Hausaufgabe 6: Server-side Programming

Laden Sie ihre Lösung für die Hausaufgabe in einer ZIP-Datei auf ISIS bis zum 20. Februar um 23:59 Uhr hoch. Die ZIP-Datei soll dabei ausschließlich den Ordner `ha06` 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

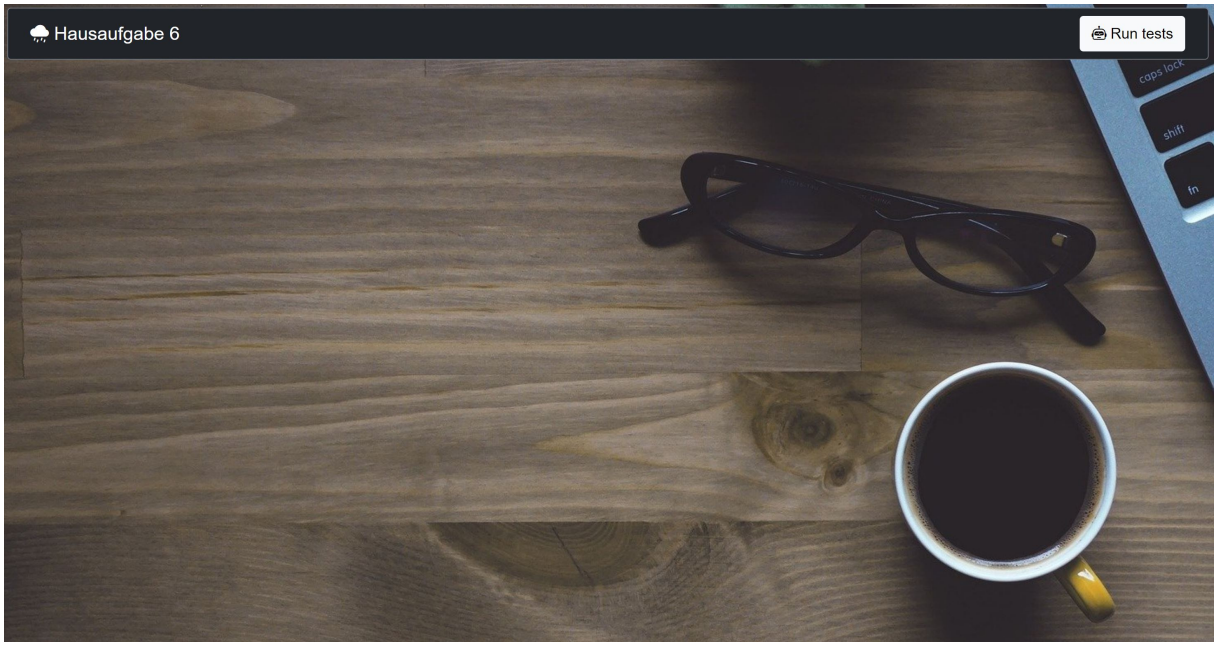

Abbildung 1: Screenshot der Vorgabe nach dem ersten Start. **Beachten Sie:** Die Vue-Vorgabe ist weitestgehend leer und wird ausschließlich zum Ausführen der Tests verwendet. Das Lösen dieser Hausaufgabe erfordert keine Implementierung in Vue.

Zur Bearbeitung müssen Sie Node.js installieren. Gehen Sie dazu auf `https://nodejs.org/`, laden Sie die entsprechende Version[1] für Ihr Betriebssystem her 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 `ha06/` aus der Vorgabe an einem beliebigen Ort auf Ihrem Rechner. Gehen Sie dann im Terminal in das Verzeichnis `ha06/` und führen Sie dort folgenden Befehl aus.

```
npm install
```

