation zeigt Ihnen die Entwicklung und Entstehung unseres Textverarbeitungsprogrammes Textis. Entstanden ist das Projekt im Rahmen der Projektarbeit im Lernfach 5 (Software zur Verwaltung von Daten anpassen) an der Carl Benz Schule in Koblenz, vom 11.04.2024 bis zum 05.06.2024. Realisiert haben wir das Projekt sowohl in den dafür gegebenen Schulstunden, so wie in unserer Freizeit.

Erstellt wurde das Programm aus einem Team, bestehend aus Robin Buchholz, Luca Rohmann und Falk-André Siegel aus der Klasse BSIT23d. Das Ziel des Projektes war es, unsere im ersten Lehrjahr erworbenen Fähigkeiten und Kenntnisse in Java in einem realistischen Projekt anzuwenden und bei Bedarf zu erweitern.

Unser Schwerpunkt lag darauf zwei GUI Bibliotheken kennen zu lernen und mit diesen eine geeignete Benutzeroberfläche für unseren Texteditor zu erstellen. Begonnen haben wir mit der Entwicklung mit der älteren Java Swing Bibliothek und haben später unser Programm aufgrund von Funktionsproblemen auf Java-FX umgestellt. Eine Herausforderung dabei war, dass ein leichtes Grund Verständnis für HTML und CSS braucht um die Funktionsweise des benötigten RICH-Text-Field zu verstehen.

In dieser Dokumentation werden Sie den gesamten Entwicklungsprozess von Textis von Anfang bis Enden nachvollziehen können und sehen welche Herausforderungen und Probleme während der Zeit aufgetreten sind und wie wir diese gelöst haben.

Projektbeschreibung

Unser Ziel, beim Projektstart, war es ein Notizprogramm mit einer Grafischen Benutzeroberfläche in Java zu entwickeln, dies hat sich am Beginn der Zweiten Woche geändert. Wir haben uns für diesen Schritt entschieden, weil ein Texteditor außerhalb des Programmes speichert und wir somit mit Windows Schnittstellen Arbeiten müssen und dies spannende Möglichkeiten für das Programm ermöglichen. Des weiteren bietet ein Texteditor mehr Potenzial für zusätzliche und Zukünftige Funktionen. Genauere Details dazu entnehmen Sie bitte dem Wochenbericht 2.

Der Texteditor soll eine Grafische Benutzeroberfläche (GUI) für eine Benutzerfreundliche Interaktion mit dem Programm aufweisen. Die GUI soll dem Benutzer alle benötigten Optionen und Formatierungsmöglichkeiten, einfach von Form von leicht bedienbaren Buttons zur Verfügung stellen. Dabei soll das Programm dem Benutzer mit einer Optisch ansprechbaren und leichten GUI überzeugen.

Das Programm soll es dem Benutzer ermöglichen in den gewöhnlichen Dateiformaten seine Arbeit speichern zu können. Bei der Speicherung dieser sollen alle Formatierung mit übernommen werden, so dass diese, durch eine erneute Öffnung beibehalten werden und von anderen Programmen verwendet werden können.

Drüber hinaus soll es dem Benutzer ermöglicht werden, bereits erstellte Dokumente öffnen und bearbeiten zu können, auch hier soll die von anderen Programmen erstellte Formatierung mit übernommen und angewendet werden.

Zusätzlich soll der Texteditor dem Benutzer die Möglichkeit geben, sich diesen nach seinen eigenen Wünschen anzupassen. Sei es durch eine Auswahl der System Sprache oder die Möglichkeit einen Dark- / Light-Mode anzuwenden. Dafür wird ihm ein Einstellung Menü zur Verfügung gestellt. Diese Benutzerpräferenzen soll das Programm zwischen Speichern und bei einem erneuten öffnen laden und wieder aktivieren, damit der Anwender diese Einstellungen nicht jedes Mal wieder erneut treffen muss.

Außerdem soll dem Anwender durch ein Hilfe Menü Tipps und Hilfe Stellen gegeben werden. Diese sollen direkt über das Programm erreichbar sein, damit er sich schnellstmöglich im Programm einfindet und sich mit diesem vertraut machen kann.

Unser Programm wird für die Anwendung auf einem Windows Computer (Windows 10 und 11), mit Java 17 in Verbindung mit der Java-FX Bibliothek und dem dazugehörigem Eclips-Plug-In entwickelt. Zur einfachen zusammen Arbeit wird zusätzlich das Plug-In Code-Together installiert, welches es uns ermöglicht zeitgleich im selben Programm Code zuarbeiten.

Pflichtenheft

Pflichtenheft stand 11.04.2024

Das Pflichtenheft spiegelt die Anforderungen auf Moodle, gestellt von Herrn Dietrich wieder.

Minimale Anforderungen

- Einfache Datentypen (z.B. int, double)
- Schleifen (z.B. for, while)
- Bedingungen (z.B. if-else)
- Verbunddatentyp für die Speicherung (z.B. Arrays)
- Methoden verwenden

Optionale Anforderungen

- ggfs. Struktogramm, PAP zur Logik des Programms
- UML-Diagramme
- Datenbank anlegen
- Benutzeroberfläche für das Programm entwickeln (z.Bsp. JavaFX, Java swing)
- ggfs. Menü erstellen (z.Bsp. einem Spiel Schwierigkeitsgrad wählen)
- Kollaboratives Arbeiten am Projekt mit einem Tool, wie z.B. GitHub
- mit Web-API arbeiten (z.Bsp. aktuelle Wetterdaten nutzen)
- Programm als lauffähige Datei abgeben

Bewertungskriterien

- Erfüllung der minimalen Anforderungen
- Bearbeitung der optionalen Anforderungen für sehr gute Noten notwendig
- Code-Qualität und Lesbarkeit (sinnvolle Variablenbezeichnungen, Code strukturiert und sinnvoll kommentieren)
- Benutzerfreundlichkeit der Anwendung
- Dokumentation und Erklärung des Codes
- Fristgerechte Abgabe und Präsentation (inkl. Beantwortung von Rückfragen zur Präsentation)
- regelmäßige Projektdokumentation (wöchentlich)
- Abgabe der Dokumentation als PDF

Lastenheft

Lastenheft stand 11.04.2024

Das Ziel dieses Projekts ist die Entwicklung eines Notizbuches in Java mit einer grafischen Benutzeroberfläche (GUI) in Java Swing. Das Notizbuch soll es Benutzern ermöglichen, Notizen zu erstellen, zu speichern, zu bearbeiten und zu löschen.

Die Realisierung der Programm-Funktionen erfordert den Einsatz verschiedener Konzepte und Techniken der Java-Programmierung, darunter die Verwendung von Datentypen, Schleifen, Bedingungen, Arrays und Methoden. Durch die grafische Benutzeroberfläche wird eine benutzerfreundliche Interaktion mit dem Notizbuch gewährleistet.

Das Notizbuch soll dem Benutzer eine klare Strukturierung und Organisation der Notizen bieten.

Die Anforderungen entnehmen Sie bitten dem Pflichtenheft.

Lastenheft stand 15.04.2024

Das überarbeitete Ziel dieses Projekts ist die Entwicklung eines Texteditors in Java mit einer grafischen Benutzeroberfläche (GUI) in Java Swing. Der Editor soll es Benutzern ermöglichen, Dokumente zu erstellen, zu speichern und vorhandene Text Dokumente zu öffnen.

Die Realisierung der Programm-Funktionen erfordert den Einsatz verschiedener Konzepte und Techniken der Java-Programmierung, darunter die Verwendung von Datentypen, Schleifen, Bedingungen, Arrays und Methoden. Durch die grafische Benutzeroberfläche wird eine benutzerfreundliche Interaktion mit dem Texteditor gewährleistet.

Der Texteditor soll dem Benutzer eine klar Strukturierte GUI bieten, mit welcher er die grundsätzlichen Funktionen der Textverarbeitung nutzen kann.

Die Anforderungen entnehmen Sie bitten dem Pflichtenheft.

Lastenheft stand 29.04.2024

Das erneut überarbeitete Ziel dieses Projekts ist die Entwicklung eines Texteditors in Java mit einer grafischen Benutzeroberfläche (GUI) in Java FX. Der Editor soll es Benutzern ermöglichen, Dokumente zu erstellen, zu speichern und vorhandene Text Dokumente zu öffnen und umfangreich Formatieren zu können.

Die Realisierung der Programm-Funktionen erfordert den Einsatz verschiedener Konzepte und Techniken der Java-Programmierung, darunter die Verwendung von Datentypen, Schleifen, Bedingungen, Methoden und einem Rich-Text-Field. Durch die grafische Benutzeroberfläche wird eine benutzerfreundliche Interaktion mit dem Texteditor gewährleistet.

Der Texteditor soll dem Benutzer eine klar Strukturierte GUI bieten, mit welcher er die grundsätzlichen Funktionen der Textverarbeitung und umfangreiche Möglichkeiten der Textformatierung nutzen kann.

Die Anforderungen entnehmen Sie bitten dem Pflichtenheft.

Wochen Berichte

Wochen Bericht 1 (11.04.2024 bis zum 14.04.2024)

Projekt Start am 11.04.2024

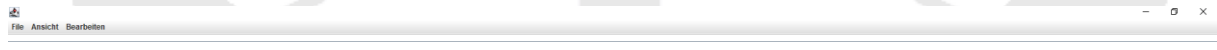
Wir haben uns in der Gruppe dazu entschieden ein Notizprogramm mit einer GUI zu erstellen. Die Unterrichtsstunde (am 11.04.2024), haben wir dazu genutzt unser Projekt zu Definieren und uns eine Microsoft Teams Gruppe zu erstellen um auch außerhalb der Schule an unserem Projekt arbeiten zu können. Darüber hinaus haben wir uns auch ein entsprechendes Eclipse Plug-In, Code together, installiert um Zeitgleich am Code Arbeiten zu können.

