FH MÜNSTER
University of Applied Sciences

MSB FB Wirtschaft
Münster School of Business

Web-Entwicklung II

Wintersemester 2017/18

Übungen zum Kapitel 2

Prof. Dr. Norman Lahme-Hütig
Informatik/Wirtschaftsinformatik, Schwerpunkt Web Engineering

Corrensstraße 25
D-48149 Münster

fon +49 (0)251.83 65-190
fax +49 (0)251.83 65-525

norman.lahme-huetig@fh-muenster.de
www.fh-muenster.de

Übung 1
Aufgaben: Module

1. Erstellen Sie ein Verzeichnis `myapp` mit folgenden Dateien

Modul	Erläuterung
`person.js`	• Gibt auf der Konsole die Meldung „executing person.js“ aus • Exportiert ein Objektliteral, das eine Person mit Namen und Alter beschreibt (via `exports.person = ...`)
`foo.js`	• Importiert das Objekt des Moduls `person` • Deklariert eine Funktion `change`, die den Namen der importieren Person ändert, und exportiert diese (via `module.exports = ...`)
`bar.js`	• Importiert das Objekt des Moduls `person` • Deklariert eine Funktion `change`, die das Alter der importieren Person ändert, und exportiert diese (via `module.exports = ...`)
`app.js`	• Importiert das Objekt des Moduls `person` • Ruft die vom Modul `foo` exportierte Funktion auf • Ruft die vom Modul `bar` exportierte Funktion auf • Gibt Name und Alter des importieren Person-Objekts aus

Übung 1
Aufgaben: Module

2. Führen Sie Ihre Anwendung mit folgendem Befehl aus

```
$ node app.js
```

3. Zeigen Sie grafisch die Abhängigkeiten zwischen den Modulen auf
4. Erweitern Sie das Modul `foo` um die Deklaration einer beliebigen weiteren Funktion, ohne Sie zu exportieren. Kann diese Funktion innerhalb von `app.js` aufgerufen werden?
5. Beantworten Sie folgende Fragen:
   - Wie häufig wird jedes Modul ausgeführt?
   - Was fällt Ihnen in Bezug auf das Person-Objekt auf, das zwischen den Modulen ausgetauscht wird?
   - Welche Vorteil sehen Sie bei der Verwendung von Modulen?

Übung 2

Aufgaben: Globale Objekt

1. Demonstrieren Sie mit der REPL, dass auf ein Standardobjekt (z. B. `Math` oder `console`) sowohl direkt als auch über das globale Objekt zugegriffen werden kann
2. Zeigen Sie, dass beide Zugriffsmöglichkeiten dasselbe Objekt liefern
3. Demonstrieren Sie mit der REPL den Zusammenhang zwischen globalen Variablen und dem globalen Objekt
4. Erstellen Sie eine Datei `app.js` mit folgendem Inhalt:

```
var foo = 'bar';
console.log(global.foo);
```

   Führen Sie die Anwendung mit `node app` aus und erläutern Sie die Ausgabe

Übung 3

Aufgaben: Standardobjekte

1. Erstellen Sie ein Verzeichnis `standardobjekte` mit der Datei `app.js`, die das Arbeitsverzeichnis ermittelt und auf der Konsole ausgibt
2. Rufen Sie Ihre Anwendung einmal aus dem Verzeichnis `standardobjekte` und einmal aus dessen Oberverzeichnis heraus auf

```
$ node app.js
```

```
$ node standardobjekte/app.js
```

   Was fällt Ihnen auf?

3. Lassen Sie nun das Modulverzeichnis (`__dirname`) ausgeben. Führen Sie erneut die Anwendung aus beiden Verzeichnissen heraus aus. Was fällt Ihnen auf?

Übung 4

Aufgaben: Kernmodule

1. Erstellen Sie eine Webanwendung **Aufgabenverwaltung**, bestehend aus einem Verzeichnis **`plain-tm`** mit folgenden Dateien

Datei	Erläuterung
`tasks.json`	• Enthält ein JSON-Array von Aufgaben, z. B. `[{"title": "Einkaufen"}, {"title": "Putzen"}, {"title": "Lernen"}]`
`src/app.js`	• Liest die Datei `tasks.json` ein, wandelt den eingelesenen String in ein JavaScript-Array um und speichert dieses in der Konstante `tasks` • Startet einen Web-Server, der auf Port 3000 horcht • Jede eingehende Anfrage wird mit einem dynamisch erstellten HTML-Dokument beantwortet, dass die zuvor eingelesenen Aufgaben enthält

```
plain-tm
  src
    app.js
  tasks.json
```

Aufgaben

- Einkaufen
- Putzen
- Lernen

2. Starten Sie den Server mit **`node app`** und testen Sie Ihre Anwendung im Browser über die URL **`http://localhost:3000`**

Übung 5

Aufgaben: Express – Routing

1. Erstellen Sie eine erste Express-Variante der Webanwendung Aufgabenverwaltung im Verzeichnis **express-tm**
   - Erzeugen Sie eine initiale `package.json`-Datei im Verzeichnis `express-tm`
   - Installieren Sie das Node.js-Paket `express`
   - Übernehmen Sie die Dateien `tasks.json` und `app.js` aus der Übung 4
   - Ändern Sie `app.js`, sodass für den Web-Server nun Express verwendet wird
     - Statische Dateien sollen aus dem Verzeichnis public ausgeliefert werden
   - Erstellen Sie eine geeignete `style.css`-Datei und binden Sie diese in das generierte HTML-Dokument ein
2. Setzen Sie einen Breakpoint an der Stelle, an der eine HTTP-Anfrage beantwortet wird, und debuggen Sie Ihre Anwendung. Folgen Sie hierzu den Hinweisen Ihres Dozenten.

```
express-tm
  node_modules
  src
    public
      css
        style.css
    app.js
  package.json
  tasks.json
```

