BACHELORARBEIT

CONTENTANBIETER-DASHBOARD - ENTWICKLUNG EINER PLATTFORM ZUR INTEGRATION UND VERWALTUNG VON LERNINHALTEN DER HPI SCHUL-CLOUD

CONTENT PROVIDER DASHBOARD - DEVELOPMENT OF A PLATFORM FOR INTEGRATING AND MANAGING LEARNING CONTENT OF THE HPI SCHUL-CLOUD

Katharina Blaß und Adrian Jost

CHAIR

Internet Technologies and Systems

SUPERVISORS

Prof. Dr. Christoph Meinel

Alexander Kremer

Arne Oberländer

29. Juli 2019

Katharina Blaß und Adrian Jost: *Contentanbieter-Dashboard - Entwicklung einer Plattform zur Integration und Verwaltung von Lerninhalten der HPI Schul-Cloud,* Bachelorarbeit, © 29. Juli 2019

ABSTRACT

Der Lern-Store der HPI Schul-Cloud soll einen zentralen Zugang zu digitalen Bildungsinhalten für Schüler und Lehrer schaffen. Dafür ist es notwendig, den Lern-Store mit vielfältigen und qualitativ hochwertigen Materialien zu füllen. Ziel dieser Arbeit ist daher, eine Plattform zu entwickeln, über die Inhalteanbieter ihre digitalen Lernmaterialien eigenständig in den Store einpflegen und auch verwalten können. Die größte Herausforderung besteht darin, sowohl kleine als auch etablierte Inhalteanbieter zu unterstützen und unter Berücksichtigung ihrer technischen Möglichkeiten entsprechende Schnittstellen bereitzustellen. Dazu zählen vor allem die Funktionalitäten, Inhalte direkt über die Schul-Cloud zu hosten oder in größeren Mengen in den Lern-Store zu importieren. Zusätzlich werden notwendige Verwaltungsoptionen wie das Filtern und nachträgliche Bearbeiten der Metadaten sowie geeignete Authentifizierungsmechanismen umgesetzt. Mithilfe dieser Arbeit soll der erste Schritt in Richtung eines vielfältigen Lern-Stores getan werden.

INHALTSVERZEICHNIS

1

EINLEITUNG

Adrian Jost

1.1 DIE HPI SCHUL-CLOUD UND DER LERN-STORE

2019 hat die Bundesregierung den DigitalPakt Schule beschlossen mit dem Ziel die Digitalisierung an deutschen Schulen weiter voranzutreiben. Dieser soll Schulen die finanziellen Mittel bereitstellen, um eine moderne technische Infrastruktur aufzubauen.[42]
Auch die HPI Schul-Cloud[1], ein Projekt, welches 2016 vom Hasso-Plattner-Institut und MINT-EC gegründet wurde, hat das Ziel, die voranschreitende Digitalisierung des deutschen Unterrichtsalltages zu unterstützen. Mit verschiedenen Tools wie einen Unterrichtseditor versucht das, vom Bundesministerium für Bildung und Forschung unterstützte Projekt, eine moderne Lern- und Lehrinfrastruktur anzubieten.
Ein Teil der Schul-Cloud ist der Lern-Store. Dieser soll Schülern und Lehrern einen einheitlichen, zentralen Zugang zu vielfältigen Lernmaterialien ermöglichen. Die Inhalte können anschließend in die Unterrichtsgestaltung der Lehrer integriert werden und Schülern als Lernquelle dienen.[27]

1.2 ZIELSETZUNG

Essentieller Bestandteil eines solchen Lern-Stores ist die Verfügbarkeit von qualitativ hochwertigen Lernmaterialien. Aktuell gibt es jedoch keine einheitliche Schnittstelle für Inhalteanbieter, über die sie ihre Inhalte einpflegen und verwalten können. Diese Arbeit soll eine Lösung für dieses Problem entwickeln.
Bereits vor 2 Jahren wurde ein Prototyp einer solchen Plattform entwickelt. Dieser unterstützt jedoch nur rudimentäre Funktionen der Inhalteverwaltung und wurde daher nie produktiv eingesetzt. Im Rahmen dieser Bachelorarbeit soll der Prototyp zu einer Plattform erweitert werden, mit welchem Inhalteanbieter jeder Größe ihre Inhalte über den Lern-Store vertreiben können. Dabei sollen sowohl kleine,

1 https://schul-cloud.org

unabhängige als auch große, etablierte Inhalteanbieter berücksichtigt werden. Nur so kann eine vielfältige Mischung aus innovativen Inhalten geschaffen werden.

1.3 AUFBAU DER ARBEIT

Zu Beginn der Arbeit wird eine Anforderungsanalyse durchgeführt, welche die Anforderungen an das Content-Dashboard klar definiert. Anschließend erfolgt eine Analyse von bereits etablierter Software zur Inhalteverwaltung mit dem Ziel, diese als Basis des Lern-Stores zu nutzen.
In den folgenden Kapiteln soll die Umsetzung der einzelnen, in der Anforderungsanalyse definierten Funktionalitäten thematisiert werden. Die dabei auftretenden Herausforderungen werden analysiert und verschiedene Lösungsmöglichkeiten evaluiert. Auf dieser Grundlage werden fundierte Entscheidungen zur Lösung der Probleme getroffen und deren Implementierung erörtert.
Zudem soll die erfolgreiche Umsetzung der Anforderungen durch mehrere Tests validiert werden.
Abschließend wird das Ergebnis der Arbeit zusammengefasst und ein Ausblick für weitere Forschungsmöglichkeiten und Verbesserungspotential gegeben.

1.4 TECHNISCHES UMFELD

Die Arbeit baut auf einer bestehenden Codebasis[2] auf. Dabei wird im Backend eine MongoDB-Datenbank[3] in Kombination mit dem Node.JS Framework FeathersJS[4] und einer ElasticSearch Volltextsuche[5] verwendet. Im weiteren Verlauf wird auf dieses System als `Content-Server` verwiesen. Zusätzlich gibt es bei der Schul-Cloud einen weiteren Server, welcher ähnliche Technologien verwendet. Dieser wird als `schulcloud-server` bezeichnet und bildet den Kern der Schul-Cloud.

Der `schulcloud-client` bildet das Frontend der Schul-Cloud. Teile dieser Oberfläche werden aktuell in das Framework Vue.js[6] migriert. Um eine zukünftige Interoperabilität der Systeme zu vereinfachen, soll auch das Frontend des Content-Dashboards auf Vue.js aufbauen.

2 https://github.com/schul-cloud/schulcloud-content
3 https://www.mongodb.com/de
4 https://feathersjs.com/
5 https://www.elastic.co/de/
6 https://vuejs.org/

2

ANFORDERUNGSANALYSE

Katharina Blaß

Zu Beginn der Arbeit ist es wichtig, Anforderungen zu sammeln, zu analysieren und für die folgende Implementierung festzulegen. Dieser Schritt ist elementar, um ein Produkt zu entwickeln, das den Bedürfnissen der späteren Anwender, den Inhalteanbietern, entspricht. Hierbei ist eine Unterscheidung in funktionale und nicht-funktionale Anforderungen vorzunehmen.

2.1 FUNKTIONALE ANFORDERUNGEN

Für die Definition der funktionalen Anforderungen ist es notwendig, die bestehenden Inhalteanbieter im Schul-Cloud Lern-Store sowie das bisherige Verfahren zum Einpflegen ihrer Inhalte zu analysieren. Weiterhin gilt es die beteiligten Parteien nach ihren Erfahrungen und Wünschen zu befragen. Diese Analyse ergibt deutlich, dass sich die Bedürfnisse der vorhandenen Inhalteanbieter aufgrund ihrer unterschiedlichen Größe und technischen Möglichkeiten sehr stark unterscheiden.
Eine weitere wichtige Erkenntnis der Analyse ist, dass Inhalte bislang auf unterschiedlichsten Wegen in den Lern-Store gelangen. Einige werden über Crawler eingepflegt, andere über den Prototypen des Content-Dashboards, und wieder andere aufwendig von Hand über die hinter dem Lern-Store liegende Ressourcen-Datenbank.
Im Folgenden gilt es, die benötigten Funktionalitäten zu bestimmen, mit deren Hilfe das Content-Dashboard zur einzigen Schnittstelle zum Hinzufügen neuer Inhalte in den Lern-Store ausgebaut werden kann. Jedem Inhalteanbieter soll unter Berücksichtigung seiner Möglichkeiten eine Funktionalität zum Einpflegen seiner Inhalte bereitstellt werden.

Datei-Hosting
Viele der kleineren Inhalteanbieter im Schul-Cloud Lern-Store haben keine Möglichkeit, ihre Lernmaterialien eigenständig zu hosten. Dies hat zur Folge, dass sie ihre Inhalte über Speichermedien wie USB-Sticks an ihre Kunden verteilen. Für den Einsatz dieser Materialien in der Schul-Cloud ist es bislang nötig, dass ein Schul-Cloud Mitarbeiter diese Inhalte aufwendig von Hand in den Lern-Store, bzw.

die dahinterliegende Ressourcen-Datenbank, einpflegt. Eine Verbesserung dieser Situation ist ein Bedürfnis sowohl der Inhalteanbieter als auch der Schul-Cloud Mitarbeiter und Aufgabe des Datei-Hosting Features. Inhalteanbieter sollen mit der Funktionalität die Möglichkeit erhalten, ihre Inhalte selbstständig bei der Schul-Cloud hochzuladen und automatisch vom System in den Lern-Store einpflegen zu lassen. Sie müssen hierfür lediglich die zu einem Inhalt gehörenden Dateien bereitstellen. Diese sollen über die Oberfläche des Content-Dashboard hochgeladen werden können. Ergänzende Informationen zum Inhalt, dessen Metadaten, sollen zusätzlich vom Inhalteanbieter abgefragt werden. Anschließend soll das System die Daten verarbeiten und in der Ressourcen-Datenbank speichern. Danach sind sie für die Nutzer des Lern-Stores abrufbar.

Massenimport
Die großen und etablierten Inhalteanbieter wie Klett oder Cornelsen, haben ihre Inhalte bei sich gehostet und bieten diese auf eigenen Vertriebsplattformen an. Gelingt es, sie von einer Partnerschaft mit der Schul-Cloud zu überzeugen, benötigen sie eine Möglichkeit, ihre vielen Inhalte auf einmal in den Lern-Store einzupflegen. Eine Massenimport-Funktionalität soll ihnen diese Möglichkeit bereitstellen und dabei ihren Wunsch nach einem schnellen Vorgang erfüllen können. Die Inhalteanbieter müssen lediglich ihre Inhalte, gesammelt in einem standardisiertem Dateiformat, bereitstellen. Diese Datei soll in der Oberfläche eingelesen und die enthaltenen Inhalte automatisiert in den Lern-Store integriert werden können.

Massenbearbeitung
Die eingepflegten Inhalte werden bisher regelmäßig von Schul-Cloud Mitarbeitern gewartet und an Veränderungen angepasst. Dabei ist es ihnen sowie den Inhalteanbietern wichtig, dass die Verwaltung und nachträgliche Bearbeitung von Inhalten zukünftig durch die Inhalteanbieter erfolgen kann. Eine entsprechende Massenbearbeitungs-Funktionalität soll daher den Inhalteanbietern die nachträgliche Korrektur von Fehlern beim Einpflegen ihrer Inhalte oder das Ändern von Metadaten erlauben.

Filter
Voraussetzung für die Verwaltung von Inhalten ist das Suchen und Filtern von Inhalten. Während eine Volltextsuche über ElasticSearch bereits von dem Prototypen des Content-Dashboards unterstützt wird, stellt das Filtern eine neue Funktionalität dar. Diese soll es den Inhalteanbietern erlauben, Inhalte zu sortieren oder nach konkreten Werten in ihren Metadaten zu filtern.

Authentifizierung
Neben den für die Nutzer bestimmten Funktionalitäten, sollte das System auch Mechanismen zum Schutz der Nutzer-Identitäten und Inhalte bereitstellen. Dabei soll die Systemgrenze von außen mithilfe geeigneter Authentifizierungsmechanismen gesichert werden. Inhalteanbieter müssen sich vor der Nutzung des Content-Dashboards anmelden. Auch der direkte Zugriff auf die einzelnen Routen im Ba-

ckend soll durch entsprechende Berechtigungsprüfungen gegen unautorisierten Zugriff abgesichert werden.

2.2 NICHT-FUNKTIONALE ANFORDERUNGEN

Neben den Funktionalitäten des Systems soll auch die Art und Weise ihrer Umsetzung in diesem Kapitel festgelegt werden. Diese nicht-funktionalen Anforderungen für das Content-Dashboard umfassen Verständlichkeit, Bedienbarkeit und Performance.

Verständlichkeit
Inhalteanbietern ist eine schnelle und einfache Nutzung des Content-Dashboards wichtig. Die einzelnen Funktionalitäten sollen daher leicht zu verstehen und intuitiv zu bedienen sein. Um dies zu erreichen, werden an dieser Stelle fünf Akzeptanzkriterien einschließlich möglicher Umsetzungen definiert.
Das erste Kriterium ist die Umsetzung einer einheitlichen Oberfläche durch die Verwendung wiederkehrender Basiskomponenten. Häufig verwendetet Elemente, wie beispielsweise Buttons oder Eingabefelder, sollen über die Grenzen der Funktionalitäten hinweg das gleiche Styling erhalten. Auf diese Weise kann der Nutzer auch in unbekannter Umgebung vom gewohnten Aussehen einer Komponente auf ihr Verhalten schließen.
Weiterhin sollen alle verwendeten Buttons selbsterklärend sein, sodass ihre auslösende Aktion im Vorfeld eindeutig erkennbar ist. Hierfür sollen sie mit unterstützenden Icons und Beschriftungen im Hoverstate versehen werden.
Ein weiteres Akzeptanzkriterium besteht darin, nach allen vom Nutzer ausgeführten Aktionen, wie das Speichern eines neuen Inhaltes, Feedback über das Ergebnis dieser Aktion anzuzeigen. Diese Benachrichtigungen sollen in Form von Toasts/Snackbars[1], Bannern[2] oder Dialogen[3] geschehen.
Darüber hinaus soll der Nutzer daran gehindert werden, versehentlich einen Vorgang abzubrechen oder Inhalte zu löschen. Dafür ist es wichtig, die bevorzugte Handlung deutlich hervorzuheben. Hierfür wird jede Aktion in eine primäre oder sekundäre Handlung eingeteilt und mithilfe unterschiedlicher Farben gekennzeichnet. Endgültige, sekundäre Handlungen, wie beispielsweise das Löschen eines Inhaltes, sollen erst nach einer zusätzlichen Bestätigung des Nutzers ausgeführt werden.
Als letztes Akzeptanzkriterium soll die Unterstützung mehrerer Sprachen in der Theorie möglich sein. Deutsch ist dabei die Ausgangssprache. Mithilfe sogenannter Language-Files, sollen die auf der Seite verwendeten Beschriftungen in andere Sprachen übersetzt werden können.
Die Einhaltung der Kriterien und den Erfolg der Verständlichkeits-

1 https://material.io/design/components/snackbars.html
2 https://material.io/design/components/banners.html#banner
3 https://material.io/design/components/dialogs.html#dialogs

Anforderung sollen am Ende des Projektes durch User-Tests überprüft werden.

Bedienbarkeit
Die Anforderung der Bedienbarkeit setzt den Fokus auf die Themen Accessibility und responsive Design. Beides sind wichtige Eigenschaften einer modernen Webanwendung, mit dem Ziel, sie für jedermann und jede Gerätegröße benutzbar zu machen.

Accessibility umfasst Prinzipien und Techniken, mit denen auch situativ oder körperlich eingeschränkte Menschen eine Website und alle ihre Funktionalitäten bedienen können. Die Web Content Accessibility Guidelines (WCAG) definieren hierfür den Standard. Darauf basierend, hat auch Google Richtlinien veröffentlicht, die Entwickler bei der Implementierung Accessibility-tauglicher Websites anleiten sollen. Daran soll sich auch die Entwicklung des Content-Dashboards orientieren.
Nutzer, die eine dauerhafte oder temporäre motorische Beeinträchtigungen haben, können eine Website oftmals nicht auf normalem Wege mit Maus bedienen. Sie benötigen alternative Navigationsmöglichkeiten, wie eine Tastatur- oder Sprachsteuerung. Ersteres ist leicht umzusetzen, indem entweder die Standard HTML Elemente verwendet oder eigene Komponenten um ein Fokus-Style ergänzt werden. Hilfssysteme, wie beispielsweise Screen Reader für Sehbehinderte, benötigen einige zusätzliche Optimierungen der Website. Dazu gehört unter anderem, Eingabefelder mit beschreibenden Labels zu assoziieren und Multimediainhalte wie Bilder um Alternativtexte zu ergänzen, die anstelle des Bildes vorgelesen werden können. Für Menschen mit lediglich leichter Sehschwäche kann das Erlebnis einer Website allein durch ein unterstützendes Styling verbessert werden. Dazu zählt die Verwendung kontrastreicher Farben und Formen sowie ausreichend großer Schriftgrößen[22] [41].
Die erfolgreiche Umsetzung dieser Maßnahmen wird im Test-Kapitel durch einen Accessibility-Test kontrolliert.

Das zweite Akzeptanzkriterium, responsive Design, befasst sich mit der Möglichkeit Websites auch auf mobilen Endgeräten gut les- und bedienbar anzuzeigen[24]. Inhalteanbieter werden für ihre Arbeit am Content-Dashboard zwar kein Handy verwenden, vielleicht aber ein Tablet oder unterschiedlich große Monitore. Das Content-Dashboard soll daher für alle Displaygrößen ab Tablet optimiert werden.

Performance
Performance ist ein wichtiger Aspekt für gute User Experience einer Webanwendung. Besonders für die flüssige Nutzung auf mobilen Geräten spielt sie eine große Rolle. Denn je größer eine Website ist, desto mehr Bandbreite und CPU-Leistung benötigt ein mobiles Endgerät für das Herunterladen und Verarbeiten der Seite. Eine schlechte Performance wird an dieser Stelle nur sehr kurz von den Nutzern der Website toleriert, bevor sie aufhören, die Seite zu verwenden[40].
Aus diesem Gründen sollen sämtliche Lade- oder Verarbeitungsvorgänge im Content-Dashboard mit angemessener Performance ablau-

fen. Um dies zu erreichen, sollte keine Unterseite größer als 1 MB sein oder mehr als 50 Netzwerkanfragen senden. Diese Kennzahlen lassen sich am Ende der Arbeit mithilfe eines Performance-Tests über die Chrome Developer-Tools überprüfen. Dabei ist das Ziel, eine Performancebewertung von mindestens 90/100 auf allen Seiten des Content-Dashboards zu erreichen. Neben dieser messbaren Performance gilt es außerdem, jederzeit die für den Nutzer wahrnehmbare Wartezeit zu optimieren.

3

BESTEHENDE SYSTEME

Adrian Jost

3.1 MOTIVATION

Bevor die Implementierung der Plattform beginnt, sollte geprüft werden ob es bereits Systeme gibt, welche Teile der Anforderungen erfüllen und als Grundlage dienen können. Zentraler Bestandteil des neuen Content-Dashboards soll sein, die notwendigen Dateien von Lerninhalten direkt über die Schul-Cloud bereitstellen zu können. Es gibt bereits einige Anbieter am Markt, welche Software zur Cloud-Dateiverwaltung in Form von Content-Management-Systemen (kurz CMS) bereitstellen. Diese sollen im folgenden bezüglich Ihrer Nutzbarkeit für unseren Anwendungsfall evaluiert werden.

3.2 VERGLEICHSKRITERIEN

Dabei gilt es zunächst einige Vergleichskriterien festzulegen, anhand welcher wir die Systeme bewerten können. Diese sollen die im vorherigen Kapitel beschriebenen Anforderungen an das Content-Dashboard ergänzen.
Von hoher Relevanz sind eine gute Dokumentation und die Anpassbarkeit der jeweiligen Systeme. Nur dann kann die Software auf die speziellen Bedürfnisse der Schul-Cloud zugeschnitten werden. Ebenso wichtig ist daher aus Sicht der Entwickler eine einfache Einrichtung und Handhabung der Systeme.
Abgesehen davon soll das System auch für den Endnutzer leicht zu bedienen sein, weshalb die Benutzeroberfläche der Software ebenfalls teil der Evaluierung ist. Dazu zählt, wie sich die Nutzer bei der Software authentifizieren können und wie sie in die bestehende Infrastruktur zu integrieren ist. Wünschenswert ist hier eine Single-Sign-On[1] Integration in Form von OAuth2, sodass bestehende Schul-Cloud Accounts zur Authentifizierung genutzt werden können.
Wie bereits in der Anforderungsanalyse erwähnt sollte zudem min-

1 https://www.security-insider.de/was-ist-single-sign-on-sso-a-631479/

destens ein Dateiupload und eine Möglichkeit zur Metadatenverwaltung vorhanden oder integrierbar sein.

3.3 VERGLEICH

Für den Vergleich stellen sich die Plattformen EnterMediaDB[2], Edu-Sharing[3] und Nextcloud[4] am vielversprechendsten heraus.

3.3.1 *EnterMediaDB*

EnterMediaDB ist ein auf Java 8u31 basierendes Content-Management-System (CMS). Die Einrichtung der Software ist dank der bereitgestellten Docker Images leicht möglich. Leider mangelt es jedoch an öffentlich zugänglicher Dokumentation zur Anpassung der Software. EnterMediaDB scheint das Produkt hauptsächlich über Kostenintensive Supportpläne zu finanzieren. In Kombination mit der antiquiert wirkenden Oberfläche, welche grundsätzliche Usability Regeln verletzt, der veralteten Softwarebasis (die verwendete Java Version stammt aus 2014) und einem Mangel an Schnittstellen zur Erweiterung ist die Software nicht für unsere Anwendungszwecke geeignet.

3.3.2 *Edu-Sharing*

Edu-Sharing ist eine Cloudplattform zur Verwaltung von Bildungsinhalten. Dabei können Inhalte Gruppiert, Geteilt, aber auch neu erstellt werden. Eine Suche nach diesen Materialien ist ebenfalls möglich. Damit scheint edu-sharing die Ideale Grundlage für einen Lern-Store. Für Entwickler bietet die Plattform allerdings weniger Möglichkeiten.
Das Aufsetzen der Software ist zwar gut möglich, das Einrichten einer Entwicklungsumgebung ist hingegen ein großes Hindernis. Zudem ist der Anteil öffentlich Zugänglicher Dokumentation, zur Entwicklung von Erweiterungen, sehr gering.
Dieses System kann daher ebenfalls nicht als Grundlage für das Content-Dashboard dienen.

3.3.3 *Nextcloud*

Die Plattform Nextcloud setzt den Fokus ihrer Entwicklung klar auf das Verwalten von Dateien. Das System basiert auf der aktuellen PHP 7.3 Umgebung[12] und wird regelmäßig aktualisiert[19]. Besonders

2 https://entermediadb.org/
3 https://edu-sharing.com/
4 https://nextcloud.com

hervorzuheben ist, dass Nextcloud eine sehr gute Plugin Unterstützung bietet. So ziemlich jeder Teil der Benutzeroberfläche lässt sich über Plugins erweitern oder ersetzen. Das UI sieht dabei ansprechend aus und ist sehr gut an das Design der Schul-Cloud anpassbar. Aufgrund all dieser positiven Faktoren entwickeln wir einen ersten Prototypen des Dashboards basierend auf Nextcloud. Dazu wird die Dateiübersicht um eine Metadateninformation erweitert und ein Konzept zur Auslieferung der Dateien entwickelt.
Dabei stellt sich jedoch heraus, dass auch dieses System eine Schwachpunkte hat. Durch den starken Fokus auf Dateien ist es nur schwer möglich eine enge Kopplung zwischen, den von uns benötigten, Metadaten und einer Gruppe an Dateien zu erzeugen und dem Endnutzer verständlich zu präsentieren.
Zudem liefert Nextcloud sämtliche Dateien über eine ID referenziert aus, sodass Dateireferenzen nicht erhalten bleiben. Ein einfaches Hosting von Websites ist so nicht möglich. Stattdessen wäre es notwendig über die API einen zusätzlichen Server anzubinden, welcher die Dateien ausliefert.
Ein weiteres Problem stellt die Nutzerauthentifizierung dar. Nextcloud bietet zwar Single-Sign-On-Schnittstellen an, jedoch muss Nextcloud dabei stets als Master angebunden werden. Dies bedeutet, dass sich Schul-Cloud Nutzer nicht über ihren bestehenden Schul-Cloud Account bei Nextcloud anmelden können. Es gibt zwar ein Plugin welches dieses Problem beheben soll[1], jedoch ist dieses kaum dokumentiert. Aufgrund dieser Komplikationen ist auch Nextcloud für uns keine gute Systemgrundlage und wir entschieden uns gegen die Verwendung.

