Dokumentation: Entwicklung einer Spike Solution zur Literatureingabe im ModCat

1. Projektbeschreibung

Ziel des Projekts war die Erweiterung des ModCat-Protoypen mit einer Funktion zum Einpflegen von Literatur. Dazu sollte eine Spike-Solution implementieren werden.

Kurzbeschreibung:
In einem Modul werden Literaturquellen referenziert. Beim Anlegen einer Literaturquelle sind die Metadaten wie Titel, Autoren usw. einzugeben. Es ist notwendig, den Populationsprozess möglichst zu automatisieren.

Aufgaben:
Services für Abfragen der Metadaten von Literaturen recherchieren, die notwendigen Funktionalitäten erheben und das User Interface entwerfen inkl. Implementierung der notwendigen Funktionen.

Einzusetzende Technologien:
Für HTTP-Requests soll die Abfrage der Metadaten über RESTful APIs erfolgen. Die Anwendungstechnologien JavaScript und Vue.js sind zu nutzen und für Graph-Änderungen Turtle sowie SPARQL.

2. Vorgehen

Zur Umsetzung der in Kapitel 1 beschrieben Ziele musste auch mit Blick auf die Corona-bedingte Fernarbeit eine möglichst zielführende Vorgehensweise entwickelt werden. In Absprache mit den Kursleiterinnen wurde sich auf folgendes Vorgehen geeinigt:

- Identifizieren und Planen von Arbeitspaketen, die zur Erfüllung der Projektziele führen. Diese wurde mit Beschreibungen, Zuständigkeiten und Daten in einem gemeinsamen Trello-Board eingepflegt. So konnte zudem der Fortschritt auf Seiten des Projektteams und der Kursleiterinnen geordnet verfolgt werden
- Wöchentliche Meetings mit den Kursleiterinnen, in denen Fortschritte und Probleme gemeinsam diskutiert wurden. Gegen Ende des Projekts wurden daraus Ad-Hoc-Meetings.
- Nutzung von Teams, um auf allen Seiten auch außerhalb von Meetings kurze Fragen schnell zu beantworten.
- Entwicklung der Spike Solution mit Docker und Visual Studio Code. Es wurde sich für Docker entschieden, um auf allen PCs der Projektteilnehmer eine identisch reproduzierbare Entwicklungsumgebung zu schaffen. Änderungen wurden in einem

GitHub-Repository festgehalten, dessen Grundlage ein Fork des bestehenden ModCat-Repositories darstellt.

Im nächsten Schritt wurden die Arbeitspakete durch die Projektteilnehmer bearbeitet. Diese werden an dieser Stelle kurz vorgestellt.

