Einführung und Setup

Was ist React? | Vorteile & Einsatzgebiete

React ist eine JavaScript-Bibliothek zur Entwicklung von Benutzeroberflächen, die von Facebook entwickelt und 2013 veröffentlicht wurde. Ihre Hauptstärke liegt in der Möglichkeit, wiederverwendbare UI-Komponenten zu erstellen, die das Entwickeln komplexer und interaktiver Webanwendungen erleichtern.

Vorteile von React:

- Komponentenbasierte Architektur: Entwickeln Sie wiederverwendbare und isolierte Komponenten.
- Virtuelles DOM: React sorgt für schnelle und effiziente UI-Updates, indem es Änderungen im virtuellen DOM verfolgt und nur die notwendigen Teile der realen DOM aktualisiert.
- Unidirektionaler Datenfluss: Einfache Datenverwaltung und Debugging dank des unidirektionalen Datenflusses.
- Große Entwickler-Community und Unterstützung: Umfangreiche Dokumentation, Tools und Bibliotheken dank der großen Community.

Einsatzgebiete von React:

- Single-Page Applications (SPAs)
- Interaktive Benutzeroberflächen in Webanwendungen
- Mobile Apps in Kombination mit React Native
- Serverseitiges Rendering für bessere SEO

Installation und Vorbereitung

Um mit der Entwicklung in React zu beginnen, müsst ihr zunächst Node.js und den Node Package Manager (NPM) auf eurem System installieren. Diese Tools sind unerlässlich, um die erforderlichen Pakete und Abhängigkeiten herunterzuladen und euer Projekt zu verwalten.

Installation von Node.js und NPM

1. Besucht die offizielle Node.js-Website und ladet die neueste LTS-Version herunter.
2. Folgt den Installationsanweisungen für euer Betriebssystem.

Erstellen eines neuen React-Projekts

1. Öffnet eure Kommandozeile oder euer Terminal.
2. Gebt den folgenden Befehl ein, um ein neues React-Projekt mit Create-React-App zu erstellen:

npx create-react-app mein-projektname

!! Der Projektname muss klein geschrieben sein!!

3. Navigiert in das neu erstellte Projektverzeichnis:

cd mein-project

4. Installation von Web-Vitals

npm install web-vitals

5. Starten der Entwicklungsumgebung mit

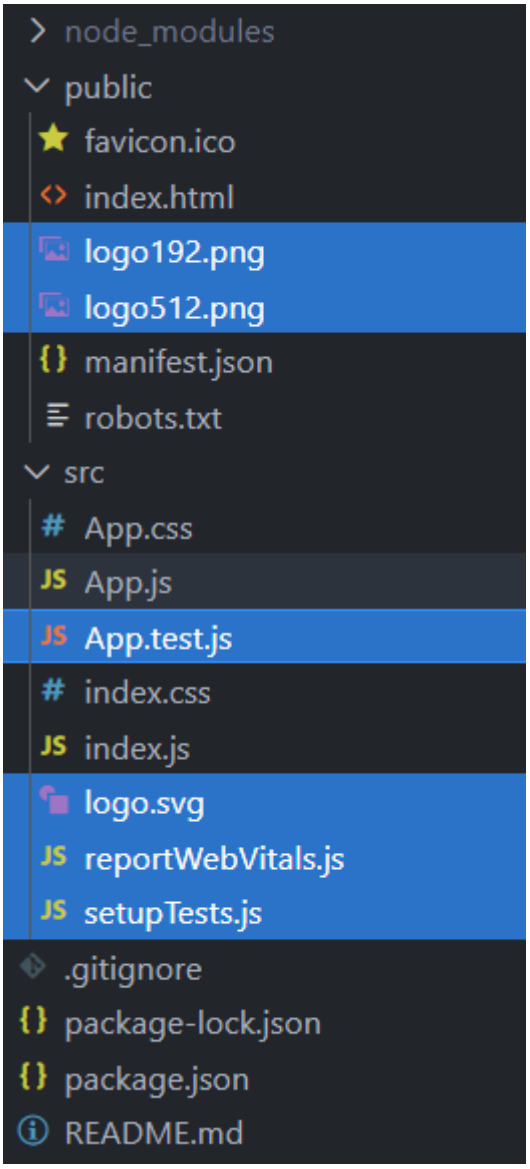
npm start

6. Euer Browser öffnet sich automatisch und zeigt eure React-Anwendung unter [URL] an.

Nachdem ihr die grundlegenden Vorbereitungen getroffen habt, könnt ihr euch nun mit der Projektstruktur und dem Erstellen erster Komponenten beschäftigen.

