Ein Website-Baukasten für interaktive Vorlesungsskripte und Arbeitsblätter

Abschlussarbeit
zur Erlangung des Grades Bachelor of Science

von

John Hundhausen

Fachgebiet:	Theoretische Informatik / Formale Methoden
Gutachter:	Prof. Dr. Martin Lange Prof. Dr. Claude Draude
Betreuer:	Dr. Norbert Hundeshagen
Prüfungsdatum:	9.03.2023

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen verwendet habe und nur die nach der Prüfungsordnung der Universität Kassel zugelassenen Hilfsmittel verwendet habe.

Kassel, März 2023

John Hundhausen

Inhaltsverzeichnis

1 Einleitung

1.1 Didaktischer Hintergrund und Motivation

Lehrmaterialien bestehen heutzutage aus mehr als nur Text und Bildern. Moderne Web-Technologien machen es möglich, interaktive Visualisierungen, verschiedene Übungen und sogar Übungen, die automatisch korrigiert werden, bereitzustellen.

Verschiedene Lehrmaterialien können auf unterschiedliche Weise den Vorlesungsbetrieb bereichern. Es ist Konsens, dass Visualisierungen eine effektive Lehrmethode darstellen [1]. Interaktive Visualisierungen, zum Beispiel von Algorithmen, erlauben es zudem, angepasst an den Lernenden Ideen zu erkunden und dabei selbst zu experimentieren [2]. Automatische Korrektursysteme haben den Vorteil, dass eine große Menge an Aufgaben bzw. Tests angeboten werden kann, ohne dass Arbeitszeit der Studenten oder der Mitarbeiter des Fachgebiets für das Kontrollieren aufgebracht werden muss. Als ehemaliger Mitarbeiter und Kontrolleur von Übungen im Fachgebiet Theoretische Informatik / Komplexe Systeme an der Universität Kassel weiß ich, wie lange es dauern kann, Übungen zu kontrollieren und zu bewerten. Ein anderer Vorteil ist, dass die Studierenden direkt ein Feedback erhalten. Für einen langfristigen Erfolg beim Lernen ist wiederholtes Testen eine effektivere Methode, als sich Lehrmaterialien nur erneut durchzulesen [3], [4]. Trotzdem scheint das erneute Durchlesen von Texten und Notizen eine häufiger angewandte Lernmethode zu sein [5]. Eine höhere Verfügbarkeit und leichtere Durchführbarkeit von Tests könnte daher dafür sorgen, dass Studenten effizienter lernen. Aktive Lehrmethoden könnten auch dabei helfen, dass die Konzentration der Lernenden nicht so schnell nachlässt [6].

Im Fachgebiet Theoretische Informatik / Formale Methoden an der Universität Kassel entstehen in verschiedenen Projekten neue Visualisierungen, Web-Apps für Aufgabentypen und automatische Korrektursysteme, die den Lehrbetrieb bereichern sollen. Beispiele dafür sind eine Webanwendung zur prädikatenlogischen Modellprüfung von Graphstrukturen [7], ein Reduktions-Trainer im Kontext der Berechenbarkeitstheorie [8], [9], eine Webanwendung zur Modellierung von Problemen durch boolesche Formeln [10] und deren Visualisierung und Überprüfung [11] sowie ein Backendservice um endliche Automaten zu benoten [12]. Klassische Inhalte wie Vorlesungsskripte und Arbeitsblätter werden hingegen meistens in LaTeX oder in Markdown geschrieben und als PDF bereitgestellt.

Eines der Hauptziele des Fachgebietes im Bereich der Entwicklung und Erforschung von Bildungstechnologien ist eine Vereinheitlichung und Verknüpfung dieser Lernangebote. Dazu sollen die verschiedenen Projekte direkt in die Vorlesungsskripte und Arbeitsblätter integriert werden und als einheitliche Website bereitgestellt werden.

Studenten sollen durch diese einheitliche Lernumgebung eine bessere Lernerfahrung

geboten bekommen. Die interaktiven Inhalte wie Visualisierungen und Aufgaben sollen alle direkt passend zum Inhalt der Vorlesungsskripte oder Übungsblätter bereitstehen und so verfügbarer werden. Die interaktiven Lernangebote sollen nicht nur einzeln für sich verfügbar sein, sondern zusammengeführt ein einheitliches und verknüpftes interaktives Lernangebot bieten. Die Studenten sollen nicht verschiedene Webseiten aufrufen müssen, um beispielsweise die Aufgaben eines Arbeitsblattes zu erledigen, oder selbst auf einer anderen Website ein Beispiel eingeben müssen, um eine interaktive Visualisierung geboten zu bekommen.

Damit dies praktikabel genug für den Vorlesungsbetrieb ist, soll es für den Dozenten und andere Mitarbeiter des Fachgebiets auf einfache Weise möglich sein, diese interaktiven Vorlesungsskripte und Arbeitsblätter zu erstellen und zu bearbeiten.

1.2 Anforderungen und Ziel der Arbeit

Zur Erfüllung dieser Ziele soll nun ein zellbasierter Website-Baukasten erstellt werden, der es auf benutzerfreundliche Art und Weise erlaubt, diese komplexen Vorlesungsskripte und Arbeitsblätter zu erstellen. Dieser Website-Baukasten soll es ermöglichen, die verschiedenen Lehrmaterialien zusammenzuführen. Interaktive Lerninhalte sollen in Arbeitsblätter und Vorlesungsskripte integrierbar sein. Dabei sollen moderne Web-Technologien benutzt werden. Damit eine große Anzahl von interaktiven Inhalten entwickelt werden kann, sollte es einfach sein, neue Arten von Inhalten zu implementieren. Da in dem Fachgebiet bereits einige komplexe Web-Anwendungen für den Lehrbetrieb existieren, sollte es auch einfach sein existierende Software zu integrieren Die Webanwendung soll dabei Daten der Benutzer speichern können, damit zum Beispiel Lösungen von Aufgaben oder die Benotung einer Übung, gespeichert werden können. Zum Erstellen von klassischen Inhalten soll der Baukasten dabei die Auszeichnungssprache Markdown verwenden. Ein großer Teil der Vorlesungsskripte besteht meist aus mathematischen Ausdrücken, deswegen ist auch der Mathemodus aus LaTeXzu integrieren. Somit können Inhalte ähnlich wie gewohnt verfasst werden. Des Weiteren soll, um eine effiziente Bedienung zu ermöglichen, die Bedienbarkeit durch die Tastatur beachtet werden. Um Studenten zu ermöglichen unterwegs Lehrmaterial Arbeitsblätter und Vorlesungsskripte anzuschauen, sollen diese hinzu kommend auch auf mobilen Geräten verfügbar sein. Das bedeutet nicht, dass dies für alle interaktiven Inhalte erforderlich ist. Außerdem muss der Editor selbst nicht auf mobilen Geräten funktionieren. Es gibt zudem viele denkbare Erweiterungsmöglichkeiten und Ideen für zukünftige Projekte, die den Studierenden und Dozenten einen Mehrwert bringen sollen. Diese müssen bei der Entwicklung mitbedacht werden. Die gemeinsame Schnittstelle, die den Tools durch den Baukasten vorgegeben wird, soll zur Kommunikation zwischen den verschiedenen Projekten genutzt werden können. Somit könnten Projekte miteinander Verbunden werden. Diese Software soll später auch zur Lerndiagnostik genutzt werden. Es sollen Lerninhalte und Feedback ggf. gezielt bereitgestellt werden können. In einer späteren Version soll es zudem auch möglich sein, die Inhalte als PDF oder LaTeX-Dokument zu exportieren.

Gegenstand dieser Arbeit ist nun ein Prototyp, der einen solchen zellbasierten Website-Baukasten für interaktive Vorlesungsskripte und Arbeitsblätter in grundlegender Funktionalität umsetzt. Damit soll die Frage erforscht werden, wie moderne Ansprüche an Vorlesungsskripte und Arbeitsblätter sowohl technisch als auch konzeptionell umgesetzt werden können. Hierbei liegt der Fokus auf den Fachgebieten der Informatik und auf die praktikable Erstellung von Lehrmaterialien. Genau genommen beschäftigen wir uns zwar mit den Anforderungen des Fachgebiets Theoretische Informatik / Formale Methoden der Universität Kassel, das Konzept lässt sich aber auch auf ähnliche Fachgebiete übertragen. Im Rahmen dieser Arbeit ist nur das Frontend entstanden. Das Backend wurde parallel in Abstimmung in einem Masterprojekt entwickelt [13]. Im Verlauf der Arbeit werden für ein besseres Verständnis Teile der Konzepte und Technologien des Backends beschrieben, wobei diese im Text kenntlich gemacht sind. Für ein tieferes Verständnis des Backends ist dennoch auf dessen Dokumentation[1] zu verweisen.

Das Konzept der entstandenen Web-App ist ein zellbasierter bzw. blockbasierter Website-Baukasten. Eine Instanz der Anwendung besteht dabei aus verschiedenen Seiten, die beliebig geschachtelt werden können. Diese Seiten stellen entweder ein Kapitel eines Vorlesungsskriptes oder ein Übungsblatt dar. Eine Seite besteht aus verschiedenen Abschnitten, die Blöcke genannt werden. Diese Blöcke stellen verschiedene Formen von Inhalten da. Es gibt Markdown-Blöcke, um Text zu schreiben, aber auch andere Formen von Blöcken, die verschiedenste Inhalte und interaktive Elemente darstellen können. Jeder Block hat einen Bearbeitungsmodus, damit der Dozent die Inhalte bearbeiten kann, und einen normalen Modus, der das für die Studenten sichtbare Ergebnis darstellt.

Ein Kernkonzept dieses Projekts ist es, dass durch ein Plugin-System möglichst einfach neue Arten von Blöcken bzw. Arten von Inhalten entwickelt werden können. Das Plugin-System ist eines der Alleinstellungsmerkmale dieses Projekts. Durch die Kombination von verschiedenen Möglichkeiten neue Blocktypen zu implementieren können verschiedene Anforderungen erfüllt werden. Die Backend-Funktionalität eines neuen Blocks kann hierbei entweder durch eine neue Klasse im Backend oder durch das Einbinden eines externen Backend-Endpunktes realisiert werden [13]. Die Frontend Funktionalität kann entweder durch Svelte-Komponenten realisiert werden oder durch die Einbindung einer beliebigen Technologie mittels eines Iframes. So können leicht neue native Blocktypen erstellt werden, aber es kann auch bestehende Software integriert werden.

Zudem können für verschiedene Inhaltsformen angepasste Editoren implementiert werden, so ist es möglich auch komplexe Inhalte einfach zu bearbeiten. Eine komplexe Turing-Maschine könnte beispielsweise durch einen visuellen Editor einfacher bearbeitbar sein als durch eine textuelle Eingabe.

Wichtig ist, dass nur ein Prototyp entwickelt werden sollte, um eine grundlegende mögliche Umsetzung zu erforschen. Die entstandene Software hat nicht den Anspruch, sowohl aus Sicht der Benutzerfreundlichkeit als auch aus technischer Sicht eine fertige Software darzustellen.

[1] Die Dokumentation des Backends ist Teil des Teaching-Book-Repositorys

1.3 Verwandte Projekte

In diesem Projekt soll ein zellbasierter Editor entstehen. Beispiele für bekannte zell- bzw. blockbasierte Editoren sind Notion[2] und JupyterLab[3]. Diese beiden Editoren sind die Hauptinspiration für dieses Projekt. Diese Projekte haben grundlegende Ähnlichkeit mit unserem Prototypen, haben aber einen anderen Fokus und andere Ziele.

Notion[2] ist ein Web-Editor für Dokumente, die aus Text, Management-Tools und anderen integrierbaren Inhalten bestehen. Notion wird laut ihrer Webseite in mehreren großen Firmen benutzt. Ein Workspace in Notion besteht aus einer beliebigen Anzahl von Seiten, die beliebig geschachtelt werden können. Ein Seite in Notion besteht aus Blöcken, die verschiedene Inhalte darstellen. Der Fokus von Notion liegt nicht auf dem Erstellen von Lehrmaterialien. Notion löst aber auch das Problem, verschiedene Formen von Inhalten in einem Dokument unterzubringen. Notion ist in Rezensionen als besonders benutzerfreundlich bewertet wurden [14]. Ähnlich wie Notion besteht eine Instanz unserer Software aus geschachtelten Seiten, welche wiederum aus Blöcken bestehen, die verschiedene Formen von Inhalten darstellen. Wegen der Benutzerfreundlichkeit und dem minimalistischen Aussehen ist die Benutzeroberfläche sehr stark von Notion inspiriert. Der Grundaufbau der Seite ist gleich und allgemein sehen die Anordnung und das Design Notion sehr ähnlich.

EditorJS[4] ist ein Block-Style-Editor, der als JavaScript-Library verfügbar ist. Dieser hat auch ein sehr Notion-ähnliches Design und ein Plugin-System. Für diese Web-App war EditorJS aber nicht geeignet, da viele Features trotzdem anders sind und nur die Benutzeroberfläche an sich interessant ist.

In den so genannten Notebooks des Jupyter-Projekts[3] sollen interaktives Programmieren und Dokumentation zusammengebracht werden. Diese Notebooks sind ein zellbasiertes Dateiformat. Ein Notebook besteht aus Code-Zellen und Markdown-Zellen. Markdown-Zellen dienen zum Schreiben von Texten, die in HTML konvertiert werden. Code-Zellen dienen zum Schreiben von Programmabschnitten, welche ausgeführt werden können. Die Programmausgabe wird dann passend zur Zelle angezeigt. Auch eine Ausgabe von HTML oder Diagrammen ist hier möglich. Da ein Notebook einen Variablenraum hat, können sich Zellen auch Daten teilen. Jupyter hat auch ein Plugin-System, welches weitere Features ermöglicht. Um diese Notebooks zu bearbeiten bietet das Jupyter-Projekt den Editor JupyterLab an. Der Aufbau der Seiten unserer Software erinnern eher an Jupyter als an Notion, da Zellen bzw. Blöcke nur untereinander platziert werden können. Zudem hat eine Markdown-Zelle in Jupyter Notebooks einen Modus, in dem das gerenderte Markdown angezeigt wird und einen, in dem ein Editor zum Editieren angezeigt wird. In unserem Block-System haben alle Blöcke diese zwei Arten von Modi. Genauso wie im Backend dieses Projekt wird auch in Jupyter das Programm Pandoc benutzt, um Markdown in HTML zu konvertieren. Eine Anforderung an das Projekt war die Bedienbarkeit mit der Tastatur. Die Tastenkombinationen sind hierbei an Jupyter angelehnt. Dadurch sollen sich Jupyter-Nutzer schnell eingewöhnen können. Eine Anforderung an den Prototypen ist, dass sich in einer späteren Version

[2] Notion: https://www.notion.so/de-de
[3] Jupyter-Projekt: https://jupyter.org/
[4] EditorJS: https://editorjs.io/

Blöcke, ähnlich wie in Jupyter, Daten teilen können.

Es gibt auch viele andere auf Webtechnologien basierende Projekte, die versuchen klassische und interaktive Lehrmaterialien zusammenzuführen.

Schon 2003 gab es beispielsweise das Projekt "SmartFrame" mit dem Ziel, interaktive Webseiten anstelle von PDFs anzubieten, um die Lernerfahrung zu verbessern. SmartFrame benutzt für das Erstellen von Lehrmaterialien ein XML-ähnliches Format und erlaubt das Einbinden von interaktiven Inhalten. SmartFrame hat sogar schon das Ziel, Inhalte adaptiv auf den Nutzer anzupassen.[15]

Ein neueres Projekt basierend auf moderneren Technologien ist das PseuCo-Book. Das PseuCo-Book ist dabei ein Beispiel für die Erstellung eines Lehrbuchs, welches auf einem extra für diese Zwecke entwickelten Dateiformat basiert. Das bedeutet um interaktive Lehrbücher zu schreiben, können in einem XML ähnlichen Format Texte und interaktive Inhalte gemischt werden. Das Format basiert auf einem Framework geschrieben in React[5] und kann durch eigene Module erweitert werden. Das PseuCo-Book bietet zudem auch einen PDF-Export an.[16]

Anders als unser Projekt basiert das Projekt auf einem Dateiformat. Unser Projekt soll es aber ermöglichen angepasste Editoren für verschiedenen Inhalte anzubieten. Außerdem besitzt die resultierende Webseite kein Backend mit einer Datenbank, also müsste die Speicherung von benutzerspezifischen oder zur Bewertung notwendigen Daten extra entwickelt werden. Das Plugin-System hat zudem nicht die gleiche Flexibilität. In unserem Plugin-System sollen Erweiterungen nicht nur durch Framework-spezifischen Code implementiert werden können. Da das Projekt in dem Framework React geschrieben ist, müssen Erweiterungen auch in React geschrieben werden, oder es müsste eine weitere Schnittstelle implementiert werden.

Auch ein neueres Projekt ist das "First-Order Logic Workbook". Dieses Projekt benutzt einen Jupyter ähnlichen zellbasierten Editor, um interaktive Inhalte und Text zusammenzuführen. Ebenfalls beschäftigt sich dieses Projekt mit der Aufgabe bereits bestehende Technologie zu integrieren.[17] Dieses Projekt basiert aber nur auf Browserseitigen interaktiven Inhalten [17]. Die Berechnung und Speicherung einer Note sollte aber beispielsweise nicht auf einem Server stattfinden. Ein Student könnte sonst seine Daten einfach ändern. Außerdem können so keine Daten zu Lerndiagnostik gesammelt werden. Es ist auch nicht klar, ob es möglich wäre, die Software weiterzuentwickeln und adaptieren, da auch kein Zugriff auf den Programmcode besteht.

In der Universität Kassel nutzen viele Kurse die Lernplattform Moodle[6]. Moodle wird dafür benutzt Vorlesungen zu organisieren. An einem Ort werden hier Informationen zentral angeboten. Trotzdem werden Arbeitsblätter und Vorlesungsskripte meistens als PDF hinterlegt. Unser Projekt hat nicht den Zweck, Moodle selbst zu ersetzen. Unser Projekt bietet ein neue Möglichkeit Vorlesungsskripte und Arbeitsblätter anzubieten. Auf Plattformen wie Moodle könnte dann auf diese Inhalte verwiesen werden.

In unserem Projekt wollen wir den Ansatz eines zellbasierten Editors wie in Jupyter verknüpfen mit einer minimalistischen und benutzerfreundlichen Benutzeroberfläche

[5] React: https://reactjs.org/
[6] Moodle: https://moodle.org/

wie in Notion. Zusätzlich wollen wir ein Plugin-System schaffen, welches sowohl erlaubt, nativen Code zu schreiben, als auch vorhandene Software zu integrieren. Außerdem wollen wir ein Backend statt ein Dateiformat benutzen, damit auch benutzerspezifische Inhalte gespeichert werden können. Die Kombination dieser Konzepte an sich stellen schon eine Abgrenzung dar. Auch wenn es Projekte gibt die durchaus Features bieten, die für unser Projekt interessant sind, ist der Umbau dieser nicht zielführend. Die Features, Ziele und Anforderungen stimmen nicht genug überein, sodass ein Umbau nicht lohnenswert erscheint. Der Gesamtansatz und die Kombination von Konzepten ist bei diesem Prototyp anders. Außerdem soll der Website-Baukasten nicht das Format einer Plattform wie Moodle übernehmen, sondern gezielt klassische Formate wie Vorlesungsskripte und Arbeitsblätter durch moderne Technologien erweitern.