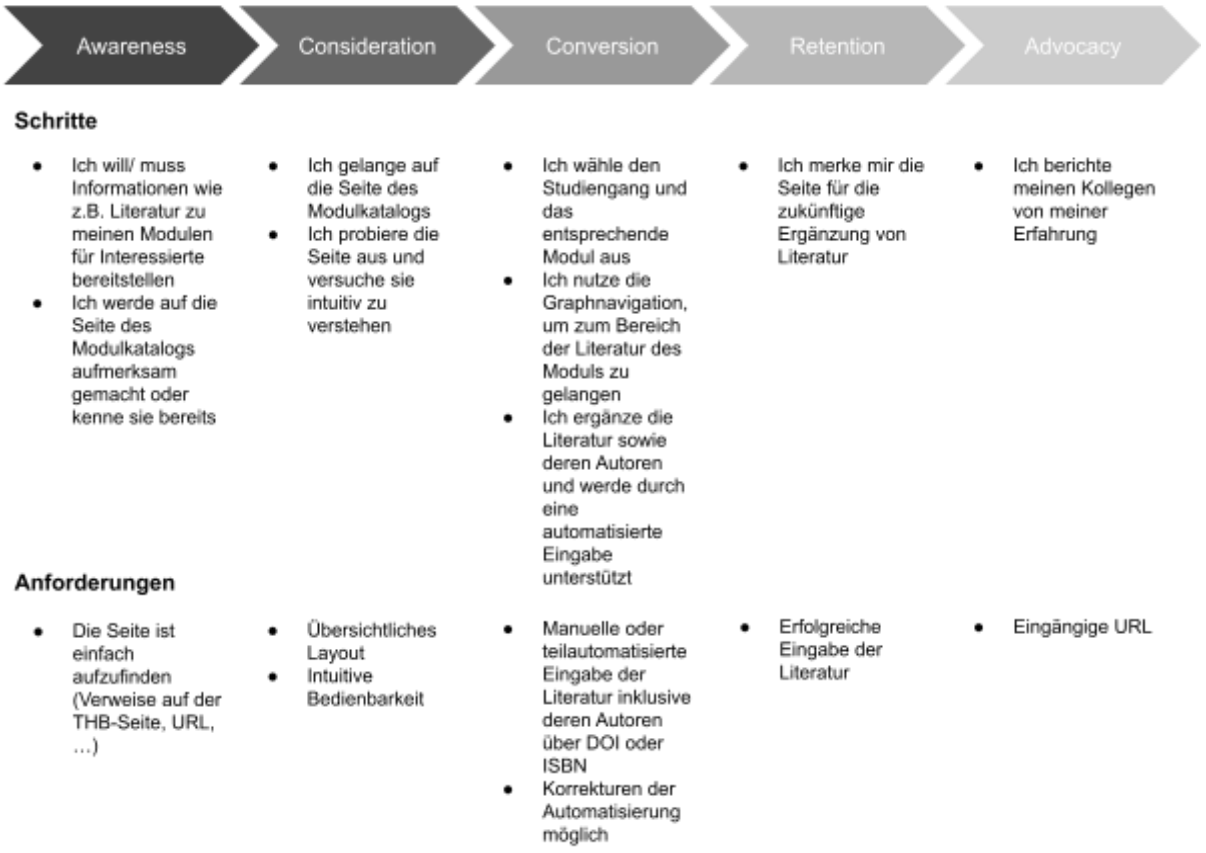
AP1: Identifizieren von Datenfeldern (IST-Analyse)
Zur Erweiterung des ModCats mit einer Funktionalität für die Literatureingabe musste zunächst geklärt werden, über welche Datenfelder die Graph-Datenbank grundsätzlich verfügt.

AP2: Identifizieren von APIs
Mit Blick auf das halbautomatisierte Einpflegen von Literatur mussten APIs identifiziert werden, über die relevante Informationen von Literatur automatisiert abgerufen werden können. Identifiziert wurde die doi.org-API für das Abrufen von Informationen mit Hilfe einer DOI und die Google Books API für das Abrufen anhand einer ISBN. Zudem wurde eine Möglichkeit gefunden zu prüfen, ob Literatur mit einer ISBN an der örtlichen Hochschulbibliothek vorhanden ist.

AP3: Schema-Erweiterung (SOLL)
Nachdem der IST-Zustand und die Felder einer API-Response bekannt waren, konnten fehlende Felder im Graph-Schema identifiziert werden. Als einzige Erweiterung identifiziert wurde das Attribut isbn, das im Graphen die Domain Book (schema:Book) erweitern soll.

