

Softwareprojekt 2

Node.js und Datenbanken

Einstieg in Express.js

Prof. Dr. Darius Schippritt

Büro L4.2-E02-140

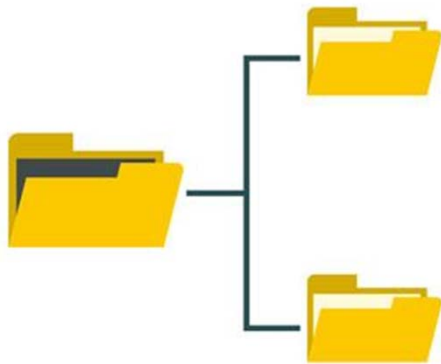
darius.schippritt@hshl.de

Überblick

- **Das letzte Mal... und Lernziele**
- Node.js und Datenbanken
- Einstieg in Express.js
- Zum Schluss...

Das letzte Mal...

Node Package Manager



Zugriff auf das
Dateisystem

ECMAScript 6



Processes

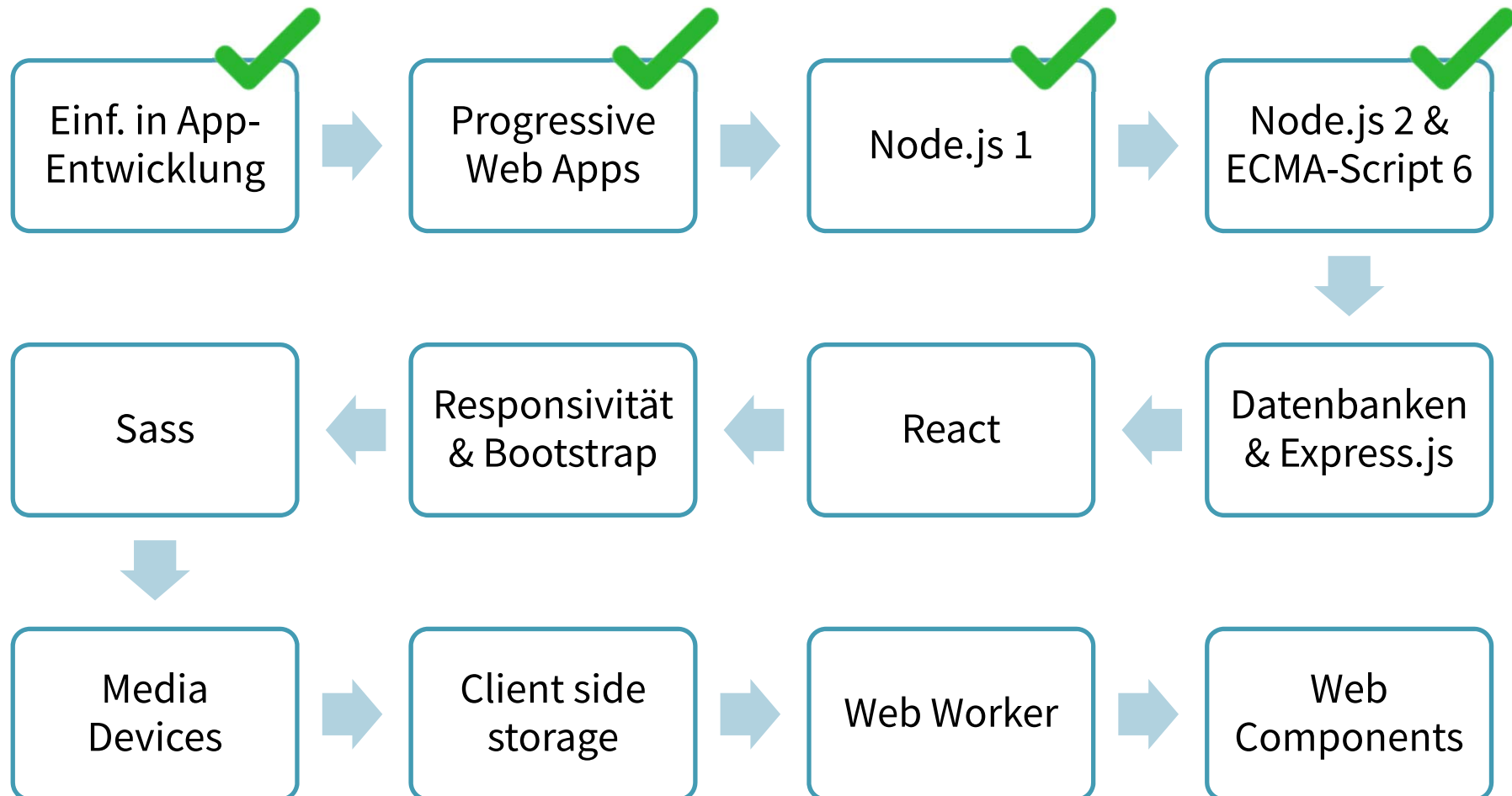
Event Loop



Streams

Upload von Dateien

Ziele und Inhalte



Lernziele



- Sie können aus Node.js auf MySQL- und MongoDB-Datenbanken zugreifen.
- Sie können bei MongoDB CRUD-Zugriffe durchführen.
- Sie können mit express das Routing steuern.
- Sie können mit express Formulardaten verarbeiten.
- Sie können in Node.js per REST Daten zwischen Client und Server austauschen.