Dass es ältere Projekte wie SmartFrame gibt, zeigt auf, wie lange schon an der Idee, Webtechnologien in der Lehre einzusetzen, geforscht wird. Die neuen Projekte zeigen, wie aktuell das Problem dennoch ist. Außerdem sind die neuen Projekte ein Indiz dafür, dass es noch keine Lösung gibt, die weitverbreitet und zufriedenstellend ist.

1.4 Aufbau der Arbeit

Hauptsächlich beschäftigt sich diese Arbeit mit dem entstandenen Prototypen, welcher grundlegend auf den Anforderungen des Fachgebietes und der Kombination verschiedener Konzepte basiert. Zuerst werden dabei die Features und die Benutzeroberfläche aus der Sicht eines potentiellen Nutzers betrachtet. Nachfolgend wird die technische Umsetzung erläutert. Anschließend wird das Plugin-System beschrieben. Um den Prototyp und den Stand der Entwicklung besser einzuordnen wurden Nutzertests und eine Befragung durchgeführt. Die Ergebnisse dieser werden kurz vorgestellt. Zum Schluss wird darauf eingegangen, wie die zukünftige Weiterentwicklung aussehen könnte und wie weitere geplante Features umgesetzt werden könnten.

Der Arbeitstitel des Projekts heißt Teaching-Book. Im weiteren Verlauf der Arbeit wird deswegen auch das Gesamtprojekt als Teaching-Book bezeichnet und dabei das Frontend als Teaching-Book-Frontend und das Backend als Teaching-Book-Backend. Der Programmcode für den Prototypen ist auf einem Gitlab-Repository [7] des Fachgebietes zu finden. Auf diesem Repository befindet sich auch die Backend-Dokumentation.

[7] Teaching-Book-Gitlab-Repository: https://syre.fm.cs.uni-kassel.de/tools-for-students/teaching-book.

2 Das Konzept aus der Sicht der Nutzer

In diesem Kapitel wird das Konzept aus der Sicht eines potentiellen Nutzers besprochen. Es wird erklärt, wie der Website-Baukasten angewendet werden soll, um Arbeitsblätter und Vorlesungsskripte umzusetzen. Dabei wird der Aufbau der Benutzeroberfläche erläutert und die jeweiligen Features, die diese bereitstellt.

Um eine neue Vorlesung vorzubereiten, ist es vorgesehen, eine neue Instanz des Teaching-Book-Projektes aufzusetzen. Eine Instanz des Teaching-Book-Projektes ist für eine ganze Vorlesung gedacht. Das Vorlesungsskript, Arbeitsblätter und andere Materialien einer Vorlesung sollen auf der gleichen Instanz erstellt werden. Im weiteren Verlauf sprechen wir je nach Kontext vom Teaching-Book, wenn die Instanz gemeint ist, oder das Projekt.

Das Teaching-Book ist ein zell-/blockbasierter Editor, ähnlich wie JupyterLab, Jupyter Notebooks oder Notion. Ein Teaching-Book besteht aus mehreren Seiten, die beliebig geschachtelt werden können. Diese Seiten bestehen wiederum aus Blöcken, die die verschiedenen Formen von Inhalten darstellen. Die Grundstruktur der Seite und generell das Design sind sehr Notion-ähnlich. Besonders Notion-ähnlich sind dabei die Bedienelemente der Blöcke.

Es gibt grundlegend zwei Pfade bzw. URLs, mit denen auf eine Seite im Teaching-Book zugegriffen werden kann. Der Pfad `view/[pageId]` öffnet die Seite in der Ansicht für normale Nutzer bzw. Studenten. Der Pfad `edit/[pageId]` öffnet die Seite in der Ansicht für Dozenten bzw. in der Bearbeitungsansicht. `[pageId]` steht hier stellvertretend für eine Zahl, die zur Identifikation der Seite dient. In Abb. 2.2 und 2.4 ist die gleiche Seite auf beiden Pfaden zu sehen. In der Bearbeitungsansicht gibt es Möglichkeiten um Seiten zu verwalten und zu bearbeiten, ansonsten sehen die Ansichten relativ ähnlich aus. Das sorgt dafür, dass der Dozent direkt einen Eindruck davon bekommt, wie das Skript auf den Studenten wirkt, ohne die Seite selbst noch ein zweites Mal in der normalen Ansicht zu öffnen.

Beide Seiten sind mit einem einfachen Login-Bereich geschützt, welcher in Abb. 2.6 zu sehen ist. Außerdem ist das Einloggen wichtig, da in diesem Projekt Blöcke nicht nur allgemeine Daten, sondern auch benutzerspezifische Daten speichern können.

Darüber hinaus wurde die normale Ansicht auch für mobile Geräte entwickelt. Die mobile Version dieser Ansicht ist in Abb. 2.7 und 2.8 zu sehen.

2.1 Erstellen, Löschen und Verschieben von Seiten und Unterseiten

Links in der Benutzernutzeroberfläche befindet sich die Sidebar, die auf- und zugeklappt werden kann und in Abb. 2.1 rot umrandet ist. Die Sidebar kann verwendet werden, um zwischen den verschiedenen Seiten zu wechseln und eine Übersicht über alle Seiten zu gewährleisten. Sie ist das Inhaltsverzeichnis des Teaching-Books. In der Bearbeitungsansicht können die Seiten verwaltet werden. Es können hier neue Seiten und Unterseiten erstellt, verschoben und gelöscht werden. Neue Seiten können über den "add page"-Button hinzugefügt werden. Ist der Mauszeiger über ein Seitenelement in der Sidebar, erscheinen in diesem Element ein Plus und ein Mülleimersymbol rechts an der Seite. Über das Plus können Unterseiten zur jeweiligen Seite erstellt werden. Über das Mülleimersymbol kann eine Seite gelöscht werden. Per Drag-and-Drop können Seiten verschoben und geschachtelt werden.

2.2 Verwaltung einer Seite

Erstellt man eine neue Seite, wird diese automatisch geöffnet. In Abb. 2.1 ist in Orange die Optionsleiste markiert. Hier kann die Seite ebenfalls gelöscht werden oder die Sichtbarkeit einer Seite eingestellt werden. Eine Seite ist nach dem Erstellen standardmäßig nicht öffentlich. Das bedeutet, dass diese nur in der Bearbeitungsansicht angesehen werden kann. Eine Seite ist standardmäßig gelistet. Eine ungelistete Seite wird im Bearbeitungsmodus genauso angezeigt wie eine normale Seite. In der normalen Ansicht hingegen gibt es für eine ungelistete Seite keine Sidebar und auf einer anderen Seite taucht die ungelistete Seite auch nicht in der Sidebar auf. Dieses Feature soll es auch ermöglichen, Seiten zu erstellen, die nicht Teil der eigentlichen Vorlesungsskripte sind, wie zum Beispiel Übungen oder Probeklausuren.

Jede Seite hat einen Titel, der in Abb. 2.1 grün markiert ist, dieser wird auch zur Anzeige in der Sidebar verwendet. Dieser Titel kann in der Bearbeitungsansicht einfach wie ein normales Eingabefeld bearbeitet werden, wird das Eingabefeld verlassen, wird der Titel automatisch gespeichert.

2.3 Löschen, Selektieren und Bearbeiten von Blöcken

Eine Seite besteht aus Blöcken. Diese Blöcke stellen verschiedene Formen von Inhalten dar. Ist ein Block selektiert, so wird dieser blau umrandet, wie in Abb. 2.1 und Abb 2.2 zu sehen ist. Ein neuer Text- bzw. PandocHtml-Block kann über den "add Block"-Knopf unten auf der Seite hinzugefügt werden.

Ist der Mauszeiger nicht auf der Sidebar und auf der gleichen Höhe eines Blocks, wird zu dem Block eine Reihe von Optionen angezeigt, die wir Kontrollleiste nennen. Diese besteht aus vier Symbolen. Die Kontrolleiste ist in der Abb. 2.1 gelb markiert. Über das Plus-Symbol kann mit einem Linksklick ein Menü geöffnet werden, welches alle Blockarten anzeigt. Wird eine dieser Blockarten ausgewählt, wird ein neuer Block

dieser Art unter dem jeweiligen, zu dem das Menü gehört, erstellt. Mit Rechtsklick auf das Plus-Symbol geschieht das Gleiche, nur dass der Text bzw. PandocHtml-Block direkt erzeugt wird, ohne dass ein Menü geöffnet wird. Über das Mülleimer-Symbol kann ein Block gelöscht werden. Über das Symbol ganz rechts, welches aus sechs Punkten besteht, kann der Block per Drag and Drop verschoben werden. Über das Symbol, das aussieht wie ein Stift und ein Notizblock, kann der Bearbeitungsmodus eines Blocks geöffnet werden.

Ein Block hat, genauso wie die Seite selbst, zwei Modi. Den Bearbeitungsmodus und den normalen Modus. Standardmäßig ist auch in der Bearbeitungsansicht jeder Block im normalen Modus. Der normale Modus soll das Ergebnis zeigen, welches auch normale Studenten sehen. Der Bearbeitungsmodus soll es ermöglichen, die Inhalte je nach Blockart zu verändern. Der Bearbeitungsmodus ist nur in der Bearbeitungsansicht verfügbar. Ist ein Block, wie in Abb. 2.3 zu sehen, im Bearbeitungsmodus, so wird er mit einer gestrichelten Linie umrandet dargestellt. Im Bearbeitungsmodus besteht die Kontrollleiste nur noch aus einem Diskettensymbol, welches dazu dient, den bearbeiteten Block zu speichern und den Bearbeitungsmodus wieder zu verlassen. Blöcke werden also nicht automatisch gespeichert. Damit noch besser erkennbar ist, dass ein Block im Bearbeitungsmodus ist, wird die Kontrollleiste bei diesen nicht ausgeblendet. Bei manchen Blöcken, wie zum Beispiel dem PandocHTML-Block, kann der Bearbeitungsmodus auch durch einen Doppelklick auf den jeweiligen Block aktiviert werden.

2.4 Blocktypen

In dieser Version des Prototypen gibt es drei Arten von Blöcken:

1. **PandocHtml-Block**: Der PandocHtml-Block ist der Block, um Texte zu verfassen. Texte können im Bearbeitungsmodus in der Auszeichnungssprache Markdown geschrieben werden, wobei auch mathematische Ausdrücke, wie aus Latex bekannt, erlaubt sind. Der verwendete Markdown Standard nennt sich "Pandoc's Markdown" [1]. Wenn eine Markdown-Eingabe gespeichert wird, dann wird diese durch das Programm Pandoc[2] in HTML konvertiert [13]. Im normalen Modus wird die Eingabe also als formatierter Text angezeigt. In Abb. 2.3 ist ein PandocHtml-Block im Bearbeitungsmodus zu sehen und in Abb. 2.2 im normalen Modus.

2. **WoFa-Block**: Der WoFA-Block erlaubt es, automatisch korrigierte Aufgaben über endliche Automaten zu stellen. Im Bearbeitungsmodus können eine Lösung und eine Vorlage für einen endlichen Automat in textueller Form angegeben werden. In der normalen Ansicht hat der Nutzer dann die Vorlage zur Verfügung, um eine Lösung zu formulieren, die er dann speichern und automatisch bewerten lassen kann. Das Plugin wurde implementiert, um als Beispiel für eine automatisch korrigierte Aufgabe zu stehen und damit zu zeigen, wie ein typischer

[1] Pandoc's Markdown: https://pandoc.org/MANUAL.html
[2] Pandoc: https://pandoc.org/

Blocktyp aufgebaut sein könnte. Das Plugin soll zudem die technischen Aspekte einer nativen Implementierung aufzuzeigen.

3. **Calculator-Block**: Der Calculator-Block ist ein einfacher Rechner und kann genutzt werden, um ein Beispiel für eine arithmetische Rechnung zu liefern. Im normalen Modus können zwei Zahlen und eine arithmetische Operation angegeben werden. Die daraus resultierende Berechnung kann der Calculator jetzt ausführen. Im Bearbeitungsmodus sieht der Block genau gleich aus, die Eingabe kann aber gespeichert werden. Wird der Block dann im normalen Modus geöffnet, wird die gespeicherte Eingabe als Initialwert der Eingabefelder genutzt. Dieser Rechner hat selbst keinen großen Nutzen. Das Plugin dient dazu, den vermutlich typischen Aufbau eines Blocktyps aufzuzeigen, der ein interaktives Beispiel darstellt. Außerdem zeigt er die technischen Aspekte einer Implementierung durch einen Iframe eingebundenen Web-App.

Umrandet in der Farbe Lila ist in Abb. 2.1 der Header zu sehen. Links im Header lässt sich die Sidebar ein- und ausklappen. Links befindet sich ein Download-Symbol, mit diesem lassen sich die Download Optionen für Latex und PDF anzeigen. Die Download-Optionen selbst sind noch nicht implementiert. Daneben gibt es die Möglichkeit zum Ausloggen. Ganz links befindet sich nur in der Bearbeitungsansicht ein weiteres Drop-Down-Menü, welches auf eine Anleitung zu Pandoc's Markdown[1] verweist und über das sich eine Übersicht über alle Tastenkombinationen öffnen lässt, welche in Abb. 2.5 zu sehen ist.

2.5 Tastenkombinationen

Alle Aktionen in der Kontrollleiste sind auch durch Tastenkombinationen ausführbar. Die Tastenkombinationen sollen die Geschwindigkeit beim Arbeiten für erfahrene Nutzer erhöhen. Die Tastenkombinationen werden immer auf dem selektierten bzw. blau umrandeten Block ausgeführt. Zusätzlich gibt es verschiedene Tastenkombinationen, um einen anderen Block auszuwählen. Wird ein Block selektiert, wird auch der Fokus des Browsers um den Block gesetzt und wird ein Element in einem Block fokussiert, wird dieser Block selektiert. Es ist also wie im Webbrowser üblich möglich, durch Drücken der Tab-Taste einen Block auszuwählen oder die Kontrollleiste zu bedienen. Die Tastenkombinationen sind ähnlich wie die von JupiterLab. Somit sollen sich Jupyter-Nutzer schneller an die Software gewöhnen.