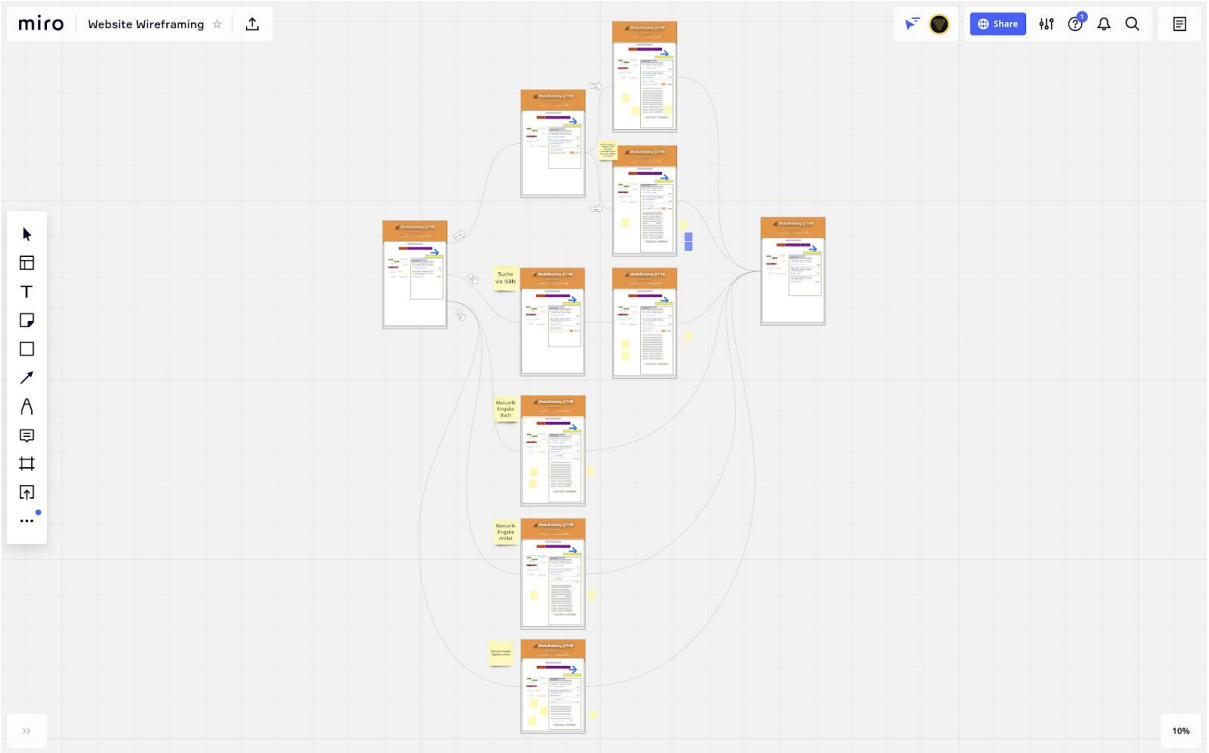
AP4: User Journey erstellen
In diesem AP wurde eine grundlegende User Journey erstellt, die den Prozess der Literatureingabe durch Lehrende darstellt. Dazu wurde sich am ACCRA-Modell orientiert, dessen Ergebnis in der folgenden Abbildung zu sehen ist.

Lehrende



AP5: Frontend-Wireframes erstellen

Im nächsten Schritt sollten auf Basis der User Journey weitreichende Wireframes entwickelt werden. In der Durchführung wurden jedoch MockUps erstellt, da sie einen klareren Blick über den Aufbau und die Funktionsweise bieten. In der folgenden Abbildung ist ein Überblick über die in Miro erstellten Views zu sehen.



Abgebildet werden die unterschiedlichen Sichten, je nachdem ob eine halbautomatisierte Eingabe über DOI oder ISBN stattfindet, oder ob alle Eingaben manuell stattfinden.

AP6: SPARQL-Abfragen erstellen

Nachdem die Views und der Prozess konzeptuell modelliert wurden, wurden notwendige SPARQL-Queries definiert, die zum Abrufen und Ändern von Daten in der Graph-Datenbank notwendig sind. Zum Testen der Queries wurde der Fuseki-Server der TH-Brandenburg verwendet.

