

Node.js: Einführung



- Eigenschaften und Vorteile einordnen können
- Das Versioning verstehen
- Das erste Programm und die erste einfache Web-Anwendung erstellen und starten können
- Den Aufbau von Node.js und das Modulsystem verstehen
- Wichtige globale Objekte, Variablen und Funktionen kennen lernen
- JavaScript Modulsysteme wie CommonJS und ECMAScript verwenden können
- Die Begriffe Destructuring verstehen
- Das Framework Express zur Entwicklung komplexerer Web-Anwendungen einsetzen können
- Den Aufbau von Express verstehen und insbesondere den Begriff Middleware einordnen können
- Eine Web-Anwendung nach dem MVC-Pattern erstellen können
- Dabei das Routing, das Bereitstellen von statischen Inhalten sowie den Umgang mit POST-Parametern anwenden können
- Anhand der Anleitungen im Exkurs durch die Middleware Passport Web-Anwendungen mittels Authentifizierung schützen können

Literaturhinweis

Node.js, Das umfassende Handbuch, Sebastian Springer, Rheinwerk Computing, ISBN 978-3-8362-6255-2

Eigenschaften und Vorteile

- Serverseitige JavaScript-Plattform mit eventgetriebenem Ansatz
- JavaScript-Code wird äußerst performant und ressourcenschonend außerhalb des Browsers ausgeführt und zwar auf der *JavaScript-Engine V8* von Google die ursprünglich für Google Chrome entwickelt wurde
- JavaScript relativ einfach erlernbar
- Node.js unterstützt mehrere JavaScript-Versionen abhängig von der Version der verwendeten V8-Engine. JavaScript ist standardisiert in Spezifikation ECMA-262 (Abk. European Computer Manufacturers Association)
- Gut dokumentiert und Open Source
- Modularer Ansatz erlaubt das Einbinden einer Vielzahl von Bibliotheken für die unterschiedlichsten Zwecke durch den *Node Package Manager* (npm)
- Nonblocking I/O ermöglicht asynchrone Verarbeitung

Versioning¹

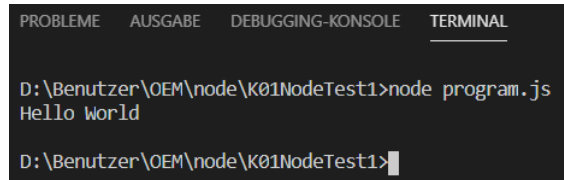
LTS <i>Long-Term-Support-Version</i>	Current
Gewährleistet längere Supportzeit (30 Monate)	Bibliotheken müssen noch angepasst werden
Gerade Versionsnummern:	von Current zu LTS
Ungerade Versionsnummern:	haben kurzes Support-Intervall

¹ Siehe <https://nodejs.org/en/about/releases/>

Das erste Programm²

program.js

```
console.log('Hello world');
```



A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'PROBLEME', 'AUSGABE', 'DEBUGGING-KONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active. The terminal shows the command 'D:\Benutzer\OEM\node\K01NodeTest1>node program.js' followed by the output 'Hello World' on the next line. The prompt 'D:\Benutzer\OEM\node\K01NodeTest1>' is visible at the bottom.

PROBLEM:

Bei jeder Programmänderung muss Programm beendet und neu gestartet werden

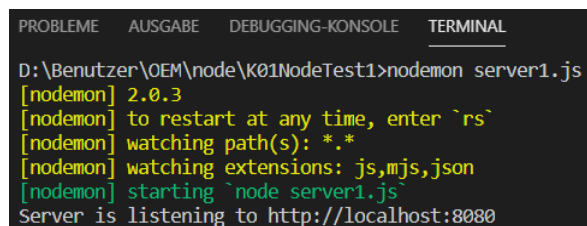
LÖSUNG:

```
$ npm install nodemon  
$ nodemon program.js
```

