Übung 04
Catalog mit Vue / AXIOS
SWP-W
5abHWII

November 12, 2024

Übungsleiter: Albert Greinöcker

Ziel der Übung:

- Kommunikation mit einem Server mittels AXIOS
- Wiederholung von REST mit Flask

1 Allgemeine Aufgabenstellung

Die Universitätsbibliothek Graz hat vor einiger Zeit die Digitalisierung ihres Zettelkataloges veranlasst, ein Teil dieses Datenbestandes ist die Basis für diese Übung[1].

2 Vorbereitung

2.1 Server

Es existiert bereits ein Server (unter den vue.js-Beispielen unter flask/catalog zu finden). Hier ist alles wie gehabt, eine Neuerung ist dieser import und die Anwendung von CORS:

```
from flask_cors import CORS # Muss mit pip installiert werden
cors = CORS(app)
```

Per default ist der Zugriff von Webseiten auf externe Ressourcen aus Sicherheitsgründen unterbunden, diese eine Anweisung erlaubt das.

Es existiert eine Suchfunktion:

```
http://127.0.0.1:5000/cat-search/mitterer
```

und ein REST-Service, der einzelne Karten Abfragen, editieren und löschen kann:

```
http://127.0.0.1:5000/cat-item/12514600 (HTTP GET)
```

[1] Die Originalanwendung ist unter https://webapp.uibk.ac.at/alo/cat/startpage.jsp zu finden

2.2 Datenbank

Es gibt ein DDL file (`catalog.sql`) im Projekt und in der Moodle-Abgabe. Mit diesem Befehl wird eine SQL-Datenbank erzeugt und befüllt:

```
mysql -uroot -p < catalog.sql
```

Die Bibliothek `PyMySQL` muss dafür installiert werden.

Die erzeugte Datenbank ist relativ überschaubar:

```
CREATE TABLE card (
    ID INT PRIMARY KEY,
    DESCRIPTION VARCHAR(1024),
    THUMB VARCHAR(512)
);
```

Wer `SQLite` bevorzugt: Es ist ein fix fertiges Datenbank-File bei der Moodle-Abgabe[2]. Man muss dann allerdings in der Anwendung noch den Pfad für die einzelnen Bilder vor dem relativen Pfad in der Datenbank angeben: `http://webapp.uibk.ac.at/ubi/cat/`

2.3 Client

In dieser Anwendung sollten wir schon `SFC`[3] verwenden. Dafür müssen folgende Voraussetzungen geschaffen werden bzw. Installation vorgenommen werden.

2.3.1 node.js und npm

2.4 Erzeugen/Holen eines SFC-Projektes

Für das Erzeugen eines boilerplate-Codes für ein Projekt gibt es nach den Installationen folgende Möglichkeiten:

a. `vue create` (siehe `https://cli.vuejs.org/guide/creating-a-project.html`

b. `npm create vue@3` bzw. `npm init vue@latest` Nicht benötigte Komponenten müssen gelöscht werden

c. Kopieren des Projekts `03_sfc/01_first` aus unseren vue-projekt auf github.

Dann muss man in das entsprechende Verzeichnis wechseln, wo der Code liegt und diese Befehle ausführen:

a. `npm install` # Hole die Abhängigkeiten und baue das Projekt

b. `npm run lint` # Führt eine statische Codeanalyse durch

c. `npm run dev` # Startet die Website im developer mode

d. `npm run build` # baut statische HTML-Seite für den Produktivmodus (liegt alles im Verzeichnis `dist`)

2.4.1 vue als node-Modul

Wir arbeiten jetzt nicht mehr mit Javascript-importen sondern mit Modulen, die lokal zur Anwendung installiert werden und dann in den Compilierungsprozess eingebunden werden. Über `npm` lässt sich `vue` leicht installiern:

```
npm init vue@latest
```

[2] Wurde auf ähnliche Weise erzeugt: `sqlite3 catalog.db < catalog.sql`
[3] Single File Components