AP7: Vue.js-App erweitern

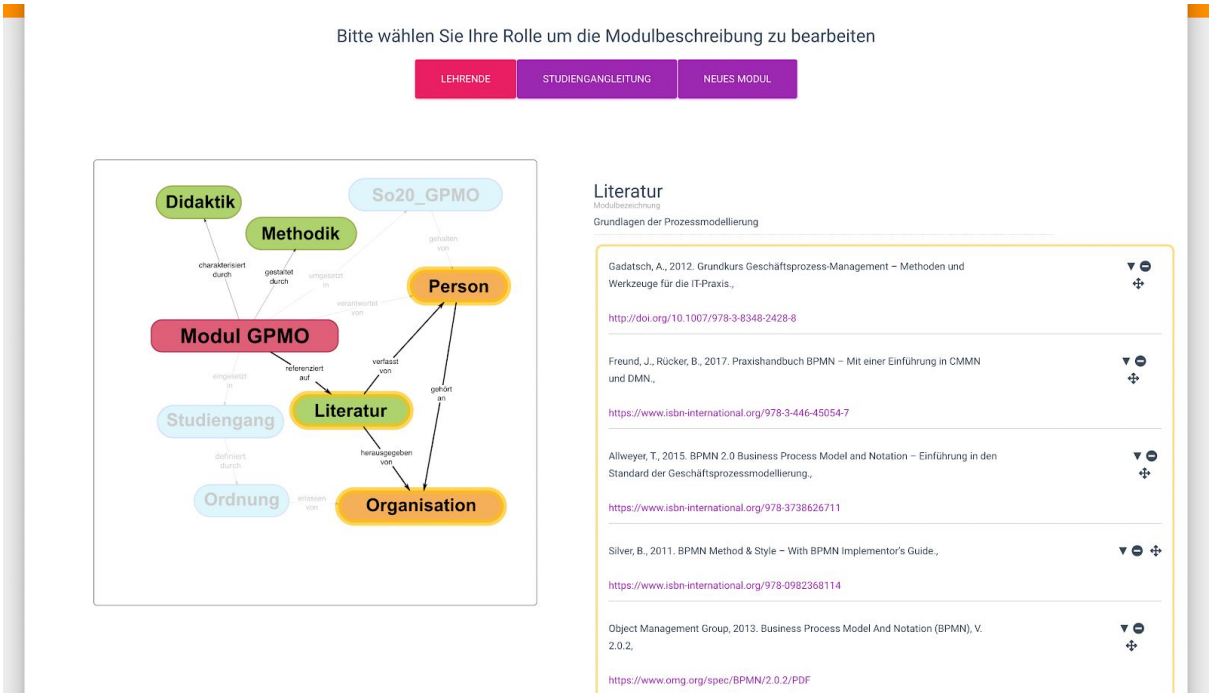
In diesem letzten Schritt wurden alle Ergebnisse der letzten APs zusammengeführt und teilweise in Form einer Spike-Solution umgesetzt. Dazu wurde das bestehende ModCat-Projekt durch das Projektteam beispielhaft abgeändert. Dieses AP wird im nächsten Kapitel näher beleuchtet.

3. Erweiterung Vue-App

In diesem Kapitel wird die Umsetzung der Spike Solution kurz vorgestellt. Diese beinhaltet noch diverse Fehler und nicht implementierte Features, die für ein Produktivsystem notwendig wären. Dennoch werden wichtige Dinge implementiert und die grundsätzliche Funktionalität demonstriert.