3.4 ERGEBNIS

Nach dieser Analyse externer Systeme haben wir entschieden, keines der vorgestellten Produkte zu verwenden und stattdessen eine neue Plattform für unseren spezifischen Anwendungsfall zu entwickeln. Dabei sollen die Metadaten im Vordergrund stehen. Diese Analyse schließt allerdings nicht aus, dass wir für einzelne, kleinere Anwendungsfälle nicht trotzdem Bibliotheken verwenden, welche die Entwicklung beschleunigen. Dies muss für die jeweiligen Tools separat evaluiert werden. Zudem kann jederzeit in der späteren Entwicklung auf die hier gesammelten Erkenntnisse zurückgegriffen werden.

4

DATEI HOSTING

Adrian Jost

4.1 MOTIVATION

Nicht jeder Inhalteanbieter hat die Möglichkeiten seine Inhalte im Web verfügbar zu machen.
Ziel der Schul-Cloud ist es, die Inhalte dieser Anbieter ebenfalls im Lern-Store anbieten zu können.
Bisher werden solche, noch nicht im Web verfügbare, Lerninhalte auf manuelle Art und Weise in die Schul-Cloud eingepflegt. Dies bedeutet Konkret, dass die auszuliefernden Daten auf einem Datenmedium zur Schul-Cloud transportiert werden. Anschließend werden sie von einem Schul-Cloud Mitarbeiter händisch über einen FTP Zugang auf einen Webhosting Server hochgeladen. Dieser stellt die Daten unter einer eigenen Adresse bereit, sodass die Inhalte anschließend in den Lern-Store eingepflegt werden können.

Dieses bestehende System hat jedoch einige Nachteile. Zum einen kann keinerlei Zugriffsprüfung stattfinden. Das bedeutet, wer auch immer die URL zum Inhalt erhält, kann auch auf diese Zugreifen. Dies stellt gerade für kommerzielle, Lizenzrechtlich geschützte Inhalte ein Problem dar. Zum anderen ist sowohl beim Aktualisieren als auch beim initialen Upload der Dateien ein erheblicher Aufwand für die Schul-Cloud vorhanden. Dieses System skaliert offensichtlich nicht.

4.2 ANFORDERUNGEN

Das Content-Dashboard soll daher eine Möglichkeit bekommen, noch nicht im Internet verfügbare Lerninhalte einpflegen zu können und anschließend digital über die Schul-Cloud an Lern-Store-Benutzer ausliefern zu können.

Dabei ist es von essentieller Bedeutung, dass Dateireferenzen erhalten bleiben. Dies bedeutet konkret, dass die Ordnerstruktur der hochzuladenden Dateien auch nach dem Upload erhalten bleibt. Andernfalls

würde beim Hosting von beispielsweise Websites die Referenz zu Bildern und anderen eingebetteten Medien verloren gehen.

Zudem ist es uns wichtig, einen konsistenten Datenstand sicherstellen zu können. Bei der sequenziellen Bearbeitung von Dateien kann es vorkommen, dass eine Datei verändert wird, bevor die Referenz auf die Datei angepasst wurde. Einige Lern-Store Benutzer würden für diesen Zeitraum Invalide Daten zu sehen bekommen. Das gleiche Problem kann auftreten, wenn es während des Uploads zu einem Verbindungsabbruch kommen sollte.
Um das zu vermeiden soll es möglich sein Dateien erst sequentiell zu bearbeiten und anschließend gemeinsam die Änderung zu veröffentlichen.

4.3 BESTEHENDE SYSTEME

Nun kann man leicht auf die Idee kommen, dass eine solche Aufgabe mittels einem klassischen, extern eingebundenen, Datei-Hoster[1] umgesetzt werden kann. Bei genauerer Analyse stellen sich dabei jedoch einige Probleme heraus.

Meist ist die Integration einer Nutzerverwaltung bei den Tools nicht möglich. Viele Anbieter sind konzeptionell darauf ausgelegt als eigenständiges Tool zu agieren. Dies bedeutet, dass kaum Schnittstellen vorhanden sind um Accounts für bestehende Anbieter zu synchronisieren. Dies ist jedoch eine Anforderung, welche unser Dashboard allgemein erfüllen muss, damit wir die Kontrolle über die Daten behalten.

Zudem müssen zu den einzelnen Lerninhalten stets Metadaten gespeichert werden, welche für den Lern-Store zwingend erforderlich sind. Dies ist eine Schul-Cloud spezifische Anforderung und daher bei klassischen Filehosting Anbietern nicht implementiert.

Wenn wir die externen Systeme ausschließlich für den Datei-Upload und das Bearbeiten einbinden können, so gibt es kaum einen Mehrwert gegenüber dem aktuellen Stand. Zum Einpflegen von Lerninhalten müssen die Inhalteanbieter weiterhin zwei Systeme verwenden. Der Hosting-Anbieter für den Upload der Dateien und anschließend unser Dashboard für das Verwalten der zugehörigen Metadaten. Der einzige Vorteil wäre, dass alles von der Schul-Cloud bereitgestellt werden würde.

Wir haben uns daher dazu entschieden, keine externe Lösung für die Dateiverwaltung zu verwenden.
Durch die Eigenentwicklung haben wir ein hohes Maß an Kontrolle über den Zugriff auf Inhalte. Außerdem wird der Upload-Prozess so stark an die Erfassung der notwendigen Metadaten gekoppelt.

1 https://www.omkt.de/webhosting-definition/

4.4 UMSETZUNG

4.4.1 *Auswahl des Datenspeichers*

Als erste Frage während der Umsetzung stellt sich, wo wir die gehosteten Dateien speichern können. Es sollen möglichst viele Speicheranbieter unterstützt werden. um einen späteren Anbieterwechsel zu ermöglichen. So kann die Kompatibilität zu zukünftig in der SchulCloud verwendeten Diensten sichergestellt werden.

Dieses Ziel erreichen wir durch Abstraktion des Datenspeichers. Bei der Recherche haben sich die drei Abstraktionsbibliotheken:
pkgcloud[2], unifile[3] und node-flydrive[4] als besonders umfangreich und zielführend herausgestellt.
Bei der Auswahl der geeigneten Bibliothek sind für uns die folgenden Faktoren besonders relevant: Code-Dokumentation, Unterstützte Datendienste, Performance und Support.

Anforderung	pkgcloud	unifile	node-flydrive
Support	aktiv mit 69 contributors	scheint nicht mehr aktiv entwickelt zu werden. letzter Commit 2018	aktiv mit 7 contributors
Code-Dokumentation	Vollständige Beispiele für CRUD vorhanden + viele GitHub Issues mit Problemlösungen	Nur Dokumentation einzelner Componenten, keine vollständigen Beispiele	sehr einfache API, jedoch kaum Troubleshooting dokumentiert.
Performance	basiernd auf Streams	basierend auf Streams	basierend auf Streams
Speicherdienste	Amazon, Azure, Google, HP, Openstack, Rackspace	FTP, SFTP, Dropbox, GitHub, RemoteStorage, WebDAV	FTP, Amazon, Digital Ocean Spaces

Tabelle 1: Vergleich von Speicherabstraktionsbibliotheken

2 https://github.com/pkgcloud/pkgcloud
3 https://github.com/silexlabs/unifile
4 https://github.com/Slynova-Org/node-flydrive

Fazit: Alle Bibliotheken sind hinsichtlich ihrer Performance unkritisch, da sie auf Streams basieren und der Server daher nicht durch eine große Datei in die Knie gezwungen werden kann. Jedoch unterscheiden sie sich stark im Umfang und der Entwicklungsaktivität. pkgcloud sticht hier positiv hervor. Außerdem unterstützt die Bibliothek die größte Anzahl an großen Enterprise-Anbietern.

Um bei pkgcloud den Anbieter zu wechseln muss lediglich die Initialisierungsfunktion angepasst werden. Alle weiteren Anfragen sind über sämtliche angebotenen Anbieter hinweg konstant.

```
const client = require('pkgcloud').storage.createClient({
  provider: 'provider-name',
  // ... Provider specific credentials
});
```

Listing 1: PKG Cloud - Providerauswahl

Dabei wird ein einfaches Interface, für den Dateizugriff, mit nur 5 Methoden[33] angeboten. Diese decken die Grundfunktionen Upload, Download, löschen und auflisten von vorhanden Dateien ab und sollten für den Anwendungsfall ausreichend sein.

Leider werden jedoch einfache Speicheranbieter wie FTP, WebDAV oder das lokale Dateisystem von pkgcloud nicht unterstützt. Diese spielen im Produktiveinsatz jedoch, aufgrund von Sicherheitsbedenken oder Skalierungsproblemen, eine eher untergeordnete Rolle. Daher haben wir uns für die Verwendung von pkgcloud entschieden.
Für den eigentlichen Speicher betreibt die Schul-Cloud bereits einen Amazon S3 kompatiblen Server, welcher daher als Datenspeicher verwendet wird.

4.4.2 *Identifikation der Dateien*

Widmen wir uns nun der bereits erwähnten Anforderung, Dateireferenzen zwischen Dateien auch nach dem Upload beizubehalten. Daraus ergibt sich die Fragestellung, wie Dateien von uns eindeutig identifiziert werden können. Dies ist sowohl für die Erhaltung von Dateireferenzen als auch für das Anknüpfen weiterer Tools relevant. Ein solches Tool kann das hinzufügen von DRM-Schutz[17] sein, oder eine Berechtigungsprüfung.

Wenn wir die Anforderungen genauer analysieren, so stellt sich heraus, dass es von einer Datei teilweise mehrere Versionen gibt. Eine davon ist veröffentlicht, alle weiteren sind temporär. Dabei reicht es aus, pro Benutzer und Datei eine temporäre Version abspeichern zu können.

Wir kennen über eine Datei somit die folgenden Informationen:

- ist sie veröffentlicht oder temporär
- wer hat sie hochgeladen
- zu welchem Lerninhalt (Ressource) gehört sie
- der Pfad unter welchem sie hochgeladen wurde

4.4.2.1 *Ansatz 1 - Dateinamen*

Nun könnte aus den gegebenen Informationen leicht, nach folgendem Schema, ein eindeutiger Identifier entwickelt werden welcher als Dateiname genutzt werden kann.

```
/{tempOrPublicFolder}/{userId}/{resourceId}/{filepath}
```

`{root}` gibt dabei an, ob es sich um eine temporäre oder veröffentlichte Datei handelt. Die `{userId}` dient als Prävention vor Dateikonflikten zwischen verschiedenen Benutzern und die `{resourceId}` verhindert Namenskonflikte zwischen verschiedenen Inhalten. Der `{filepath}` selbst ist in diesem Rahmen anschließend stets ein eindeutig, da dieser Teil des Pfades die Position der Datei auf dem System des Uploaders repräsentiert.

Dabei besteht jedoch das Risiko, die Maximallänge von Dateinamen des verwendeten Datenspeichers zu überschreiten[6].
Ebenso müssten für alle weiteren Informationen (Status des DRM Schutzes, Erstelldatum, u.vm.), welche pro Datei gespeichert werden sollen entweder das Schema erweitert oder eine zusätzliche Datenbank betrieben werden. Das erweitern des Schemas hätte zur Folge, dass sämtliche Dateien umbenannt werden müssten.

Um eine Datei mit diesem Ansatz zu veröffentlichen muss sie umbenannt werden, da sich ein Teil der ID (`{tempOrPublicFolder}`) in diesem Fall ändert. Die Analysierten Speicherabstraktionsanbieter unterstützen jedoch allesamt nicht das umbenennen von Dateien. Stattdessen muss die Datei heruntergeladen und unter neuem Namen hochgeladen werden. Dieser Prozess ist Zeitintensiv und belastet die Netzwerkverbindung des Servers sehr.
Es hat sich das herausgestellt, das insbesondere das letzte Problem gegen diesen Ansatz der Dateiidentifikation spricht.

4.4.2.2 *Ansatz 2 - Datenbank*

Ein anderer Ansatz ist, sämtliche Metadaten einer Datei in einer separaten Datenbank zu speichern. Diese hat den Vorteil, dass die gespeicherten Metadaten unabhängig von der Datei selbst aktualisiert werden können. Die Datenbank kümmert sich dabei auch um die Zuordnung von einer neuen zufälligen ID zu einem extern im WWW

verfügbaren Pfad. Diese ID wird im Dateispeicher als Dateiname genutzt. Als zufällige ID kann dabei die eindeutige ID verwendet werden, welche sowieso von der Datenbank zur Identifikation des Datenbankeintrages genutzt wird.
Für den Zugriff auf eine Datei muss bei diesem Ansatz stets die Datenbank angefragt werden. Diese löst die vorhandenen Informationen zu einer Datei-ID auf. Erst mit dieser Datei-ID kann die Datei beim Speicheranbieter angefragt werden.

Diese Variante löst alle Probleme, welcher der zuvor erwähnte Ansatz vorzuweisen hatte. Daher entschieden wir uns für ihn.

4.4.3 *Upload Workflow*

Der eigentliche Upload von Dateien kann ein sehr langwieriger Prozess sein. Besonders wenn große Dateien hochgeladen werden, ohne den Verarbeitungsfortschritt anzuzeigen, kann es schnell zu Frustration beim Endnutzer kommen.
Dies tritt immer dann auf, wenn ein klassisches HTML Form Input Element für den Dateiupload genutzt wird. Hier wird vom Browser meist nur in einer Ecke der kleine Hinweis "Datei wird hochgeladen" angezeigt.
Wir wollen dieses fehlende Feedback unbedingt vermeiden. Bei der Recherche nach alternativen entdeckten wir FilePond[5].

FilePond ist eine Bibliothek, welche einen einfachen Dateiupload ermöglicht. Sie nutzt einen speziellen Workflow, der Dateien zunächst in einen temporären Ordner hochlädt. Erst beim Speichern des Formulars wird die Datei dann an den finalen Speicherort verschoben.
Auf diese Weise kann der eigentliche Upload der Dateien bereits im Hintergrund stattfinden, während der Rest des Formulars ausgefüllt wird. Der Nutzer muss dadurch nur selten auf das hochladen der Datei warten.

Leider erfüllt die Bibliothek zwei unserer Grundlegenden Anforderungen nicht.
Erstens wird im Frontend kein Strukturierter Dateiupload ermöglicht.
Zweitens unterstützt der bereitgestellte Servercode nicht die Speicheranbieter, welche für uns relevant sind. Zudem ist er nicht mit der zuvor ausgewählten JavaScript-Bibliothek pkgcloud kompatibel.
Der dritte Kritikunkt ist, das der Server auf PHP basiert, welches in der bestehenden Schul-Cloud Infrastruktur noch nicht verwendet wird. Da nur wenige Projektmitglieder bereits Erfahrungen mit PHP haben, wäre die Anpassung des Codes daher ein Hindernis.
Aufgrund dessen wird diese Bibliothek nur als Inspiration für den Upload-Workflow genutzt.
Mit dem Workflow von FilePond wird sowohl die Anforderung der

5 `https://pqina.nl/filepond/`

Datenkonsistenz erfüllt als auch die gefühlt benötigte Zeit für den Upload reduziert.

Der von uns umgesetzte FilePond Workflow[2] sieht daher folgendermaßen aus:

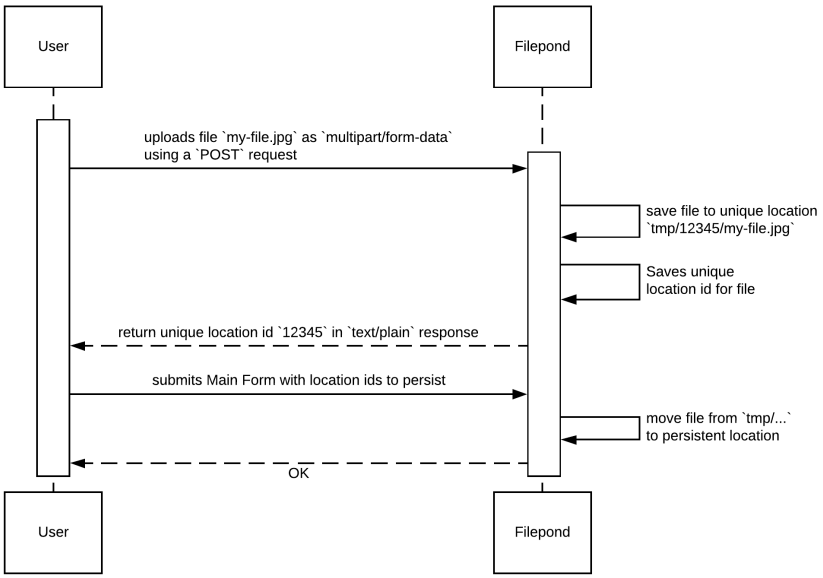

Abbildung 1: FilePond Upload Workflow

4.4.4 *Upload Benutzeroberfläche*

Nachdem jetzt geklärt ist, wie der Upload ablaufen soll, gilt es eine Oberfläche (UI) für diesen Prozess zu entwickeln.
Wie bereits erwähnt ist es uns besonders wichtig, die Ordnerstruktur der hochgeladenen Dateien beizubehalten. Daraus ergibt sich, dass das UI einen Dateibaum beinhalten muss.
Idealerweise soll sich dieser ähnlich wie der Dateiexplorer eines Desktop-Betriebssystems verhalten. Dies bedeutet, dass durch Drag & Drop von Ordnern und Dateien auf einen Ordner A diese in den Ordner A kopiert werden. Jeder Ordner in unserem Dateibaum soll daher Drag & Drop unterstützen. Die Desktop Betriebssysteme unterstützen dabei auch, das kopieren von mehreren Dateien und Ordnern gleichzeitig. Dies möchten wir ebenfalls unterstützen.
Dabei stellt sich jedoch heraus, dass HTML Inputfelder lediglich drei Formen von Dateiauswahl unterstützen. Je nach Einstellung kann der Nutzer: nur eine Datei, mehrere Dateien oder einen einzelnen Ordner zum Upload auswählen. Das Hochladen von mehreren Ordnern wird von den HTML Eingabeelementen nicht unterstützt. Wir benötigen daher eine Alternative. Diese wird mit dem JavaScript Event `drop` bereitgestellt.
In dem geworfenen Event ist unter `event.dataTransfer.items` eine Liste an Dateien und Ordnern angegeben, welche traversiert werden

kann. Traversiert man diese Einträge rekursiv, so kann man sich den Dateibaum der abgelegten Dateien rekonstruieren[6].

All jenen Benutzern auf Geräten ohne Drag & Drop-Unterstützung können wir diesen Workflow leider nicht anbieten, weshalb zusätzlich pro Ordner zwei Knöpfe angeboten werden sollen. Einer für den Upload von Ordnern und einer für weitere Dateien. Andernfalls ist eine Bedienung auf mobilen Geräten wie Tablets unmöglich. Auch Nutzer, welche keine Maus als Eingabegerät nutzen können würden aus dem System ausgeschlossen werden. Leider fehlte uns die Zeit, auch diese Art des Uploads zu implementieren.

Jede einzelne Datei kann nach erfolgreichem Drag & Drop mit dem zuvor beschriebenen FilePond Workflow per `XMLHttpRequest` oder `fetch` hochgeladen werden. Dabei kann das Dateiobjekt mit zusätzlichen Metadaten angereichert werden. Beispielsweise kann in dem Objekt der Uploadfortschritt gespeichert werden.
Damit können wir dem Nutzer für jede einzelne Datei und aggregiert auch für Ordner einen Uploadfortschritt bereitstellen.

Das aggregieren von Werten und die automatische Anpassung des UIs geschieht dabei durch das reactivity System von Vue.js[7] und computed Properties[8]. Dadurch ist es außergewöhnlich einfach, auf all die Events, welche während des Dateiuploads entstehen, zu reagieren (progress, error, ...). Es ist jedoch zu beachten, dass Vue.js reactivity nur auf Objektattribute anwenden kann, welche entweder bei der Initialisierung bereits im Objekt vorhanden sind oder über `Vue.set` [16] dem Objekt hinzugefügt wurden.

Im folgenden ist die implementierte Benutzeroberfläche für den Dateiupload abgebildet.

6 https://gist.github.com/adrianjost/1bca0be7af6b71078d62cb2da1f1f941
7 https://vuejs.org/v2/guide/reactivity.html
8 https://vuejs.org/v2/guide/computed.html

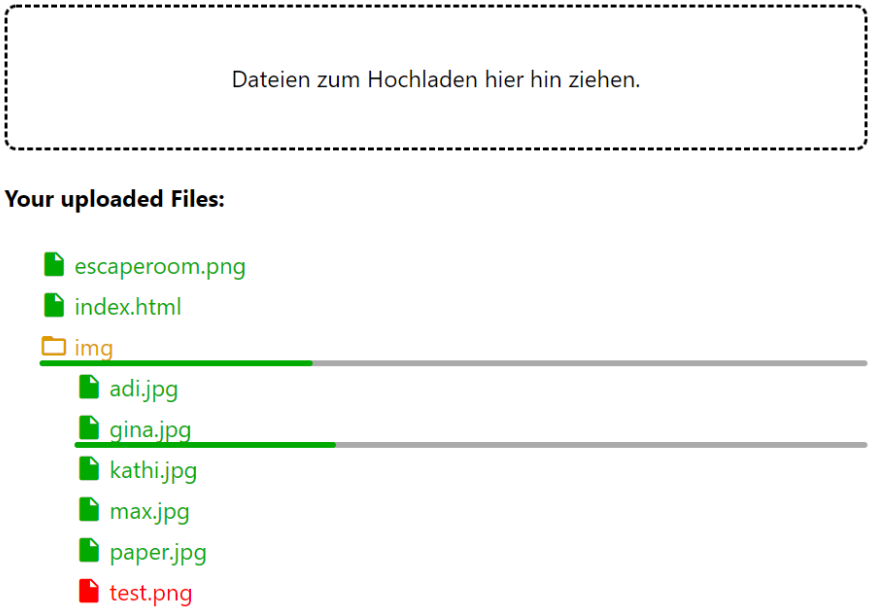

Abbildung 2: Datei Upload Benutzeroberfläche

4.4.5 *Auslieferung von Lerninhalten*

Der letzte wichtige Punkt für das Hosting von Lerninhalten im Lern-Store der Schul-Cloud ist das ausliefern der hochgeladenen Dateien. Dies ist dank der zuvor definierten Punkte sehr einfach umzusetzen. Dazu muss im Server lediglich eine Wildcard-Route[9] definiert werden. Diese antwortet auf alle Anfragen, welche dem Schema `files/get/:resourceId/*` entsprechen.
Die Route muss sich nun lediglich den Pfad der angeforderten Datei aus der URL auslesen. Anschließend muss eine Datenbankabfrage stattfinden, welche die Datei-ID für jene Datei heraussucht, die den angegebenen Pfad aufweist, zur passenden Ressource gehört und den Status `Veröffentlicht` hat. Mit dieser FileID kann anschließend einfach ein Dateistream über pkgcloud vom Speicheranbieter zum Benutzer gesendet werden. Die Zugriffsüberprüfung kann dabei in den entsprechenden Hooks des Service geschehen.

Zusammenfassung

Mit Abschluss dieses Kapitels haben wir den vollständigen Prozess des Hostings von Dateien abgebildet und können Inhalteanbietern von nun an die Möglichkeit bieten, ihre Inhalte ohne externe Dienste im Lern-Store bereitzustellen.

9 `https://modernweb.com/the-basics-of-express-routes/`

5

CSV-IMPORT

Katharina Blaß

5.1 MOTIVATION

Die großen Inhalteanbieter wie Klett oder Cornelsen bieten eine Vielzahl an digitalen Lernressourcen an. Bisher können sie diese lediglich einzeln und von Hand über das Datei-Hosting Feature in den LernStore einpflegen. Dieser Vorgang ist bei größeren Mengen an Inhalten äußerst mühselig und langwierig. Um das massenhafte Einpflegen zu vereinfachen, braucht das Dashboard ein Tool zum automatisierten Hochladen mehrerer Inhalte auf einmal. Dies kann mithilfe einer Import-Funktion über ein standardisiertes Format erfolgen.

5.2 ANFORDERUNGEN

Funktionaler Schwerpunkt des Imports ist das erfolgreiche Einlesen einer Datei in einem kompatiblen Format mit dem anschließenden Einpflegen der ausgelesenen Inhalte in die Ressourcen-Datenbank.
Dabei soll auch der Fall abgedeckt werden, dass die Datei nicht exakt dem von uns gewünschtem Schema entspricht. Den Inhalten aus der Datei könnte beispielsweise ein Beschreibungstext fehlen. Dieser müsste dann einer alternativen Spalte aus der Datei entnommen oder manuell ergänzt werden können. Um diesen Fall abzudecken, soll der Nutzer möglichst viel Handlungs- bzw. Korrekturspielraum haben.
Darüber hinaus soll das Tool auch große Dateien mit bis zu 10000 Einträgen ordnungsgemäß, ohne Fehler oder Timeouts und bei angemessener Performance verarbeiten können. Die für den Nutzer wahrnehmbare Performance ist hier entscheidend gegenüber der wahren Verarbeitungsgeschwindigkeit. Der Import von großen wie kleinen Dateien soll sich schnell und flüssig anfühlen.
Zur Umsetzung dieser Anforderung wird das Tool mit drei verschieden großen Dateien getestet. Nur wenn der Import in allen Teilprozessen diese 3 Dateien schnell und fehlerfrei verarbeiten kann, gilt die Anforderung der Skalierbarkeit als erfüllt.

