Webbasierte Anwendungen SoSe 2018
Übungsaufgaben - Asynchrone Funktionen und Promises

Die folgenden Aufgaben sollen Ihr Grundverständnis in Javascript und Node.js weiter festigen. Im Kern geht es hier um asynchrone Programmierung. Bitte bereiten Sie die Aufgaben für den **23.04.2018** vor.

Grundlegende Informationen

In Javascript können Funktionen als Parameter einer anderen Funktion übergeben werden. Diese Übergabefunktionen nennt man Callbacks. Ein Beispiel dafür sind die Funktionen *readFile* und *writeFile* aus den vorhergehenden Aufgaben.

```
fs.readFile("/etc/passwd", function(err, data) {
        if(err) throw err;
        console.log(data);
});
```

Die Verwendung von Callbacks wird jedoch problematisch, wenn mehrere asynchrone Funktionen sequentiell ausgeführt werden müssen. Hierzu müssen mehrere Callbacks ineinander geschachtelt werden, wodurch der Code unübersichtlich und schwieriger zu warten ist. Dieses Problem nennt man auch Callback Hell. Das folgende Snippet zeigt ein Beispiel, in dem mehrere Dateien eingelesen werden:

```
fs.readFile("/user/home/file1.json", function(err, data1) {
        fs.readFile("/user/home/file2.json", function(err, data2) {
                fs.readFile("/user/home/file3.json", function(err, data3) {
                        // nutze die Daten aus file1 und file2 mit file3
                });
        });
});
```

Ein anderes Problem ist, dass auftretende Fehler innerhalb von asynchrone Funktionen, nicht mit Hilfe von try catch Blöcken aufgefangen werden können. In Node.js wird dies meistens durch einen Error-Objekt gelöst, welcher als erster Parameter der Callback Funktion übergeben wird. Dieses Objekt wird gefüllt, sobald ein Fehler, in der gerade ausgeführten Operation, auftritt. Zur Fehlerbehandlung muss dieses Objekt in einer bedingten Anweisung abgefragt werden. Dadurch wird der Code noch komplexer und unübersichtlicher.

Um diese Probleme zu entschärfen, gibt es in Node.js die Möglichkeit, Promises zu verwenden. Bei einem Promise handelt es sich um ein Objekt, welches den aktuellen Zustand eines asynchronen Funktionsaufrufs repräsentiert.

Das folgende Beispiel zeigt ein Promise anhand einer HTTP Anfrage:

```
function getUser() {
	var options = { url: “https://example.com/user/1“; }
	return new Promise(function(resolve, reject) {
		request.get(options, function(err, resp, body) {
			if(err) reject(err);
			else resolve(JSON.parse(body));
		});
	});
};

var userdetails;
var userPromise = getUser();

userPromise.then(function(result) {
	userdetails = result;
}).then(function(result) {
	// …
}).catch(function(err) {
	console.log(err);
});
```

Im Vergleich zur Verwendung von Callbacks wird der Code durch Promises übersichtlicher und es lassen sich Fehler eleganter behandeln.

Aufgabe 1 - Callbacks

Schreiben Sie ein Programm in Node.js, welches die Dateien stadte.json und mehr_stadte.json einliest und anschließend beide Datensätze zusammenführt. Nutzen Sie hier auch die **aynchronen Funktionen** aus den Aufgaben des vorherigen Aufgabenblatts. Das Programm soll am Ende ein Array mit allen Städten der beiden Dateien enthalten und das selbe Format zur Ausgabe nutzen.

Tip: Javascript bietet bereits Funktionen zum Zusammenführen von Arrays an.

Aufgabe 2 - Promises

Schreiben Sie ein Programm in Node.js. welches Promises verwendet um die Ziele aus Aufgabe 1 zu erreichen. Weitere Informationen zu Promises und Beispiele finden Sie hier oder hier.