

Hausaufgabe: REST-API Schritt für Schritt

Ziel der Hausaufgabe

1. Schrittweise Implementierung der REST-API bis Aufgabe 3.
2. Code verstehen und nachvollziehen.
3. API mit Postman testen und erklären, was im Hintergrund passiert.

Teil 1: Implementierung

Schritt 1: Projektvorbereitung

- Erstelle einen neuen Ordner für dein Projekt.
- Öffne die Konsole und initialisiere ein Node.js-Projekt:

```
npm init -y
```

- Installiere die Pakete:

```
npm install express cors
```

- Erstelle eine Datei `server.js` und füge das Grundgerüst ein:

```
const express = require('express');
const cors = require('cors');
const app = express();
const port = 4000;

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

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

Schritt 2: GET-Route für Items

- Füge folgenden Code hinzu:

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

Schritt 3: POST-Route zum Hinzufügen von Items

- Implementiere die Route:

```
app.post('/items', (req, res) => {  
  const { name, type } = req.body;  
  const newItem = {  
    id: items.length + 1,  
    name,  
    type,  
    power: Math.floor(Math.random() * 41) + 10,  
  };  
  items.push(newItem);  
  res.status(201).json(newItem);  
});
```

Schritt 4: POST-Route für einen Kampf

- Implementiere die Kampfroute:

```
app.post('/battle', (req, res) => {  
  const { id } = req.body;  
  const selectedItem = items.find(item => item.id === id);  
  
  if (!selectedItem) {  
    return res.status(404).json({ message: 'Gegenstand nicht gefunden' });  
  }  
  
  const opponent = {  
    name: 'Dunkler Krieger',  
    power: Math.floor(Math.random() * 41) + 10,  
  };  
  
  const result = selectedItem.power >= opponent.power ? 'Gewonnen!' : 'Verloren!';  
  
  res.json({  
    playerItem: selectedItem,  
    opponent,  
    result,  
  });  
});
```

Teil 2: Code verstehen und nachvollziehen

- **Gehe den Code durch** und versuche zu verstehen, was jeder Abschnitt bewirkt.
- Schreibe dir Notizen oder Fragen auf, um Unklarheiten später zu klären.

Teil 3: Tests mit Postman

1. Starte den Server: `node server.js`

2. Führe die folgenden Tests durch:

- **GET-Anfrage:** Rufe die Route `/items` auf und überprüfe, ob eine leere Liste zurückgegeben wird.

- **POST-Anfrage:** Sende eine Anfrage an `/items` , um einen neuen Gegenstand hinzuzufügen. Beispiel-Body:

```
{  
  "name": "Flammenschwert",  
  "type": "Waffe"  
}
```

- **GET-Anfrage erneut:** Rufe `/items` auf und überprüfe, ob der Gegenstand hinzugefügt wurde.
- **Kampfanfrage (Post-Anfrage):** Sende eine Anfrage an `/battle` mit der ID eines Gegenstands. Beispiel-Body:

```
{  
  "id": 1  
}
```

- **Überprüfe die Antwort:** Was wird zurückgegeben? Wer hat gewonnen?

Teil 4: Erklärung

- **Schreibe mit eigenen Worten auf**, was passiert, wenn du eine Anfrage an den Server stellst.
- Erkläre die Prozesse für:
 - Das Hinzufügen eines neuen Gegenstands
 - Den Kampf zwischen deinem Gegenstand und dem Gegner

- **Bereite dich darauf vor**, deine Erklärung in der nächsten Stunde zu teilen.

Abgabe der Hausaufgabe

- **Erstelle ein Textdokument** (z.B. eine Word-Datei oder ein PDF), in dem du die Ergebnisse deiner Tests dokumentierst.

- Notiere für jeden Test:
 - **Was du gesendet hast** (inklusive der Route, der Anfrageart wie `GET` oder `POST`, und dem JSON-Body, falls verwendet).
 - **Was du als Antwort erhalten hast** (inklusive des vollständigen JSON-Outputs).

- **Optional:** Füge **Screenshots aus Postman** ein, um deine Tests zu veranschaulichen. Achte darauf, dass die Screenshots klar und leserlich sind.

Beispiel für die Dokumentation:

1. Test: GET-Anfrage an `/items`

- **Anfrageart:** GET
- **Gesendet:** Keine zusätzlichen Daten
- **Antwort:** `[]` (leere Liste)
- **Screenshot:** [Screenshot einfügen]

2. ...