Web-Entwicklung: Übungsblatt 4

In den bisherigen Übungen haben Sie mit Mock-Datensätzen gearbeitet. In dieser Übung sollen Sie zum ersten Mal eine echte entfernte API, nämlich PokéAPI - The RESTful Pokémon API, ansprechen.

Die asynchrone Kommunikation mit der API setzt einen HTTPS-Client voraus. Das Node.js-Standardmodul `https` bietet entsprechende Funktionalitäten an. Dabei kommt das Callback-Entwurfsmuster zum Einsatz.

Aufgabe 1: Dokumentationen

Setzen Sie sich zunächst mit der Dokumentation des Node.js-Standardmoduls https sowie der Dokumentation der PokéAPI auseinander. Sehr hilfreich ist sicherlich auch die Anleitung "Anatomy of an HTTP Transaction".

Hinweis: Üblicherweise finden Sie für andere npm-Module oder APIs ähnliche Dokumentation. Die Qualität dieser Dokumentationen ist sehr unterschiedlich und zumindest ein Indikator für die Reife eines Moduls. Achten Sie also bei der Auswahl von Modulen für Ihre eigenen Projekte auch darauf.

Aufgabe 2: Asynchrone Kommunikation via Callbacks

Erstellen Sie eine `GET`-Anfrage für eine PokéAPI-Ressource Ihrer Wahl und inspizieren Sie das von der API zurückgelieferte Objekt. In einem Callback soll der *body* der HTTP-Antwort ausgegeben werden.

Testen Sie auch, ob Ihr Callback sinnvoll auf Fehler reagiert. Dazu können Sie den Dienst httpstat.us einsetzen, der ebenfalls über eine HTTPS-API HTTP-Antworten mit dem gewünschten HTTP-Fehlercode ausliefert. Eine weitere Fehlerart, die Sie behandeln sollten, sind Anfragen an gar nicht existierende URLs.

```
https.get('https://pokeapi.co/api/v2/pokemon/arcanine', onResponse).on('error',
onError);
// Error: getaddrinfo ENOTFOUND www.doesnotexist.de
https.get('https://httpstat.us/404', onResponse).on('error', onError);
// Error: Server responded with HTTP status code 404
https.get('https://www.doesnotexist.de', onResponse).on('error', onError);
// { ... }
```

Aufgabe 3: Asynchrone Kommunikation via Promises

Wie Sie in der Vorlesung gesehen haben, ist `Promise` ein ES6-Standardobjekt zur Vereinfachung asynchroner Programmabläufe.

Wandeln Sie Ihren Lösung von Aufgabe 2 so um, dass die asynchrone Funktionalität Ihrer Callbacks in einem Promise-Objekt gekapselt wird. Rufen Sie dann für jede zu testende URL eine selbstgeschriebene Funktion `get` auf, die ein solches Promise-Objekt zurückliefert.

```
get('https://pokeapi.co/api/v2/pokemon/arcanine').then(onResponse).catch(onError);
get('https://httpstat.us/404').then(onResponse).catch(onError);
get('https://www.doesnotexist.de').then(onResponse).catch(onError);
```

Aufgabe 4: Asynchrone Kommunikation via Promises und async/await

Wie Sie in der Vorlesung gesehen haben, sind `async` und `await` neue Schlüsselwörter, die in Kombination mit Promises eingesetzt werden können, und asynchrone Programmabläufe syntaktisch weiter vereinfachen. Insbesondere ist damit der Fehlerabfang mithilfe von try-catch-Blöcken möglich.

Schreiben Sie eine asynchrone Funktion `request`, welche unter Nutzung der neuen Schlüsselwörter die Funktion `get` aus Aufgabe 3 aufruft und sich im Erfolgs- und Fehlerfall genauso verhält wie die Lösung von Aufgabe 2 und 3.

```
request('https://pokeapi.co/api/v2/pokemon/arcanine');
request('https://httpstat.us/404');
request('https://www.doesnotexist.de');
```