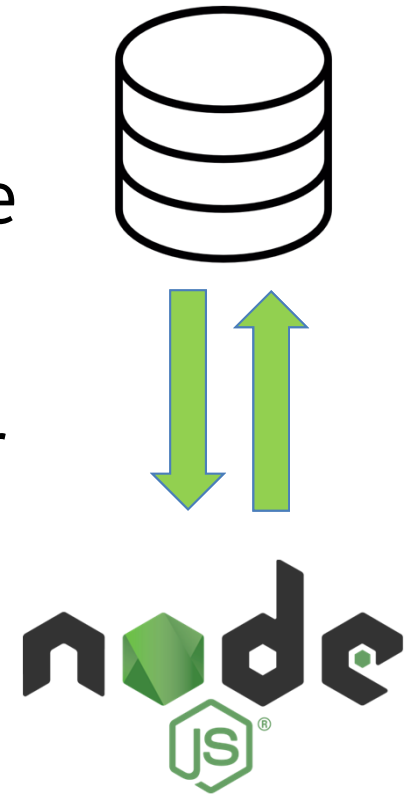
Überblick

- Das letzte Mal... und Lernziele
- **Node.js und Datenbanken**
- Einstieg in Express.js
- Zum Schluss...

Node.js und Datenbanken

Einführung

- Node.js ermöglicht Zugriff auf externe Datenbanken
- Verschiedene externe Module stellen Konnektoren für die Datenbanken zur Verfügung
- DB-Module müssen per **npm** installiert werden
- Zugriff erfolgt auf eigenständige DB
- Für MySQL z.B. im XAMPP-Paket



Node.js und Datenbanken

Zugriff auf MySQL

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

var con = mysql.createConnection({
  host: "localhost",
  database: "hshl",
  user: "php",
  password: "12345678"
});

con.connect(function(error) {
  if (error) {
    throw error;
  }
  else {
    con.query("SELECT * FROM user",
      function (error, result, fields) {
        if (error) {
          throw error;
        }
        else {
          console.log(result);
          console.log(result[0].login);
        }
      }
    );
  }
});
});
```

- Verwendetes Modul **mysql**
- Erstellung eines Connection-Objektes
- Aufbau der Verbindung mit **.connect()**
- **.query** führt SQL-Befehl aus
- Optionaler Parameter **fields** liefert Informationen über Tabellenstruktur

Node.js und Datenbanken

Zugriff auf MySQL im Webserver-Skript

```
http.createServer(function(request, response) {  
    response.writeHead(200, {"content-type": "text/html; charset=utf-8"});  
  
    var con = mysql.createConnection({  
        host: "localhost",  
        database: "hshl",  
        user: "php",  
        password: "12345678"  
    });  
  
    var message = "123";  
    EventEmitter.on("db-ready", sendResponse);  
  
    con.connect(function(error) {  
        if (error) {  
            //throw error;  
            message = "Connection error: "+error.message;  
            EventEmitter.emit("db-ready");  
        }  
        else {
```

ACHTUNG!!
DB-Zugriff erfolgt
asynchron

Node.js und Datenbanken

Zugriff auf MySQL im Webserver-Skript

```
else {
    con.query("SELECT * FROM user", function (error, result, fields) {
        if (error) {
            //throw error;
            message = "Query error: "+error.message;
            EventEmitter.emit("db-ready");
        }
        else {
            message = result[0].login;
            EventEmitter.emit("db-ready");
        }
    });
}
});

function sendResponse() {
    var htmlResponse = getHTMLStruct("Webtechnologien",
    "<h1 style='color : red; '>" + message + "</h1>");

