

Backend Entwicklung mit Nodejs



Übersicht

- Grundlagen zu Backend Entwicklung
- Grundlagen zu Nodejs
- Node Package Manager NPM



Was ist eine Web Applikation?

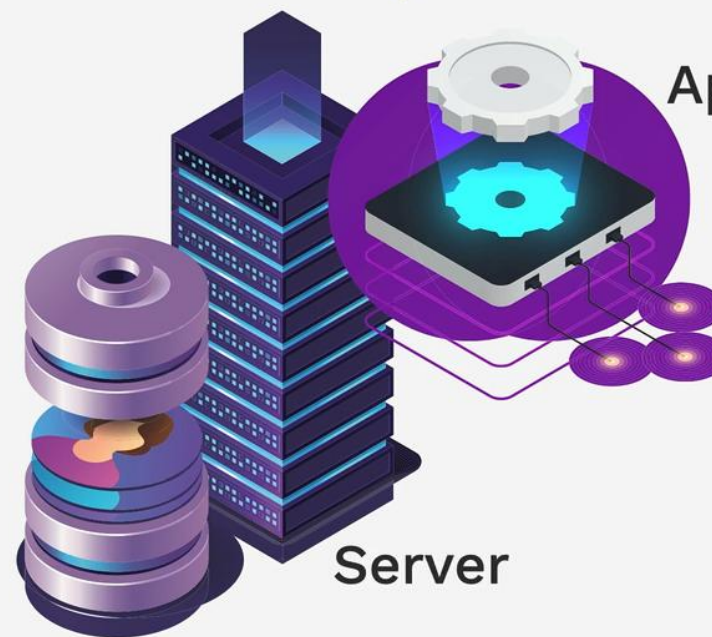
Client-Side

Client



Server-Side

Application



Server

Database

Frontend



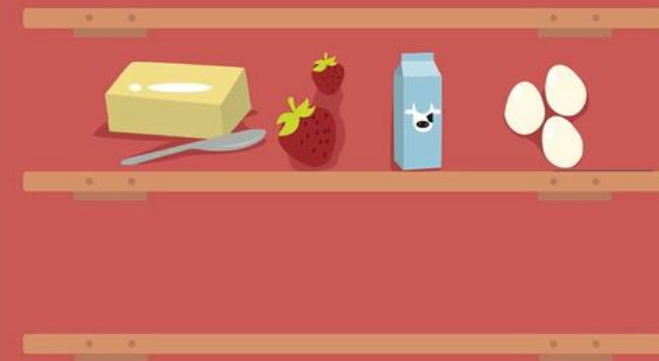
Restaurant

Backend



Kitchen

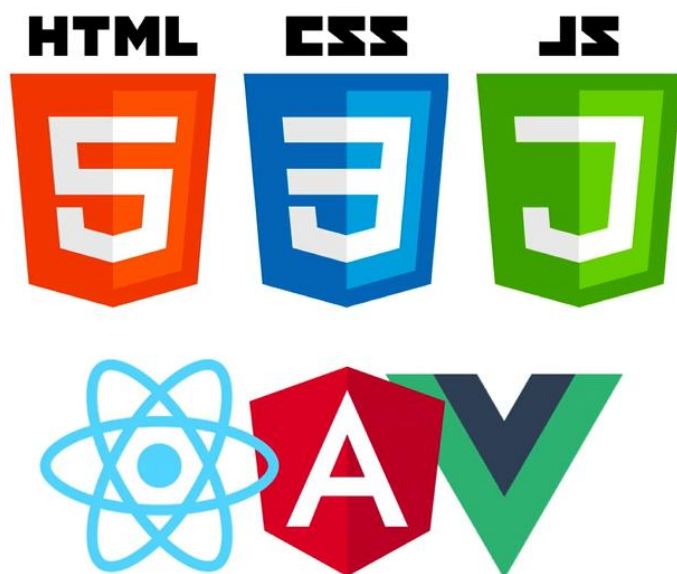
Datenbank



Pantry

Frontend - Backend Technologien

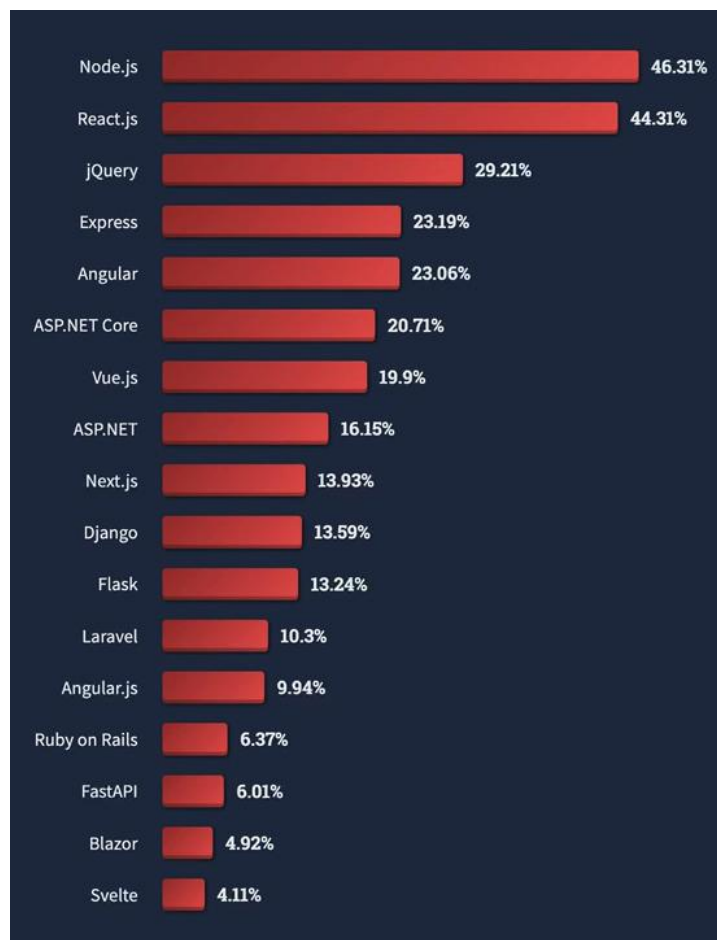
Front-end



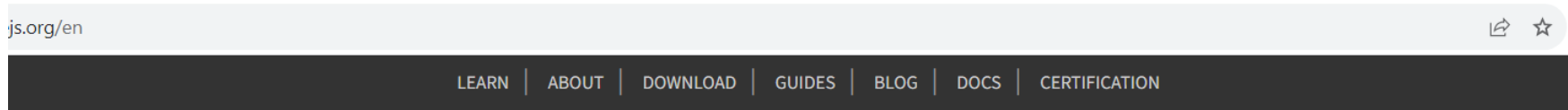
Back-end



Technologien im Trend



Download Nodejs



Node.js® is an open-source, cross-platform JavaScript runtime environment.

Download Node.js®

20.10.0 LTS

Recommended For Most Users

21.4.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

For information about supported releases, see the [release schedule](#).

- <https://nodejs.org/en>
- PC neu starten

Was ist Nodejs?

- *As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications*
- Javascript Runtime: Ermöglicht es Javascript außerhalb vom Browser laufen zu lassen → Serverseitige Programmierung
- Asynchronous and event-driven:
Ermöglicht eine effiziente parallele Verarbeitung ohne abgegebenen Requests gegenseitig zu blockieren



Vorteile von Nodejs



JS Fullstack



Scales



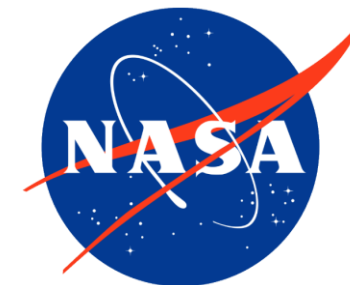
Non-blocking



Ecosystem

Wer nutzt Nodejs?

●● Medium



ebay



Node im Terminal ausführen

- Jeder Javascript Befehl kann mit Hilfe von Node in der Konsole bzw. dem VS Code Terminal ausgeführt werden
- Um Javascript auszuführen muss zuvor der Befehl node ausgeführt werden
- Um eine Datei auszuführen wird der Befehl node + Dateiname eingegeben (Bsp: node ./index.js)

```
▼ TERMINAL node + ▾ 🗑  
○ PS C:\xampp\htdocs\CodersBay\Webentwicklung\Webentwicklung_23_24> node  
Welcome to Node.js v18.14.0.  
Type ".help" for more information.  
> let a = 3  
undefined  
> a+5  
8  
> |
```

Node Native Modules

- Mit Node.js out-of-the-box mitgelieferte Module / Funktionen
- Doku dazu: <https://nodejs.org/docs/latest/api/>



Node Übung Filesystem

- Das Native Modul Filesystem dient dazu, Dateien über Node zu erstellen, auszulesen, bzw. anzupassen
- Gemeinsam:
Erstelle eine neue Text-Datei „message.txt“ mit Hilfe des fs Moduls
- Übung für euch unter
0. Eure Codebeispiele/Name/Node:
Mach dich mit der Node Doku vertraut und schreibe ein Node Programm um die zuvor erstellt Text Datei auszulesen und in der Konsole auszugeben
- Erweitere die Datei um folgenden Text: „\nHallo von [Dein Name]“
Tipp: **append** text to a **File**

```
const fs = require('fs');
let message = "Hallo von Dominic"

fs.writeFile('message.txt', message, (err) => {
  if (err) throw err;
  console.log('The file has been saved!');
});
```

```
Hallo von Dominic
Hallo von Mario
Hallo von Marwa
Hallo von Matthias
Hallo von Wolfgang
```

NPM Node Package Manager

- NPM kommt out-of-the-box mit der Installation von node
- Stellt eine unglaublich nützliche Sammlung an zahlreichen JavaScript Komponenten (Packages) zur Verfügung
- >2000000 Packages
- Größte Software Bibliothek der Welt