2.4.2 axios

Die Bibliothek die die Kommunikation mit dem Server übernimmt muss auch noch installiert werden:

```
npm install axios
```

Nach der Installation kann dann axios auf der Javascript-Seite folgendermaßen eingebunden werden:

```
import axios from "axios";
```

3 Aufgabenstellung

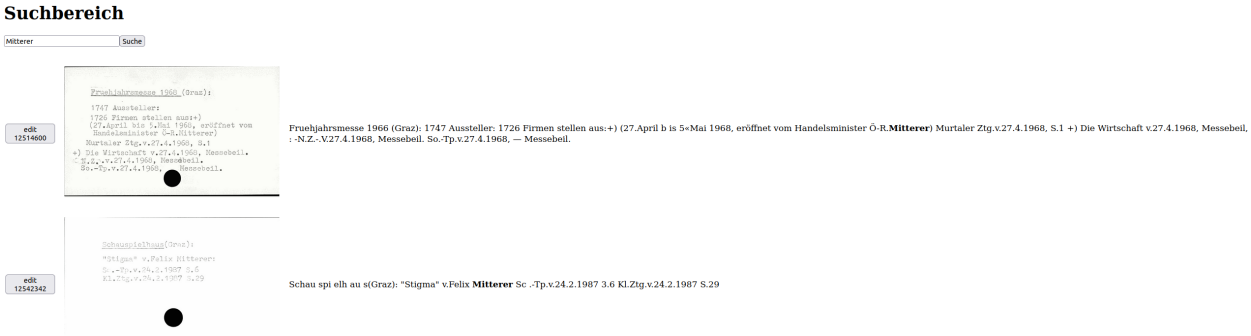
Grundsätzlich sind in der Datenbank die volltexterkannten Texte (`description`) und die Bilder (`thumb`) dazu. Ziel soll es sein, folgendes zu implementieren (mittels Aufruf der Services):

a. Suche über die Texte

b. Editieren der Texte

3.1 Suche

Ein einfaches Eingabefeld soll erstellt werden, wo ein Wort eingegeben wird. Beim Drücken auf einen Button wird eine Javascript-Methode gestartet, die den entsprechenden Service mit `Axios` aufruft. Das Ergebnis wird gleich einer Variable übergeben , die für die Anzeige der Suchergebnisse verwendet wird. So könnte das aussehen (kann auch schöner aussehen):



3.1.1 Hilfestellungen dazu

Aufruf der Services Am Besten man gibt die Funktion in eine eigene Javascript-Datei. Das sieht dann so aus:

```
export async function api_search(q)
{
    const response = await axios.get(‘http://localhost:5000/cat−search/${q}‘,{});
    return response.data;
}
```

Wenn es in einer eigenen Javascript-Datei steht entspricht das einem Modul. Mit `export` kennzeichnet man dann die Methoden und Variablen, die man für die Verwendung von außen zur Verfügung stellen möchte. Der Import in der Vue-Komponente schaut dann so aus:

```
import {api_search} from "@/api.js";
```

So kann man diese dann verwenden:

```
this.res = await api_search(this.q);
```

Darstellung der Ergebnisse Die Variable `res` muss natürlich im `data()` - Bereich deklariert werden. Für die Anzeige der Suchergebnisse würde sich z.B. eine Tabelle in Kombination mit `v-for` anbieten:

```
<table>
    <tr v-for="r in res" >
        <td>Hier kommen die Spalten hin</td>
        <td>Hier kommen die Spalten hin</td>
        <td>Hier kommen die Spalten hin</td>
    </tr>
</table>
```