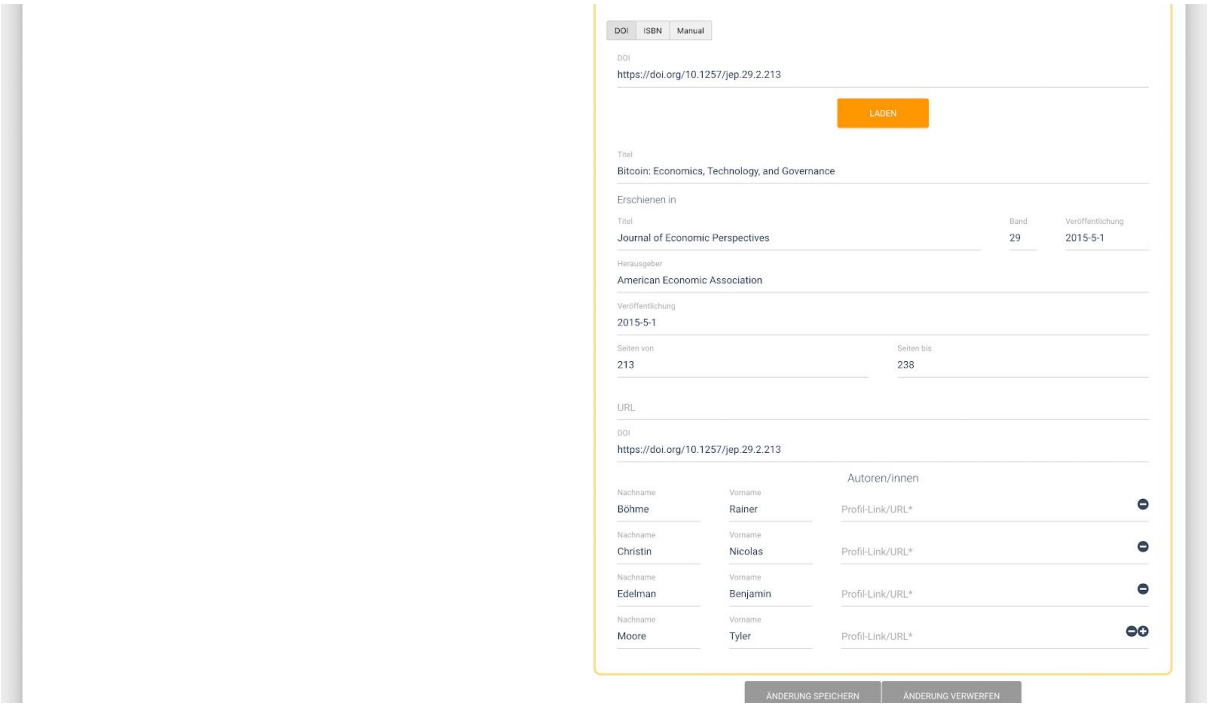
Wie im letzten Kapitel angedeutet, besteht die Literatureingabe aus mehreren Views. Nachdem ein Studiengang, ein Modul und die Rolle ausgewählt wurde, kann in der Graphnavigation der Punkt “Literatur” ausgewählt werden, wodurch das Laden der

FormLiterature.vue angestoßen wird. Hierzu wurden Änderungen an der SVGGraph.vue und modcat.svg vorgenommen. Auf dieser Übersichtsseite findet sich die gesamte Literatur eines Moduls. Dabei bestehen die Einträge aus einer Kurzbeschreibung der Literatur mit z.B. Autoren und Titel. Mit einem Klick auf das Dreieck lässt sich die Kurzbeschreibung aufklappen, wodurch mehr Metadaten zur Literatur dargestellt werden. Mit einem Klick auf das Minus- bzw. Plus-Symbol können Einträge entfernt oder hinzugefügt werden.



Nach dem Klick auf das Plus-Symbol erscheint am Ende der Seite eine Komponente, die je nach Klick auf einen Tab-Eintrag die entsprechenden Subview lädt, die zur halbautomatisierten oder manuellen Eingabe der Literatur genutzt werden kann. Mit Klick auf “Änderung speichern” werden die Insert-Queries der Subviews mit möglichen Delete-Queries verbunden und gemeinsam an den Fuseki-Server geschickt. Die Subviews halten die FormLiterature.vue durch Events und manuelle Aufrufe jederzeit anhand der aktuellen Eingaben mit der passenden Query auf dem Laufenden.
Im Folgenden werden die drei angesprochenen Subviews kurz mit ihren Funktionen vorgestellt.

FormLiteratureDOI.vue
Mit Klick auf den DOI-Tab in der FormLiterature.vue wird diese Subview geladen. Hier bekommt der Nutzer mit Hilfe einer angegeben DOI und der doi.org-API ein vorausgefülltes Formular mit den Metadaten der Literatur.



