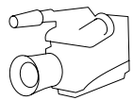
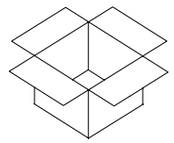
25.03.14

Exposé & Grobkonzept

WBA2 Workshop

Fachhochschule Köln
Cologne University of Applied Sciences



Viel drin
Medieninformatik an der Fachhochschule Köln

Exposé

- Was ist das Ziel des Projektes?
- An wen richtet sich das Projekt?
- Begründung der Kommunikationsparadigmen
- kurze Marktrecherche
- nicht länger als eine Seite

Anforderungen an Projektidee

- die im Workshop vorgestellten Technologien zur synchronen sowie asynchronen Kommunikation sollten zur Umsetzung der Idee notwendig sein
- angemessener Umfang für den Workload pro Gruppe
- Workshop Input soll von der Gruppe bei der Projekt-Implementation sinnvoll kombiniert und erweitert wurde

Grobkonzept

- Formales
  - (Deckblatt, Einleitung, Verzeichnisse etc.)
- Systemarchitektur
- Für beide Kommunikationsparadigmen Ressourcen bzw. Topics festlegen und begründen
- Datenstrukturen spezifizieren (JSON Schema etc.)
- Kommunikationsmodell

Hinweise

- alles begründen
- „Roter Faden“ sollte erkennbar sein
- kein totes Wissen
- wissenschaftliche Qualität bei Referenzierungen etc.
- kleiner Projektplan für das Team ist sinnvoll

Fragen?

25.03.14

npm, Express & AJAX Basics

WBA2 Workshop

Fachhochschule Köln
Cologne University of Applied Sciences

Viel drin
Medieninformatik an der Fachhochschule Köln

Module

interne Module:

- bereits aus Workshop 1 bekannt
- alle Module im Verzeichnis lib im Quellcode von Node.js
- werden im Code des Servers wie folgt eingebunden:

```
// HTTP-Modul einbinden
var http = require('http');
```

Module

externe Module

- nach Konvention im Ordner "*node_modules*"

```
//Einbinden eines externen Moduls liegt mit Namen "express" im node-modules Verzeichnis
var express = require('express');

// Einbinden einer externen Datei als Modul
var foo = require('./verzeichnis/foo.js');

// Einbinden eines externen Verzeichnisses als Modul
var foo = require('./verzeichnis/foo.js');
```

Javascript-Dateien können als eigene Module exportiert werden, indem *module.exports* verwendet wird.

module.exports = wordCount; stellt beispielsweise die Funktion *wordCount* zur Verfügung)

npm

- npm = Node Package Manager
- installieren von Modulen über den Modulnamen
  z.B. $ npm install express
- Module können auch global (Parameter *–g*) installiert werden
- „package.json" Datei zur einfachen Integration von Modulen in eigene Anwendungen und zur Verwaltung der Abhängigkeiten (dependencies)

npm

Eine Datei namens "package.json" sollte angelegt werden:

```
{
  "name": "FooJS",
  "version": "0.1.0",
  "description": "FooJS Projekt",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified! Configure in package.json\" && exit 1"
  },
  "author": "Max Mustermann",
  "readmeFilename": "README.md",
  "dependencies": {
      "express": "3.x"
  }
}
```

- Befehl *`$ npm install`* lädt und installiert dann automatisch die angegebenen Module (in "node_modules") (als Admin ggf. mit "sudo")

Express

Was ist Express?

- Framework für Webapplikationen
- Modul für Node.js
- Features:
  - Konfigurieren von Webanwendungen
  - einfach zu konfigurierendes Routing von Anfragen auf Handler
  - dynamisches Erzeugen von Webseiten
- Installation bspw. über npm

Express

Anlegen eines Webservers:

- Importieren von Express als externes Modul

```
var express = require('express');
```

- Anlegen eines Webservers der unter *localhost* auf port 3000 erreichbar ist:

```
var app = express();
// die Bindung an den Port sollte als letztes im Code stehen
app.listen(3000);
```

Express

Konfiguration des Webservers:

- u.a. über die Funktionen *app.use* und *app.set*

```
app.configure(function() {

  // Verzeichnis fuer den direkten Zugriff von Aussen freigeben
  app.use(express.static(__dirname + '/public'));

  // benötigt um Informationen des Requests zu parsen
  app.use(express.json());
  app.use(express.urlencoded());

  // möglicher Errorhandler
  app.use(function(err, req, res, next) {
    console.error(err.stack);
    res.end(err.status + ' ' + err.messages);
  });
});
```

Express

Routing:

- Funktionen *app.get, app.post, app.put* und *app.delete*
- erstes Argument ist der Pfad, danach der Callback

```
// Routing einer GET Methode auf die Ressource foo
app.get('/foo', function (req, res) {
  // hier steht was passieren soll
});

// Routing einer POST Methode auf die Ressource foo
app.post('/foo', function (req, res) {
  // hier steht was im Callback passieren soll, bspw. Veränderungen an Daten
});
```