- Kleine Datei mit 5 Einträgen (~780 B)

- Mittlere Datei mit 2000 Einträgen (~300 KB)
- Große Datei mit 10000 Einträgen (~1,43 MB)

Neben diesen beiden Anforderungen der Skalierbarkeit und des maximalen Handlungsspielraums gelten weiterhin die allgemeinen nichtfunktionalen Anforderungen des Projektes.

5.3 UMSETZUNG

Zur Verwaltung von Ressourcen können verschiedenste Systeme wie beispielsweise Excel genutzt werden. Die Wahl trifft jeder Inhalteanbieter für sich. Für den Massenimport ist es daher wichtig, ein standardisiertes Dateiformat zu akzeptieren, dass von den meisten Verwaltungssystemen unterstützt wird. Ein solches Format ist CSV. Viele Verwaltungssysteme haben eine CSV-Export-Funktion. Aus diesem Grund implementieren wir den Massenimport als CSV-Import. Jede Zeile einer CSV-Datei wird dabei als eigenständiger Inhalt interpretiert mit den Spaltenattributen als dessen Metadaten.

5.3.1 *Nutzerführung*

Der CSV-Import ist ein komplexer Prozess, der eine Vielzahl an Eingaben und Aktionen vom Benutzer erwartet. Der Nutzer muss seine Datei hochladen, unsere Metadaten ausfüllen bzw. entsprechenden Spalten zuweisen und die Inhalte bei Bedarf veröffentlichen. Nichtsdestotrotz soll der Nutzer das Tool ohne großartiges Vorwissen einfach verstehen und bedienen können. Um das zu erreichen, bedarf es einer klar strukturierten Oberfläche, die ihm jederzeit vorgibt, was er als Nächstes tun soll.
Um diese Oberfläche bestmöglich zu konzipieren, suchen wir uns Inspiration bei vorhandenen Systemen, die ähnlich komplexe Prozesse abbilden. Ergebnis dieser Recherche ist der Installationsprozess, wie ihn viele Desktopanwendungen umsetzen. Jede Installation erwartet diverse Aktionen vom Nutzer, beispielsweise welche Teilsysteme er installieren möchte oder ob er die AGB gelesen hat. Dabei ist der Prozess ähnlich komplex wie unser Import. Um den Nutzer zu unterstützen, hat man sich bei Installationen oftmals entschieden, Wizards, auch Stepper[1] genannt, zu verwenden. Diese ermöglichen es, einen komplexen Workflow in mehrere einfachere und logisch getrennte Teilschritte zu untergliedern. Jeder Teilschritt wird dabei durch eine eigene, aufgeräumte Seite dargestellt mit ausreichend Platz für Erläuterungen zur aktuellen Aufgabe. Dadurch sieht der Nutzer nur die Informationen und Interaktionsmöglichkeiten, die für ihn in diesem Moment relevant sind. Gleichzeitig weiß er jederzeit, in welchem Teilschritt er sich befindet und wie viele er noch bis zur Fertigstellung durchlaufen muss[37] [36] [8]. Mithilfe eines solchen Steppers, soll

1 https://material.io/archive/guidelines/components/steppers.html#

auch die Nutzerführung des CSV-Imports verbessert werden. Dazu lässt sich der Import-Prozess in die folgenden Teilschritte untergliedern:

1. "Datei hochladen": Der Nutzer lädt seine CSV-Datei hoch.
2. "Metadaten zuordnen": Damit die zu importierenden Daten dem vorgegebenem Metadaten-Schema entsprechen, muss jedem Metadaten-Feld eine CSV-Spalte zugeordnet werden.
3. "Importieren": Der Nutzer kontrolliert seine Eingaben.
4. "Fertig": Der Nutzer bekommt Rückmeldung über den Erfolg oder Misserfolg des Imports.

Die Schul-Cloud hat bereits eine eigene Implementierung eines Steppers[2]. Dieser wird für den Import wiederverwendet und lediglich um eine unterstützende Farbgebung ergänzt. Der aktive Teilschritt ist in der Akzentfarbe (Blau) hervorgehoben. Alle bereits absolvierten Teilschritte, sind mit einer sekundären Farbe sowie einem Häkchen gekennzeichnet.

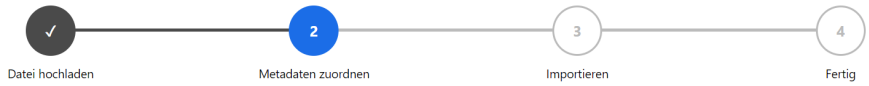

Abbildung 3: Stepper für den CSV-Import mit 4 Teilschritten

Auf diese Weise kann der Nutzer nun Schritt für Schritt durch den komplexen Prozess des Imports geführt werden.
Falls sich bei einem späteren User-Test herausstellt, dass die Nutzerführung mit Steppern nicht ausreicht, gibt es noch die Möglichkeit, Tutorials oder Hilfeseiten zu ergänzen. Beliebt ist hierbei die Bibliothek Intro.js[3]. Sie ermöglicht dem Nutzer einen Rundgang durch die Oberfläche, um ihm die wichtigsten Funktionen zu erklären. Auf solche Hilfsmittel soll zunächst verzichtet werden.

5.3.2 *Schritt 1 - CSV-Upload*

Im ersten Teilschritt des Imports soll dem Nutzer eine Option zum Einlesen einer CSV-Datei angeboten werden. Die einfachste Umsetzung ist die Nutzung eines Inputs vom Typ File, welcher beim Klick des Nutzers einen Filechooser öffnet. Über diesen kann der Nutzer seine Datei aus seinem Dateisystem auswählen.
Eine andere, weit verbreitete Variante ist Drag & Drop. Diese ermöglicht dem Nutzer, Dateien auszuwählen, indem diese aus einem bereits geöffneten Explorer in die Oberfläche gezogen werden. Drag & Drop ist oftmals zeitsparender als ein Filechooser, wird allerdings

2 `https://github.com/schul-cloud/nuxt-client/blob/master/src/components/StepProgress.vue`
3 `https://introjs.com/docs/`

nicht von mobilen Geräten unterstützt. Außerdem ist diese Variante von körperlich eingeschränkten oder älteren Menschen oft nicht benutzbar.
An dieser Stelle raten UX Experten, beide Möglichkeiten kombiniert anzubieten und so dem Nutzer die Wahl zu lassen[39].
Aus diesem Grund platzieren wir einen Input vom Type File (Zeile 6-10) innerhalb einer Dropzone, die auf das Drop-Event hört (Zeile 2). Der Nutzer kann nun entweder Drag & Drop oder per Klick in die Dropzone einen Filechooser benutzen. Zusätzlich vereinfacht Zeile 9 dem Nutzer das Suchen der richtigen Datei per Filechooser, indem nur .CSV-Dateien in seinen Ordnern angezeigt werden.

```
1  <!-- Dropzone ermöglicht Drag & Drop -->
2  <Dropzone @drop.prevent="handleDrop">
3      <label>
4          Lade hier eine CSV-Datei hoch
5          <!-- File-Input öffnet Filechooser per Klick -->
6          <input
7              @input="handleFileChosen"
8              type="file"
9              accept=".csv"
10         />
11     </label>
12 </Dropzone>
```

Listing 2: Upload-Bereich mit Drag & Drop und Filechooser

Sobald die Datei erfolgreich eingelesen wurde, kann der Nutzer in den nächsten Teilschritt wechseln.

5.3.3 *Schritt 2 - Metadaten-Zuordnung*

Die Endnutzer des Lern-Stores erwarten zu allen angebotenen Inhalten vollständige Metadaten, wie beispielsweise den Titel oder einen Beschreibungstext. Um das Vorhandensein dieser Informationen zu garantieren, müssen alle neuen Inhalte beim Einpflegen einem einheitlichen Metadaten-Schema entsprechen. Nun können Inhalteanbieter über den Import bislang beliebige CSV-Dateien hochladen. Die darin aufgelisteten Inhalte können somit je nach Spalten unterschiedlichste Metadaten mitbringen. Um diese Inhalte dennoch einheitlich in den Lern-Store einpflegen zu können, müssen sie nach dem Hochladen unserem Schema angepasst werden. Dies soll im folgenden Teilschritt durch Zuordnung der einzelnen CSV-Spalten auf unsere vorgegebenen Metadaten erfolgen. Die Gestaltung einer intuitiven Oberfläche ist dabei maßgeblich für das Verständnis dieser komplexen Aufgabe.

Eine einfache Möglichkeit für solch eine Oberfläche wäre, die auswählbaren Spaltenüberschriften als je einen Button pro Metadaten-Feld abzubilden.

Abbildung 4: Zuordnung mit Button

Der Nutzer kann so jede Zuordnung mit nur jeweils einem Klick auf den gewünschten Button ausführen. Leider nimmt diese Variante bei einer großen Anzahl an Buttons viel Platz ein und wird daher schnell unübersichtlich. Selbst wenn die bereits zugeordneten Buttons ausgeblendet werden, schließlich sollen die CSV-Spalten nur genau einmal zugeordnet werden können, wird dieser Nachteil kaum abgemildert.

Weiterhin wäre es möglich, pro Metadaten-Feld ein Select anzubieten, das die Spaltenüberschriften der CSV-Datei als Optionen bereitstellt.

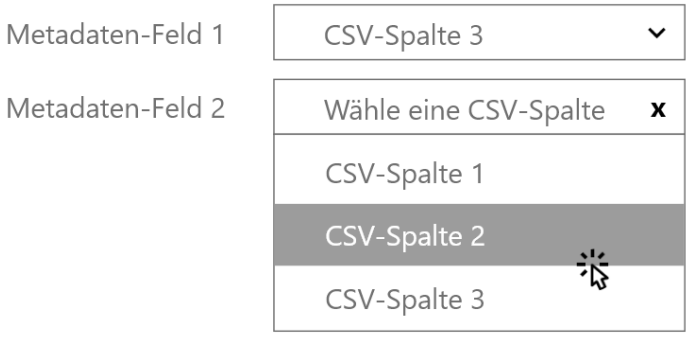

Abbildung 5: Zuordnung mit Select

Diese Variante überzeugt mit ihrer Übersichtlichkeit, denn zu jedem Zeitpunkt ist lediglich das Select des aktuell zu betrachtenden Metadaten-Feldes ausgeklappt. Allerdings ist die Select-Methode per Maus nicht ganz so schnell zu bedienen, wie die zuvor genannte Button-Lösung. Die Navigation per Tastatur funktioniert dagegen etwas besser.

Als dritte Möglichkeit ließe sich für die Zuordnung eine zweispaltige Tabelle verwenden. Diese würde alle Metadaten-Felder in der ersten und alle CSV-Spalten in der zweiten Spalte abbilden.

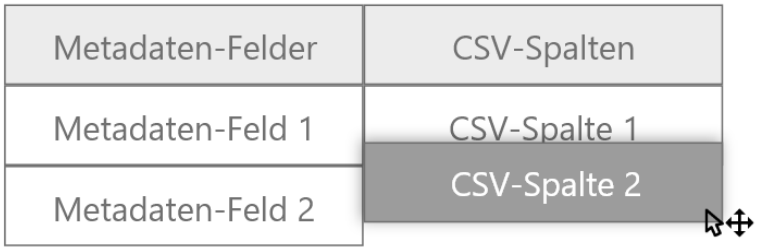

Abbildung 6: Zuordnung mit Tabelle

Der Nutzer könnte nun einzelne Zellen solange vertikal verschieben, bis eine Zeile genau einer Zuordnung zwischen einem Metadaten-Feld und einer CSV-Spalte entspricht. Dieses Verfahren wäre vermutlich schneller und übersichtlicher als alle bisher Vorgestellten. Problematisch ist jedoch, dass diese Methode ausschließlich eine 1:1 Zuordnung zulässt. Das bedeutet, es ist hier nicht möglich, einem Metadaten-Feld keine CSV-Spalte zuzuordnen. Eine solche Restriktion würde das Import-Tool allerdings beinahe unbenutzbar machen. Schließlich werden die wenigsten Inhalteanbieter exakt die richtige Anzahl an CSV-Spalten anbieten, die sich auch alle semantisch zuordnen lassen. Darüber hinaus ist es sehr schwer, diese Methode verständlich und Accessibility-tauglich umzusetzen.

Keine der oben genannten Möglichkeiten ist, für sich betrachtet, ausreichend geeignet, um die komplexe Metadaten-Zuordnung verständlich und übersichtlich darzustellen. Darum fällt die Wahl auf eine Kombination aus der Select- und der Tabellenlösung. Implementiert wird eine zweispaltige Tabelle, mit den Metadaten in der ersten, den CSV-Spalten als Selects in der zweiten Spalte. Die Zuordnung erfolgt zeilenweise, indem für das entsprechende Metadaten-Feld eine geeignete Option im Select derselben Zeile ausgewählt wird. Dabei hat das Select nur diejenigen Optionen aktiviert, die noch nicht zu den anderen Metadaten-Feldern zugeordnet wurden. So wird sichergestellt, dass jede CSV-Spalte nur maximal einem Metadaten-Feld zugeordnet wird. Gleichzeitig erleichtert das Deaktivieren vergebener Optionen die Tastaturnavigation, da die Werte übersprungen werden können. So können wir die Geschwindigkeit des Vorgangs erhöhen. Für den Fall, dass einem Metadaten-Feld keine CSV-Spalte zugeordnet werden kann, existiert eine alternative Zuordnungsoption - "keine Zuordnung möglich". Wie mit den dabei entstehenden, unvollständigen Inhalten umzugehen ist, wird im nächsten Teilschritt thematisiert.

Um dem Nutzer bei diesem komplexen Vorgang weiter zu unterstützen, befinden sich unter allen Metadaten-Feldern kurze Beschreibungen oder Beispielwerte. So soll der Nutzer eine bessere Vorstellung erhalten, was sich hinter unseren Metadaten-Feldern verbirgt, um eine semantisch korrekte Zuordnung durchführen zu können.

5.3.4 *Schritt 3 - Vorschau, Veröffentlichen, Importieren*

Nach dem erfolgreichen Zuordnen der Metadaten können die Daten in diesem Teilschritt importiert werden. Zuvor hat der Nutzer noch die Gelegenheit, die Metadaten-Zuordnung seiner CSV-Inhalte in der Vorschautabelle zu überprüfen. Hierbei werden aus Gründen der Übersichtlichkeit lediglich die ersten fünf Inhalte angezeigt. Leere Spalten in Folge einer fehlenden Zuordnung fallen an dieser Stelle sofort auf. Der Nutzer hat so eine visuelle Kontrollmöglichkeit und kann gegebenenfalls in den vorherigen Teilschritt wechseln, um die Zuordnungen anzupassen.

Aufgrund des Handlungsspielraums, den wir dem Nutzer beim Zuordnen der Metadaten lassen, können auch unvollständige Daten entstehen. Einerseits dürfen diese nicht einfach importiert werden, da sonst dem Endnutzer des Lern-Stores das Vorhandensein sämtlicher relevanten Metadaten nicht mehr garantiert werden kann. Andererseits soll der Nutzer auch nicht am Weitermachen gehindert werden, solange seine Inhalte unvollständig sind. Dies würde das Import-Feature unbenutzbar machen.

Die Lösung des Problems liegt in dem isPublished-Flag, das in der Datenbank zu jedem Inhalt gespeichert wird. Dieses ermöglicht die Unterscheidung zwischen öffentlichen und privaten Inhalten. Während öffentliche Inhalte von allen Nutzern des Lern-Stores genutzt werden können, sind private Inhalte lediglich von ihren Eigentümern einsehbar. Mithilfe des isPublished-Flags können Inhalteanbieter also entscheiden, wann sie Inhalte "live schalten" wollen und wann sie nur Ihnen sichtbar sein sollen. Hierfür müssen sie lediglich eine entsprechende Checkbox setzen.

Da private Inhalte den Nutzern des Lern-Stores verborgen bleiben, dürfen sie, auch wenn sie unvollständig sind, importiert werden. Lediglich die zwei essenziellen Informationen, nämlich den Titel und die URL, unter der der Inhalt verfügbar ist, sind weiterhin Pflichtangaben.
Entscheidet sich der Inhalteanbieter allerdings dazu, Inhalte zu importieren und zu veröffentlichen, darf er das nur, wenn sie vollständig sind. Zum Veröffentlichen setzt der Nutzer das isPublished-Flag, woraufhin sich ein Benachrichtigungsfenster öffnet. Dieses soll den Inhalteanbieter über das Ergebnis einer zuvor im Hintergrund angestoßenen Validierung informieren, die entscheidet, ob seine Daten veröffentlicht werden können. Dabei wird geprüft, ob alle geforderten Metadaten enthalten sind und diese einem vorgegebenen Datenschema entsprechen. Mehr dazu im folgenden Validierungskapitel. Bei einem positiven Validierungsergebnis können die Inhalte importiert und veröffentlicht werden. Dazu klickt der Nutzer auf den Import-Button, woraufhin ein Array von Inhalte-Objekten an den Server gepostet wird. Dieser fügt unter der Route `/resources/bulk` die Inhalte in die Ressourcen-Datenbank ein.
Ist die Validierung gescheitert, wird der Nutzer informiert, dass eine

Veröffentlichung nicht möglich ist. Er hat dann die Möglichkeit zur Korrektur seiner Metadaten-Zuordnung im vorherigen Schritt. Außerdem kann er immer noch den Import-Button klicken und so seine Inhalte ohne Veröffentlichung importieren. Das isPublished-Flag wird in diesem Fall automatisch im Server entfernt.

5.3.5 *Schritt 4 - Ergebnis*

Im vierten und letzten Teilschritt soll der Nutzer Feedback über den Erfolg oder Misserfolg des Importvorgangs erhalten. Im Fehlerfall wird ihm, neben einer entsprechenden Grafik, die Fehlermeldung sowie ein "Erneut versuchen"-Button angeboten. Weiterhin hat der Nutzer dann die Möglichkeit, zurück zur Verwaltungsseite zu navigieren oder einen neuen Import zu beginnen. Diese beiden Möglichkeiten sind dem Nutzer ebenfalls im Erfolgsfall gegeben. Darüber hinaus erscheinen eine entsprechende Grafik sowie die Angabe, wie viele Inhalte importiert und wie viele davon veröffentlicht werden konnten. Zusätzlich kann der Nutzer nun auch mit einem vordefinierten Filter zur Verwaltungstabelle wechseln. Dort werden ihm die soeben importierten Inhalte zur Bearbeitung angezeigt. Private Inhalte können so einfach vervollständigt und anschließend veröffentlicht werden.

5.3.6 *Validierung*

Die Validierung von Inhalten bei ihrer Erstellung hat das Ziel, sie auf ein einheitliches Metadaten-Schema zu prüfen. Inhalte, denen die Pflichtangaben (Titel und URL) fehlen oder deren Metadaten nicht dem vorgegebenen Datenschema entsprechen, werden nicht in der Ressourcen-Datenbank gespeichert. Beispielsweise soll die URL, unter welcher die Ressource abgegriffen werden kann, mit "/", "https://" oder "http://" beginnen.
Im Regelfall wird eine Validierung kurz vor dem Speichern der Daten in die Datenbank durchgeführt. Beim Import-Tool soll dies allerdings schon früher, im dritten Teilschritt geschehen. Auf diese Weise kann der Nutzer vor dem Speichern seiner Daten über das Ergebnis informiert werden und bei Bedarf Anpassungen vornehmen. Zu diesem Zeitpunkt kennt lediglich das Frontend die Inhalte. Es liegt daher nahe, die Validierung im Frontend zu implementieren.
Es existiert jedoch bereits eine Validierung im Backend, die später beim Speichern eines Inhaltes verwendet wird. Diese ist in dem Create-Hook der `/resources` - Route unter Verwendung des Frameworks AJV[4] implementiert. Hierfür wird ein Validierungsschema definiert, welches die Regeln enthält, die auf die zu prüfenden Daten angewendet werden sollen. Mithilfe dieses Schemas führt AJV dann eine Validierung der vom Backend übergebenen Daten durch. Sollte die Validierung scheitern, wird zurückgegeben, welche Regeln des Sche-

4 `https://github.com/epoberezkin/ajv`

mas nicht erfüllt sind.
Die Frage ist nun, ob die bestehende Validierung im Backend angepasst und wiederverwendet werden sollte oder eine neue Implementierung im Frontend die bessere Wahl wäre.

5.3.6.1 *Frontend- oder Backendvalidierung*

Die bestehende Backend-Validierung kann leicht an den zweiten Anwendungsfall angepasst werden. Das Frontend muss dann lediglich an geeigneter Stelle die Validierungs-Route im Backend anfragen und auf das Ergebnis warten. Auf diese Weise existiert nur eine einzige testbare Implementierung. Auch die Entscheidung, welches Framework genutzt werden soll, wird so nur an einer Stelle getroffen. Das ermöglicht bei Bedarf ein einfaches Austauschen von AJV. Ein weiterer Vorteil liegt in der Verlagerung des Berechnungsaufwandes vom Client hin zum Server. Dies erlaubt auch Nutzern mit vergleichsweise leistungsschwachen Rechnern eine Validierung bei guter Performance. Voraussetzung hierfür ist eine schnelle Internetverbindung. Nur so kann die Validierungsanfrage des Frontends, bei der alle zu importierenden Daten an den Server geschickt werden müssen, zügig erfolgen. Ist dies nicht gegeben, würde diese große Anfrage sehr lange Wartezeiten nach sich ziehen. Um diesen Performance-Nachteil bei schlechtem Netz zu mindern, sollte die Validierung so früh wie möglich, also mit dem Wechsel zum dritten Teilschritt, angestoßen werden. Während der Nutzer dann die Vorschautabelle kontrolliert, findet im Hintergrund bereits die zeitaufwändige Übertragung der Daten an das Backend statt. Diese kleine Optimierung der wahrnehmbaren Performance sollte in jedem Fall umgesetzt werden.

Die Alternative würde die Erstellung des Validierungsschemas und die Durchführung der Validierung auf Frontendseite umfassen. Dank ihrer einmaligen Verwendung im dritten Teilschritt des Importes ist die reine Frontend-Implementierung im Gegensatz zur Backend-Validierung leicht an sich verändernde Anforderungen anpassbar. Der entscheidende Vorteil besteht jedoch in dem geringen Netzwerk Traffic, da keine Anfragen an das Backend gestellt werden müssen. Daher ist diese Variante selbst bei schlechtem Internetzugang, aber einigermaßen guter Rechnerleistung, sehr performant. Dem gegenüber steht der Nachteil der Redundanz, schließlich existiert bereits eine semantisch identische Validierung im Backend. Diese beiden Implementierungen müssen trotz eigener Validierungsschemata und Umsetzungen konsistent gehalten werden. Eine Lösung hierfür wäre, das Framework AJV auch in der Frontend-Validierung zu verwenden. Da die Bibliothek relativ klein ist, ließe sie sich ohne Performance-Einbuße einbinden. Zusätzlich könnte das Validierungsschema des Backends über eine entsprechende Route an das Frontend ausgeliefert und dort wiederverwendet werden. Auf diese Weise gäbe es trotz zweier Implementierungen nur eine Source-of-Truth und damit ausreichende Konsistenz.

Beide vorgestellten Varianten haben ihre Vor- und Nachteile und sind besonders davon abhängig, wie gut der Nutzer technisch ausgestattet ist. Hat er einen leistungsstarken Rechner zur Verfügung eignet sich die Frontend-Methode, bei gutem Internetzugang überwiegen die Vorteile der Backend-Implementierung. Da die Ausstattung der Nutzer unbekannt ist, soll ein ausführlicher Performance-Test die Entscheidung erleichtern. Ziel des Tests ist, herauszufinden, welche Variante unter schlechten technischen Voraussetzungen eine bessere Leistung erbringt. Hierfür wird das Chrome-eigene Performance Development-Tool verwendet. Zur Unterscheidung von leistungsschwacher und -starker CPU eignen sich ein Laptop im oberen Mittelklasse-Bereich und ein schwaches mobiles Endgerät. Die Internetverbindung kann direkt über das Performance-Tool auf fast-3G und slow-3G simuliert werden. Getestet werden die beiden Implementierungen, wie in den Anforderungen verlangt, mithilfe der 3 Testdateien, bestehend aus 5, 2000 und 10000 Zeilen. Diese enthalten exakt dieselben, sich wiederholenden Inhalte. Für konstante Testbedingungen ist die Zuordnung der Metadaten im Teilschritt 2 identisch. Gemessen wird bei der Frontend-Variante die Ausführungsdauer der Validierungsmethode `validateBeforeImport()`. Bei der Backend-Implementierung fragt diese Methode lediglich die Validierungsroute im Server an, welcher die eigentliche Validierung durchführt. Daher wird hier die Zeitspanne von Ausführungsbeginn der Methode `validateBeforeImport()` bis zum Erhalt der Serverantwort gemessen.
Die Ergebnisse[11] sind tabellarisch (siehe Anhang Abbildung 23) und in 3 Diagrammen (siehe Anhang Abbildungen 24 und 25) zusammengefasst.