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

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 start
```

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

Grober Aufbau

In dieser Aufgabe wenden wir uns der Serverseite zu. Ihre Aufgabe ist es zwei Server zu implementieren. Dazu finden Sie in `ha06/server/` zwei JavaScript-Dateien: `server.js` und `express.js`. In `server.js` sollen Sie einen File-Server mittels des `http`-Moduls von Node.js implementieren. In `express.js` sollen Sie ein RESTful-Interface mittels der Erweiterung *Express* implementieren.

Die in der Vorgabe mitgelieferte Vue-Anwendung dient lediglich zum Ausführen der Tests. In dieser Aufgabe dürfen Sie ausschließlich in den Dateien `server.js` und `express.js` arbeiten. **Beachten Sie:** Verwenden Sie in dieser Aufgabe anstatt `npm run serve` den Befehl `npm run start`, um die Anwendung zu starten. Dieser Befehl startet die beiden Server und kompiliert die Vue-Anwendung.

6.1 Node HTTP-Server

Der zu implementierende File-Server soll über HTTP angesprochen werden können und es einem Client erlauben Dateien hochzuladen, zu aktualisieren oder zu löschen. Hierzu müssen die entsprechenden HTTP-Methoden `PUT`, `POST` und `DELETE` verwendet werden. Natürlich soll es dem Client möglich sein, das Dateisystem des Servers auszulesen. Hierzu muss die HTTP-Methode `GET` benutzt werden. Zusätzlich muss der Client spezifizieren, welchen Pfad dieser auslesen möchte oder unter welchem Pfad eine Datei angelegt, verändert oder gelöscht werden soll. Hierzu wird die URL der HTTP-Anfrage benutzt.

Folgend ein Beispiel: Die HTTP-Anfrage ...

`HTTP GET http://localhost:8000`**`/test/file1`**

... wird gemappt auf ...

`/path/to/ha06/server/files`**`/test/file1`**.

Beachten Sie: In `server.js` wird eine Konstante `root` definiert. Im Folgenden beschreiben wir das Verhalten des File-Servers auf die verschiedenen HTTP-Methoden.

Hinweis: In dieser Aufgabe müssen Sie mittels Node.js das Dateisystem auslesen und manipulieren. Dazu müssen Sie das Node-Modul `fs` verwenden. Schauen Sie sich die Dokumentation[3] des Moduls an und recherchieren Sie geeignete Methoden, die Sie zum Lösen dieser Aufgabe verwenden müssen. Darüber hinaus arbeiten wir zur Vereinfachung ausschließlich mit dem `Content-Type text/plain`.

[3] siehe: `https://nodejs.org/docs/latest-v16.x/api/fs.html`

HTTP GET

Erhält der Server eine HTTP-GET-Anfrage, sucht er unter dem übergebenen Pfad die angefragte Ressource. Hierzu müssen Sie unterscheiden, ob der Client einen Ordner oder eine Datei anfragt. Endet die URL mit einem `/`, so ist nach dem Inhalt eines Ordners gefragt. Sonst kann davon ausgegangen werden, dass eine Datei angefragt wurde. Kann unter dem angegebenen Pfad weder ein Ordner, noch eine Datei gefunden werden, gibt der Server eine HTTP-Antwort mit dem Status-Code `404` zurück.

Falls der Inhalt eines Ordners angefragt wurde, gibt der Server zusätzlich zu den Ordner- und Dateinamen zu jeder Ressource einen Typ (`directory` oder `file`) an. Ist nach einer Datei gefragt, wird der Inhalt der Datei zurückgegeben. Folgend zwei Beispiele zur Verdeutlichung:

`HTTP GET http://localhost:8000`**`/test/dir3/`**

Antwort:

