

05 Ressourcen bereitstellen mit Node.js

Multimedia Engineering (PS siehe 36609b)

Nils Hellwig, M.Sc.

Lehrstuhl für Medieninformatik

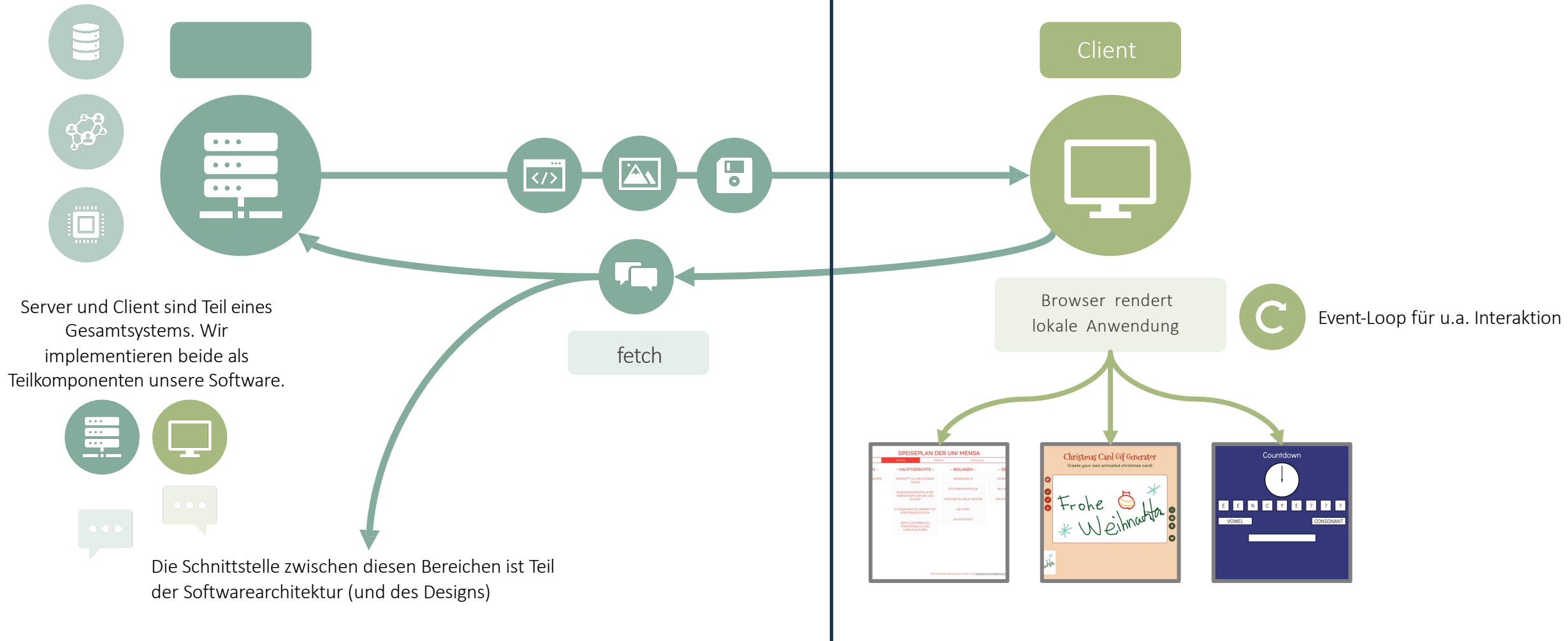
FAKULTÄT FÜR INFORMATIK UND DATA SCIENCE



Universität Regensburg

Überblick Themen

- Node.js / npm
- Express.js
- Entwicklung einer kleinen REST-API



Node.js Ökosystem

Laufzeitumgebung



JavaScript-Engine (Google V8)
API (für z.B. HTTP, Dateisystem, I/O, ...)
Tools für die Kommandozeile

Paketmanagement



Datenbank mit Bibliotheken (Paketen) für die
Entwicklung von Node.js- Anwendungen und
Werkzeuge Paketmanagement (und mehr)

Node.js Ökosystem

- **Ereignisgesteuert und asynchron:** Node.js ist auf einer Event-Loop-Technologie aufgebaut, die effizient mit vielen Verbindungen gleichzeitig umgehen kann.
- **JavaScript überall:** Sowohl im Frontend als auch im Backend kann die gleiche Programmiersprache verwendet werden.
- **Großes Ökosystem:** Über npm hat man Zugriff auf eine riesige Anzahl von Paketen mit Funktionen und Klassen