Projekt Definition

- Erstellung eines Notizprogrammes in Java
- Mit GUI, auf Basis der Swing Bibliothek
- Grundlegende Funktionen
 - Das Erstellen neuer Notizen
 - Das Speichern von Notizen
 - Das Öffnen von Notizen
 - Das Bearbeiten von Notizen
- Benutzer soll in der GUI, alle erstellten Notizen sehen.
- Die Notizen sollen im Programm gespeichert werden

Nach dem wir unser Projekt definiert haben, begonnen wir mit der Recherche und dem Üben der Java Swing Bibliothek.

Am Ende der Woche, hatten wir bereits eine einfache GUI erstellt, welche ein Textfeld für Benutzer eingaben erhielt.



Wochen Bericht 2 (15.04.2024 bis zum 21.04.2024)

Zum Beginn dieser Woche haben wir uns dazu entschieden das Projekt Thema zu verändern. So haben wir die Idee eines Texteditors verworfen und mit der Projekt Definition eines Texteditors gestartet.

Da wir mit Windows Schnittstellen arbeiten möchten, die uns beispielsweise das Windows Dialogfeld zum Speichern ermöglichen, da ein Texteditor außerhalb des Programmes die Dateien speichert. Dadurch haben wir auch die Möglichkeit uns mit Dateitypen auseinander zusetzen und wie wir diese in andere gewünschte Formate Konvertieren können.

Projekt Definition

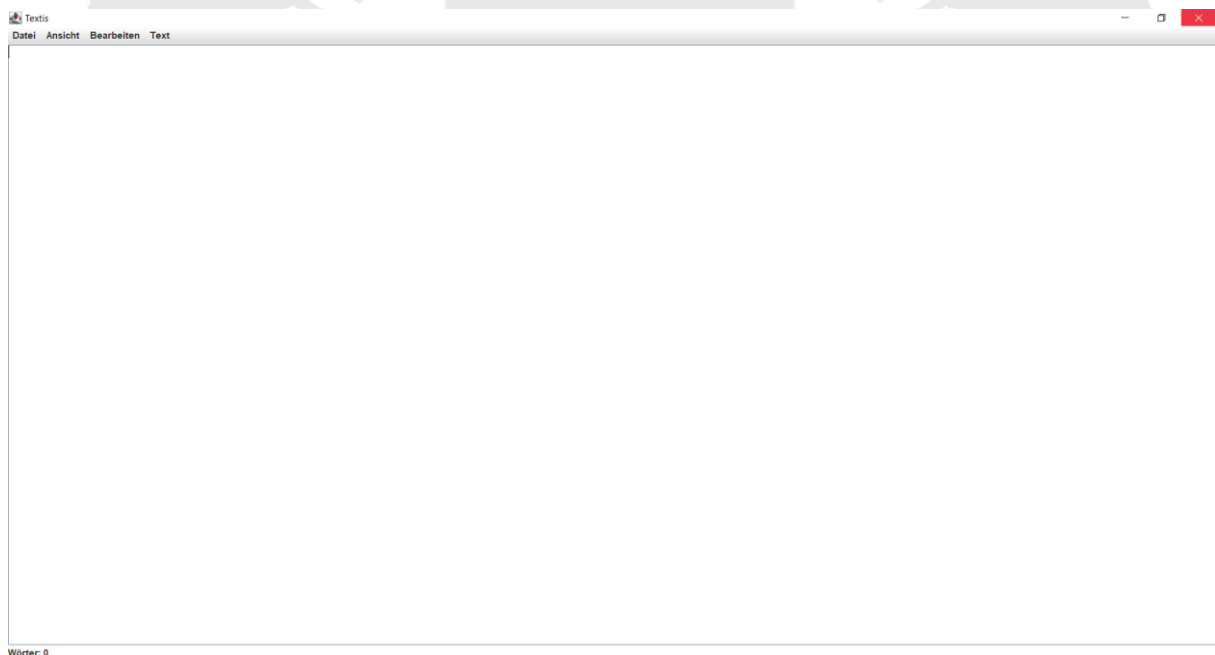
- Erstellung eines Texteditors in Java
- Mit GUI, auf Basis der Swing Bibliothek
- Grundlegende Funktionen
 - Das Erstellen neuer Text Dateien
 - Das Speichern über den Windows Datei Explorer
 - Das Speichern in gültige Datei Formate z.B.: .txt

- Das Öffnen von .txt Dateien über den Windows Datei Explorer
- Das Bearbeiten von .txt Dateien

Da wir zu diesem Zeitpunkt nur das Fenster mit einem Menü-Band und einem Eingabefeld programmiert hatten, konnten wir das vorhandene Programm ohne benötigte Anpassungen als die Grundlage für unseren neu geplanten Texteditor verwenden.

Im Laufe des Montags wurden die Speicher, „Öffnen“-Funktion, und neu (die neue Funktion wird weiter unten, in diesem Wochenbericht erklärt) – Funktionen des Programmes implementiert. Da wir zum Ermöglichen dieser Funktionen auf Windows Schnittstellen zurückgreifen war das Programm von nun an nur noch mit Windows Kompatibel (getestet haben wir Windows 10 und Windows 11).

Des weiteren wurde der Textbereich so angepasst, dass dieser nun vollflächig und nicht mehr als Strich angezeigt wird. Darüber hinaus wurde das Textfeld so angepasst, dass wenn der Benutzer bis zum unteren Rand des Textfeldes geschrieben hat, an der rechten Seite des Programmes eine Scroll Leiste erscheint, mit welcher man im Text nach oben und unten navigieren kann. Als diese Änderungen waren von Nöten, um dem Benutzer eine deutlich bessere Benutzer Erfahrung mit unserem Programm bieten zu können.



Außerdem wurde es ermöglicht dass der Benutzer den Text vergrößern und verkleinern, die Schriftart (wir haben 3 verschiedene Implementiert) auswählen und die Schriftfarbe, frei, auswählen kann. Zum Anpassen der Schriftfarbe haben wir eine Swing Komponente Implementiert, welche uns einen umfangreichen Farbeditor zur Verfügung stellte. Somit stoßen wir auch auf das erste Problem, die Änderungen, betreffen das ganze Dokument und nicht nur einzeln, Ausgewählte Text Elemente. Des weiteren wurde die Textformatierung, in Form der Textgröße nicht in die Speicherung des Dokumentes übernommen, sondern fand immer nur visuell in der GUI statt.

Am Ende des Tages haben wir noch an der Unteren Linken Seite des Programmes einen Wörter-„Zähler“ hinzugefügt, welche alle geschriebenen Wörter, ähnlich wie in Microsoft Office zählt.

In der Weiteren Woche haben wir versucht, das Programm um eine Spracheinstellungen zu erweitern. Dafür haben wir alle Texte im Programm sowie anzeigen und die Texte der Buttons, in ein 2D Array geschrieben. Durch eine Variable im Code konnte man vor dem Starten des Programmes sich eine Sprache aussuchen (Englisch oder Deutsch).

Allerdings haben wir auch damit schnell ein Problem festgestellt, wenn man versuch die Sprachen, während der Programm Laufzeit zu ändern, also über einen Button in der GUI, wird zwar die variable geändert, aber die Sprache bleibt gleich. Das Problem liegt darin, das die GUI-Elemente sich einmal, nach der Sprach Änderung Aktualisieren müssten. Nach vermehrten Ansätzen dieses Problem auf verschiedenen Wegen zu lösen, haben wir uns dazu entschlossen, diese Funktion, erst einmal hintenanzustellen und uns erst einmal um die Grundfunktionen des Programmes zu kümmern.

Im weiteren Verlauf haben wir noch Tasten Shortcuts für Einfügen, Ausschneiden und Kopieren implementiert und die entsprechenden GUI-Buttons mit ihren Funktionen versehen. Diese Funktionalitäten befinden sich innerhalb der Implementierung des Popup-Menüs.

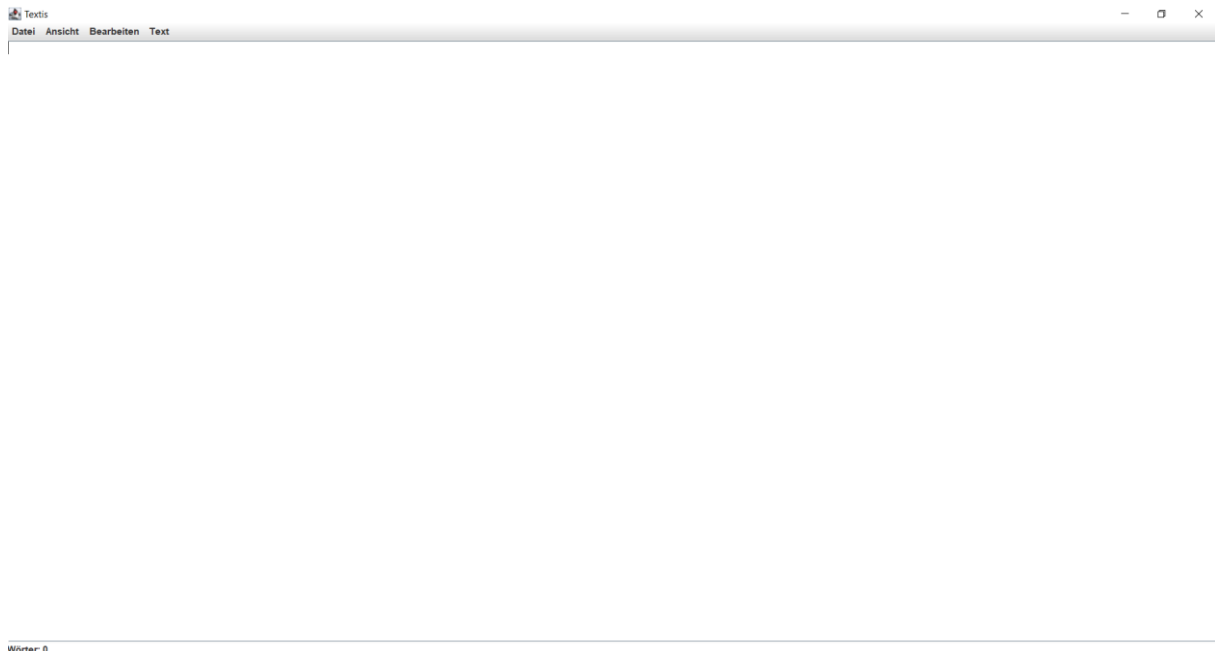
Zum Ende der Woche wollten wir noch einen Dark Mode einbauen. Also eine Funktion mit der der Benutzer sich zwischen einer Weißen und einer Schwarzen Optik entscheiden kann. Nach dem diese Funktion eingebaut war, wollten wir, dass die gemachte Einstellung auch gespeichert wird. Leider haben wir es nicht geschafft die Speicherung innerhalb dieser Woche zu realisieren.

