Hausaufgabe 4: Vue.js

Laden Sie ihre Lösung für die Hausaufgabe in einer ZIP-Datei auf ISIS bis zum 24. Januar um 23:59 Uhr hoch. Die ZIP-Datei soll dabei ausschließlich den Ordner `ha4` mit dessen Inhalt wie in der Vorgabe gegeben enthalten (Der Inhalt der Dateien muss natürlich nach Aufgabenstellung bearbeitet werden). Keine weiteren Dateien sollen hinzugefügt oder entfernt werden. Auch sollen keinen Dateien oder Ordner umbenannt werden.

Vorbereitung

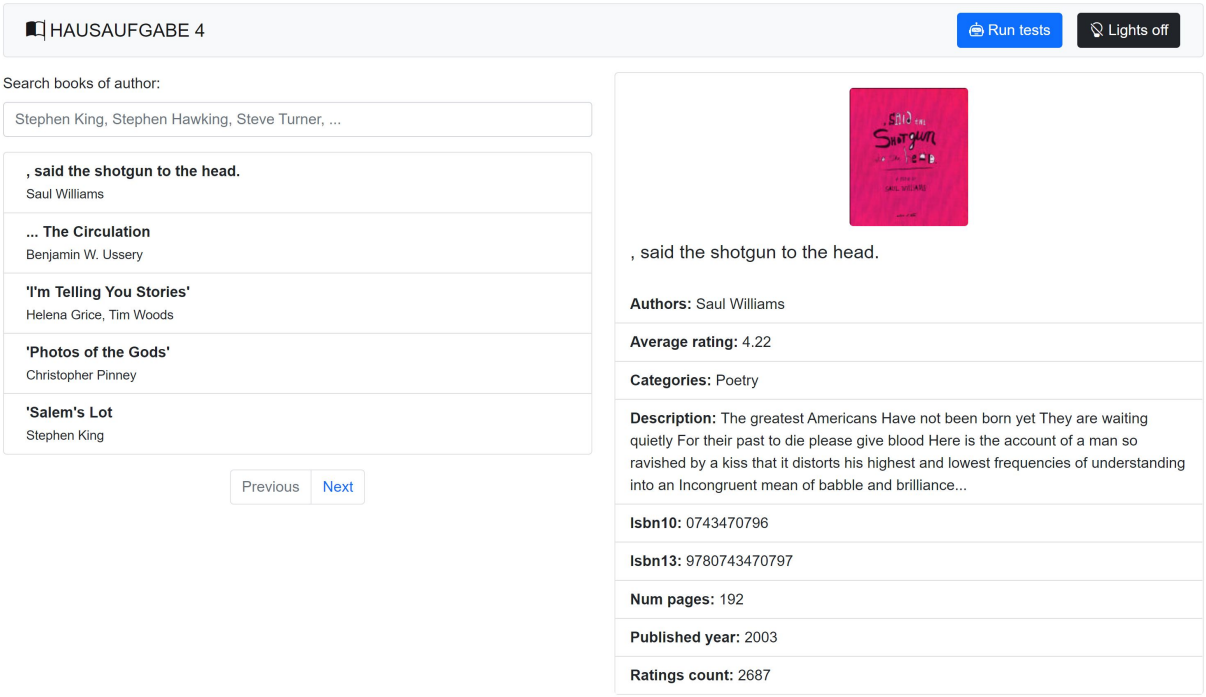

Abbildung 1: Screenshot der Vorgabe nach dem ersten Start. **Beachten Sie:** Die meisten Komponenten wurden als Bilder eingefügt, um Ihnen eine Vorstellung davon zu geben wie die Seite am Ende aussehen soll bzw. kann.

Zur Bearbeitung müssen Sie Node.js installieren. Gehen Sie dazu auf `https://nodejs.org/`, laden Sie die entsprechende Version[1] für Ihr Betriebssystem herunter und installieren Sie im Anschluss Node.js auf Ihrem Rechner. Danach können Sie den *Node Package Manager (kurz: npm*[2]*)* nutzen. Diesen benötigen wir um die Vorgabe starten zu können.

Installieren Sie zunächst die Vue.js Command-line Tools *vue-cli* mit folgendem Befehl:

```
npm install -g @vue/cli
```

Extrahieren Sie dann den Ordner `ha04/` aus der Vorgabe an einem beliebigen Ort auf Ihrem Rechner. Gehen Sie dann im Terminal in das Verzeichnis `ha04/` und führen Sie dort folgenden Befehl aus.