<https://www.npmjs.com/>

NPM Projekt starten

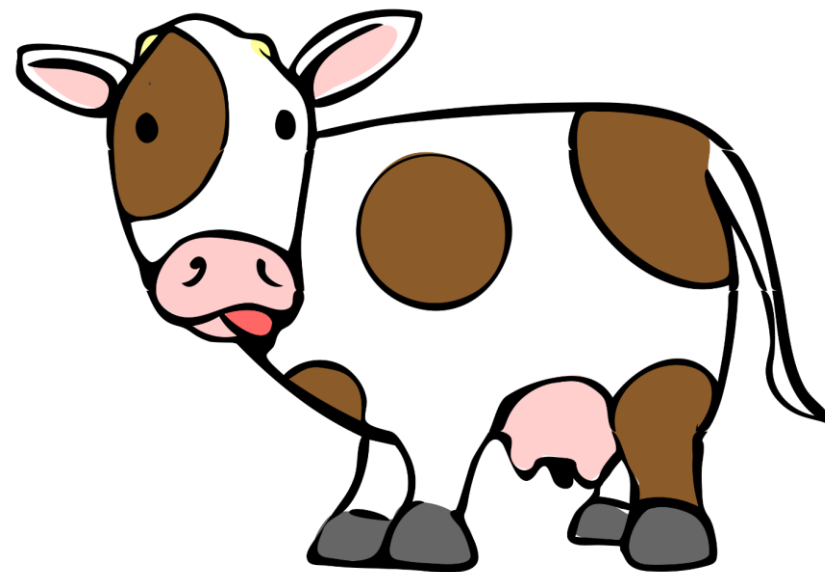
- Im VS-Code Terminal: Navigiere zum Ordner (0. Eure Codebeispiele/Node) indem du ein neues NPM Projekt starten möchtest
- Mit dem Befehl `npm init` wird ein neues Projekt gestartet
 - Beantworte die Fragen die gestellt werden
 - Ein `package.json` (Config-)file wird erstellt



<https://www.npmjs.com/>

NPM Gruppenübung „Cowsay“

- Gemeinsam schreiben wir mit Hilfe von npm ein Programm das eine Kuh zum sprechen bringt
- Modul „cowsay“ in npm library suchen und finden
- `npm install cowsay` (kurzform `npm i cowsay`)
- Lass die Kuh etwas sagen
- Lass die Kuh etwas denken



NPM Übungen Superhelden und Fibonacci 2.0

1. Nutze das npm Modul „superheroes“ um einen zufälligen Namen eines Superhelden zu generieren und in der Konsole auszugeben
2. (Optional) Nutze das npm Modul „fibonacci“ um die ersten 10 fibonacci Zahlen zu loggen



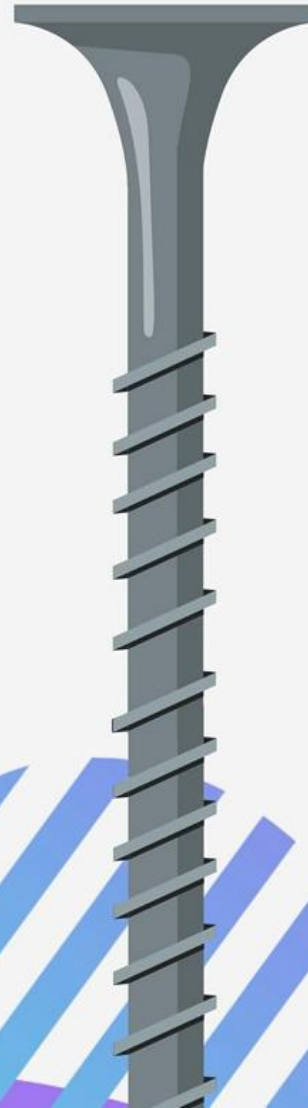
Nodejs + Express



node



Express



Was ist Express?

- Express ist ein beliebtes, leichtgewichtiges und flexibles Webanwendungs-Framework für Node.js.
- Es erleichtert die Erstellung von Webanwendungen und APIs durch Bereitstellung einer Reihe von Funktionen und Werkzeugen, um die Entwicklung zu beschleunigen
- Häufige Anwendungsfälle (werden wir alle kennen lernen):
 - **Routing:** Express ermöglicht das Definieren von Routen, um Anforderungen an bestimmte Endpunkte (URLs) zu verknüpfen
 - **Middleware:** Express verwendet Middleware, um Anforderungen zwischen dem Eintreffen und der Verarbeitung durch die Route zu verarbeiten
 - **Vorlagen (Templates):** Express unterstützt verschiedene Vorlagen-Engines (EJS), um dynamische HTML-Inhalte zu generieren.



Webserver mit Nodejs + Express

```

index.js — Web Development Projects
JS index.js x
Backend > 3.0 Intro to Express > JS index.js > ...
1 import http from "http";
2 import url from "url";
3
4 const server = http.createServer((req, res) => {
5   const parsedUrl = url.parse(req.url, true);
6
7   if (parsedUrl.pathname === "/" && req.method === "GET") {
8     res.writeHead(200, { "Content-Type": "text/html" });
9     res.end("<h1>Welcome to the homepage!</h1>");
10  } else if (parsedUrl.pathname === "/about" && req.method === "GET") {
11    res.writeHead(200, { "Content-Type": "text/html" });
12    res.end("<h1>About us</h1>");
13  } else {
14    res.writeHead(404, { "Content-Type": "text/html" });
15    res.end("<h1>Page not found</h1>");
16  }
17 });
18
19 server.listen(3000, () => {
20   console.log("Server running at http://localhost:3000/");
21 });
22

```


```

index.js — Web Development Projects
JS index.js x
Backend > 3.0 Intro to Express > JS index.js > ...
1 import express from "express";
2 const app = express();
3
4 app.get("/", (req, res) => {
5   res.send("<h1>Welcome to the homepage!</h1>");
6 });
7
8 app.get("/about", (req, res) => {
9   res.send("<h1>About us</h1>");
10 });
11
12 app.use((req, res) => {
13   res.status(404).send("<h1>Page not found</h1>");
14 });
15
16 app.listen(3000, () => {
17   console.log("Server running at http://localhost:3000/");
18 });
19
20
21
22


```

nodemon

3.0.2 • [Public](#) • Published 11 days ago

 [Readme](#)

 [Code](#) [Beta](#)

 [10 Dependencies](#)



nodemon

nodemon is a tool that helps develop Node.js based applications by automatically restarting the node application when file changes in the directory are detected.

Übung dein erster Webserver

- Schreibe deinen ersten eigenen Webserver der auf gewisse Endpoints reagiert (Beispiel „./home“)
- einen Status zurückliefert
- eine Antwort liefert in Form von minimalem HTML Bsp: `<h1>Homepage</h1>`
- Starte deinen Webserver mit der Hilfe von nodemon

```
import express from "express";
const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.send("<h1>Home Page</h1>");
});

app.post("/register", (req, res) => {
  //Do something with the data
  res.sendStatus(201);
});

app.put("/user/angela", (req, res) => {
  res.sendStatus(200);
});

app.patch("/user/angela", (req, res) => {
  res.sendStatus(200);
});

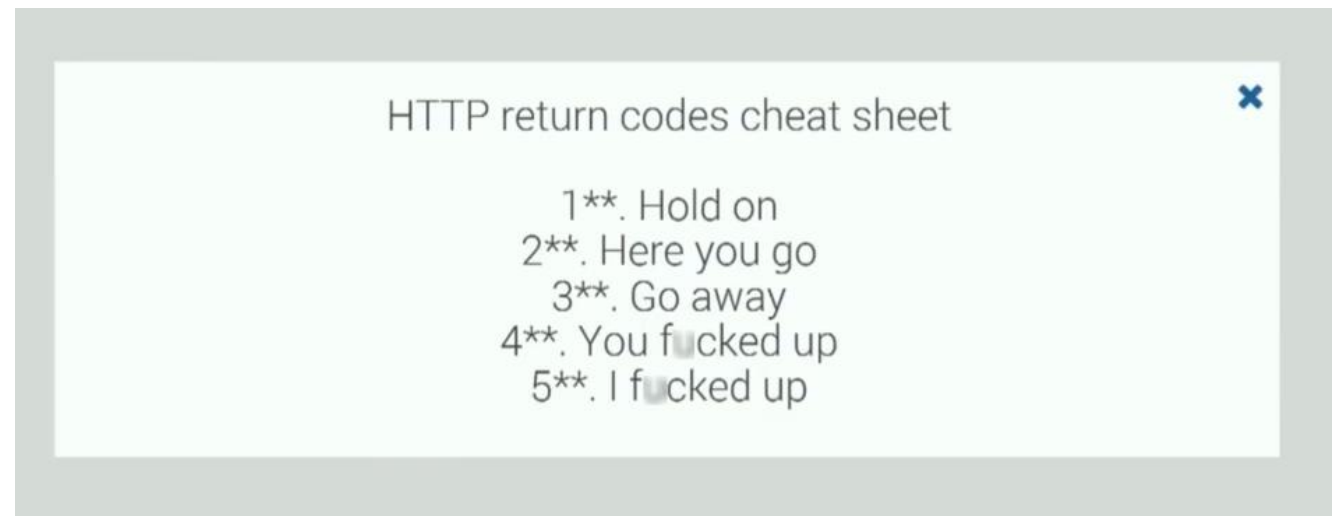
app.delete("/user/angela", (req, res) => {
  //Deleting
  res.sendStatus(200);
});

app.listen(port, () => {
  console.log(`Server started on port ${port}`);
});
```

HTTP Request Status

HTTP response status codes indicate whether a specific [HTTP](#) request has been successfully completed. Responses are grouped in five classes:

1. [Informational responses](#) (100 – 199)
2. [Successful responses](#) (200 – 299)
3. [Redirection messages](#) (300 – 399)
4. [Client error responses](#) (400 – 499)
5. [Server error responses](#) (500 – 599)



The status codes listed below are defined by [RFC 9110](#) .