Wochenrückblick (15.04.2024 bis um 21.04.2024)		
Funktioniert	In Arbeit	Funktioniert nicht richtig
Speichern, öffnen, neu	Dark Mode (mit Speicherung)	Textformatierungsoptionen
Vollflächiges Textfeld		Sprach Einstellungen

Wochen Bericht 3 (22.04.2024 bis zum 28.04.2024)

In dieser Woche haben wir mit der Implementation der Spracheinstellungen weiter gemacht. Ein Durchbruch, war das Verlagern der Sprache, von einem Array in sogenannte Properties Dateien. Dies ermöglichte es uns, testweise in einem Testprogramm Änderungen des Textes in z.B. Buttons während der Programm Laufzeit. Bei der Implementation in unser Hauptprogramm führte dies allerdings dazu, dass die Sprache wie vorher nicht übernommen wurde.

Da wir zu dieser Zeit schon viele Probleme hatten, also das Speichern der Dark Mode Einstellung, das Ändern der Sprache oder das Formtieren von Selektiertem Text, entschlossen wir uns das Projekt in dieser Woche Ruhen zulassen und sammelten Ideen wie wir diese Problematiken bewältigen können.



Wochenrückblick (22.04.2024 bis um 28.04.2024)		
Funktioniert	In Arbeit	Funktioniert nicht Richtig
Speichern, öffnen, neu		Textformatierungsoptionen
Vollflächiges Textfeld		Sprach Einstellungen
		Dark Mode (mit Speicherung)

Wochen Bericht 4 (29.04.2024 bis zum 05.05.2024)

Mit Beginn der neuen Woche beschlossen wir, dass wir das Projekt von Java Swing auf Java FX umstellen, da Swing für unsere Vorhaben nicht Flexible genug ist, z.B. für die Spracheinstellungen und uns die Optik der Swing GUI-Elemente nicht gefallen hat. Mit dieser Umstellung haben wir auch ein Rich-Text-Field eingeführt.

Der erste Tag dieser Woche wurde für Recherchen genutzt. Wir haben uns angeschaut, wie man eine GUI in JAVA FX erstellt und wie wir ein Rich-Text-Field erstellen.

Im Verlaufe dieser Woche haben wir es geschafft ein Erstellung des Fensters per Java-FX Fenster zu erstellen und dieses mit einem Rich-text-Field zu erweitern. Der Vorteil von diesem ist, dass es Texte im Format von HTML-Dokumenten verarbeitet. Per CSS können diese angepasst, bzw. Formatiert werden. Somit haben wir alle Möglichkeiten, die ein normales HTML-Dokument auch hat. Wir können Text selektiv Formatieren, also die Textfarbe ändern, den Text Hintergrund wählen, ihn vergrößern, verschiedene Schriftarten wählen und den Text Fett, Kursiv, unterstrichen oder durchgestrichen darstellen.

Des weiteren haben wir unsere GUI durch ein Menü-Band und eine Toolbar erweitert. Im Menü-Band haben wir Buttons für die Fertigen Programmfunktionen erstellt und nach den Gruppierungen File, Edit, und Help einsortiert. Als erste Funktion haben wir unter der Option File, das Erstellen eines neuen Dokumentes hinzugefügt. Wenn der Benutzer diesen Knopf drückt, wird das Rich-Text-Field gelehrt, bzw. mit einem Leeren Text überschrieben. Zusätzlich haben wir dort schon Platzhalter für Zukünftige Funktionen wie z.B. das Dokumenten speichern eingebaut.

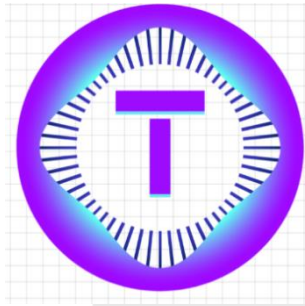
Wochenrückblick (29.04.2024 bis um 05.05.2024)			
Funktioniert	In Arbeit	Funktioniert nicht richtig	Fehlende Funktionen
Erstellung des Hauptfensters und erste GUI-Buttons auf Java-FX Basis			Speichern und öffnen von Dokumenten
Rich-Text-Field			Drucken
Menü-Bar			Sprachoptionen
Tool-Bar			Dark Mode
Formatierungsoptionen			

Wochen Bericht 5 (06.05.2024 bis zum 12.05.2024)

In dieser Woche haben wir nicht weiter am Projekt gearbeitet, aufgrund von dem Feiertag am 09.05.2024 und entsprechenden Urlaubstagen, innerhalb der Gruppen Mitglieder.

Wochen Bericht 6 (13.05.2024 bis zum 19.05.2024)

Im Beginn dieser Woche haben wir uns für den Projekt Namen: „Textis“ entschieden und ein Logo für unser Projekt erstellt. Dann haben wir das Logo in unserem Programm „eingebettet“, sodass dieses als Symbol in der Fenster- sowie in der Taskleiste sichtbar ist.

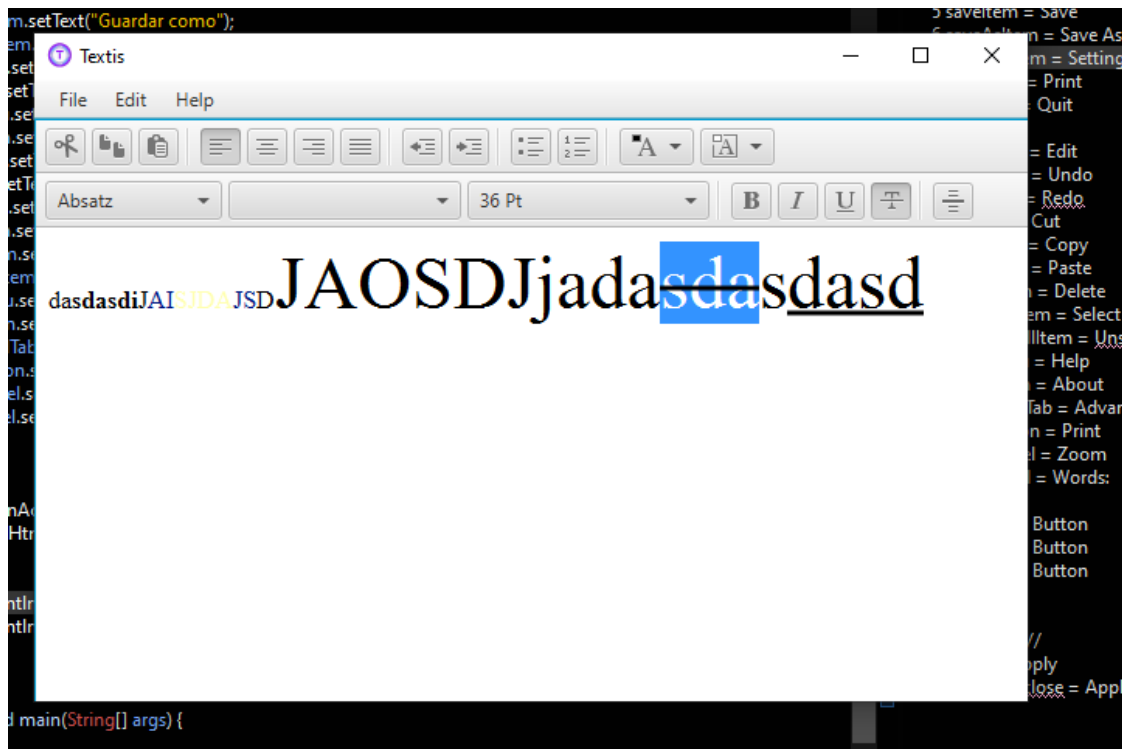


Des Weiteren haben wir uns dazu entschieden Einstellung wie z.B. die Sprach Auswahl in ein Externes Fenster auszulagern. Die Auswahl der Sprache konnten wir mit dem Wechsel auf Java FX nun Funktionsfähig Implementieren.

Im Verlauf der Woche haben wir das Speichern als HTML und TXT-Dateien und deren Erstellung der Speichern- / Öffnungsfunktion ermöglicht.

Anschließend haben wir eine kurze Bestands Analyse gemacht und ausgearbeitet, was wir noch auf jeden Fall benötigen und welche Funktionen wir noch zusätzlich hinzufügen können. Denn durch die einwöchige Pause und den Wechsel von Swing auf FX haben wir einige vorher Implementierte Funktionen verloren.

Bestandsanalyse		
Erfolgreich Implementierte Funktionen	Noch Notwendige Funktionen	Zusätzliche Funktionen
Umfangreiche Textformation	Fertigstellung unseres Settings Menüs	Druck Menü, zum ermöglich das der Benutzer die Dateien auch drucken kann.
Neu, das leeren des Textfeldes	Texte für das Help-Fensters	Weitere Datei Formate wie DOCX und PDF
GUI-Buttons mit Funktionen ausstatten	Erstellung eines Help Fensters	Wortzähler
Öffnen und Speichern von HTML und TXT-Dateien		Zoom-Funktion, um den Text „zähler“ zu holen.
Spracheinstellungen		
Externes Einstellungsmenü und Einbindung dieses ins Hauptprogramm		



Zeitgleich haben wir nun begonnen eine PowerPoint Präsentation vorzubereiten und haben uns ein Miro Board erstellt, um dort alle Dateien ablegen zu können und eine Programm Ablauf Plan zu erstellen.

Zum Ende der Woche sind wir unserem Hauptziel, der Unterstützung von .pdf und .docx Dokumenten zwar ein gutes Stück nähergekommen, konnten dies aber noch nicht Final fertigstellen, da bei der Konvertierung in unser Textfeld Fehler entstanden sind.