[1] idealerweise LTS-Version v16.13.1
[2] idealerweise v.8.1.2

```
npm install
```

Im Anschluss werden weitere Pakete installiert, die für die Ausführung der Vorgabe benötigt werden. Ist der Installationsvorgang abgeschlossen können Sie mit folgendem Befehl die Vorgabe starten.

```
npm run serve
```

Gehen Sie im Browser auf `http://localhost:8080/`. Sie sollten nun die oben abgebildete Seite (siehe Abbildung 1) sehen.

Grober Aufbau

In `ha04/src/assets/` finden Sie eine JSON-Datei `books.json`. Dieser Datensatz enthält Daten zu knapp 7000 Büchern. Betrachten Sie den folgenden Eintrag als Beispiel:

```
1  {
2    "isbn13": 9780002005883,
3    "isbn10": "0002005883",
4    "title": "Gilead",
5    "subtitle": "",
6    "authors": "Marilynne Robinson",
7    "categories": "Fiction",
8    "thumbnail": "%THUMBNAIL_URL%",
9    "description": "A NOVEL THAT READERS ...",
10   "published_year": 2004,
11   "average_rating": 3.85,
12   "num_pages": 247,
13   "ratings_count": 361
14 }
```

In dieser Hausaufgabe soll eine Web-Anwendung in Vue.js entwickelt werden mittels der die Daten aus dem Datensatz angezeigt werden können. Diese Web-Anwendung muss:

- den Datensatz einlesen und die Daten aufbereiten,
- mehrere Einträge in einer kompakten Listenansicht visualisieren,
- diese Listenansicht navigierbar machen (Stichwort: *pagination*),
- ermöglichen das ein Eintrag aus der Listenansicht ausgewählt werden kann, damit dieser in einer Detail-Ansicht genauer betrachtet werden kann
- und dem Nutzer erlauben gezielt nach Büchern einer bestimmten Autorin bzw. Autors zu filtern.

Darüber hinaus wollen wir einen *dark mode* implementieren, der auf Klick eines Buttons ein- und ausgeschaltet werden kann (um das Bearbeiten der Hausaufgabe nachts etwas angenehmer zu gestalten).

4.1 Aufgabe 1: JSON-Datei einlesen

Betrachten Sie die Hauptkomponente `App.vue`. In dieser ist ein sogenanntes *computed property* `booksList` definiert. Dieses *computed property* soll alle Einträge aus dem Datensatz enthalten. Sortieren Sie alle Bücher alphabetisch nach dem Buchtitel (**Hinweis:** Nutzen Sie hierfür bitte `String.localeCompare()`[3]). **Beachten Sie:** Manche Bücher wurden von mehreren Autorinnen bzw. Autoren geschrieben. In der JSON-Datei wurden diese mit einem Semikolon `';'` getrennt. Aus ästhetischen Gründen wollen wir aber das Komma gefolgt von einem Leerzeichen als `', '` als Trennzeichen verwenden.

4.2 Aufgabe 2: BooksList.vue

Implementieren Sie nun die Komponente `BooksList.vue`. In `App.vue` wurde eine Variable `windowSize` mit dem Wert `5` definiert. Diese gibt an wie viele Elemente in `BooksList.vue` angezeigt werden sollen. Reichen Sie die anzuzeigenden Elemente aus dem Datensatz an `BooksList.vue` weiter.

Damit die automatisierten Tests durchlaufen, müssen Sie ein Container-Element mit der ID `#books` kennzeichnen. Verwenden Sie die Klassen `.book-title` und `.book-authors` in Ihrer Auflistung. Implementieren Sie eine Möglichkeit Einträge aus der Liste auszuwählen. Hierzu müssen Sie eine Methode `selectItem()` implementieren, die einen Index übergeben bekommt und diesen an `App.vue` kommuniziert. In `App.vue` wird dieser Index in der Variable `selectedIndex` gespeichert.

Hinweis: Sie sind vollkommen frei in der visuellen Gestaltung dieser Komponente. Zum Bestehen der Tests müssen Sie sich lediglich an die oben beschriebene Struktur halten und die Auswahl eines Listeneintrags ermöglichen. Der Index des ausgewählten Elements muss der Komponente `App.vue` kommuniziert werden und diese muss den Wert in `selectedIndex` speichern.

4.3 Aufgabe 3: BooksListPagination.vue

Betrachten Sie nun die Komponente `BooksListPagination.vue`. In dieser sollen zwei Buttons implementiert werden: ein Zurück- und ein Weiter-Button. Der Zurück-Button ist auf der ersten Seite nicht anklickbar. Der Weiter-Button ist entsprechend auf der letzten Seite nicht anklickbar. Bei einem Klick auf dem Zurück- bzw. Weiter-Button wird die Variable `page` in `App.vue` um 1 entsprechend dekrementiert bzw. inkrementiert. Die angezeigten Elemente in `BooksList.vue` müssen sich entsprechend ändern.