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

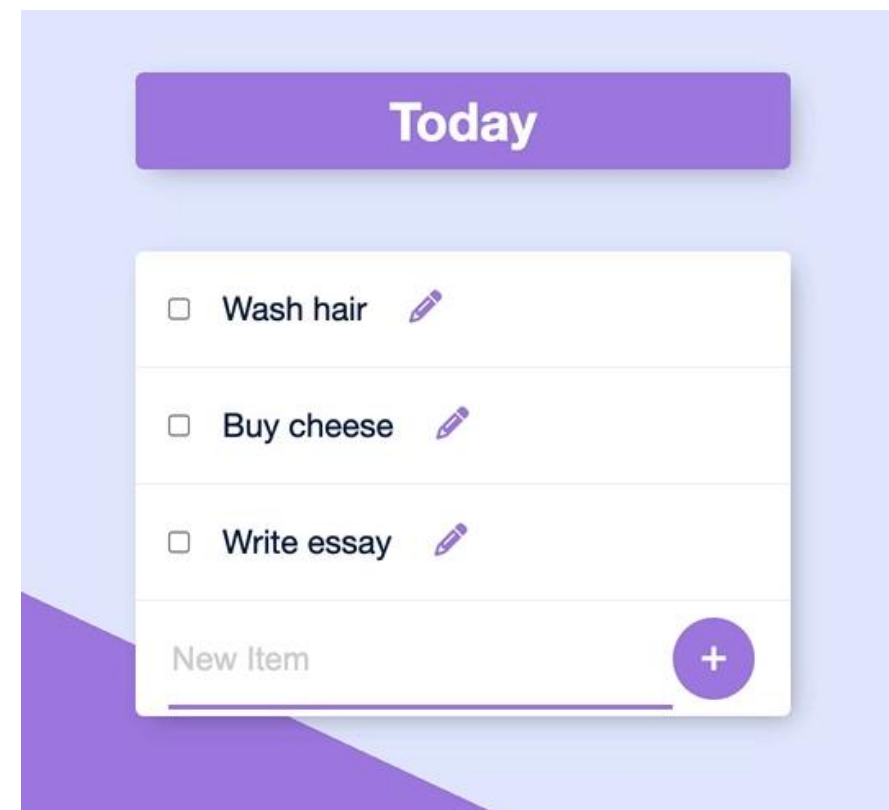
Postman

- Mit Hilfe von Postman können HTTP Requests abgegeben werden ohne ein aufwändiges Frontend bauen zu müssen
- Erstelle einen Postman Account
- Download Postman Desktop um localhost endpoints testen zu können



Vorbereitung auf Full-Stack Projekt To-Do Liste

- Zielbild: Webapplikation für eine einfache To-Do Liste wobei Daten dafür in einer mySQL Datenbank liegen. Ein vorgeschalteter Login (Username + Passwort) dient zur Autorisierung
- 1. Frontend mit HTML, CSS, Javascript aufbauen mit Testdaten (Hardcodiert Username + Passwort) und temporäre Änderungsfunktionen (Refresh der Seite setzt alle Änderungen wieder zurück)
- 2. Daten werden aus einer mySQL Datenbank geladen und Änderungen in der Datenbank persistiert



Files als Response schicken

- Um ganze HTML Dateien als Antwort zu senden werden folgende Pakete benötigt
 - `Import {dirname} from „path“`
 - `Import {fileURLToPath} from „url“`
 - Tipp: die Pakete sind bereits im package.json File konfiguriert, um sie nutzen zu können führe den Befehl „npm install“ im VS Code Terminal aus
- Um die BaseURL der verwendeten Datei zu erhalten verwende folgende Variable:
 - `const __dirname = dirname(fileURLToPath(import.meta.url));`
- Um bei einem Endpoint eine Datei als Antwort zu senden verwende die Methode
 - `res.sendFile(__dirname + „/Pfad zu deinem HTML File“)`

```
import express from "express";
import { dirname } from "path";
import { fileURLToPath } from "url";
const __dirname = dirname(fileURLToPath(import.meta.url));

const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.sendFile(__dirname + "/public/index.html");
});
```

Files als Response außerhalb des Parentfolders

- Möchten wir eine Datei als Antwort schicken, die sich außerhalb des Ordners (Ebene höher) in der die Server Datei abgelegt ist, befindet benötigt es folgende Schritte
- `import path from "path"`
 - Pfad Package mit der Datei und Ordnerpfade verwendet werden können
- `Res.sendFile(path.join(__dirname + "../frontend/index.html"))`
 - `path.join` verknüpft den absoluten Pfad der Server Datei mit einem relativen Pfad der auszuliefernden Datei
- `app.use(express.static(„Relativer Pfad zu Statischen Files“))`
 - Um statische Files (wie HTML, CSS, Bilder usw.) auszuliefern die durch relativen Pfad in einem File eingebettet sind welches als Antwort geschickt wird.
 - Angenommen, du hast eine Datei namens „styles.css“ im angegebenen Verzeichnis. Wenn jemand auf `http://localhost:3000/styles.css` zugreift, wird Express automatisch die Datei aus diesem Verzeichnis aufrufen.

```
import express from "express";
import bodyParser from "body-parser";
import mysql from "mysql"
import path from "path"
import { dirname } from "path";
import { fileURLToPath } from "url";

const app = express();
const port = 3000;
const __dirname = dirname(fileURLToPath(import.meta.url));

app.use(express.static("PFAD ZU STATISCHEN DATEIEN"));

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
```

Übung Webserver mit Files als Response

- Erstelle einen Webserver mit einem GET Endpoint „/“ der ein HTML File zurückliefert (Ressource Node/Middleware/public/index.html)
- Um ganze HTML Dateien als Antwort zu senden werden folgende Pakete benötigt
 - Import {dirname} from „path“
 - Import {fileURLToPath} from „url“
 - Tipp: die Pakete sind bereits im package.json File konfiguriert, um sie nutzen zu können führe den Befehl „npm install“ im VS Code Terminal aus
- Um die BaseURL der verwendeten Datei zu erhalten verwende folgende Variable:
 - `const __dirname = dirname(fileURLToPath(import.meta.url));`
- Um bei einem Endpoint eine Datei als Antwort zu senden verwende die Methode
 - `res.sendFile(__dirname + „/Pfad zu deinem HTML File“)`

```
import express from "express";
import bodyParser from "body-parser";
import mysql from "mysql"
import path from "path"
import { dirname } from "path";
import { fileURLToPath } from "url";

const app = express();
const port = 3000;
const __dirname = dirname(fileURLToPath(import.meta.url));

app.use(express.static("PFAD ZU STATISCHEN DATEIEN"));

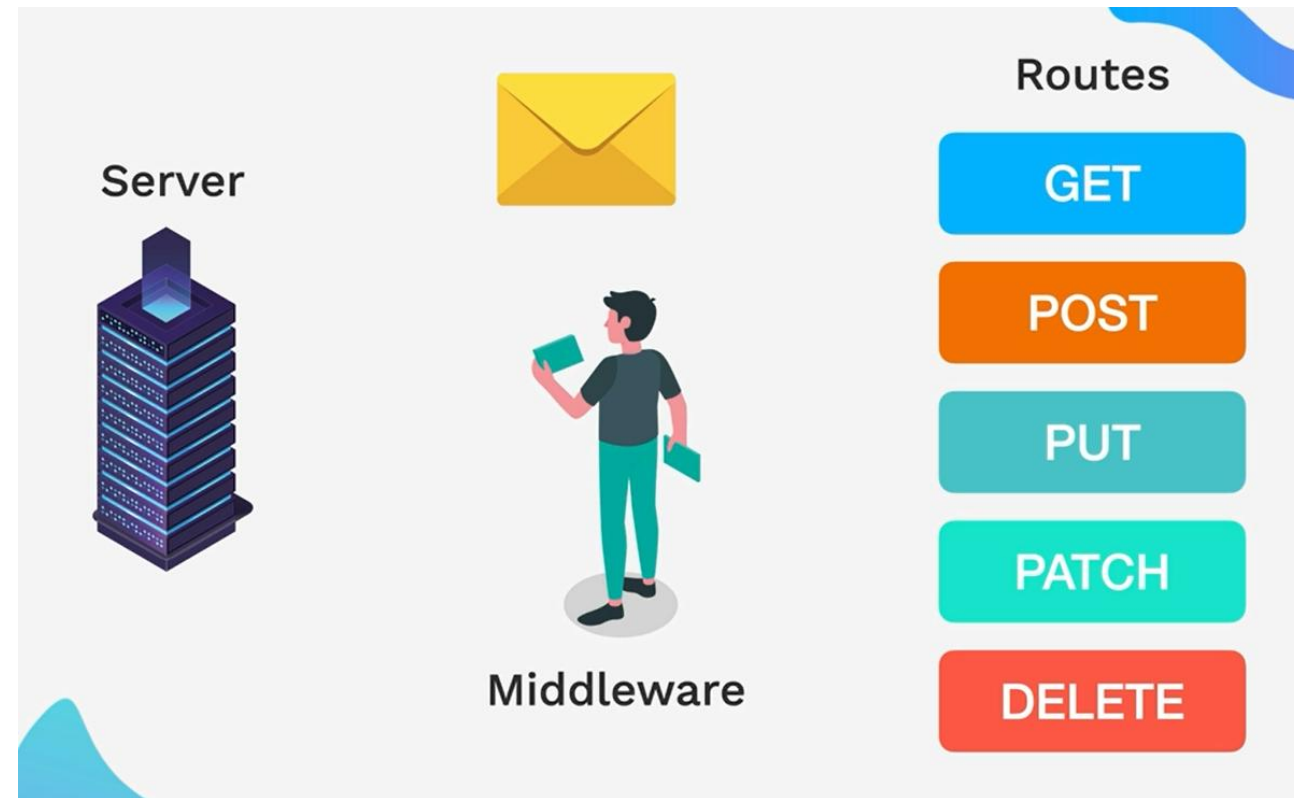
app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'index.html'));
});
```