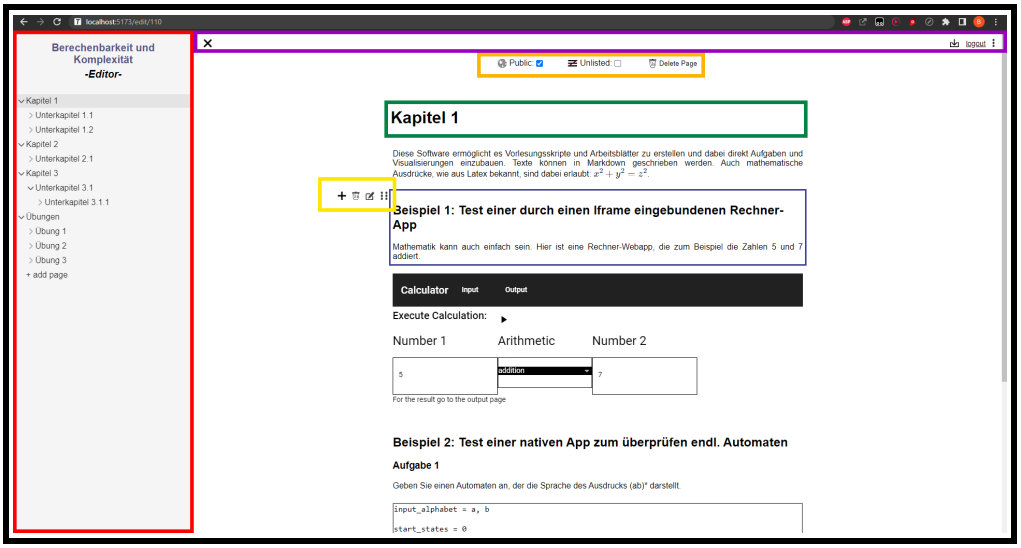


Abbildung 2.1: Bearbeitungsansicht - wichtige Bereiche Farbig markiert

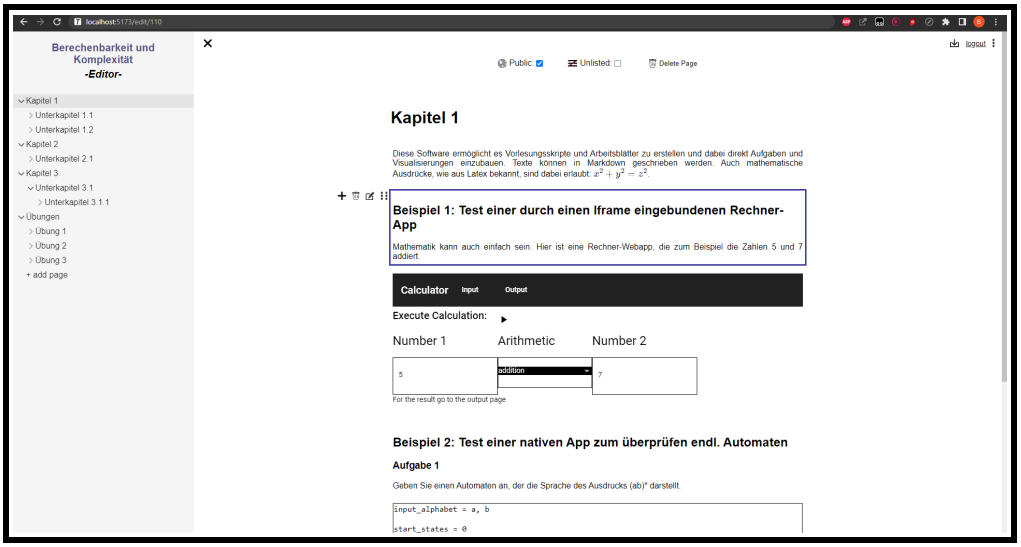


Abbildung 2.2: Bearbeitungsansicht

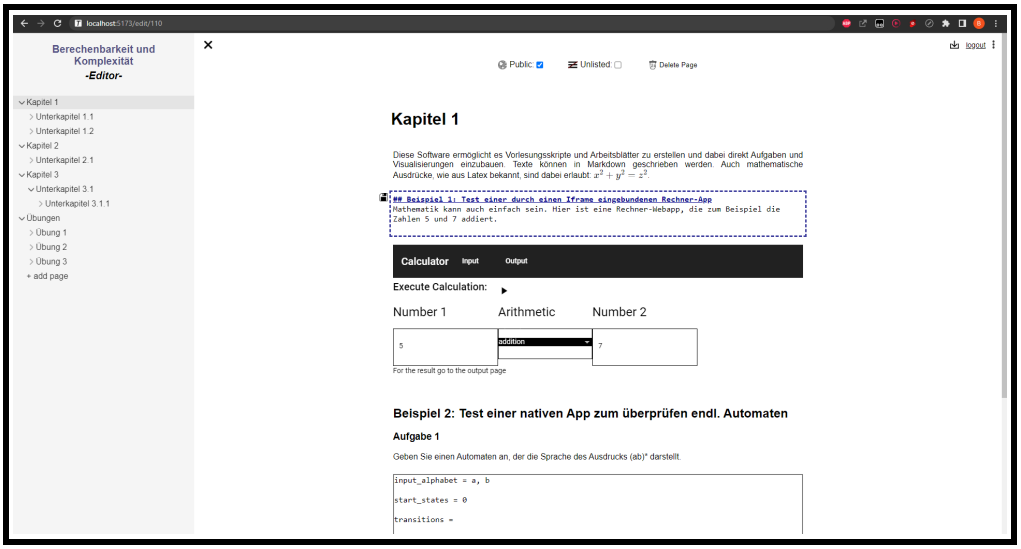


Abbildung 2.3: Markdown-Block im Bearbeitungsmodus

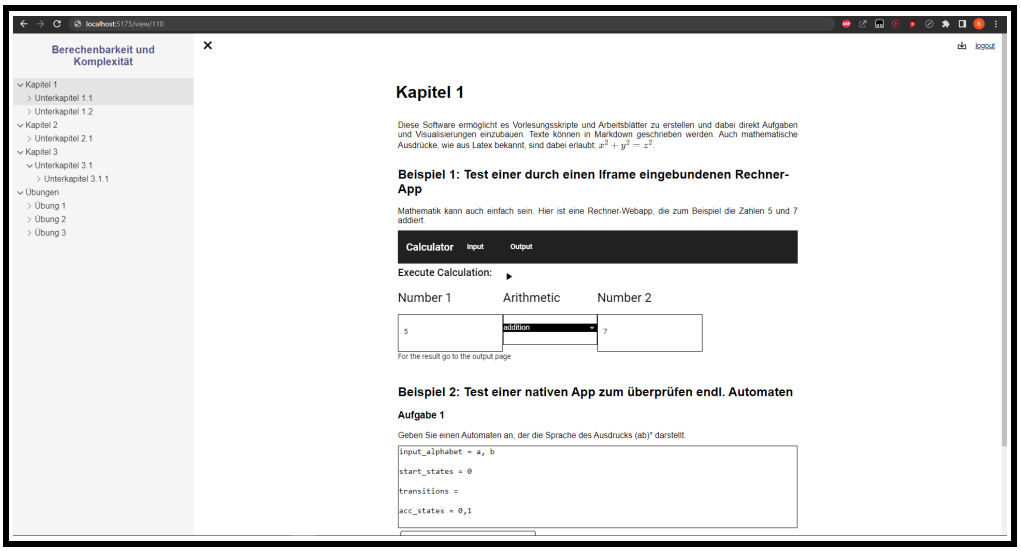


Abbildung 2.4: normale Anischt

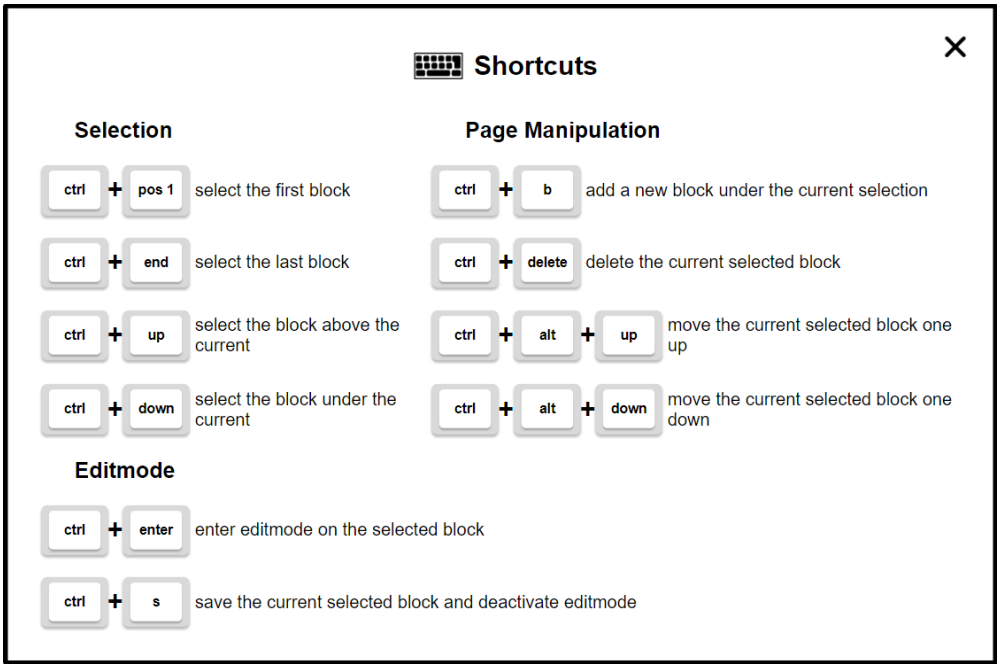


Abbildung 2.5: Shortcuts

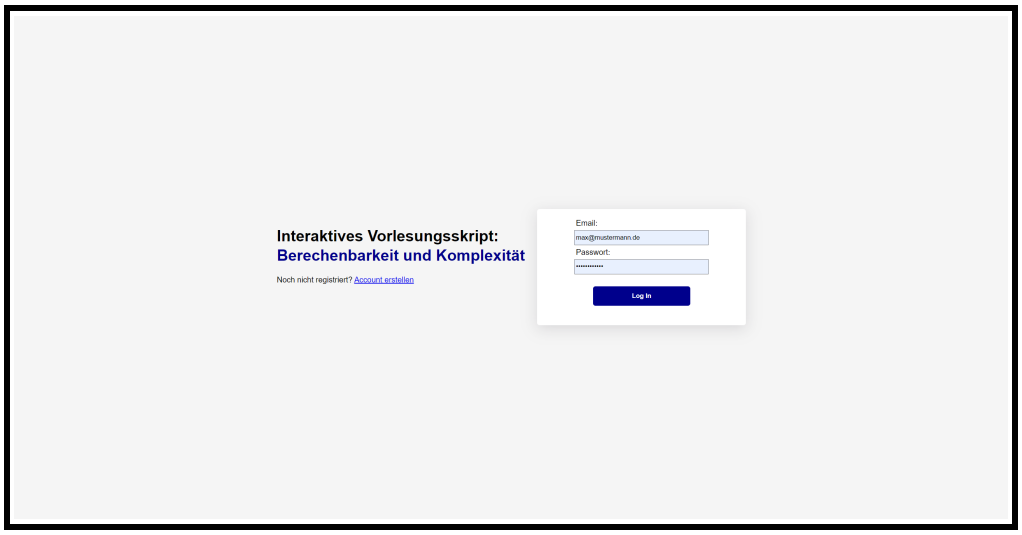


Abbildung 2.6: Login-Bereich

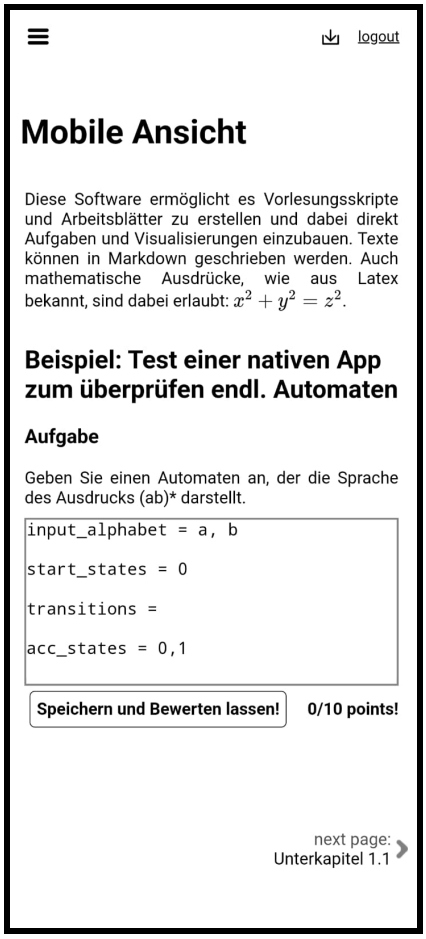


Abbildung 2.7: normale Ansicht (mobile)

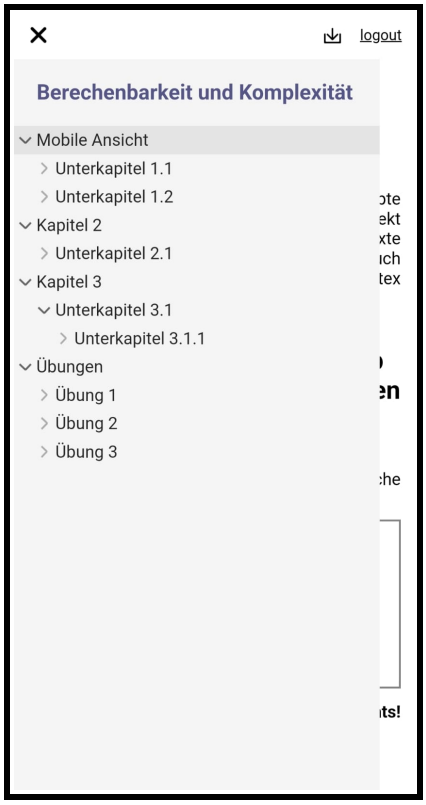


Abbildung 2.8: normale Ansicht - Sidebar (mobile)

3 Technische Umsetzung

In diesem Kapitel soll die technische Umsetzung des Projekts besprochen werden. Die Architektur ist eindeutig in Front- und Backend aufgeteilt. Zuerst werden jeweils die verwendeten Technologien diskutiert und danach wird der Aufbau erläutert.

3.1 Backend-Technologien

Das Backend baut hierbei auf Python[1] und Django[2] auf und ist für die Authentifizierung und Datenverwaltung verantwortlich [13].

Django

Django[2] ist ein Webframework für Python[1] Django bietet Lösungen zum Datenbankenmanagement, Authentifizierung und Berechtigungsverwaltung. Außerdem bietet Django ein Admin-Interface um das Backend zu verwalten. Somit bietet Django grundlegende Funktionalitäten, die auch für das Teaching-Book-Backend gebraucht werden [13].

Pandoc

Eines der Default-Plugins ist PandocHtml, dieses ermöglicht es Texte in Markdown zu verfassen. Diese sollen dann zu HTML gerendert werden und in einer späteren Version auch als LaTeX- und PDF-Export zur Verfügung stehen. Diese Funktionalität wird auch auf Backend-Seite umgesetzt [13]. Das Umwandeln in die Formate wird durch Pandoc[3] realisiert [13].

Pandoc[3] ermöglicht es, von einem Format zu verschiedenen Anderen zu konvertieren, darunter Markdown, Latex, PDF und HTML. Des Weiteren ist der Markdown-Standard von Pandoc sehr umfangreich. Im Pandocs-Markdown kann direkt der Mathemodus wie aus Latex bekannt benutzt werden.

Das Rendern der mathematischen Ausdrücke wird allerdings nicht durch Pandoc, sondern durch Katex[4] um Frontend umgesetzt.

Pandoc bietet also die umfangreichen Möglichkeiten, die wir benötigen, um Markdown in HTML umzuwandeln, aber auch um später weitere Formate sowohl als Ausgabe als auch als Eingabe zu implementieren.

[1] Python: https://www.python.org/
[2] Django: https://www.djangoproject.com/
[3] Pandoc: https://pandoc.org/index.html
[4] Katex: https://katex.org/

3.2 Frontend-Technologien

Das Frontend ist in dem Frontend-Framework Svelte[5] in Typescript[6] umgesetzt worden und stellt die Benutzeroberfläche bereit. Für Routing wurde die Library Svelte-Navigator[7] verwendet. Das Rendern der mathematischen Ausdrücke erfolgt mithilfe der Library Katex[4]. Um das Editieren von Text bzw. Code zu ermöglichen, wurde die Library CodeMirror verwendet.

Das Rendern der mathematischen Ausdrücke erfolgt mithilfe der Library Katex4, während der Code- und Texteditor durch die CodeMirror-Library bereitgestellt wird.

Svelte

Svelte[5] ist ein modernes und performantes, aber noch relativ neues Frontend-Framework, welches es ermöglicht, die Benutzeroberfläche in verschiedene Komponenten zu unterteilen. Diese Komponenten werden jeweils durch eine Datei beschrieben, in der CSS, HTML und JavaScript oder Typescript benutzt werden können, um das Verhalten der Komponente gekapselt zu beschreiben. Natürlich lassen sich Komponenten auch schachteln und es können auch CSS und JavaScript Dateien importieren. Svelte bietet zusätzlich eine syntaktische Erweiterung, um einfach typische Frontend-Funktionalität umzusetzen, wie zum Beispiel das Einbinden von Variablen oder Funktionen in die HTML-Struktur.

Svelte ermöglicht es, reaktiven und deklarativen Code zu schreiben. Anders als andere Frameworks realisiert Svelte diese Funktionalität nicht zur Laufzeit, sondern lagert diesen Schritt auf einem Compiler aus. Die Syntax, die Svelte dafür verwendet, ist zudem darauf ausgelegt, möglichst benutzerfreundlich zu sein.[18] Svelte ist 2021 und 2022 zu einem der beliebtesten Web-Frameworks unter Entwicklern gewählt worden [19] [20].

Diese Eigenschaften machen zu einer guten Wahl für eine neue Web-App, die die Anforderungen hat, leicht erweiterbar zu sein und auch von unerfahrenen Programmierern weiterentwickelt werden kann. Gerade dadurch, dass native, also Framework-spezifische Plugins entwickelt werden sollen, ist es wichtig, dass das Framework leicht zu lernen ist bzw. dass ohne viel Framework spezifisches Wissen Funktionalität hinzugefügt werden kann.

Client-Side-Routing mit Svelte-Navigator

Als Routing Library wurde Svelte-Navigator[7] verwendet. In Svelte-Navigator werden Routen durch Svelte-Komponenten und deren Attribute direkt im Quellcode festgelegt. Es sind nur zwei Pfade für die App notwendig, deswegen ist diese Struktur hier sehr übersichtlich.

Eine Alternative stellt Svelte-Kit[8] dar. In Svelte-Kit funktioniert Routing über die Ordnerstruktur. Über die Verschachtelung von Ordnern und Dateien mit bestimmten

[5] Svelte: https://svelte.dev/
[6] Typescript: https://www.typescriptlang.org/
[7] Svelte-Navigator: https://github.com/mefechoel/svelte-navigator
[8] Svelte-Kit: https://kit.svelte.dev/

Namen können verschiedene Routen dargestellt werden. Jeder Pfad hat eine Load Funktion, in der Daten geladen werden können, die nach der Navigation zur Verfügung stehen. Svelte-Kit ist mehr als eine Routing-Library und bietet noch weitere Features. Ein Nachteil von Svelte-Kit ist, dass es sehr neu ist (Veröffentlichung von Version 1.0 am 6.12.2022 [21])

Dem ersten Eindruck nach, sind komplexe Routing-Features wie Server-Side-Rendering relativ leicht in Svelte-Kit zu implementieren. Server-Side-Rendering bedeutet, dass der Inhalt der Seite beim ersten Aufruf auf dem Server vorbereitet und als fertige Seite an den Client gesendet wird [22]. Das hat Vorteile bei der wahrgenommenen Performance und für das Ranking in Suchmaschinen [22].

Hier wurde sich für Svelte-Navigator anstelle von Svelte-Kit entschieden. Diese Entscheidung wurde unter anderem deswegen getroffen, weil die Struktur von Svelte-Kit das Projekt im ersten Moment komplizierter erscheinen ließ. Mit Svelte-Navigator die Routen im Quelltext festzulegen, erschien für die zwei Pfade intuitiver. Weitere Features sind momentan nicht notwendig. Wenn sich in einer späteren Version des Projektes die Anforderungen an eine Routing-Library ändern, sollte in Betracht gezogen werden, zu Svelte-Kit zu wechseln.

Typescript

Typescript[6] ist eine Erweiterung von JavaScript, die es ermöglicht, Typen zu definieren und für Variablen festzulegen. Ein Typescript Programm kann dadurch auf Typenfehler überprüft werden.

CodeMirror

CodeMirror[9] ist eine Texteditor-Library, welches ein umfangreiches Plugin-System bietet, um die Syntax verschiedenster Programmiersprachen hervorzuheben, aber auch um Features, die die Benutzerfreundlichkeit verbessern, einzubauen, wie zum Beispiel Autovervollständigung oder die Möglichkeit, Änderungen rückgängig zu machen.

Da fast alle Features per Plugin hinzugefügt werden eignet sich CodeMirror sehr gut für dieses Projekt. So bleibt die gesamte Web-App klein und der Benutzer wird nicht von unnötigen Features von den eigentlichen Inhalten abgelenkt. Außerdem bietet CodeMirror Support für mobile Geräte. Die Alternative Monaco[10] von Microsoft bietet diesen Support beispielsweise nicht.

Katex

Katex[4] ist eine Bibliothek, die es ermöglicht, einen mathematischen Ausdruck in Latex in HTML-Code umzuwandeln. Zusätzlich gibt Katex auch MathML aus, um die Barrierefreiheit zu erhöhen. Katex wirbt damit, die schnellste Library zu sein.

[9] CodeMirror: https://codemirror.net/
[10] Monaco: https://microsoft.github.io/monaco-editor/

3.3 Softwarearchitektur - Backend

Das Backend wurde in einem separaten Projekt entwickelt und ist nicht Teil dieser Arbeit. Wir besprechen in diesem Kapitel zusammenfassend die Rolle des Backends im Gesamtprojekts bzw. den groben Aufbau und das Datenmodell, da diese wichtige Bestandteile sind, um das Gesamtkonzept zu verstehen. Wie jeweilige Anfragen zum Server auszusehen haben, wird hier nicht erklärt. Weitere Details können in der Backend-Dokumentation oder im Repository des Projekts nachgelesen werden. Das Backend ist für die Verwaltung des Datenmodells verantwortlich. Außerdem verwaltet es die Benutzer und implementiert die Rechteverwaltung. Das Backend liefert eine Rest-API, um die Features nach außen zu tragen. Das Backend muss außerdem Funktionalität bereitstellen, um die verschiedenen Plugins zu verwalten, die im Weiteren auch Tools oder Blocktypen genannt werden.[13]

3.3.1 Datenmodell

In Abb. 3.1 ist das Datenmodell als ERM-Diagramm ohne Attribute skizziert.