```
[
  { "path": "dir1", "type": "directory" },
  { "path": "file1", "type": "file" },
  { "path": "file2", "type": "file" },
  { "path": "file3", "type": "file" },
  { "path": "file4", "type": "file" }
]
```

`HTTP GET http://localhost:8000`**`/test/file1`**

Antwort:
`/test/file1`

HTTP POST

Erhält der Server eine HTTP-POST-Anfrage, wird dieser eine Ressource unter dem angegebenen Pfad erstellen. Auch hier müssen Sie wieder zwischen Ordner und Dateien unterscheiden. Es soll möglich sein eine Ressource unter einem Ordnerpfad zu erstellen, der noch nicht existiert. Der Server soll in diesem Falle alle fehlenden Ordner erstellen. Soll eine Datei erstellt werden, wird der Inhalt der Datei im HTTP-Body übergeben. Existiert eine Datei bereits, so soll diese nicht überschrieben werden, sondern der Server soll eine HTTP-Antwort mit dem Status-Code `400` zurückgeben. Folgend zwei Beispiele zur Verdeutlichung:

`HTTP POST http://localhost:8000`**`/test/dir4/`**

Erstellt den Ordner `dir4` in `/path/to/ha06/server/files`**`/test/`**.

`HTTP POST http://localhost:8000`**`/test/dir5/dir6/file1`**

Erstellt die Datei `file1` in `/path/to/ha06/server/files`**`/test/dir5/dir6`**.

`HTTP PUT`

Erhält der Server eine HTTP-PUT-Anfrage, so soll der Inhalt einer **Datei** durch den Inhalt des HTTP-Bodys ersetzt werden. Ordnerinhalte können mit dieser HTTP-Methode nicht aktualisiert werden. Existiert eine Datei unter dem angegebenen Pfad **nicht** soll eine HTTP-Antwort mit dem Status-Code `404` zurückgeben.

`HTTP DELETE`

Erhält der Server eine HTTP-DELETE-Anfrage, soll eine Ressource unter dem übergebenen Pfad gelöscht werden. Es soll möglich sein Ordner oder Dateien zu löschen. Wird ein Ordner angefragt, so wird der Ordner und alle Unterordner und Dateien, die sich in diesem befinden, ebenfalls gelöscht. Wird eine Datei angefragt, so wird diese gelöscht. Existiert eine Datei unter dem angegebenen Pfad **nicht** soll eine HTTP-Antwort mit dem Status-Code `404` zurückgeben.

6.2 Express.js

In `express.js` sollen Sie ein RESTful (eher REST-like) Interface mittels Express implementieren. In dieser Aufgabe verwenden wir die JSON-Datei `books.json`. Dieser Datensatz enthält Daten zu knapp 7000 Büchern. Betrachten Sie den unten aufgeführten Eintrag als Beispiel. Der Datensatz soll mittels HTTP ausgelesen und manipuliert werden können. Da es sich um JSON-Objekte handelt, müssen alle Anfragen den `Content-Type: application/json` Header setzen.