    response.end(htmlResponse);
}
```

Node.js und Datenbanken

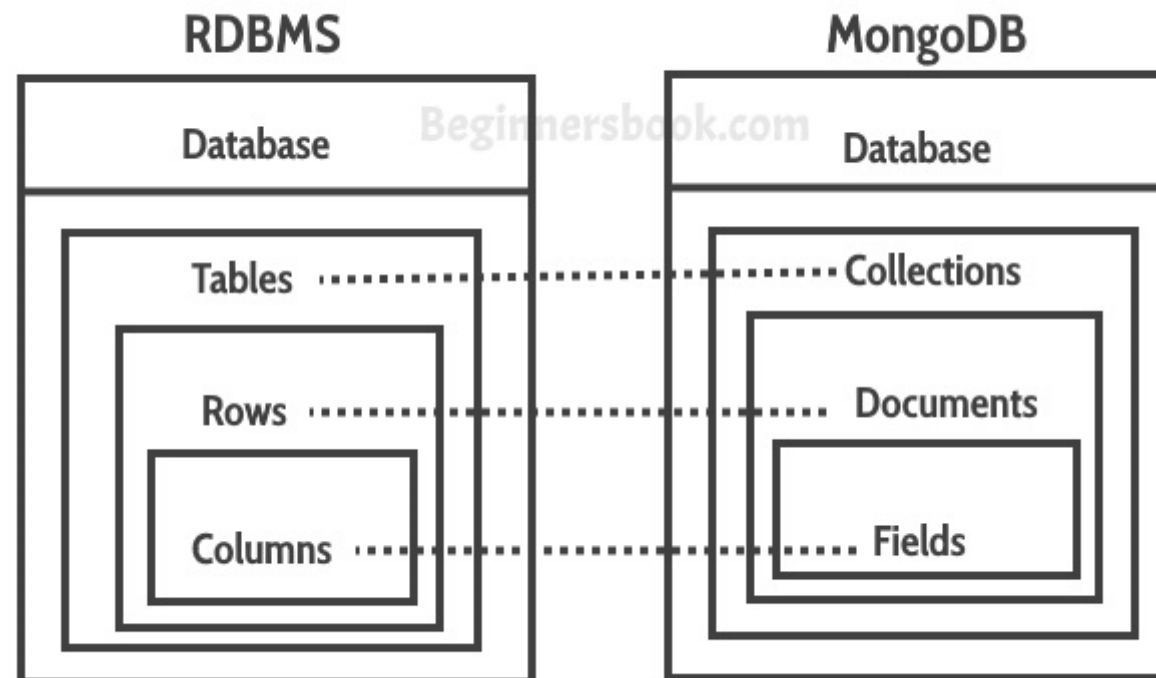
MongoDB



- MongoDB ist die Standarddatenbank im MEAN-Stack
- Name von humongous = gigantisch abgeleitet
- Ist eine dokumentenorientierte NoSQL-Datenbank
- Erlaubt direkte Speicherung von JSON-Dokumenten
- Kostenlos unter AGPL-Lizenz nutzbar

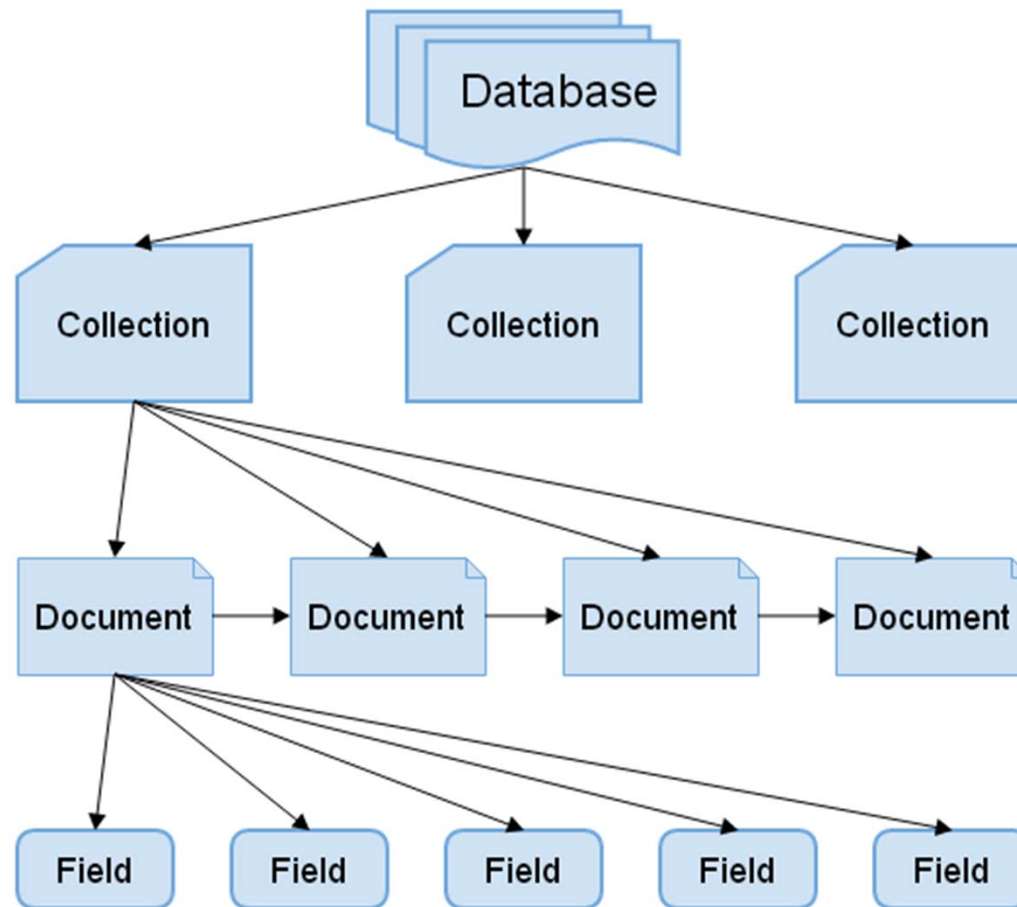
Node.js und Datenbanken

MongoDB



Node.js und Datenbanken

MongoDB



Node.js und Datenbanken

MongoDB

- Download der kostenlosen Community Server Version unter <https://www.mongodb.com/download-center?jmp=nav#community>
- Start per `mongod.exe`