Middleware



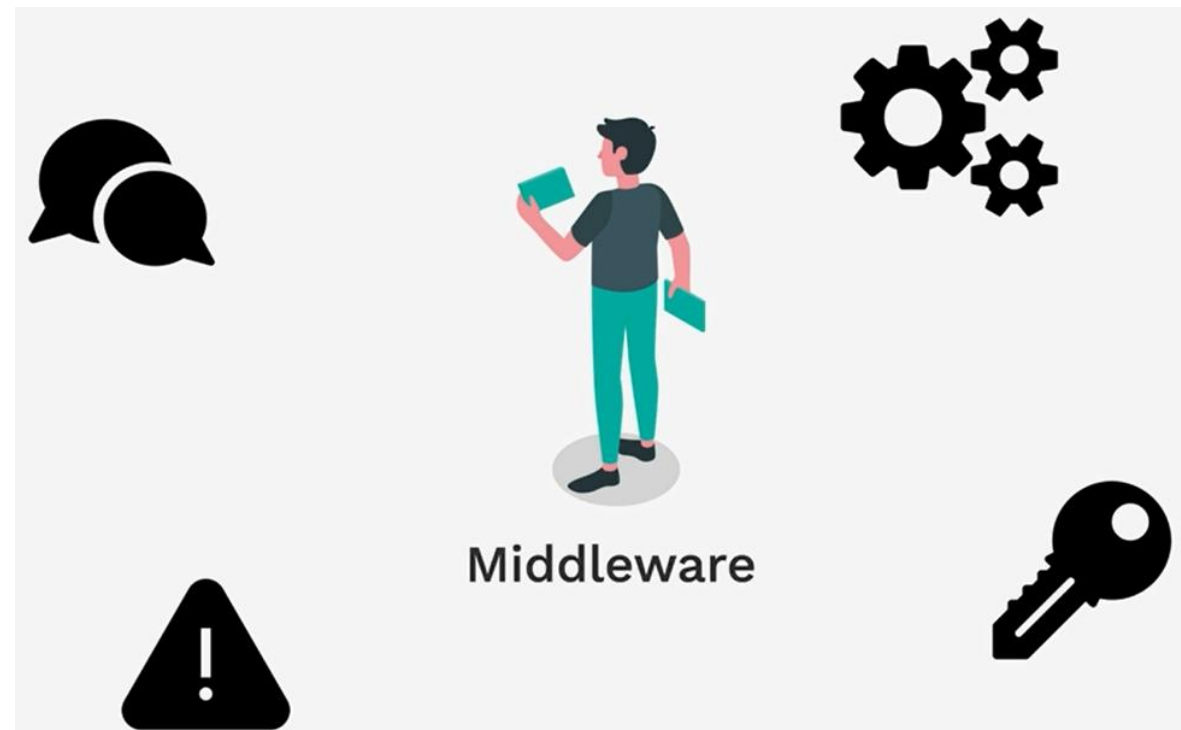
Wozu Middleware?

- Middleware wird verwendet, um HTTP-Anfragen und -Antworten zu verarbeiten, bevor sie an die Endpunkte der Anwendung gelangen.
- Es ermöglicht die Ausführung von Funktionen zwischen dem Empfang der Anfrage und der abschließenden Antwort.



Anwendungsfälle Middleware

- Middleware wird verwendet um:
 - den Inhalt von eingehenden POST-Anfragen zu parsen.
 - um Anfragen und deren Zeitstempel zu protokollieren (Logging)
 - um den Zugriff auf bestimmte Routen zu beschränken.
 - um aufgetretene Fehler abzufangen und eine entsprechende Antwort zu senden.



Middleware bodyParser

- Die Middleware bodyParser vereinfacht den Umgang mit dem Inhalt von HTTP Requests
- bodyParser nimmt den body des Requests und parsed den Inhalt in ein JSON (wird in kommenden Folien erklärt) Format
- import bodyParser from „body-parser“
- Um bodyParser für den Server nutzbar zu machen wird folgende Zeile implementiert
„app.use(bodyParser.urlencoded({extended:true}))

```
import express from "express";
import { dirname } from "path";
import { fileURLToPath } from "url";
import bodyParser from "body-parser";
const __dirname = dirname(fileURLToPath(import.meta.url));

const app = express();
const port = 3000;

app.use(bodyParser.urlencoded({extended:true}))

app.get("/", (req, res) => {
  res.sendFile(__dirname + "/public/index.html");
});

app.post("/submit", (req, res) => {
  console.log(req.body)
  res.sendStatus(200)
})
```

Middleware bodyParser

- **app.use**: Registriert Middleware in einer Express-Anwendung, um sie für alle eingehenden Anfragen zu verwenden.
- **bodyParser.urlencoded**: Spezifiziert, dass die Middleware den Anfragekörper im URL-kodierten Format parsen soll. Dieses Format wird oft verwendet, wenn Formulardaten von HTML-Formularen übermittelt werden.
- **{ extended: true }**: Dies ist eine Konfigurationsoption für bodyParser. Wenn extended auf true gesetzt ist, ermöglicht dies das Parsen von URL-kodierten Daten mit komplexen Strukturen (wie verschachtelte Objekte). Auf den Inhalt kann mit „**req.body**“ zugegriffen werden

```
import express from "express";
import { dirname } from "path";
import { fileURLToPath } from "url";
import bodyParser from "body-parser";
const __dirname = dirname(fileURLToPath(import.meta.url));

const app = express();
const port = 3000;

app.use(bodyParser.urlencoded({extended:true}))

app.get("/", (req, res) => {
  res.sendFile(__dirname + "/public/index.html");
});

app.post("/submit", (req, res) => {
  console.log(req.body)
  res.sendStatus(200)
})
```

Übung zu bodyParser

- Erweitere deinen Node Server um einen POST „/submit“ Endpoint
- Wenn dieser Endpoint aufgerufen wird, dann wird folgender Text angezeigt:
„Der User [eingegebener Username] hat sich gerade angemeldet“

Middleware morgan

- Die middleware morgan vereinfacht das logging von Anfragen
- Alle Anfragen werden automatisch in der Konsole gelogged
- Sehr einfach implementiert:
`app.use(morgan('combined'))`

```
import morgan from 'morgan';

const app = express();

// Verwende das 'combined'-Format von Morgan
app.use(morgan('combined'));
```

```
:::1 - - [22/Jan/2024:21:20:36 +0000] "POST /submit HTTP/1.1" 200 2 "http://localhost:3000/"
"Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Mobile Safari/537.36"
```


Middleware DIY

- Benutzerdefinierte Middleware erlaubt die Erstellung eigener Funktionen für die Verarbeitung von Anfragen und Antworten in Express.
- spezielle Aufgaben werden vor dem Erreichen des Endpunkts in den Anfragezyklus eingefügt werden.
- Mit `app.use(Funktionsname)` wird die Middleware für den Server aktiv gesetzt
- Die Funktion sollte folgende Parameter beinhalten
 - `req, res, next`
 - Am Ende der Funktion sollte `next()` verwendet werden, wenn an den tatsächlichen Endpoint weitergeleitet werden soll

```
app.use(logger)

function logger(req, res, next){
  console.log(req.body.username)
  next();
}
```

Übung zu Middleware DIY

- Schreibe eine Middleware `checkPassword(req, res, next)` die prüft ob das eingegebene Passwort mit einem Passwort deiner Wahl übereinstimmt
- Wenn der „submit“ Endpoint aufgerufen wird und das Passwort nicht stimmt dann soll eine Überschrift mit dem Text „Stop - Passwort nicht gültig“ angezeigt werden
Tipp: `req.url` liefert den angeforderten Endpoint
- Wenn das Passwort korrekt ist dann wird an den „submit“ endpoint weitergeleitet
- ```
if(req.url==="/submit")
 {...}
else{next()}
```






**EJS**

**<%= EJS %>**

Embedded JavaScript templating.


# Visual Studio Extension



## EJS language support v1.3.3

DigitalBrainstem | 1,023,159 | ★★★★★ (29)

2019 - EJS language support for Visual Studio Code.

[Uninstall](#) 

# Wozu brauchen wir EJS?

- EJS (Embedded JavaScript) ist ein serverseitiges Templating-System, das es ermöglicht, dynamische HTML-Inhalte auf dem Server zu generieren.
- EJS erlaubt die direkte Einbettung von JavaScript-Code in HTML-Dateien. Dafür wird `<% eingebetter JS Code %>` verwendet
- Um innerhalb von eingebettetem JS Code HTML einzubinden wird die folgende Syntax verwendet `{ %> HTML Inhalt <% }`
- Durch die Verwendung von `<%= Variablenname %>` können in EJS Variablen, die durch den Server übermittelt wurden, eingebunden werden
- Die Dateiendung lautet `.ejs`, statt `.html`

```
<body>
 <% for(i=0; i<5; i++) {{ %>
 <h1>Hallo, liebe Codersbay Schüler</h1>
 <% }} %>
</body>
```

```
<body>
| <h1>Hello, <%= username %></h1>
</body>
</html>
```

# Wie bindet man EJS am Server ein?

- Damit der Server ejs verwenden kann müssen wir zunächst das Package dafür installieren über „npm i ejs“
- Mit der folgenden Zeile wird ejs beim server registriert  
`app.set(„view engine“,“ejs“)`
- Um durch einen Endpoint ein ejs File auszuliefern wird die Methode `res.render(Dateipfad, Object)` verwendet
- Im Objekt werden Parameter für das ejs file übergeben

```
app.set('view engine', 'ejs');
```

```
app.post("/submit", (req, res) => {
 res.render(__dirname + "/public/ejsexample.ejs",
 {
 username: req.body.username
 })
 res.sendStatus(200)
})
```

# Übung zu EJS

- Erstelle ein EJS File indem der Username und das Passwort die durch das Formular übermittelt wurden nochmal ausgegeben werden
- Passe deinen server so an, dass beim Endpoint „submit“ nun das ejs File als Antwort gerendert wird und als
- Folgender Text wird durch das EJS File angezeigt:  
„Hallo [Username] dein Passwort lautet [Passwort]“