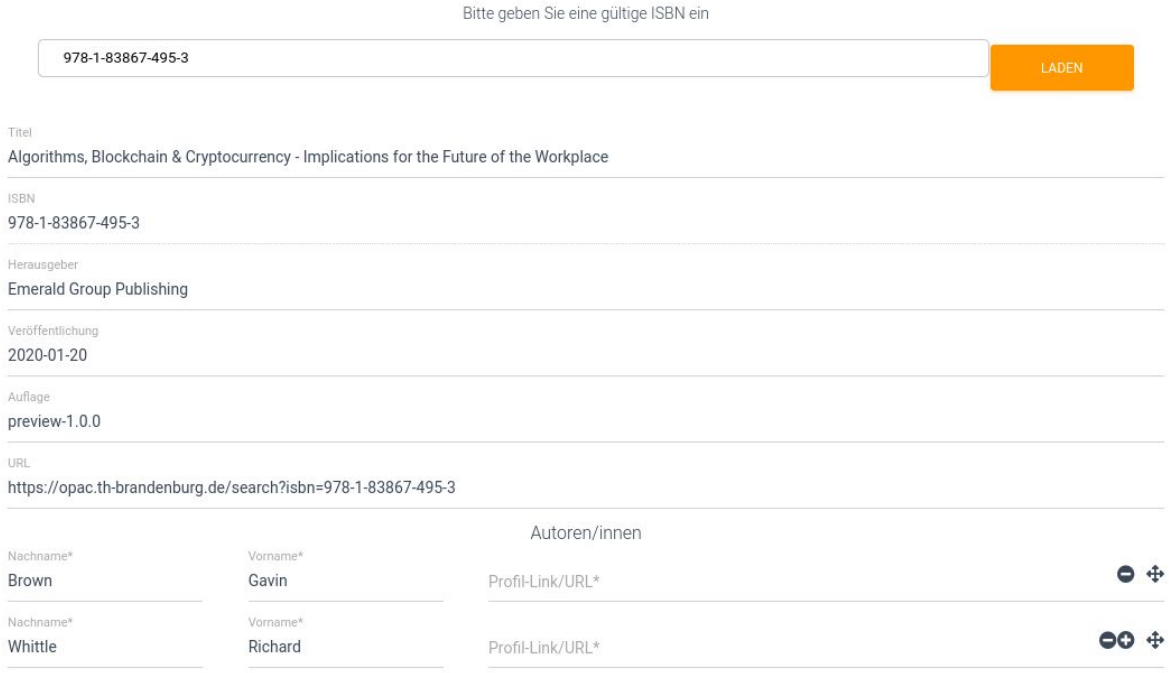
Mit Klick auf den Laden-Button wird mit der angegebenen DOI eine Anfrage mit Hilfe der queryDoiData-Methode an die doi.org-API gesendet, dessen Ergebnis in der rawDoiData-Variable gespeichert wird. Das Ergebnis wird dann in der cleanedDoiData-Computed-Variable umstrukturiert, um auf Variationen im Antwort-Schema einzugehen und im HTML die Ergebnisse sauber darzustellen. Mit Änderung dieser Computed-Variable wird zudem im watch-Teil geprüft, ob diese DOI eine ISBN besitzt. Ist dies der Fall, wird versucht ein OPAC-Link zu generieren. Dazu wird ein Permalink mit der ISBN generiert und mit axios aufgerufen. Enthält die Antwort des OPACs den String “in die Merkliste”, so ist die Literatur im OPAC vorhanden und der Link in cleanedDoiData wird durch den OPAC-Link ersetzt. Die Inhalte der cleanedDoiData werden nebenbei automatisiert in das Formular übertragen, wo sie durch den Nutzer angepasst werden können. Die Felder des Formulars unterscheiden sich je nachdem, ob es sich bei der Literatur hinter der DOI um ein Buch oder einen Artikel handelt. Zudem wurden erste Grundlagen zur Prüfung von Autor-Duplikaten entwickelt. Gibt ein Nutzer einen Autoren ein, dessen Vor- und Nachnamen es schon in der Graph-Datenbank gibt, so erscheint ein PopUp. In diesem kann der Nutzer aus einer Liste von Autoren mit gleichem Namen auswählen, ob der eingegeben Autor schon im System bekannt ist. Die Daten aus dieser Auswahl werden beim Anlegen des Autors aktuell jedoch noch nicht berücksichtigt. Über ein Change-Event am Formular und an diversen anderen Stellen wie z.B. nach der Query und dem Abrufen des OPAC-Links wird die generateQuery-Methode aufgerufen. In dieser wird mit Hilfe der Daten im Formular eine Insert-Query generiert. Dessen Ergebnis wird durch das Aussenden eines Events an die übergeordnete FormLiterature.vue weitergeben. So kann sichergestellt werden, dass alle Änderungen gemeinsam an den Fuseki-Server gesendet werden können. Alle weiteren Subviews folgen in ihrer Grundlogik eine sehr ähnliche Logik. Es ist zudem zu erwähnen, dass es in der View noch einige Bugs gibt und

