Erweiterung der REST-API um ein React-Frontend

Übersicht der heutigen Schritte

1. Erstellen eines neuen React-Projekts
2. Aufbau der Projektstruktur
3. Installation der notwendigen Pakete
4. Aufbau der `App.js`-Datei mit Hooks
5. Integration der REST-API mit `axios`
6. Erstellung einer benutzerfreundlichen Oberfläche
7. Projekt ausführen und testen

1. Erstellen eines neuen React-Projekts

- **Öffne dein Terminal** und gehe in einen geeigneten Verzeichnisbereich
- **Befehl zur Erstellung**: Verwende `npx create-react-app`, um ein neues React-Projekt zu erstellen

```
npx create-react-app rest-api-game-frontend
```

- **Was passiert hier?**
  - `npx`: Führt eine einmalige Ausführung eines npm-Pakets durch
  - `create-react-app`: Tool zum Erstellen einer neuen React-Anwendung mit allen wichtigen Voreinstellungen
- **Wechsle in den Projektordner**:

```
cd rest-api-game-frontend
```

2. Aufbau der Projektstruktur

- Deine Projektstruktur sieht nun so aus:

```
rest-api-game-frontend/
├── public/       # Statische Dateien (z.B. index.html)
├── src/          # Haupt-Quellcode der Anwendung
│   ├── App.js    # Hauptkomponente der Anwendung
│   ├── App.css   # Stile für die Hauptkomponente
│   ├── index.js  # Einstiegspunkt der Anwendung
│   └── components/ # (Optional: eigene Komponenten)
├── package.json  # Projekt- und Abhängigkeitsinformationen
└── README.md     # Projektbeschreibung
```

- **Wichtig**: `src/` enthält den gesamten JavaScript-Code, während `public/` statische Dateien wie HTML und Bilder enthält

3. Installation der notwendigen Pakete

- **axios**: Eine Bibliothek, die HTTP-Anfragen vereinfacht
- **Warum axios?**
  - Macht das Abrufen und Senden von Daten an die API einfacher und übersichtlicher
- **Installiere axios**:

```
npm install axios
```

- **Was passiert hier?**
  - `npm install`: Installiert ein Paket und speichert es in der `package.json`-Datei
  - `axios`: Die Bibliothek, die wir hinzufügen

4. Aufbau der `App.js`-Datei

- **App.js** ist die Hauptkomponente der React-Anwendung
- **Vollständiger Code**: Der vollständige Code für `App.js` befindet sich in diesem Repository:
  - GitHub-Link: App.js

5. Überblick über die wichtigsten Code-Abschnitte

1. Importiere Pakete und initialisiere den State

```
import React, { useState, useEffect } from 'react'; // React und Hooks importieren
import axios from 'axios'; // axios für HTTP-Anfragen importieren
import './App.css'; // CSS-Datei für Stile importieren

function App() {
  // useState-Hooks für die Verwaltung der Daten
  const [items, setItems] = useState([]); // Speichert die Liste der Gegenstände
  const [name, setName] = useState(''); // State für die Eingabe des Namens
  const [type, setType] = useState(''); // State für die Eingabe des Typs des Gegenstands
```

5. Überblick (Fortsetzung)

2. useEffect: Daten von der API abrufen

```
useEffect(() => {
  // Beim ersten Rendern: Abrufen der Gegenstände von der API
  axios.get('http://localhost:4000/items')
    .then(response => {
      setItems(response.data); // Daten aus der Antwort in items speichern
    })
    .catch(error => {
      console.error('Fehler beim Abrufen der Gegenstände', error); // Fehlerbehandlung
    });
}, []); // [] bedeutet, dass dieser Effekt nur einmal ausgeführt wird
```

- **useEffect**: Wird ausgeführt, wenn die Komponente zum ersten Mal gerendert wird
- **axios.get**: Führt eine GET-Anfrage aus, um die Gegenstände abzurufen
- **setItems**: Speichert die abgerufenen Gegenstände im State

5. Überblick (Fortsetzung)

3. Funktion zum Hinzufügen eines neuen Gegenstands

```
const addItem = () => {
  // POST-Anfrage senden, um einen neuen Gegenstand hinzuzufügen
  axios.post('http://localhost:4000/items', { name, type })
    .then(response => {
      // Erfolgreiche Antwort: Neuen Gegenstand zur Liste hinzufügen
      setItems([...items, response.data]); // Liste aktualisieren
      setName(''); // Name-Eingabefeld zurücksetzen
      setType(''); // Typ-Eingabefeld zurücksetzen
    })
    .catch(error => {
      console.error('Fehler beim Hinzufügen des Gegenstands', error); // Fehlerbehandlung
    });
};
```

- **axios.post**: Sendet eine Anfrage mit den Eingabedaten (name, type)
- **setItems**: Aktualisiert die Liste der Gegenstände im State
- **Eingabefelder zurücksetzen**: Leert die Eingabefelder nach dem Hinzufügen

6. Eingabefelder und Button

Eingabefelder für den Namen und die Art des Gegenstands

```
<input
  type="text" // Text-Eingabefeld
  placeholder="Name des Gegenstands" // Platzhaltertext
  value={name} // Wert aus dem State name
  onChange={(e) => setName(e.target.value)} // Aktualisiert den State, wenn der Benutzer tippt
/>
<input
  type="text" // Text-Eingabefeld
  placeholder="Art des Gegenstands" // Platzhaltertext
  value={type} // Wert aus dem State type
  onChange={(e) => setType(e.target.value)} // Aktualisiert den State, wenn der Benutzer tippt
/>
<button onClick={addItem}>Hinzufügen</button> // Klick ruft die Funktion addItem auf
```

- **value**: Verknüpft das Eingabefeld mit dem State
- **onChange**: Ändert den State, wenn der Benutzer eine Eingabe macht

7. Rendering der Gegenstände

Anzeige der Liste der Gegenstände

```
<ul>
  {items.map(item => (
    <li key={item.id}> // key hilft React, jedes Element eindeutig zu identifizieren
      {item.name} - Stärke: {item.power} // Zeigt den Namen und die Stärke des Gegenstands an
    </li>
  ))}
</ul>
```

- **items.map**: Durchläuft die Liste der Gegenstände
- **key**: Einzigartige ID für jedes Listenelement, um die Leistung zu verbessern

8. Projekt ausführen und testen

- **Starte das React-Frontend**:

```
npm start
```

- **Öffne** `http://localhost:3000` in deinem Browser
- **Teste die App**: Füge Gegenstände hinzu und überprüfe, ob sie korrekt angezeigt werden

Zusammenfassung der heutigen Einheit

- **Projektstruktur**: Verstehen, wo welcher Code hingehört
- **Hooks**: Verwendung von `useState` und `useEffect`
- **axios**: Kommunikation mit der REST-API
- **Übung**: Experimentiere und erweitere die App, um ein besseres Verständnis zu bekommen

- **Vollständiger Code**: Besuche das Repository auf GitHub für den gesamten Code