**<%= EJS %>**

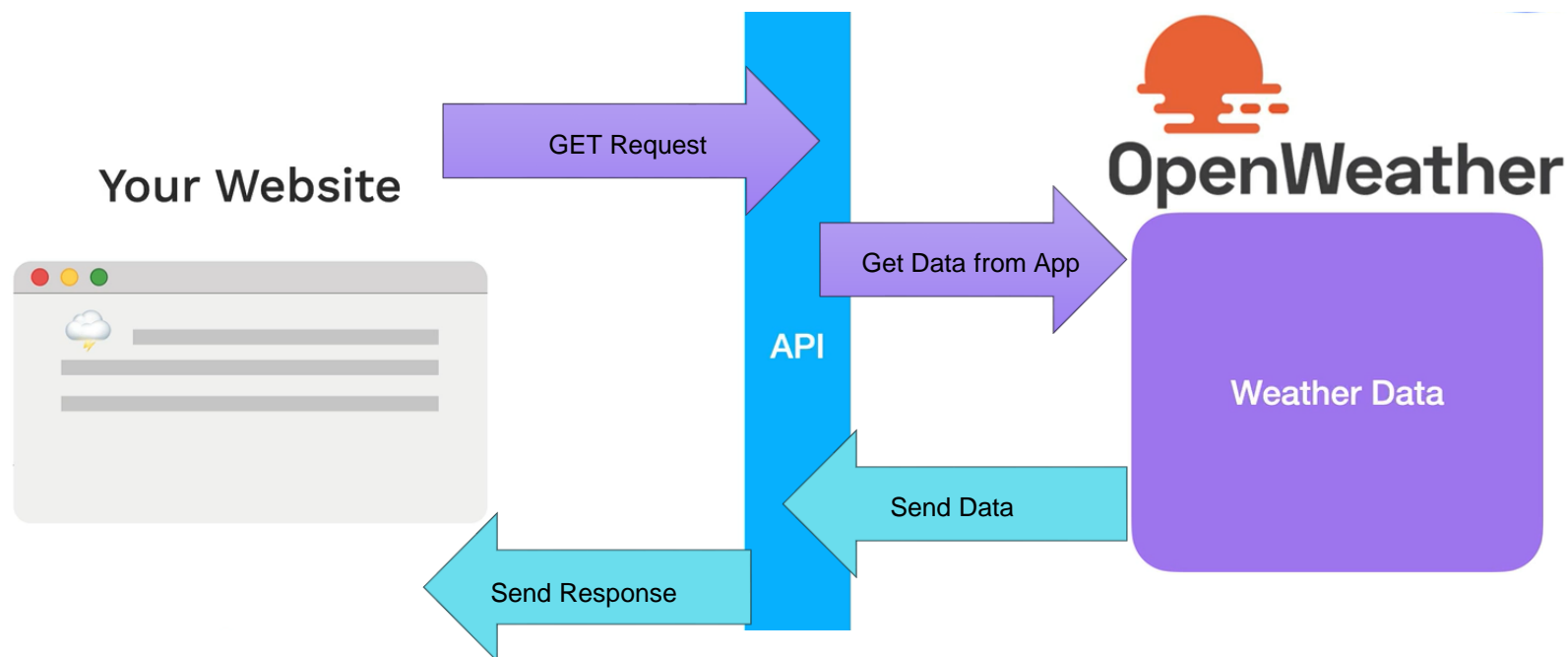
Embedded JavaScript templating.

# Application Programming Interfaces (APIs)



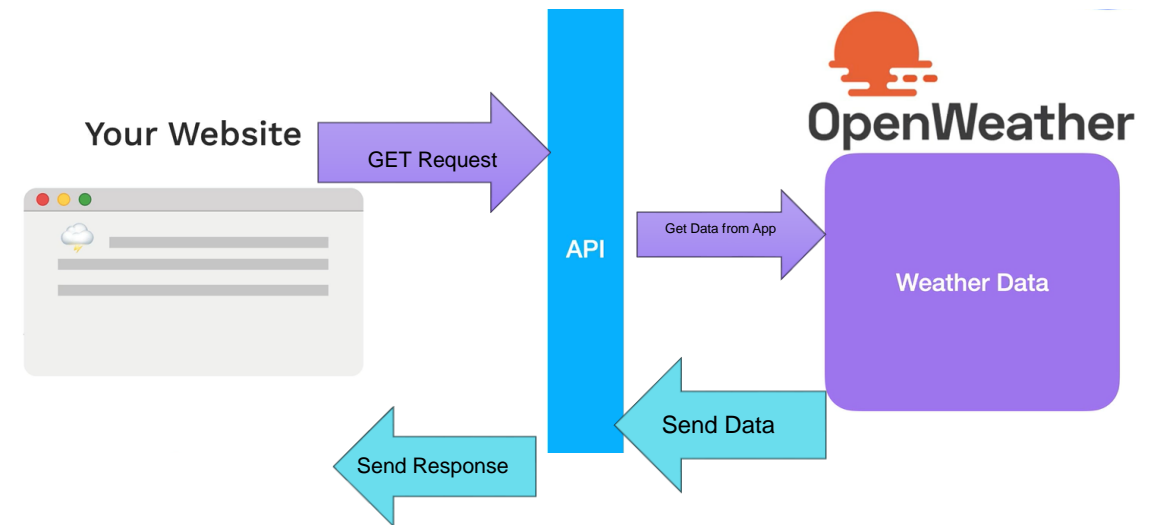


# Was ist eine API?

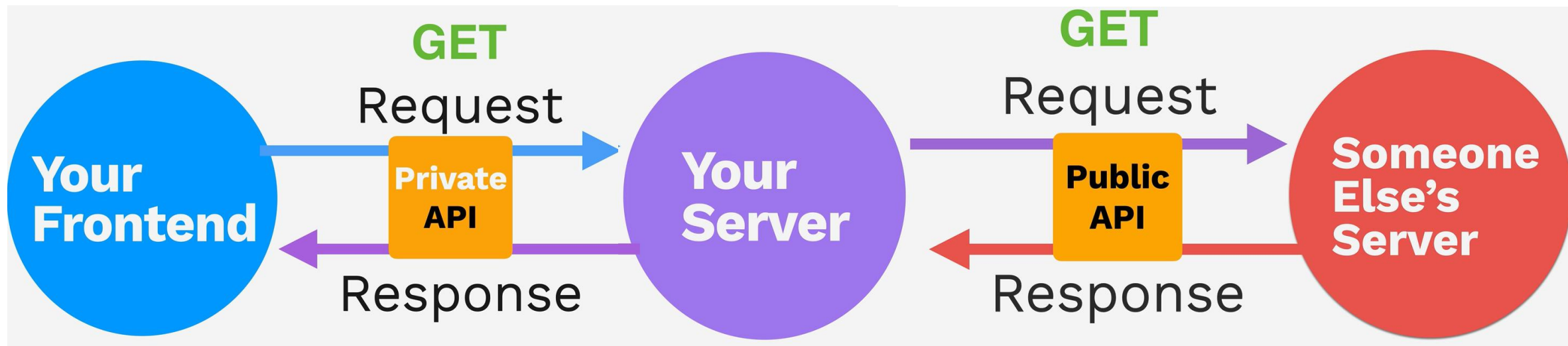


# Was ist eine API?

- Eine API (Application Programming Interface) ist eine Schnittstelle, die es Anwendungen ermöglicht, auf Daten oder Funktionen einer anderen Softwarekomponente zuzugreifen.
- Sie definieren die Regeln und Protokolle, nach denen verschiedene Softwarekomponenten miteinander kommunizieren können, unabhängig von ihrer internen Implementierung.



# Private API vs Public API



# HTTP Protokoll

- **GET:** Die GET-Methode wird verwendet, um Daten von einem Server abzurufen.
- **POST:** Mit der POST-Methode können Daten an einen Server gesendet werden, um Ressourcen zu erstellen oder zu aktualisieren. Die Daten werden im Body der Anfrage übermittelt.
- **PUT:** Die PUT-Methode wird verwendet, um Daten an einen bestimmten URI zu senden, um die Ressource an diesem Ort zu erstellen oder zu aktualisieren.
- **PATCH:** Die PATCH-Methode wird verwendet, um partielle Aktualisierungen an einer Ressource vorzunehmen. Es wird nur der Teil der Daten übermittelt, der aktualisiert werden soll.
- **DELETE:** Mit der DELETE-Methode können Ressourcen auf einem Server gelöscht werden. Die Anfrage gibt an, welche Ressource gelöscht werden soll.

GET

POST

PUT

PATCH

DELETE

# Query Parameter

- Query-Parameter sind zusätzliche Informationen, die an eine URL angehängt werden, um spezifische Anfragen zu formulieren.
- Sie folgen dem Fragezeichen in der URL und bestehen aus Schlüssel-Wert-Paaren (z. B. ?key1=value1&key2=value2).
- Sie ermöglichen es, Filter, Sortierung oder andere Parameter an eine Serveranfrage anzuhängen.
- In Node.js und Express können sie mit req.query abgerufen werden.

```
bored-api.appbrewery.com/endpoint?query=value
```

# Path Parameter

- Pfadparameter sind Teile der URL, die dazu verwendet werden, variable Daten in den Pfad einer RESTful-API-Anfrage einzufügen.
- Sie werden in der Regel in geschweiften Klammern dargestellt, z. B. /users/{userId}.
- Sie ermöglichen es, dynamische Teile in der URL zu definieren, wie z. B. die ID eines Benutzers oder einer Ressource.
- In Node.js und Express können sie mit req.params abgerufen werden

```
bored-api.appbrewery.com/endpoint/{path-parameter}
```

# Übung zu Query / Path Parameter

- Nutze Postman um Anfragen an eine Public API zu stellen um bestimmte Aktivitäten zu bekommen wenn dir langweilig ist
- Nutze dafür die „bored API“, die unter folgendem Link dokumentiert ist <https://bored-api.appbrewery.com/>
- Sende eine Abfrage für alle Aktivitäten mit dem Typ „cooking“ und der Anzahl der Teilnehmer = 2
- Sende eine Abfrage um eine spezielle Aktivität mit dem key = 3943506 zu erhalten



# API Standards



GraphQL

{SOAP}

{REST:API}

gRPC



# Regeln für REST APIs

- Jede Ressource in einer RESTful API sollte durch **einen eindeutigen Identifikator** wie eine URL identifiziert werden.
- RESTful APIs folgen dem **CRUD-Prinzip** (Create, Read, Update, Delete) und nutzen die entsprechenden HTTP-Methoden (POST, GET, PUT/PATCH, DELETE) für die Interaktion mit Ressourcen.
- RESTful APIs sind **zustandslos**, was bedeutet, dass jede Anfrage vom Client alle Informationen enthält, die der Server zur Verarbeitung benötigt. Der Server speichert keinen Zustand zwischen Anfragen.
- **Standard Datei Format** wird als Antwort verwendet (JSON, XML,..) Im Normalfall wird **JSON** verwendet
- **Client und Server** sind voneinander **unabhängig**, und software-architektonisch getrennt.