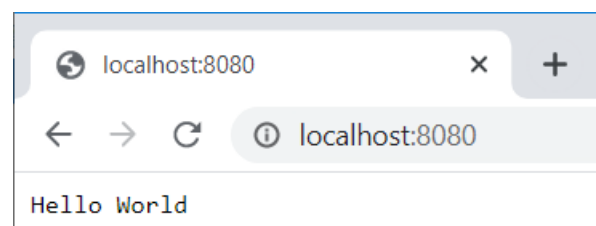
Web-Server antwortet mit Text

server1.js

```
const http = require('http');3  
const server = http.createServer(function(request, response) {  
  response.writeHead(200,  
    { 'content-type': 'text/plain; charset=utf8' });  
  response.write('Hello ');  
  response.end('world');  
});  
server.listen(8080, function() {  
  console.log('Server is listening to http://localhost:8080');  
});
```



A screenshot of a terminal window with a dark background. At the top, there are four tabs: 'PROBLEME', 'AUSGABE', 'DEBUGGING-KONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active. The terminal shows the command 'D:\Benutzer\OEM\node\K01NodeTest1>nodemon server1.js'. The output includes: '[nodemon] 2.0.3', '[nodemon] to restart at any time, enter `rs`', '[nodemon] watching path(s): *.*', '[nodemon] watching extensions: js,mjs,json', '[nodemon] starting `node server1.js`', and 'Server is listening to http://localhost:8080'.



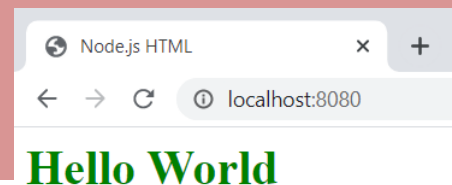
A screenshot of a web browser window. The address bar shows 'localhost:8080'. The page content displays 'Hello World'.

² node -v verwendete Node-Version

³ Einbinden des Http-Moduls welches in Node.js standardmäßig enthalten ist

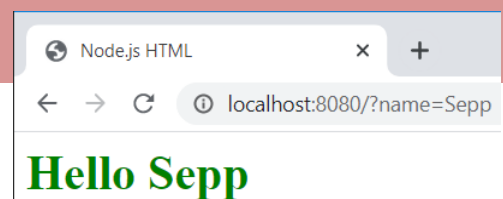
Der Web-Server antwortet mit HTML

```
...
const server = http.createServer(function(request, response) {
  response.writeHead(200,
    { 'content-type': 'text/html; charset=utf8' });
  const body = `<DOCTYPE html>
    <html>
      <head>
        <meta charset="utf-8">
        <title>Node.js HTML</title>
      </head>
      <body>
        <h1 style="color:green">Hello world</h1>
      </body>
    </html>`;
  response.end(body);
});
```



Der Web-Server generiert dynamische Antwort

```
...
const url = require('url');4
const server = http.createServer(function(request, response) {
  response.writeHead(200,
    { 'content-type': 'text/html; charset=utf8' });
  const parsedUrl = url.parse(request.url, true)5;
  const body = `<DOCTYPE html>
    <html>
      <head>
        <meta charset="utf-8">
        <title>Node.js HTML</title>
      </head>
      <body>
        <h1 style="color:green">Hello ${parsedUrl.query.name}</h1>
      </body>
    </html>`;
  response.end(body);
});
```



⁴ Einbinden des Kernmoduls url

⁵ Dadurch erhält query-Eigenschaft die Eigenschaft name mit Inhalt Sepp, sonst würde in query der Text name=Sepp stehen