Auch im Datenmodell besteht ein Teaching-Book aus mehreren Seiten und eine Seite aus mehreren Blöcken. Ein Block wird aus einem Tool abgeleitet. Ein Tool beschreibt die Struktur und Funktionalität eines Blockes und stellt eine Vorlage dafür bereit. Ein Block besitzt abhängig von seinem Tool verschiedene Datenbereiche mit verschiedenen Datenfeldern. An einem Tool können außerdem Aktionen registriert werden, die dann auf einem Block ausgeführt werden können. Diese Aktionen sind Funktionalitäten, die entweder nativ durch das Backend bereitgestellt werden, oder durch Endpunkte anderer externen Backends bereitgestellt werden können. Ein Tool stellt also ein Plugin bzw. einen Blocktypen dar.[13]

3.3.2 Datenbereiche in Blöcken bzw. Tools

Jeder Block eines Tools hat die Datenbereiche `tool_data`, `public_data`, `secret_data` und `user_data`. Diese bestehen je nach Tool aus Datenfeldern verschiedener Typen.[13]

Der Datenbereich `tool_data` wird für ein Tool angelegt, ist also für jeden Block eines Tools gleich. Die Datenbereiche `public_data` und `tool_data` werden für jeden Block jeweils angelegt, sind aber für jeden Benutzer gleich. Der Datenbereich `user_data` wird jeweils für jeden Block und jeden Benutzer angelegt. Dieser Datenbereich ist also für jeden Block anders und für jeden Benutzer.[13]

Die Datenbereiche `tool_data` und `secret_data` sind nur von registrierten Aktionen von Administratoren bzw. Benutzern mit dem Status `Staff` veränderbar.[13]

Welche Felder ein Block eines Tools im jeweiligen Datenbereich anbietet, und welche Werte initial im Datenbereich festgelegt sind, wird über das Tool festgelegt, ähnlich wie eine Klasse die Datentypen für ein Objekt der Klasse festlegt. Auch die Typen eines Datenfeldes werden so festgelegt. Ist der initiale Wert eines Datenfeldes ein Integer, kann auch in Zukunft nur ein Integer in diesem Datenfeld stehen. Ein Datenfeld kann zudem den Typ JSON haben. So sind auch komplexere Datenfelder möglich.[13]

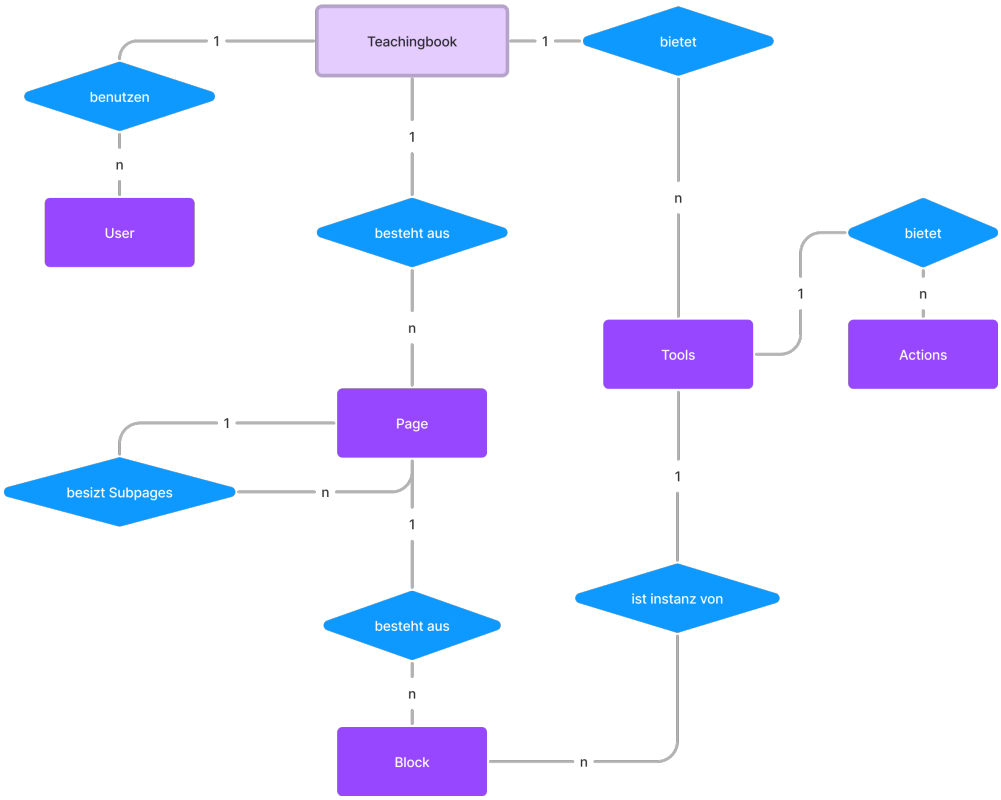


Abbildung 3.1: ERM-Diagramm

3.3.3 Aktionen

Eine Aktion ist eine Art Funktionalität, die auf einem Block ausgeführt wird. Dabei können Daten aus dem Block und Tool des Blocks entnommen und verändert werden. Diese Aktionen können direkt im Backend, aber auch durch das Registrieren externer Endpunkte realisiert werden. Diese Aktionen können dazu benutzt werden, zugriffsbeschränkte Daten zu bearbeiten, auch wenn ein User die Aktion ausführt, der diese Daten eigentlich nicht bearbeiten darf. Um in die Datenbereiche zu schreiben gibt es standardmäßig jeweils eine Aktion für jeden Block bzw. Blocktypen, die genauso heißt wie der Datenbereich. Für die zugriffsbeschränkten Datenbereiche sind diese Standardaktionen selbst zugriffsbeschränkt.[13] Mehr dazu kann im Kapitel 4.2.2 nachgelesen werden.

3.4 Softwarearchitektur - Frontend

Das Frontend ist für die Benutzeroberfläche verantwortlich. Die Benutzeroberfläche bietet, wie in dem Kapitel 2 besprochen, zwei Ansichten. Eine normale Ansicht für die normalen Nutzer bzw. Studenten und eine Bearbeitungsansicht für die Dozenten bzw. `Staff`-User. Hier wird nun passend zu dieser Benutzeroberfläche der technische Hintergrund besprochen. Dabei gehen wir folgende Aspekte durch: die Pfadstruktur, die Komponenten- und Modul-Struktur, sowie die Verwaltung der Daten und des Zustandes der Benutzeroberfläche. Für die Komponenten `BlockContainer` und `IframeContainer` wird hier zudem genauer die Funktionsweise beschrieben, da diese Komponenten wichtige Schlüsselrollen beim Laden und Verwalten von Blöcken übernehmen.

3.4.1 Ansichten und URL bzw. Pfad Struktur

Wie in Abbildung 3.2 und 3.3 zu sehen ist, ist die Web-App in zwei Pfade bzw. Ansichten unterteilt. Es gibt den Pfad `/view/[pageId]`. Unter dessen URL befindet sich die jeweilige Ansicht der jeweiligen Seite für normale Benutzer. `[pageId]` steht hier stellvertretend für eine Zahl, die für die Identifikation der jeweiligen Seite im Datenmodell im Backend steht. Genauso funktioniert auch der zweite Pfad `/edit/[pageId]`. Dieser führt aber zur Bearbeitungsansicht der jeweiligen Seite. Beide Pfade sind durch einen Login-Bereich geschützt, da für manche Blöcke auch benutzerspezifische Daten gespeichert werden. Die Web-App benutzt Client-Side-Rendering[11]. Die Url wird durch Svelte-Navigator auf der Client-Seite interpretiert. Jeder Aufruf des Frontend lädt also erstmal die gleichen Daten herunter, danach wird die URL durch Svelte-Navigator auf der Client-Seite interpretiert und die nötigen weiteren Dateien und Inhalte werden nachgeladen.

3.4.2 Komponenten- und Modulstruktur

Svelte ist ein komponentenbasiertes Framework. Die Web-App ist also in verschiedene Svelte-Komponenten unterteilt, die eine Baumstruktur bilden, die in einer Benutzeroberfläche resultiert. Logik, die nicht direkt zu einer Svelte-Komponente gehört, wird in sogenannte Module ausgelagert. Die Ordnerstruktur folgt dabei den Abhängigkeiten der Komponenten. In Abb. 3.2 und 3.3 ist ein vereinfachtes Abhängigkeitsdiagramm dieser Struktur zu sehen. Abb. 3.4 zeigt die globalen Module, die von beiden Pfaden benutzt werden. Zusätzlich gibt es die verschiedenen Plugins bzw. Tools, die, wenn sie nativ in Svelte umgesetzt wurden, in einem extra dafür vorgesehenen Ordner liegen.

Die meisten anderen Komponenten werden nicht zwischen den zwei Pfaden bzw. Ansichten geteilt. Die Sidebar-Komponente des Editors ist beispielsweise nicht die gleiche wie die der normalen Ansicht. Sie sehen zwar zum größten Teil gleich aus, aber die Komponente vom Editor besitzt deutlich mehr Funktionalität, als die der normalen Ansicht. Dadurch, dass zwischen diesen Ansichten nur wenige Komponenten geteilt werden, können Verzweigungen für die unterschiedlichen Berechtigungen vermieden

[11] Client-Side-Rendering: https://www.patterns.dev/posts/client-side-rendering/

werden. Für die Komponenten der normalen Ansicht ist meistens klar, dass diese nur auf Funktionen normaler Nutzer zugreifen sollten, und für die Komponenten in der Bearbeitungsansicht ist meistens klar, dass diese von Nutzern mit Bearbeitungsrechten aufgerufen werden.

3.4.3 Verwaltung von Daten und Zustand der Benutzeroberfläche

Daten, die speziell nur lokal in einer Komponente gebraucht werden, werden in der Regel in dieser auch lokal verwaltet. Übergeordnete Daten, die in übergeordneten Komponenten erzeugt werden und nur von dort geändert werden, die also der Baumstruktur entsprechen, werden über die Attribute der Komponenten am Baum heruntergereicht. Falls ein Zustand nicht in das typische Baum-Schema passt, also beispielsweise von einer untergeordneten Komponente geändert werden muss, wird diese hier meistens in Modulen ausgelagert und mit den in Svelte eingebauten Stores umschlossen. Ein Store ist ein Container für eine Variable, wird eine neue Variable in diesen Container gesetzt, bekommen alle Komponenten, die sich bei diesem Container registriert haben, die neue Variable mitgeteilt [23]. In den Abbildungen sind diese rot abgebildet. Diese Stores bieten dadurch eine Möglichkeit, aus der Baumstruktur auszubrechen, da so auch eine nicht verwandte Komponente den Zustand der App beeinflussen kann [24].

Daten, die vom Server kommen, sind extra in globale Stores ausgelagert. Diese Stores halten ein Datenmodell, welches mit dem Server synchron und in sich selbst synchron sein soll. Daher sollten diese nicht direkt angesprochen werden, sondern nur über ihr jeweiliges Managerobjekt.

Soll eine Komponente zum Beispiel Daten über die momentan aufgerufene Seite erhalten, kann in dieser der dafür vorgesehenen Store `currentPage` importiert werden. Soll diese Komponente nun eine andere Seite laden oder diese bearbeiten, muss diese Komponente über den `currentPageManager` die jeweilige Methode aufrufen. Der `currentPageManger` würde dann die Änderung an den Server weiterleiten und den Store demnach aktualisieren. Würde der Name einer Seite geändert werden, würde diese zudem dafür sorgen, dass der `PageTreeManager` den `PageTree` aktualisiert, also dass die Information über die verfügbaren Seiten aktualisiert werden.

Auch wenn eine Aktion auf einem Block ausgeführt werden soll oder dessen Daten geändert werden sollen, geschieht dies über den `currentPageManager`. Der ToolStore hat bisher keinen Manager. Für die Tools gibt es aber bisher keine Bearbeitungsmöglichkeit aus der Benutzeroberfläche heraus.

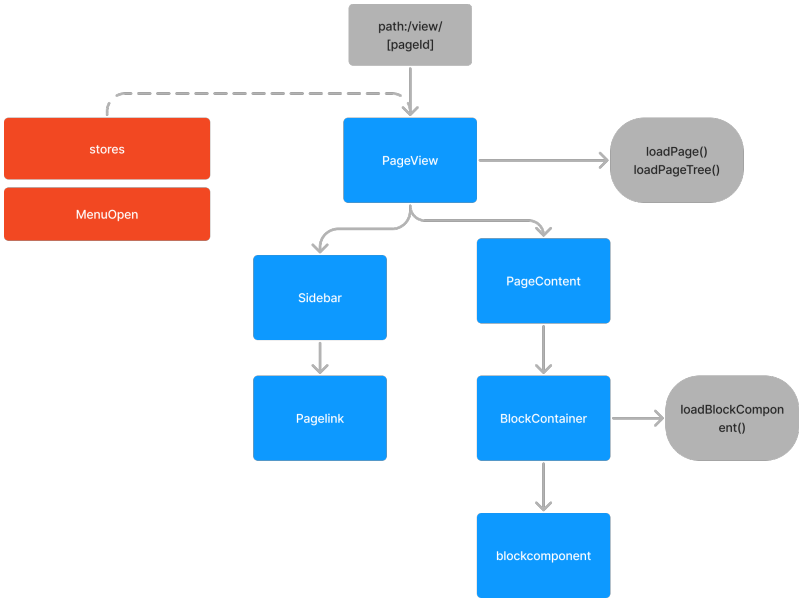


Abbildung 3.2: Vereinfachtes Abhängigkeitsdiagramm von der View-Ansicht

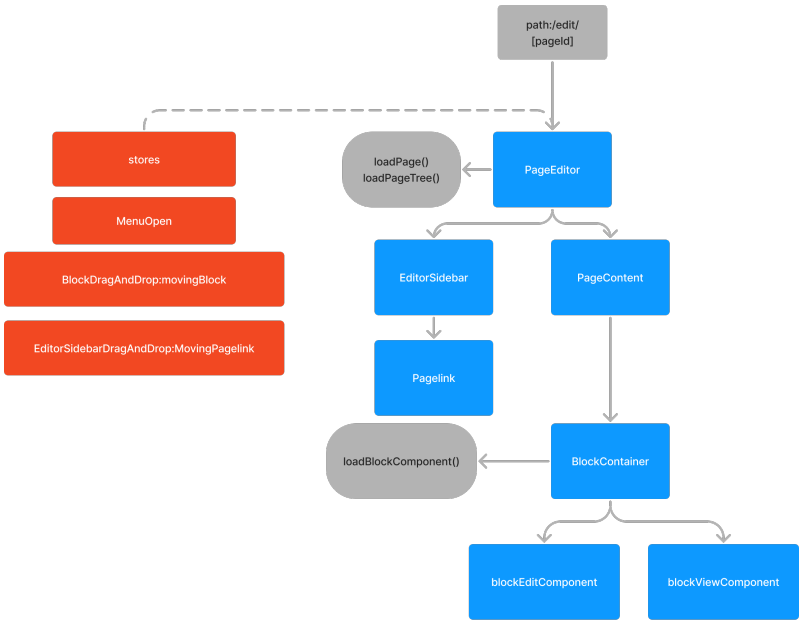


Abbildung 3.3: Vereinfachtes Abhängigkeitsdiagramm von der Editor-Ansicht

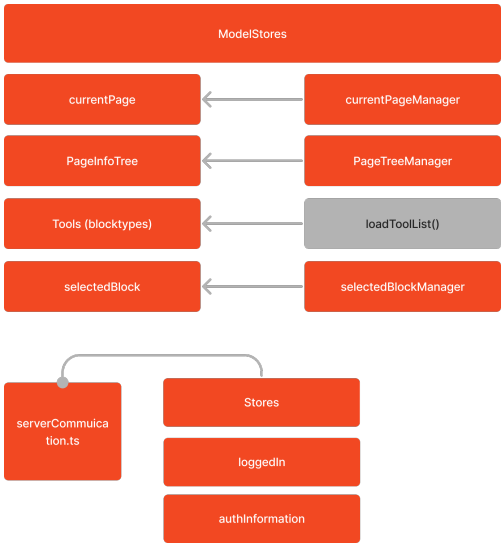


Abbildung 3.4: Globale Module und Stores

3.4.4 Verwaltung von Blöcken durch BlockContainer

Jeder Block wird durch eine Komponente mit dem Namen `BlockContainer` verwaltet. Der `BlockContainer` wird mit den jeweiligen Daten des Blocks und den Daten des Tools, wie diese im Server gespeichert sind, erzeugt.

Für jeden Blocktyp gibt es eine View- und eine Editor-Komponente. Diese können durch das Modul `blockRegistry` dynamisch importiert werden. Standardmäßig zeigt ein BlockContainer einen Block mithilfe seine View-Komponente an. Aktiviert ein Nutzer den Bearbeitungsmodus, lädt der BlockContainer stattdessen die Edit-Komponente. Ein `BlockContainer` bzw. ein Block wird standardmäßig wie auch für den normalen Nutzer angezeigt. Handelt es sich um einen Iframe-Block, wird eine Komponente `IframeContainerView` oder `IframeContainerEditor` geladen und angezeigt.

3.4.5 Iframe-Kommunikation und Iframe-Verwaltung

Der jeweilige `IframeContainer` abstrahiert die Kommunikation, das Laden und das Verwalten des eigentlichen Iframes. So kann, bis auf das Laden der Komponenten, der `BlockContainer` ein Iframe-Plugin, wie jedes andere Plugin, behandeln. Der Container lädt in den Iframe die jeweilige URL des Tools. Danach versucht er, über ein einfaches Protokoll eine Verbindung zu dem Iframe aufzubauen.

Standardmäßig erlaubt der Browser nicht, dass Daten einer externen Web-App in einem Iframe von der übergeordneten Web-App gelesen oder manipuliert werden. Will eine Web-App in einem Iframe mit der übergeordneten Web-App Daten teilen, müssen

die Daten über die Methode `Window.postMessage` ausgetauscht werden. Die Methode `Window.postMessage` ist Teil der Browser-API und ermöglicht es, beliebige Datenobjekte zu verschicken. Durch das "message"-Event können diese Daten dann empfangen werden.[25]

Im `IframeContainer` und in der `IframeConnection` Bibliothek wurde ein einfaches Kommunikationsprotokoll implementiert. Der jeweilige `IframeContainer` übernimmt die Abstrahierung und Implementierung der Kommunikation auf der Seite des Teaching-Books und die "teachingBookConnection"-Library auf der Seite des Iframes.

Dieses Protokoll baut eine Verbindung auf und teilt der Iframe-Web-App mit, zu welchem Block diese gehört, dies ist wichtig, da im weiteren Protokoll die Identifikationsnummer des Blocks zur Identifizierung benutzt wird und bei jeder Nachricht von einer Iframe-Web-App mitgeschickt wird. Da die Höhe des Inhalts des Iframes nicht bekannt ist, verschickt die `teachingbookConnection` nach dem Aufsetzen der Verbindung immer die aktuelle Höhe der Iframe-Web-App an den `IframeContainer`, sodas dieser den Iframe auf genau die Höhe anpassen kann. Alternativ wird nur zu Beginn eine statische Höhe gesetzt.

Zudem bekommt die Iframe-Web-App immer die aktuellen Block-Daten mitgeteilt. Soll eine Block-Aktion ausgeführt werden, sollte meistens auch auf die Antwort dieser gewartet werden. Deswegen muss diese Art von Anfrage besonders behandelt werden. Schickt eine Iframe-Web-App eine solche Anfrage und ist die Block-Aktion erfolgreich ausgeführt worden, schickt das Teaching-Book-Frontend eine Antwort zu der jeweiligen Anfrage. Die Anfrage wird dabei mit einem Integer zur Identifikation versehen. Die Antwort wird dann mit demselben Integer zurückgeschickt. Die Anfrage kann so der richtigen Antwort zugeordnet werden.

Für den Bearbeitungsmodus gibt es zusätzlich noch eine Nachricht vom Teaching-Book, wenn der Versuch unternommen wird, den Bearbeitungsmodus zu verlassen. Der Bearbeitungsmodus wird aber nur verlassen, wenn die Iframe-Web-App die Nachricht "saved" an das Frontend schickt, damit der Bearbeitungsmodus nicht geschlossen wird, bevor die bearbeiteten Daten gespeichert werden können.