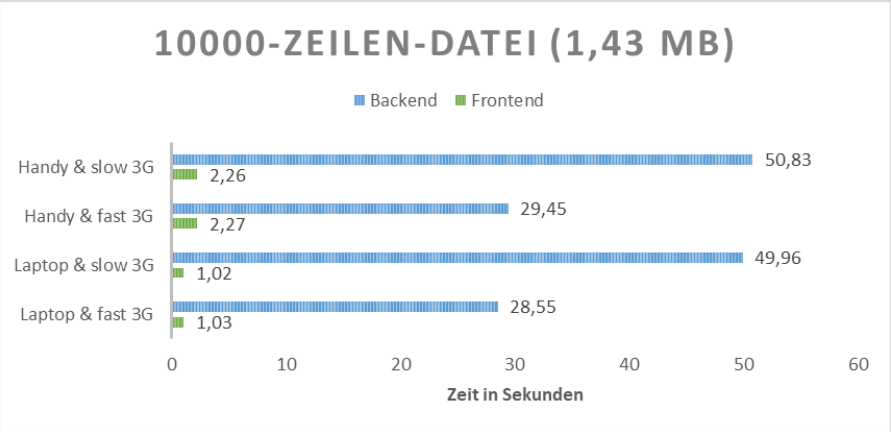

Abbildung 7: Performancevergleich 10000-Zeilen Datei

Besonders das Diagramm zur 10000-Zeilen Datei zeigt deutlich, dass der Validierungsvorgang im Frontend um ein Vielfaches schneller ist, als im Backend. Selbst wenn die CPU zusätzlich noch über das Performance-Tool maximal (6-fach) gedrosselt werden würde, bliebe die Frontend-Implementierung weiterhin klarer Sieger. Aus diesem Grund fällt die Entscheidung auf eine Validierung im Frontend.

5.3.6.2 *Performance Optimierung*

Neben dem frühzeitigen Anstoß der Validierungsberechnung, gibt es weitere Möglichkeiten zur Verbesserung der Performance. Zwei Probleme, die optimiert gehören, fielen im Performance-Test besonders auf: die vorbereitende Formatierung der zu importierenden Ressourcen und die Validierung selber.

Die CSV-Inhalte müssen vor dem Import in das vom Server akzeptierte Datenformat gebracht werden. Hierbei handelt es sich um ein Array von Inhalte-Objekten. Diese Formatierung findet vor der Validierung mit dem Wechsel von Teilschritt 2 zu 3 statt. Auf diese Weise verlängert sich die Wartezeit auf das Validierungsergebnis erheblich. Ein späterer Zeitpunkt für die Formatierung ist jedoch nicht möglich, da die Vorschau-Tabelle in Teilschritt 3 die so strukturierten Daten zur Anzeige benötigt. Im Anschluss an die Formatierung folgt die Validierung. Erst nach deren Abschluss wird die neue Seite gerendert. Bis dahin hat der Nutzer weder Interaktionsmöglichkeiten mit der alten Seite noch sieht er Fortschritt.

Um die Ursache dieses Problems zu verstehen, ist ein kurzer Exkurs in die Arbeitsweise des Browsers notwendig.

Der Browser arbeitet, ohne weitere Optimierungen, mit nur einem Thread, dem Mainthread. Auf diesem findet die Abarbeitung von JavaScript, Rendering und Nutzer-Interaktionen mithilfe des sogenannten Event-Loops statt.
Die folgende Abbildung visualisiert ein simples Modell von der Arbeitsweise des Event-Loops.

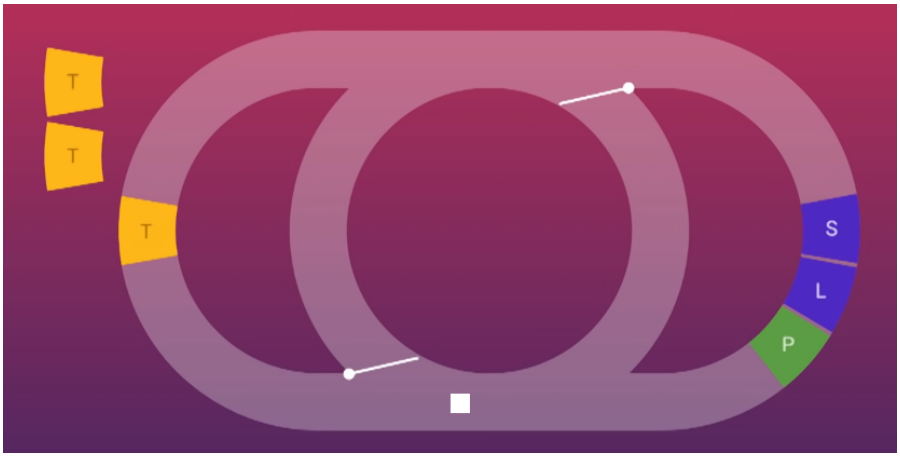

Abbildung 8: Event-Loop [7]

Solange keine Aufgaben anstehen, dreht der Event-Loop wie ein Auto Runde für Runde entlang der inneren Bahn. Zur Abarbeitung von Rendering-Aufgaben muss er regelmäßig die rechte Umleitung fahren. Wird nun beispielsweise ein Klick-Event gefeuert, auf das ein EventListener hört, so wird die JavaScript-Aufgabe an der linken Umleitung in den Task-Stack eingereiht. Bei der nächsten Gelegenheit nimmt der Event-Loop diese Umleitung und arbeitet die gesamte

Aufgabe ab. Dabei ist es ihm egal, wie lange die Abarbeitung dauert. Erst danach bewegt er sich weiter und kann über die rechte Umleitung die, in der Zwischenzeit angefallenen, Rendering-Aufgaben abarbeiten. Auf diese Weise können lange JavaScript-Aufgaben, wie unsere Formatierung oder Validierung den Mainthread so blockieren, sodass die neue Seite nicht gerendert werden kann und der Browser auf keine Eingaben mehr reagiert[7] [15] [32].

Das Problem ist mithilfe der Methode `setTimeout()` lösbar. Diese Funktion erlaubt es, eine bestimmte Zeit zu warten, bevor ein Callback ausgeführt wird. Aus Sicht des Browsers wird lediglich der Callback nach Ablauf der Zeit als neue Aufgabe dem Task Stack hinzugefügt und die aktuelle Aufgabe damit abgeschlossen. Dadurch hat der Browser die Chance eine weitere Runde im Event-Loop zu drehen und Rendering Aufgaben auszuführen, bevor er sich dem Task Stack erneut widmet und den Callback abarbeitet[7].
Diese Funktion nutzen wir nun, um ein Promise zu bauen, dessen resolve-Funktion nach Ablauf des Timeouts in Zeile 3 ausgeführt wird.

```
1 const promiseYield = (duration) => {
2     return new Promise((resolve, reject) => {
3         setTimeout(resolve, duration);
4     });
5 };
```

Listing 3: Promise mit Timeout-Trick

Mit diesem Promise soll der Browser die ersten Inhalte formatieren, anschließend die neue Seite rendern und dann die restlichen Ressourcen verarbeiten. Für die Umsetzung werden die zu importierenden Inhalte in Chunks fester Größe aufgeteilt. Anschließend wird über das Array dieser Chunks iteriert und für jeden Chunk ein promiseYield (siehe Listing 3) mit der Formatierungsmethode als resolve-Funktion zurückgegeben (Zeile 3-5). Wichtig ist hierbei, den Timeout in Zeile 3 auf 0 Millisekunden zu setzen, um keine unnötigen Verzögerungen zu erzielen.

```
1 const chunkPromises = this.chunkArray(this.csv.content,
  ↪  chunkSize).map(
2     (resources) => {
3         return promiseYield(0).then(() => {
4             return resources.map(formatResource);
5         });
6     }
7 );
```

Listing 4: Formattierung mit PromiseYield

Aus Sicht des Browsers passiert bei der Formatierung nun Folgendes. Anstatt eine große blockierende Aufgabe abzuarbeiten, wird sie in kleine Chunk-große Tasks zerlegt. Diese werden in der richtigen Reihenfolge im Task-Stack eingereiht und nacheinander abgearbeitet. Zwischendurch hat der Browser die Gelegenheit, auch den Rendering-Stack abzubauen. So sind weitere Nutzer-Interaktionen und das Laden der neuen Seite von Teilschritt 3 nebenher möglich.

Eine kleine Besonderheit ist die Formatierung der ersten 5 Inhalte. Diese werden, wie bereits erwähnt, für die Anzeige in der Vorschau-Tabelle benötigt und sollen so schnell wie möglich bereitstehen. Daher werden diese synchron formatiert und das oben beschriebene Verfahren erst auf die darauffolgenden Inhalte angewendet.

Auch die nachfolgende Validierung wird auf dieselbe Weise in Chunks aufgeteilt und mit dem promiseYield asynchron ausgeführt. Dieses Vorgehen verbessert zwar nicht die reale Performance, dafür aber deutlich die User Experience. Für den Nutzer ist die Validierung selbst bei einer 10000-Zeilen Datei und 6x gedrosselter CPU kaum mehr wahrnehmbar, wo sie zuvor mehrere Sekunden sämtliche Interaktionen blockiert hat.

Zusammenfassung

Mit Abschluss dieses Kapitel erhalten größere Inhalteanbieter die Möglichkeit, mehrere Inhalte auf einmal in den Lern-Store einzupflegen. Dank der Nutzung von Steppern werden die Nutzer durch die zu bewältigenden Teilschritte des CSV-Imports geleitet. Diese umfassen den Hochlade-Prozess der CSV-Datei, das Zuordnen der Metadaten, die Möglichkeit zur Kontrolle und Veröffentlichung der Inhalte sowie eine abschließende Ergebnisseite. Eine optimierte Validierung der zu importierenden Daten im Frontend garantiert eine konsistente und nachvollziehbare Trennung zwischen öffentlichen und privaten Inhalten.

6

FILTER

Adrian Jost

6.1 MOTIVATION

Filter sind ein essentieller Bestandteil eines Stores. Sie bilden den Einstiegspunkt zu den Inhalten und sorgen dafür, dem Benutzer die Inhalte anzuzeigen, nach welchen er sucht. Allgemein betrachtet ist auch eine Suchleiste ein Filter. Wie wichtig solche Filter sind, ist leicht am Beispiel von Suchmaschinen zu sehen. Heute ist es unmöglich geworden, ohne Filter, an die gewünschten Informationen zu gelangen[29][43]. Obwohl das Filtern also solch ein wichtiger Punkt für ein online Business ist, gibt es nur sehr wenige Tools, welche das Einbinden solcher Filter ermöglichen. Dies lässt sich mit der Schwierigkeit begründen, die notwendige Datenbanksyntax in die natürlichsprachliche Welt zu übersetzen. Erschwert wird die Entwicklung eines universellen Filter Plugins zudem durch die großen Syntaxunterschiede über verschiedene Datenbanksysteme hinweg.

Vor 2 Jahren entstand daher ein erster Prototyp eines erweiterbaren Filters. In dem Seminar "Web-Programmierung und Web-Frameworks" wurde die erste Version des in dieser Arbeit behandelten Dashboards entwickelt. Dabei stellte sich heraus, dass Filter zwingend Notwendig sind um die eingepflegten Lerninhalte effektiv verwalten zu können.
Bei der Implementierung wurde sich auf eine reine FeathersJS Kompatibilität beschränkt. Wobei auch dieser Standard nur rudimentär abgedeckt wurde. Dieses Filtermodul wurde seitdem in kleinen Schritten an die Bedürfnisse der Schul-Cloud angepasst und wird seit einiger Zeit im Verwaltungs- und Aufgabenbereich der Schul-Cloud erfolgreich genutzt.

Die Architektur war dabei von Beginn an so Modular, dass das Hinzufügen von neuen Filtern mit nur wenigen Zeilen Code möglich ist[20]. Dies wurde ermöglicht, indem jeder Filter eine einheitliche Schnittstelle zur Kommunikation bereitstellen musste.

Natürlich gibt es aber auch Kritikpunkte an der aktuellen Implementierung.

So ist es beispielsweise unmöglich das Design der Filter an das Projekt anzupassen ohne den Quellcode direkt manipulieren zu müssen.

Zudem ist das Bundle mit über 300kb unverhältnismäßig groß und erhöht massiv die initiale Seitenladezeit. Dies kann zwar mit effektiven Cachingmethoden reduziert werden, die schiere Menge an Code sorgt aber dennoch auf schwachen Endgeräten zu erhöhter Time to Interactive (TTI)[1]. Das reine parsen des Codes benötigt laut Lighthouse Report[2] bereits über 500ms auf einem durchschnittlichem Mobilgerät. Dies resultiert in einer gesamt TTI im aktuellen Lern-Store von ungefähr 7s. Das Ziel sollten jedoch unter 5s sein[34].

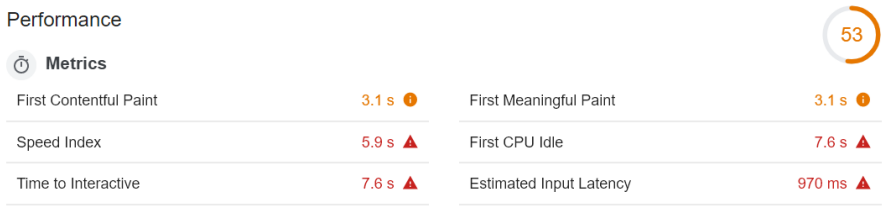

Abbildung 9: Aktueller Lern-Store Lighthouse Performance Score

Ein weiterer Kritikpunkt ist die starke Bindung an FeathersJS, wodurch ein Architekturwechsel nur schwer möglich ist, ohne das Plugin gänzlich neu zu entwickeln.

Des Weiteren ist auch das setzen von vordefinierten Filtern recht umständlich gelöst. Dieses Feature kann für den Nutzer jedoch sehr hilfreich sein. Beispielsweise um nach dem Import von Daten diese direkt gefiltert in der Übersicht anzeigen zu können.

Bei der Einbindung des bestehenden Filtermoduls in das neue Dashboard fehlten uns einige Filterarten. Beispielsweise fehlte eine Freitexteingabe. Das Design der Filter passt ebenfalls nicht zu dem neuen Dashboard. Die bestehenden Filter waren dabei zu restriktiv für einzelne Use-Cases definiert, wodurch eine Erweiterung um neue Filterarten zu viel redundantem Code geführt hätte. Ein Beispiel für solch restriktiven Code ist der `limit`-Filter[3].

Aufgrund all dieser Probleme konnten wir das bestehende Filtermodul nicht übernehmen und benötigten eine Alternative.

6.2 ANFORDERUNGEN

Ziel von uns ist es, die bestehenden Filter insoweit anzupassen, dass das Modul Plattformagnostischer wird und zudem das Design besser anpassbar ist. Gleichzeitig soll das programmatische Anpassen von Filtern erleichtert werden, indem die ausgewählten Filter anhand des

1 https://docs.google.com/document/d/1GGiI9-7KeY3TPqS3YT271upUVimo-XiL5mwWorDUD4c/preview
2 https://developers.google.com/web/tools/lighthouse/
3 https://github.com/schul-cloud/schulcloud-feathers-filter-module#limit

aktuellen Queries ausgelesen werden. Ziel ist es dabei nicht, Adapter für andere Datenbanktypen bereitzustellen, sondern lediglich die Architektur für eine solche Anpassung zu schaffen.

6.3 BESTEHENDE SYSTEME

Tatsächlich gibt es schon eine Menge an UI-Tools welche eine Grafische Oberfläche für das Bearbeiten von Queries bereitstellen. Diese haben jedoch allesamt das Problem, dass der Fokus in der Entwicklung scheinbar auf vollständiger Sprachmächtigkeit lag, was darin resultiert, dass das UI zwar für einen Datenbankadministrator sehr gut funktioniert, für einen weniger Technik affinen Endnutzer jedoch überfordernd und nicht zielführend ist. Es findet zudem keinerlei Abstraktion des zugrundeliegenden Datenschemas statt, was das Entwerfen von Anfragen für eine Systemfremde Person nahezu unmöglich macht.

Beispielhaft sei hier ein Screenshot einer React Implementierung erwähnt:

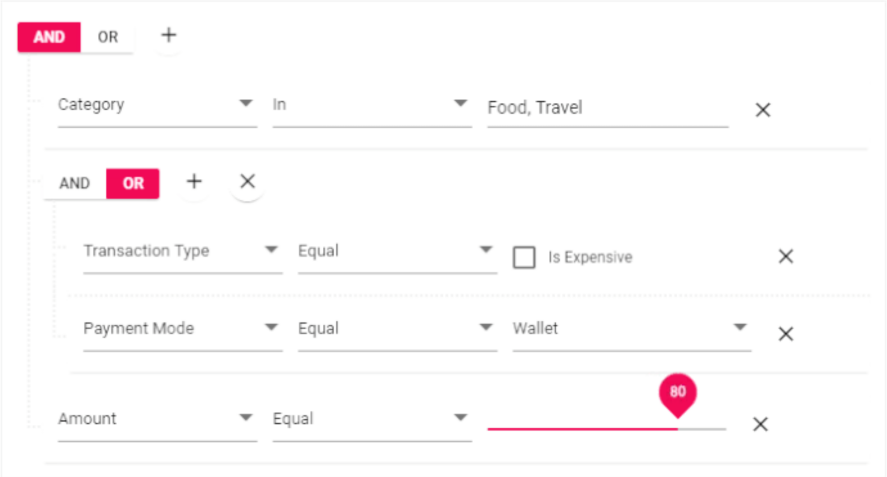

Abbildung 10: React Query Builder [28]

6.4 UMSETZUNG

Anhand der gesetzten Ziele lässt sich herleiten, dass es eine Schnittstelle für Parser geben muss. Dieser Parser soll sowohl die Konvertierung von Queries in einen Filterstatus als auch das Konvertieren eines Filterstatus in ein neues Query unterstützen. Damit einher geht eine abstrakte Filter-Syntax, welche alle möglichen Queries abdeckt und als Grundlage für die Parser dient.
Zusätzlich sollte die Benutzeroberfläche Slots[4] mit einer einheitlichen API bereitstellen, um das verwendete (HTML-)Markup an den Einbindungsort anpassen zu können.

4 https://vuejs.org/v2/guide/components-slots.html

Die Konfiguration, welche Filter verfügbar sind, soll über ein JS-Objekt geschehen.

6.4.1 *Konzepte für eine anpassbare Benutzeroberfläche*

Wir analysieren die Umsetzung einer modularen Benutzeroberfläche am Beispiel des Dialogs. Der Dialog soll als Rahmen dienen, in welchem sämtliche Eingaben des Nutzers geschehen. Dazu muss der angepasste Dialog logischerweise Zugriff auf die Daten haben, welche im Dialog angezeigt werden sollen. Die benötigten Daten werden jedoch aus der Filterkonfiguration abgeleitet. Das bedeutet, sie werden erst im Filtermodul selbst generiert und sind von außen nicht direkt Zugreifbar. Nun gibt es zwei mögliche Ansätze dieses Problem zu lösen.

6.4.1.1 *Ansatz 1 - Slots*

Ansatz eins ist es, die Daten dem Elternelement, welches den Dialog definiert bereitzustellen.
Glücklicherweise bietet Vue.js mit Scoped Slots[5][6] genau dies an.

```
<!-- Filter -->
<template>
  <slot name="dialog" v-bind:data="dialogBodyContent">
    Default Dialog: {{dialogBodyData}}
  </slot>
</template>

<!-- User -->
<template>
  <filter>
    <template v-slot:dialog="{data}">
      Custom Dialog: {{data}}
    </template>
  </filter>
</template>
```

Listing 5: Custom Dialogs - Scoped Slots Beispiel

Nun ist es aber so, dass gegebenenfalls für den eigentlichen Dialog Body das Standarddesign genutzt werden soll. Also vordefinierte Inputs oder Layouts verwendet werden sollen. Diese sollten bei Verwendung eines eigenen Dialogs nicht neu Implementiert werden müssen.

5 https://vuejs.org/v2/guide/components-slots.html#Scoped-Slots
6 https://fdietz.github.io/2018/09/13/vue-js-component-composition-with-scoped-slots.html

Die in der Kindkomponente definierten Vue.js Komponenten sind im Elternelement nicht definiert, weshalb es nicht möglich ist, im Elternelement mittels `<component :is="DialogBody"/>` die Komponente `DialogBody` einzubinden.

Wenn man jedoch eine Vue.js Komponente importiert, so ist diese zunächst einmal nur ein JS-Objekt. Dieses Objekt kann über die scoped-Slots der Elternkomponente bereitgestellt werden. Die Elternkomponente kann anschließend über `<component :is="componentObject"/>` die entsprechende Komponente darstellen.

Wird dieser Ansatz genutzt, so ergibt sich folgender Code:

```
<!-- Filter -->
<template>
  <slot name="dialog" v-bind:data="dialogBodyContent"
  ↪ v-bind:component="dialogBodyComponent">
    Default Dialog
    <component :is="dialogBodyComponent"
    ↪ :data="dialogBodyData"/>
  </slot>
</template>
<script>
import dialogBodyComponent from "./dialogBodyComponent"
export default {
  data() {
    return {
      dialogBodyContent: { /* ... */ },
      dialogBodyComponent,
}}}
</script>

<!-- User -->
<template>
  <filter>
    <template v-slot:dialog="{data, component}">
      Custom Dialog
      <component :is="component" :data="data"/>
    </template>
  </filter>
</template>
<script>
export default {
  components: {
    filter: () => import("./filter"),
}}
</script>
```

Listing 6: Custom Dialogs - Scoped Slots Beispiel mit Komponenten wiederverwendung

Somit es möglich, dass Design der Dialogs vollständig anzupassen, ohne mit der eigentlichen Logik in Verbindung zu kommen. Es muss lediglich an der entsprechenden Stelle der neue "virtuelle-Slot" mit den entsprechenden Daten angegeben werden.

6.4.1.2 *Ansatz 2 - Component Props*

Der zweite mögliche Ansatz ist, die zu verwendenden Komponenten im ganzen an die Kindkomponente (Filter Modul) zu übergeben.

Die Filterkomponente muss dafür lediglich entsprechende Attribute (`props`) bereitstellen. Mit diesem Ansatz wird die Implementierung der eigenen UI Module weiter von der Einbindung gekapselt. Die Entwicklung der eigenen Komponenten muss in separaten Komponenten stattfinden. Dadurch wird teilen von Code begünstigt und hoffentlich zukünftige Entwicklungsresourcen gespart.
Bei der Implementierung der eigenen Komponenten muss bei diesem Ansatz zudem weniger über die eigentliche Implementierung des Dialoginhalts bekannt sein. Es reicht, wenn der Dialog einen Slot bereitstellt, in welchen der Dialoginhalt gerendert werden kann.

```
1  <!-- Filter -->
2  <template>
3    <component :is="componentDialog" @apply="..."
     ↪  @cancle="...">
4      Body Content
5    </component>
6  </template>
7  <script>
8  import componentDefaultDialog from
   ↪  "./componentDefaultDialog"
9  export default {
10   props: {
11     componentDialog: {
12       default: () => componentDefaultDialog
13 }}}
14 </script>
15
16 <!-- User -->
17 <template>
18   <filter :componentDialog="customDialog" />
19 </template>
20 <script>
21 import customDialog from "./customDialog"
22 export default {
23   components: {
24     filter: () => import("./filter"),
25   },
26   data(){
27     return {
28       customDialog
29 }}}
30 </script>
```

Listing 7: Custom Dialogs - Beispiel mit Komponenten-Props

Ansatz zwei hat keine relevanten Nachteile gegenüber dem ersten Ansatz. Die Lesbarkeit und Modularität des Codes wird mit diesem

Ansatz jedoch weiter verbessert, da der syntaktische Overhead und das notwendige Domänenwissen reduziert wird. Wir entschieden uns daher für die Implementierung von Ansatz 2.

6.4.2 *Anpassbare Eingabefelder*

Vorrangegangen wurde bereits gezeigt, wie sich der Dialog austauschen lässt. Für die eigentlichen Eingabeelemente kann dieses Konzept übernommen werden, wobei jedoch die Komponente nicht als Attribut auf Root-Ebene übergeben wird, sondern im Konfigurationsobjekt als eine Eigenschaft der Eingabeelemente. Die Komponente muss erneut zwingend eine festgelegte API implementieren. Diese Beschränkt sich auf ein `v-model` (zwingend erforderlich), welches stets die aktuelle Eingabe enthält, und ein Array an `options`, welche die Auswahlmöglichkeiten definiert. Dabei ist das `options`-Attribut jedoch optional.