Node.js: Dein erstes Node.js-Skript

- Erstelle eine Datei app.js:

```
javascript
```

```
console.log("Hallo, Node.js!");
```

- Führe das Skript aus:

```
bash
```

```
node app.js
```

npm (Node Package Manager) - Voraussetzungen

- Haben wir Node.js installiert, so haben wir standardmäßig auch npm installiert.
- Überprüfe die Installation:

```
bash
```

```
node -v    # Zeigt die Node.js-Version an
```

```
npm -v     # Zeigt die npm-Version an
```

npm (Node Package Manager) – Ein neues Projekt erstellen

- npm init legt den Rahmen für ein neues Projekt fest und erstellt die zentrale Konfigurationsdatei package.json.
- Die package.json Datei enthält Metadaten sowie Abhängigkeiten (Packages) die geladen werden müssen.

```
bash
```

[Code kopieren](#)

```
mkdir mein-projekt  
cd mein-projekt  
npm init
```


npm (Node Package Manager) – Ein neues Projekt erstellen

package.json

```
json

{
  "name": "mein-projekt",
  "version": "1.0.0",
  "description": "Mein erstes npm-Projekt",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {},
  "devDependencies": {}
}
```

Häufige Befehle wie npm run build, npm run test oder npm start automatisieren z. B. den Build-Prozess, Tests oder den Start eines Servers.

Pakete, die **zur Laufzeit** des Programms benötigt werden

Pakete, die **nur zur Entwicklung** gebraucht werden (z. B. Test-Tools, Linter)

npm (Node Package Manager) – Ein neues Projekt erstellen

- Das Feld "main" in der package.json gibt an, welche Datei als Einstiegspunkt dient wenn das Node.js Projekt als Paket von einem anderen Paket genutzt wird.

```
json
{
  "name": "mein-modul",
  "version": "1.0.0",
  "main": "index.js"
}
```

npm (Node Package Manager) – Pakete installieren

- Pakete installieren:

```
bash  
  
npm install express
```

- Globale Pakete

```
bash  
  
npm install -g nodemon
```

npm (Node Package Manager) – Pakete installieren

- Pakete deinstallieren:

```
bash
```

```
npm uninstall express
```

npm (Node Package Manager) – Pakete installieren

- **node_modules** ist ein Verzeichnis, das von npm (Node Package Manager) erstellt wird, wenn man Abhängigkeiten für dein Node.js-Projekt installiert
- Es enthält alle externen Bibliotheken, die ein Projekt benötigt, um zu funktionieren.
- Diese Module werden durch die in der package.json-Datei definierten Abhängigkeiten festgelegt.

Module laden: require() vs. import

Merkmal	require() – alter Standard	import – moderner Standard
Import Syntax	<code>const fs = require('fs');</code>	<code>import fs from 'fs';</code> (Nur möglich falls <code>"type": "module"</code> in <code>packages.json</code>)
Export Syntax	Nutzt die globale Variable <code>module</code> um Klassen/Funktionen/Variablen zu exportieren: <code>module.exports = ...</code>	<code>export default</code> oder <code>export {...}</code>

```
{
  id: '.',
  path: '/mein/projekt',
  filename: '/mein/projekt/mathe.js',
  loaded: false,
  exports: {}, // ← Dieses Objekt wird mit
  children: [],
  ...
}
```

js

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

js

```
const { add, subtract } = require('./math');  
console.log(add(5, 3)); // 8  
console.log(subtract(5, 3)); // 2
```

1000+ packages found


Sort by: Default ▾

react-dropdown

React dropdown component

[react](#) [react-component](#) [component](#) [dropdown](#) [select](#)

📄 266.216

 **fraserxu** • 1.11.0 • 3 years ago • 📦 153 dependents • 📄 MIT**@redocly/react-dropdown-aria**

Simple and accessible React dropdown component

[react](#) [dropdown](#) [select](#) [aria](#) [react-dropdown](#) [react-dropdown-aria](#)

📄 91.139

 **romanhotsiy** • 2.0.12 • 4 years ago • 📦 16 dependents • 📄 MIT**@reach/dropdown**

React dropdown menu.