4 Plugin-System

Ein wichtiges Element des Projekts ist das flexible Plugin-System. Dieses ermöglicht es auf verschiedene Weise, mit geringem Aufwand neue Arten von Blöcken zu implementieren. Somit ist es realistisch, in einem geringen Zeitraum viele Inhaltsformen zu implementiert. Mögliche Arten von Blöcken sind dabei neue Visualisierungsmöglichkeiten, Aufgabentypen oder interaktive Beispiele. Diese werden im weiteren auch Tools, Blocktypen oder Plugins genannt. Dieses Kapitel soll eine Übersicht über das Plugin-System geben und ein Handbuch für zukünftige App-Entwickler darstellen. Zuerst wird erklärt, welche verschiedenen Möglichkeiten es gibt neue Blocktypen zu implementieren. Dabei wird besprochen welche Vor- und Nachteile diese haben und welche Implementierungsmethode wann zu bevorzugen ist. Danach folgt eine Beschreibung des Prozesses zum Erstellen von neuen Plugins. Diese Beschreibung erläutert erst den Anmeldeprozess, den jedes Plugin durchlaufen muss. Anschließend werden die zwei verschiedenen Implementierungsmöglichkeiten im Frontend beschrieben. Zudem wurde für beide Implementierungsmöglichkeiten ein Beispiel implementiert.

4.1 Plugin Implementierungsarten

Das Besondere an diesem Plugin-System ist, dass entweder nativer framework-spezifischer Code geschrieben werden kann oder alternativ andere Technologien verwendet werden können. Auf der Frontendseite gibt es die Möglichkeit, entweder Svelte-Komponenten zu schreiben oder Web-Apps durch Iframes einzubinden. Es gibt grundlegend also erstmal zwei Arten von Plugins.

Es ist eines der Kernkonzepte des Plugin-Systems, dass die Daten der Blöcke und Nutzer zentral im Teaching-Book-Backend vorliegen. Backendfunktionalität, um Daten zu speichern, gibt es daher bereits für jeden Block im Teaching-Book-Backend [13]. Weiter Backendfunktionalität kann über einen externen Backend-Service-Endpunkt realisiert werden, der im Teaching-Book-Backend registriert werden kann [13]. Außerdem können aus dem Frontend bzw. Iframe oder aus den Svelte-Komponenten weitere Services aufgerufen werden. Für Plugins, die einen gewissen Standard darstellen oder einen Teil des Projekts selbst darstellen, kann Backendfunktionalität auch nativ als Teil des Projektes programmiert werden [13].

Im Moment ist es nicht ratsam, direkt in einem fremden Backend mit dem Teaching-Book zu kommunizieren, da das Teaching-Book-Frontend nicht signalisiert bekommt, wenn sich der Zustand eines Blocks ändert.

4.1.1 Die Wahl der richtigen Implementierungsart

Die wichtigsten Punkte sind, dass ein Iframe-Plugin immer dann eine gute Wahl ist, wenn das Plugin auch außerhalb des Teaching-Book verwendet werden soll oder bereits eine Web-App existiert. Ansonsten ist eher dazu zu raten, Svelte-Komponenten zu schreiben. Die Nachteile von Iframe-Plugins halten sich aber gerade für Plugins, die nur einige wenige Male pro Seite auftauchen, in Grenzen, weswegen es akzeptabel ist, wenn es dem Entwickler wichtig ist, eine andere Technologie zu verwenden. Für die Wartbarkeit und Geschwindigkeit sollten trotzdem native Plugins der bevorzugte Ansatz für Neuentwicklungen sein. Auch sollte darauf geachtet werden, dass sehr oft benutzte Plugins keine Iframes sind, damit ein Dokument nicht aus zu vielen Iframes besteht.

Vorteile und Nachteile von Svelte-Komponenten-Plugins

Dadurch, dass Svelte das Framework ist, in welchem der Rest des Frontends geschrieben wurde, funktionieren hier viele Features nahtloser und besser. Eine Svelte-Komponente lädt schneller, der Zustand bleibt auch erhalten, wenn er im Dokument verschoben wird und momentan funktioniert Fehlerbehandlung für Backend-Requests besser.

Ein anderer Vorteil ist, dass durch Svelte die Komponente jeweils nur aus einer Datei bestehen muss, das ist praktisch für kleine Visualisierungen oder einfache Plugins.

Für eine Svelte-Komponente muss kein weiterer Server aufgesetzt werden, da das Plugin zu einem nativen Teil der Web-App wird. Es muss keine neue Web-App geschrieben werden, sondern nur eine Komponente. Die Struktur von Svelte ist einfach, sodass auch ohne viel framework-spezifischen Code zu schreiben viel erreicht werden kann. Der Nachteil liegt vor allem darin, dass eben svelte-spezifischer Code geschrieben werden muss. Wichtig ist hier aber zu erwähnen, dass Svelte allgemein ein sehr beliebtes Framework bei Entwicklern ist [19] [20].

Trotzdem kann das natürlich ein Hindernis sein, wenn bereits eine Web-App vorliegt oder der Entwickler ein anderes Framework bevorzugt. Bei größeren Plugins, die auch außerhalb des Teaching-Books als eigene Web-App funktionieren sollen, könnte es einfacher sein, diese auch direkt separat zu entwickeln. Gerade da davon ausgegangen werden muss, dass viele Studenten weitere Plugins als Teil ihrer Bachelorarbeit oder ähnliches erstellen, wäre es eine zu große Einschränkung, diese zu zwingen eine bestimmte Technologie zu nutzen.

Wenn möglichst viele Plugins in der gleichen Technologie geschrieben sind, ist das natürlich auch ein Vorteil für die Wartbarkeit, weil die Entwickler weniger Zeit damit verbringen neue Frameworks zu lernen.

Einen neuen Blocktypen durch native Svelte-Komponenten zu entwickeln ist dann empfehlenswert, wenn sich noch nicht für eine Technologie entschieden wurde, das Plugin die Geschwindigkeit oder die Integration benötigt, sehr oft im Teaching-Book verwendet wird, oder generell, wenn das Tool vor allem als Plugin für das Teaching-Book entwickelt wird.

Vorteile und Nachteile von Iframe-Plugins

Der größte Vorteil eines Iframe-Plugin ist, dass die Wahl der Technologie frei ist und das Plugin als eigenständige Web-App entwickelt werden kann. Das ist vor allem ein Vorteil, wenn ein Tool auch außerhalb des Teaching-Books verwendet werden soll, oder wenn bereits existierende Web-Apps eingebunden werden sollen, da nur sehr wenig dafür verändert werden muss. Die meisten Nachteile entstehen durch die Beschränkungen von Iframes und durch die begrenzten Features, die das Interface zwischen dem Iframe-Plugin und dem Teaching-Book bereitstellt.

Eine Web-App hat standardmäßig keine Möglichkeit, direkt auf Daten oder generell den Zustand einer externen Web-App in einem Iframe zuzugreifen [25]. Es gibt die Möglichkeit, über eine Web-API mit der Web-App ähnlich wie mit einem Server zu kommunizieren, das bedeutet, Daten können explizit geschickt und empfangen werden [25]. Damit Daten und Funktionalität zwischen Iframe-Plugins in Teaching-Book geteilt werden können, wurde ein Protokoll umgesetzt, welches ermöglicht, mit dem Teaching-Book-Frontend zu kommunizieren. Dieses bietet die Möglichkeit, blockspezifische Daten zu erhalten oder zu verändern und Block-Aktionen auszuführen. Dieses Protokoll wird hinter dem Modul `teachingbookConnection` versteckt. Diese kann importiert und verwendet werden, ohne das Protokoll selbst zu kennen. Ein Iframe-Plugin ist dadurch in seiner Funktionalität durch die Features, die die `teachingbookConnection` mit sich bringt, begrenzt. Es ist davon auszugehen, dass dieses in Zukunft weitere Funktionalität erhalten muss. Im Moment bietet die `teachingbookConnection` keine Möglichkeit, Fehlerbehandlung für die Anfragen an das Teaching-Book zu betreiben. Im Gegensatz zu den nativen Plugins funktioniert das Editieren durch doppeltes klicken nicht, da das dafür nötige Event nicht weitergeleitet wird. Dies sind weitere Nachteile, die bei weiterer Entwicklung behoben werden können.

Iframes haben einen Performance-Nachteil [26]. Im Prinzip verhalten sie sich wie eine eigenständige Web-App, die parallel ausgeführt wird [26]. Ressourcen, wie zum Beispiel benötigter Speicherplatz, werden dadurch beeinträchtigt [26]. Wenn weitere Ressourcen, wie ein großes Framework, heruntergeladen werden, wird die Seite sehr schnell relativ groß. Viele Iframes auf einer Seite könnten also zu einem Problem werden [26].

Ist ein Plugin auch als eigenständige Web-App verfügbar, existiert bereits eine Web-App, welches die Features des erwünschten Plugins implementiert, oder soll eine andere Technologie verwendet werden,ist ein Iframe-Plugin eine Wahl, die zwar Nachteile hat, die sich aber in Grenzen halten. Tools, die sehr häufig verwendet werden, sollten nicht als Iframe-Plugin entwickelt werden.

4.2 Implementierung neuer Plugins

Dieses Kapitel behandelt, welche Schritte ein Plugin durchläuft und wie Daten über einen Block im Teaching-Book-Backend gespeichert werden. Danach wird erklärt, wie weitere Backend-Funktionalität, wenn nötig, hinzugefügt werden kann. Anschließend folgt ein Unterkapitel, das sich mit der Implementierung auf der Frontend Seite durch

Svelte-Komponenten beschäftigt und eines, welches sich mit der Möglichkeit beschäftigt, Web-Apps durch Iframes einzubinden.

4.2.1 Plugin Registrieren und Daten festlegen

Jedes Plugin bzw. Tool, wie es im Backend genannt wird, muss zuerst beim Teaching-Book-Backend über eine Http-Request am Endpunkt "POST tools" registriert werden. Neue Tools können nur Benutzer mit einem `Staff`-Status anlegen. Dieser Status gibt die Berechtigung an, ob ein Benutzer Bearbeitungsrechte hat. Für die Request ist ein Authentifizierungstoken nötig, der über eine Http-Request angefragt werden kann. Ein neues Tool muss mit einem Namen angelegt werden, der mit einem Großbuchstaben anfängt. Dieser dient zur Identifikation des Tools, er muss also einzigartig sein.[13]

Ein Tool hat automatisch verschiedene Datenbereiche, auf die ein Block zugreifen kann und in denen die Daten eines Blocks und Tool gespeichert werden sollten. Es gibt die Datenbereiche `tool_data`, `public_data`, `secret_data` und `user_data`. Diese unterscheiden sich durch ihre Sichtbarkeit bzw. Tragweite. Der Bereich `tool_data` ist für jeden Block des gleichen Tools gleich. Die Bereiche `public_data` und `secret_data` werden pro Block angelegt. Der Bereich `user_data` hingegen wird pro User zu einem Block angelegt. Die Bereiche `tool_data` und `secret_data` sind zugriffsbeschränkt, diese sind nur für "Staff"-Nutzer oder für im Backend registrierte Aktionen einsehbar und veränderbar. Für normale Benutzer sind nur `public_data` und `user_data` zur Bearbeitung freigegeben. Wenn ein Tool nun registriert wird, können initiale Daten für diese Datenbereiche angelegt werden. Der Datentyp jedes neuen Feldes, welches in diese Bereiche abgelegt wird, wird zudem durch diese initialen Daten festgelegt.[13]

Optional kann angegeben werden, ob es gewünscht ist, dass sich der Block im Bearbeitungsmodus öffnet, wenn auf den Block doppelt geklickt wird [13]. Für Iframe-Plugins funktioniert dies momentan nicht.

Wenn es sich um ein Iframe-Tool handelt, muss das `is_iframe` Attribut auf `true` gesetzt werden und es muss eine URL angegeben werden. Wenn für den Bearbeitungsmodus eine andere URL benutzt werden soll, muss hierfür eine extra Admin-URL angegeben werden, ansonsten wird die gleiche benutzt [13].

4.2.2 Backend-Funktionalität hinzufügen und Block-Aktion registrieren

Um externe Backend-Funktionalität hinzuzufügen, kann ein Service-Endpunkt eines externen Backends bzw. Webserver als sogenannte Block-Aktion im Teaching-Book-Backend registriert werden[13]. Eine andere Möglichkeit ist es, direkt im Frontend externe Service-Endpunkte anzusprechen. Das hat den Nachteil, dass diese nur mit Daten aufgerufen werden können, die für den jeweiligen Benutzer zugänglich sind. Für spezielle Blocktypen, welche eigentlich Teil des Projekts selbst sind, können solche Aktionen auch direkt im Backend Code hinterlegt werden, wie es zum Beispiel bei dem PandocHtml-Tool der Fall ist [13].

Eine Block-Aktion hingegen wird so registriert, dass bei der Registrierung festgelegt wird, welche Daten sie aus dem Block benutzt und welche Daten sie im Block ablegt.

Dadurch kann sie auch auf Daten zugreifen, die nicht von einem normalen Nutzer einsehbar sind. Trotzdem kann die Aktion von einem normalen Nutzer aufgerufen werden. Eine Block-Aktion für ein Tool muss im Backend mit einer Anfrage zum Endpunkt "POST tools/:tool_name/ " registriert werden. Auch dafür ist eine Authentifizierung als `Staff`-Nutzer nötig. Eine Aktion wird mit einem eindeutigen Namen registriert. Ansonsten benötigt die Registrierung noch die URL, die Http-Methode und wenn nötig den Content-Typ des Endpunktes, die die Aktion ansprechen soll. Zusätzlich kann angegeben werden, welche Daten in den Header und welche Daten in den Body der Anfrage gespeichert werden sollen. Außerdem kann angegeben werden, wie die Daten aus der Antwort der Anfrage im Block gespeichert werden. Diese Angaben folgen einer einfachen Zuordnung. $\{'solution' : user\_data[sol]\}$ würde beispielsweise bedeuten, dass dem Feld Solution der Request dem Feld `sol` im jeweiligen `user_data` Objekt zugeordnet wird. So eine Zuordnung kann jeweils für den Body, Header und für die Antwort erfolgen. Um Daten der Anfrage, besser formatieren zu können, gibt es zusätzlich eine kleine Sprache, die es auch erlaubt, mehrere Daten, verknüpft mit Zeichenketten, in ein Feld zu schreiben.[13] Im Frontend gibt es je nach Implementierungsart Methoden, um für einen Block eine registrierte Block-Aktion auszuführen.

4.2.3 Native Plugins durch Svelte-Komponenten

Nachdem ein Plugin im Server mit den nötigen Aktionen, wie zuvor erklärt, registriert wurde, kann das Frontend des Plugins durch Svelte-Komponenten realisiert werden. Dieser Prozess soll im Folgenden erklärt werden. Für das Verständnis ist nur ein Basiswissen zum Thema Webentwicklung nötig. Um die folgende theoretische Beschreibung des Vorgehens zu ergänzen, wurde das WoFA-Tool auf diese Weise implementiert. Für ein neues Plugin bzw. einen neuen Blocktypen, der durch Svelte-Komponenten realisiert wird, sollte ein eigener Ordner angelegt werden, welcher alle Komponenten und Ressourcen des Plugins beinhaltet, dieser kann dann im Ordner `src/blocks` abgelegt werden. Alternativ kann für ein Plugin ein externes Repository angelegt werden. Jedes Plugin braucht mindestens zwei Svelte-Komponenten. Eine Komponente für den Bearbeitungsmodus und eine Komponente für die normale Ansicht. Zudem braucht jedes Plugin eine Konfigurationsdatei, die den Namen des Tools enthält und auf die beiden Komponenten verweist. Diese muss in dem Modul `blockRegistry` registriert werden. Im Weiteren soll dies noch detaillierter erläutert werden.

viewComponent und editComponent

Die Komponente für den Bearbeitungsmodus nennen wir im folgenden, wie im Quellcode, `editComponent` und die Komponente für die normale Ansicht `viewComponent`. Im `editComponent` kann davon ausgegangen werden, dass der Nutzer über den `Staff`-Status verfügt. Es können also auch eingeschränkte Datenbereiche bearbeitet werden. Im `viewComponent` hingegen, hat der Nutzer eventuell nur eingeschränkte Rechte. Es wird erwartet, dass `editComponent` und `viewComponent` ein Attribut `Block` bereitstellen, damit die jeweilige Komponente Zugriff auf die Daten des jeweiligen Blocks erhält. Darüber hinaus, soll im `editComponent` eine asynchrone Funktion `onSave` ex-

portiert werden, die aufgerufen wird, wenn der Bearbeitungsmodus verlassen wird bzw. versucht wird, die Änderung an dem jeweiligen Block zu speichern. Außerdem wird erwartet, dass das `editComponent` eine asynchrone Funktion `onEnterEditMode` exportiert, die aufgerufen wird, wenn der Bearbeitungsmodus betreten wird. Aktionen eines Tools kann eine Komponente ausführen, indem es den `CurrentPageManager` importiert und hier die Methode `blockAction` aufruft. Die Daten im Attribute `block` werden so automatisch aktualisiert. Ist es nötig, ein Text oder Code-Editor einzubinden, wird dazu geraten, CodeMirror[1] zu benutzen. CodeMirror[1] ist bereits installiert. Der Default Markdown-Editor im PandocHtml-Tool baut darauf auf. Den gleichen Editor zu benutzen hat Vorteile für Benutzbarkeit und Ladegeschwindigkeit. Der Benutzer gewöhnt sich an Features und es muss keine weitere Editor-Bibliothek geladen werden. Außerdem erfüllt der Editor die Anforderungen, auf mobilen Geräten zu funktionieren.

Konfigurationsdatei bzw. BlockConfig

Die Konfigurationsdatei eines Blocks muss ein Objekt exportieren, welches das Interface `BlockConfig` erfüllt. Dieses Objekt soll den Namen des Block-Typs bzw. des Tool beinhalten, genauso, wie es im Teaching-Book-Backend registriert worden ist. Außerdem muss das Objekt zwei asynchrone Funktionen bereitstellen, welche jeweils die Svelte-Komponente zurückgeben, die im jeweiligen Modus angezeigt wird. Der `viewComponentImport` importiert die Komponente, die dafür verantwortlich ist, wie das Tool normalen Nutzern dargestellt wird. Der `editComponentimport` importiert hingegen die Komponente, die zum Editieren des Blocks dient, also im Bearbeitungsmodus angezeigt wird. Auch unter `src/blocks`, in der Datei `blockRegistry.ts` befindet sich die `blockRegistry`. Jedes Plugin muss seine Konfiguration hier registrieren. Die `blockRegistry` ist eine JavaScript-Map, die vom Typ String zu dem Interface `BlockConfig` zuordnet. Hier muss ein neuer Eintrag, eine Zuordnung von dem Attribut `typeName` der neuen `BlockConfig` zu der `BlockConfig` selbst, hinzugefügt werden.