```
C:\Progs\mongodb\bin>mongod.exe
2018-01-08T06:27:21.271-0800 I CONTROL [initandlisten] MongoDB starting : pid=11404 port=27017 dbpath=C:\data\db\ 64-bit
...
2018-01-08T06:27:21.281-0800 I STORAGE [initandlisten] exception in initAndListen: 29 Data directory C:\data\db\ not found, terminating
2018-01-08T06:27:21.281-0800 I NETWORK [initandlisten] shutdown: going to close listening sockets...
2018-01-08T06:27:21.283-0800 I NETWORK [initandlisten] shutdown: going to flush diaglog...
2018-01-08T06:27:21.283-0800 I CONTROL [initandlisten] now exiting
2018-01-08T06:27:21.287-0800 I CONTROL [initandlisten] shutting down with code:100
```

- Standardmäßig werden Daten in `c:\data\db` erwartet
- Fehlermeldung, falls Verzeichnis nicht vorhanden

Node.js und Datenbanken

MongoDB

- Alternativer Datenpfad durch Angabe des Parameters dbpath, z.B.:
 - **dbpath=c:\progs\mongodb\data\db**
- MongoDB lauscht auf Port 27017

```
C:\Progs\mongodb\bin>mongod.exe -dbpath=c:\progs\mongodb\data\db
2018-01-08T06:34:19.726-0800 I CONTROL [initandlisten] MongoDB starting : pid=2780 port=27017 dbpath=c:\progs\mongodb\data\db
...
2018-01-08T15:34:20.160+0100 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'c:\progs\mongodb\data\db\diagnostic.data'
2018-01-08T15:34:20.164+0100 I NETWORK [thread1] waiting for connections on port 27017
```

Node.js und Datenbanken

Zugriff auf MongoDB

- Benötigt Modul **mongodb**
- Bei Zugriff auf DB legt MongoDB diese automatisch an, falls nicht vorhanden
- Collection entspricht Relation bei MySQL
 - Wird automatisch angelegt beim Speichern des 1. Datensatzes

```
var mongo = require("mongodb").MongoClient;

mongo.connect("mongodb://localhost:27017/", {useNewUrlParser: true},
  function(error, client) {
    if (error) {
      throw error;
    }
    else {
      const collUser = client.db("newDB").collection("user");

      client.close();
    }
  });
```


Node.js und Datenbanken

Zugriff auf MongoDB – Create

- `createData()` erzeugt einen oder mehrere Datensätze in einer Collection

```
var mongo = require("mongodb").MongoClient;

mongo.connect("mongodb://localhost:27017/", {useNewUrlParser: true},
  function(error, client) {
    if (error) {
      throw error;
    }
    else {
      const collUser = client.db("newDB").collection("user");

      createData(collUser,
        [{vorname:"Darius", name:"Schippritt"},
        {vorname:"Max", name:"Mustermann"}]);

      client.close();
    }
  });

function callbackFunction(result, message = "") {
  if (message !== "") {
    console.log(message+": "+result);
  }
}
```

ACHTUNG!!
DB-Zugriff erfolgt
asynchron

Node.js und Datenbanken

Zugriff auf MongoDB – Create

```
createData(collUser,  
  [{vorname:"Darius", name:"Schippritt"}  
  ,{vorname:"Max", name:"Mustermann"}]);
```

```
function createData(collection, data, callback = callbackFunction) {  
  if (!Array.isArray(data)) {  
    collection.insertOne(data, function(error, result) {  
      if (error) {  
        throw error;  
      }  
      else {  
        callback(result, "data insertet");  
      }  
    });  
  }  
  else {  
    collection.insertMany(data, function(error, result) {  
      if (error) {  
        throw error;  
      }  
      else {  
        callback(result, data.length+" data insertet");  
      }  
    });  
  }  
}
```

Node.js und Datenbanken

Zugriff auf MongoDB – Read

- `.find()` liefert alle Ergebnisse
- `.findOne()` liefert nur das erste Ergebnis

```
readData(collUser, {name:"Mustermann"});
```

```
function readData(collection, query = {}, callback = callbackFunction) {  
  collection.find(query).toArray(function(error, result) {  
    if (error) {  
      throw error;  
    }  
    else {  
      callback(result, "found "+result.length+" results");  
    }  
  });  
}
```

Node.js und Datenbanken

Zugriff auf MongoDB – Update

- `.updateMany()` aktualisiert alle Treffer
- `.updateOne()` aktualisiert einen Treffer