 **chancestrickland** • 0.18.0 • 3 years ago • 📦 2 dependents • 📄 MIT


📄 351.065

@trendmicro/react-dropdown

React Dropdown component

[dropdown](#) [react](#) [react-dropdown](#)


📄 16.794

 **cheton** • 1.4.0 • 5 years ago • 📦 7 dependents • 📄 MIT**react-dropdown-aria**

Simple and accessible React dropdown component

[react](#) [dropdown](#) [select](#) [aria](#) [react-dropdown](#) [react-dropdown-aria](#)

📄 11.705

 **jfangrad** • 3.0.0 • 4 years ago • 📦 2 dependents • 📄 MIT**react-dropdown-menu**

React dropdown menu

 **tommy351** • 0.0.2 • 10 years ago • 📦 4 dependents • 📄 MIT

📄 5.186

Externe Node.js Packages

Externe Node.js Packages: Axios

```
import axios from 'axios';

axios.get('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => {
    console.log('Daten:', response.data);
  })
  .catch(error => {
    console.error('Fehler:', error);
  });
```

Externe Node.js Packages: Nodemon

- Nodemon hilft dabei, den Entwicklungsprozess zu beschleunigen, indem es den Server automatisch neu startet, sobald Änderungen an den Quelldateien vorgenommen werden.
- So muss der Server nicht manuell gestoppt und neu gestartet werden, jedes Mal, wenn man Änderungen vornimmst.

Externe Node.js Packages: Nodemon

- Du kannst Nodemon global oder lokal installieren, im Beispiel global:

```
bash
```

[Code kopieren](#)

```
npm install -g nodemon
```

- Statt `node server.js` führen wir `nodemon server.js` aus.

Backend mit Express.js

- Express ist ein leichtgewichtiges Web-Framework für Node.js, das das Erstellen von Webservern und APIs stark vereinfacht.
- Es bietet eine einfache Schnittstelle zum Umgang mit Routen, Middleware etc.
- Es basiert vollständig auf dem HTTP-Protokoll – jede Route in Express reagiert auf eine HTTP-Anfrage.

Backend mit Express.js: Grundstruktur

- `import express from 'express';`: Lädt das Express-Framework, das den Aufbau von Webservern vereinfacht.
- `const app = express();`: Erstellt eine Express-App Instanz – das zentrale Objekt, mit dem du Middleware, Routen und Serverfunktionen definierst.
- `app.get('/users', () => {...});`: Definiert eine Route – Express prüft URL und Methode und ruft bei Übereinstimmung die Handler-Funktion auf.
- `app.listen(PORT);`: Startet den Server und lässt ihn auf einem bestimmten Port auf eingehende HTTP-Anfragen warten.

Backend mit Express.js: Grundstruktur - Beispiel

```
// ⓘ Liste von Benutzern abrufen (GET)
app.get("/users", (req, res) => {
  const users = [
    { id: 1, name: "Alice" },
    { id: 2, name: "Bob" },
  ];
  res.json(users);
});
```

```
// ⓘ Neuen Benutzer hinzufügen (POST)
app.post("/users", (req, res) => {
  const { name } = req.body;
  if (!name) {
    return res.status(400).json({ error: "Name fehlt" });
  }
  res.json({ id: 3, name }); // Beispiellantwort
});
```

HTTP – Hypertext Transfer Protocol

- **Definition:** HTTP ist ein Kommunikationsprotokoll, das es ermöglicht, Daten über das Internet zu übertragen.
- **Verwendung:** Es wird hauptsächlich für den Austausch von Webseiten und deren Ressourcen zwischen Client und Server verwendet.

HTTP-Requests

- **HTTP-Request** sind Nachrichten, die von einem Client an einen Server gesendet werden, um Daten anzufordern oder zu senden.
- **HTTP-Methoden** definieren, welche Art von Aktion der Request ausführt.

HTTP-Methoden

- GET – Ruft Daten vom Server ab, ohne etwas zu verändern
- POST – Sendet neue Daten an den Server, z. B. zum Erstellen von Ressourcen
- UPDATE – Verändert eine vorhandene Ressource
- DELETE – Löscht eine Ressource vom Server

HTTP Status Codes – 2xx (Erfolgreich)

- **200 OK:** Die Anfrage war erfolgreich und die Antwort enthält die angeforderten Daten.
- **201 Created:** Die Anfrage war erfolgreich und ein neues Element wurde erstellt (z.B. beim Erstellen eines neuen Datensatzes).
- **202 Accepted:** Die Anfrage wurde akzeptiert, aber die Verarbeitung ist noch nicht abgeschlossen.
- **204 No Content:** Die Anfrage war erfolgreich, aber es gibt keine Rückgabe (z. B. bei einer erfolgreichen DELETE-Anfrage).
- **206 Partial Content:** Der Server sendet nur einen Teil der angeforderten Ressource (z.B. bei einem Download).