Projektstruktur und erste Komponenten

Folgende Dateien können für unser Übungsprojekt gelöscht werden.



Anschließend muss die Datei *index.js* noch angepasst werden.

1. Enternen des Ipmorts: import reportWebVitals from './reportWebVitals';

2. Aufruf von WebVitals löschen: reportWebVitals(console.log);

Die Datei *App.js*

1. Enternen des Ipmorts: import logo from './logo.svg';

2. Img-Tag löschen:

3. Inhalt der Funktion Löschen: und nur ein leeres zurück geben

In der Datei *App.css* muss der Inhalt gelöscht werden.

Im *manifest.json* die logos entfernen:

Zum Testen im Terminal npm start eingeben

Grundlagen von React

Komponenten: In React besteht eine Anwendung aus einer Hierarchie von Komponenten. Jede Komponente beschreibt einen Teil der Benutzeroberfläche und wie dieser Teil visuell dargestellt wird. Komponenten können in anderen Komponenten verschachtelt werden, um komplexere Nutzeroberflächen zu schaffen.

JSX: React verwendet JSX, eine Erweiterung der JavaScript-Syntax, die es ermöglicht, HTML-artigen Code direkt in JavaScript zu schreiben. JSX macht den Code lesbarer und verständlicher, indem es die Struktur der Benutzeroberfläche direkt im Code darstellt.

Zustand und Eigenschaften: Komponenten in React können Zustand (state) und Eigenschaften (props) haben. Der Zustand ist ein internes Objekt, das den aktuellen Status der Komponente speichert. Eigenschaften sind Parameter, die von einer übergeordneten Komponente an eine untergeordnete Komponente weitergegeben werden.

Beispiele:

Erstellt im src Ordner einen Unterordner components
State

Erstellt die Datei counter.js im Ordner components

```
1   import { useState } from "react";
2
3   function Counter() {
4     const [count, setCount] = useState(0); // State mit useState Hook
5
6     return (
7       <div>
8         <p>Der aktuelle Zählerstand ist: {count}</p>
9         <button onClick={() => setCount(count + 1)}>+1</button>
10        <button onClick={() => setCount(count - 1)}>-1</button>
11      </div>
12    );
13  }
14
15  export default Counter;
16
```

Nun muss man die Komponente in der App.js einbinden:

```jsx
import './App.css';
import Counter from './components/counter';

function App() {
  return (
    <div>
      <Counter />
    </div>
  );
}

export default App;
```

Props

Erstellt die Datei greeting.js im Ordner components

```jsx
// name = Propertie
function Greeting({name}) {
    return <h1>Hallo, {name}!</h1>;
}

export default Greeting;
```

Nun muss man die Komponente in der App.js einbinden:

Versucht es doch mal selbst. Und Ruft Greeting 2x auf mit verschiedenen Namen.

Virtuelles DOM: React verwendet ein virtuelles DOM (Document Object Model), um Änderungen zu verfolgen und nur die Teile der Benutzeroberfläche neu zu rendern, die sich tatsächlich geändert haben. Dies verbessert die Leistung und sorgt für eine reibungslose Benutzererfahrung.