4.2.4 Beispielimplementierung: WoFA

Um den Entwicklungsprozess eines nativen Plugins zu verdeutlichen und ein Beispiel für diese Art von Implementierung zu zeigen, wurde auf diese Weise das WoFA-Plugin implementiert. Der Frontend-Quellcode für dieses Plugin ist Teil des Teaching-Book-Repository. Das WoFA-Tool soll es einem Dozenten ermöglichen, in einem Vorlesungsskript Aufgaben zu dem Thema endliche Automaten zu stellen. Der Dozent bzw. "Staff"-Nutzer soll dabei eine Lösung und ein Template für einen endlichen Automat in textueller Form angeben. Der normale Nutzer bzw. Leser des Vorlesungsskriptes kann dann versuchen, einen Automaten anzugeben, welche der Sprache der Lösung entspricht. Der Benutzer hat dazu das Template zur Verfügung und kann seinen Lösungsversuch speichern und automatisch bewerten lassen. Die Bewertung erfolgt dabei über ein externes WoFA-Backend. Dieses WoFA-Backend ist in einem sepparaten Projekt entstanden [12] und stellt unter anderem einen Endpunkt bereit, bei dem zwei Automaten in einem speziellen Textformat angeben werden können. Das

[1] CodeMirror: https://codemirror.net/

WoFA-Backend kann dann eine Bewertung ausrechnen, die davon abhängt, wie nah die Sprache des einen Automaten an der des anderen liegt [12]. In diesem Fall also, wie nah der Lösungsversuch an der Lösung liegt. In den Abbildungen 4.1 und 4.2 sehen wir das Ergebnis. In der Abbildung 4.1 ist das Tool im Ansichtsmodus zu sehen. Abbildung 4.2 ist das das Tool im Bearbeitungsmodus zu sehen. Das Tool befindet sich in den Abbildung eingebettet in einer Seite mit Aufgabenstellung. Der Text, der die Aufgabenstellung darstellt, gehört nicht zum eigentlichen Tool.

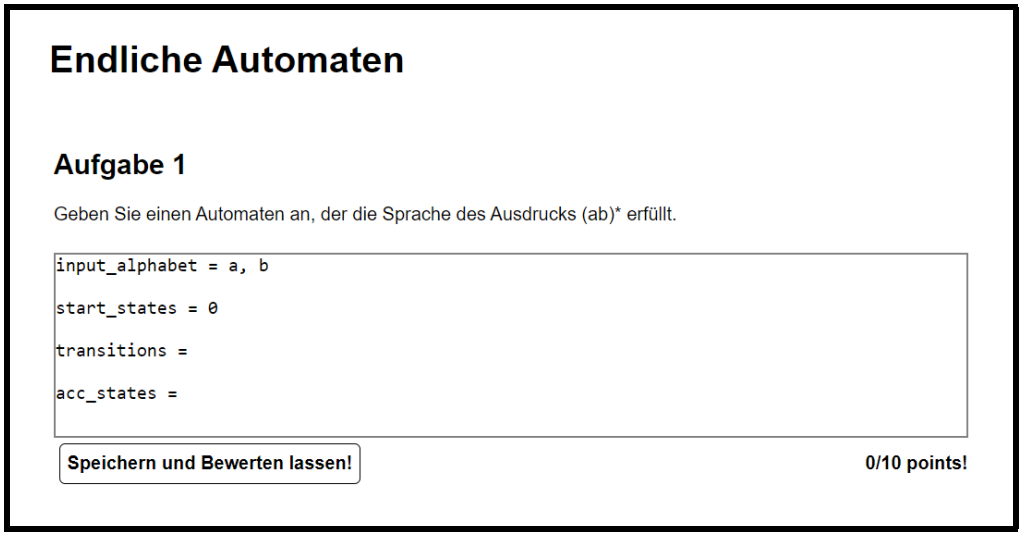


Abbildung 4.1: WoFA-Tool im Ansichtsmodus

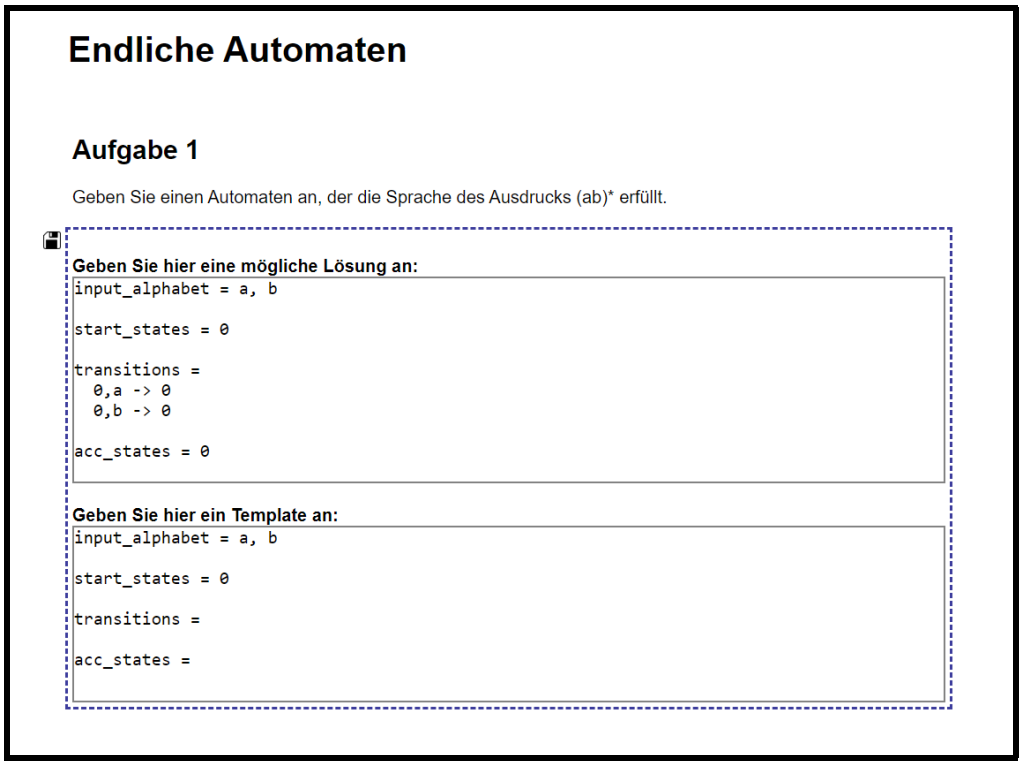


Abbildung 4.2: WoFA-Tool im Bearbeitungsmodus

4.2.5 Implementierung von Iframe-Plugins

Damit eine beliebige Technologie genutzt werden kann, kann ein neuer Blocktyp auch als eigene Web-App realisiert werden. Diese Web-App kann dann über eine Bibliothek namens `teachingbookConnection` mit dem eigentlichen Teaching-Book-Frontend kommunizieren. In diesem Kapitel wird der Prozess, ein neues Plugin als Web-App in einem Iframe zu implementieren, zusammenfassend erklärt, ergänzend dient als Beispiel das Calculator-Tool. Zuerst muss das Iframe-Plugin, wie die anderen Plugins, per Anfrage an das Teaching-Book-Backend registriert werden, wie es im Kapitel 4.2.1 beschrieben wurde. Wichtig ist hierbei, dass die Flag `is_iframe` gesetzt wird und das die URL zu der Web-App angegeben wird. Optional kann für den Bearbeitungsmodus eine `admin_url` registriert werden [13], ansonsten wird auch hier die normale URL benutzt. Nun kann bereits ein Block dieses Typs erstellt werden. Die Web-App kann aber noch nicht mit dem Teaching-Book kommunizieren. Außerdem hat ein Iframe des Blocktypens noch keine angepasste Größe, da diese aus der Iframe-Web-App heraus kommuniziert werden muss.

Verbindung zum Frontend aufsetzen

Zur Kommunikation mit dem Teaching-Book-Frontend muss die Web-App die Bibliothek `teachingbookConnection` importieren, welche sich im Teaching-Book-Repository finden lässt.

Nun muss bei der Initialisierung der Web-App die Funktion `setUpTeachingBook-Connection` aus dem Modul `teachingbookConnection` aufgerufen werden. Diese Funktion stellt die Verbindung zu dem Teaching-Book-Frontend her. Die Funktion erwartet aus Sicherheitsgründen die URL des Teaching-Book-Frontends.

Höhe des Iframes/Blocks konfigurieren

Standardmäßig konfiguriert diese Methode die Verbindung so, dass immer die aktuelle `documentElement`-Höhe der Web-App an das Teaching-Book-Frontend gesendet wird, so dass das Teaching-Book-Frontend den jeweiligen Iframe dynamisch auf diese Höhe konfigurieren kann. Optional kann alternativ in der `setUpTeachingBookConnection` Funktion eine statische Höhe angegeben werden. Wenn es nicht erwünscht ist, dass das Plugin eine Scrollbar hat, sollte diese durch eine CSS-Regel in den äußeren Containern der Web-App deaktiviert werden. Auch, wenn der Iframe auf die Größe der Web-App gesetzt wird, würde sonst durch die Verzögerung der Initialisierung die Scrollbar aufflackern. Soll die Höhe dynamisch gesetzt werden, sollte außerdem darauf geachtet werden, dass kein äußerer Container eine statische Höhe besitzt, die verhindert, dass der Container seine Größe ändert. Auch eine Höhe, die in Prozent angegeben wurde, hätte hier diesen Effekt.

Kommunikation mit dem Teaching-Book

Um nun im weiteren Verlauf mit dem Teaching-Book kommunizieren zu können, gibt die Methode `setUpTeachingBookConnection` ein Objekt der Klasse `Teachingbook-`

`Connection` zurück, welches wir `teachingbookConnection` nennen, wie das Modul. Dieses Objekt muss nicht gespeichert werden. Die `teachingbookConnection` ist ein Singleton, welche durch die Methode `getTeachingBookConnection` von überall aus aufgerufen werden kann. Die `teachingbookConnection` ermöglicht es, die aktuellen Daten des Blocks zu erhalten und über Datenänderungen informiert zu werden. Des Weiteren kann erfragt werden, ob sich der Block im Bearbeitungsmodus befindet. Zusätzlich dazu ermöglicht die `teachingbookConnection` das Ausführen bzw. Anfragen von Block-Aktionen. Für den Bearbeitungsmodus ist es außerdem wichtig, durch die `teachingbookConnection` festzulegen, welche asynchrone Funktion aufgerufen wird, wenn der Bearbeitungsmodus verlassen wird bzw. die neuen Eingaben gespeichert werden sollen. Hier ist wichtig, das der Bearbeitungsmodus nur verlassen wird, wenn die hier angegebene Funktion erfolgreich terminiert.

4.2.6 Beispielimplementierung: Calculator-Tool

Als Beispiel wurde die Web-App "SimpleApp"[2] als "Calculator"-Tool implementiert. Die "SimpleApp" ist eine sehr einfache Web-App, die nur zu Testzwecken im Fachgebiet "Theoretische Informatik / Formale Methoden" entstanden ist. Die Web-App basiert auf dem Frontend-Framework Angular[3] und hat ein Node.js-Express-Backend[4]. Sie eignet sich dazu, zu zeigen, dass durch diese Art der Implementierung auch bereits existierende Web-Apps, die auf anderer Technologie basieren, leicht durch Iframes in das Teaching-Book integriert werden können. Der Quellcode für dieses Plugin ist im Repository der "SimpleApp"[2] unter dem Zweig "teachingbookView" zu finden.

Diese Web-App besteht aus zwei Pfaden bzw. zwei Tabs, wie in Abbildung 4.3 und Abbildung 4.4 zu sehen ist. Auf einem Pfad können zwei Zahlen und eine arithmetische Operation eingegeben werden. Wird dann auf den "Execute Calculation"-Pfeil geklickt, rechnet ein Server die eingegebene Rechnung aus. Das Ergebnis wird dann auf dem zweiten Tab mit dem Namen "output" angezeigt. Das Tool soll die Möglichkeit bieten, dass ein Nutzer mit `Staff`-Status im Bearbeitungsmodus eine Berechnung vordefinieren kann bzw. seine Eingaben speichern kann. Im Ansichtsmodus sieht ein normaler Nutzer dann initial, die Berechnung, die der `Staff`-Nutzer eingestellt hat. Der normale Nutzer kann die Eingaben dann aber auch verändern. Diese Funktionsweise ist immer dann geeignet, wenn ein Plugin interaktive Beispiele oder Visualisierungen implementieren soll. Die Abbildung 4.3 und 4.4 zeigt das fertige eingebundene Plugin im Bearbeitungsmodus und die Abbildung 4.5 zeigt das Plugin im Ansichtsmodus.

[2] Repository der SimpleApp: https://syre.fm.cs.uni-kassel.de/mherwig/simple-app
[3] Angular: https://angular.io/
[4] express.js: https://expressjs.com/de/

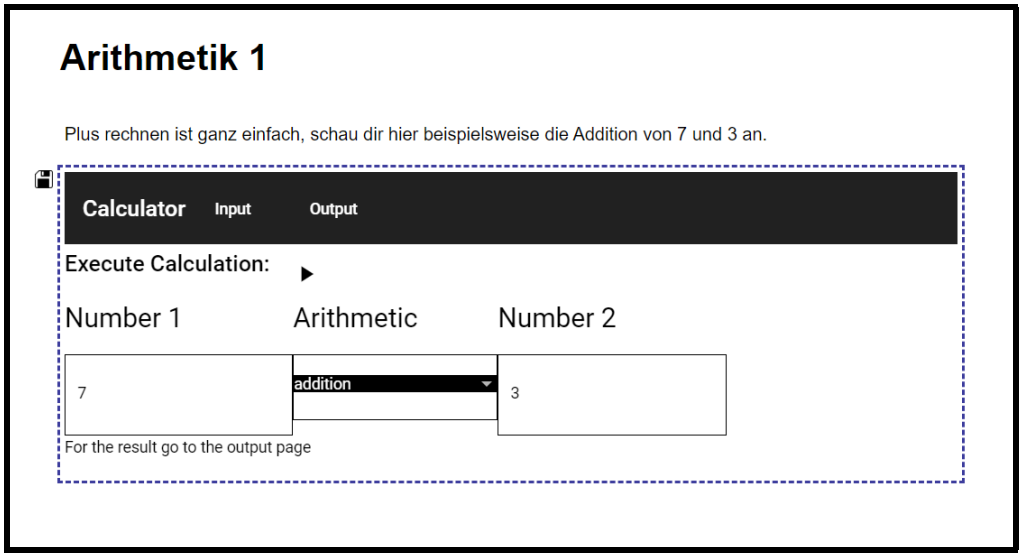


Abbildung 4.3: Calculator im Bearbeitungsmodus

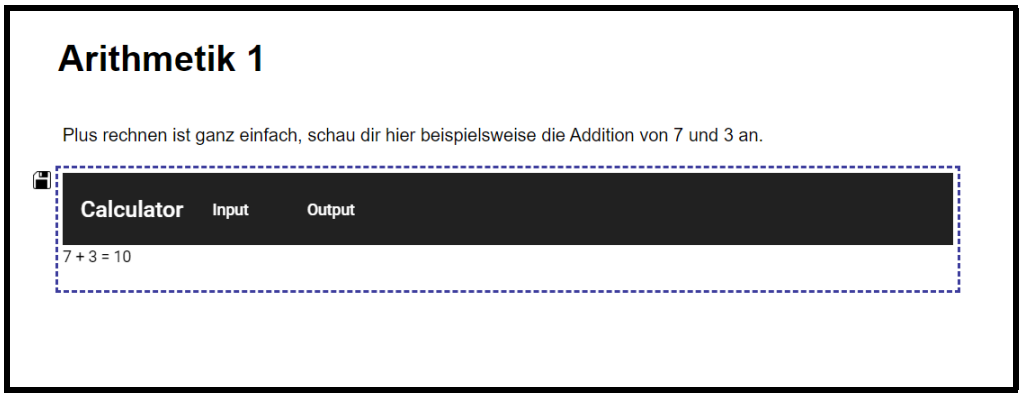


Abbildung 4.4: Calculator im Bearbeitungsmodus auf dem Ausgabepfad

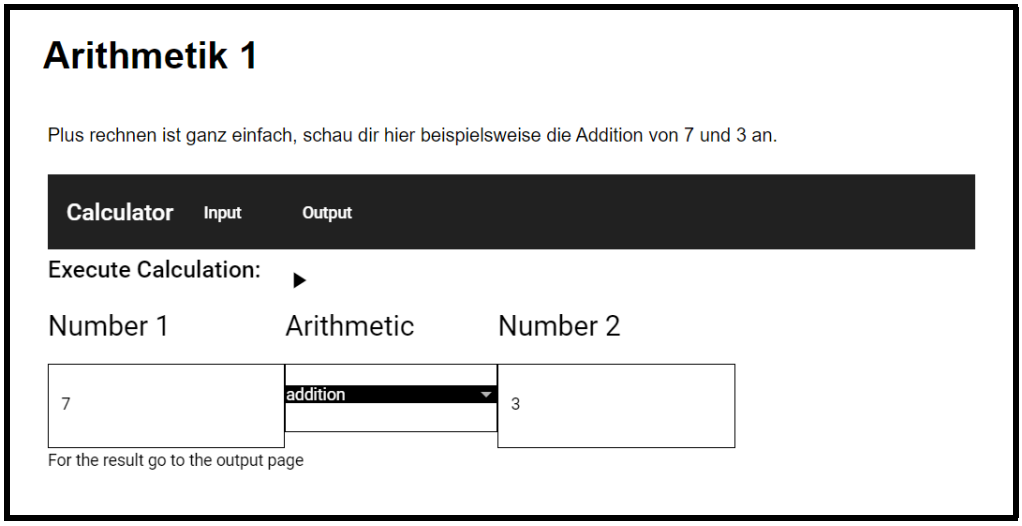


Abbildung 4.5: Calculator im Ansichtsmodus

5 Teaching-Books in der Praxis

Um die Praxistauglichkeit des Prototypen besser einordnen zu können wurde eine Nutzerstudie durchgeführt. Die Probanden haben in dieser Nutzerstudie verschiedenes Feedback gegeben. Die allgemeine Bewertung des Prototypen ist positiv ausgefallen. Dennoch wurden, wie bei einem Prototyp zu erwarten, einige Probleme und Limitierungen festgestellt. In diesem Kapitel werden die Nutzerstudie und das entstandene Feedback diskutiert. Darüber hinaus werden allgemein Probleme und Verbesserungspotential besprochen.

5.1 Durchführung der Nutzerstudie

Die Nutzerstudie wurde mit Mitarbeitern des Fachgebiets Theoretische Informatik / Formale Methoden der Universität Kassel durchgeführt. Schließlich wurde die Software mit dem Fokus auf diesem Fachgebiet entwickelt. Diese Nutzerstudie soll dabei keine vollständige Überprüfung darstellen, sondern nur dabei helfen, den Prototypen besser einzuordnen. Teil der Durchführung waren ein Nutzertest und eine zweiteilige Befragung. Insgesamt haben sechs Personen an dem Nutzertest teilgenommen, wobei zwei Personen der Gruppe die Software schon im Vorhinein kannten, da sie in den Entwicklungsprozess involviert waren. Die anderen vier Teilnehmer kannten die Software noch nicht. Einer der Teilnehmer ist jedoch nicht am Vorlesungsbetrieb beteiligt.

Die Durchführung von dieser Nutzerstudie kann in vier Phasen unterteilt werden:

1. Zuerst sollten die Nutzer den ersten Teil der Befragung ausfüllen, in der sie mit zwei Fragen zu ihrem Vorwissen befragt wurden. Dieser erste Teil der Befragung, ist in Abb. 5.1 zu sehen.