HTTP Status Codes – 4xx (Clientfehler)

- **400 Bad Request:** Die Anfrage war ungültig oder schlecht formatiert.
- **401 Unauthorized:** Der Client muss sich authentifizieren, um auf die Ressource zugreifen zu können.
- **403 Forbidden:** Der Client hat keine Berechtigung, auf die angeforderte Ressource zuzugreifen.
- **404 Not Found:** Die angeforderte Ressource wurde auf dem Server nicht gefunden.
- **405 Method Not Allowed:** Die angeforderte HTTP-Methode ist für die angeforderte Ressource nicht erlaubt (z. B. ein GET auf eine POST-only-Ressource).

HTTP Status Codes – 5xx (Serverfehler)

- **500 Internal Server Error:** Ein allgemeiner Serverfehler ist aufgetreten.
- **501 Not Implemented:** Der Server unterstützt die angeforderte Methode nicht.
- **502 Bad Gateway:** Der Server hat von einem anderen Server eine ungültige Antwort erhalten (z.B. bei einem Proxy oder Gateway).
- **503 Service Unavailable:** Der Server ist vorübergehend nicht verfügbar (z. B. wegen Überlastung oder Wartungsarbeiten).
- **504 Gateway Timeout:** Der Server hat bei der Kommunikation mit einem anderen Server eine Zeitüberschreitung erfahren.
- **505 HTTP Version Not Supported:** Der Server unterstützt die angeforderte HTTP-Version nicht.

HTTP Status Codes

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

REST (Representational State Transfer): Architekturstil

- **Representational:** Darstellung oder Repräsentation, bezogen auf die Daten oder Ressourcen, die der Server an den Client sendet (z. B. JSON, XML, HTML) = standardisiertes Format, unabhängig von JavaScript
- **State:** Zustand, der zwischen Client und Server übertragen wird, um eine Aktion oder Anfrage zu ermöglichen (ohne, dass der Server den Zustand speichert).
- **Transfer:** Übertragung dieser Darstellung und Zustandsinformationen über das Netzwerk, üblicherweise via HTTP.

HTTP-Method: GET

- Wird verwendet, um **Daten vom Server abzurufen**.

javascript

```
app.get('/api/message', (req, res) => {  
  res.json({ message: "Hallo aus Node.js!" });  
});
```

HTTP-Method: GET

Request-Objekt (req): Repräsentiert die eingehende HTTP-Anfrage des Clients an den Server.

- Enthält viele nützliche Informationen wie:

`req.method` — HTTP-Methode (GET, POST, etc.)

`req.url` — angefragte URL

`req.headers` — HTTP-Header als Objekt

`req.params` — Route-Parameter (z.B. bei `/user/:id`)

`req.query` — Query-Parameter (z.B. `?search=abc`)

`req.body` — Körper der Anfrage (z.B. JSON-Daten bei POST, wenn Middleware wie `express.json()` verwendet wird)

`req.cookies` — Cookies (wenn Middleware wie `cookie-parser` installiert ist)

HTTP-Method: GET

Response-Objekt (res): Repräsentiert die HTTP-Antwort, die der Server an den Client zurückschickt.

- Bietet Methoden zum Senden von Daten und Steuerung der Antwort:

`res.send()` – sendet eine Antwort (Text, Buffer, JSON)

`res.json()` – sendet JSON-Daten (automatisch Header Content-Type: application/json)

`res.status()` – setzt den HTTP-Statuscode (z.B. 200, 404)

`res.redirect()` – leitet den Client auf eine andere URL weiter

`res.set()` – setzt HTTP-Header in der Antwort

`res.end()` – beendet die Antwort (ohne Daten senden, 200) [`res.send()` und `res.json()` rufen intern auch `res.end()` auf], danach kann keine Manipulation am Response mehr vorgenommen werden.

HTTP-Method : POST

- Wird verwendet, um **Daten an den Server zu senden** (z.B. JSON-Objekt).

javascript

```
app.post('/api/message', (req, res) => {  
  const data = req.body;  
  res.json({ status: "Daten empfangen", receivedData: data });  
});
```

HTTP-Method : PUT

- Wird verwendet, um **Daten auf dem Server zu aktualisieren**

javascript

```
app.put('/api/user/:id', (req, res) => {  
  const userId = req.params.id;  
  const updatedUser = req.body;  
  res.json({ status: "Benutzerdaten aktualisiert", userId: userId,}  
});
```

HTTP-Method : DELETE

- Wird verwendet, um eine **Ressource vom Server zu löschen**.

javascript

```
app.delete('/api/user/:id', (req, res) => {  
  const userId = req.params.id;  
  res.json({ status: "Benutzer gelöscht", userId: userId });  
});
```

