Die wichtigsten Konzepte in React

Was ist React?

- **React** ist eine JavaScript-Bibliothek, die von Facebook entwickelt wurde.
- Wird verwendet, um interaktive und dynamische Benutzeroberflächen zu erstellen.
- **Komponenten-basiert**: Der Aufbau einer React-Anwendung besteht aus wiederverwendbaren Komponenten.

Die wichtigsten Konzepte in React

1. **Komponenten**: Bausteine einer React-Anwendung.
2. **State**: Daten, die sich ändern können und die Komponente neu rendern.
3. **Props**: Daten, die von einer Komponente an eine andere weitergegeben werden.
4. **Hooks**: Funktionen, die es uns ermöglichen, den State und den Lebenszyklus einer Komponente in funktionalen Komponenten zu nutzen.

Aufbau einer einfachen React-Komponente

Beispiel einer Komponente

```
import React from 'react'; // Importiere die React-Bibliothek

function Welcome() {
  // Eine einfache Funktion, die JSX zurückgibt
  return <h1>Willkommen bei React!</h1>; // Gibt eine Überschrift zurück
}

export default Welcome; // Macht die Komponente exportierbar
```

Erklärung der Komponente

- **import React from 'react'**: Importiert die React-Bibliothek, die für JSX benötigt wird.
- **function Welcome()**: Erstellt eine funktionale Komponente, die HTML-ähnlichen Code (JSX) zurückgibt.
- **export default**: Macht die Komponente verfügbar, damit sie in anderen Dateien verwendet werden kann.

JSX: JavaScript XML

- JSX ist eine Erweiterung von JavaScript, die es uns ermöglicht, HTML-ähnlichen Code in JavaScript zu schreiben.
- **Beispiel:**

```
const element = <h1>Hallo, Welt!</h1>;
```

- Hinter den Kulissen wird JSX in JavaScript-Methoden übersetzt, die von React verarbeitet werden.

Props in React

Was sind Props?

- **Props (Properties)**: Ermöglichen es uns, Daten von einer übergeordneten Komponente an eine untergeordnete Komponente zu übergeben.

Beispiel mit Props

```
function Welcome(props) {
  return <h1>Hallo, {props.name}!</h1>; // Nutzt den Namen aus den Props
}

// Verwendung der Komponente mit Props
<Welcome name="Max" />
```

- **props.name**: Greift auf die übergebenen Daten zu.
- **<Welcome name="Max" />**: Übergibt "Max" als Prop an die Komponente.

State in React

Was ist State?

- **State**: Daten, die sich im Laufe der Zeit ändern können und das erneute Rendern der Komponente auslösen.

Beispiel: State in einer Komponente

```jsx
import React, { useState } from 'react'; // useState-Hook importieren

function Counter() {
  const [count, setCount] = useState(0); // Initialisiert den State mit 0

  return (
    <div>
      <p>Du hast {count} Mal geklickt</p>
      <button onClick={() => setCount(count + 1)}>Klicken</button> // Erhöht den Zähler
    </div>
  );
}

export default Counter;
```

Erklärung des State-Beispiels

- **useState(0)**: Initialisiert den State "count" mit dem Wert 0.
- **setCount**: Funktion, um den State zu aktualisieren.
- **onClick**: Ereignishandler, der den Zähler erhöht, wenn der Button geklickt wird.

Die Wichtigkeit von Hooks

Einführung in Hooks

- **Hooks**: Funktionen, die es uns ermöglichen, den State und den Lebenszyklus einer Komponente in funktionalen Komponenten zu nutzen.

Wichtige Hooks

- **useState**: Verwaltet den State.
- **useEffect**: Ermöglicht den Zugriff auf den Lebenszyklus einer Komponente, z.B. zum Abrufen von Daten.

useEffect: Der Effekt-Hook

Beispiel: Datenabruf mit useEffect

```
import React, { useState, useEffect } from 'react'; // Importiert useState und useEffect

function DataFetcher() {
  const [data, setData] = useState([]); // State für die Daten
  useEffect(() => {
    // useEffect wird beim ersten Rendern der Komponente ausgeführt
    fetch('https://api.ichbineinBeispiel.com/data') // Daten von einer API abrufen
      .then(response => response.json())
      .then(data => setData(data)); // Daten im State speichern
  }, []); // Leeres Array sorgt dafür, dass der Effekt nur einmal ausgeführt wird

  return (
    <div>
      {data.map(item => (
        <p key={item.id}>{item.name}</p> // Zeigt die abgerufenen Daten an
      ))}
    </div>
  );
}
export default DataFetcher;
```

Erklärung des useEffect-Beispiels

- **useEffect**: Wird ausgeführt, wenn die Komponente geladen wird.
- **fetch**: Führt einen HTTP-Request aus, um Daten von einer API abzurufen.
- **setData**: Speichert die abgerufenen Daten im State.

Beispielaufgabe: Frontend für eine REST-API

Was haben wir gelernt?

- **Komponenten**: Erstellen wiederverwendbarer UI-Bausteine.
- **Props und State**: Übergeben und verwalten von Daten.
- **Hooks**: useState und useEffect zur Datenverwaltung und Lebenszyklussteuerung.

Nächste Schritte

- **Übung**: Setzt das Gelernte um und erstellt ein Frontend für die REST-API.
- Nutzt die Konzepte von React, um eine interaktive Benutzeroberfläche zu entwickeln.