Übung 08

1 AJAX und REST

Erstellen Sie eine Webseite mit der eine Filmliste in Form einer Tabelle ausgeben werden kann. Die Daten der Filmliste sollen Serverseitig verwaltet werden und nur über GET-Requests dem Client zur Verfügung gestellt bzw. zugesendet werden. Hierfür werden folgende Hilfsmittel zur Verfügung gestellt:

- **filmArchiv.html:** Nutzen Sie dieses Html-Dokument für die Umsetzung aller Aufgaben dieser Übung.
- **Express-Server:** Express-Server-Skelett, wird für die serverseitige Verwaltung der $spielfilme.json$ genutzt. Setzen Sie hier die serverseitige Verarbeitung einer GET-Requests um.

Setzten Sie als erstes die Funktion für das $input$-Element $ShowAllFilms$ um. Beim klicken des $ShowAllFilms$-Buttons soll eine HTML-Tabelle erstellt werden, die alle in der $spielfilme.json$ vorhandenen Filme mit Titel, Erscheinungsdatum und Regie auflistet. Erstellen Sie weiterhin eine serverseitige GET-Funktion, die bei einem clientseitigen GET-Request die Information der $spielfilme.json$ an den Client weiterreicht. Setzen Sie die Server-Client-Kommunikation REST-konform um.

Lösungsweg

Beginnen wir mit der Umsetzung der serverseitigen GET-Methode. Dafür öffnen wir die `app.js`-Datei, die im Express-Server hinterlegt ist und erstellen eine Funktion, die bei einem GET-Request aufgerufen wird:

```
app.get('/myMovies', (req, res) => {
});
```

Nun haben wir eine leere Funktion erstellt, die aufgerufen wird, sobald der Server einen GET-Request auf das Verzeichnis `/myMovies` erhält. Weiterhin übernimmt unsere Funktion zwei Parameter, `req` und `res`, welche für request und response stehen. Füllen wir nun unsere Funktion mit Inhalt:

```
app.get('/myMovies', (req, res) => {
      var filmList = fs.readFileSync('./myMovies/spielfilme.json', 'utf8');
  res.status(200);
  res.json(filmList);
});
```

Da unsere GET-Funktion die Informationen der *spielfilme.json* an den Client weitergeben soll, muss die Funktion die Datei `spielfilme.json` auslesen. Dies erreichen wir mithilfe der Funktion `readFileSync()` aus dem Package `fs` (File System). Hierbei übergeben wir der Funktion den Pfad an dem sich unsere `spielfilme.json` befindet und dessen Kodierung. Den Inhalt speichern wir dann in der Variabel `filmList`. Anschließend setzten wir den Status des Request auf `OK` (Code 200) und schicken die Variabel `filmList`, welche alle Information der `spielfilme.json` enthält, an den Client als Antwort zurück.

Schauen wir uns nun die Clientseite an. Hier sollen über einen Knopfdruck alle Filminformationen innerhalb einer Tabelle ausgegeben werden. Der dazugehörige Knopf ist bereits in dem beigelegten HTML-Dokument hinterlegt. Wir müssen diesen also nur in unserer externen Javascript Datei referenzieren und ihm eine Funktion zuweisen, die ausgeführt wird, sobald der Knopf betätigt wird:

```
const showAllFilmbttn = document.querySelector('#allFilms');
  showAllFilmbttn.addEventListener('click', function(){
});
```

Kümmern wir uns nun zunächst erstmal um die benötigten Informationen, mit denen wir die Tabelle füllen sollen. Diese befinden auf dem Server in der `spielfilme.json`-Datei. Durch die Implementierung der oben erstellten GET-Funktion, wissen wir, das der Server auf einen GET-Request an der Stelle `/myMovies` mit dem Inhalt der `spielfilme.json`-Datei antwortet. Erstellen wir also unseren ersten GET-Request mithilfe der `fetch()`-Funktion:

```
showAllFilmbttn.addEventListener('click', function(){
  fetch("http://localhost:8080/myMovies",{
    method:"get",
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    }
  }).then(function(response){
    response.json().then(function(data){
      let filmListe = JSON.parse(data);
      showFilmTable(filmListe);
    });
  });
});
```

Im ersten Parameter der `fetch()`-Funktion übergeben wir den Server den wir ansprechen wollen. Anschließend übergeben wir die Request-Methode mit der wir den Server ansprechen, in unserm Fall ist dies `get`. Zusätzlich übergeben wir ein paar header-Informationen, hier deklarieren wir den erwarteten Datentyp als JSON.