```
updateData(collUser, {name:"Mustermann"}, {$set: {vorname:"Tim"}});
```

```
function updateData(collection, query = {}, values = {},  
  callback = callbackFunction) {  
  collection.updateMany(query, values, function(error, result) {  
    if (error) {  
      throw error;  
    }  
    else {  
      callback("", result.result.nModified+" dataset(s) updated");  
    }  
  });  
}
```

Node.js und Datenbanken

Zugriff auf MongoDB – Delete

- `.deleteMany()` löscht alle Treffer
- `.deleteOne()` löscht den ersten Treffer

```
deleteData(collUser, {vorname:"Tim"});
```

```
function deleteData(collection, query = {}, callback = callbackFunction) {  
    collection.deleteMany(query, function(error, result) {  
        if (error) {  
            throw error;  
        }  
        else {  
            callback("", result.result.n+" dataset(s) deleted");  
        }  
    });  
}
```

Überblick

- Das letzte Mal... und Lernziele
- Node.js und Datenbanken
- **Einstieg in Express.js**
- Zum Schluss...

Einstieg in Express.js

Übersicht

express

- Express bzw. Express.js stellt ein Framework für Node.js dar und wird als Modul implementiert
- erweitert Node.js um zahlreiche serverseitige Funktionen
- Es ist unter der MIT-Lizenz kostenlos nutzbar
- Einige der Kernfunktionen sind:
 - Ermöglicht Applikationen, die auf HTTP-Requests antworten
 - Ermöglicht Routing Tables für verschiedene HTTP-Methoden und URLs
 - Erlaubt die Generierung dynamischer Webseiten auf Basis von Templates

Einstieg in Express.js

Installation

express

- Installieren Sie express über die Kommandozeile per Befehl **npm install express --save**
- Weitere sinnvolle Module sind
 - **body-parser**: bietet Unterstützung für das Handling von Formularen, Text, JSON...
 - **cookie-parser**: Unterstützung für das Cookie-Handling
 - **multer**: unterstützt das Parsen von multipart/form-data (Dateien)
- Installieren Sie ggf. diese zusätzlichen Module über die Kommandozeile

Einstieg in Express.js

Erste Express-Anwendung

```
var express = require("express");
var app = express();

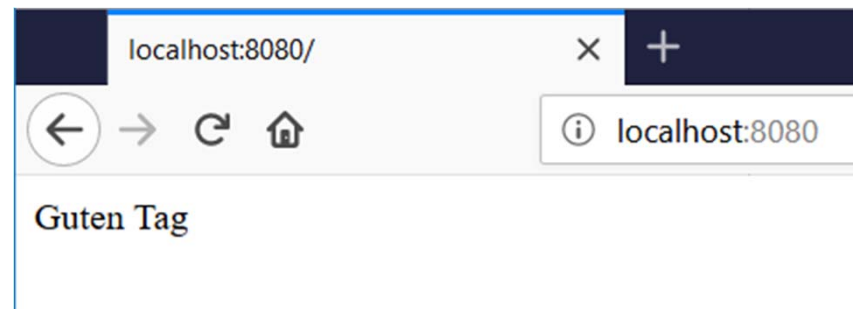
app.get("/", function(request, response) {
  response.send("Guten Tag");
});

var server = app.listen(8080, function() {
  var host = server.address().address;
  var port = server.address().port;

  console.log("Express app listening at http://%s:%s", host, port);
});
```

App antwortet auf GET-Anfrage und Pfad /

Express app listening at http://:::8080



Einstieg in Express.js

Routing

```
app.get("/", function(request, response) {  
  response.send("Guten Tag");  
});  
  
app.get("/show", function(request, response) {  
  response.send("SHOW");  
});  
  
app.post("/test", function(request, response) {  
  response.send("TEST");  
});  
  
app.get("/*1*", function(request, response) {  
  response.send("1111");  
});
```

App antwortet auf GET-Anfrage und Pfad /show

App antwortet auf POST-Anfrage und Pfad /test

App antwortet auf GET-Anfrage und Pfad /*1*

http://localhost/1
http://localhost/ab1cd



Einstieg in Express.js

Bereitstellung statischer Dateien

- Mit Express können statische Dateien „durchgeschleust“ werden
- **`.use(express.static())`** definiert das Verzeichnis, in dem die Dateien liegen

```
var express = require("express");
var app = express();

app.use(express.static("static"));

app.get("/", function(request, response) {
  response.send("Guten Tag");
});

var server = app.listen(8080, function() {
  var host = server.address().address;
  var port = server.address().port;

  console.log("Express app listening at http://%s:%s", host, port);
});
```

Verzeichnisstruktur

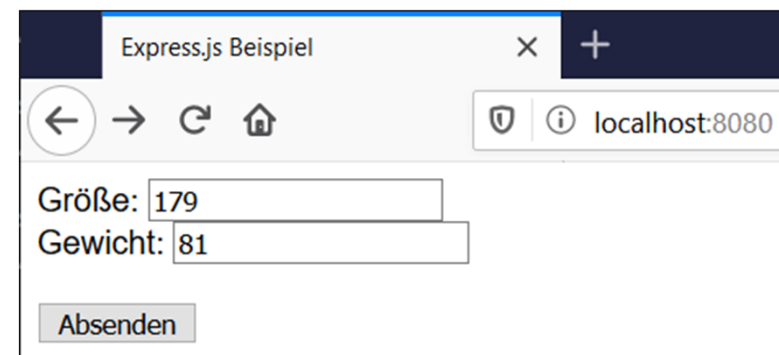
```
./
- static
  - css
  - images
    - darthvader.jpg
```

Einstieg in Express.js

Verarbeitung von Formulardaten per GET

- Route „form_get“ verarbeitet GET-Daten
- Aufruf ohne Route liefert Formular.html
- **JSON.stringify()** wandelt Antwort in JSON-Objekt um