{REST:API}

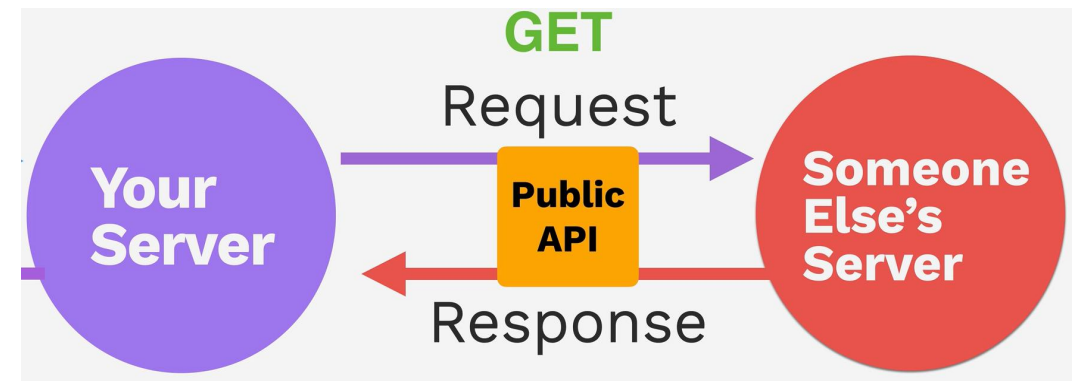
# Was ist JSON?

- JSON (JavaScript Object Notation) ist ein leichtgewichtiges Datenformat, das zur einfachen Darstellung strukturierter Daten verwendet wird.
- JSON verwendet Schlüssel-Wert-Paare, wobei Schlüssel (Strings) mit Werten (Strings, Zahlen, Booleans, Objekte, Arrays oder null) verknüpft sind.
- JSON ist für Menschen leicht lesbar und schreibbar, was es zu einem häufig verwendeten **Format für Datenaustausch in Webanwendungen und APIs** macht.
- JSON unterstützt eine hierarchische Struktur, wodurch komplexe Daten durch die Einbettung von Objekten und Arrays einfach repräsentiert werden können

```
{
 "name": "John Doe",
 "age": 30,
 "city": "Exampleville",
 "isStudent": false,
 "courses": ["Math", "History", "English"],
 "address": {
 "street": "123 Main Street",
 "zipCode": "56789",
 "country": "Exampleland"
 }
}
```

# Server Side API Requests

- Bis hierher haben wir Requests nur über unser Formular im Frontend bzw über Postman versendet
- Häufig findet eine Kommunikation von Server zu Server statt wobei ein Server die API des anderen aufruft
- So kann unsere Applikation Daten (in Form von JSON) von einer public API abrufen und im Frontend anzeigen lassen
- Häufig wird das Package Axios dafür verwendet, um das Package zu installieren führen den Befehl „npm i axios“ aus



# Server Side API Requests mit Axios

## OHNE AXIOS

```
import https from "https";

app.get("/", (req, res) => {
 const options = {
 hostname: "bored-api.appbrewery.com",
 path: "/random",
 method: "GET",
 };

 const request = https.request(options, (response) => {
 let data = "";
 response.on("data", (chunk) => {
 data += chunk;
 });

 response.on("end", () => {
 try {
 const result = JSON.parse(data);
 res.render("index.ejs", {activity: data})
 } catch (error) {
 console.error("Failed to parse response:", error.message);
 res.status(500).send("Failed to fetch activity. Please try again.");
 }
 });
 });

 request.on("error", (error) => {
 console.error("Failed to make request:", error.message);
 res.status(500).send("Failed to fetch activity. Please try again.");
 });

 request.end();
});
```

## MIT AXIOS

```
app.get("/endpoint", (req, res) => {
 let yourUrl = "YOUR API URL";
 try {
 axios.get(yourUrl).then((response) => {
 res.status(200).send("your Message"+response.data);
 });
 } catch (error) {
 res.status(500).send("Fehler beim Aufruf der Api");
 }
});
```

AXIOS

# Server Side API Requests mit Axios

- `axios.get(URL)` wird verwendet, um eine GET-Anfrage an die angegebene API-URL (`yourUrl`) zu senden.
- `.then(response => {...})` wartet auf die Antwort der API und führt daraufhin den Code im inneren Block aus
- Die Daten der Antwort werden unter `response.data` gespeichert
- Bei erfolgreicher API-Antwort wird ein HTTP-Statuscode 200 gesendet, und die Antwort an den Client enthält den Text "your Message" zusammen mit den Daten (`response.data`) der API-Antwort.

```
app.get("/endpoint", (req, res) => {
 let yourUrl = "YOUR API URL";
 try {
 axios.get(yourUrl).then((response) => {
 res.status(200).send("your Message"+response.data);
 });
 } catch (error) {
 res.status(500).send("Fehler beim Aufruf der Api");
 }
});
```

# Übung wo befindet sich die ISS?

- Schreibe einen zusätzlichen Endpoint in deiner API „/iss“
- Nutze Axios um die API für den Standort der ISS zu ermitteln aufzurufen  
<https://api.wheretheiss.at/v1/satellites/25544>
- Lies dir die Beschreibung der Schnittstelle durch und versuche die API richtig zu verwenden
- Gib die Werte für Breiten (latitude) und Längengrad (longitude) als Response im Frontend zurück
- Gib beide Werte in Google Maps mit , getrennt ein um herauszufinden wo sich die ISS im Moment befindet



**Breitengrad:-12.642526272661**  
**Längengrad: -43.270061844827**

# Synchrone vs. Asynchrone Funktionen

- Synchronen Funktionen führen Code sequenziell aus, blockieren dabei den Ausführungsfaden und warten auf den Abschluss jeder Aufgabe
- Asynchrone Funktionen ermöglichen es, Code nicht blockierend auszuführen und auf das Ergebnis von Aufgaben zu warten, ohne den gesamten Programmfluss zu stoppen.
- Der await - Operator wird innerhalb von asynchronen Funktionen verwendet, um auf das Ergebnis einer asynchronen Operation zu warten.
- Alternativ kann .then verwendet werden um Code auszuführen sobald das Ergebnis der asynchronen Funktion verfügbar ist

```
async function fetchData() {
 const response = await fetch('https://api.example.com/data');
 const data = await response.json();
 console.log(data);
}
```

```
function fetchData() {
 fetch('https://api.example.com/data')
 .then((response) => response.json())
 .then((data) => {
 console.log(data)
 })
}
```



# Welcome to the Rapid API Hub

## Discover and connect to thousands of APIs

### Categories

Sports  
Finance  
Data  
Entertainment  
Travel  
Location  
Science  
Food  
Transportation  
Music  
Business  
Visual Recognition  
Tools  
Text Analysis  
Weather  
Gaming  
SMS

### Discover More APIs

Browse through our collections to learn about new use cases to implement in your app

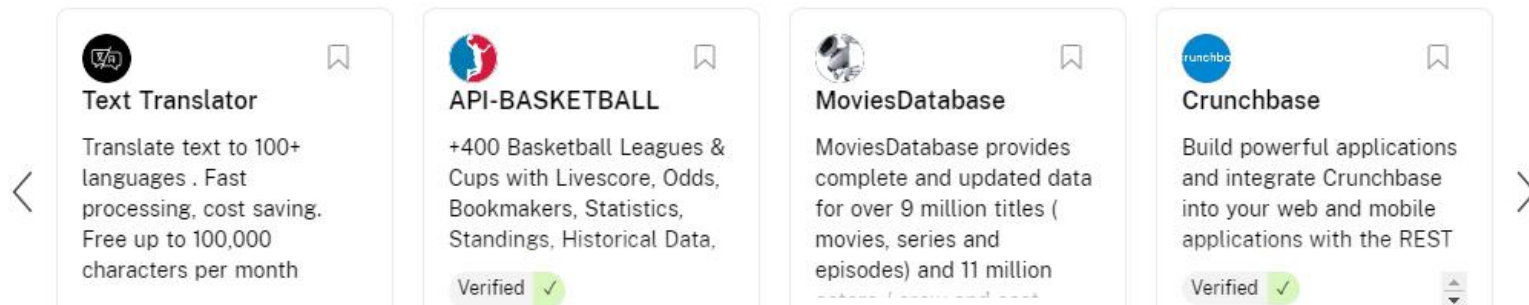


<https://rapidapi.com/hub>

### Recommended APIs

[View All](#)

APIs curated by RapidAPI and recommended based on functionality offered, performance, and support!