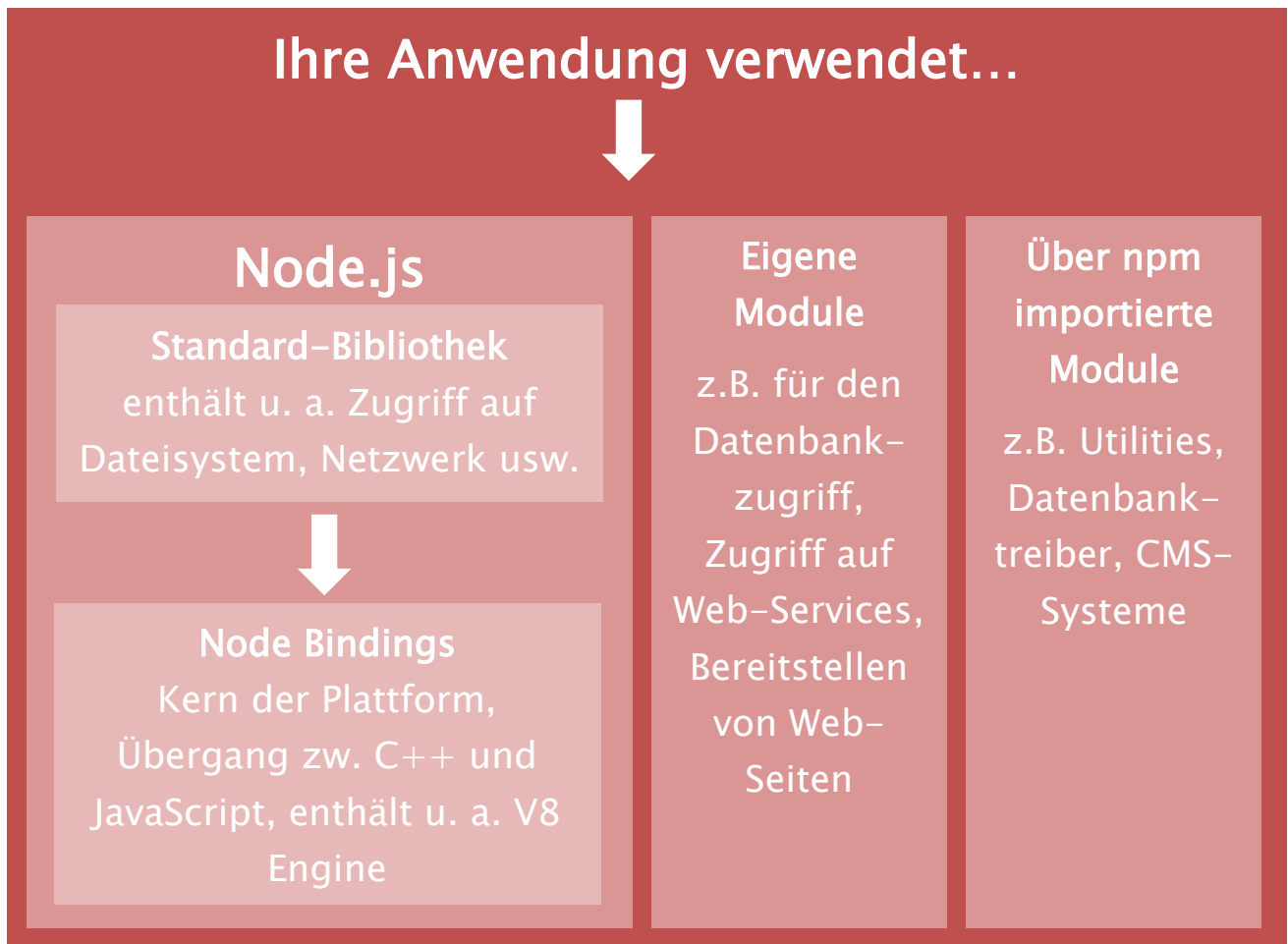
⁶ parse() ist **deprecated**, Möglichkeit `const myUrl = new URL(request.url, 'http://localhost:8080');`; `myUrl.searchParams.get('name')`

Module

... sind Komponenten die eine bestimmte Aufgabe erledigen und über Schnittstellen zusammenarbeiten

Node.js hat modularen Aufbau und auch Ihre Anwendung sollte diesen haben

Aufbau von Node.js



Dokumentation der Standard-Bibliothek <https://nodejs.org/api>
Stabilitätsindex für jedes Modul (Deprecated, Experimental, Stable)