Interaktionen in React

Ereignishandhabung: React ermöglicht die Handhabung von Benutzerinteraktionen durch Ereignisse wie Klicks, Mausbewegungen und Tastendrücke. Diese Ereignisse können über JSX an Funktionen gebunden werden, die als Ereignishandler fungieren.

```jsx
import { useState } from "react";

function ClickHandler() {
  const [message, setMessage] = useState("Klicke den Button!");

  function handleClick() {
    setMessage("Button wurde geklickt! 🎉");
  }

  return (
    <div>
      <p>{message}</p>
      <button onClick={handleClick}>Klick mich!</button>
    </div>
  );
}

export default ClickHandler;

```

Formulare und Eingaben: Formulare sind ein wesentlicher Bestandteil der Benutzerinteraktion. React bietet eine einfache Möglichkeit, Formulare zu erstellen und deren Eingaben zu verwalten. Der Zustand der Komponente kann verwendet werden, um die Werte von Formularelementen zu speichern und zu aktualisieren.

```jsx
import { useState } from "react";

function FormExample() {
  const [name, setName] = useState("");

  function handleChange(event) {
    setName(event.target.value); // Speichert den Eingabewert im State
  }

  function handleSubmit(event) {
    event.preventDefault(); // Verhindert das Neuladen der Seite
    alert(`Hallo, ${name}!`);
  }

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Dein Name:
        <input type="text" value={name} onChange={handleChange} />
      </label>
      <button type="submit">Absenden</button>
    </form>
  );
}

export default FormExample;
```

Lebenszyklusmethoden: Komponenten in React durchlaufen einen Lebenszyklus mit verschiedenen Phasen, wie Initialisierung, Aktualisierung und Demontage. Lebenszyklusmethoden ermöglichen es Entwicklern, auf diese Phasen mit benutzerdefiniertem Verhalten zu reagieren, z. B. das Abrufen von Daten von einem Server oder das Bereinigen von Ressourcen.

```jsx
import { useEffect, useState } from "react";

function LifecycleExample() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log("Komponente wurde geladen! 🔄");

    return () => {
      console.log("Komponente wird entfernt! ❌");
    };
  }, []); // Leeres Array bedeutet, der Effekt läuft nur einmal beim Laden

  return (
    <div>
      <p>Zähler: {count}</p>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}

export default LifecycleExample;
```

Durch die Kombination dieser Grundlagen und Interaktionsmöglichkeiten bietet React eine leistungsstarke und flexible Plattform zur Entwicklung moderner Webanwendungen.

Tic-Tac-Toe Game:

Qualifiziertes Fazit:

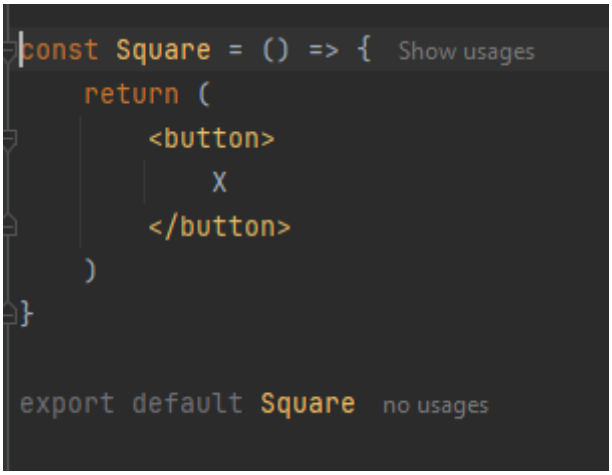
React hat sich als eines der führenden JavaScript-Frameworks für die Entwicklung von Benutzeroberflächen etabliert. Es bietet Entwicklern eine Vielzahl von Werkzeugen und Konzepten, die die Erstellung interaktiver und leistungsstarker Webanwendungen erleichtern. Durch die Nutzung des virtuellen DOM und der Lebenszyklusmethoden wird der Entwicklungsprozess effizienter und die Benutzererfahrung verbessert. Die Fähigkeit, Formulare und Ereignisse einfach zu handhaben, macht React zu einem bevorzugten Werkzeug für moderne Webentwicklung. Insgesamt ermöglicht React eine robuste und skalierbare Entwicklung, die sowohl für kleine als auch für große Projekte geeignet ist.

Einführung in React - Tic-Tac-Toe Spiel

Square-Komponente

Zuerst erstellen wir eine neue Komponente, die ein Feld darstellt, in das der User ein X oder ein O setzen kann. Dafür erstellen wir im components-Verzeichnis einen Ordner square und legen darin die Datei Square.jsx an.