Wochen Bericht 7 (20.05.2024 bis zum 26.05.2024)

In dieser Woche hatten wir aufgrund von Feier- und Urlaubstagen, nicht genug Zeit um uns um die großen Probleme des Programmes zu kümmern. So haben wir in dieser Woche nur einen Lade Bildschirm erstellt. Unser Hauptprogramm braucht ca. 10 bis 15 Sekunden, bis es gestartet ist, weshalb wir uns für die Erstellung eines Ladebildschirms entschieden haben. Dieser ist statisch, das bedeutet das er eine gewisse, vor eingestellte Zeit wartet und dann verschwindet. Es gäbe auch die Möglichkeit einen Dynamischen Ladenbildschirm einzubauen, welcher den Aktuellen Lade Stand des

Programmes abfragt, und während der gesamten Startdauer des Hauptprogrammes angezeigt wird.



Der Dynamische Ladebildschirm wäre auf jeden Fall die bessere, und sauberere Variante gewesen, allerdings wäre dieser auch deutlich komplexer und schwieriger geworden und da dies nur eine Zusatz-“Funktion” unseres Programmes war haben wir uns gegen die schwierigere- und für die statische Methode entschieden.

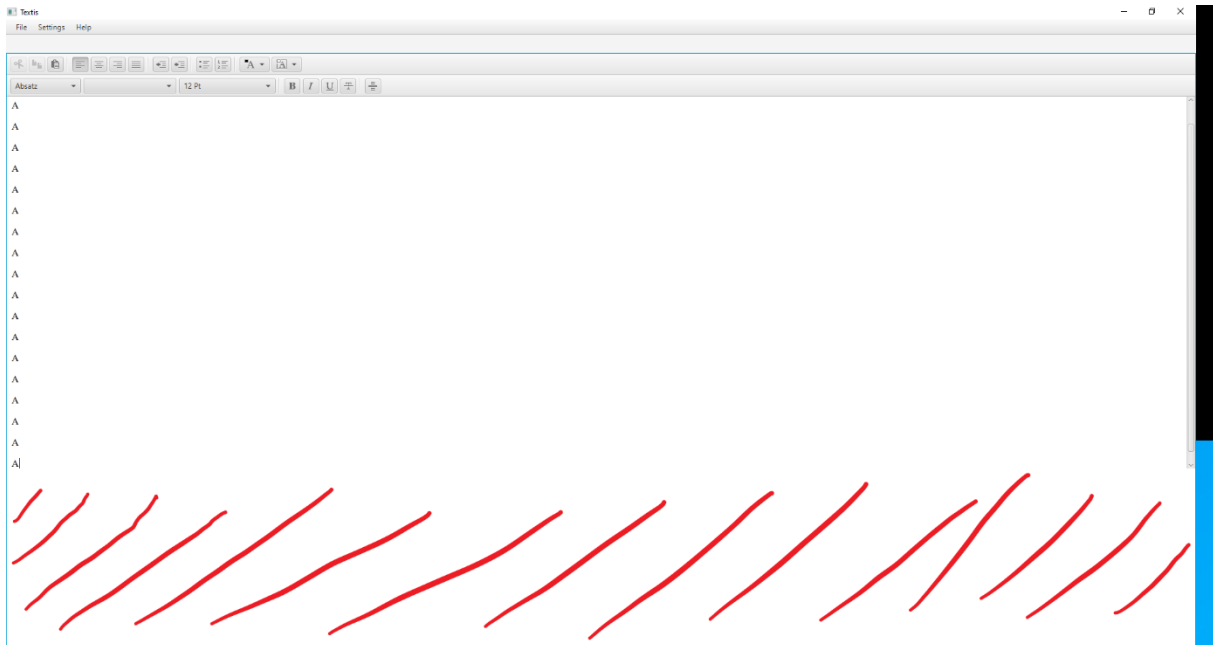
Wochen Bericht 8 (27.05.2024 bis zum 02.06.2024)

In dieser Woche haben wir eine bestandst Analyse erstellt, um zu schauen, welche Aufgaben wir noch haben und welche wir eventuell aus dem Projekt nehmen müssen.

Verlegung des Einstellungsmenüs von einem externen Menü, zu einer direkt integrierter Schallfläche in der GUI
PDF / DOCX Unterstützung
Wort-Zähler

Bei der Erstellung unserer Bestandsanalyse, haben wir alle Funktionen des Programmes getestet. Dabei wurde ein schwerwiegendes Problem entdeckt. Wenn das Programmfenster über eine Höhe von ca. 700 px skaliert wird, hört dort das Textfeld auf, weiter mit zu skalieren. Dies bedeutet das unter dem Textfeld ein ungenutzter “Balken” entsteht. Dies ist uns leider erst sehr spät aufgefallen, da das Textfeld und das Programm, jeweils einen weißen Hintergrund haben. Nach mehrmaliger Überprüfung des Programm Codes, und verschieden versuchen das Textfeld zu zwingen über 700 px zu skalieren, konnten wir das Problem durch eine lange Recherche nach zwei Tagen feststellen. Nämlich liegt der Fehler in der verwendeten Bibliothek. Unser Textfeld ist ja, ein HTML-Editor-Feld. Dieses bekommt von der Bibliothek, aus welcher wir es Importieren, eine maximale Größe übergeben, welche wir nicht über schreiben / umgehen können. Eine Lösung für diese s Problem wäre es, eine neue Bibliothek für das Textfeld einzubinden, nach dem wir uns dies angeschaut hatten, sind wir leider zu dem Entschluss gekommen, dass durch einen notwendigen-, grundlegendem-Umbau der GUI, und eine deutlich komplexere Implementation dieser-, der Wechsel zu einer neuen Bibliothek für uns in der restlichen Projekt Zeit leider nicht mehr realisierbar ist. Dieses Problem sehen Sie im nachfolgenden Bild, der “Balken” wurde für eine bessere Sichtbarkeit

rot markiert.



Wochen Bericht 9 (03.06.2024 bis zum 05.06.2024)

In der Finalen Woche mussten wir uns eine Lösung für den „unsichtbaren-Balken“ ausdenken.

Da wir, wie im vorherigem Bericht geschrieben, zu wenig Zeit hatten um auf eine neue Bibliothek zu setzen mussten wir uns etwas ausdenken mit dem wir den Balken verschleiern könnten. Nach langer überlegen sind wir auf die Idee gekommen, dass man zukünftig das Programm in zwei Lizenzvarianten anbieten könnten. Einmal eine Freie- Werbe Finanzierte- und eine Kostenpflichtige Version. So hätten wir den Balken mit einem Werbebanner überblenden können. Nach einer kurzen Recherche habe wir festgestellt, dass dies ohne ein Kleingewerbe eine Rechtliche Grauzone wäre, da wir für den Einbau von Echter Werbung eine WEB API wie zum Beispiel Google AdMob intrigieren müssten, da dadurch aber Umsätze generiert werden würden. Deswegen haben wir uns dazu entschieden einen Platzhalter einzubauen. Wir haben uns eine Handvoll von Bilder ausgesucht, welche „Werbung“-Darstellen sollen. Durch eine Web API wird bei jedem Programm Start eine zufallszahl Generiert, welche dann entscheidet welches Bild nun angezeigt wird. (Dies hätten wir auch per Java Random-Funktion lösen können, so konnten wir aber eine Web-API einbinden und somit den Umgang mit einer üben).

Als letzte Änderung am Programm haben wir noch das HelpMenu fertig gestellt, damit der Benutzer innerhalb des Programmes einfach Hilfestellungen finden kann.

Da wir das Öffnen und Speicher von pdf und docx Dokumente immer noch nicht Fehlerfrei abschließen konnten, haben wir uns dazu entschlossen diese Funktionen aus dem Programm zu entfernen, da diese nur Fehler in der Anwendung erzeugt haben.

Da wir das Programm als lauffähige Datei abgebe wollten haben wir mit verschiedenen Tools versucht, aus unserem Programm eine .exe-Datei zu erstellen. Da dies Leider nicht Funktioniert hat haben wir alle Programm-Dateien als Jar-File exportiert und im selben Ordner eine Batch Datei erstellt die unabhängig vom PC Textis ausführen kann mit den benötigten VM Argumenten. Wichtig hierbei ist das man die neuste JavaSDK Version auf dem PC installiert hat und sich der Textis Ordner

im Download Ordner befindet. Damit es aber nach eine Exe aussieht wurde eine Verknüpfung der Batch Datei erstellt und angepasst. Die Daten im Audio und Picture Ordner sind nur da, um zu sehen war alles verwendet wird. Diese Bilder und Audios sind aber in dem Jar-File selbst nochmals vorhanden damit kein Layer 8 Problem entsteht. Für die Erstellung der Batchdatei sehen Sie sich: Programm Start an.



Code Dokumentation

In diesem Abschnitt der Dokumentation werden die Wichtigsten abteile den Codes erklärt. Dabei werden Querverweise in den Wochenberichten erfolgen, um aufzuzeigen wann welche Funktion umgesetzt wurde. Wichtig zum Aufbau, dieses Abschnittes der Dokumentation ist, dass dieser Nach den Phasen der Projektziele aufgebaut ist. Da das Notiz Programm innerhalb der Esten sieben Tage revidiert wurde und in dessen Code in den Texteditor integriert wurde, gehen wir hier nur auf den Swing- und den FX- Texteditor ein.

Swing-Editor

Diese Code-Dokumentation bezieht sich auf die Wochenberichte 1 bis 3, welche den Projekt Zeitraum vom 11.04.2024 bis zum 28.04.2024 abdecken.

„Öffnen“-Funktion

In diesem Abschnitt gegen wir auf die Öffnen Funktion, diese gilt bespielt Haft auch für die Speicher Funktion da diese sehr ähnlich aufgebaut ist. Die Speicher Funktion nutzt dieselben Methoden aus Bibliotheken nur in einem andren Reihen folgen, weswegen wir uns dafür entschieden haben, beispielhaft die Öffnen Funktion zu beschreiben.