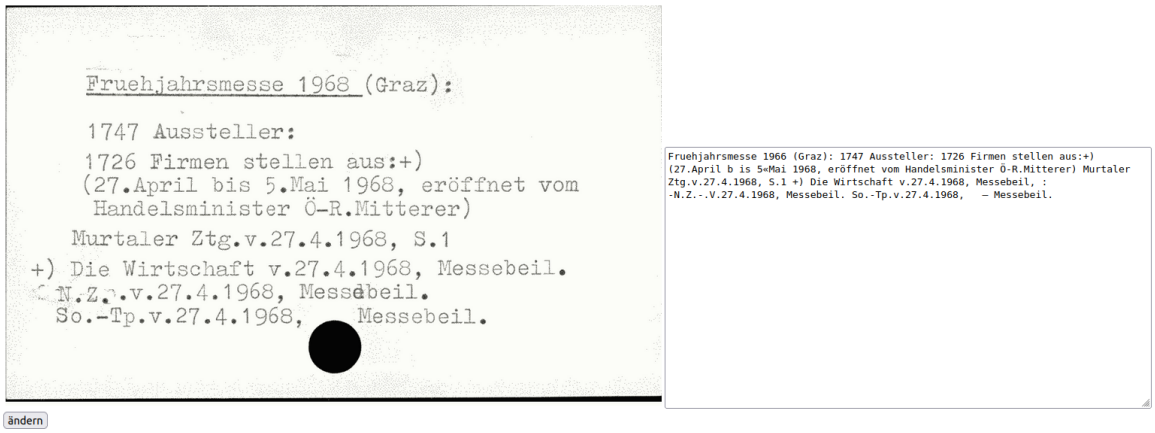
Kennzeichnen der Suchtreffer Es kann, so wie im Bild gezeigt, mal versucht werden, das gesuchte Wort fett darzustellen. Das geht sehr einfach mit der map-Funktion auf das Suchergebnis-Array in Kombination mit der String-Methode `replace`.

3.2 Editieren

Wenn man in den Suchergebnissen den Button links drückt, dann soll z.B. oben ein Bereich aufgehen, der den Text in einem Textarea und das Bild daneben anzeigt. Beim Drücken auf einen Button soll dann eine Methode aufgerufen werden, die den Inhalt des Textes in der Datenbank über den Service speichert (Servermethode PATCH bereits implementiert).

So könnte das Editieren aussehen:

Editierbereich



3.2.1 Hilfestellungen dazu

Aufruf des Services Gleich wie oben muss eine AXIOS-Methode erstellt und in der Vue-Komponente importiert werden. Zum Unterschied von voher ist die Aufrufmethode patch und man muss einen Parameter (c entspricht dem editierten Kärtchen) mitgeben:

```
export async function api_patch(c)
{
    const response = await axios.patch(‘http://localhost:5000/cat−item/${c.id}‘,
    {
        params:
        {
            description : c.description
        }
    });
    return response.data;
}
```

Für das Holen des Kärtchen-Objektes empfiehlt es sich eine neue AXOS-Methode zu schreiben, die mittels GET die entsprechenden Infos abholt (ist am Server schon implementiert).

3.3 Änderungs-History

Damit die Änderungen auch protokolliert sind, müssen diese zusammen mit dem Änderungsdatum in der Datenbank in einer eigenen Tabelle gespeichert werden. Folgendes ist dafür zu tun:

```
def init_db():
# Erzeugen der Tabellen fuer die Klassen, die oben deklariert sind (muss nicht sein, wenn diese schon existiert)
    Base.metadata.create_all(bind=engine)

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

3.3.1 Änderung am Server

Ein eigener REST-Service für diese Änderungen soll erstellt werden, der die Änderungen abfragt. Es sind nur 3 Attribute notwendig: `id`, `datum` und `description`.. De Rest-Service ist gleich wie `CatalogREST` aufzubauen und ein SQL-Alchemy-Objekt so wie `Catalog`.

3.3.2 Weitere AXIOS-Calls

Natürlich müssen diese dann auch von der Clientseite her aufgerufen werden.

3.3.3 Anzeige

Beim Editieren müssen die einzelnen Versionen auch angezeigt werden und wenn man darauf klickt, wird der Version aus der History als aktueller Text gespeichert.