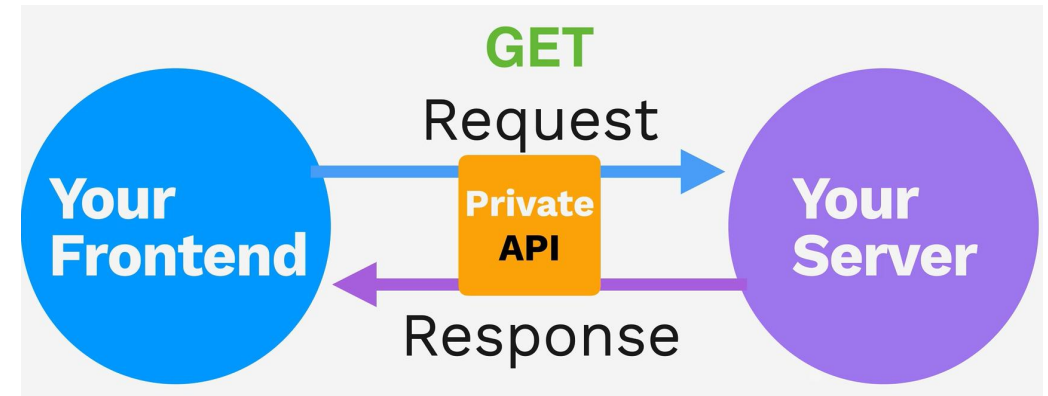
# Übung Witze API

- Erstelle eine API (neuer node-server) die Witze erzählt und verwaltet
- Nutze dafür die Ressource  
„0. Eure Codebeispiele/[Name]/Node/WitzeAPI“
- In den Ressourcen ist das Grundgerüst der API gegeben.  
Ziel ist es nun **mindestens** den „/random“ Endpoint zu erstellen und mit Logik zu befüllen.
- Um die API zu testen könnt ihr die JSON Datei  
„JokeAPI.postman\_collection.json“ in Postman importieren
- In der Postman Doku erhältst du eine genaue Beschreibung der einzelnen Endpoints  
<https://documenter.getpostman.com/view/6048123/2s9XxsTv8Y>
- Freiwillige Hausübung: Einen oder mehrere andere Endpoints implementieren



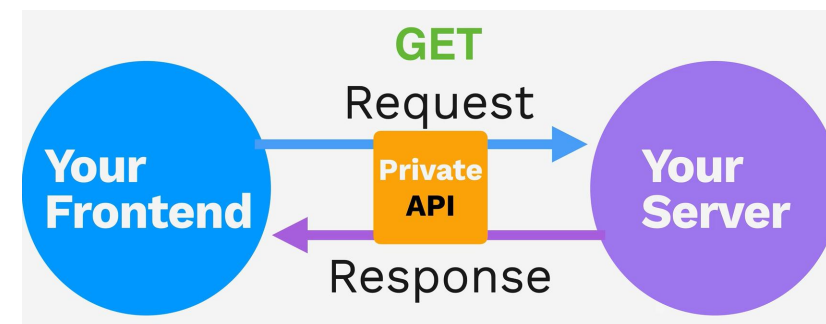
# API Aufruf über Frontend

- Wir haben bislang eine API über AXIOS im Backend oder über Postman aufgerufen
- Im nächsten Schritt werden wir eine API direkt über JS Frontend aufrufen (ohne Node.js)
- Die bereits bekannte AXIOS Methode kann sowohl im Node Backend als auch im JS Frontend verwendet werden
- Sehr häufig wird die JS native Funktion `fetch()` verwendet die wir genauer kennen lernen werden



# Frontend API Aufruf mit fetch() (1)

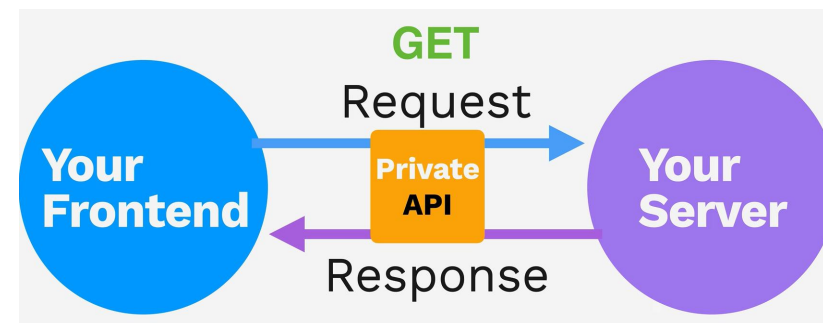
- Um vom Frontend einen API Aufruf abzusetzen wird die Methode `fetch()` verwendet
- Als ersten Parameter wird die URL der API angegeben die man aufrufen möchte (Bsp: `https://api.example.com/data`)
- Als 2. Parameter wird ein Objekt übergeben mit folgenden Properties:
  - **method:** Ermöglicht das Festlegen der HTTP-Methode, standardmäßig wird GET verwendet (GET benötigt keinen body)
  - **headers:** Übermittelt das Format indem der Body übertragen wird (Bsp: `"Content-Type" : "application/json"`)
  - **body:** Übermittelt Daten, die an den Server gesendet werden sollen. Im Fall einer REST API wird hier ein JSON Objekt übergeben
- Die Methode `JSON.stringify(data)` wandelt ein JS Objekt in JSON um. Umgekehrt wäre `JSON.parse()`



```
function fetchData() {
 fetch('https://api.example.com/data',{
 method:"POST",
 headers: {
 "Content-Type" : "application/json"
 },
 body: JSON.stringify({key: "value"})
 })
 .then((response) => response.json())
 .then((data) => {
 console.log(data)
 })
}
```

## Frontend API Aufruf mit fetch() (2)

- Fetch ist eine asynchrone Funktion deren Antwort erst verfügbar ist wenn eine Abfrage erfolgreich war. Deshalb wird mit dem ersten `.then(...)` auf die Antwort der fetch Funktion gewartet
- Innerhalb der ersten `.then` Methode wird `response.json()` verwendet um die Antwort des Servers in ein JSON Object zu parsen.
- Da `.json()` wiederum eine asynchrone Funktion ist muss erneut auf die Antwort gewartet werden. Deshalb wird in einem zweiten `.then` auf das geparste JSON Objekt gewartet und im Parameter `data` gespeichert



```
function fetchData() {
 fetch('https://api.example.com/data',{
 method:"POST",
 headers: {
 "Content-Type" : "application/json"
 },
 body: JSON.stringify({key: "value"})
 })
 .then((response) => response.json())
 .then((data) => {
 console.log(data)
 })
}
```

# Tipp: API Aufrufe in Funktion kapseln

- Um unerwartete Fehlversuche beim Aufruf der API abzufangen wird häufig ein try /catch Block verwendet
- Um auch verschiedene Fehlerfälle abzudecken, kann der Code für einen einfach API Aufruf mehrere Zeilen beinhalten
- Deshalb ist es ratsam die Funktion in ein externes File auszulagern und bei Bedarf zu importieren

```
export const sendPostRequest = async (endpoint, data) => {
 const jwt = getCookie("jwt");
 const endpointUrl = baseUrl + endpoint;

 try {
 const res = await fetch(endpointUrl, {
 method: 'POST',
 headers: {
 Authorization: `Bearer ${jwt}`,
 },
 body: JSON.stringify(data)
 });
 const Response = await res.text();
 if (!res.ok) {
 toast.error(
 "Ein unbekannter Fehler ist aufgetreten"
);
 }
 }
 return Response;
}

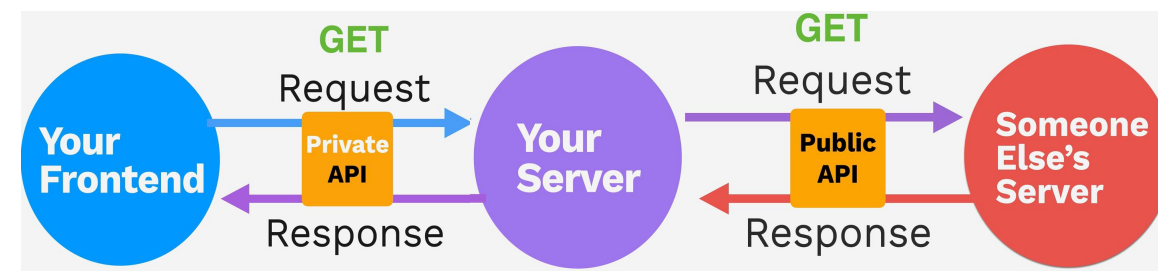
catch (error) {
 if (error.message.includes("Token")) {
 toast.error(
 "Ein Fehlers ist aufgetreten, überprüfe deine Internetverbindung"
);
 }
 throw error;
}
};
```

# Übung gemeinsam: ISS API über Frontend

- Schreibe einen zusätzlichen Endpoint in deiner API „/issdata“
- Nutze Axios um die API für den Standort der ISS zu ermitteln aufzurufen <https://api.wheretheiss.at/v1/satellites/25544>
- Gib die Werte für Breiten (latitude) und Längengrad (longitude) als Response in Form von JSON zurück
- Implementiere einen Button „ISS Daten aktualisieren“, der beim Klick die fetch Methode nutzt um Daten vom Backend Endpoint /issdata zu holen und anzuzeigen



**Breitengrad:-12.642526272661**  
**Längengrad: -43.270061844827**



# Übung: Witze API Aufruf mit fetch()

- Implementiere ein Frontend über das der zuvor entwickelte Witze API Endpoint „/random“ aufgerufen wird und gib den Zufallswitz im Frontend aus.
- Implementiere einen Button der beim Klick immer wieder einen neuen Zufallswitz erzeugt und im Frontend anzeigt
- Verwende dafür einen Event Listener für den Button der beim Klick die fetch-Methode aufruft und die Antwort des /random Endpoints im Frontend ausgibt
- Freiwillige Hausübung: Fontend entwickeln um auch die anderen Endpoints verwenden zu können

ZUFALLS-WITZ

ID: 4

Typ: Science

Witz: "How do you organize a space party? You planet!"





# Node.js-Anbindung an MySQL-Datenbank





# Node.js-Anbindung an MySQL-Datenbank (1)

- Um mit einer MySQL-Datenbank zu interagieren, wird das npm-Package „mysql“ installiert und importiert
- Um den Zugang zur Datenbank zu konfigurieren, wird die Methode `mysql.createConnection({...})` verwendet wobei folgende Properties im Objekt übergeben werden (Beispiele für lokale Entwicklung):
  - `host` ("localhost")
  - `user` ("root")
  - `password` ("")
  - `database` ("Name der Datenbank")
- Über die Methode `connection.connect()` wird die Verbindung zur Datenbank aufgebaut

```
import mysql from "mysql"

const connection = mysql.createConnection({
 host: 'localhost',
 user: 'deinBenutzername',
 password: 'deinPasswort',
 database: 'deineDatenbank'
});

connection.connect((err) => {
 if (err) {
 console.error('Fehler bei der Verbindung zur Datenbank:', err);
 } else {
 console.log('Erfolgreich mit der Datenbank verbunden');
 }
});
```

# Node.js-Anbindung an MySQL-Datenbank (2)

- Eine SQL-Abfrage wird in einer const Variable gespeichert (häufig query genannt)
- Über die Methode `connection.query(..)` wird die Query an die Datenbank gesendet und eine Antwort in Form eines Javascript Objekts zurückgegeben.
- `res.json(results)` wird verwendet um das Ergebnis als JSON-Antwort zu senden
- Innerhalb der `connection.query(..)` Methode werden folgende Parameter übergeben
  - Die Query die in der const Variable gespeichert wurde
  - Eine Arrow Funktion mit den Parametern `error` und `results`, um die Ergebnisse der Abfrage verwenden zu können
- Über den Befehl `connection.end()` wird die Verbindung zur Datenbank wieder geschlossen notwendige Ressourcen freigegeben

```
app.get("/endpoint", (req, res) => {
 const query = 'SELECT * FROM deineTabelle';
 connection.query(query, (error, results) => {
 if (error) throw error
 res.json(results);
 });
 connection.end();
});
```

## Node.js-Anbindung an MySQL-Datenbank (3)

- Um sicherzustellen, dass die Datenbankverbindung auch geschlossen wird, wenn der Server beendet wird, werden folgende Codezeilen verwendet

```
process.on('SIGINT', () => {
 connection.end();
 process.exit();
});
```

- **SIGINT** ist das Signal, das normalerweise durch das Drücken von Ctrl+C in der Konsole ausgelöst wird, um ein laufendes Programm zu unterbrechen und zu beenden
- **process.exit()** Dieser Code beendet den Node.js-Prozess, wenn das SIGINT-Signal empfangen wird. Es beendet das Programm vollständig.