6.4.3 *Anpassbare Layouts*

Neben einfachen Eingabefeldern, kann ein Filter auch aus einer Kombination von Eingaben bestehen. Beispielsweise ist die Sortierung von Elementen über zwei Eigenschaften definiert. Einerseits muss die Eigenschaft nach der sortiert werden soll angegeben werden, andererseits wird die Information benötigt, ob ab- oder aufsteigend sortiert werden soll.
Aus diesem Grund haben wir jeden Filter als ein Layout mit mehreren Eingabefeldern definiert. Damit es möglich ist die Eingabefelder frei anzuordnen müssen auch die Layoutkomponente austauschbar sein. Dies ist möglich, indem die Layoutkomponenten Slots nach einem festen Namensschema anbieten. Eingabefeld 1 wird stets in den Slot mit dem Namen `input-1` gerendert, Eingabefeld 2 in den Slot `input-2` usw.. Der Ersteller der Layoutkomponenten hat dabei die volle Kontrolle darüber, wo, welches Eingabefeld platziert werden soll.
Dabei ist es auch möglich, Slots dynamisch zu generieren. Auf diese Art und Weise können generische Layouts definiert werden, welche beliebig viele Eingabefelder akzeptieren können.

```
<template>
  <section>
    <div
      v-for="(value, key, index) in $scopedSlots"
      :key="key"
      class="slot-wrapper"
    >
      <slot :name="key" :class="`slot-${index}`" />
    </div>
  </section>
</template>
```

Listing 8: Custom Layouts - Beliebig viele Slots anbieten

6.4.4 *Datenstruktur*

Um die Filter zu konfigurieren, benötigt es einer Datenstruktur, welche die nötigen Information für die Parser bereitstellen kann. Gleichzeitig muss in der Konfiguration auch angegeben werden können, welche Komponenten wo genutzt werden sollen.

Wie bereits weiter oben definiert, besteht ein Filter stets aus einem Layout und einer Menge an Eingabefeldern. Dabei ist jeweils die entsprechende Komponente anzugeben.
Daraus ergibt sich die folgende Grundstruktur der Konfiguration:

```
const config = [
  { // Filter 1
    layout: ComponentLayout1,
    inputs: [
      {
        input: ComponentInput1,
      }, // ... Input 2
    ]
  }, // ... Filter 2
]
```

Listing 9: Filter-Datenstruktur

In diesem Beispiel ist bereits die Komponentenzuordnung eingetragen.
Welche Eigenschaften die einzelnen Objekte zusätzlich besitzen müssen, wollen wir nun näher erläutern. Betrachten wir dabei zunächst die Oberste Ebene, auf welcher das Layout angegeben wird. Auf dieser Ebene können Eigenschaften angegeben werden, welche für die Anzeige des Filters relevant sind. Dazu zählen beispielsweise ein Titel

(`title`). Auch eine Angabe, wie der angewendete Filter dem Nutzer präsentiert werden soll kann hier stattfinden (`chipTemplate`). Dies kann dabei sowohl mittels Template-String[7] als auch per Funktion definiert werden. Dadurch ermöglichen wir es, auch komplexe Kombinationen von Eingabefeldern in einfache, für den Endnutzer verständliche, Aussagen konvertieren zu können. Eine der wichtigsten Attribute ist jedoch `parser`. Über dieses Attribut kann für einen einzelnen Filter ein eigener Parser angegeben werden. Dies ist immer dann hilfreich, wenn Eingabefelder auf komplexere Art und Weise zu einem Query zusammengesetzt werden sollen.
Für alle weitere Eigenschaften von Layouts möchten wir auf die Projektdokumentation[8] verweisen.

Bei den Eigenschaften der Eingabefelder wird es nun interessanter. Neben der obligatorischen Angabe, welche Komponente genutzt werden soll, liegen hier die entsprechenden Informationen für den Parser, wie dieser das Eingabefeld konvertieren soll. Dabei ist es offensichtlich notwendig, dass entsprechende Datenfeld anzugeben, nach welchem gefiltert werden soll. Des weiteren muss der Operator angegeben werden, wie das Datenfeld im Query mit dem Eingabewert verglichen werden soll. Dabei beschränkten wir uns auf die folgenden Operatoren: `>`, `>=`, `=` und `includes`. Zusätzlich führen wir die Eigenschaft `applyNegated` ein, welche den Operator negiert. Mit der Kombination aus diesen Operatoren sind die meisten Anwendungsfälle abgedeckt. Für alles weitere kann der Parser auf Filterebene erweitert werden, sodass durch die Beschränkung der Operatoren keine Beschränkung des Query Mächtigkeit stattfindet.

6.4.5 *Parser*

Die notwendigen Parser sind sehr spezifisch für die verschiedenen Datenbanktypen zu entwickeln. Daher möchten wir in dieser Arbeit nicht näher auf die Implementierung dieser eingehen. Zu erwähnen ist es dennoch, dass jeder Parser aus zwei Teilen besteht. Ein Teil konvertiert die Eingaben des Nutzers unter Berücksichtigung der Konfiguration in ein entsprechendes Datenbankquery. Der andere Teil implementiert die entgegengesetzte Richtung. Letzterer Teil ist wichtig, um das setzen von vordefinierten Filtern zu vereinfachen.
Des weiteren muss jeder Parser pro Gruppe eine Schnittstelle bereitstellen, an welcher der Parser die vom Nutzer in der Konfiguration definierten Parsing-Funktionen[9] ausführt.

7 https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/String/replace#Eine_Zeichenkette_als_Parameter_angeben
8 http://docs.vue-filter-ui.surge.sh/2-Configuration.html#filter
9 https://en.wikipedia.org/wiki/Parsing

Zusammenfassung

Damit sei die Architektur geschaffen um dem Nutzer an verschiedenen Stellen der Schul-Cloud als auch dem Lern-Store das Filtern von Inhalten zu ermöglichen. Dabei kann das Design leicht an das Projekt angepasst werden. In den Grafiken 26 und 27 ist die Einbindung der neuen Filter in das Content-Dashboard abgebildet. Dabei werden die für das Dashboard entwickelten Eingabefelder wiederverwendet. Da mit der neuen Implementierung keine externe Bibliothek mehr verwendet wird, konnte die Bundlegröße auf weniger als 100kB reduziert werden, was die Performance massiv verbessert.

7

MASSENBEARBEITUNG VON METADATEN

Adrian Jost

7.1 MOTIVATION

Wird das vorgestellte Dashboard über längere Zeit hinweg genutzt, so kann es durchaus vorkommen, dass verschiedene Metadaten angepasst werden müssen. Dies sind meist kleine Korrekturen von beispielsweise Rechtschreibfehlern. Es kommt aber auch vor, dass große Anpassungen vorgenommen werden müssen. Ein Beispiel dafür ist der Domain-Umzug von gehosteten Inhalten. In diesem Fall kann es vorkommen, dass die Inhalte zwar bereits auf eine neue Adresse umgezogen sind, die Einträge im Lern-Store aber noch nicht angepasst worden sind. In diesem Fall sind die Inhalte für einen gewissen Zeitraum nicht erreichbar, was zu Frustration beim Nutzer und Imageschaden des Inhalteanbieters führen kann.
Eine Möglichkeit für die nachträglichen (Massen)-Bearbeitung von Metadaten ist daher notwendig.

7.2 ANFORDERUNGEN

Das Tool soll die Bearbeitung einzelner sowie mehrerer Einträge auf einmal erlauben. Dabei soll neben dem vollständigem Ersetzen von Attributen auch eine partielle Ersetzung oder Erweiterung möglich sein. Dies soll von einer Übersichtlichen Darstellungsform unterstützt werden, auf welcher Datenfelder schnell geprüft und angepasst werden können.

7.3 BESTEHENDE SYSTEME

Die Massenbearbeitung von Inhalten ist kein neues Problem und wurde bereits vielfach gelöst. Dabei wurde jedoch meist eine Anpassung an die entsprechenden Daten und Anforderungen vorgenommen, weshalb es keine Lösung gibt, welche direkt zu übernehmen geht. Es sollte sich bei der Umsetzung dennoch stets an bestehenden

Ansätzen orientiert werden, damit der Nutzer den Umgang mit dem System nicht von Grund auf neu erlernen muss.

Betrachtet man das Content-Management-System Wordpress[1], so unterstützt dieses System sowohl eine Schnellbearbeitungsfunktion in der Übersicht, als auch eine Massenbearbeitung der eingestellten Artikel.
Zum schnellen bearbeiten der Daten sind viele Texte in Form von Eingabefeldern dargestellt. Auf diese Weise kann das Ändern von Werten direkt in der Übersicht erfolgen.
Bei der Massenbearbeitung wird diese Ansicht um eine Zeile mit den zu bearbeitbaren Werten ergänzt. In dieser können Werte eingetragen werden, welche auf alle aktuell sichtbaren Artikel angewendet werden sollen.
Dabei ist die Auswahl der bearbeitbaren Werte jedoch eingeschränkt und eine Funktion zum partiellen ersetzen fehlt.

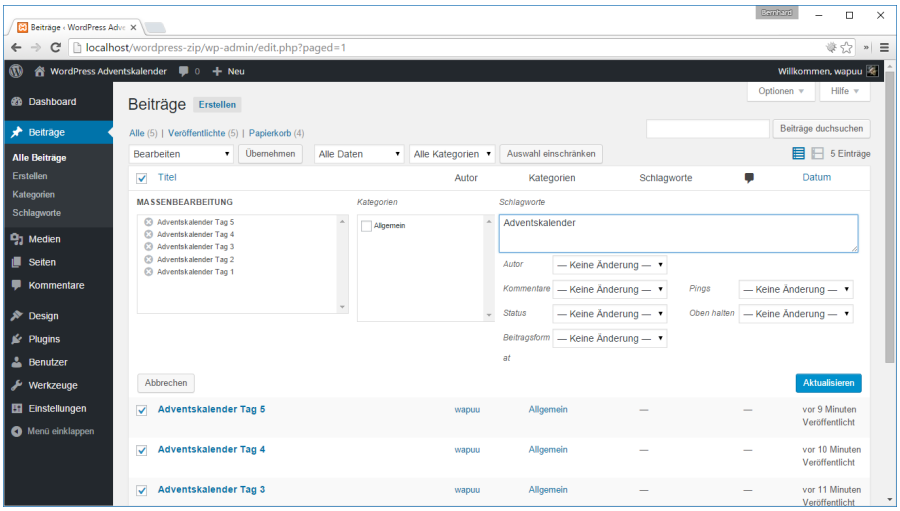

Abbildung 11: Wordpress Massenbearbeitung [21]

Ein anderer Ansatz zur Massenbearbeitung wird beispielsweise von Google Mail[2], phpMyAdmin[3] oder Nextcloud[4] verwendet. Dieser Ansatz ist Aktionsbasiert. In der Übersicht können die Inhalte ausgewählt werden, auf welche eine Aktion angewendet werden soll. Anschließend kann, aus einer Liste, eine Aktionen ausgewählt werden, welche auf die Inhalte angewendet wird. Die Aktion kann dabei teils in einem sich öffnenden Dialog weiter spezifiziert werden.

1 https://de.wordpress.org
2 https://mail.google.com
3 https://www.phpmyadmin.net/
4 https://nextcloud.com

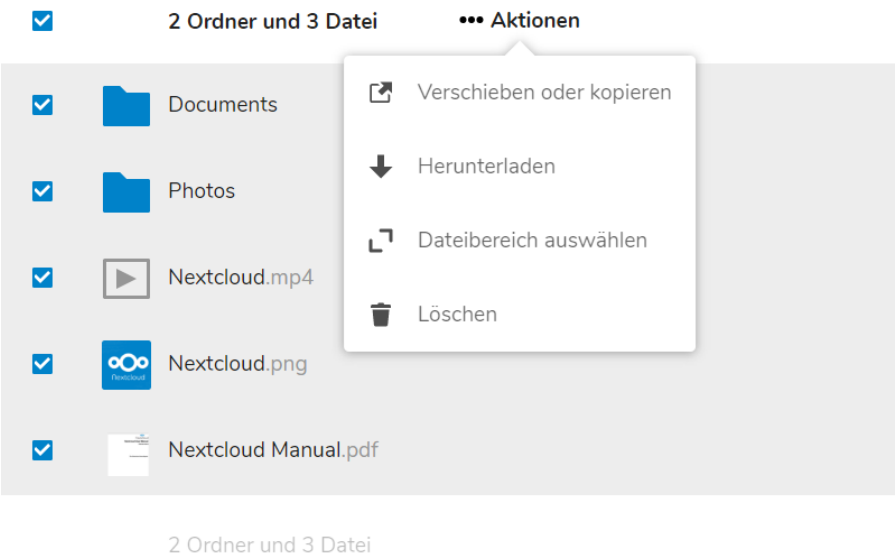

Abbildung 12: Nextcloud - Aktionsbasierte Massenbearbeitung

Dieser Ansatz ist für den Lern-Store jedoch weniger passend, da nur wenige Änderungen mit klaren Aktionen beschreibbar sind. Lediglich für einfache Aktionen wie das veröffentlichen eines Inhaltes halten wir diese Herangehensweise für Umsetzbar.
Wir haben uns daher dazu entschieden einen Wordpress ähnlichen Ansatz umzusetzen.

7.4 UMSETZUNG

Bei der Umsetzung müssen drei Entwurfsentscheidungen getroffen werden.
Zuerst wird wegen der Komplexität der Massenbearbeitung ein Konzept für eine intuitive Benutzeroberfläche benötigt. Dabei muss auch die vorhandene Query-Syntax für den Anwendungsfall angepasst werden. Aus der Anpassung der Query-Syntax folgt, dass auch der Server entsprechend angepasst werden muss.

7.4.1 *Entwurf des Userinterfaces*

Da unsere Zielgruppe weniger technisch versiert ist und wir die Einstiegshürde möglichst gering halten möchten, entscheiden wir, unser Interface an bestehende Tabellenkalkulationsprogramme anzupassen. Diese Form der Datenverarbeitung ist meist bereits aus dem Alltag bekannt und hat sich als effiziente und übersichtliche Lösung bewährt.

Die Hauptansicht ist somit eine Tabelle, in welcher die Daten direkt bearbeitet werden können. Jede Zeile repräsentiert dabei einen Lerninhalt.

Um das bearbeiten von einzelnen Inhalten zu ermöglichen, haben wir jede Zeile um eine Spalte "Actions" erweitert, in welcher der Nutzer seine Änderungen auf den Inhalt anwenden kann. Im Gegensatz zu globalen Actions, wie ihn MacOS, Google Drive und Google Mail nutzen, berücksichtigt unser Ansatz das Prinzip der Nähe[5]. Dadurch ist für den Nutzer klarer ersichtlich, auf welche Daten die Aktion angewendet wird[18]. Dies schafft Sicherheit, was gerade bei kritischen Aufgaben wie dem löschen von Daten wichtig ist.

Abbildung 13: Inline Datenbearbeitung

Die Option zur Bearbeitung von mehreren Inhalten ist ebenfalls in die Tabelle integriert. Dadurch muss der Nutzer kein zusätzliches UI für eine semantisch ähnliche Aufgabe erlernen.
Dabei nehmen wir an, dass der komplexere Anwendungsfall "Suchen & Ersetzen" seltener als der einfache "Wert Ersetzen" Fall auftritt. Daher haben wir das UI in 2 Modi aufgeteilt.
Modus 1 ermöglich lediglich, Attribute im gesamten zu überschreiben. Dies findet Anwendung, wenn die Lizenz von Inhalten ausgetauscht werden soll.
Zusätzlich gibt es den erweiterten Modus um nur Teile der Attribute zu verändern. Dieser erweitert die Tabelle um einen inline-Suche in den jeweiligen Attributspalten. Ist dieses Suchfeld aktiv, so wird in allen Attributen nur jener Teil ersetzt, auf welchen der Suchbegriff zutrifft.

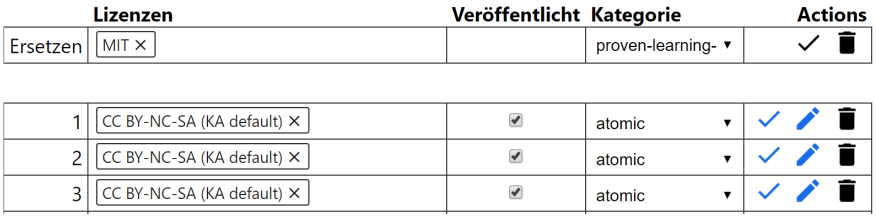

Abbildung 14: Einfacher Massenbearbeitungsmodus

Bei der Überprüfung des UIs gegen die Anforderungskriterien stellt sich ein weiteres Problem heraus. Die Handhabung von leeren Eingabewerten ist mit der bisherigen Oberfläche nicht klar definiert. Für eine leere Eingabe gibt es zwei mögliche Intentionen des Nutzers.
Möglichkeit 1: Der Nutzer hat die Eingabe aktiv leer gelassen hat, um den Wert in der entsprechenden Spalte unverändert zu lassen.
Möglichkeit 2: Es wurde kein Wert eingetragen um alle betreffenden Werte zu entfernen.
Die Intention muss daher von vom Nutzer erfragt werden. Dies kann auf zwei Arten umgesetzt werden.

Eine Variante ist es, dem Nutzer beim Abspeichern mit einem Dialog nach seiner Intention zu allen unklaren Feldern zu befragen.

5 http://www.kommdesign.de/texte/gestaltpsychologie1.htm#naehe

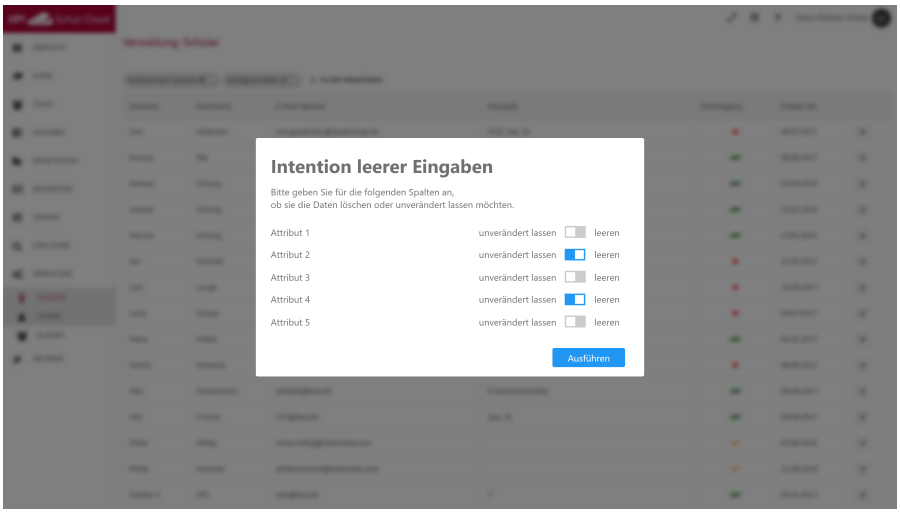

Abbildung 15: Auswahldialog - Intention leerer Eingaben

Dies hat den Vorteil, dass wir dem Nutzer den Grund der Nachfrage detailliert erklären können. Der Nutzer muss somit weniger über das System wissen und erhält an kritischen Stellen Unterstützung.
Es entsteht jedoch der Nachteil, dass bei Änderungen in nur wenigen Spalten den Nutzer stets zu sehr vielen Eingaben befragt werden muss. Zudem ist auch dem Nutzer unklar, wie wir mit den Eingabefeldern umgehen, bis er sich schließlich traut auf "Speichern" zu klicken und der Dialog erscheint. Dieses Gefühl der Unsicherheit wollen wir vermeiden.

Wir haben uns daher für Möglichkeit zwei entschieden, welche dem Nutzer an jeder Eingabe mit einer Checkbox wählen lässt, was seine Intention ist. Dies hat den Vorteil, dass der Nutzer nie im unklaren gelassen wird, welche Aktion er beim Speichern auslösen wird. Er kann direkt im eigentlichen Bearbeitungsmodus die Auswahl treffen. Dies reduziert Interaktionstiefe um eine Ebene und schafft Sicherheit.

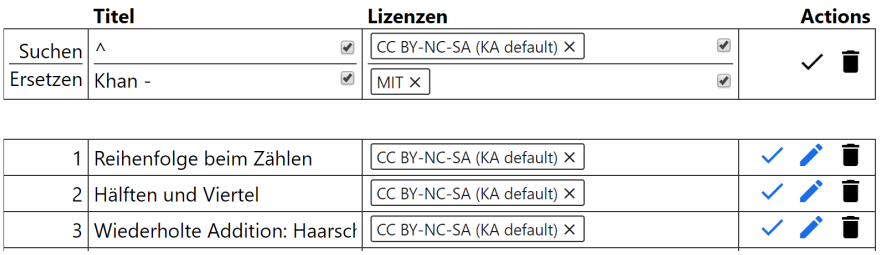

Abbildung 16: Erweiterter Massenbearbeitungsmodus

Ein Problem dieser Ansatzes ist es, dass der Nutzer stets eine zusätzlich Angabe machen muss. Daher haben wir uns entschieden Anhand der Eingabe des Nutzers eine Vorauswahl zu setzen. Diese kann jedoch jederzeit vom Nutzer überschrieben werden.

Das automatisierte Setzen der Checkbox soll dabei auf Eingaben nach folgenden Regeln reagieren:

Eingabe (alt)	Eingabe (neu)	Checkbox (alt)	resultierender Checkbox Status
✗	✓	✗	✓
✗	✓	✓	✓
✓	✗	✗	✗
✓	✗	✓	✗

7.4.2 *Aufbau des Queries*

Zu klären ist weiterhin, wie die eigentliche Serveranfrage zum bearbeiten von mehreren Werten auf einmal aussieht. Es stellt sich heraus, dass die vorhandenen Technologien MongoDB und FeathersJS bereits eine Option zum Bearbeiten von vielen Werten auf einmal anbieten. Dabei werden alle CRUD Anfragetypen[6] unterstützt[3]. Für das erstellen von Inhalten kann statt einem einzelnen Eintrag einfach eine Liste an neuen Einträgen gesendet werden.
Bei der Bearbeitung und dem Löschen von Inhalten muss statt einer ID des zu bearbeitenden Wertes ein Query gesendet werden. Die Anfrage wird dabei auf alle Werte angewendet, auf welche das Query zutrifft.

Das Query könnte dabei nach folgenden Schemata aufgebaut werden:

1. Die Anfrage besteht aus einer Liste an IDs (`$in[_id]:[...]`). Diese müssten, aufgrund der verwendeten Pagination [7] bei der Darstellung der Tabelle, mit einer zusätzlichen Anfrage ausgelesen werden.
2. Das vom Nutzer, für die aktuelle Ansicht verwendete, Search Query wiederzuverwenden
3. Eine Kombination aus den ersten beiden Ansätzen. Dabei wird das bestehende Query durch eine eine Liste an ausgeschlossenen Werten erweitert. Auf diese Weise kann dem Nutzer angeboten werden, Einträge von der Massenbearbeitung auszuschließen.

Idee 3 ist dabei am besten für unsere Anwendung geeignet, da sie die wenigsten Netzwerkanfragen benötigt um die Anforderungen zu erfüllen. Sämtliche notwendigen Information sind bereits aus der vorangegangenen Anfrage bekannt, welche für die Darstellung der Suchergebnisse getätigt wurde.

Mit diesem Ansatz entsteht jedoch ein weiteres Problem. Für die Suche in Lerninhalten wird von der Schul-Cloud Elasticsearch verwendet, welche das Query um zusätzliche Optionale Attribute erweitert.

6 https://glossar.hs-augsburg.de/CRUD
7 https://www.tonymarston.net/php-mysql/pagination.html

Eines dieser Attribute lautet `$match[_all]: "..."`. MongoDB und FeathersJS verstehen dieses Attribut jedoch nicht nativ, weshalb sie es bei der Anfrageverarbeitung ignorieren. Dies hat zur Folge, dass das Query auf viel mehr Werte zutrifft als bei der Suche ausgegeben werden.

Aufgrund dieses Problems ist ein Umschwenken auf den ersten Ansatz erforderlich.
Zunächst wird eine Anfrage ausgeführt, welches die IDs zu allen Einträgen ausliest. Dazu muss lediglich das vom Nutzer generierte Query um zwei Eigenschaften erweitert werden.
Einerseits muss die Pagination des Servers ausgeschalten werden, des weiteren sollte die zu übertragenden Datenfelder auf die reine ID des Inhalts beschränkt werden.

```
const query = {
  ...searchQuery,   // pass existing user generated query
  $limit: '-1',     // disable limit
  $select: ['_id'], // only get id from resources
}
// GET http://content-server/search/?{query}
```

Listing 10: FeathersJS Query zum Anfragen aller IDs

Anschließend wird aus den erhaltenen IDs ein neues Query gebaut, welches für die Eigentliche Massenbearbeitung genutzt wird. Dieses Query besteht nur noch aus einer Überprüfung der IDs

```
const ids = find("http://content-server/search/?query");
const query = {
  "$in[_id]": ids,
}
// PATCH http://content-server/resources/?{query}
```

Listing 11: FeathersJS Query zur Anwendung auf Menge an IDs

Leider hat auch dieser Ansatz ein gravierendes Problem. Bei der REST Schnittstelle über welche wir mit dem Backend kommunizieren wird das Query kodiert über die URL mit gesendet. Bei hinreichend großen Datenmengen (>2000 Einträge) wird die URL dadurch so lang, dass die von FeathersJS definierte maximale URL-Länge überschritten wird.