```
{
  "isbn13": 9780002005883,
  "isbn10": "0002005883",
  "title": "Gilead",
  "subtitle": "",
  "authors": "Marilynne Robinson",
  "categories": "Fiction",
  "thumbnail": "%THUMBNAIL_URL%",
  "description": "A NOVEL THAT READERS ...",
  "published_year": 2004,
  "average_rating": 3.85,
  "num_pages": 247,
  "ratings_count": 361
}
```

Liste von Bucheinträgn auslesen

Erhält der Server eine HTTP-GET-Anfrage an die URL `/book`**`s`**, gibt dieser eine Liste von Bucheinträgn zurück. Da die gesamte Liste sehr lang ist, sollte idealerweise nur eine Teilmenge

der gesamten Liste zurückgegeben werden. Daher soll es dem Client möglich sein Start- und Endindex zu spezifizieren. Hierzu kann der Client die Query-Parameter `from` und `to` setzen. Beide Parameter sind optional, wobei `from` standardmäßig auf `0` und `to` auf `from + 5` gesetzt ist. **Hinweis:** Überlegen Sie sich, welche Rückgaben zu erwarten sind, wenn nur `from` oder nur `to` gesetzt ist.

Bestimmten Bucheintrag auslesen

Erhält der Server eine HTTP-GET-Anfrage an die URL `/book/:id`, gibt dieser einen konkreten Bucheintrag zurück. Die in der URL übergebene ID soll als ISBN13 interpretiert werden. Kann kein Bucheintrag mit der entsprechenden ISBN13 gefunden werden, soll der Server eine HTTP-Antwort mit dem Status-Code `404` zurückgeben. **Hinweis:** Es können durchaus mehrere Bucheinträge mit derselben ISBN13 existieren. Zur Vereinfachung ignorieren wir diese Fälle. Die automatisierten Tests nutzen nur Bucheinträge für die keine Duplikate existieren.

Bucheintrag hinzufügen

Bei einer HTTP-POST-Anfrage an die URL `/book` wird ein im HTTP-Body übergebener Bucheintrag dem Datensatz hinzugefügt. Existiert bereits ein Bucheintrag mit derselben ISBN13, gibt der Server eine HTTP-Antwort mit dem Status-Code `400` zurück.

Hinweis: Änderungen an dem Datensatz sind nicht persistent. Die JSON-Datei ist von diesen Änderungen nicht betroffen. Wenn Sie den Server neu starten, wird der ursprüngliche Zustand wieder hergestellt.

Bucheintrag bearbeiten

Bei einer HTTP-PUT-Anfrage an die URL `/book/:id` wird der Bucheintrag mit der als ID in der URL übergebenen ISBN13 durch den im HTTP-Body übergebenen Bucheintrag ersetzt. Existiert **kein** Bucheintrag mit derselben ISBN13, gibt der Server eine HTTP-Antwort mit dem Status-Code `404` zurück.

Hinweis: Änderungen an dem Datensatz sind nicht persistent (siehe Hinweis oben).

Bucheintrag löschen

Bei einer HTTP-DELETE-Anfrage an die URL `/book/:id` wird der Bucheintrag mit der als ID in der URL übergebenen ISBN13 gelöscht. Existiert **kein** Bucheintrag mit derselben ISBN13, gibt der Server eine HTTP-Antwort mit dem Status-Code `404` zurück.

Hinweis: Änderungen an dem Datensatz sind nicht persistent (siehe Hinweis oben).

*** BONUS *** Eigenschaften von Bucheinträgen lesen und verändern

Erstellen Sie für jede Eigenschaft eines Bucheintrags einen Endpunkt, um den entsprechenden Wert der Eigenschaft auszulesen (im Format `{ property: value }`) und einen weiteren Endpunkt, um den entsprechenden Wert der Eigenschaft zu ändern. Betrachten Sie die folgenden Beispiele zur Verdeutlichung:

`HTTP GET http://localhost:8001`**`/book/9780002005883/title`**

Antwort:

```
{ "title": "Gilead" }
```

`HTTP GET http://localhost:8001`**`/book/9780002005883/description`**

Antwort:

```
{ "description": "A NOVEL THAT READERS ..." }
```

`HTTP GET http://localhost:8001`**`/book/9780002005883/ratings_count`**

Antwort:

```
{ "ratings_count": 361 }
```

Mittels einer HTTP-PUT-Anfrage auf die jeweilige URL, kann der Wert verändert werden. Dazu muss im HTTP-Body der Anfrage das Wertepaar im folgenden Format angegeben werden `{ property: value }`. Betrachten Sie zur Verdeutlichung das folgende Beispiel:

`HTTP PUT http://localhost:8001`**`/book/9780002005883/title`**

HTTP-Body:

```
{ "title": "Gilead - UPDATED" }
```

`HTTP GET http://localhost:8001`**`/book/9780002005883/title`**

Antwort:

```
{ "title": "Gilead - UPDATED" }
```

Hinweis: Sie müssen die Endpunkte für alle Eigenschaften eines Bucheintrags implementieren. Sie können davon ausgehen, dass die Liste der Eigenschaften in dem gezeigten Beispiel-Bucheintrags vollständig ist und dass es keine weiteren Eigenschaften gibt.