Diese Funktion wurde im Zeitraum des 15.04.2024 bis zum 21.04.2024, nachzulesen im Wochen Bericht 2 (15.04.2024 bis zum 21.04.2024).

```
JMenuItem mntmNewMenuItem_1 = new JMenuItem(language[2][languagechoose]);
mntmNewMenuItem_1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        FileDialog fileDialog = new FileDialog(projekttogether.this, language[2][languagechoose], FileDialog.LOAD);
        fileDialog.setVisible(true);

        String directory = fileDialog.getDirectory();
        String filename = fileDialog.getFile();

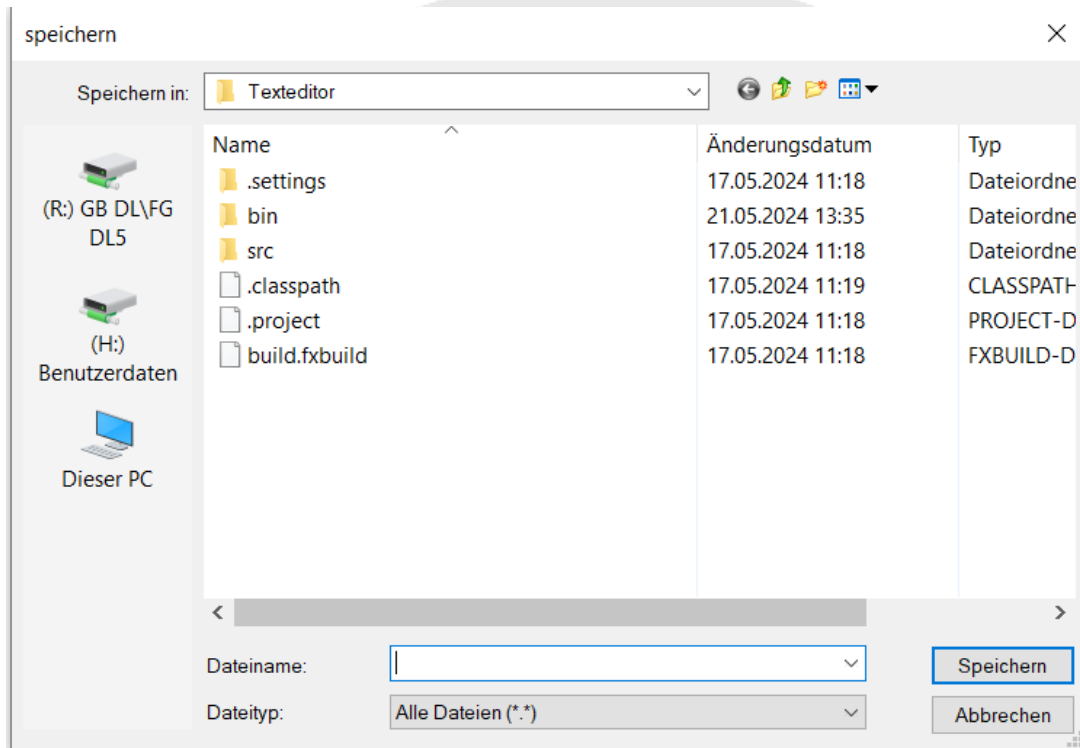
        if (directory != null && filename != null) {
            String filePath = directory + filename;
            currentFileName = filename;
            updateTitle();

            try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
                StringBuilder content = new StringBuilder();
                String line;
                while ((line = reader.readLine()) != null) {
                    content.append(line).append("\n");
                }
                textArea.setText(content.toString());
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
});
mnNewMenu.add(mntmNewMenuItem_1);
```

In diesem Programm ausschnitt wurde die öffnen Funktion realisiert. Sobald der Benutzer auf Speichern klickt, wird der "Windows-File Dialog" geöffnet, also das native Windows Dialogfeld.

```
FileDialog fileDialog = new FileDialog(projekttogether.this, language[2][languagechoose], FileDialog.LOAD);
fileDialog.setVisible(true);
```

Mit den oben gezeigten Code Zeilen, wird als erstes ein Dialogfenster erzeugt. Diesem wird ein Parentframe, also das dazugehörige Hauptfenster, zugewiesen. Also in unsrem Falle Projekttogether, das wird mit dem Ausdruck: Projekttogether.this, - erzielt. Das Dialog-Fenster unterstützt verschiedene Modis, wird benötigen zum Öffnen einer Datei also den „Öffnen“-Modus. Dafür muss man das Dialog Fenster entsprechen einstellen, dies realisiert man in dem man, das Dialog Fenster auf Load setzt. Dies wird mit dem Ausdruck: FileDialog.LOAD, erreicht. Dadurch wird nun ein Dialogfenster erzeugt, welche dem Hauptfenster zugewiesen und für die Öffnung einer Datei konfiguriert ist.



Nach dem der Benutzer einen Dateipfad (eine Datei) ausgewählt hat, und das Dialogfeld geschlossen hat, liest der Code den Namen der Datei aus.

```
if (directory != null && filename != null) {  
    String filePath = directory + filename;  
    currentFileName = filename;  
    updateTitle();  
}
```

Dafür wird geprüft, ob das Pfad den der Benutzer gewählt hat, nicht leer ist und die Datei auch einen lesbaren Namen hat, wenn dies der Fall ist, wird der Programm Namen durch den, der ausgewählten Datei ersetzt.

Wenn dies funktioniert hat, steht nun die „Konvertierung“ des Dokumenttextes in dem Textfeld unseres Programmes an.

```
try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {  
    StringBuilder content = new StringBuilder();  
    String line;  
    while ((line = reader.readLine()) != null) {  
        content.append(line).append("\n");  
    }  
    textArea.setText(content.toString());  
} catch (IOException ex) {  
    ex.printStackTrace();  
}
```

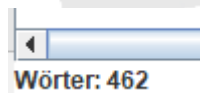
Der `BufferedReader` ist eine Bibliotheks-Funktion, welche es ermöglicht Dateien zeilenweise zu lesen. Der ausgelesene Inhalt wird in einen `StringBuilder` (auch eine Bibliotheks-Funktion) eingefügt. Sobald der komplette Text einer Datei erfasst wurde, wird dieser aus dem `StringBuilder` dem Textfeld übergeben und somit ist der „Konvertierungs“-Vorgang beendend.

Die Speicher Funktion läuft diese Logik rückwärts, der Benutzer wählt erst einen Speicherort und eventuell den Datei Namen, anschließend wird der Text aus dem Textfeld gelesen, und in eine Datei konvertiert.

Wörter-„Zähler“

In diesem Abschnitt gehen wir auf die Wort-Zähl Funktion ein, Diese Funktion wurde im Zeitraum des 15.04.2024 bis zum 21.04.2024, nachzulesen im Wochen Bericht 2 (15.04.2024 bis zum 21.04.2024).

Das Programm bietet am unteren linken Rand eine „Live“-zähler für die Anzahl der im Dokument / Text sich befindenden Wörter, ähnlich wie man es von Microsoft Word gewohnt ist.



Um dies zu ermöglichen haben wir zwei Methoden erstellt. Die Methode `updateWordCount` dient dabei als Ursprung.

```
private void updateWordCount() {  
    String text = textArea.getText();  
    int wordCount = countWords(text);  
    wordCountLabel.setText(language[26][languagechoose] + wordCount);  
}
```

Jedes Mal, wenn der Benutzer eine Taste, im Eingabefeld drückt, wird diese Methode aufgerufen. Dafür wird der Methode der komplette Text übergeben. Diesen Text übergibt die Methode `updateWordCount` dann an die Methode `countWords`. Diese berechnet die Anzahl der Wörter.

```
private int countWords(String text) {  
    if (text == null || text.isEmpty()) {  
        return 0;  
    }  
    String[] words = text.trim().split("\\s+");  
    return words.length;  
}
```

Im ersten Schritt wird geprüft ob Wörter geschrieben worden, wenn das nicht der Fall ist, wird an die updateWordCount Methode eine Null übergeben, und das Label in der GUI entsprechend aktualisiert. Wenn der Text nicht null ist, wird dieser in ein Array übertragen. Jedes Leerzeichen dient bei diesem Vorgang als ein Trennzeichen. Alles zwischen zwei Trennzeichen wird in einen Speicherplatz des Arrays übertragen. Im Anschluss zählt das Programm die Länge des Arrays und gibt diese der updateWordCount Methode zurück. Wobei die Länge des Arrays als Anzahl der Wörter dient. Mit dieser Zahl wird dann das Label in der GUI-Aktualisiert.

Spracheinstellungen

```
JMenuItem mntmNewMenuItem_2 = new JMenuItem(language[1][languagechoose]);
mntmNewMenuItem_2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        textArea.setText("");
        currentPageCharacters = 0;
        currentFileName = null;
        updateTitle();
    }
});
mnNewMenu.add(mntmNewMenuItem_2);
```

Im oben stehenden Code sieht man die Erstellung eines Menü-Buttons, dieser soll bei Betätigung, den aktuell Text des Textfelds löschen, den Dokumenten Namen in New ändern und die Wortzähler zurücksetzen. Diese Funktion verwenden wir hier nun exemplarisch, um die Implementation der Verschiedenen Sprachen zu demonstrieren.

In der ersten Zeile sieht man den Aufruf des Arrays, wo bei der ersten Variable für das gewählte Wort, in diesem Fall Neue steht, und languagechoose steht für die ausgewählte Sprache. Je nach dem Wert dieser, wird in den Button das Wort in einer anderen Sprache geladen.