Dieses Problem lässt sich auf drei Wegen lösen.
Option 1 ist, das Limit im Server zu erhöhen.
Diese Option ist am einfachsten umzusetzen. Sie ignoriert jedoch die Tatsache, dass nicht nur der Server ein URL-Limit hat. Im Gegensatz

zur Spezifikation einer URL nach RFC2616[4] implementieren nämlich auch Browser eine URL Längenbeschränkung[14]. Auf dieses hat eine Website keinen Einfluss, weshalb es früher oder später hier zu einem Problem kommen würde.

Wir könnten den Server allerdings auch auf Websockets[8] umstellen. Bei der Kommunikation über Websockets besteht keine Längenbeschränkung, da das Query als Datenfeld gesendet wird[25].
Die Option ist für uns aber ebenfalls nicht praktikabel, da sie eine Architekturumstellung des Backends und der Clients zur Folge hätte.

Die dritte Option ist, die Logik für das Sammeln der IDs in den Server zu verlagern. Für Anfragen innerhalb des Servers besteht das URL-Längenproblem ebenfalls nicht. Dadurch kann ein Service geschrieben, welcher die eben beschriebene Logik problemfrei serverseitig umsetzen kann.
Auf die genaue Umsetzung wollen wir im nächsten Kapitel eingehen. Zuvor muss das Query aber so zu erweitert werden, dass auch die Suchen & Ersetzen Funktion abgedeckt wird.

Mit dem bis hier vorgestellten Ansatz ist es bereits möglich, die Werte von Datensätzen vollständig zu überschreiben. Es fehlt noch eine Möglichkeit, nur Teile der Datenfelder zu überschreiben. Dazu muss das Query um eine Information erweitert werden, welche für jedes Datenfeld beschreibt, welcher Teil dessen ersetzt werden soll.
Dies wird von den verwendeten Technologien nicht nativ unterstützt, weshalb die Querysyntax um eigene Befehlssätze erweitert werden muss. Dieses Attribut nennen wir `$replace` und ist syntaktisch ähnlich zu dem von Elasticsearch bereitgestellten `$match` Parameter[9] aufgebaut.

```
const query = {
  $replace: {
    title: "Khan",
    tags: "Klasse-4",
  }
}

// QueryString =>
↪  /batch/?$replace[title]=Khan&$replace[tags]=Klasse-4
```

Listing 12: Beispiel der neuen $replace Syntax

Im neuen `$replace`-Attribut wird für jedes Datenfeld ein Suchbegriff angegeben. Bei der Bearbeitungsanfrage soll nur der dort angegebene Teil der vorhandenen Daten überschrieben werden.

8 https://www.html5rocks.com/de/tutorials/websockets/basics/
9 https://www.elastic.co/guide/en/elasticsearch/reference/5.5/query-dsl-match-query.html

Wird dabei für ein Datenfeld ein leerer Wert angegeben, so ist dies als "ersetze *nichts* mit *x*" zu Interpretieren. Dadurch können vorhandene Werte leicht um weitere ergänzt werden.

Mit diesem einfachen Suchen & Ersetzen Ansatz ist jedoch noch nicht jeder Anwendungsfall abgedeckt. Beispielsweise ist es unmöglich, eine exakte Stelle zum ersetzen anzugeben um ein Präfix vor allen Werten anzufügen. Daher soll im `$replace` Parameter auch alternativ zu einem String ein Regulärer Ausdruck[10] angegeben werden können. Die Matches (1-n) sollen dabei als `$n` im neuen String verfügbar sein.

Mit diesen Erweiterungen der Querysyntax sind sämtliche spezifizierten Anwendungsfälle abgedeckt. Als Query für die Massenbearbeitung kann einfach das vorhandene Query weiterverwendet werden, welches gegebenenfalls um die `$replace`-Syntax erweitert wurde. Die weitere Interpretation und Konvertierung zu einem ID-Query erfolgt im Server.

7.4.3 *Server Implementierung*

Wie bereits im Abschnitt Query angesprochen, unterstützt FeathersJS bereits nativ das bearbeiten von mehreren Werten. Die Performance ist dabei nahezu identisch zum Manipulieren eines einzelnen Datensatzes. Diesen Performancevorteil wollen wir mit unsere neuen Features bestmöglich ausnutzen und wann immer es geht darauf zurückgreifen.

Die wichtigste Architekturentscheidung ist dabei, ob ein eigener Service für das bearbeiten von mehreren Werten entwickelt wird, oder die neue Syntax mit Hooks im vorhandenen Ressource-Service[11] ergänzt wird.
Das erweitern mit Hooks erscheint semantisch am sinnvollsten, da lediglich vorhandene Funktionalität erweitert, beziehungsweise an die neuen Anforderungen angepasst wird.
Dennoch entschieden wir uns für einen eigenen Service, da dies eine sauberere Architektur ermöglicht. Gegenebenfalls werden, für das Bearbeiten von mehreren Werten, andere Rechteprüfungen benötigt. Dies würde die Hooks im bestehenden Service stark vergrößern und zu Unübersichtlichkeit führen.
Ein weiteres Gegenargument ist, dass wir die Anfrage eventuell in Chunks aufsplitten müssen um Größenbeschränkungen einhalten zu können. Dies resultiert bei einer reinen Hook Lösung in schwer zu kontrollierenden Rekursiven Datenbankaufrufen.

Die Entwicklung eines separaten Services ist daher die sauberere Lösung. Der neue Service implementiert das gesamte CRUD Interface und leitet gegebenenfalls die Anfrage direkt an den bestehenden

10 https://www.regexbuddy.com/regex.html
11 https://github.com/schul-cloud/schulcloud-content/blob/master/src/services/resource/resource.service.js

Ressourcen-Service weiter. Auf diese Weise kann der Performancevorteile der Nativen Implementierung ausgenutzten werden.
Es wird dabei die im vorherigen Kapitel beschriebene Ansatz zur Generierung des Queries durch eine zusätzliche GET Abfrage implementiert. Aufgrund dessen dass es sich nun um einen Serverinternen Aufruf handelt gibt es die zuvor beschriebenen URL-Längenlimit Probleme kaum noch. Lediglich Elasticsearch limitiert die Anzahl der gleichzeitig bearbeitbaren Werte aktuell noch auf etwas über 10.000 Werte. Dies ist jedoch ein vertretbares Limit, da es recht unwahrscheinlich ist, dass so viel Werte mit einer Anfrage bearbeitet werden sollen. Sollte sich der Anwendungsfall in Zukunft jedoch ergeben, so könnte die Anfrage in Blöcken ausgeführt werden. Dabei würde Pagination genutzt werden.

Für die Unterstützung der neuen `$replace`-Syntax gibt es, aufgrund des fehlenden Unterstützung durch FeathersJS, leider keine andere Möglichkeit als alle Daten manuell zu aktualisieren. Dabei muss jeder Wert ausgelesen, manipuliert und anschließend in die Datenbank geschrieben werden.
Das auslesen kann dabei weiterhin mit einer einzigen Anfrage geschehen. Das Schreiben in die Datenbank ist jedoch nur gestaffelt möglich. Jeder Eintrag muss mit einer separaten Anfrage geschrieben werden. Dies verursacht einen massiven Performanceverlust. Eine Frameworkunterstützung um mehrere Werte per ID referenziert mit nur einer Anfrage zu aktualisieren ist leider nicht in Planung[26].
Wir führen diese Logik aufgrund dieses Performanceproblems daher nur aus, wenn ein `$replace` im Query angegeben wird.

Zusammenfassung

Zusammenfassend können mit dem neuen Service sowohl im ganzen, als auch partiell, Werte ersetzt werden. Anfragen können dabei weiterhin wie in der FeathersJS Dokumentation beschrieben durchgeführt werden. Die Problematik welche durch die Verwendung von Elasticsearch entstanden ist wurde durch die Serverimplementierung ausgeglichen. Die einzige Besonderheit ist die die neue `$replace`-Syntax für das partielle Ersetzen von Werten.

8

AUTHENTIFIZIERUNG UND BERECHTIGUNGSPRÜFUNG

Katharina Blaß

8.1 MOTIVATION

Ein essenzieller Bestandteil für die Nutzbarkeit unserer Plattform ist das Verwalten und Sichern der Benutzeraccounts und Routen. Jeder Inhalteanbieter soll über seinen Account auf unsere Features und seine firmeninternen Inhalte zugreifen können. Um eine missbräuchliche Nutzung durch Dritte zu vermeiden, müssen diese Accounts mithilfe geeigneter Authentifizierungsmechanismen und die Routen durch Berechtigungsprüfung geschützt werden.
Im bestehenden System wurde beim Login des Nutzers ein JSON Web Token[1] (JWT) beim Schul-Cloud Server[2] angefragt. Der Access Token wurde im Cookie und im Local Storage gespeichert und bei jeder Anfrage an den Content Server im Authorization-Header mitgesendet. Dieser prüfte vor Gewährung des Zugriffs auf die einzelnen Routen lediglich das Vorhandensein eines JWTs, ohne diesen jemals beim Schul-Cloud Server auf seine "Echtheit" zu prüfen. Im Content-Server selber bestand somit lediglich eine leicht zu umgehende Zugriffskontrolle, jedoch keine Authentifizierung (diese übernahm der Schul-Cloud Server).
Um den zukünftigen Nutzern des Content-Dashboards ein sicheres System anzubieten, müssen also sowohl die Authentifizierung, als auch die Berechtigungsprüfungen überarbeitet werden.

8.2 ANFORDERUNGEN

Das Authentifizierungsverfahren soll dem Benutzer eine Anmeldung über Benutzernamen und Passwort ermöglichen. Für den Fall, dass wir die Authentifizierung vom Schul-Cloud Server beibehalten wollen, soll der Content-Server auch einen JWT zur Anmeldung akzeptieren und verifizieren können. Darüber hinaus sollen zukünftige Ent-

1 https://medium.com/ag-grid/8076ca679843
2 https://github.com/schul-cloud/schulcloud-server

wickler zu Testzwecken auf einzelne Routen zugreifen können, ohne jedes Mal die gesamte Authentifizierung und Rechteprüfung durchlaufen zu müssen. Bisher sendet der Entwickler ein Basic-Auth Token im Authorization-Header seiner Anfrage mit. Aus diesem wird im Content-Server Nutzername und Passwort extrahiert und mit hardgecodeten Accounts aus der `Development.json`[3] verglichen. Bei Übereinstimmung ist der Entwickler berechtigt, auf die angefragte Ressource zuzugreifen. Dieses oder ein ähnliches Verfahren soll auch die neue Authentifizierung ermöglichen können.

Die Zugriffskontrolle soll alle Routen von außen vor unberechtigtem Zugriff von authentifizierten Nutzern schützen. So sollen Inhalteanbieter nur auf firmeneigene Inhalte, spätere Lern-Store-Nutzer lediglich auf freigeschaltete/öffentliche Ressourcen zugreifen können. Interne Aufrufe sollen dagegen ohne Rechteprüfung erlaubt werden, um voneinander abhängige Services oder die Tests nicht zu behindern.

8.3 ANALYSE BESTEHENDER BIBLIOTHEKEN

Authentifizierung ist in jeder Web-Anwendung essenzieller Bestandteil und wurde dementsprechend oft und gut bereits von vielen Menschen implementiert. Das Rad an dieser Stelle neu zu erfinden, ergibt wenig Sinn. Die Frage lautet daher eher, welches bereits existierende Framework am besten die oben beschriebenen Anforderungen erfüllen kann. Wie sich bei der Recherche herausgestellt hat, erfüllen fast alle, darunter auch FeathersJS Authentication[4], Passport.js[5] und Permit[6], diese Anforderungen[31] [35]. Dabei fiel unsere Wahl letztendlich auf FeathersJS Authentifizierungsmodul. FeathersJS bietet eine REST-API für Webanwendungen und wird bereits an vielen Stellen im Schul-Cloud Server und Content-Server genutzt. Somit wäre das Authentifizierungsmodul einfach zu integrieren und es gäbe genügend Expertenwissen zu FeathersJS unter den Schul-Cloud Entwicklern. Das folgende Sequenzdiagramm veranschaulicht die Arbeitsweise des Moduls.

3 https://github.com/schul-cloud/schulcloud-content/blob/master/config/development.json
4 https://docs.feathersjs.com/api/authentication/server.html
5 http://www.passportjs.org/
6 https://github.com/ianstormtaylor/permit

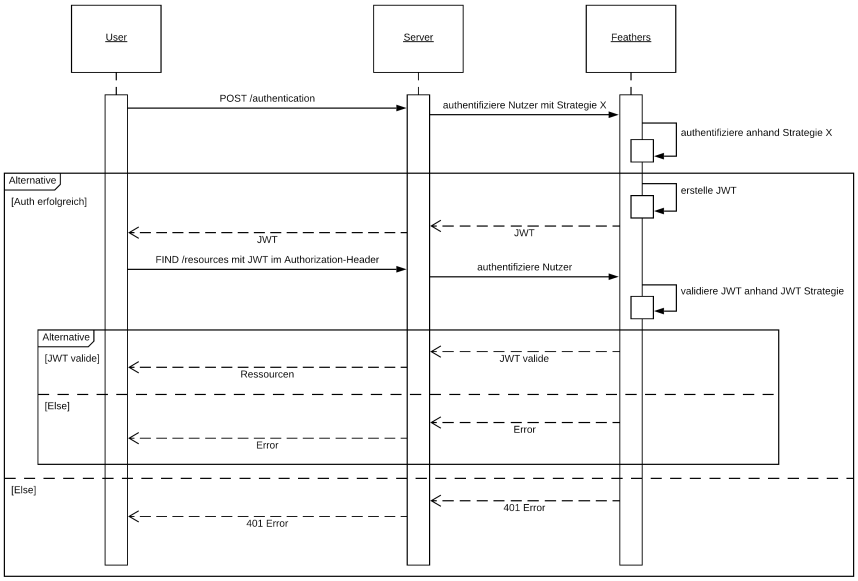

Abbildung 17: FeathersJS Authentication: Benutzer fragt Ressourcen an

FeathersJS Authentication bietet unter `/authentication` eine Schnittstelle, um Nutzer mittels einer geeigneten Strategie zu authentifizieren und JWTs zu generieren. Diese können die Nutzer anschließend als Access Tokens beim Routenzugriff verwenden. Die FeathersJS Authentifizierung basiert intern auf der Bibliothek Passport.js. Mit dieser können beliebige Authentifizierungsmechanismen, sogenannte Passport-Strategien, zur Anmeldung oder für den Routenzugriff verwendet werden. Strategien sind einzelne, unabhängige Module, die je nach Anwendungsfall eingebunden werden können. Neben den vielen vordefinierten Strategien, wie beispielsweise OAuth, bietet Passport.js auch die Möglichkeit, eigene Authentifizierungsmechanismen zu definieren. Diese sogenannten custom Strategies[7] machen das Framework so flexibel und erweiterbar. Feather.js Authentication kapselt nun die Vorteile von Passport.js in seiner eigenen API und vereint so Flexibilität und Modularität mit seiner gewohnten Serviceorientierten Struktur. Aus diesen Gründen entscheiden wir uns bei der Authentifizierung für FeathersJS.

Auch für die Berechtigungsprüfungen bietet FeathersJS, sowie einige andere, fertige Lösungen an. Obwohl FeathersJS Implementierung, FeathersJS-Permissions[8], besonders kompatibel zum restlichen auf diesem Framework basierenden System wäre, entscheiden wir uns am Ende dagegen. Der Grund hierfür liegt unter anderem in der Art und Weise der Berechtigungsprüfung: FeathersJS erwartet die Berechtigungen als ein Array von Rollen-Methoden-Paaren am User-Objekt (z.B. `permissions: [ 'admin: get' ]`). Am zu schützenden Service werden dann die Rollen angegeben, die auf den Service zugreifen dürfen. Hat ein Nutzer diese Rolle in seinen Permissions enthalten, so darf er den Service die zugehörigen Methoden anfragen. Wir wol-

7 http://www.passportjs.org/packages/passport-custom/
8 https://github.com/feathersjs-ecosystem/feathers-permissions

len den Berechtigungs-Overhead jedoch nicht auf Seite der User, sondern in den Hooks der Services. Dort soll das User-Objekt pro Methode auf bestimmte Kriterien geprüft werden und nichts weiter als seine Rolle mit in die Prüfung einbringen. Auf diese Weise wird das User-Objekt nicht unnötig aufgebläht. Außerdem wird die Logik der Berechtigungsprüfung als Teil des Service gekapselt, sodass Änderungen am Service und dessen Berechtigungen nur an einem Ort stattfinden und insbesondere die User-Datenbank nicht beeinflussen.

Alternativ zu einer fertigen Lösung können die Hooks auch selber implementiert werden. Das ist bei Berechtigungsprüfungen oft nötig, da sie mitunter sehr anwendungsspezifisch implementiert werden müssen. Auch hier ist das der Fall, weshalb wir uns für die eigene Implementierung entscheiden. Mehr dazu später.

8.4 UMSETZUNG DER AUTHENTIFIZIERUNG

8.4.1 *Kapselung*

Bevor die eigentliche Implementierung beginnen kann, muss zunächst einmal eine grundlegende Designentscheidung getroffen werden. Wie weiter oben schon erwähnt, läuft die Authentifizierung bisher über den Schul-Cloud Server. Die Frage ist nun, ob wir das so weiterführen oder lieber den Content-Server vollständig vom Schul-Cloud Server loslösen wollen.
Bliebe die Authentifizierung Sache des Schul-Cloud Servers, müsste er erst einmal um eine weitere Nutzertabelle, nämlich für die Inhalteanbieter, erweitert werden. Anschließend könnte das Frontend des Content-Dashboards den Schul-Cloud Server mit den Login-Daten des Nutzers um Authentifizierung bitten. Dieser gäbe ein JWT zurück, welchen das Frontend bei jeder zukünftigen Anfrage an den Content-Server im Authentifizierungs-Header mit senden müsste. Der Content-Server würde dann den JWT dem Schul-Cloud Server zur Prüfung schicken, um dessen Gültigkeit sicherzustellen, bevor er die Anfrage beantworten würde.
Diese Variante hätte den Vorteil, dass alle Nutzer, sowohl die der Schul-Cloud als auch des Content-Dashboards, an einem zentralen Ort hinterlegt wären. Auch die Authentifizierung würde dann Aufgabe eines Servers sein, sodass vielleicht vorhandene Implementierungen wiederverwendet werden könnten. Dafür würde der Schul-Cloud Server, der in erster Linie kein reiner Authentifizierungsserver ist, zu einem "Hausmädchen für alles"-Server aufgebläht werden.
Theoretisch wäre es schöner, wenn es einen dritten Server gäbe, der sämtliche Nutzertabellen enthielte und ausschließlich für Authentifizierung zuständig wäre. Schul-Cloud Server und Content-Server könnten dann, völlig losgelöst voneinander und sich auf ihre Aufgaben konzentrierend, arbeiten, während der Authentifizierungs-Server die Nutzerverwaltung kapselt. Ein weiterer Nachteil, die Authentifizierung im Schul-Cloud Server zu belassen, wäre der hohe Traffic.

Bei jeder Anfrage müsste der Content-Server den Schul-Cloud Server um Prüfung des JWTs bitten, bevor er die eigentliche Anfrage beantworten kann.

Eine andere Möglichkeit wäre, den Content-Server die Authentifizierung selber ausführen zu lassen und ihn so vom Schul-Cloud Server zu entkoppeln. Deutlicher Vorteil wäre die komplette Unabhängigkeit vom Schul-Cloud Server. Das reduziert den Traffic deutlich und würde das gesamte Projekt nicht mehr nur semantisch eigenständig machen. Möglicherweise gäbe es in diesem Fall jedoch redundanten Authentifizierungs-Code in beiden Servern. Das lässt sich zu diesem Zeitpunkt noch nicht klar sagen.

8.4.2 *Wahl der Strategien*

Das FeathersJS Authentifizierungsmodul unterstützt mehrere Möglichkeiten zur Authentifizierung von Nutzern bei der Anmeldung sowie beim Routenzugriff, sogenannte Strategien. Darunter befinden sich vordefinierte OAuth-Strategien[9] zur Anmeldung mit existierendem Google, Facebook, GitHub oder Twitter Accounts, die Anmeldung mit Nutzernamen und Passwort (local-Strategie[10]) sowie die Authentifizierung durch Vorlage eines JWT. Zusätzlich kann man dank Passport.js custom strategies noch beliebige eigene Strategien schreiben oder die über 300 Strategien[38] anderer Nutzer verwenden. Bei einer so großen Auswahl ist eine Vorauswahl notwendig.

OAuth ist eine äußerst praktische und beliebte Möglichkeit, die es einem Nutzer erlaubt, sich mit einem Account (z.B. Google-Account) bei mehreren Plattformen anzumelden. Wäre die Zielgruppe unserer Plattform ein privater Verbraucher und nicht ein geschäftlich handelnder Inhalteanbieter, so wäre diese Strategie gut geeignet. Doch bei den Nutzern des Dashboards ist es fraglich, ob sie Accounts bei den großen Providern zur geschäftlichen Nutzung haben. Falls nicht wäre OAuth hier nicht sinnvoll. Wir entscheiden uns daher erstmal auf diese Strategie zu verzichten, können sie jedoch notfalls schnell und einfach nachrüsten.

Die lokale Anmeldung über Nutzernamen und Passwort ist die wohl einfachste Möglichkeit einen Nutzer zu authentifizieren. Hierbei werden Benutzername sowie Passwort beim Login an den Server gesendet. Dieser prüft dann dessen Existenz in der hinterlegten Nutzerdatenbank und gibt im Erfolgsfall das JWT zur Autorisation späterer Anfragen zurück. Voraussetzung ist natürlich die vorherige Registrierung des Nutzers, wobei Passport.js das erhaltene Passwort, mit BCrypt gehasht, in der Datenbank speichert. Dieser Hashing-Algorithmus gilt als sicher und robust[13]. Die lokale Strategie wird von Nutzern erwartet, ist einfach zu verwenden und verhältnismäßig sicher, dank BCrypt. Sie wird daher im Dashboard bereitgestellt.

9 http://www.passportjs.org/docs/oauth-api/
10 https://docs.feathersjs.com/api/authentication/local.html

Auf Grund der vorherigen Entscheidung, den Content-Server vom Schul-Cloud Server loszulösen, besteht zumindest bei der Anmeldung keine Verwendung für die JWT Strategie, die FeathersJS anbietet. Allerdings wird diese Strategie an anderer Stelle benötigt, nämlich beim Prüfen der nach der Authentifizierung erhaltenen JWTs beim Routenzugriff.

Wir verwenden also zunächst die lokale und JWT Strategie zur Authentifizierung. Je nach Situation kann die eine oder andere Strategie, zum Beispiel OAuth oder eine selbst Implementierte, später ergänzt werden.

8.4.3 *Authentifizierung im Entwicklungs-Modus*

Im Laufe der Entwicklung oder beim Debugging eines Service muss ein Entwickler oft Anfragen z.B. mit Postman testen. Mittlerweile sind jedoch nahezu alle Services vor nicht-authentifizierten Nutzern geschützt. Somit müsste sich ein Entwickler momentan erst authentifizieren, um einen JWT zu erhalten, bevor er damit auf den Service zugreifen darf. Auch die lokal laufenden Tests müssen jedes Mal erst die Authentifizierung erfolgreich durchlaufen, bevor sie die eigentliche Route testen können. Dieser Mehraufwand beim Testen soll reduziert werden. Dazu muss eine Möglichkeit gefunden werden, wie ein Entwickler beim Zugriff auf einen Service direkt authentifiziert werden kann, ohne dass er sich zuvor ein JWT holen muss. Weiterhin soll es möglich sein, dass sich der Entwickler zum Testen der Oberfläche ohne "echten" Account einloggen kann.

8.4.3.1 *Basic-Auth Strategie*

Für diese Fälle gab es bereits im bestehenden System eine Lösung namens Authentifizierung mittels Basic-Auth. Hierbei wurde bei jedem Routenzugriff im Authorization-Header der Anfrage ein Basic-Auth Token mit gesendet, welches aus Nutzernamen und Passwort gebildet wurde. Die Zugangsdaten wurden dann im Content-Server aus dem Token ausgelesen und mit zwei hardgecodeten Testnutzern in der Konfigurations-Datei abgeglichen. Im Fall der Übereinstimmung war der Nutzer erfolgreich authentifiziert. Die Testnutzer standen nicht in jeder Konfigurations-Datei, sondern lediglich in der `development.json` und waren demnach auch nur im Entwicklungs-Modus verfügbar. Im Production-Modus war eine Authentifizierung auf diese Art also nicht möglich.