In Square.jsx erstellen wir die Square-Komponente. Sie gibt vorerst nur einen Button mit dem Text X zurück:

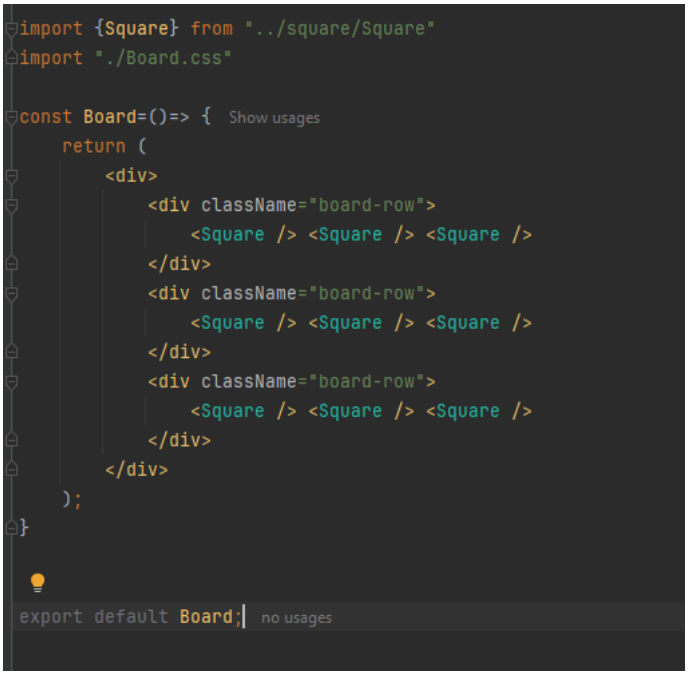


export default Square;

Board-Komponente

Jetzt erstellen wir die Board-Komponente. Dazu legen wir im components-Verzeichnis einen Ordner board an und erstellen darin die Dateien Board.jsx und Board.css.

In Board.jsx erstellen wir die Board-Komponente und importieren die Square-Komponente sowie die Board.css:



Jetzt fügen wir die Board-Komponente in App.js ein, damit sie auf der Website angezeigt wird:

```
import "./App.css"
import {Board} from "./components/board/Board"

function App() {  Show usages
    return <div>
        <Board/>
    </div>
}

export default App  Show usages
```