Um zur oben genannten Funktion in der GUI den Text auszuwählen wird ein Aufruf auf das Array erstellt, dies funktioniert in dem man das Array aufruft um dann auf die Gewünschte Speicherzelle zuzugreifen. Allgemein sieht so ein Zugriff so aus: GUI-Button () <- in diese Klammern schreibt man normalerweise den gewünschten Text hinein, um auf das 2D-Array zuzugreifen sieht es so aus: GUI-Button (Array-Name [erste Dimension] [Zweite Dimension])

In unserem Fall greifen wir auf das Array language zu, in diesem gibt es 2-Dimensionen, man kann sich dies wie ein Regal vorstellen, es gibt mehrere Regal Bretter und auf jedem Regal Brett stehen mehrere Kisten. Die Regalbretter stellen die erste Dimension und die Kisten die zweite Dimension dar. Also sind die Wortgruppen zum Beispiel neu, bei uns ein Regalbrett, während die verschiedenen Wortvarianten (also sprachen), in den Kisten sind. Oder man kann sich ein 2D-Array auch wie ein Schachbrett vorstellen, in welchem jedes Feld durch die Kombination durch Zahlen und Buchstaben einmal—und so zu identifizieren ist.

Um nun auf das Wort neu zu zugreifen, müssen wir erst in unserem Array nachschauen an welcher Stelle es sich befindet. (ein Array beginnt immer bei null, das bedeutet das die Erste Wortgruppe Null- und die erste dort drin, enthält Sprache auch auf dem Speicher 0 liegt. Neu befindet sich in der Zweiten Wortgruppe, also auf Speicherplatz 1, die Deutsche Wortvariante befindet sich auf dem

ersten Speicherplatz, also Null, somit ist der Speicherplatz des Wortes neu 1:0, damit ist es einmal und für uns zu identifizieren.

Nachdem uns nun der Speicherplatz des Arrays bekannt ist, können wir nun den Aufruf erstellen und dem GUI-Button seinen gewünschten Text zuteilen. Der Finale Aufruf sieht wie folgt aus: GUI-Button (language[1][0]).

Da wir mit dieser Varianten des Aufrufes nun zum Ändern der Sprache auf Englisch den Code immer händisch abändern müssen, ist eine Änderung der Sprache noch nicht während der Programm Laufzeit möglich.

Um dies zu ermöglich müssen wir den Array Aufruf „dynamisch“ gestalten. Da alle Elemente immer dieselbe Sprache verwenden sollen, kann man eine Variable zur Sprachfestlegung erstellen. In unserem Fall die languagechoose Variable. Nun können wir unserem Array Aufruf anpassen, in dem wir die Zahl zur Sprachauswahl gegen die Variable austauschen. Der neue Array Aufruf seit nun so aus: GUI-Button (language[1][languagechoose]). Somit zeigt die GUI diesen Knopf immer in der Sprache an, welche mit der languagechoose-Variable am Programm beginn festgelegt wurde.

Somit ist eine Grundlage zur Sprachanpassung während der Laufzeit ermöglicht.

```
private int languagechoose = 0;  
private String[][] language = {  
    {"Datei", "File", "Archivo", "Fichier", "文件"},  
    {"neu", "new", "nuevo", "nouveau", "新"},  
};
```

FX-Editor

Diese Code-Dokumentation bezieht sich auf die Wochenberichte 4 bis 9, welche den Projekt Zeitraum vom 29.04.2024 bis zum 05.06.2024 abdecken.

Erstellung des Fensters per Java-FX

In diesem Abschnitt werden die Erstellung und „Konfiguration“ des Main-Fensters erklärt.

```
@Override  
public void start(Stage primaryStage) {
```


In der public void start, wird das „Haupt“-Fenster unserer Anwendung aus der Java-FX Bibliothek abgerufen. Nun kann auf das Stage- „Objekt“ zugegriffen werden. Alle nachfolgenden Codebeispiele und Erklärungen befinden sich in der „showMainStage“-Methode.

```
// Primary Icon  
Image Logo = new Image("file:\\Users\\020700012023\\OneDrive - CGM\\Desktop\\Textis\\Logo.png");  
primaryStage.getIcons().add(Logo);
```

In diesen Zeilen wird ein Bild in eine Variable geladen. Diesem Bild wird nun an das Icon des Fensters angeheftet, Dadurch wird das Standard FX Logo überschrieben und unser Logo ist nun in der Steuerleiste den Fenster- und in der Taskleiste sichtbar.

```
// File Menu and Buttons  
MenuBar menuBar = new MenuBar();  
Menu fileMenu = new Menu("File");  
MenuItem newItem = new MenuItem("New");  
MenuItem openItem = new MenuItem("Open...");  
MenuItem saveAsItem = new MenuItem("Save As...");  
SeparatorMenuItem separatorMenuItem1 = new SeparatorMenuItem();  
MenuItem settingsItem = new MenuItem("Settings");  
MenuItem printItem = new MenuItem("Print");  
MenuItem quitItem = new MenuItem("Quit");
```

In diesem Code wird eine Menü Bar festgelegt, welche unterhalb des Fensterrahmens angezeigt wird. Dieser wird nun ein Menü hinzugefügt, nämlich das „File“ Menü.



File Settings Help

Dem File Menü werden nun Menü Items hinzugefügt, diese Menü Items dienen als Halter für Buttons.

```
// Adding all File Buttons  
fileMenu.getItems().addAll(  
    newItem, openItem, saveAsItem, separatorMenuItem1,  
    settingsItem, printItem, quitItem  
);
```

In diesem Abschnitt werden nun die Buttons die dem Benutzer, zur Ausführung verschiedener Programm Funktionen bereitstehen, hinzugefügt.

Dieser Vorgang wird nun auch für die Weiteren Menüs Settings und Help hinzugefügt. Wichtig zu erwähnen ist, dass das Menü Item Settings vorher für das Edit Menü bestimmt war. Das Edit Menü wurde entfernt und das als extra Fenster, erstellte Settings Menü, wurde in Programm integriert,

mehr dazu in den Wochenberichten.

```
menuBar.getMenus().addAll(fileMenu, settingsMenu, helpMenu);|
```

In dieser Zeile werden der Menü Leiste, die Menüs zugewiesen. Dies geschieht unabhängig von der Erstellung der Menüs und ist zum Anzeigen der Menüs notwendig.

Erstellung der Speichern- / Öffnungsfunktion

In diesem Abschnitt gehen wir auf die Öffnen Funktion, diese gilt beispielhaft auch für die Speicher Funktion da diese sehr ähnlich aufgebaut ist. Die Speicher Funktion nutzt dieselben Methoden aus Bibliotheken nur in einer anderen Reihenfolge, weswegen wir uns dafür entschieden haben, beispielhaft die Öffnen Funktion zu beschreiben.

```
FileChooser fileChooser = new FileChooser();  
FileChooser.ExtensionFilter txtFilter = new FileChooser.ExtensionFilter("Text files (*.txt)");  
FileChooser.ExtensionFilter htmlFilter = new FileChooser.ExtensionFilter("HTML files (*.html)");  
fileChooser.getExtensionFilters().addAll(txtFilter, htmlFilter);
```

Mit dem gezeigten Code rufen wir einen sogenannten FileChooser-Dialog auf, also ein Windows Dialogfenster zur Auswahl von Dateien. Dies ermöglicht es dem Benutzer, sich eine Datei, welche auf dem System des Benutzers befindet, aufzurufen. Da unser Programm nur bestimmte Dateitypen verarbeiten kann, bauen wir Filter ein, damit der Benutzer nur unterstützte Dateien auswählen kann und keinen ungewollten Fehler entstehen.

Damit der Benutzer das Dialogfenster sieht und mit diesem interagieren kann, muss es eingeblendet werden. Das Dialogfenster rufen wir mit der Methode: "showOpenDialog" auf.

```
File file = fileChooser.showOpenDialog(primaryStage);
```

Falls der Benutzer eine Datei ausgewählt hat (d.h. das file-Objekt ist nicht null), wird der Inhalt dieser Datei gelesen. Dazu wird ein BufferedReader verwendet, der zeilenweise den Inhalt der Datei in einen StringBuilder einliest.


```
if (file != null) {  
    try {  
        StringBuilder content = new StringBuilder();  
        BufferedReader reader = new BufferedReader(new FileReader(file));  
        String line;  
        while ((line = reader.readLine()) != null) {  
            content.append(line).append("\n");  
        }  
        reader.close();  
    } catch (IOException ex) {  
        ex.printStackTrace();  
    }  
}
```

```
String fileName = file.getName().toLowerCase();  
if (fileName.endsWith(".txt")) {  
    htmleditor.setHtmlText(content.toString());  
} else if (fileName.endsWith(".html")) {  
    htmleditor.setHtmlText(content.toString());  
}
```

Nach dem erfolgreichen Einlesen des Dateiinhalts wird der Dateiname überprüft, um festzustellen, ob es sich um eine Text- oder HTML-Datei handelt. Diese Unterscheidung ist wichtig, da der HTML-Editor HTML-Inhalte korrekt rendern kann, während reiner Text ohne HTML-Tags angezeigt wird. Der Inhalt wird dann entsprechend im HTML-Editor gesetzt.

Erstellung der „Neu“ Funktion

Bei der Neu-Funktion wird der Aktuelle Text, gegen eine Leere Zeichenfolge getauscht.

```
public void newFile() {  
    // Der Inhalt des HTMLEditors wird auf eine leere Zeichenkette gesetzt  
    htmleditor.setHtmlText("");  
}
```

Ladebildschirm

Der Ladebildschirm wurde als "externes"-Programm geschrieben, welches wir in unseren "Main"-Code aufrufen.

```
private Stage splashStage;  
private ProgressBar progressBar; // Objekt zum Darstellen des Ladebalkens
```

Im ersten Schritt erstellen wir zwei Objekte, eins für die Stage des Ladebildschirm-Fensters, um dort GUI-Elemente platzieren zu können. Das zweite Objekt "ProgressBar" wird zum Erstellen und Darstellen des Ladebalkens benötigt.

```
splashStage = new Stage();  
splashStage.initStyle(StageStyle.UNDECORATED);  
  
Scene scene = new Scene(root, 500, 300);  
splashStage.setScene(scene);
```

Im oben gezeigten Code-Ausschnitt, passen wir das Fenster an, als erstes, übergeben wir dem Fenster die Eigenschaft "UNDECORATED", dies bedeutet das die Titelleiste am oberen Rand des Ladebildschirms, ausgeblendet werden, damit der Benutzer dieses nicht schließen kann.

```
ImageView splashImage = new ImageView(new Image("file:C:\\User  
splashImage.setFitWidth(400); // Setzen der Breite  
splashImage.setFitHeight(200); // Setzen der Höhe  
splashImage.setPreserveRatio(true); // Verhältnis beibehalten  
root.setCenter(splashImage); // Platzieren des Bildes
```

Damit auf dem Ladebildschirm auch noch unser Logo angezeigt wird, müssen wir dieses aus einer Datei "auslesen", und per GUI-Element auf die Gewünschte Fläche der GUI setzen.