```
app.get("/", function(request, response) {  
  response.sendFile(__dirname+"/index.html");  
});  
  
app.get("/form_get", function(request, response) {  
  responseJSON = {  
    size:request.query.size,  
    weight:request.query.weight,  
    bmi:(request.query.weight/  
          (request.query.size*request.query.size)  
          *10000)  
  };  
  response.end(JSON.stringify(responseJSON));  
});
```



Express.js Beispiel

Größe: 179

Gewicht: 81

Absenden



127.0.0.1:8080/form_get?size=179&weight=81

{"size":"179","weight":"81","bmi":25.28010985924285}

Einstieg in Express.js

Verarbeitung von Formulardaten per POST

extended:false → Bibliothek = querystring
extended:true → Bibliothek = qs

```
var parser = require("body-parser");
var urlParser = parser.urlencoded({extended:false});

app.post("/form_post", urlParser, function(request, response) {
  responseJSON = {
    size:request.body.size,
    weight:request.body.weight,
    bmi:(request.body.weight/(request.body.size*request.body.size)*10000)
  };
  response.end(JSON.stringify(responseJSON));
});
```

Express.js Beispiel

localhost:8080

Größe:

Gewicht:

Absenden

127.0.0.1:8080/form_post

127.0.0.1:8080/form_post

{"size":"179","weight":"81","bmi":25.28010985924285}

Einstieg in Express.js

Upload von Dateien

- Verwendung des Moduls **multer**
- Upload-Formular mit **enctype="multipart/form-data"**
- Upload erfolgt automatisch; umbenennen mit **fs.rename()**

```
var parser = require("body-parser");
var multer = require("multer");
var upload = multer({dest: "uploads/"});
var fs = require("fs");

app.post("/file_upload", upload.single("file"), function(request, response) {
  var file = __dirname + "/" + request.file.originalname;
  fs.rename(request.file.path,
    request.file.destination+request.file.originalname,
    (error) => {
      if (error)
        throw error;
      responseJSON = {
        message: "File uploaded successfully",
        filename: request.file.originalname
      };
      response.end(JSON.stringify(responseJSON));
    });
});
```

request.file

```
{ filename: 'file',
  originalname: 'screenshot google coding.png',
  encoding: '7bit',
  mimetype: 'image/png',
  destination: 'uploads/',
  filename: 'c0e7044168af5ee5a38fee288138356d',
  path: 'uploads\\c0e7044168af5ee5a38fee288138356d',
  size: 145413 }
```


Einstieg in Express.js

Upload von Dateien

- **multer** erlaubt auch den Upload mehrerer Dateien

```
<input type="file" name="files" multiple required>
```

```
app.post("/file_upload", upload.array("files"), function(request, response) {  
  for (var index in request.files) {  
    var reqFile = request.files[index];  
    var file = __dirname + "/" + reqFile.originalname;  
    fs.rename(reqFile.path,  
      reqFile.destination+reqFile.originalname,  
      (error) => {  
        if (error)  
          throw error;  
      });  
  }  
  responseJSON = {  
    message: request.files.length+" files uploaded successfully"  
  };  
  response.end(JSON.stringify(responseJSON));  
});
```

Einstieg in Express.js

Verwendung von Cookies

- Verwendung des Moduls **cookie-parser**
- **.cookie(name, value, expire)** setzt Wert
- **.cookies[name]** ruft Wert ab
- **.clearCookie(name)** löscht Cookie

```
var express = require("express");
var cookieParser = require("cookie-parser");
var app = express();
app.use(cookieParser());

app.get("/cookie_set", function(request, response) {
  response.cookie("hshl", "myValue", {expire : new Date() + 10000})
  .end("set cookie");
});

app.get("/cookie_get", function(request, response) {
  console.log(request.cookies);
  response.end("Cookie: "+request.cookies["hshl"]);
});