2. Danach wurde der Zweck der Software und die Bedienung erklärt. Dabei wurde die Software live vorgeführt. Die Shortcuts wurden hier zwar kurz erwähnt, wurden aber nicht vorgeführt, sie waren also nicht Fokus der Studie.

3. Nun sollten die Probanden den Prototypen ausprobieren. Die Probanden haben einen Laptop mit ihrer eigenen Instanz des Prototyps erhalten, in der sich bereits in die Bearbeitungsansicht eingeloggt wurde. Die Probanden wurden nun damit beauftragt, eine Seite zu erstellen, die einer Vorlage entspricht, die sie in ausgedruckter Form zur Verfügung gestellt bekommen haben. Die Vorlage soll nur einen Leitfaden bieten, um wichtige Funktionen auszuprobieren und ein Beispiel für die verschiedenen Blocktypen bieten. Ziel war hier nicht, dass die Probanden die Vorlage genauestens umsetzen. Die Vorlage ist in Abb. 5.3 zu sehen.

4. Zuletzt wurde der zweite Teil der Befragung ausgefüllt. In diesem Teil der Befragung ging es darum, Feedback zu geben und den Prototypen zu bewerten. Dieser erste Teil der Befragung, ist in Abb. 5.2 zu sehen.

Die Nutzerstudie wurde in zwei Gruppen mit jeweils drei Teilnehmern durchgeführt. Die Teilnehmer durften jederzeit Fragen stellen und haben auch mündlich Feedback gegeben.

5.2 Ergebnisse der Nutzerstudie

Die meisten Teilnehmer kannten die Tools oder haben zumindest schon mal davon gehört haben. Drei Personen benutzen Markdown ab und zu und zwei regelmäßig. Nur eine Person kannte Markdown nicht. Von JupyterLab hatten alle Teilnehmer schon mal etwas gehört, wobei einer es ab und zu benutzt und zwei es regelmäßig benutzen.

Die allgemeine Bewertung des Prototypen ist positiv ausgefallen. Die meisten Nutzer fanden die Bedienung sowohl benutzerfreundlich als auch intuitiv. Die durchschnittliche Bewertung beider Fragen zu diesem Thema lag bei vier von fünf Punkten, wobei der größte Abstand in beider Richtung ein Punkt war. Die kurze Einführung, um das Konzept zu verstehen, haben alle Teilnehmer positiv bewertet, also als ausreichend zum Verständnis der Software angesehen. Vier von sechs Teilnehmer gaben an, dass sie die Software für den Vorlesungsbetrieb nutzen würden. Einer der anderen Teilnehmer hat angegeben, dass er sie vielleicht für die richtigen Themen nutzen würde, er mache es abhängig vom Vorbereitungsaufwand und den zur Verfügung stehenden Tools. Das ist ein Argument dafür, dass das Plugin-System möglichst einfach sein muss, damit es möglichst viele qualitativ hochwertige Plugins geben kann. Ein anderer hat nur mit einem positiven Statement geantwortet, ohne klares Ja oder Nein. Ihm gefällt die direkte Nutzung der Tools ohne getrennten Zugriff.

Ein häufiger und wichtiger Kritikpunkt an der Software war, dass nicht klar genug erkennbar war, wann sich ein Block im Ansichtsmodus oder im Bearbeitungsmodus befindet. Gerade beim Calculator-Block ist der Bearbeitungsmodus nur schwer von dem normalen Modus zu unterscheiden, da hier sich nur die Umrandung ändert und das Speichersymbol erscheint. Dieses Problem könnte gelöst werden, indem der Bearbeitungsmodus noch deutlicher erkennbar gemacht wird. Ein Teilnehmer schlägt beispielsweise vor, den Bearbeitungsmodus farbig zu hinterlegen. Ein anderer Ansatz, der von einem der Teilnehmer bevorzugt werden würde, wäre es, die Bearbeitungsansicht und die normale Ansicht klarer zu trennen. Es könnten in der Bearbeitungsansicht ausschließlich die Blöcke im Bearbeitungsmodus gezeigt werden. Um das Ergebnis zu sehen, könnte die normale Ansicht in einem Splitt-Screen angezeigt werden. Ein Argument für den blockweisen Moduswechsel ist, dass Änderungen nach dem Speichern direkt an Ort und Stelle sichtbar sind, wodurch Fehler eventuell leichter gefunden werden. Zudem ist die Ansicht so näher an der der Studenten. Außerdem zeigt der Wechsel zwischen den Modi an, welche Änderungen bereits gespeichert sind. Ein Kompromiss wäre es, beide Möglichkeiten zu implementieren. So könnten Nutzer selbst entscheiden, welchen Modus sie bevorzugen.

Die Art und Weise, wie Blöcke hinzugefügt werden, wurde auch oft kritisiert. Blöcke werden über den Plus-Knopf links neben einem Block hinzugefügt. Dies wurde nicht als intuitiv empfunden, weil die Aktion nicht auf dem ausgewählten Block stattfindet, aber diesen Eindruck erweckt. Einer der Teilnehmer scheint nicht verstanden zu haben, dass ein neuer Block erzeugt wird, wenn auf das Plus geklickt wird. Dies ist auch ein Indiz dafür, dass dieser Knopf nicht eindeutig genug ist. Eine Lösung könnte es sein, zwischen den Blöcken einen Knopf für das Hinzufügen von Blöcken hinzuzufügen. Ein alternativer Lösungsansatz ist es, das Symbol so zu ändern, dass es eindeutiger anzeigt, dass der Block unter dem momentan selektierten hinzugefügt wird.

Mehrere Teilnehmer haben vorgeschlagen, dass ein Block direkt gespeichert werden könnte, wenn er verlassen wird bzw. "rausgeklickt" wird. Diese Änderung könnte stören, da Änderungen so eventuell ungewollt gespeichert werden. Wird ein Block gespeichert, wird der Bearbeitungsmodus verlassen. Wird nun der Bearbeitungsmodus beim Verlassen eines Blocks direkt geschlossen, könnte Block-übergreifendes Arbeiten erschwert werden.

Einer der Teilnehmer, der die Software schon kannte, hat sich explizit vorgenommen, die Tastenkombinationen auszuprobieren. Hier wurde ähnliches kritisiert. Eine Tastenkombination könnte die aktuelle Zelle direkt speichern und eine neue erstellen. Der Teilnehmer findet, dass zu viele Tastenkombinationen gedrückt werden müssen, um eine Aufgabe auszuführen.

Ein Teilnehmer kritisierte, dass Blöcke unsichtbar sein können. Ein Markdown-Block, der erstellt wurde, wird in Blau umrandet. Wird nun etwas anderes selektiert, ohne etwas vorher zu schreiben, ist dieser Block quasi unsichtbar, da dieser keinen Inhalt hat. Dieser Umstand kann wahrscheinlich verbessert werden, indem ein neuerstellter Block automatisch in den Bearbeitungsmodus geschaltet wird, das ist auch ein weiterer Verbesserungsvorschlag, der von einem Nutzer gemacht wurde. So ist dieser nicht unsichtbar und der Benutzer könnte direkt anfangen zu schreiben. Dieser Lösungsansatz stellt somit eine zusätzliche Verbesserung dar.

Ebenfalls wurde bei den leeren Markdown-Blöcken kritisiert, dass diese nicht immer durch doppeltes Klicken bearbeitbar sind. Dieser Fehler wird durch ein technisches Detail verursacht und kann behoben werden. Das Doppelklick-Event wird auf dem Container des Inhalts und nicht auf dem Block selbst registriert, das sorgt dafür, dass ohne einen Inhalt kein Doppelklick-Event ausgeführt wird.

Für den Markdown-Block wurde ein Markdown-Cheat-Sheet vorgeschlagen. Für Nutzer, die die Markdown Syntax nicht kennen, wäre dies sinnvoll. Alternativ könnte, inspiriert vom Markdown-Editor "editor.md"[1], ein Menü eingeführt werden, das ausgewählte Texte automatisch mit den richtigen Markdown-Befehlen formatiert.

Auch wurde das Fehlen von Bestätigungsabfragen angesprochen. Das Löschen von Blöcken sollte mit einer Bestätigungsabfrage versehen werden. So wird vermieden, dass Benutzer aus Versehen Blöcke löschen. Ebenso sollte es eine Bestätigungsabfrage geben, wenn für einen ungespeicherten Block die Änderungen verloren gehen würden.

Zum Wofa-Tool wurde explizit das Feedback gegeben, dass das Angeben eines Templates nach dem Abspeichern keinen Effekt hat. Das Template ist hier die Vorlage für

[1] editor.md: https://pandao.github.io/editor.md/en.html

einen Automat in textueller Form, der angezeigt wird, wenn der Benutzer noch keine Lösung eingegeben hat. Vermutlich hat der Anwender hier bereits eine Lösung gespeichert, dann wird nicht mehr das Template angezeigt, sondern die gespeicherte Lösung. Das Problem ist, dass der Unterschied zwischen Template und gespeicherter Lösung im normalen Modus nicht erkennbar ist. Ist noch keine Eingabe gespeichert worden, könnte der Text farbig anders dargestellt werden, damit der Unterschied klarer wird. Zusätzlich könnte eine Funktion helfen, die die Benutzerdaten für einen Block in der Bearbeitungsansicht zurücksetzt.

Weitere kleinere Änderungsvorschläge und aufgetretene Probleme sind:

- Das Icon zum Verschieben wurde nicht als passend empfunden.
- Ein Teilnehmer bewertet negativ, dass die Kontrollleiste für einen Block erst auftaucht, wenn die Maus auf der Höhe des Blocks ist.
- Das X-Symbol für das Einklappen des Menüs wurde als nicht intuitiv empfunden.
- Unten auf der Seite gibt es einen Knopf, um auf die nächste Seite zu gelangen. Hier könnte auch ein Knopf sein, der zur vorherigen Seite führt.
- Wenn die Website ohne Seiten-ID aufgerufen wird, wird die Web-App nicht vollständig geladen, da keine Seite geladen wird.
- Der Name des Buches ist im Moment noch nicht änderbar.
- Im WoFA-Block ist der "Speichern und Bewerten"-Knopf ohne direktes Feedback. Es gibt keinen direkten visuellen Effekt, wenn er geklickt wird.
- Zu Beginn der Studie, hat eine Frontend-Instanz erst nach neuem Laden der Seite vollständige funktioniert. Der Grund dafür ist unbekannt.

5.3 Weitere Limitierungen

Zusätzlich sollen hier weitere Probleme diskutiert werden, die außerhalb des Nutzertests entdeckt wurden.

Im Moment wird die Seite durch einen einfachen Login-Bereich geschützt. Loggt sich jemand nun mit einem normalen Account in die Bearbeitungsansicht ein, ist das im Prototypen tatsächlich erstmal möglich. Es nützt den Benutzer aber nichts, weil die sicherheitskritischen Funktionen nicht funktionieren. Es wird das Frontend geladen, aber das Backend verweigert die sicherheitskritischen Funktionen. Im Moment gibt es im Backend keine Möglichkeit den Status eines Accounts abzufragen, also wurde das Problem vorest nicht behoben.

In einer fertigen Software sollte des weiteren darauf geachtet werden, dass Fehlermeldungen auf verständliche Weise an der Benutzer weitergegeben werden. Im Prototypen werden viele Fehler nicht in der Benutzeroberfläche angezeigt. Im Plugin-System ist das auch ein Problem. Auf technischer Ebene gibt es für Iframe-Blöcke noch keine

Möglichkeiten, Fehler zu behandeln, die auf der Seite des Teaching-Books geschehen, das muss in das Protokoll der Bibliothek noch hinzugefügt werden.

Aus technischer Sicht wurden zudem viele Funktionalitäten nicht optimiert. In den Modulen für das Datenmodell auf Seiten des Frontends gibt es Optimierungspotential. Wird beispielsweise ein Block verschoben, könnte diese Funktion im Frontend gespiegelt werden, sodass die Daten der Seite nicht neu von Server geladen werden müssen. Ähnliche Optimierungen könnten beispielsweise für das Erstellen eines neuen Blocks vorgenommen werden.

Ein weiteres technisches Problem ist ein Sicherheitsproblem mit den verschiedenen Datenbereichen der Blöcke. Hier gibt es zu wenig Datenbereiche. Es gibt beispielsweise keinen zugriffsbeschränkten Datenbereich für einzelne Nutzer. Dieser wäre aber für die Benotung wichtig. Sonst könnte ein Student seine Noten ändern, indem er einfach eine solche Anfrage an den Server stellt. Dieses Problem könnte durch das Hinzufügen weiterer Datenbereiche gelöst werden.[13]

Es gibt 3 Sichtbarkeitsbereiche und 3 Zugriffsbeschränkungen. Also werden maximal 5 weitere benötigt, um alle Kombinationen abzudecken. Alternativ könnten für Felder Zugriffsbeschränkung und Datenbereich getrennt festgelegt werden.

Da die Architektur im Moment nur auf Http-Anfragen setzt, also das Frontend nur Änderungen erhält, wenn es danach fragt, kann es zu Problemen kommen, wenn mehrere Nutzer gleichzeitig ein Vorlesungsskript bearbeiten. Ein Nutzer sieht beispielsweise eine neu erstellte Seite nicht, bis er die Seite neu lädt oder eine Aktion ausführt, welche dafür sorgt, dass das Datenmodell neu vom Server geladen wird. Wenn zwei Nutzer dieselbe Seite bearbeiten, können ähnliche Probleme auftauchen. Die Architektur könnte so erweitert werden, dass zusätzlich eine bestehende Verbindung von Frontend zum Backend aufgebaut wird, damit das Frontend über Änderungen benachrichtigt werden kann.

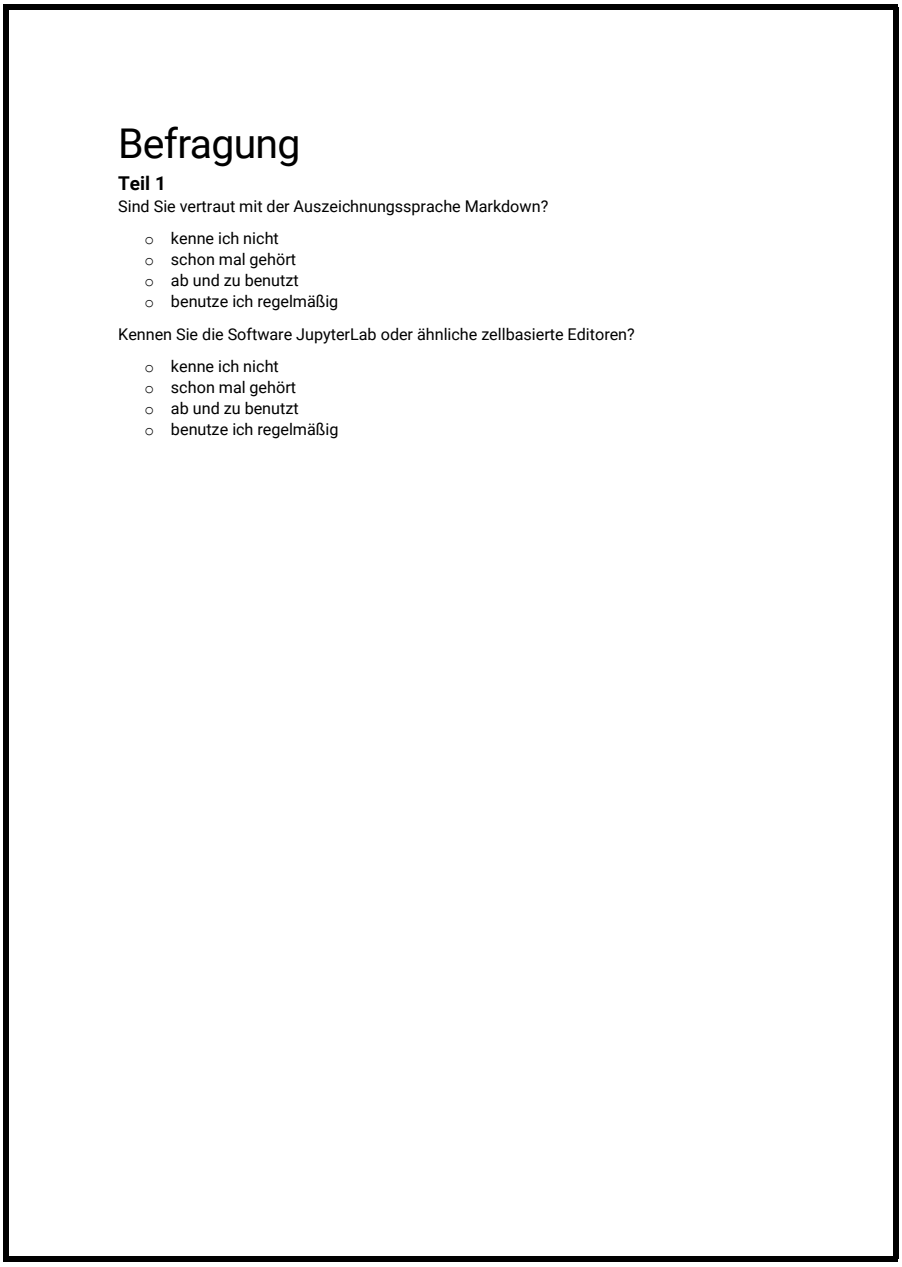
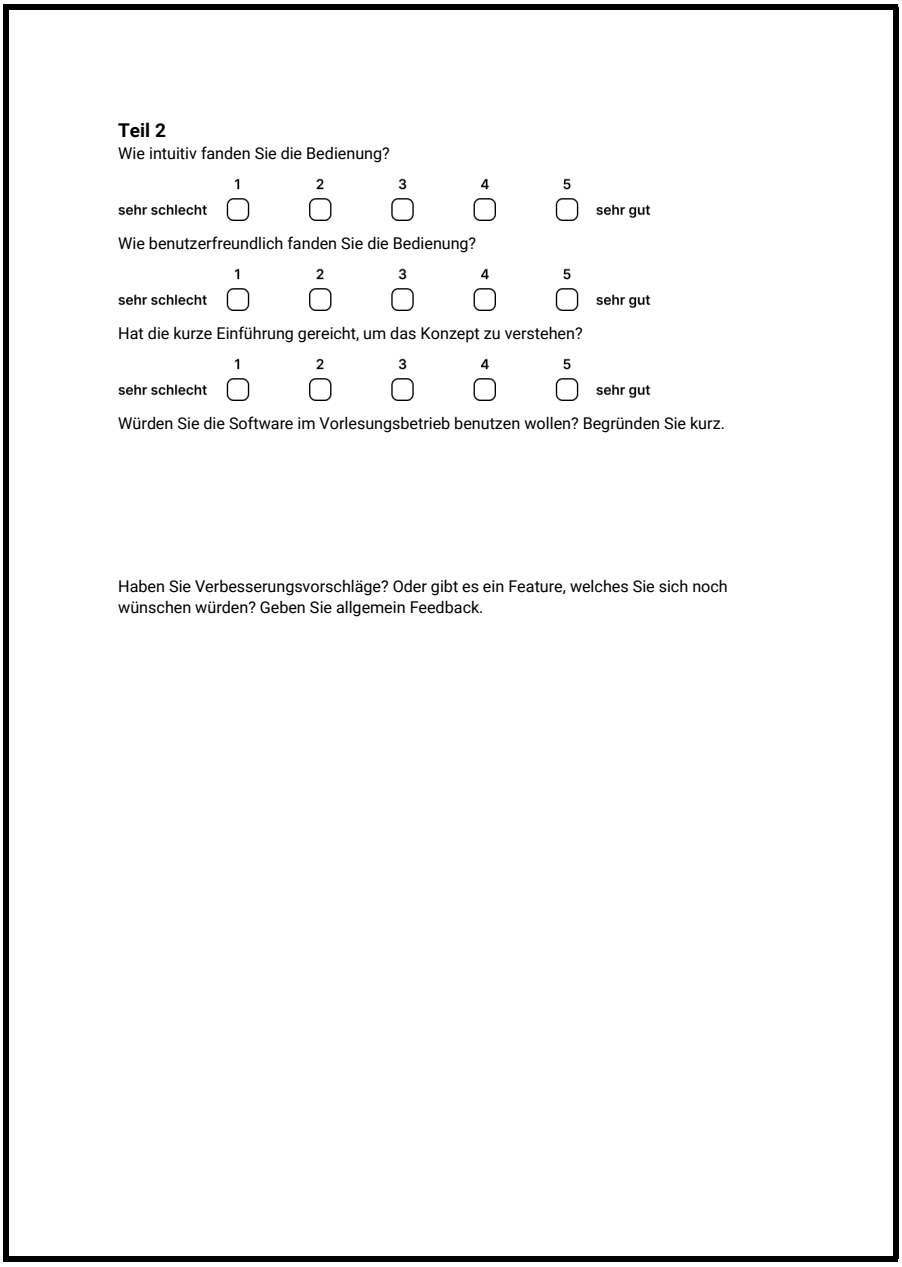


Abbildung 5.1: Befragung Teil 1



Teil 2

Wie intuitiv fanden Sie die Bedienung?