Edge-Cases nicht behandelt werden. So erscheint das PopUp z.B. nur, wenn der Nutzer manuell die Autoren eingibt oder ändert. Und durch die fehlende Dokumentation der API und die vielen Fälle werden aktuell noch nicht alle möglichen Fälle der API-Antworten behandelt. Ähnlich zu den nächsten Subviews ist auch diese somit noch sehr Fehleranfällig.

FormLiteratureISBN.vue

Nach dem Klick auf das Plus für Literatur hinzufügen und nach der Auswahl (Klick auf Button) “ISBN” wird die FormLiteratureISBN.vue geladen.

Es erscheint in einem weiteren Eingabefeld die Aufforderung eine ISBN einzugeben. Nach der Eingabe und Klick auf “Laden” wird das zugrunde liegende HTML-Formular mit den Metadaten der geladenen Literatur befüllt (siehe Abbildung).



Der Nutzer kann nun bis auf das ISBN-Feld Änderungen vornehmen und die Literatur bearbeiten. Im Folgenden wird auf einige wichtige Methoden eingegangen.

Die Methode cleanedISBNData() erzeugt das cleanedData-Objekt in welchem die API-Response gespeichert wird und an einigen Stellen zur späteren Query-Generierung erweitert wird.

Die Methode addAutor() wird aufgerufen, wenn jemand auf das Plus-Symbol zum Autor*in hinzufügen klickt und erweitert das Array der Autoren im cleanedData-Objekt.

Die Methode removeAutor() wird aufgerufen, wenn jemand auf das Minus-Symbol zum Autor*in entfernen klickt und reduziert das Array der Autoren im cleanedData-Objekt um den bestimmten Autor*in.

Die Methode checkAutor() prüft ob der hinzugefügte Autor schon vorhanden ist und wiederverwendet werden kann, indem es eine Query zum Fuseki schickt und den prüft, ob der Autor schon vorhanden ist.

Die Watch-Methode existingAuthors() erzeugt das PopUp bei korrekter Änderung der Variablen.
Die Methode handleAutorSelection() gibt das Ergebnis der Auswahl im PopUp zurück und der Nutzer*in kann prüfen, ob es sich dabei um ein und dieselbe Person handelt oder doch ein anderer Autor gemeint ist und kann entweder den vorhandenen Autor auswählen oder verwerfen und weiter fortfahren.
Die Methode queryISBNData() erzeugt den API-Request der Abfrage der ISBN und schreibt die Response in das Objekt rawISBNData.
Die Methode generateQuery() erzeugt die Query, welche zum erzeugen der Literatur (inkl. Autor und Herausgeber) als Buch zum Fuseki gesendet wird.
Die Computed-Methode cleanedISBNData() restrukturiert die API-Response, wobei die gleichnamige Watch-Methode prüft, ob es einen OPAC-Link zu der Literatur gibt und tauscht den vorhandenen Link gegen diesen aus.
Es gestaltete sich als problematisch, die API-Response zu verarbeiten und die vue mit Daten zu befüllen. Ein weiteres Problem war die Erweiterung der Autoren*innen. Am zeitintensivsten gestalteten sich die Fehlerbehebungen und Bugfixes von kleineren Problemen die bei der Entwicklung auftraten. Die größte Herausforderung war die Kommunikation mit dem Fuseki und das formulieren der Queries, welche zwar funktionieren, aber sicher nicht perfekt sind.
Es wurden bei weitem noch nicht alle Fehler abgefangen. So kann es vorkommen, dass bei fehlenden Parametern der API-Response das Formular nicht entsprechend geladen wird oder das Herausgeber und Autoren doppelt angelegt werden. Es kann auch vorkommen, dass unvollständige Literatur angelegt wird, da noch keine Pflichtfelder definiert wurden.