Als erstes definieren wir unser Bild und weisen ihm die Datei zu. Im nächsten Schritt geben wir die Höhe und Breite des Bildes in Pixeln an. Mit ".setpreserveRatio(true);" kann man dafür sorgen, dass das Seitenverhältnis des Bildes nicht verändert wird, damit dieses nicht verzogen, dargestellt wird. Als letztes setzen wir das Bild in die Mitte des Fensters.

```
public void show() { // Zeigt das Ladefenster an  
    splashStage.show();  
}  
  
public void hide() { // Versteckt das Ladefenster  
    splashStage.hide();  
}  
  
public void updateProgress(double progress) {  
    Platform.runLater(() -> progressBar.setProgress(progress));  
}
```

Des Weiteren gibt es noch drei Methoden. Wobei die erste Methode zum Anzeigen und die zweite zum Ausblenden des Ladebildschirms ist. Diese Methoden werden, über unser Hauptprogramm, "aktiviert". Mit der letzten Methode wird der Fortschritt, im Ladebalken aktualisiert.

Im Hauptprogramm gibt es noch dazugehörigen Code.

```
@Override
public void start(Stage primaryStage) {
    splashScreen = new SplashScreen();
    splashScreen.show();

    // Simuliere Ladeaufgaben im Hintergrund
    new Thread(() -> {
        for (int i = 0; i <= 100; i++) {
            try {
                Thread.sleep(50); // Simulierte Ladezeit
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            splashScreen.updateProgress(i / 100.0);
        }

        Platform.runLater(() -> {
            splashScreen.hide();
            showMainStage(primaryStage);
        });
    }).start();
}
```

Dieser Code simuliert mit einer for-Schleife eine Ladezeit. Das Programm geht 100-mal, jedes Mal steht für einen Prozent des Ladebalkens, in die Schleife und "pausiert" die GUI für 50ms. Dadurch entsteht die warte Zeit. Diese haben wir so angepasst, dass diese ungefähr mit dem Start Zeit der haupt-GUI übereinstimmt und der Benutzer das gefüllt eines richtigen Ladebildschirmes, bekommt.



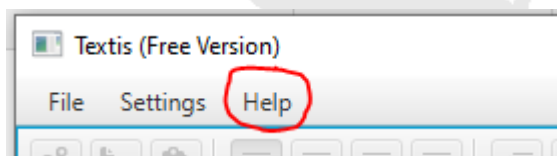
Erstellung der Spracheinstellungen

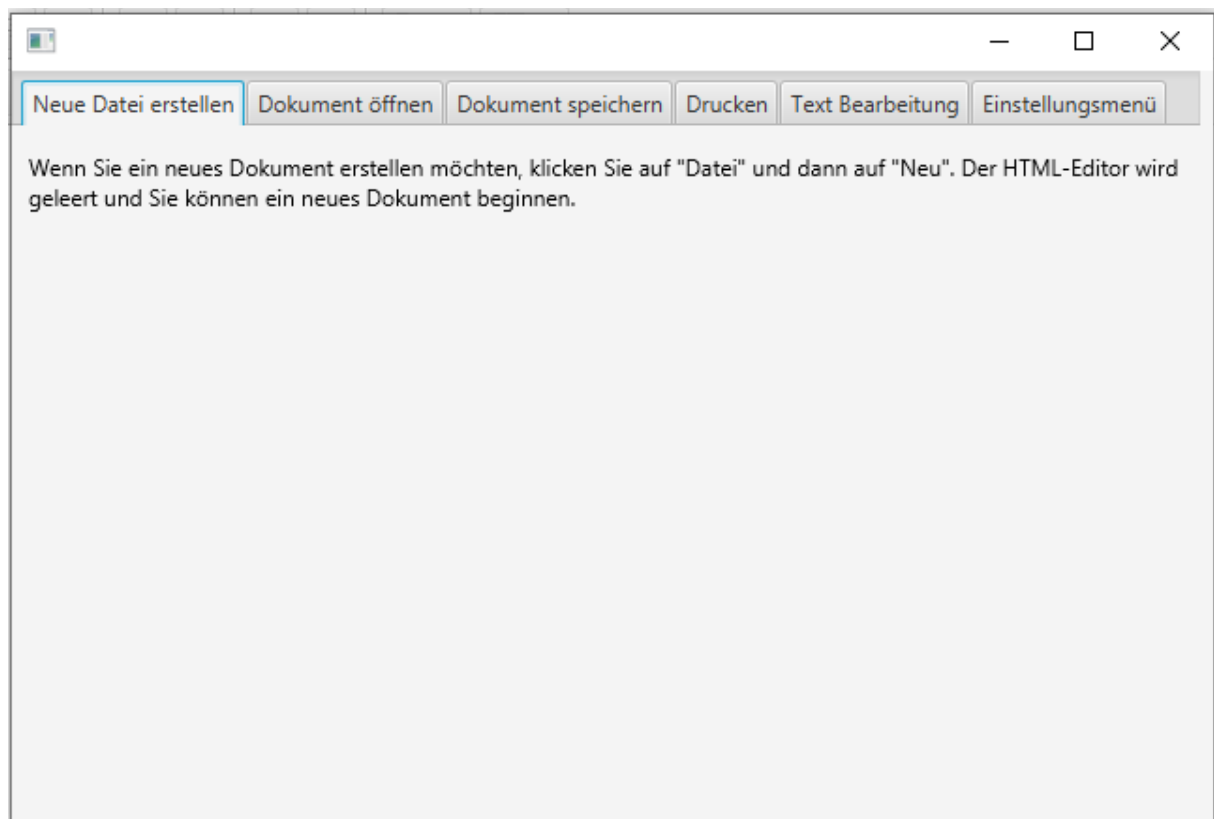
Durch das Betätigen der jeweiligen Sprach Buttons z.B. German (Englisch ist die Grundeinstellung). Die Inhalte der Buttons wird dann durch den Befehl `.setText()` mit dem neuen Inhalt einfach ersetzt. Dies wird dann für die jeweiligen Sprachen (aktuell: Englisch, Deutsch, Spanisch und Italienisch) wiederholt.

```
germanItem.setOnAction(e ->{  
    fileMenu.setText("Datei");  
    newItem.setText("Neu");  
    openItem.setText("Öffnen");  
    saveAsItem.setText("Speichern als");  
    printItem.setText("Drucken");  
    quitItem.setText("Schließen");  
    settingsMenu.setText("Einstellungen");  
    englishItem.setText("Englisch");  
    germanItem.setText("Deutsch");  
    spanishItem.setText("Spanisch");  
    italianItem.setText("Italienisch");  
    helpMenu.setText("Hilfe");  
    aboutItem.setText("Über Textis");  
});
```

Erstellung des Help-Menü

Das HelpMenu wird über eine separate Class erzeugt. Diese wird durch einen Button im Menü Band der GUI aufgerufen.





Das HelpMenu ist in verschiedene Tabs aufgeteilt, jeder Tab steht dabei für einen Hilfe Text.

```
@Override
public void start(Stage primaryStage) {
    String[] tabTitles = {
        "Neue Datei erstellen",
        "Dokument öffnen",
        "Dokument speichern",
        "Drucken",
        "Text Bearbeitung",
        "Einstellungsmenü"
    };

    String[] tabTexts = {
        "Wenn Sie ein neues Dokument erstellen möchten, klicken Sie auf \"Datei\" und da",
        "Um ein bestehendes Dokument zu öffnen, klicken Sie auf \"Datei\" und dann auf \",
        "Um das aktuelle Dokument zu speichern, klicken Sie auf \"Datei\" und dann auf \",
        "Wenn Sie Ihr Dokument drucken möchten, klicken Sie auf \"Drucken\".",
        "Das Menü \"Bearbeiten\" bietet Funktionen zur Textbearbeitung.",
        "Wenn Sie die Sprache der Benutzeroberfläche ändern möchten, öffnen Sie die \"Ei"
    };
}
```

```
@Override
public void start(Stage primaryStage) {
    String[] tabTitles = {
        "Neue Datei erstellen",
        "Dokument öffnen",
        "Dokument speichern",
        "Drucken",
        "Text Bearbeitung",
        "Einstellungsmenü"
    };
}
```

Im ersten Schritt werden zwei Arrays erstellt, tabTitles speichert die Title- und tabTexts speichert die entsprechenden Texte für die Tabs. (Wie genau ein Fenster in Java-FX erstellt wird entnehmen Sie bitte)

```
for (int i = 0; i < tabTexts.length; i++) {
    Tab tab = new Tab(tabTitles[i]);
    AnchorPane tabContent = new AnchorPane();
    Text tabText = new Text(tabTexts[i]);
    tabText.setLayoutX(9.0);
    tabText.setLayoutY(27.0);
    tabText.setWrappingWidth(615.224609375);
    tabContent.getChildren().add(tabText);
    tab.setContent(tabContent);
    tabPane.getTabs().add(tab);
}
```

Die Tabs werden mit Hilfe dieser for Schleife erstellt. Diese Schleife „setzt Tab Title und Tab Text zusammen“ und fügt diese dem Fenster hinzu.