Für einen vereinfachten Routenzugriff soll diese bestehende Lösung als eine alternative Authentifizierungsstrategie teilweise übernommen werden.

Wie bereits zu Beginn des Kapitels erwähnt bietet FeathersJS dank Passport.js die Möglichkeit, eigene Strategien zu definieren, also festzulegen, ab wann ein Nutzer als authentifiziert gilt. Diese Möglich-

keit soll nun genutzt werden, um die Basic-Auth Authentifizierung einführen. Wir definieren also eine neue passport-custom Strategie. Diese erwartet einen Verifizierer, der die Entscheidung trifft, ob der Nutzer authentifiziert ist oder nicht. In diesem Fall soll der Verifizierer zunächst die Credentials aus dem Basic-Auth Token im Authorization-Header auslesen. Gibt es keine, wird der erste Fehler mit `return done(null, false)` geworfen. Ansonsten wird der Nutzer in der Nutzer-Datenbank gesucht. Gibt es den Nutzer, so vergleicht der Verifizierer das für diesen Nutzer gespeicherte und mit BCrypt gehashte Passwort mit dem aus den Credentials stammenden Passwort. Bei Übereinstimmung wird die UserID des Nutzers mit `return done(null, {\textit{id: user.}id})` zurückgeben und der Nutzer ist erfolgreich authentifiziert[30] [9] [5].

```
module.exports = function(){
    return function(app) {
        const verifier = (req, done) => {
            (...)
      // parse Nutzername & Passwort aus dem Basic-Auth Token im Authorization-Header
            const credentials = basicAuth.parse(authHeader);
            if (!credentials) {
                return done(null, false);
            }

      // suche den Nutzer anhand seines Nutzernamens in der User-Datenbank
            userModel.findOne({ username: credentials.name}, function (err, user) {
                if (err) { return done(err); }
                if (!user) {
                    return done(null, false, { message: 'Incorrect username' });
                }
                const hashedPassword = user.password;
        // vergleiche das Passwort des Tokens mit dem gehashten Passwort des Nutzers aus der Datenbank
                if (bcrypt.compareSync(credentials.pass, hashedPassword)){
          // bei Übereinstimmung der Passwörter ist Nutzer authentifiziert
          // gebe dessen Id zurück
                    return done(null, {_id: user._id});
                } else return done(null, false, { message: 'Incorrect password' });
            });
        };
        // registriere die strategy in der app.passport Instanz
        this.passport.use('basicAuth', new Strategy(verifier));
    };
};
```

Listing 13: selbst definierte Basic-Auth Strategie

Diese Strategie ist lediglich für den Testfall gedacht. Sie erlaubt den Entwicklern, sich erst direkt beim Routenzugriff zu authentifizieren. Auch die bisher verwendeten Crawler, deren Authentifizierung im bestehenden System auf Basic-Auth basiert hat, werden nun weiter

unterstützt.
Dank Passport.js custom-strategy konnte nun auf einfache Weise ein modulares und wiederverwendbares Authentifizierungsverfahren gebaut werden. Im Normalfall soll beim Routenzugriff weiterhin die Standard JWT-Strategie verwendet werden, die das vom Authentifizierungsservice vergebene JWT abfragt. Aus diesem Grund werden beide Strategien im `AuthenticateHook` gekapselt.
Dieser Hook soll bei jedem externen Zugriff auf eine Route den Nutzer authentifizieren. Falls der Authorization-Header existiert und ein Basic-Auth Token enthält, soll die Basic-Auth Strategie verwendet werden. Ansonsten wird immer auf die Standard JWT Strategie zurückgegriffen[30].

```
const authenticateHook = () =>
    commonHooks.iff(
        // falls Anfrage extern ist
        commonHooks.every(commonHooks.
        ↪  isProvider('external')),
        commonHooks.iffElse(
          // falls der Authorization-Header ein Basic-Auth
          ↪  Token enthält
          ctx => (ctx.params.headers['authorization'] ||
          ↪  '').startsWith('Basic'),
          // dann Authentifizierung mit der Basic-Auth
          ↪  Strategie
          authentication.hooks.authenticate('basicAuth'),
          // sonst Authentifizierung mit der JWT-Strategie
          ↪  als Fallback
          authentication.hooks.authenticate('jwt')
        )
    );
module.exports = authenticateHook;
```

Listing 14: authenticateHook kapselt die zwei definierten Strategien

8.4.3.2 *Demo-Nutzer*

Auch für die Nutzung der Basic-Auth Strategie braucht der Entwickler einen Benutzeraccount, bestehend aus Nutzernamen und Passwort. Hierfür eignen sich Test- bzw. Demonutzer. Die Frage ist, wo diese Accounts, die in den Strategien abgeglichen werden, gespeichert werden - in der Datenbank oder wie bisher in der Konfigurations-Datei.
Das Speichern in der `Development.json` hat den Vorteil, dass die Testnutzer auch nur im Development-Modus existieren. Im Production-Modus bliebt die Datenbank immer sauber und das System sicher, sollte jemand an die Accounts gelangen. Letzteres ist besonders wichtig, da der gesamte Quellcode OpenSource bei GitHub einsehbar ist.

Besondere Nutzer, die es vielleicht doch im Production-Modus geben soll wie beispielsweise einen Superhero, müssten allerdings von Hand in die Datenbank geschrieben werden.
Ebenso gut könnten diese sowie alle anderen Testnutzer auch durch einen Seed in die Datenbank eingepflegt werden. In dem Fall wäre nur ein Kommando notwendig, um alle Nutzer zu erhalten. Diese Methode hätte zusätzlich den klaren Vorteil, dass es nur eine Source of Truth gäbe, also eine Quelle, an der Nutzer stehen können. Dafür stünden dann auch die Testnutzer im Production-Modus in der Datenbank, was absolut unsicher wäre, falls jemand an die Accounts gelangt.

Wir entscheiden uns hier für das Speichern aller Nutzer in der Datenbank, die allerdings nur im Entwicklungs-Modus durch einen Seed befüllt wird. So kann der Vorteil der Single Source of Truth genutzt werden, da die Strategien lediglich in der Datenbank nach Nutzern suchen müssen anstatt zusätzlich in der `Development.json`. Außerdem wird das potentielle Sicherheitsleck behoben, da die Datenbank im Production-Modus sauber und sicher bleibt. Ein Superhero könnte, falls benötigt, nachträglich von Hand in die Datenbank eingetragen werden.

8.4.3.3 *Login*

Mit diesen neuen Test-Accounts können sich Entwickler nun auch im Frontend einloggen, falls sie dieses testen möchten. Obwohl an dieser Stelle auch die Basic-Auth Strategie wiederverwendet werden könnte, belassen wir es bei der bisherigen local Strategie. Benutzer sowie Entwickler müssen gleichermaßen mit ihren (Test-) Accounts die `/authentication`-Route anfragen und sich ein JWT holen. Auf diese Weise braucht das Frontend nicht angepasst zu werden, welches sonst beim Login die Credentials sowohl als Basic-Auth Token im Authorization-Header sowie im Request-Body hätte mit senden müssen. Nur dank der vorherigen Entscheidung, die Testnutzer in die Datenbank zu schreiben, anstatt in die Konfigurations-Datei, kann dieser Mehraufwand vermieden werden.

8.5 UMSETZUNG DER BERECHTIGUNGSPRÜFUNG

8.5.1 *Nutzerhierarchie*

Nachdem die Authentifizierung implementiert ist, fehlt noch die Berechtigungsprüfung. Bevor diese allerdings umgesetzt werden kann, bedarf es eines theoretischen Konzepts zur Hierarchie der Nutzer:
Jeder Benutzer ist seiner Firma zugeordnet und sieht nur die Inhalte derselben. Diese Inhalte können auch von allen seinen Kollegen bearbeitet werden. Solchen normalen Benutzern wird die Rolle `user` zugeteilt. Um neue Benutzer innerhalb einer Firma hinzuzufügen oder

zu löschen, braucht es eine Rolle mit mehr Rechten, dem `admin`. Er ist berechtigt, alle Nutzer seiner Firma zu verwalten. Seine Erstellung bei der Registrierung einer neuen Firma ist Aufgabe des `superhero` der Schul-Cloud. Diese Rolle hat sämtliche Berechtigungen. Dennoch sollen die Schul-Cloud Mitarbeiter mit dieser Rolle möglichst wenig Arbeit haben und lediglich neue Firmen in das dafür vorgesehene Datenbank Schema eintragen und ihnen einen Admin erstellen. Dieser kümmert sich um den Rest.

8.5.2 *Implementierungsdetails*

Mit einem klaren Rollen- und Aufgabenkonzept kann es nun zum nächsten Schritt gehen, nämlich sich einen Überblick über die benötigten Permission-Hooks für die einzelnen Services zu verschaffen. Wie an der Berechtigungstabelle (siehe unten) klar zu erkennen ist, ist die Mehrheit der Berechtigungsprüfungen sehr spezifisch. Mit einer fertigen Bibliothek wäre das schwer umsetzbar gewesen.

Service	Methode	Authentifizierte Nutzer			Nicht-authentifizierte Nutzer
		Superhero	Admin	User	Lern-Store Nutzer
/users	find	Ja	Ja, beschränkt auf Firma	Nein	Nein
	get	Ja	Ja, beschränkt auf Firma	Ja, beschränkt auf sich selbst	Nein
	create	Ja	Ja, beschränkt auf Firma	Nein	Nein
	patch	Ja	Ja, beschränkt auf Firma	Ja, beschränkt auf sich selbst	Nein
	update	Ja	Ja, beschränkt auf Firma	Nein	Nein
	delete	Ja	Ja, beschränkt auf Firma	Nein	Nein
/search	find	Ja	Ja, beschränkt auf Firma und öffentliche Inhalte	Ja, beschränkt auf Firma und öffentliche Inhalte	Ja, beschränkt auf öffentliche Inhalte
/files/upload	create	Ja	Ja, beschränkt auf Firma	Ja, beschränkt auf Firma	Nein
/files/manage	patch	Nein	Nein	Nein	Nein
/files/get*	find	Ja	Ja, beschränkt auf Firma und öffentliche Inhalte	Ja, beschränkt auf Firma und öffentliche Inhalte	Ja, beschränkt auf öffentliche Inhalte
/files/structure	get	Ja	Ja, beschränkt auf Firma	Ja, beschränkt auf Firma	Nein
/files/thumbnail	patch	Nein	Nein	Nein	Nein
/redirect	get	Ja	Ja	Ja	Ja
/resources	ALLE	Ja	Ja, beschränkt auf Firma	Ja, beschränkt auf Firma	Nein
/resources/resource-schema	find	Ja	Ja	Ja	Nein
/resources/bulk	create	Ja	Ja, beschränkt auf Firma	Ja, beschränkt auf Firma	Nein
	patch	Ja	Ja, beschränkt auf Firma	Ja, beschränkt auf Firma	Nein
	delete	Ja	Ja, beschränkt auf Firma	Ja, beschränkt auf Firma	Nein
/resources_filepaths	ALLE	Nein	Nein	Nein	Nein

Abbildung 18: Berechtigungstabelle

Die Implementierung der Berechtigungen anhand der Tabelle im Backend gestaltet sich simpel. Um die Zuordnung zu einer Firma umzusetzen, reicht es, jedem Nutzer eine FirmenID zu geben, welche Schlüssel im Firmen-Schema der Datenbank ist. Auch jede angelegte Ressource bekommt die FirmenID ihres Erstellers zugeordnet. Auf diese Weise lässt sich einfach prüfen, ob derjenige, der die Ressource anfragt, derselben Firma angehört und somit berechtigt ist. Für die zweite wichtige Prüfung, nur öffentliche Inhalte anzuzeigen, wird das isPublished-Flag an jeder Ressource auf seinen Wert geprüft. Hier reicht bei den meisten Methoden wie FIND eine einfache Erweiterung des Queries um `isPublished: true` bzw. `$ne:{ isPublished: false }` (für diejenigen, alten Inhalte, die keine solche Flag haben).

Auch im Frontend brauchen wir eine Prüfung der Rolle des Nutzers, um zu ermitteln, ob auf die Nutzerregistrierungs-Seite zugegriffen werden darf. Als Superhero und Admin ist das der Fall, als normaler User jedoch nicht. Die Registrierungsseite besteht hierbei aus der Route `/users`, die eine Tabelle aller existierenden Nutzer mit inline Bearbeitung und Löschen Funktion bereitstellt. Um Nutzer zu erstellen, muss die Seite `/users/registration` verwendet werden. Beide Seiten sollen für Benutzer mit der Rolle `user` nicht erreichbar sein. Um das zu erzielen, wird im Vue.js Router einen beforeEnter-Guard[11] eingebaut, der jeden `user` auf die Seite, von der er kam, zurückleitet, anstatt die geschützten Seiten anzuzeigen.

```
1  {
2    path: "/users",
3    beforeEnter: (to, from, next) => {
4      if ((store.getters["user/GET_USER"] || {}).role ===
       ↪  "user") {
5        next(from);
6      } else {
7        next();
8      }
9    },
10 },
```

Listing 15: beforeEnter-Guard schützt die Routen im Vue-Router

Zusätzlich wird an anderer Stelle die Nutzerverwaltung aus der Menüleiste herausgefiltert, sollte es sich um einen `user` handeln. Der normale Nutzer bekommt so bereits im Frontend keine Möglichkeit angeboten, um Nutzer zu bearbeiten. Auf diese Weise wird klar kommuniziert, was der Nutzer darf und was nicht, anstatt bestimmte Features einfach im Backend zu verbieten.

11 https://router.vuejs.org/guide/advanced/navigation-guards.html#per-route-guard

Zusammenfassung

Mit Abschluss dieses Kapitels ist das Content-Dashboard vor unberechtigtem Zugriff aller Art geschützt. Das verwendete Authentifizierungsmodul von FeathersJS erlaubt den Nutzern eine Anmeldung mit Benutzernamen und Passwort. Außerdem wird mithilfe der selbst definierten Basic-Auth Strategie eine alternative Authentifizierungsmethode für einen vereinfachten Routenzugriff im Entwicklungs-Modus angeboten. Die implementierten Berechtigungsprüfungen in den Hooks der Services schützen die Ressourcen und Nutzer-Accounts vor authentifiziertem, aber unberechtigtem Zugriff.

9

TESTS

Katharina Blaß

Die erfolgreiche Umsetzung der in der Anforderungsanalyse festgelegten, nicht-funktionalen Anforderungen an das Content-Dashboard soll in diesem Kapitel kontrolliert werden. Dazu wird jede der drei Anforderungen - Verständlichkeit, Bedienbarkeit und Performance - durch einen entsprechenden Test geprüft.

9.1 USER-TEST

Zur Überprüfung der Verständlichkeit aller Funktionalitäten des Content-Dashboards werden zwei User-Tests durchgeführt. Ziel ist es dabei hauptsächlich, Schwachstellen in der Benutzeroberfläche aufzudecken, die eine schnelle und intuitive Benutzung verhindern. Dabei liegt der Fokus auf den Funktionalitäten Datei-Hosting, CSV-Import und Massenbearbeitung, da nur sie eine testbare Oberfläche anbieten. Einer der Testkandidaten ist Michael Ciuberski vom Kreismedienzentrum Oberhavel. Zu seinen alltäglichen Aufgaben gehört, Lizenzen für Inhalte zu erwerben, die dann über Antares im Schulmedienportal für die Schulen im Landkreis Oberhavel freigeschaltet werden. Der zweite Test findet mit Benno Köhler statt, einem ehemaligen Lehrer und privaten Anbieter eigener Lernmaterialien. Seine Inhalte vertreibt er bislang über eine eigene Website oder verteilt sie direkt an die interessierten Schulen über externe Speichermedien. Dabei gehört er als kleinerer Inhalteanbieter exakt in die Zielgruppe des Datei-Hostings, der Hauptfunktionalität des Content-Dashboards. Mit diesen beiden Kandidaten werden nun einige Szenarien zum Testen der drei oben genannten Funktionalitäten durchgeführt und ihre wichtigsten Anmerkungen im Folgenden zusammengefasst.

Die Reaktion der beiden Tester auf die Datei-Hosting Funktionalität ist positiv. "Ich würde sagen, wenn ich die Dateien bei der SchulCloud hoste, habe ich es viel leichter. Dann habt ihr das Problem, sie verfügbar zu halten. Warum sollte ich das selbst hosten?", so der erste Eindruck von Benno Köhler. Im Laufe des Tests fallen den Testern auch einige unentdeckte Schwachstellen des Features auf. Der Bereich zum Hochladen der Dateien (die Dropzone) ist erst sichtbar, wenn in einem entsprechendem Select die Option "Bei der Schul-

Cloud hosten" ausgewählt wird. Das "Verstecken" dieses Bereichs sorgt für Unsicherheit beim Einpflegen eines neuen Inhaltes. An dieser Stelle würden die Tester einen Stepper, wie er beim CSV-Import verwendet wird, bevorzugen. Dieser würde die Seite klarer strukturieren und die Dropzone genau dann anzeigen, wenn sie relevant wird. Ein weiterer Kritikpunkt sind die teilweise unklaren Bedeutungen einiger Metadaten, wie Mime-Type und ihrer Auswahlmöglichkeiten. "Alles was wir da [in den Lern-Store] reinstellen werden als Lehrer sind Lernobjekte.", argumentiert Benno Köhler bei dem Versuch die Kategorie seines Inhaltes auf eine der aufgelisteten Optionen (Lernobjekt, atomic, bewährtes Lernobjekt) festzulegen. Diese Metadaten sollten daher in Zukunft noch einmal überdacht werden. Zu guter Letzt wünschen sich die Tester bei dem Metadaten-Feld Lizenz eine Auswahlmöglichkeit aus bestehenden Lizenzen anstelle der Freitext-Eingabe. An dieser Stelle lohnt sich ein Blick auf die Bachelorarbeit von Hannes Kohlsaat und Ivan Ilic[23], die sich mit dem Thema Lizenzen ausführlich beschäftigt haben. Ihr entwickelter Lizenz-Editor ist als Prototyp bereits in das Content-Dashboard integriert. Damit haben Inhalteanbieter die Möglichkeit, direkt im Dashboard neue Lizenzen, angepasst auf ihre Bedürfnisse, zu erstellen. Diese könnten in Zukunft bei der Erstellung neuer Inhalte im Metadaten-Feld Lizenz als Auswahlmöglichkeit in einem Select angezeigt werden.

Das zweite Testszenario thematisiert den CSV-Import. Auch hier kann die Oberfläche beide Tester auf den ersten Blick überzeugen. Besonders der Stepper wird aufgrund seiner klaren Nutzerführung geschätzt, wie Michael Ciuberski mit folgenden Worten erklärt: "Da war so richtig vorgegeben Step-by-Step was man machen muss. Das fand ich gar nicht verkehrt!". Lediglich bei der Metadatenzuordnung benötigen beide Tester etwas mehr Zeit, um die Aufgabe zu verstehen. "Eine Lösung für dieses Problem könnte darin bestehen, dass man die Excel-Datei in einer Form vorgibt, so wie man sie ausfüllen soll", schlägt Benno Köhler vor. Auf diese Weise könnte zwar die Metadatenzuordnung entfallen, dafür müssten bestehende Inhalte im Vorfeld in das Schema der Excel- bzw. CSV-Datei gebracht werden. Die Zuordnungsmöglichkeit soll diesen Mehraufwand vermeiden und bleibt daher bestehen. Möglicherweise könnte ein unterstützender Erklärungstext ausreichen, um ihre Verständlichkeit zu verbessern.

Die nächste Aufgabe der Tester besteht darin, die Metadaten mehrerer Inhalte zu bearbeiten. "Man könnte das für jeden Inhalt einzeln machen, aber ich will mir ja die Arbeit sparen", bestätigte Michael Ciuberski die Wichtigkeit/Notwendigkeit des Massenbearbeitung Features. Auch die Funktion, die in der Verwaltungstabelle sichtbaren Spalten nach Bedarf und Bildschirmgröße ein- oder auszublenden, wird gern genutzt. Kritisch setzten sich die Tester dagegen mit der Suchen & Ersetzen Funktion auseinander, deren Verhalten sie anfangs, besonders bei leeren Eingaben, nur schwer verstehen. "Ich denke eine ausführliche Anleitung wäre da schon eine große Hilfe!", erklärt Michael Ciuberski. Dieser Vorschlag wird im Anschluss an den User-Test direkt umgesetzt.

Im Anschluss an die Szenarien werden die beiden Tester nach weiteren Wünschen an das Content-Dashboard befragt. Benno Köhler bemerkt das Fehlen jeglicher Preisangaben und Abrechnungsmöglichkeiten. Diese seien allerdings seine Meinung nach essentiell, für eine Nutzung des Dashboards in der Praxis. Das Thema Abrechnung liegt nicht im Rahmen dieser Arbeit, aber sollte in naher Zukunft ergänzt werden. Grundlage hierfür sind ausführliche Nutzungsstatistiken zu den einzelnen Inhalten. Möglich wären die Messung und Anzeige eines jährlichen Nutzungsverlaufs, einer bundesweiten Verteilung oder die Anzahl der Klicks pro Inhalt. Letzteres sei laut Benno Köhler nicht sehr zuverlässig. "Ist das jetzt eine Person, die nur einen Klick gemacht hat oder eine Person die kann tausend Klicks gemacht haben.", argumentiert er. Wichtiger sei ihm eine monatliche und jährliche Nutzungserfassung, die bis auf die Schulebene vordringt. Auch das Thema Statistiken sollte in Zusammenhang mit möglichen Abrechnungsmodellen in zukünftigen Arbeiten näher untersucht werden.

Die Auswertung der User-Tests zeigt ein überwiegend positives Ergebnis. Obwohl einige Kleinigkeiten sowie wenige konzeptionelle Verbesserungsmöglichkeiten im System aufgedeckt wurden, schienen die getesteten Funktionalitäten verständlich genug zu sein. Alle in der Anforderungsanalyse festgelegten Maßnahmen bzw. Akzeptanzkriterien sind weitestgehend umgesetzt. Damit gilt die Verständlichkeits-Anforderung als erfüllt. "Das System läuft, wie man es möchte. Intuitiv lässt sich alles machen. [...] Ich könnte mir vorstellen, dass ich eines Tages bei der Schul-Cloud meinen Content vollständig einstelle", lobt Benno Köhler und beendet damit den User-Test.

9.2 ACCESSIBILITY-TEST

Die Bedienbarkeits-Anforderung an das Content-Dashboards erfordert die Umsetzung der Akzeptanzkriterien Accessibility und responsive Design. Letzteres kann über die Chrome Developer-Tools durch Simulation verschiedener Geräte und Displaygrößen mit positivem Ergebnis getestet werden. Dieses Kriterium ist damit erfüllt.

Zur Kontrolle der Accessibility kann ebenfalls ein entsprechender Test über die Chrome Developer-Tools durchgeführt werden[10]. Das Tool überprüft, ob eine Tastaturnavigation möglich ist, die Texte kontrastreich dargestellt werden und sämtliche Elemente mit Namen, Labels und Alternativtexten für die Unterstützung von Screen Readern ausgestattet sind. Das Ergebnis zeigt eine herausragende Accessibility-Bewertung von 100/100 auf der Login-, Nutzerverwaltungs- und CSV-Importseite. Lediglich bei der Verwaltungstabelle und der Datei-Hosting-Seite wird ein geringerer Score von 89/100 erreicht. Der Grund liegt in den fehlenden Labels bei den auf diesen Seiten verwendeten Tag-Inputs, die über eine externe Bibliothek eingebunden sind. Zur Behebung des Problems, wurde bereits ein entsprechender Issue in dem GitHub-Repository der Bibliothek angelegt. Davon abgesehen,

gibt keine weiteren Handlungsmöglichkeiten, um das Testergebnis zu verbessern. Daher gilt auch das Accessibility-Akzeptanzkriterium als umgesetzt und die Bedienbarkeit somit als erreicht.

9.3 PERFORMANCE-TEST

Die dritte und letzte zu testende Anforderung ist die Performance. Mithilfe der Chrome Developer-Tools werden die, in der Anforderungsanalyse festgelegten, Akzeptanzkriterien geprüft[10]. Jede Unterseite soll demnach nicht größer als 1MB sein, maximal 50 Netzwerkanfragen senden und eine Performance-Bewertung von mindestens 80/100 erreichen.

Die ersten zwei Kriterien werden mithilfe des Network Developer-Tools getestet. Dabei stellt sich heraus, dass keine der Seiten größer als 600 KB ist. Die Anzahl an Netzwerkanfragen pro Unterseite liegt bei maximal 41 Anfragen. Diese beiden Akzeptanzkriterien gelten damit als erfüllt. Eine Gesamteinschätzung der Performance für jede Unterseite liefert der Lighthouse Report über das Audits Developer-Tool. Dieses Tool testet viele Metriken, unter anderem die Zeit, bis das erste Element gerendert, der Hauptinhalt geladen und die Seite interaktiv ist. Das Ergebnis übertrifft die Anforderungen sogar mit einer Bewertung von 100/100 auf allen Unterseiten. Damit ist die Performance-Anforderung an das Content-Dashboard ebenfalls erfüllt.

