1 Node JS

Was muss ein Webserver können?
- HTTP Anfragen annehmen
- Actions ausführen basierend auf der Anfrage URL
- HTTP Antworten absenden

Request: Methoden GET, PUT, POST, ...

Response: Methoden writeHead, setHeader, statusCode, statusMessage, write, end

```
response.writeHead(200, {'Content-Length': body.length,'
    Content-Type': 'text/plain'});
response.setHeader("Content-Type", "text/html");
response.statusCode = 404;
response.statusMessage = 'Not found';
response.write("Data");
response.end("Data");
```

Module:
Node verwendet für die Module Verwaltung npm.

Import/Export

```
export router; // Variable
import router from "./file.js";
export {function, otherFunction}; // several Functions
import controller from './controller.js';
export const noteService = new NoteService(); // Class
import {noteService} from "./noteServices.js";
import express from "express"; // ES6
```

package.json:
- Beinhaltet die Informationen zum Projekt
- Wird benötigt um es zu publishen
- Wird benötigt um Module zu installieren
- Definiert Skripts (bsp: npm run test)

```
{ "name": "my_package",
"description": "",
"version": "1.0.0",
"main": "index.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1" },
"repository": {
"type": "git",
"url": "https://github.com/mgfeller/my_package.git" },
"keywords": [],
"author": "",
"license": "ISC",
"bugs": {
"url": "https://github.com/mgfeller/my_package/issues" },
"homepage": "https://github.com/mgfeller/my_package"
```

2 Express

Server starten:

```
import http from "http";
import express from "express";
const app = express();
const server = http.createServer(app);
const hostname = '127.0.0.1'; const port = '3000';
server.listen(port, hostname, () => {
  console.log('Running at http://${hostname}:${port}/');
});
```

JSON (JavaScript Object Notation):
- Ist ein Daten-Austauschformat
- Wird verwendet um Daten zu senden und speichern
- Hat im Web XML verdrängt
- Wird oft mit AJAX verwendet
- Datentypen: String, Number, Boolean, Array, Object, null
- JSON-Helper: JSON.parse & JSON.stringify

MVC-Pattern:
- Model: Daten und Datenaufbereitung
- Controller: Verknüpft die View mit den Daten
- View: Darstellen der daten

Middleware:
- Wird für Request Bearbeitung gebraucht
- Middleware ist ein Stack von Anweisungen welche für einen Request ausgeführt wird
- Neue Middleware registrieren mit: app.use(..);

2.1 Routing

Router-Middleware:

```
// Middleware befindet sich auf dem Express Objekt
import express from "express";
const router = express.Router;
// HTTP Methoden (get, put, post, delete)
router.get('/', function(req, res) {
    res.send('hello world'); });
// Mehrere Methoden auf selbem Link mit .route
app.route('/book') // oder router.route(..)
    .get(function(req, res) {res.send('Get a book');})
    .post(function(req, res) {res.send('Add a book');})
```

BodyParser-Middleware:

```
import bodyParser from "body-parser";
app.use(bodyParser.json());
```

Static-Middleware:
- Aufgabe: Statische Files ausliefern
- Nutzen wie folgt:
  app.use(express.static(__dirname + '/public'));
  app.use(express.static(path.join(path.resolve(), 'public')));
- Es sind mehrere static-routes möglich

Custom-Middleware:
Hat 3 Parameter: request, response, next

```
function myDummyLogger( options ){
    options = options ? options : {};
    return function myInnerDummyLogger(req, res, next) {
        console.log(req.method +":"+ req.url);
        next(); } }
app.use(myDummyLogger());
```

Error-Middleware:
- Bearbeitet Errors, welche von Middlewares generiert werden
- Hat 4 Parameter: error, request, response & next
- Sollte als letzte Middleware registriert werden
- Wird aufgerufen, falls ein error-Objekt dem Next-Callback übergeben wird.

```
app.use(function(err, req, res, next) {
    console.error(err.stack);
    res.status(500).send('Something broke!'); });
```

2.2 Model

Ziel: Die Daten sollten in einem Module verwaltet und abgespeichert werden. Möglichkeiten: In Memory (array), JSON, NoSQL-Datenbanken (nedb), Sql-Datenbanken, Oracle-datenbank.

```
// beispiel: nedb
import Datastore from "nedb";
const db = new Datastore({ filename: './data/order.db',
     autoload: true });
// insert
db.insert(order, function(err, newDoc) {
    if(callback) {
        callback(err, newDoc); } });
// search: findOne oder findAll
db.findOne({ _id: id }, function (err, doc) {
    callback( err, doc); });
// update
db.update({_id: id}, {$set: {"state": "DELETED"}}, {},
     function (err, doc) {
    publicGet(id,callback); });
```

2.3 View

Ziel: Trennen von Controller und View mittels Template Engine. Express bietet eine render Methode an: app.render(view, [locals], callback);

```
// view engine setup
app.set('views', path.join(path.resolve(), 'views'));
app.set('view engine', 'hbs');
```

2.4 Session & Security

Session:
Beim ersten "Connect"vom Client wird eine Session-Id erstellt und als Cookie zum Client geschickt. Die Session-Daten werden auf dem Server abgespeichert. → Wiederspricht REST

Nutzen: HTTP-Stateless umgehen und z.B. Login Status von user abspeichern, oder allgemein Daten Server-Seitig einem Benutzer zuordnen. Ermöglicht tracking.

```
// Cookie verwenden:
app.use(require("cookie-parser")());
// Session benötigt Cookies
app.use(session({ secret:    '1234567', resave: false,
     saveUninitialized: true}));
```

2.5 Rest & Ajax

Token:
Ziel: Stateless Server. **Idee: Bei jeder Anfrage muss für die Authorisierung ein Token mitgegeben werden. Vorteil:** Jede Anfrage kann zu einem beliebigem Server gesendet werden. **Nachteile:** Was passiert wenn der Token geklaut wird? → Ablaufdatum kurz setzen, Token invalidieren.

JWT-Token: import jwt from 'express-jwt';

2.6 Web Sockets

Das klassische Model vom Request-Response hat 2 Probleme: Der Server kann keine Nachricht an den Client schicken. Jede Anfrage öffnet eine neue Verbindung.
Dieses Model erschwert es real-time Apps zu machen (Games, Chats). Lösung: WebSockets ermöglichen "bi-directional", "always-on"Kommunikation.

3 EcmaScript

4 Typescript

5 Responsive Design

6 Testing

7 Security

8 Accessibility

9 Animation

10 UX Research, Information Architecture

11 Internationalization

12 Dev-Ops