Erstellen einer REST-API mit Express.js

Eine Vorbereitung auf die heutige Übung

Was ist Express.js?

- **Express.js** ist ein minimaler und flexibler Web-Framework für Node.js.
- Es erleichtert die Entwicklung von Webanwendungen und APIs.
- Bietet eine Vielzahl an Methoden, um Routen und Middleware zu erstellen.

Express.js installieren

```
npm install express
npm install cors
```

- **Voraussetzung**: Node.js und npm müssen installiert sein.
- Installation von express und cors über die Kommandozeile.

Ein einfaches Grundgerüst

```
const express = require('express');
const app = express();
const port = 3000;

app.listen(port, () => {
  console.log(`Server läuft auf Port ${port}`);
});
```

- **require('express')**: Lädt das Express-Modul.
- **app.listen()**: Startet den Server auf dem angegebenen Port.

Middleware hinzufügen

- Middleware verarbeitet Anfragen, bevor sie an die Route gelangen.
- Beispiel für JSON-Parsing:

```
app.use(express.json());
```

- express.json() ist eine eingebaute Middleware-Funktion von Express, die den Body einer HTTP-Anfrage, die im JSON-Format gesendet wurde, analysiert und diese Daten in ein JavaScript-Objekt umwandelt.

CORS aktivieren

- **CORS**: Cross-Origin Resource Sharing
- Erforderlich für Anfragen von anderen Domains.

```
const cors = require('cors');
app.use(cors());
```

- Die Methode cors() fügt den HTTP-Antworten deiner API automatisch die notwendigen CORS-Header hinzu, sodass Browser wissen, dass Anfragen von anderen Domains erlaubt sind.
- Mit app.use(cors()); erlaubst du alle Cross-Origin-Anfragen zu deiner API.

GET-Methode

```
app.get('/items', (req, res) => {
  res.json(items);
});
```

- **app.get()**: Antwortet auf HTTP-GET-Anfragen.
- **res.json()**: Sendet eine JSON-Antwort zurück.

POST-Methode

```
app.post('/items', (req, res) => {
  const newItem = req.body;
  items.push(newItem);
  res.status(201).json(newItem);
});
```

- **app.post()**: Verarbeitet HTTP-POST-Anfragen.
- **req.body**: Enthält die Daten, die im Body der Anfrage gesendet wurden.
- **res.status(201)**: Antwort mit dem Statuscode 201 (Erstellt).
- **:id** ist ein Platzhalter für eine Variable in der URL. In einer Route wie "/items/:id" bedeutet der Doppelpunkt :, dass id ein dynamischer Wert ist, der in der URL bereitgestellt wird.

PUT-Methode

```
app.put('/items/:id', (req, res) => {
  const id = req.params.id;
  const updatedItem = req.body;
  items[id] = updatedItem;
  res.json(updatedItem);
});
```

- **app.put()**: Aktualisiert vorhandene Ressourcen.
- **req.params.id**: Greift auf die URL-Parameter zu.

DELETE-Methode

```
app.delete('/items/:id', (req, res) => {
  const id = req.params.id;
  items.splice(id, 1);
  res.status(204).send();
});
```

- **app.delete()**: Löscht eine Ressource.
- **res.status(204)**: Statuscode 204 (Kein Inhalt) wird zurückgegeben.

Erklärung der `items`-Liste

- **Definition**: `items` ist ein einfaches Array, das in der Anwendung verwendet wird, um Daten zu speichern.
- **Beispiel**:

```
let items = []; // Speichert die Daten
```

- **Verwendung**:
  - Die `items`-Liste dient dazu, Gegenstände (wie z.B. Objekte mit `name` und `type`) temporär zu speichern.
  - In einer echten Anwendung würde man stattdessen eine Datenbank verwenden.

Erklärung des `req`-Objekts

Was ist das `req`-Objekt?

- `req`: Kurzform für **Request** (Anfrage)
- Enthält alle Informationen zur HTTP-Anfrage, die der Server empfängt.

`req.body`

- **Beschreibung**: Enthält die Daten, die im Body einer Anfrage gesendet werden (z.B. bei POST- oder PUT-Anfragen).
- **Beispiel**:

```
app.post('/items', (req, res) => {
  const newItem = req.body; // `newItem` enthält die gesendeten Daten
  items.push(newItem); // Fügt den neuen Gegenstand zur `items`-Liste hinzu
  res.status(201).json(newItem);
});
```

- **Hinweis**: `req.body` funktioniert nur, wenn `express.json()` als Middleware verwendet wird.

`req.params`

- **Beschreibung**: Enthält Parameter aus der URL, die für die Anfrage definiert wurden.
- **Beispiel**:

```
app.put('/items/:id', (req, res) => {
  const id = req.params.id; // `id` ist der Parameter aus der URL
  const updatedItem = req.body; // Neue Daten für den Gegenstand
  items[id] = updatedItem; // Aktualisiert den Gegenstand in der `items`-Liste
  res.json(updatedItem);
});
```

`req.query`

- **Beschreibung**: Ein Objekt, das alle Query-Parameter (Parameter in der URL nach dem `?`) enthält.
- **Beispiel**:

```
app.get('/search', (req, res) => {
  const searchTerm = req.query.term; // Greift auf den `term`-Query-Parameter zu
  res.send(`Suchergebnis für: ${searchTerm}`);
});
```

Validierung in Express.js

- Validierung hilft sicherzustellen, dass Eingaben korrekt sind.
- **Beispiel für eine Middleware**:

```
function validateItem(req, res, next) {
  const { name, type } = req.body;
  if (typeof name !== 'string' || !['Waffe', 'Trank', 'Rüstung'].includes(type)) {
    return res.status(400).json({ message: 'Ungültiger Gegenstand!' });
  }
  next();
}
```

- **Hinweis**: Verwende diese Middleware in deiner POST-Route.

Zufällige Werte generieren

- Verwende `Math.random()` und `Math.floor()`:

```
const randomValue = Math.floor(Math.random() * 41) + 10; // Zufällig zwischen 10 und 50
```

Zusammenfassung

- **Express.js** vereinfacht die Erstellung von APIs.
- Wichtig sind Methoden wie **GET**, **POST**, **PUT**, **DELETE**.
- **Middleware**, **CORS**, und **req-Objekte** sind entscheidend für viele Anwendungen.

Fragen?

- Gibt es noch Unklarheiten oder Fragen zu Express.js?
- Nach der Pause starten wir mit der praktischen Umsetzung!