Zusammenfassung

Das Testkapitel hat bewiesen, dass alle in der Anforderungsanalyse definierten nicht-funktionalen Anforderungen erfolgreich umgesetzt wurden. Das Content-Dashboard ist verständlich, bedienbar und liefert eine gute Performance.

10

ZUSAMMENFASSUNG

Katharina Blaß

Diese Arbeit hatte das Ziel, das Content-Dashboard zu einer Verwaltungsplattform für Inhalteanbieter auszubauen, über die sie ihre Lernmaterialien eigenständig in den Schul-Cloud Lern-Store einpflegen können.
Die unterschiedlich großen Inhalteangebote und technischen Voraussetzungen der Anbieter sorgten für den Bedarf verschiedener Funktionalitäten. Diese wurden in der Anforderungsanalyse charakterisiert und umfassen Datei-Hosting, CSV-Import, Filter, Massenbearbeitung und Authentifizierungsmechanismen. Nichtfunktionaler Schwerpunkt lag auf den Anforderungen der Verständlichkeit, Bedienbarkeit und Performance aller Funktionalitäten.

Die erfolglose Suche nach einer geeigneten, externen Lösung für die Hauptfunktionalität des Content-Dashboards, dem Datei-Hosting, führte zu der Entscheidung einer vollständig eigenen Implementierung. Das Datei-Hosting ermöglicht den kleineren oder privaten Inhalteanbieter, ihre Inhalte direkt über die Schul-Cloud zu hosten. Über eine Oberfläche können dazu sämtliche zum Inhalt gehörenden Dateien hochgeladen, gespeichert und über den Lern-Store ausgeliefert werden. Um auch große Inhalteanbieter schnell in den Lern-Store zu integrieren, ermöglicht der CSV-Import, viele Inhalte auf einmal aus einer CSV-Datei zu einzupflegen. Alle bereitgestellten Inhalte können in der Verwaltungstabelle, auf der Startseite des Content-Dashboards, von ihren Anbietern verwaltet werden. An dieser Stelle ist auch das Entfernen von Inhalten aus dem Lern-Store mit nur wenigen Klick möglich. Die Suche nach bestimmten Inhalten in der Tabelle wird durch die bestehende Textsuche von ElasticSearch sowie einem neuen, wiederverwendbarem Filtermodul unterstützt.
Darüber hinaus erlaubt die Massenbearbeitungsfunktionalität, die Metadaten mehrerer Inhalte nachträglich zu bearbeiten. Dank der integrierten Suchen & Ersetzen Funktion kann sogar eine partielle Bearbeitung der Metadaten erfolgen.
Weiterhin sorgen die modularen Authentifizierungsmechanismen sowie die im Server integrierten Berechtigungsprüfungen für den Schutz der Inhalte und der Benutzer-Accounts. Inhalteanbieter können sich mit Benutzernamen und Passwort anmelden, Entwicklern stehen zu Testzwecken Demo-Accounts zur Verfügung.

Im abschließendem Test-Kapitel wurde die Umsetzung der nicht-funktionalen Anforderungen mit positivem Resultat überprüft. Damit sind alle definierten Anforderungen erfüllt und das Ziel der Arbeit erreicht. Das Content-Dashboard bietet nun alle benötigten Funktionalitäten, um Inhalte anbieterseitig in den Schul-Cloud Lern-Store integrieren zu können. Damit ist ein entscheidender Schritt in Richtung eines vielfältigen Lern-Stores getan.

10.1 AUSBLICK

Trotz des erfolgreichen Projektergebnisses gibt es noch einige Möglichkeiten, das Content-Dashboard weiter zu verbessern. Neben kleineren Anpassungen in der Oberfläche, könnten zukünftig vor allem die im Test-Kapitel angesprochenen Wünsche der beiden Testkandidaten realisiert werden. Darunter zählen die Einführung von Preis- und Abrechnungsmodellen sowie die Erhebung aussagekräftiger Nutzungsstatistiken. Außerdem könnten die weiteren, im Rahmen des diesjährigen Bachelorprojektes entstandenen Tools zu den Themen Lizenz-Editor[23] und DRM-Schutz[17] dauerhaft in das Content-Dashboard integriert werden. Auf diese Weise könnten die Inhalte über das Dashboard nicht nur verwaltet, sondern auch rechtssicher eingesetzt werden. Ein weiteres interessantes Thema ist die Gruppierung von zusammen gehörenden Inhalten. Bücher derselben Buchreihe oder aufeinander aufbauende Lernvideos könnten als Teil einer Serie oder Sammlung markiert und den Lehrer und Schülern vorgeschlagen werden.
Das Content-Dashboard bleibt auch weiterhin ein interessantes Thema für zukünftige Arbeiten mit viel Anpassungspotential. Dennoch ist die Plattform schon bereit, um in der Schul-Cloud eingesetzt zu werden und den Inhalteanbietern den Weg in den Lern-Store zu ebnen.

A

ANHANG

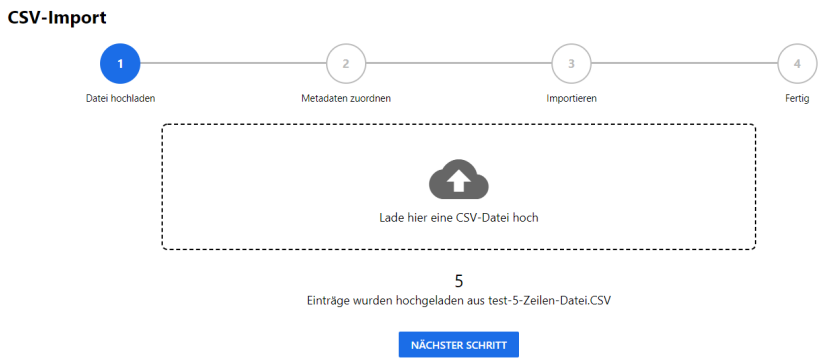

Abbildung 19: Teilschritt 1 - Upload

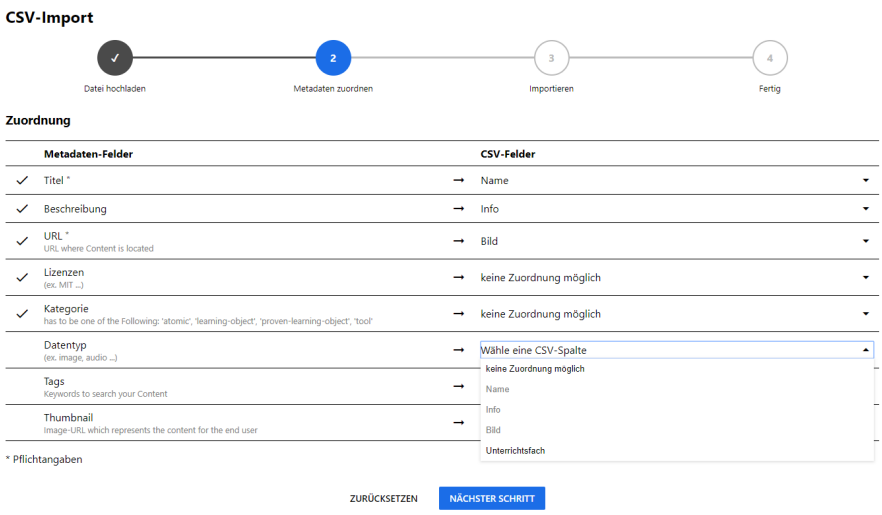

Abbildung 20: Teilschritt 2 - CSV Mapping

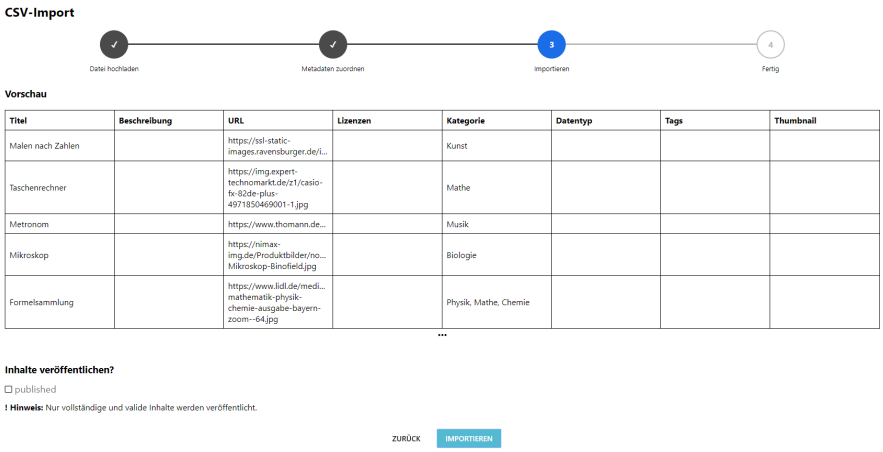

Abbildung 21: Teilschritt 3 - Vorschau

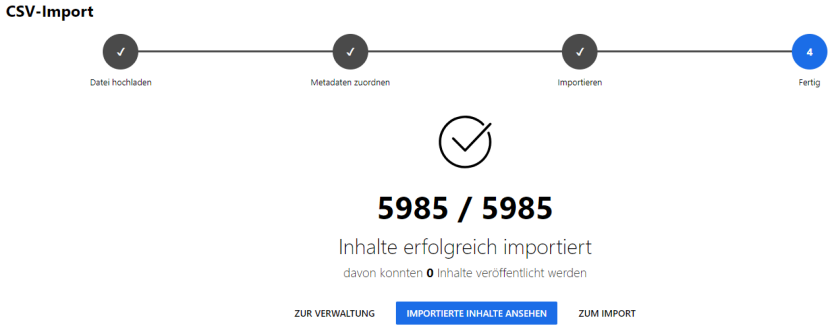

Abbildung 22: Teilschritt 4 - Ergebnis

		Dateigröße		
		5-Zeilen (784 B)	2000-Zeilen (293 KB)	10000-Zeilen (1,43 MB)
Frontend	*Laptop & fast 3G*	45,01 ms	276,67 ms	1,03 s
	Laptop & slow 3G	48,20 ms	274,85 ms	1,02 s
	Handy & fast 3G	132,81 ms	662,71 ms	2,27 s
	Handy & slow 3G	130,76 ms	656,33 ms	2,26 s
Backend	*Laptop & fast 3G*	1,15 s	6,63 s	28,55 s
	Laptop & slow 3G	4,07 s	13,23 s	49,96 s
	Handy & fast 3G	1,21 s	6,90 s	29,45 s
	Handy & slow 3G	4,16 s	13,50 s	50,83 s

Abbildung 23: Validierung - Performancevergleich als tabellarische Zusammenfassung

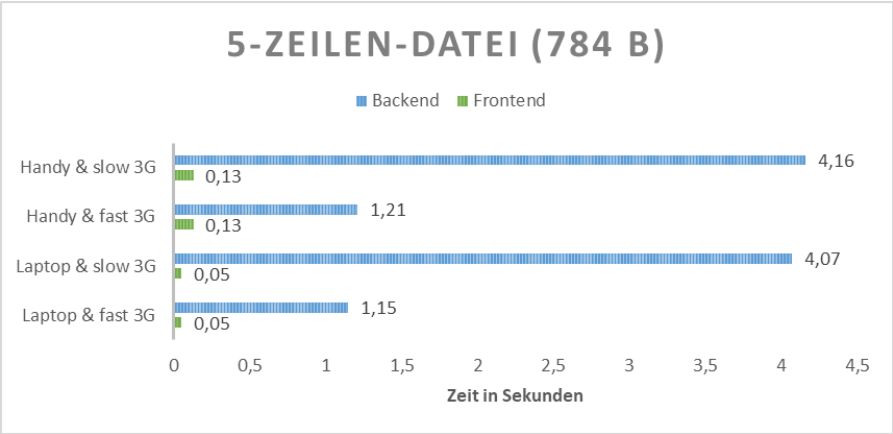

Abbildung 24: Validierung - Performancevergleich 5-Zeilen Datei

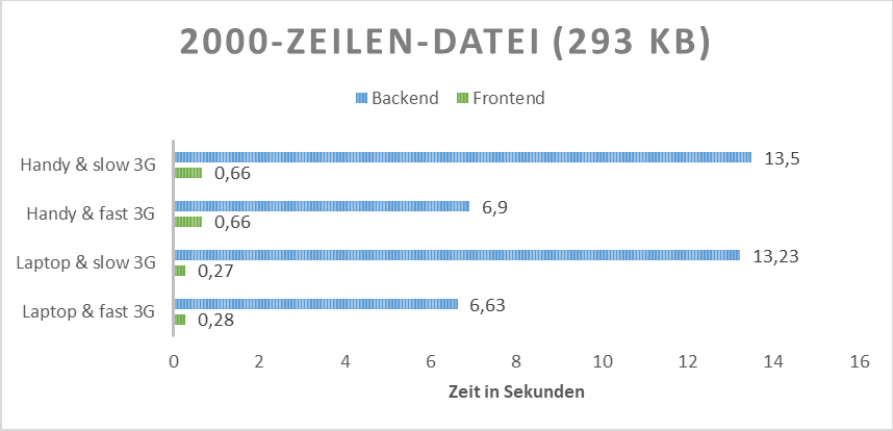

Abbildung 25: Validierung - Performancevergleich 2000-Zeilen Datei

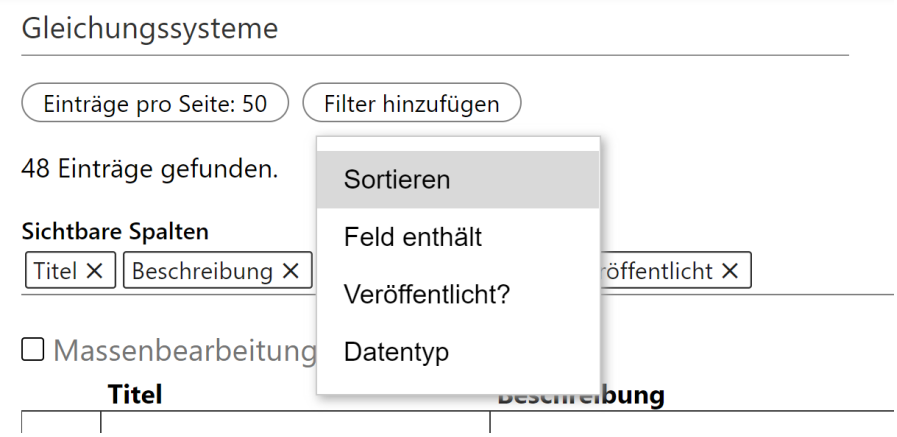

Abbildung 26: Filter - Auswahl von Filter

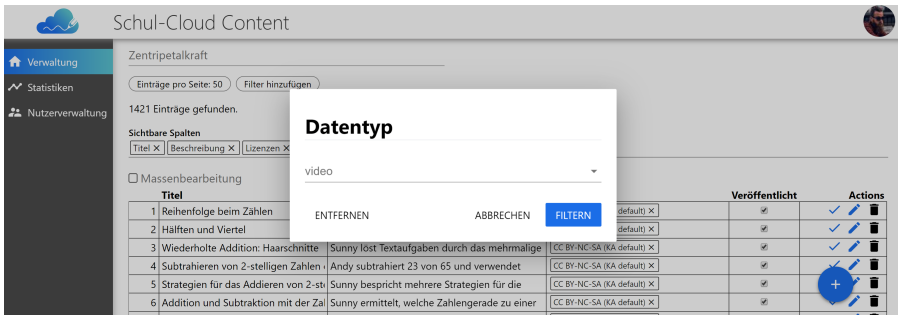

Abbildung 27: Filter - Option von Filter setzen. Wiederverwendung der Select Komponente

LITERATUR

[1] URL: https://github.com/zorn-v/nextcloud-social-login *(Zuletzt zugegriffen am: 17. 07. 2019)*.

[2] URL: https://pqina.nl/filepond/docs/patterns/api/server/#process *(Zuletzt zugegriffen am: 13. 02. 2019)*.

[3] URL: https://docs.feathersjs.com/api/services *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[4] URL: http://www.faqs.org/rfcs/rfc2616.html *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[5] URL: http://www.passportjs.org/docs/authenticate/ *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[6] *AWS-Dokumentation » Amazon Simple Storage Service (S3) » Entwicklerhandbuch » Arbeiten mit Amazon S3-Objekten » Objektschlüssel und Metadaten*. URL: https://docs.aws.amazon.com/de_de/AmazonS3/latest/dev/UsingMetadata.html#object-keys *(Zuletzt zugegriffen am: 10. 07. 2019)*.

[7] Jake Archibald. *In The Loop*. 2018. URL: https://www.youtube.com/watch?v=cCOL7MC4Pl0 *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[8] Nick Babich. "Wizard Design Pattern". In: *UX Planet* (7. Apr. 2017). URL: https://uxplanet.org/wizard-design-pattern-8c86e14f2a38 *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[9] Mike Bell. URL: http://www.passportjs.org/packages/passport-custom/ *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[10] Katharina Blaß und Adrian Jost. *Berichte des Performance- und Accessibility-Tests*. URL: https://github.com/schul-cloud/schulcloud-content-editor/tree/BA-Adrian-Katharina-Test-Results/performance-a11y-tests *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[11] Katharina Blaß und Adrian Jost. *Berichte des Performance-Tests zur Validierung*. URL: https://github.com/schul-cloud/schulcloud-content-editor/tree/BA-Adrian-Katharina-Test-Results/Import%20Performance-Tests *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[12] Marius Blüm. *System requirements - Server - PHP Runtime*. URL: https://docs.nextcloud.com/server/16/admin_manual/installation/system_requirements.html *(Zuletzt zugegriffen am: 17. 07. 2019)*.

[13] Daniel Boterhoven. "Why you should use BCrypt to hash passwords". In: *Medium* (8. Dez. 2016). URL: https://medium.com/@danboterhoven/why-you-should-use-bcrypt-to-hash-passwords-af330100b861 *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[14] Inc Boutell.Com. *WWW FAQs: What is the maximum length of a URL?* URL: https://boutell.com/newfaq/misc/urllength.html *(Zuletzt zugegriffen am: 24. 07. 2019).*

[15] *Concurrency model and Event Loop.* URL: https://developer.mozilla.org/de/docs/Web/JavaScript/EventLoop *(Zuletzt zugegriffen am: 24. 07. 2019).*

[16] Chris Fritz. *Vue.js Documentation - Vue.set.* URL: https://vuejs.org/v2/api/#vm-set *(Zuletzt zugegriffen am: 12. 05. 2019).*

[17] Carolin Goerke und Elias Maaß. "Umsetzung von Digital Rights Management in der HPI Schul-Cloud: Automatisierte DRM Mechanismen, rechtliche Einschränkungen und Alternativen". Bachelorarbeit. Hasso-Plattner-Institut, 2019.

[18] *Inplace Editor Design Pattern.* URL: http://ui-patterns.com/patterns/InplaceEditor *(Zuletzt zugegriffen am: 23. 07. 2019).*

[19] Morris Jobke. *Maintenance and Release Schedule.* URL: https://github.com/nextcloud/server/wiki/Maintenance-and-Release-Schedule *(Zuletzt zugegriffen am: 17. 07. 2019).*

[20] Adrian Jost und Emanuel Metzenthin. *Add New Filter.* URL: https://github.com/schul-cloud/schulcloud-content-editor/wiki/add-new-filter *(Zuletzt zugegriffen am: 14. 07. 2019).*

[21] Bernhard Kau. *Adventskalender Tag 4: Mehrere Artikel auf einmal bearbeiten.* URL: https://kau-boys.de/2206 *(Zuletzt zugegriffen am: 23. 07. 2019).*

[22] Meggin Kearney, Dave Gash, Alice Boxhall und Rob Dodson. *Accessibility.* URL: https://developers.google.com/web/fundamentals/accessibility/ *(Zuletzt zugegriffen am: 24. 07. 2019).*

[23] Hannes Kohlsaat und Ivan Ilic. "Konzeption und Implementierung eines Lizenzeditors für den ODRL Standard im Rahmen der HPI Schul-Cloud". Bachelorarbeit. Hasso-Plattner-Institut, 2019.

[24] Pete LePage. *Responsives Webdesign: Grundlagen.* URL: https://developers.google.com/web/fundamentals/design-and-ux/responsive/?hl=de *(Zuletzt zugegriffen am: 24. 07. 2019).*

[25] David Luecke. URL: https://github.com/feathersjs/feathers/issues/615#issuecomment-314146801 *(Zuletzt zugegriffen am: 24. 07. 2019).*

[26] David Luecke. URL: https://github.com/feathersjs/feathers/issues/397 *(Zuletzt zugegriffen am: 24. 07. 2019).*

[27] Christoph Meinel, Jan Renz, Matthias Luderich, Vivien Malyska, Konstantin Kaiser und Arne Oberländer. *Die HPI Schul-Cloud: Roll-Out einer CloudArchitektur für Schulen in Deutschland. Technische Berichte Nr. 125.* 2019. ISBN: 978-3-86956-453-1. URL: https://s3.hidrive.strato.com/schul-cloud-hpi/Dokumente/Die-HPI-Schul-Cloud_Roll-Out-einer-Cloud-Architektur-f%C3%BCr-Schulen-in-Deutschland.pdf *(Zuletzt zugegriffen am: 23. 07. 2019).*

[28] Sridhar Narasimhan. *Introducing New React Query Builder UI Component*. URL: https://www.syncfusion.com/blogs/post/new-react-query-builder-ui-component.aspx *(Zuletzt zugegriffen am: 23. 07. 2019)*.

[29] NewMediaAngels. *The Importance of Filtering for E-Commerce Websites*. URL: https://newmediaangels.com/importance-filtering-e-commerce-websites/ *(Zuletzt zugegriffen am: 14. 06. 2019)*.

[30] Bryan Nielsen. *FeathersJS Auth Recipe: Custom Auth Strategy*. URL: https://github.com/feathersjs/docs/blob/master/guides/auth/recipe.custom-auth-strategy.md *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[31] Michael Rambeau. URL: https://bestofjs.org/tags/auth *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[32] Anoop Raveendran. "JavaScript Event Loop Explained". In: *Medium* (27. Dez. 2017). URL: https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4 *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[33] Charlie Robbins und Ken Perkins. URL: https://github.com/pkgcloud/pkgcloud#file *(Zuletzt zugegriffen am: 23. 06. 2019)*.

[34] Aswin S. *Score a perfect 100 in Lighthouse audits — Part 1*. URL: https://medium.com/@aswin_s/score-a-perfect-100-in-lighthouse-audits-part-1-3199163037 *(Zuletzt zugegriffen am: 18. 07. 2019)*.

[35] Jonathan Saring. "6 JavaScript User Authentication Libraries for 2019". In: *Medium* (6. Dez. 2018). URL: https://blog.bitsrc.io/6-javascript-user-authentication-libraries-for-2019-6c7c45fbe458 *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[36] Adam Silver. "Better Form Design: One Thing Per Page (Case Study)". In: *Smashing Magazine* (22. Mai 2017). URL: https://www.smashingmagazine.com/2017/05/better-form-design-one-thing-per-page/ *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[37] Max Snitser. "HOW TO IMPROVE UX OF WEB FORMS". In: (9. Dez. 2018). URL: https://maxsnitser.com/blog/how-to-improve-ux-of-web-forms *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[38] Sebastian Springer. *Node.js-Module: Authentifizierung mit Passport*. URL: https://entwickler.de/online/javascript/passport-579800408.html *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[39] Osvaldas Valutis. "Drag and Drop File Uploading". In: *CSS-Tricks* (6. Nov. 2016). URL: https://css-tricks.com/drag-and-drop-file-uploading/ *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[40] Jeremy Wagner. *Why Performance Matters*. URL: https://developers.google.com/web/fundamentals/performance/why-performance-matters/ *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[41] *WebAIM's WCAG 2 Checklist*. URL: https://webaim.org/standards/wcag/checklist *(Zuletzt zugegriffen am: 24. 07. 2019)*.

[42] *Wissenswertes zum DigitalPakt Schule*. URL: \textbf{https://developers.google.com/web/fundamentals/design-and-ux/responsive/?hl=de} *(Zuletzt zugegriffen am: 29. 07. 2019)*.

[43] *eCommerce Site Search Best Practice*. URL: https://www.spacebetween.co.uk/blog/the-importance-of-a-good-ecommerce-search-bar *(Zuletzt zugegriffen am: 14. 06. 2019)*.

ABBILDUNGSVERZEICHNIS

LISTINGVERZEICHNIS

SELBSTSTÄNDIGKEITSERKLÄRUNG

Hiermit versichere ich, dass ich die Bachelorarbeit selbständig und ohne unzulässige Hilfe Anderer angefertigt habe und dass ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit besonders gekennzeichnet und die entsprechenden Quellen angegeben habe. Ich erkläre hiermit weiterhin die Gültigkeit dieser Aussage für die Implementierung des Projektes.

Potsdam, 29. Juli 2019

Katharina Blaß

Adrian Jost