app.get("/cookie_delete", function(request, response) {
  response.clearCookie("hshl");
  response.end("Cookie deleted");
});
```


Einstieg in Express.js

RESTful API

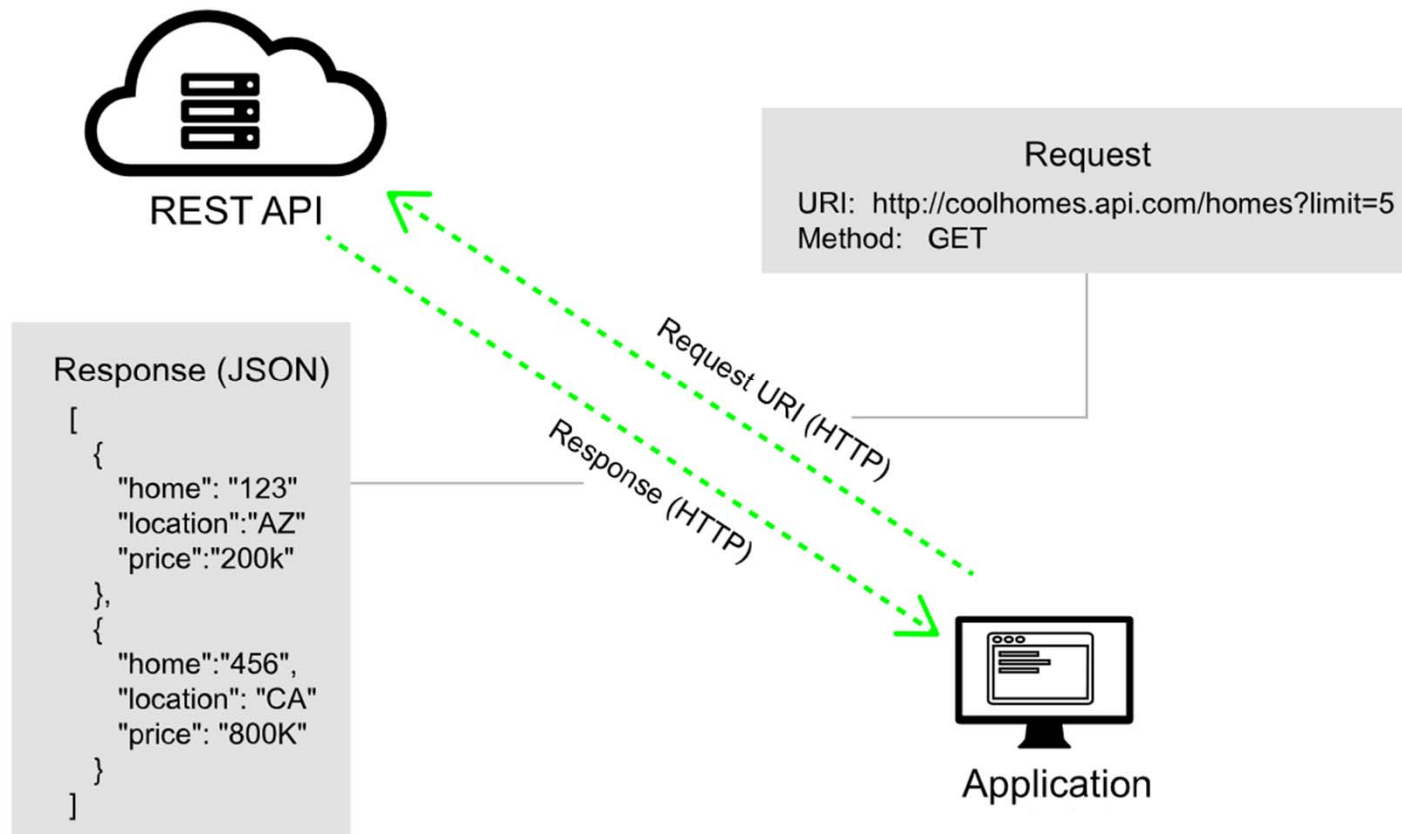


- REST steht für Representational State Transfer ~ Übertragung des aktuellen Zustands einer Information
- Alternative zu SOAP und WSDL
- Asynchrone kleine Pakete → schnelle Reaktionszeiten
- Vier HTTP-Methoden
 - GET: Abrufen/Lesen von Informationen
 - PUT: Schreiben neuer Information
 - DELETE: Löschen von Informationen
 - POST: Update bestehender oder Schreiben neuer Information

Einstieg in Express.js

RESTful API Kommunikationsmodell

REST API model



Referenz: https://idratherbewriting.com/learnapidoc/images/restapi_restapi.svg

Einstieg in Express.js

RESTful API

```
{
  "user1":{
    "name":"Bob",
    "vorname":"Sponge",
    "email":"sbob@bikini-bottom.tv"
  },
  "user2":{
    "name":"Skywalker",
    "vorname":"Luke",
    "email":"luke_skywalker@jedi.com"
  },
  "user3":{
    "name":"America",
    "vorname":"Captain",
    „mail“:"c.america@firstavenger.gov"
  }
}
```

- Datei users.json repräsentiert eine JSON-basierte Datenbank
- Anwendung bietet folgende REST-Methoden
 - listUsers → GET
 - getUser → GET
 - addUser → POST
 - removeUser → DELETE

Einstieg in Express.js

RESTful API

Methode listUsers liefert Inhalt von users.json