FormLiteratureManual.vue
Mit einem Klick auf den Tab Manuell wird schließlich das Eingabeformular zur manuellen Eingabe einer neuen Literaturquelle im unteren Bereich der Literaturseite geladen. Im Gegensatz zu den beiden anderen Optionen müssen die Daten an dieser Stelle komplett von Hand eingegeben werden.
Als erstes muss dazu über eine DropDown-Menü ausgewählt werden, ob ein Buch (default), ein Artikel oder ein sonstiges digitales Dokument als Literaturquelle neu erfasst werden soll. Mit der DropDown-Auswahl wird dann die Sichtbarkeit der entsprechenden Formularfelder gesteuert. Im Anschluss können dann die einzelnen Daten der Literaturquelle, wie z. B. Titel, Herausgeber, Datum der Veröffentlichung bis hin zu den Autoren manuell erfasst werden.
Am Ende der Eingabe wird dann, wie bei den beiden anderen Optionen DOI und ISBN, ein Query-String aus den Eingaben zusammengesetzt (generateQuery()), der an das zentrale Formular FormLiterature.vue übergeben wird. Mit dem Klick auf den Button “Änderung speichern” wird danach die Insert-Query erzeugt und wie oben beschrieben an den Fuseki-Server gesendet.

4. Zusammenfassung und Ausblick

Zusammenfassend sollen die erreichten Ergebnisse, aber auch die im Zuge des Projektverlaufes ersichtlich gewordenen ToDo´s und weiteren Optimierungsbedarfe dargestellt werden.
Folgende Funktionalitäten wurden von den Projektteilnehmern realisiert, teilweise realisiert bzw. bergen weitere Optimierungsbedarfe:

- Zu den einzelnen Modulen können Literaturquellen komplett neu angelegt werden und bestehende Quellen (Bezug zum Modul) können auch wieder gelöscht werden.
- Literaturquellen können über die drei Optionen DOI, ISBN oder manuell neu angelegt werden, dabei können die Angaben vor dem Speichern jeweils manuell korrigiert werden.
- Autor/innen können mit Vor-, Nachnamen sowie einem entsprechenden Profillink erfasst werden, wobei eine automatisierte Suche der ORCID bzw. einer XING-, Researchgate- oder LinkedIn-ID bisher nicht realisiert ist. Hier ist der Nutzer aktuell noch gefragt.
- Zu den Literaturquellen können die Herausgeber/ Verlage erfasst werden, bei denen die URI, wie bei den Autor*innen, jedoch mit Hilfe einer UUID (Universally Unique Identifier) realisiert ist. Der Abgleich und die Zuordnung einer Wikidata-IDs sowie die Vermeidung von Redundanzen bleibt als Optimierungsbedarf bestehen.
- Eine Validierung bzw. Prüfung der Pflichtangaben in Bezug auf die Eingabefelder findet bisher noch nicht statt und müsste noch implementiert werden.
- Im zentralen Literaturformular wurde eine Literaturübersicht implementiert, die über einen Ausklappmechanismus verfügt. Im Sinne von Optimierungen sollten ggf. die Literaturzusammenfassungen (summary) überarbeitet, die Ausgabefelder in der ausgeklappten Literatur erweitert sowie die Funktionalität zum Verschieben der Literaturposition implementiert werden.
- Als weiterer Optimierungsbedarf muss abschließend noch das Design sowie das Auffinden von Fehlern bzw. Testing benannt werden. Zwar wurde die App gegen Ende der Projektlaufzeit von den Teilnehmern nochmal entsprechend getestet und aufgefundene Fehler beseitigt, dennoch können bei anderen, neuen Eingaben Fehler nicht komplett ausgeschlossen werden.

Insgesamt konnte von den Projektteilnehmern eine umfassende Grundlage für die Erfassung von Literaturquellen erarbeitet werden. So konnten diverse wichtige Funktionen, wie z.B. die Anbindung an APIs und die Kommunikation mit der Graph-Datenbank realisiert werden. Dennoch ist das Potenzial für weitere Funktionen und einer erhöhten Stabilität der Spike Solution ersichtlich.