Übung 5

Aufgaben: Express – Routing

3. Erweitern Sie die Anwendung express-tm wie folgt:
   - Erstellen Sie ein Unterverzeichnis `routes` mit einer Datei `tasks.js`
   - In der Datei `tasks.js` soll ein Router-Objekt exportiert werden, das auf eine Anfrage gegen die relative URL `'/'` mit dem dynamisch generierten HTML-Dokument antwortet. Dazu soll das Modul initial die `tasks.json`-Datei einlesen.
   - In der Datei `app.js` soll das Router-Objekt importiert und unter `'/tasks'` eingehängt werden. Ferner sollen Anfragen gegen `http://localhost:3000` nach `http://localhost:3000/tasks` umgeleitet werden.

```
express-tm
  node_modules
  src
    public
    routes
      tasks.js
    app.js
  package.json
  tasks.json
```

Übung 6

Aufgaben: Express – Templates

1. Erweitern Sie die Anwendung `express-tm`, sodass nun die HTML-Generierung mit der Template Engine Handlebars erfolgt
   - Erstellen Sie ein Unterverzeichnis `views` mit einer Datei `tasks.hbs`
   - In der Datei `tasks.hbs` soll das Template für die HTML-Datei enthalten sein
   - Ändern Sie die Datei `tasks.js`, sodass nun die Template Engine zum Rendern der HTML-Seite verwendet wird
   - Erweitern Sie die Datei `app.js` um den Code zur Konfiguration der Template Engine
2. Fügen Sie Ihrer Anwendung ein Layout `main.hbs` sowie ein Partial `task.hbs` hinzu

```
express-tm
  node_modules
  src
    public
    routes
      tasks.js
    views
      layouts
        main.hbs
      partials
        task.hbs
      tasks.hbs
    app.js
  package.json
  tasks.json
```

Übung 7
Aufgaben: MongoDB

1. Fügen Sie Ihrem Projekt eine MongoDB-Datenbank mit Testdaten hinzu
   - Erstellen Sie ein Verzeichnis `db` mit dem Unterverzeichnis `data` und einer Datei `tasks.json`, letztere mit folgendem Inhalt

```
{"id": "5c24e89a-e5b1-4ad2-a89b-a1e58594e87d", "title": "Einkaufen"}
{"id": "5ec1befc-4f54-415f-bc04-1f31b68e702c", "title": "Putzen"}
{"id": "97be35ab-4a1a-4801-9802-8ce97af06022", "title": "Lernen"}
```

   - Starten Sie MongoDB*

```
$ mongod -dbpath db/data
```

   - Importieren Sie die Testdaten

```
$ cd db
$ mongoimport --db express-tm --collection tasks --file tasks.json
```

```
express-tm
  db
    data
    tasks.json
  node_modules
  src
  package.json
```

*Ggf. müssen Sie vorher den Dienst stoppen: `sudo service mongod stop`

Übung 7

Aufgaben: MongoDB

2. Erweitern Sie die Anwendung `express-tm`, sodass die Daten nun aus der zuvor angelegten MongoDB-Datenbank gelesen werden
   - Erweitern Sie die Datei `app.js`, sodass dort zunächst die Verbindung zur Datenbank `express-tm` hergestellt wird, bevor der Server hochfährt
   - Ändern Sie die Datei `routes/tasks.js`, sodass die Daten nun aus der Datenbank gelesen werden

```
express-tm
  db
  node_modules
  src
    public
    routes
      tasks.js
    views
    app.js
  package.json
```

Übung 7
Aufgaben: MongoDB

3. Erweitern Sie die Anwendung, sodass neue Aufgaben hinzugefügt werden können
   - Installieren Sie das Paket uuid

     ```
     $ npm i uuid
     ```

   - Erweitern Sie die Datei views/`tasks.hbs` um ein Formular zur Eingabe einer neuen Aufgabe
   - Erweitern Sie die Datei `routes/tasks.js`, sodass bei einem POST an `/tasks` in der DB eine neue Aufgabe gespeichert wird und ein Redirect zu `/tasks` erfolgt. Für die Generierung einer neuen Id gehen Sie wie folgt vor:

     ```
     const uuidv4 = require('uuid/v4');
     // ...
     task.id = uuidv4();
     ```

```
express-tm
  db
  node_modules
  src
    public
    routes
      tasks.js
    views
      layouts
      partials
      tasks.hbs
    app.js
  package.json
```

Übung 7
Aufgaben: MongoDB

4. Ändern Sie die Anwendung, sodass die Aufgaben nach dem Erstellungsdatum absteigend sortiert werden (Datei: tasks.js)
   - Hierfür ist beim Anlegen einer neuen Aufgabe das Erstellungsdatum unter der Eigenschaft `createdAt` wie folgt abzulegen

     ```
     task.createdAt = new Date().getTime();
     ```

   - Diese Eigenschaft ist als Sortierkriterium beim Ermitteln aller Aufgaben zu berücksichtigen

```
express-tm
  db
  node_modules
  src
    public
    routes
      tasks.js
    views
    app.js
  package.json
```