Express: Query Parameter

- Query Parameter sind zusätzliche Informationen, die an die URL einer Anfrage angehängt werden.

```
bash
```

```
http://localhost:3000/api/products?category=electronics&price=100
```

Express: Query Parameter

- In Express kannst du Query Parameter mit req.query abrufen.

javascript

```
app.get("/api/products", (req, res) => {  
  const category = req.query.category; // Abfrage des 'category' Parameters  
  const price = req.query.price;      // Abfrage des 'price' Parameters  
  
  res.json({  
    category: category,  
    price: price  
  });  
});
```

HTTP Header

- HTTP-Header ermöglichen die Kommunikation von Metadaten und Einstellungen zwischen Client und Server.
- Beispiel für einen HTTP-Header bei einem Request des Clients

```
makefile
```

```
GET /api/user HTTP/1.1  
Host: example.com  
Authorization: Bearer <token>  
Content-Type: application/json
```

HTTP Header

- Beispiel für einen HTTP-Header in der Response (Antwort)

yaml

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json
```

Content-Type: Gibt den Medientyp der Antwort an (z.B. application/json)

Tool:
<https://www.postman.com/>

The screenshot displays the Postman web interface. At the top, there's a navigation bar with 'Overview' selected. Below it, the URL bar shows a GET request to `https://raw.githubusercontent.com/lsakZhang/ABSA-QUAD/refs/heads/mast...`. The 'Headers' tab is active, showing a table with 7 hidden headers. The response section at the bottom shows a 200 OK status, 275 ms response time, and 51.68 KB body size. The response headers table is visible below.

Key	Value	Description
Key	Value	Description

Key	Value
Connection	keep-alive
Content-Length	51992
Cache-Control	max-age=300
Content-Security-Policy	default-src 'none'; style-src 'unsafe-inline'; sandbox
Content-Type	text/plain; charset=utf-8
ETag	W/"0ac3cc5d5851a4800b7c65fe12eaa17b82493d9b...
Strict-Transport-Security	max-age=31536000

Middlewares (in Express.js)

- Eine Middleware ist eine Funktion, die bei jeder HTTP-Anfrage vor der eigentlichen Antwort-Logik ausgeführt wird.
- Sie dient dazu, **die Anfrage zu analysieren, zu verändern, abzufangen oder weiterzugeben**.
- Middlewares können Aufgaben wie Logging, Authentifizierung, Validierung oder Datenmanipulation übernehmen.

- Beispiel:

```
app.use((req, res, next) => {  
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);  
  next(); // Weiter zur nächsten Middleware oder Route  
});
```

Middlewares (in Express.js) – req-Objekt

```
app.use((req, res, next) => {  
  // ... code  
  next();  
});
```

- `req.method` – z. B. "GET", "POST", "DELETE"
- `req.url` – die Pfadangabe der Anfrage, z. B. "/api/users"
- `req.headers` – alle mitgesendeten HTTP-Header
- `req.body` – der Request-Body (nur wenn `express.json()` verwendet wird)
- `req.query` – Query-Parameter wie in `/search?q=express`

```
app.use((req, res, next) => {  
    // ... code  
    next();  
});
```

Middlewares (in Express.js) – res-Objekt

- `res.send()` – sendet einfache Texte oder Objekte
- `res.json()` – sendet JSON (setzt automatisch Header)
- `res.status()` – setzt den HTTP-Statuscode (z.B. 200, 404)
- `res.set()` – setzt manuell Header
- Es ist möglich, in einer Middleware bereits eine Antwort senden!

```
app.use((req, res, next) => {  
    if (!req.headers.authorization) return res.status(401).send('Nicht autorisiert');  
    next(); // Nur wenn autorisiert  
});
```

```
app.use((req, res, next) => {  
  // ... code  
  next();  
});
```

Middlewares (in Express.js) – next-Objekt

- `next()` ist eine Funktion, mit man Express mitteilt, dass es zur nächsten passenden Middleware oder Route weitergehen soll.
- Wenn du `next()` nicht aufrufst, endet die Anfrage dort.

Externe Middlewares: CORS Policy

- Ein prominentes Beispiel ist die cors-Middleware des „cors“-Packages, die den Umgang mit CORS im Browser übernimmt.
- Was ist CORS: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS>

Externe Middlewares: `express.json()`

- `app.use(express.json());` wandelt den rohen HTTP-Request-Body (nur bei Content-Type: application/json) in ein JavaScript-Objekt um und hängt es an `req.body` an.

Node.js als Webserver: Express.js

Demo

Literatur

- <https://www.w3schools.com/nodejs/>
- Technical Details & more: <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>