Vorteile

- **Wiederverwendbarkeit**
- **Bessere Testbarkeit** durch kleine, abgeschlossene Komponenten
- **Parallelisierbarkeit** durch Unabhängigkeit der Komponenten
- **Austauschbarkeit** der Komponenten durch definierte Schnittstellen

Globale Objekte/Variablen/Funktionen, Laden von Modulen**modulesExampleSync.js**

```
console.log(`${__filename} ${__dirname}`);
const filename = 'output.txt';
const fs = require('fs')7;
if (fs.existsSync(filename)) {
  fs.truncateSync(filename);
}
let counter = 1;
const interval = setInterval(function() {
  console.log(`${counter} iteration`);
  fs.appendFileSync(filename, `${counter}\n`);
  if (counter++ >= 3) {
    clearInterval(interval);
  }
}, 1000);
```

```
D:\Benutzer\OEM\node\K01NodeTest1>node modulesExampleSync.js
D:\Benutzer\OEM\node\K01NodeTest1\modulesExampleSync.js
D:\Benutzer\OEM\node\K01NodeTest1
1 iteration
2 iteration
3 iteration
```

8

```
output.txt X
K01NodeTest1 > output.txt
1 1
2 2
3 3
4
```

FRAGE:

Welche global verfügbaren Objekte/Variablen/Funktionen werden in diesem Programm verwendet?

(HINWEIS: Die richtige Anzahl ist 6)

BEMERKUNG:

Für die Funktionen zur Dateimanipulation existieren auch die *asynchronen* Varianten

⁷ Modul fs (File System) wird geladen

⁸ Programm darf nicht mit nodemon gestartet werden

JavaScript-Modulsysteme

CommonJS-Module

`require()` liefert Objekt zurück,
das für öffentliche Schnittstelle
der Datei steht

ECMAScript-Module

Neuerer Standard der in Zukunft
verwendet werden soll, hat
derzeit noch Stabilitätsindex
Experimental


CommonJS-Module

`module-commonJS.js`

```
const a = 3;
function add(a, b) {
  return `${a} + ${b} = ${a + b}`;
}
const user = {
  name: 'Sepp',
  password: 'seppentinen'
}
class User {
  constructor(name, password) {
    this.name = name;
    this.password = password;
  }
  doSomething() {
    console.log(`${this.name} doesSomething`);
  }
}
module.exports = { a, add, user, User };
```

`use-commonJS.js`

```
const commonJS = require('./module-commonJS');
console.log(commonJS.a);
console.log(commonJS.add(commonJS.a, 4));
console.log(commonJS.user);
const user = new commonJS.User('Elfriede', 'rike');
console.log(user.name);
user.doSomething();
```



```
D:\Benutzer\OEM\node\K01NodeTest1>node use-commonJS.js
3
3 + 4 = 7
{ name: 'Sepp', password: 'seppentinen' }
Elfriede
Elfriede doesSomething
```

Destructuring

Extrahiert Elemente aus Objekten und Arrays

`use-commonJSDestructuring.js`

```
const { a, add } = require('./module-commonJS');
console.log(a);
console.log(add(a, 4));
```

```
D:\Benutzer\OEM\node\K01NodeTest1>node use-commonJSDestructuring.js
3
3 + 4 = 7
```

Export einer Funktion

module-commonJSFunction.js

```
module.exports = function(a, b) {  
  return `${a} + ${b} = ${a + b}`;  
};
```

use-commonJSFunction.js

```
const add = require('./module-commonJSFunction');  
console.log(add(3, 4));
```

```
D:\Benutzer\OEM\node\K01NodeTest1>node use-commonJSFunction  
3 + 4 = 7
```

ECMAScript-Module

VORAUSSETZUNGEN:

- Datei package.json mit folgendem Inhalt:

```
{ "type": "module" }
```

- Beim Importieren muss *Dateinamenserweiterung* angegeben werden

module-ECMAScript.js

```
export const a = 3;  
export function add(a, b) {  
  return `${a} + ${b} = ${a + b}`;  
}  
export const user = {  
  name: 'Sepp',  
  password: 'seppentinen'  
}  
export class User {  
  constructor(name, password) {  
    this.name = name;  
    this.password = password;  
  }  
  doSomething() {  
    console.log(`${this.name} doesSomething`);  
  }  
}
```

use-ECMAScript.js

```
import { a, add, user as pureUser, User }  
  from './module-ECMAScript.js';  
console.log(a);  
console.log(add(a, 4));  
console.log(pureUser);  
const user = new User('Elfriede', 'rike');  
console.log(user.name);  
user.doSomething();
```

```
D:\Benutzer\OEM\node\K01NodeTest1>node use-ECMAScript.js  
(node:20568) ExperimentalWarning: The ESM module loader is experimental.  
3  
3 + 4 = 7  
{ name: 'Sepp', password: 'seppentinen' }  
Elfriede  
Elfriede doesSomething
```


module-ECMAScriptExpanded.js

```
function mul(a, b) {  
  return `${a} * ${b} = ${a * b}`;  
}  
  
function div(a, b) {  
  return `${a} / ${b} = ${a / b}`;  
}  
  
export * from './module-ECMAScript.js';  
export { mul as default, mul, div };
```

use-ECMAScriptExpanded.js

```
import { add, a, user as pureUser, User, mul }  
  from './module-ECMAScriptExpanded.js';  
import x from './module-ECMAScriptExpanded.js';  
import * as y from './module-ECMAScriptExpanded.js';  
console.log(x(1, 2));  
console.log(y.add(5, 6));  
console.log(y.mul(5, 6));  
console.log(y.div(5, 6));  
  
console.log(a);  
console.log(add(a, 4));  
console.log(mul(a, 4));  
console.log(div(a, 4)); // Fehler  
  
console.log(pureUser);  
const user = new User('Elfriede', 'rike');  
console.log(user.name);  
user.doSomething();
```

```
D:\Benutzer\OEM\node\K01NodeTest1>node use-ECMAScriptExpanded.js  
(node:3532) ExperimentalWarning: The ESM module loader is experimental.  
1 * 2 = 2  
5 + 6 = 11  
5 * 6 = 30  
5 / 6 = 0.8333333333333334  
3  
3 + 4 = 7  
3 * 4 = 12  
{ name: 'Sepp', password: 'seppentinen' }  
Elfriede  
Elfriede doesSomething
```

Express

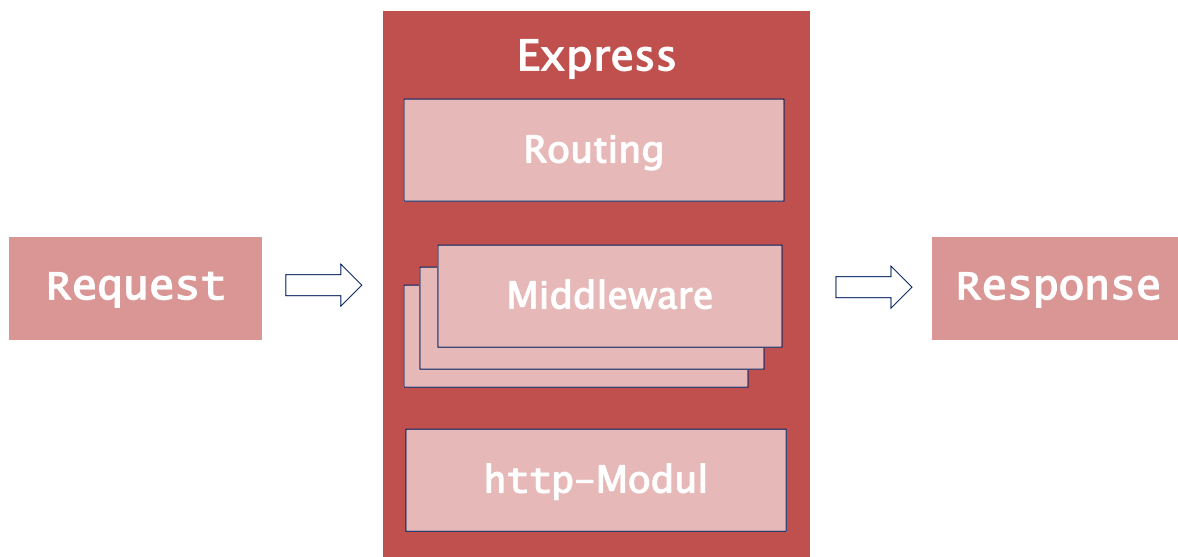
... ist ein Framework zur Erstellung zur Entwicklung von Webserver-Anwendungen