Express

Umgang mit Request & Response:

- Request Handling erfolgt jeweils in der Callback Funktion der Route
- zwei Parameter die zur Verarbeitung genutzt werden, können an die Callback-Funktion übergeben werden:
  - *`req`* *(*Request)
  - *`res`* (Response)

Express

Request:

- Um das im Request übergebene Objekt (bspw. JSON) serverseitig zu parsen sind zwingend folgende Konfigurationen von Express notwendig:

```
app.use(express.json());
app.use(express.urlencoded());
```

- dadurch wird der body des Request Objektes automatisch geparst und es können anschließend weitere Aktionen u.a. über folgende Funktionen auf dem *`req`* ausgeführt werden:

```
// zugriff auf den body des requests
req.body;
```

Express

Response:

- Zugriff auf den Response Parameter geht u.a. über folgende Funktionen:

```
// schreibt Informationen in den Header der Response
res.writeHead(200, "OK");

// kann benutzt werden um Informationen an den Client zu senden
res.write("Informationen die der Client erhalten sollte");

// MUSS verwendet werden um Response absendbar zu machen
res.end();
```

- Für die Response können im Head auch weitere Parameter festgelegt werden z.B.:

```
'content-type':'text/html'
```

AJAX & jQuery

- um auf dem Client das verwenden der HTTP Methoden zu vereinfachen
- jQuery import:
  - <script src=„http://code.jquery.com/jquery-1.11.0.min.js“></script>
- POST asynchron via AJAX und jQuery absetzen:
  - auch nur über Formular möglich, aber besser mit AJAX
  - `event.preventDefault();` nicht vergessen
  - die Nutzdaten am Besten in einem Format übertragen mit dem der Server umgehen kann (bspw. ein `data` Objekt im JSON Format)

AJAX & jQuery

- Dazu clientseitig ein script in die HTML Seite einbinden:
  In die Funktion, welche beim "submit" des Formulares ausgeführt wird, kommt dann der AJAX POST, z.B. :

```
$.ajax({
  type: 'POST',
  url: '/foo',
  data: JSON.stringify(data),
  contentType: 'application/json'
  }).done(function(){
    alert(data.name+' wurde hinzugefuegt.');
  }).fail(function(e){
    alert(data.name+' konnte nicht hinzugefuegt werden. ('+JSON.stringify(e)+')');
});
```

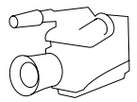
- die Anwendungslogik, also was dann bei dem POST auf die Ressource */foo* passiert findet sich dann in dem Callback der Route vom Node-Server

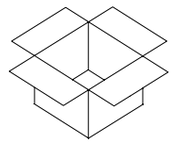
25.03.14

Aufgaben - Workshop 2

WBA2 Workshop

Fachhochschule Köln
Cologne University of Applied Sciences



Viel drin
Medieninformatik an der Fachhochschule Köln

Aufgaben

- Für die Bearbeitung der Aufgaben, können die HTML Seite bzw. Fragmente des POST Formulares aus dem ersten Workshop als Grundlage verwendet werden!
- Nutzen sie ansonsten nur:
  - Server: das Express Modul
  - Client: AJAX & jQuery
- Abgabe bis Sonntag 30.3.2014 18 Uhr im github.

Aufgabe 1

Schreiben sie einen Webserver (localhost Port=3000) mittels Express, der bei einem GET auf die Ressource "/planeten" eine HTML Seite zurück gibt, welche die Planeten aus dem ersten Workshop in einer Tabelle darstellt. Die Planeten sollten zur besseren Verarbeitung in einer JSON Datenstruktur gespeichert sein (vorerst als lokale Variable).

Ausgabe so ähnlich ausreichend:

Planeten	Entfernung zur Sonne in Mio km	Durchmesser in km
Merkur	58	4.879
Erde	150	12.734

Aufgabe 2

Über ein Formular in der HMTL Seite soll nun mittels AJAX und jQuery ein POST auf die Ressource "/planeten" erfolgen. Das übergebene Objekt sollte eine json Datenstuktur sein, damit diese möglichst einfach serverseitig weiterverarbeitet werden kann.

Handeln Sie die Daten des POST-Requests serverseitig, indem sie den Request-Body auf der Konsole loggen und dann ggf. die jeweiligen Änderungen an den Daten vornehmen.

Der nächste Schritt ist nun das Senden einer Response an den Client, welche Aufschluss über den Erfolg der Methode gibt. Der Client sollte je nach Response eine Meldung anzeigen (alert) und das Formular sollte anschließend geleert werden.

Um den Erfolg des POST-Request zu kontrollieren, können Sie nun den GET-Request aus Aufgabe 1 wiederholen um die nun erweiterte Tabelle einzusehen.

Fragen?