```
var express = require("express");
var app = express();
var fs = require("fs");

var userData = null;

app.get("/listUsers", function(request, response) {
  response.end(JSON.stringify(readUserData()));
});

var server = app.listen(8080, function() {
  var host = server.address().address;
  var port = server.address().port;

  console.log(
    "Express app listening at http://%s:%s"
    , host, port);
});

function readUserData(renewData = false) {
  if (userData === null || renewData) {
    userData = JSON.parse(
      fs.readFileSync(
        __dirname+"/users.json", "utf8"));
  }
  return userData;
}
```

```
{
  "user1":{
    "name":"Bob",
    "vorname":"Sponge",
    "email":"sbob@bikini-bottom.tv"
  },
  "user2":{
    "name":"Skywalker",
    "vorname":"Luke",
    "email":"luke_skywalker@jedi.com"
  },
  "user3":{
    "name":"America",
    "vorname":"Captain",
    "email":"c.america@firstavenger.gov"
  }
}
```

Einstieg in Express.js

Ausgabe anpassen

Methode **getUser** liefert Inhalt von **user[id]**

z.B.: **http://localhost:8080/getUser=3**

```
app.get("/getUser=:id", function(request, response) {  
    data = readUserData();  
    var user = data["user"+request.params.id];  
    response.end(JSON.stringify(user));  
});
```

```
{  
    "name": "America",  
    "vorname": "Captain",  
    "email": "c.america@firstavenger.gov"  
}
```

Einstieg in Express.js

RESTful API

Methode **addUser** fügt user4 ein und gibt erweiterte Liste zurück

```
app.get("/addUser", function(request, response) {  
  fs.readFile(__dirname+"/users.json", "utf8",  
    function(error, data) {  
      data = readUserData();  
      data["user4"] = {  
        "name" : "Schippritt",  
        "vorname" : "Darius",  
        "email" : "darius.schippritt@hshl.de"  
      };  
      response.end(JSON.stringify(data));  
    }  
  });  
});
```

```
{  
  "user1":{  
    "name":"Bob",  
    "vorname":"Sponge",  
    "email":"sbob@bikini-bottom.tv"  
  },  
  "user2":{  
    "name":"Skywalker",  
    "vorname":"Luke",  
    "email":"luke_skywalker@jedi.com"  
  },  
  "user3":{  
    "name":"America",  
    "vorname":"Captain",  
    "email":"c.america@firstavenger.gov"  
  },  
  "user4":{  
    "name":"Schippritt",  
    "vorname":"Darius",  
    "email":"darius.schippritt@hshl.de"  
  },  
}
```

Einstieg in Express.js

Ausgabe anpassen

Methode **removeUser** entfernt **user[id]** und gibt restliche Liste zurück

z.B.: **http://localhost:8080/removeUser=2**

```
app.get("/removeUser=:id", function(request, response) {  
    data = readUserData();  
    delete data["user"+request.params.id];  
    response.end(JSON.stringify(data));  
});
```

```
{  
  "user1":{  
    "name":"Bob",  
    "vorname":"Sponge",  
    "email":"sbob@bikini-bottom.tv"  
  },  
  "user3":{  
    "name":"America",  
    "vorname":"Captain",  
    "email":"c.america@firstavenger.gov"  
  }  
}
```

Überblick

- Das letzte Mal... und Lernziele
- Node.js und Datenbanken
- Einführung in Express.js
- **Zum Schluss...**

Folgendes sollten Sie nun (beantworten) können:

- Mit welcher Funktion wird gegen eine MySQL-Datenbank ein SQL-Befehl ausgeführt?
- Wie lässt sich auf das Ergebnis einer MySQL-Anfrage reagieren?
- Was bewirkt die Funktion **`.use(express.static())`** ?
- Wie kann auf Cookies zugegriffen werden?
- Was ist das besondere an REST? Wie erfolgt das Verbindungsmanagement?



Zum Schluss...

Weiterführende Links und Literatur

[Node.js] <https://nodejs.org/en/>

[Node.js API-Beschreibung] <https://nodejs.org/api/>

[Tutorial für Node.js] <http://www.w3ii.com/nodejs/default.html>

[npm] <https://www.npmjs.com/>

[Tutorial für MongoDB] <https://www.tutorialspoint.com/mongodb/>

[Tutorial für MongoDB] https://www.w3schools.com/nodejs/nodejs_mongodb.asp

[Express.js] <http://expressjs.com/de/>

[Tutorial für Express.js] <https://www.tutorialspoint.com/expressjs/>

Golo Roden: „Node.js & Co: Skalierbare, hochperformante und echtzeitfähige Webanwendungen professionell in JavaScript entwickeln“, dpunkt.Verlag GmbH, 1. Auflage, 2012.

Sebastian Springer: „Node.js: Das Praxisbuch“, Rheinwerk Computing, 2. Auflage, 2016.

Zum Schluss...

Quellen

[sonstige Abbildungen] <https://pixabay.com> und <https://icon-icons.com>

Vielen Dank für Ihr Interesse!