Löst Standardaufgaben wie

- Umgang mit Request, Auflösen von URLs
- Session-Management, Authentifizierung, Dateiuploads, usw.
- Open-Source-Projekt

Aufbau



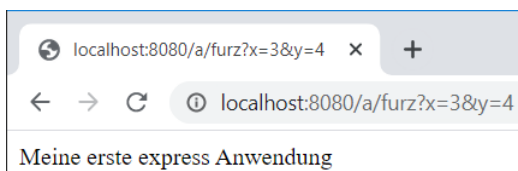
- Grundlage ist http-Modul von Node.js
- Über zur Verfügung gestelltes Request- und Response-Objekt kann auf Anfrage des Clients zurückgegriffen und Antwort manipuliert werden
- **Routing** legt fest welche Methode abhängig von der Http-Methode und URL den Request bearbeitet
- **Middleware**-Komponenten können bei Bedarf integriert werden und führen zusätzliche Aufgaben auf Request und Response aus (z.B. Cookiebehandlung, Zugriff auf POST-Parameter, Auslieferung von statischen Inhalten (CSS))

Installation und Initialisierung

```
$ npm init // fügt package.json ein
$ npm install express
```

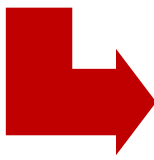
index.js

```
const express = require('express');
const app = express();
app.get('/a/:b', (request, response) => {
  console.log(request.protocol);
  console.log(request.method);
  console.log(request.originalUrl);
  console.log(request.path);
  console.log(request.params);
  console.log(request.params.b);
  console.log(request.query);
  console.log(request.query.x);
  response.send('Meine erste express Anwendung')9;
});
app.listen(8080, () => console.log('Server listen on port 8080'));
```



```
D:\Benutzer\OEM\node\K01NodeTest1>node index.js
Server listen on port 8080
http
GET
/a/furz?x=3&y=4
/a/furz
{ b: 'furz' }
furz
{ x: '3', y: '4' }
3
```

http://localhost:8080/a/furz?x=3&y=4

 path /a/furz
 params { b: 'furz' }
 query { x: '3', y: '4' }

params	query
in <i>Route</i> Parametername mit : gekennzeichnet	in <i>URL</i> mit ? eingeleitet, mit & getrennt und mit name=wert gesetzt
request.params.b	request.query.x request.query.y

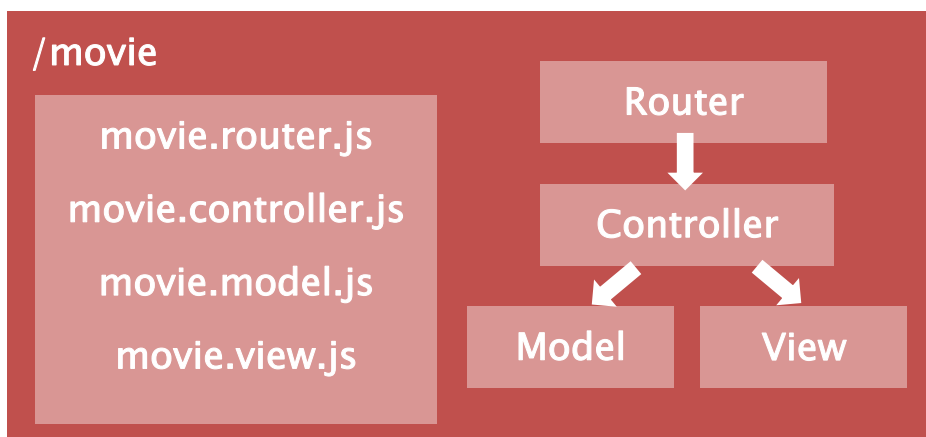
⁹ send() darf nicht wiederholt aufgerufen werden (Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client)