Anschließend warten wir mit der `then()`-Funktion auf die Antwort des Servers - unsere GET-Request. Hierbei führt die `then()`-Funktion eine definierte anonyme Funktion aus, die den Response des Servers als Übergabeparameter enthält. Bei einem Response des Servers handelt es sich zunächst immer um ein Promise. Somit müssen wir auch hier ein weiteres mal warten, bis das Promise `fullfilled` ist. Auch hierfür benutzten wir eine weitere `then()`-Funktion mit der wir auf die gesendeten Daten (`data`) des Servers zugreifen und in einer Variable speichern können.

Ebenfalls überführen wir die erhaltenen Daten noch in ein lokales JSON-Objekt, sodass wir innerhalb der Bearbeitung leichter darauf zugreifen können. Danach übergeben wir das JSON-Objekt an eine Funktion `showFilmTable()`, die unsere Tabelle auf unserer Webseite anzeigen soll. Wir implementieren die Funktion `showFilmTable()` wie folgt:

```
const showFilmTable = (filmListe) =>{
  let table = "<tr><th>Titel</th><th>Erscheinungsdatum</th><th>Regie</th></tr>";

  for (i = 0; i < filmListe.filme.length; i++) {
        table += "<tr><td>" +
        filmListe.filme[i].titel +
        "</td><td>" +
        filmListe.filme[i].datum +
        "</td><td>" +
        filmListe.filme[i].regie +
        "</td></tr>";
  }
  document.getElementById("filmTable").innerHTML = table;
}
```

Innerhalb der Funktion `showFilmTable()` deklarieren wir zuerst die Kopfzeile der Tabelle. Anschließend iterieren wir über alle Filme innerhalb der übergebenen Filmliste die wir vom Server erhalten haben und füllen die Tabelle mit dessen Informationen. Zuletzt hängen wir den erstellten Inhalt an das bereits vorhandene `Table`-Element des HTML-Dokumentes an.

2 Aufgaben

Bearbeiten Sie nun die folgenden Aufgaben.

Aufgabe 1

Setzen Sie nun die `input`-Elemente `Next` und `Previous` innerhalb einer separaten Javascript Datei `filmArchiv.js` um. Hierbei sollen beim Aufrufen der Webseite die Information des ersten Films der `spielfilme.json` in dem `div`-Element *Film* angezeigt werden. Mit einem klicken der Elemente `Next` und `Previous` soll der jeweils nächste Film bzw. vorherige Film aus `spielfilme.json` in dem `div`-Element `Film` anzeigen. Nutzen Sie hierfür den bereits umgesetzten GET-Request.

Aufgabe 2

Implementieren Sie serverseitig eine PUT-Funktion die neue Filme erhält und diese in der `spielfilme.json` hinzufügt. Setzen Sie den Status des Requests anschließend auf `Created`.

Implementieren Sie analog eine serverseitige DELETE-Funktion der den Titel eines Filmes erhält und diesen, sofern vorhanden, komplett aus der *spielfilme.json* entfernt (also nicht nur den Titel, sondern alle Informationen des Films). Sollte ein Film auf diese Art entfernt werden, soll der Server seinen Status auf OK setzten und die Meldung `Film deleted` versenden. Befindet sich kein Film mit dem erhaltenen Titel in `spielfilme.json`, soll der Status auf `Bad Request` gesetzt werden und der Grund als Meldung gesendet werden.

Implementieren Sie nun Funktionen für das Hinzufügen und Entfernen von Filmen aus der `spielfilme.json`. Die Funktionen sollen jeweils einen PUT-Request bzw. DELETE-Request mit den nötigen Daten an den Server schicken. Nutzen Sie hierfür die zur Verfügung gestellten Elemente des beigelegten HTML-Dokumentes.