# Node.js-Anbindung an MySQL-Datenbank (Server Beispiel)

```
import express from "express";
import bodyParser from "body-parser";
import mysql from "mysql"

const app = express();
const port = 3000;

const connection = mysql.createConnection({
 host: 'localhost',
 user: 'deinBenutzername',
 password: 'deinPasswort',
 database: 'deineDatenbank'
});

connection.connect((err) => {
 if (err) {
 console.error('Fehler bei der Verbindung zur Datenbank:', err);
 } else {
 console.log('Erfolgreich mit der Datenbank verbunden');
 }
});

app.use(bodyParser.urlencoded({ extended: true }));

app.get("/endpoint", (req, res) => {
 const query = 'SELECT * FROM deineTabelle';
 connection.query(query, (error, results) => {
 if (error) {
 res.status(500).send('Interner Serverfehler');
 } else {
 res.json(results);
 }
 });
 connection.end();
});

process.on('SIGINT', () => {
 connection.end();
 process.exit();
});

app.listen(port, () => {
 console.log(`Successfully started server on port ${port}.`);
});
```

# Node.js-Anbindung an Oracle-Datenbank (Server Beispiel)

```
const dbConfig = {
 user: "yourOracleUsername",
 password: "yourOraclePassword",
 connectionString: "yourOracleConnectionString", // e.g., "localhost:1521/orcl"
};

oracledb.getConnection(dbConfig, (err, connection) => {
 if (err) {
 console.error('Error connecting to Oracle Database:', err);
 } else {
 console.log('Successfully connected to Oracle Database');
 }
});

app.use(bodyParser.urlencoded({ extended: true }));

app.get("/endpoint", (req, res) => {
 const query = 'SELECT * FROM yourOracleTable';
 connection.execute(query, [], { autoCommit: true }, (error, results) => {
 if (error) {
 console.error('Error executing Oracle query:', error);
 res.status(500).send('Internal Server Error');
 } else {
 res.json(results.rows);
 }
 });
});
```

# Übung zu MySQL Anbindung mit Node.js (1)

- Baue eine mySQL Datenbank „exercises“ mit xampp und phpmyadmin auf in der sich eine Tabelle „jokes“ befindet
- Die Tabelle „jokes“ beinhaltet folgende Spalten
  - id
  - jokeText
  - jokeType
- Füge mindestens 2 Witze in deiner Tabelle ein
- Erweitere deine API um einen Endpoint „/randomjokefromdb“ mit dem ein Zufallswitz aus der Datenbank geladen und als Antwort in JSON Format zurückgegeben wird



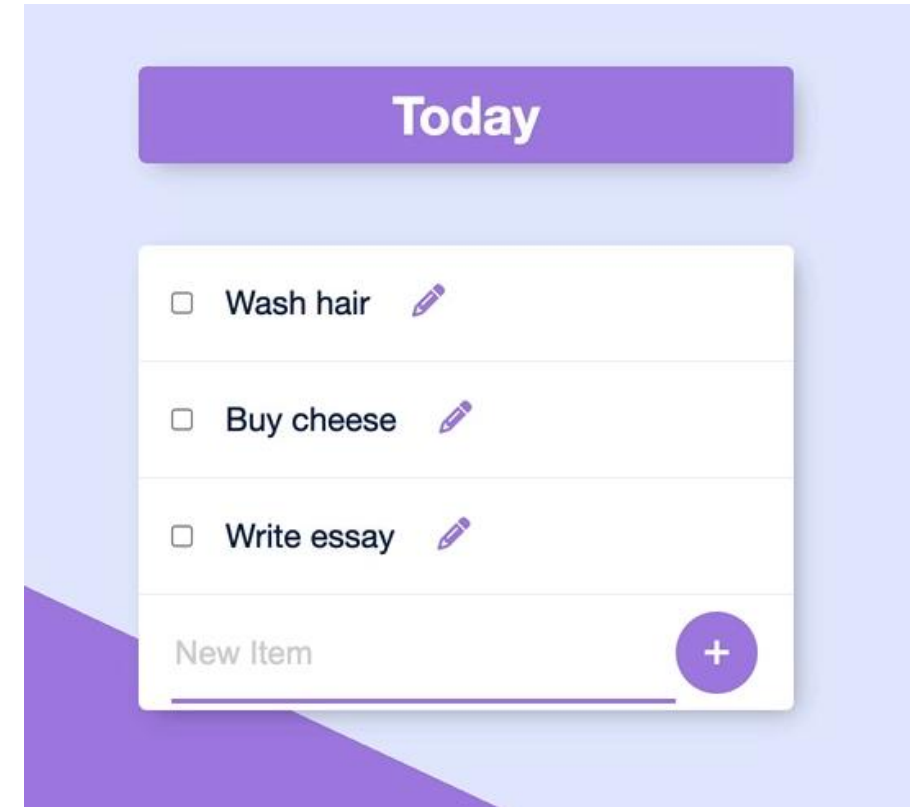
# Übung zu MySQL Anbindung mit Node.js (2)

- Erweitere deine API um einen Endpoint „/alljokesfromdb“ mit dem alle Witze aus der Datenbank geladen werden  
Über einen Button im Frontend können alle Witze in einer Liste angezeigt werden
- Erweitere deine API um einen POST Endpoint „/newjoke“ um einen Witz in der DB einzufügen
- Erweitere deine API um einen DELETE Endpoint „/deletejoke/key“ um einen Witz dessen key als Pfadparameter übergeben wurde zu löschen



# Full-Stack Projekt To-Do Liste: Ziel

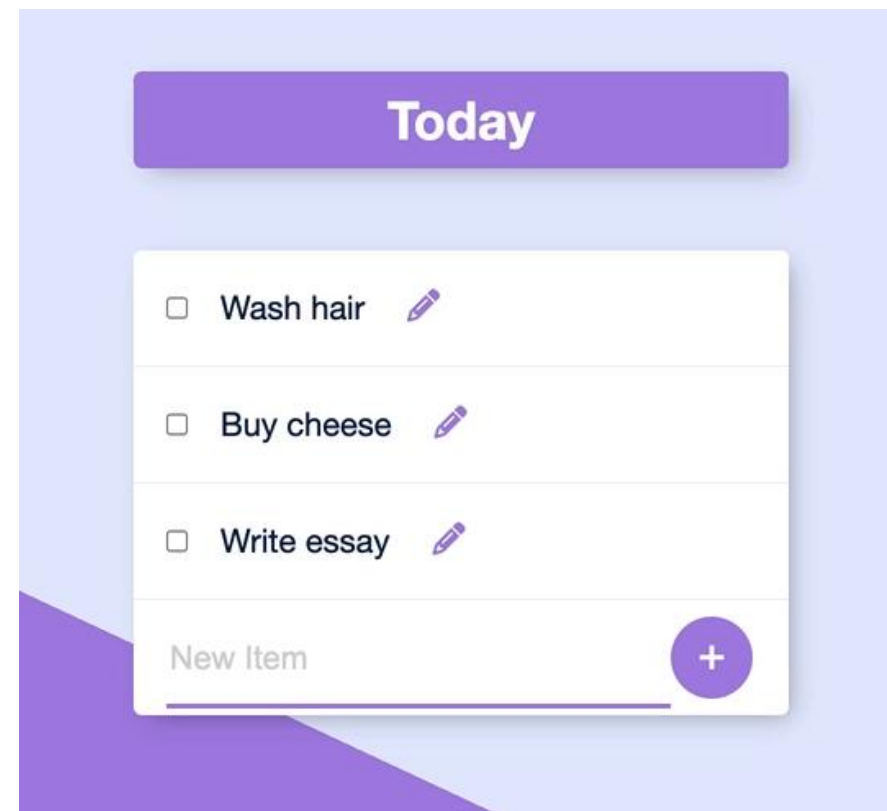
- Zielbild: Webapplikation für eine einfache To-Do Liste wobei Daten dafür in einer mySQL Datenbank liegen. Ein vorgeschalteter Login (Username + Passwort) dient zur Autorisierung
- Jedem Benutzer sind eigene ToDos zugeordnet. Wenn sich der Benutzer anmeldet dann sieht er nur eigene ToDos.
- Folgende Aktionen können in der App gemacht werden:
  - Neues ToDo erstellen
  - Bestehendes ToDo bearbeiten
  - Bestehendes ToDo abhaken (=löschen)





# Full-Stack Projekt To-Do Liste: DB Anbindung

1. Entwirf die Datenbank, die benötigt wird um alle notwendigen Informationen zu persistieren
2. Befülle die Datenbank mit Daten (Benutzer, ToDo-Items)
3. Erstelle einen neuen node server mit einem Endpoint "alltodos" um alle ToDo-Items die gespeichert sind zu laden und als JSON zurückzusenden
4. Integriere dein Backend in dein Frontend und zeige alle Items die in dem Json File zurückkommen in deinem Frontend an



# **Viel Erfolg beim Entwickeln!**