Struktur der Express–Anwendung – Das MVC–Pattern

M odel	<ul style="list-style-type: none">• Datenhaltung• Einfügen, Löschen, Ändern, Auflisten• Datenbankzugriffe
V iew	<ul style="list-style-type: none">• Darstellung• Mit Daten gefüllte HTML–Templates
C ontroller	<ul style="list-style-type: none">• Steuerungslogik• Verbindet Model und View

- Anwendung in fachliche/thematische Bereiche gegliedert (account, user, ...)
- Jeder Bereich als *Module* realisiert (mit Schnittstellen nach außen) und
- hat eigenen *Router*
- Komponententeile benannt mit `model`, `view` und `controller`

Beispiel



```
▼ K01NodeTest2
  ▼ account
    JS account.controller.js
    JS account.model.js
    JS account.router.js
    JS account.view.js
    JS login.controller.js
    JS login.model.js
    JS login.view.js
  > node_modules
  ▼ user
    JS group.controller.js
    JS group.model.js
    JS group.view.js
    JS user.controller.js
    JS user.model.js
    JS user.router.js
    JS user.view.js
  JS index.js
  {} package.json
  {} package-lock.json
  # style.css
```

index.js

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: false }));
const movieRouter = require('./movie/movie.router.js');
app.use('/movie', movieRouter);
app.get('/', (request, response) => response.redirect('/movie'));
app.use(express.static(__dirname));
app.listen(8080, () => console.log('Server listen on port 8080'));
```

- **Body-Parser-Middleware** ermöglicht auch das Verarbeiten von Daten im JSON-Format
- Muss vorher mit `npm install body-parser` installiert werden
- Request-Objekt wird um die Eigenschaft `body` erweitert um POST-Parameter von Formularen einfacher auslesen zu können
- Alle Routen die mit `/movie` beginnen werden über den Movie-Router abgewickelt
- Laden von statischen Inhalten wie CSS- oder JavaScript-Dateien

/movie/movie.router.js

```
const express = require('express');
const router = express.Router();
const { listAction, removeAction, editAction, saveAction } =
  require('./movie.controller')10;
router.get('/', listAction);
router.get('/remove/:id', removeAction);
router.get('/edit/:id?', editAction);
router.post('/save', saveAction);
module.exports = router;
```

- Verbindet Route mit Funktion
- Funktionen werden *Request* und *Response* als Parameter übergeben (siehe nächste Seite)
- Funktion soll immer mit `...Action()` benannt werden
- Optionaler Parameter mit `?` gekennzeichnet

¹⁰ Destructuring

/movie/movie.controller.js

```
const movieModel = require('./movie.model');
const movieView = require('./movie.view');

function listAction(request, response) {
  response.send(movieView.renderList(movieModel.getAll()));
}

function removeAction(request, response) {
  movieModel.remove(request.params.id);
  response.redirect(request.baseUrl)11;
}

function editAction(request, response) {
  let movie = { id: '-1', title: '', year: '' };
  if (request.params.id) {
    movie = movieModel.get(request.params.id);
  }
  response.send(movieView.renderMovie(movie));
}

function saveAction(request, response) {
  const movie = {
    id: request.body.id,12
    title: request.body.title,
    year: request.body.year
  };
  movieModel.save(movie);
  response.redirect(request.baseUrl);
}

module.exports =
{ listAction, removeAction, editAction, saveAction };
```

- Aktionsfunktionen bekommen Request- und Response-Objekt zur Verfügung gestellt

¹¹ Redirect nach /movie

¹² Wegen Body-Parser-Middleware

/movie/movie.model.js