	1	2	3	4	5	
sehr schlecht	☐	☐	☐	☐	☐	**sehr gut**

Wie benutzerfreundlich fanden Sie die Bedienung?

	1	2	3	4	5	
sehr schlecht	☐	☐	☐	☐	☐	**sehr gut**

Hat die kurze Einführung gereicht, um das Konzept zu verstehen?

	1	2	3	4	5	
sehr schlecht	☐	☐	☐	☐	☐	**sehr gut**

Würden Sie die Software im Vorlesungsbetrieb benutzen wollen? Begründen Sie kurz.

Haben Sie Verbesserungsvorschläge? Oder gibt es ein Feature, welches Sie sich noch wünschen würden? Geben Sie allgemein Feedback.

Abbildung 5.2: Befragung Teil 1

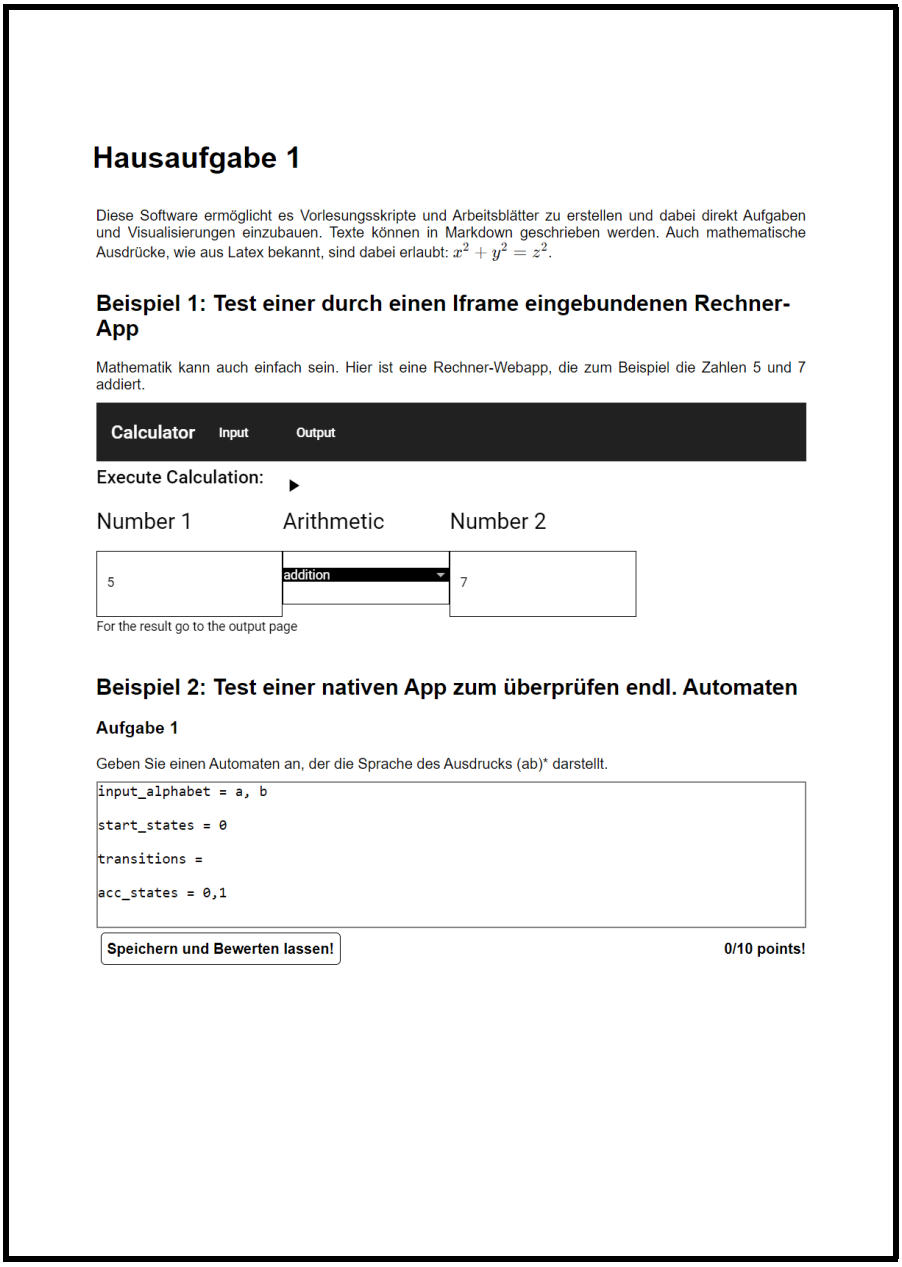


Abbildung 5.3: Vorlage

6 Konklusion & Ausblick

Um zu zeigen, wie moderne Anforderungen an Vorlesungsskripte und Arbeitsblätter umgesetzt werden können, hat der Prototyp verschiedene Konzepte kombiniert und diese für das Ziel, moderne Lehrmaterialien zu erstellen, ausgerichtet. Es handelt sich bei der entstandenen Software um einen Prototyp, die Software sollte noch überarbeitet werden, bevor sie im Vorlesungsbetrieb benutzt wird. Bei einer Nutzerstudie ist der Prototyp allgemein positiv aufgefasst wurden. Trotzdem wurden, unter anderem bei dieser Nutzerstudie, einige Probleme und Limitierungen festgestellt, die aber bei einem Prototyp zu erwarten sind. Für den Erfolg der Software müssen außerdem noch viele Tools integriert und entwickelt werden. Durch die verschiedenen Implementierungsmöglichkeiten wurde gezeigt, wie ein flexibles Plugin-System aussehen kann, um verschiedenste Ansprüchen gerecht zu werden, damit der Prozess neue Plugins zu entwickeln unterstützt wird und bereits bestehende Software leicht integrierbar ist. So ist es realistisch, dass in kurzer Zeit eine große Anzahl an Plugins entsteht.

Das Ziel in der Zukunft ist es, den Prototypen zu einer für den Einsatz fertigen Web-App zu entwickeln, die viele nützliche Features anbietet. Dafür kann wie folgt vorgegangen werden: zuerst sollten dafür die angesprochenen Probleme gelöst werden. Danach könnten die Tools, die bereits entwickelt wurden oder im Moment entwickelt werden, als Plugin für die App integriert werden. Dabei sollte versucht werden, das Plugin-System durch weitere sinnvolle Features zu erweitern. Eine extra Benutzeroberfläche für den Registrierungsprozess hätte beispielsweise das Potential, diesen Prozess zu erleichtern. Eines der Plugins, die momentan implementiert werden, ist ein Block, in dem Python-Code geschrieben und ausgeführt werden kann, ein anderes ist ein visueller Editor für endliche Automaten. Andere Kandidaten sind die in der Motivation erwähnten Anwendungen. Nun könnte einmal ein Vorlesungsskript in dieser Software implementiert werden. Dieser Prozess würde einen erweiterten Test darstellen. Während dessen kann festgestellt werden, welche der weiteren Features am wichtigsten sind, welche weiteren hilfreich wären und welche weiteren Probleme gelöst werden müssen. Zuerst wäre es ratsam, sich nur auf die Vorlesungsskripte zu fokussieren. Andere Aspekte erst danach umzusetzen hat den Vorteil, dass die Grundlage solide ist. Für Übungsblätter, die benotet werden und zum Erhalt der Studienleistung wichtig sind, müssen außerdem erst noch weitere Features entwickelt werden. Es muss erst noch ein Abgabesystem implementiert werden. Zudem muss für Abgaben eine zeitliche Beschränkung möglich sein. Für komplexe Aufgaben, für die es keine Online-Eingabe gibt, sollte direkt eine Abgabe über das Hochladen von PDFs integriert werden, um den Wechsel zu anderen Plattformen zu vermeiden.

Es gibt viele weitere zukünftige Features, die geplant sind und weitere Ideen, die während der Entwicklung entstanden sind. Einige dieser Erweiterungsmöglichkeiten waren Teil der Anforderungen, es musste darauf geachtet werden, dass diese mit maß-

vollen Aufwand umgesetzt werden können.

Eine dieser Erweiterungen soll sein, die Option einzuführen, Verknüpfungen zwischen Blöcken herzustellen. Blöcke sollen voneinander abhängen können. Ein Block, der ein Diagramm darstellt, wäre dadurch beispielsweise in der Lage, auf die Lösung einer Aufgabe eines anderen Blocks zu basieren. Für diese Implementierung ist es hilfreich, dass wir in unserem Plugin-System Daten zentralisiert vorliegen haben. Um das Feature zu implementieren, muss nun ein Block imstande sein, auf die Änderung der Daten eines anderen zu reagieren. Diese Verbindungen müssen dabei zur Laufzeit bearbeitbar sein. Dafür könnte das Prinzip von "Signal and Slots"[1] ähnlich wie in dem Framework QT[2] umgesetzt werden.

Eine weitere geplante Funktion ist der PDF-Export. Durch dieses Feature entsteht die Möglichkeit, Skripte und Übungsblätter in einem üblichen Format zu speichern oder auszudrucken. Das Tool Pandoc wird zum Umwandeln von Markdown zu HTML genutzt. Das gleiche Tool bietet aber mittels LaTeX das Umwandeln in PDF an. Es muss nur noch festgelegt werden, wie die interaktiven Blöcke behandelt werden.

Pandoc macht es auch möglich LaTeX nach HTML zu konvertieren. LaTeX könnte somit als eine alternative Möglichkeit angeboten werden um Texte zu schreiben. Dieses Feature würde noch mehr Möglichkeiten eröffnen und für LaTeX-Nutzer den Einstieg erleichtern.

Ein untergeordnetes Ziel ist es, dass durch das Projekt ein einheitliches Auftreten entsteht. Hierfür könnten eine Reihe von Spezifikationen mit CSS-Dateien, Icons und Schriftarten erstellt werden. Es sollte darauf geachtet werden, diese Spezifikationen einfach zu halten, damit der Prozess, ein neues Plugin zu schreiben, nicht zu komplex wird.

Durch die Zentralisierung von Daten können die Daten auf verschiedene Weise einfach für hilfreiche weitere Projekte genutzt werden. Die Daten können beispielsweise zur Lerndiagnostik genutzt werden.

Um Interaktionen mit anderen Studierenden und Dozenten zu erlauben, könnte direkt im Skript die Möglichkeit für Feedback, Fragen und Kommentaren eingebaut werden. Ein Kommentar oder eine Frage könnte, ähnlich wie in Google Docs[3], direkt auf eine Aufgabe oder Stelle im Vorlesungsskript verweisen. Für die Studenten könnten auch Tools eingebaut werden, mit denen sie direkt im Vorlesungsskript die Möglichkeit haben, Notizen zu machen oder Texte zu markieren.

Mit Vertrauen in eine erfolgreiche Umsetzung, kann auf einen zielführenden Einsatz der Software gehofft werden. Sie hat das Potential, den Vorlesungsbetrieb sowohl für die Studenten als auch für die Dozenten, nachhaltig zu bereichern. Außerdem hat das Projekt für zukünftige Projekte Türen geöffnet, auf die wir ebenfalls gespannt sein dürfen.

[1] QT-Framework "Signal and Slots": https://doc.qt.io/qt-6/signalsandslots.html
[2] QT-Framework https://doc.qt.io/qt-6/
[3] Google Docs: https://docs.google.com/

Literatur

[1] K. L. Vavra, V. Janjic-Watrich, K. Loerke, L. M. Phillips, S. P. Norris und J. Macnab, "Visualization in science education," *Alberta Science Education Journal*, Jg. 41, Nr. 1, S. 22–30, 2011.

[2] S. Simonak, "Using algorithm visualizations in computer science education," *Open Computer Science*, Jg. 4, Okt. 2014.

[3] A. Butler, "Repeated Testing Produces Superior Transfer of Learning Relative to Repeated Studying," *Journal of experimental psychology. Learning, memory, and cognition*, Jg. 36, S. 1118–33, Sep. 2010.

[4] J. D. Karpicke und J. R. Blunt, "Retrieval Practice Produces More Learning than Elaborative Studying with Concept Mapping," *Science*, Jg. 331, Nr. 6018, S. 772–775, 2011.

[5] J. D. Karpicke, A. C. Butler und H. L. R. III, "Metacognitive strategies in student learning: Do students practise retrieval when they study on their own?" *Memory*, Jg. 17, Nr. 4, S. 471–479, 2009, PMID: 19358016.

[6] J. J. McConnell, "Active Learning and Its Use in Computer Science," in *Proceedings of the 1st Conference on Integrating Technology into Computer Science Education*, Ser. ITiCSE '96, Barcelona, Spain: Association for Computing Machinery, 1996, 52–54, ISBN: 0897918444.

[7] J. Müller, *Eine Webanwendung zur prädikatenlogischen Modellprüfung von Graphstrukturen*, 2021. [Online]. Verfügbar: https://www.uni-kassel.de/eecs/index.php?eID=dumpFile&t=f&f=38784&token=b07f12442b09ff26b73968fb6cb7f3709249764a (besucht am 14. 02. 2023).

[8] K. Lehmann, *Toolgestütztes Lernen im Kontext der eines Reduktions-Trainers für Studierende: ein Backend-Prototyp zur automatisierten Überprüfung von Reduktionen*, 2022. [Online]. Verfügbar: https://www.uni-kassel.de/eecs/index.php?eID=dumpFile&t=f&f=38780&token=6d28d506f26579d3873af82b5a7e4280d51b39db (besucht am 14. 02. 2023).

[9] C. Weiße, *Frontend-Entwicklung eines Reduktions-Trainers für Studierende*, 2023. [Online]. Verfügbar: https://www.uni-kassel.de/eecs/index.php?eID=dumpFile&t=f&f=38802&token=040b7fb95573a22d9c93256201ceecfcc3a4a40b (besucht am 14. 02. 2023).

[10] N. Hundeshagen, M. Lange und G. Siebert, "DiMo - Discrete Modelling Using Propositional Logic," in *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, C. Li und F. Manyà, Hrsg., Ser. Lecture Notes in Computer Science, Bd. 12831, Springer, 2021, S. 242–250.

[11] M. Herwig, N. Hundeshagen, J. Hundhausen, S. Kablowski und M. Lange, *Problem-Specific Visual Feedback in Discrete Modelling*, eingereicht zur Veröffentlichung in der ITiCSE 2023, 2023.

[12] F. Bruse, M. Herwig und M. Lange, "A Similarity Measure for Formal Languages Based on Convergent Geometric Series," in *Implementation and Application of Automata*, P. Caron und L. Mignot, Hrsg., Cham: Springer International Publishing, 2022, S. 80–92, ISBN: 978-3-031-07469-1.

[13] M. Herwig, *Teaching-Book-Backend*, Master Projekt im Fachgebiet Theoretische Informatik / Formale Methoden an der Universität Kassel, Teil des Teachingbook-Repository, 2023.

[14] "Notion Reviews & Product Details." (2023), [Online]. Verfügbar: https://www.g2.com/products/notion/reviews (besucht am 27. 02. 2023).

[15] T. Reiners, S. Voß, D. Reiß und H. Schulze, "SmartFrame: An Integrated Environment for XML-Coded Learning Material," in *Wirtschaftsinformatik 2003/Band I*, W. Uhr, W. Esswein und E. Schoop, Hrsg., Heidelberg: Physica-Verlag HD, 2003, S. 613–632, ISBN: 978-3-642-57444-3.

[16] F. Freiberger, "PseuCo Book: An Interactive Learning Experience," in *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, Ser. ITiCSE '22, Dublin, Ireland: Association for Computing Machinery, 2022, 414–420, ISBN: 9781450392013.

[17] J. Kľuka, J. Šiška, M. Baluch u. a., *First-Order Logic Workbook*, Präsentation auf der FOMEO 2022, 2022. [Online]. Verfügbar: https://easychair.org/smart-program/FLoC2022/FOMEO-2022-07-31.html#talk:197288 (besucht am 17. 02. 2023).

[18] R. Harris. "Svelte 3: Rethinking reactivity." (2019), [Online]. Verfügbar: https://svelte.dev/blog/svelte-3-rethinking-reactivity (besucht am 11. 02. 2023).

[19] Stackoverflow. "Web frameworks and technologies - Loved vs. Dreaded." (2021), [Online]. Verfügbar: https://insights.stackoverflow.com/survey/2021#most-loved-dreaded-and-wanted-webframe-love-dread (besucht am 08. 02. 2023).

[20] Stackoverflow. "Web frameworks and technologies - Loved vs. Dreaded." (2023), [Online]. Verfügbar: https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-webframe-love-dread.

[21] "Announcing SvelteKit 1.0." (2022), [Online]. Verfügbar: https://svelte.dev/blog/announcing-sveltekit-1.0 (besucht am 10. 02. 2023).

[22] "Glossary." (2022), [Online]. Verfügbar: https://kit.svelte.dev/docs/glossary#ssr (besucht am 10. 02. 2023).

[23] "Svelte - Documentation." (2023), [Online]. Verfügbar: https://svelte.dev/docs (besucht am 11. 02. 2023).

[24] "Svelte - Tutorial - Stores/Writable stores." (2023), [Online]. Verfügbar: https://svelte.dev/tutorial/writable-stores (besucht am 11. 02. 2023).

[25] M. W. Docs. "Window.postMessage()." (2023), [Online]. Verfügbar: https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage (besucht am 10. 02. 2023).

[26] M. W. Docs. "iframe: The Inline Frame element." (2023), [Online]. Verfügbar: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/iframe?retiredLocale=de (besucht am 10. 02. 2023).