API Nutzung

Für die optionale Funktion haben wir uns für die API von random.org entschieden die dann auf Anfrage des Programms hin eine Zahl von 1 bis 5 wiedergibt was dann beim Start des Programms bestimmt welches Bild unten angezeigt wird um den nicht nutzbaren Bereich zu verschleiern

```
public class Main extends Application {
    private SplashScreen splashScreen;
    private ImageView bottomImage;
    String currentbottomimage;
    @Override
    public void start(Stage primaryStage) {
        //API Number request
        try {
            SSLContext sc = SSLContext.getInstance("SSL");
            sc.init(null, new TrustManager[] { (TrustManager) new X509TrustManager() {
                public X509Certificate[] getAcceptedIssuers() {
                    return null;
                }
            } }, null);
            public void checkClientTrusted(X509Certificate[] certs, String authType) {
            }
        }
    }
}
```

“Werbe-“ Bilder

Die “Werbe-“Bilder dienen hauptsächlich zur Verschleierung des nicht nutzbaren Bereiches des HTMLEditors (Wochen Bericht 9 (03.06.2024 bis zum 05.06.2024)). Diese werden zufällig beim Start durch die API ausgesucht und sobald das Fenster eine Mindestgröße von 100x700 erreicht, hat unten angezeigt. Falls es kleiner oder gleich sein sollte, wird das Bild mit dem Anchor entfernt.

```
//Picture remover/setter
ChangeListener<Number> resizeListener = (obs, oldVal, newVal) -> {
    if (primaryStage.getWidth() < 701 || primaryStage.getHeight() < 1000) {
        if (bottomImage != null) {
            root.getChildren().remove(anchorPane);
            bottomImage = null;
        }
    } else {
        if (bottomImage == null) {
            bottomImage = new ImageView(new Image(currentbottomimage));
            bottomImage.setFitWidth(1920);
            bottomImage.setFitHeight(380);
            anchorPane.getChildren().add(bottomImage);
            root.getChildren().add(anchorPane);
        }
    }
};

primaryStage.widthProperty().addListener(resizeListener);
primaryStage.heightProperty().addListener(resizeListener);
//Picture remover/setter
```

Programm Start

```
@echo off
cd /d %USERPROFILE%\Downloads\Textis\Ressources\Programm
start javaw --module-path "%USERPROFILE%\Downloads\javafx-sdk-22.0.1\lib" --add-modules
javafx.controls,javafx.graphics,javafx.fxml,javafx.web -jar Textis.jar
```

Alles befindet sich nochmals für den einfachen Download auf Github:

[YMGMengsk/Textis: Schoolproject \(github.com\)](https://github.com/YMGMengsk/Textis: Schoolproject)

Zusammenfassung und Fazit

Im Rahmen des Projekts haben wir erfolgreich einen Texteditor mit einer grafischen Benutzeroberfläche (GUI) entwickelt, der zahlreiche Funktionen wie Spracheinstellungen, Textformatierungsoptionen und eine Hilfe-Funktion bietet. Trotz einiger technischer Herausforderungen, insbesondere mit der Skalierbarkeit des Textfeldes und der Implementierung von Spracheinstellungen zur Laufzeit, konnten wir Lösungen finden, wie etwa die Einbindung einer Web-API zur Anzeige von „Werbepildern“.

Ein wesentlicher Fehler war die anfängliche Wahl von Java Swing für die GUI-Entwicklung. Swing erwies sich als nicht flexibel genug, insbesondere bezüglich der Spracheinstellungen und der gewünschten Optik. Diese Erkenntnis kam spät im Projekt, was zu einem Mehraufwand führte, als wir zu Java FX wechseln mussten. Der Wechsel erforderte eine komplette Neuentwicklung der bestehenden GUI und Anpassung der Funktionen, was die Projektzeit und Ressourcen beanspruchte.

Ein weiteres Problem war die Implementierung der Spracheinstellungen. Unsere erste Lösung, alle Texte in einem 2D-Array zu speichern, erwies sich als eher schlecht, da die GUI-Elemente nicht dynamisch aktualisiert werden konnten. Obwohl wir später auf Properties-Dateien umgestiegen sind, gelang es uns nicht, die Sprache während der Laufzeit zuverlässig zu ändern. Dies lag daran, dass die GUI-Elemente sich nach einer Sprachänderung nicht automatisch aktualisierten.

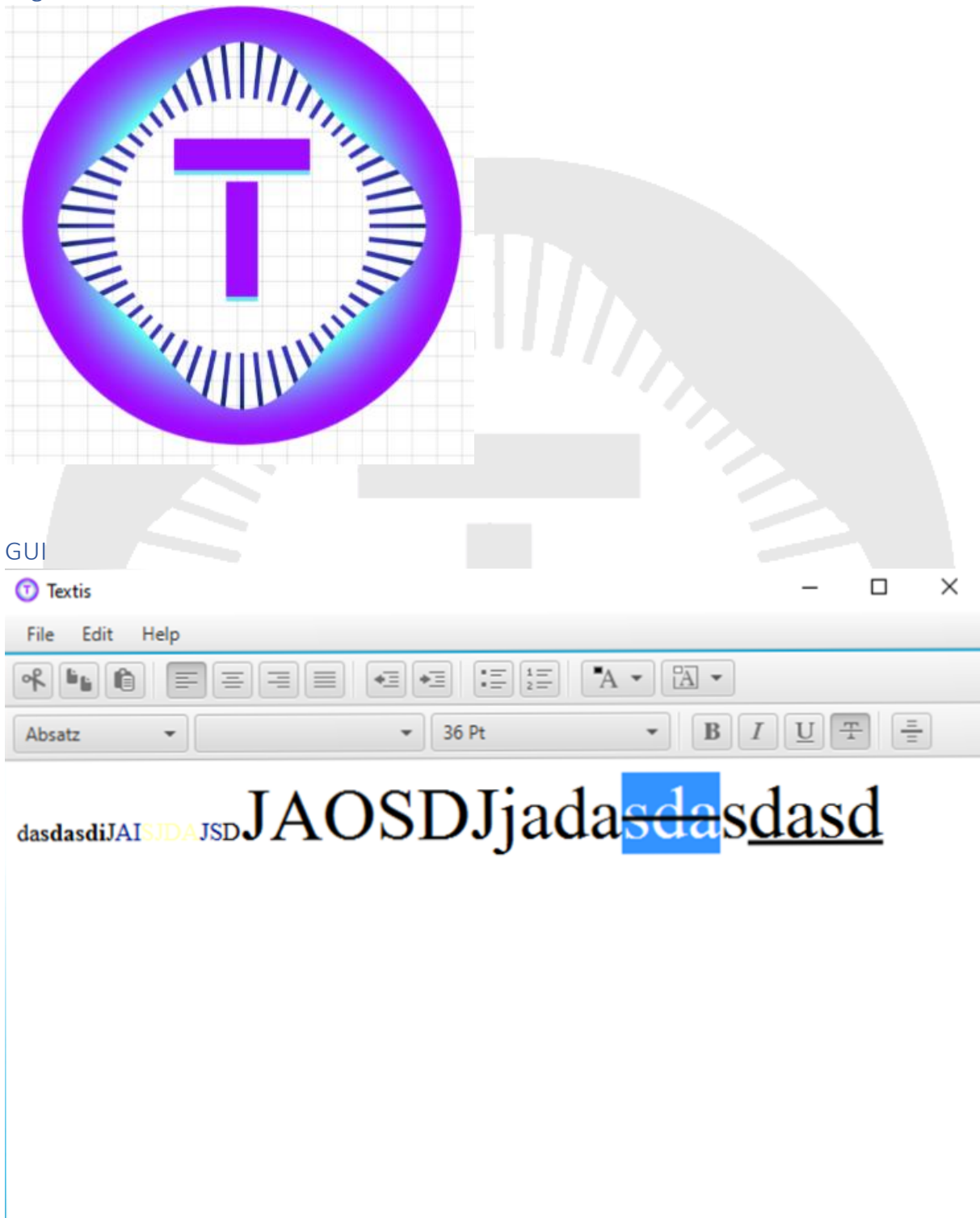
Die Textformatierungsoptionen funktionierten zunächst nur für das gesamte Dokument und nicht für individuell ausgewählte Textteile. Dieses Problem trat insbesondere bei der Nutzung von Java Swing auf und konnte erst mit dem Wechsel zu Java FX und dem Einsatz eines Rich-Text-Fields gelöst werden. Der erste Ansatz, Formatierungsänderungen nur visuell in der GUI darzustellen, ohne sie in die Dokumentenspeicherung zu integrieren, war ein Fehler, der zusätzlichen Entwicklungsaufwand verursachte.

Frühere Tests der verwendeten Bibliotheken hätte uns geholfen, die Einschränkungen von Java Swing schneller zu erkennen und den Wechsel zu Java FX früher durchzuführen. Eine detailliertere Planung und das Erstellen von Prototypen für schwierige Funktionen wie den Sprachwechsel hätten viele Probleme verhindern können. Durch frühes Testen von wichtigen Funktionen hätten wir die Machbarkeit und Stabilität besser gewährleisten können.

Insgesamt sind wir jedoch zufrieden mit dem Ergebnis und den Erfahrungen, die wir während dieses Projekts gesammelt haben. Diese Erfahrungen werden uns in zukünftigen Projekten von großem Nutzen sein.

Anlagenverzeichnis

Logo



Miro Board

Das Miro Board enthält unseren Programm Ablaufplan.

https://miro.com/app/board/uXjVKIs3cGM=/?share_link_id=114906450610

GitHub

In dem GitHub finden Sie alle Programm Dateien.

[GitHub - YMGMengsk/Textis: Schoolproject](#)

Programm Code

Denn Programm Code entnehmen Sie bitte dem GitHub. Wir bitten um Ihr Verständnis das wir den Programm Code nicht sinnvoll, hier in der Dokumentation einbinden können.

Entwicklungsumgebung

Während des gesamten Projektes diente uns Eclipse als Entwicklungsumgebung.

[Eclipse Installer 2024-03 R | Eclipse Packages](#)

Als Erweiterung haben wir das Add-In Code Together verwendet.

[CodeTogether Live | Eclipse Plugins, Bundles and Products - Eclipse Marketplace | Eclipse Foundation](#)

Als zusätzliche GUI Bibliothek haben wir JAVA-FX verwendet.

[JavaFX \(openjfx.io\)](#)

Zum Erstellen der Java-FX GUI haben wir den Scene Builder von Gluon verwendet.

[Scene Builder - Gluon \(gluonhq.com\)](#)

[YMGMengsk/Textis: Schoolproject \(github.com\)](#)