```
const { v4: uuid } = require('uuid')13;
let data = [
  { id: uuid(), title: 'Iron Man', year: '2008' },
  { id: uuid(), title: 'Thor', year: '2011' },
  { id: uuid(), title: 'Captain America', year: '2001' }
];14
function getAll() {
  return data;
}
function remove(id) {
  data = data.filter(movie => movie.id !== id);
}
function get(id) {
  return data.find(movie => movie.id === id);
}
function save(movie) {
  if (movie.id === '-1') {
    movie.id = uuid();
    data.push(movie);
  } else {
    data = data.map(item => item.id === movie.id ? movie : item);
  }
}
module.exports = { getAll, remove, get, save };
```

¹³ Muss vorher mit `npm install uuid` installiert werden. Methode `v4()` wird durch `uuid()` aufrufbar gemacht

¹⁴ Im nächsten Kapitel lernen Sie, wie die Daten in einer MySQL-Datenbank verwaltet werden können

/movie/movie.view.js

```
function renderList(movies) {
  return `
    <!DOCTYPE html>
    <html lang="en">
      <head>
        <meta charset="UTF-8">
        <title>Filmliste</title>
        <link rel="stylesheet" href="/style.css">
      </head>
      <body>
        <table>
          <tr><th>Id</th><th>Titel</th><th>Jahr</th>
          <th></th><th></th></tr>
          ${movies.map(movie =>
            `
              <tr><td>${movie.id}</td>
              <td>${movie.title}</td><td>${movie.year}</td>
              <td><a href="/movie/remove/${movie.id}">Löschen</a>
              </td><td><a href="/movie/edit/${movie.id}">Ändern</a>
              </td></tr>`.join('')}
            `
          )}
        </table>
        <a href="/movie/edit">Neu</a>
      </body>
    </html>
  `;
}

function renderMovie(movie) {
  return `
    <!DOCTYPE html>
    <html lang="en">
      <head>
        <meta charset="UTF-8">
        <title>Filmliste</title>
        <link rel="stylesheet" href="/style.css">
      </head>
      <body>
        <form action="/movie/save" method="post">
          <input type="hidden" name="id" value="${movie.id}">
          <div>
            <label for="title">Titel:</label>
            <input type="text" id="title" name="title"
              value="${movie.title}">
          </div>
          ...
          <input type="submit" value="Speichern">
        </div>
      </form>
    </body>
  </html>
  `;
}

module.exports = { renderList, renderMovie };
```

Filmliste

localhost:8080/movie

Id	Titel	Jahr	
65aae129-c929-4bcf-8c7f-e6807363f236	Iron Man	2008	Löschen Ändern
e2f708b9-a1f3-4f01-b470-b9b34290f46b	Thor	2011	Löschen Ändern
75a3635a-73d1-403b-aa8a-f591a210db1b	Capitain America	2001	Löschen Ändern

[Neu](#)

Filmliste

localhost:8080/movie/edit/c1a1c0f5-6b56-46ab-a9d0-17283e144bcd

Titel:

Jahr:

Template-Engines (z.B. Pub, Handlebars) erlauben das komfortablere Generieren von HTML-Code

Imports und Exports langsam zum Mitschreiben...

index.js

```
const movieRouter = require('./movie/movie.router.js');
```



/movie/movie.router.js

```
module.exports = router;
```

```
const { listAction, removeAction, editAction, saveAction } =  
  require('./movie.controller');
```



/movie/movie.controller.js

```
module.exports = { listAction, removeAction, editAction, saveAction };
```

```
const movieModel = require('./movie.model');  
const movieview = require('./movie.view');
```



/movie/movie.model.js

```
module.exports = { getAll, remove, get, save };
```

/movie/movie.view.js

```
module.exports = { renderList, renderMovie };
```