Hinweis: Zum Bestehen der Tests darf Ihre *pagination* nur von `page` abhängen. Für `page = 0` werden die Elemente mit dem Index 0-4 aus `booksList` angezeigt. Für `page = 1` werden die Elemente mit dem Index 5-9 aus `booksList` angezeigt. Die letzte Seite ist `1361`.

Damit ein Button nicht anklickbar ist fügen Sie dem `class`-Attribut des jeweiligen Buttons die Klasse `disabled` hinzu.

[3] siehe: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/localeCompare`

4.4 Aufgabe 4: BookView.vue

Implementieren Sie nun die Komponente `BookView.vue` in der die Details zu einem ausgewählten Buch visualisiert werden sollen. Zeigen Sie das Buch-Cover mittels eines `<img>`-Tag, den Sie mit der ID `#book-image` kennzeichnen müssen.

Heben Sie den Titel des Buches hervor. **Achtung:** Manche Bücher haben Subtitel. In diesem Fall soll dieser dem Titel angehangen werden. Zum Bestehen der Tests muss dies im folgenden Format passieren: `book.title + ' - ' + book.subtitle`. Existiert jedoch kein Subtitel, soll auch kein Bindestrich erscheinen. Der Titel und ggf. der Subtitel müssen in einem Element enthalten sein, das mit der ID `#book-title` gekennzeichnet ist.

Die restlichen Informationen sollen aufgelistet werden. Jedes Key-Value-Pair wird in einem Element mit der Klasse `.book-property` versehen. **Achtung:** Die Attribute `thumbnail`, `title` und `subtitle` dürfen sich nicht wiederholen. Sortieren Sie die keys alphabetisch, sodass `authors` zuerst und `ratings_count` zuletzt angezeigt wird.

Hinweis: Das angezeigte Buch darf nur von `selectedIndex` abhängen. Konkret bedeutet das, dass ein Buch in `BookView.vue` angezeigt werden kann, das in `BooksList.vue` nicht angezeigt wird und somit nicht auswählbar ist.

4.5 Aufgabe 5: AuthorSearch.vue

Betrachten Sie nun die Komponente `AuthorSearch.vue`. Diese soll es uns ermöglichen nach Büchern einer Autorin bzw. eines Autors zu filtern.

Implementieren Sie zuerst das *computed property* `authorsList` in `App.vue`. Dieses Array soll alle Autorinnen und Autoren enthalten. **Beachten Sie:** Wir wollen auch nach Büchern von Zweitautorinnen und -autoren filtern können. Darüber hinaus soll `authorsList` keine Duplikate enthalten und alphabetisch sortiert sein. Übergeben Sie `authorsList` an `AuthorSearch.vue`.

Implementieren Sie in `AuthorSearch.vue` ein Suchfeld in das man den Namen einer Autorin bzw. eines Autors eingeben kann. Idealerweise erhält man eine Auswahl von Vorschlägen, während man etwas in das Suchfeld eingibt. Dazu können Sie das `<datalist>`-Element nutzen. Sie können die bereits vorgegebene Methode `testSearch()` verwenden, um die oder den gesuchten Autor an `App.vue` zu übermitteln.

Nutzen Sie die Variable `filterFn`, um `booksList` nach der gesuchten Autorin bzw. Autors zu filtern. **Tipp:** Überlegen Sie sich warum `filterFn` initial auf `() => true` gesetzt wurde. Es dürfen nur noch Bücher in `BooksList.vue` erscheinen an die die Autorin bzw. der Autor geschrieben bzw. mitgeschrieben hat. Natürlich soll sich die *pagination* entsprechend verhalten. **Achtung:** Der `selectedIndex` muss zurückgesetzt werden, da dieser sonst auf ein Element außerhalb `booksList` zeigt.

4.6 Aufgabe 6: NavBar.vue - Dark Mode

Zu guter Letzt: Implementieren Sie den *dark mode*. Betrachten Sie dazu die Komponente `NavBar.vue`. Erstellen Sie hier einen Button, der auf einen Klick die Methode `toggleDarkMode()` aufruft. Kennzeichnen Sie diesen Button mit der ID `#btn-dark-mode`. Der Button soll bei aktivem `dark mode` die Schrift `Lights on` enthalten.

Um den *dark mode* wirklich zu aktivieren müssen Sie das Stylesheet ändern. In `App.vue` finden Sie die Variablen `lightThemeUrl` und `darkThemeUrl`. Der `<link>`-Tag des Stylesheets hat die Kennzeichnung `#bootstrap-theme`. Ändern Sie das `href`-Attribut entsprechend. Natürlich soll man den `dark mode` auch wieder deaktivieren können.