Spieleraktionen und Statusverwaltung

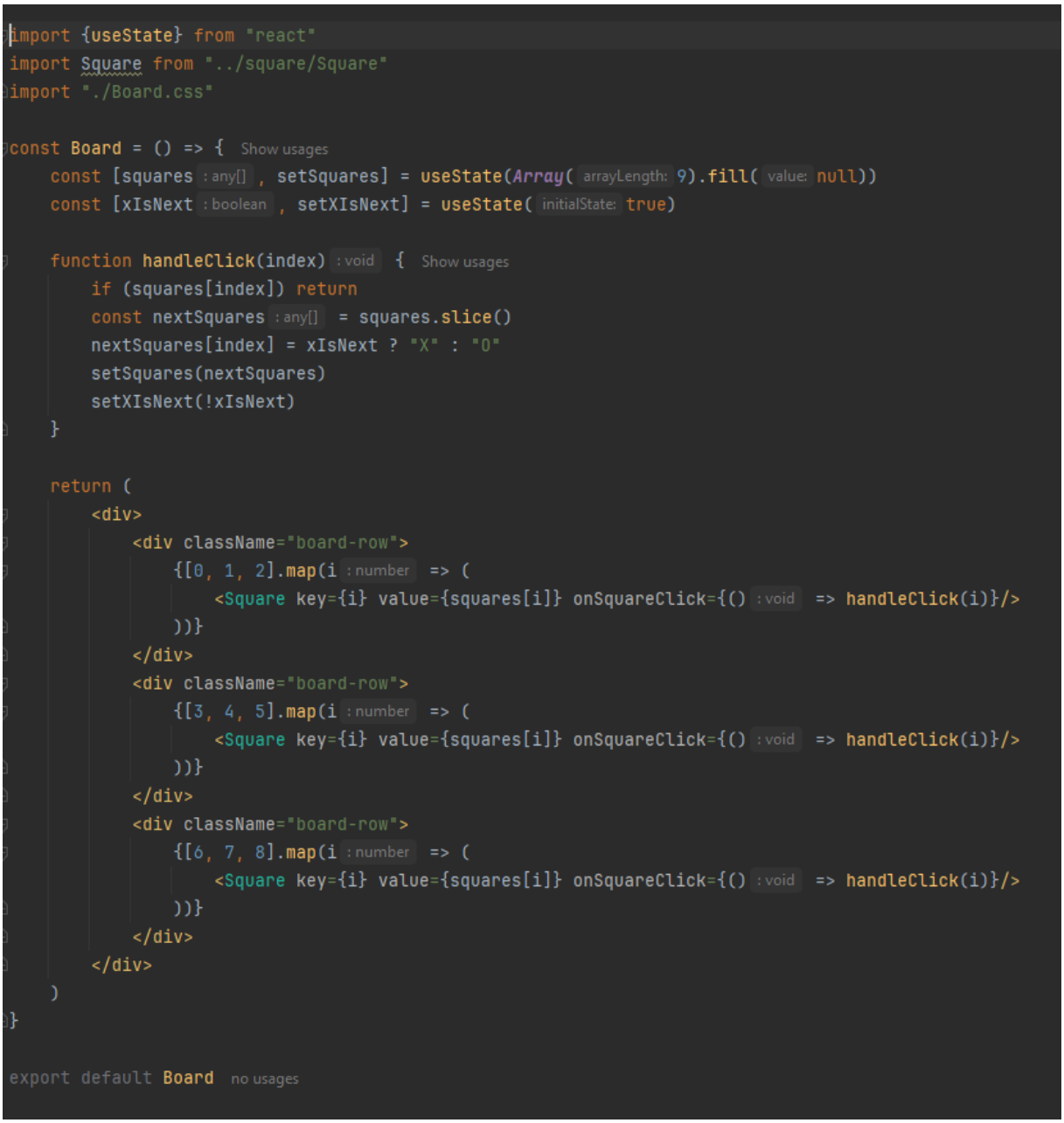
Um den Status der Felder zu verwalten, passen wir die Square-Komponente an, sodass sie dynamisch ein X oder O anzeigt:

```
import { useState } from "react";

const Square=() =>{  Show usages
    const [value, setValue] = useState( initialState: null);

    return (
        <button className="square" onClick={() : void  => setValue( value: "X")}>
            {value}
        </button>
    );
}

export default Square  no usages
```

Jetzt ändert sich das Feld, wenn man darauf klickt.

Spielablauf in der Board-Komponente verwalten

Damit die Züge abwechselnd zwischen X und O wechseln, passen wir Board.jsx an:



```
import {useState} from "react"
import Square from "../square/Square"
import "./Board.css"

const Board = () => {  Show usages
    const [squares : any[] , setSquares] = useState(Array( arrayLength: 9).fill( value: null))
    const [xIsNext : boolean , setXIsNext] = useState( initialState: true)

    function handleClick(index) : void  {  Show usages
        if (squares[index]) return
        const nextSquares : any[]  = squares.slice()
        nextSquares[index] = xIsNext ? "X" : "O"
        setSquares(nextSquares)
        setXIsNext(!xIsNext)
    }

    return (
        <div>
            <div className="board-row">
                {[0, 1, 2].map(i : number  => (
                    <Square key={i} value={squares[i]} onSquareClick={() : void  => handleClick(i)}/>
                ))}
            </div>
            <div className="board-row">
                {[3, 4, 5].map(i : number  => (
                    <Square key={i} value={squares[i]} onSquareClick={() : void  => handleClick(i)}/>
                ))}
            </div>
            <div className="board-row">
                {[6, 7, 8].map(i : number  => (
                    <Square key={i} value={squares[i]} onSquareClick={() : void  => handleClick(i)}/>
                ))}
            </div>
        </div>
    )
}

export default Board  no usages
```

Gewinner ermitteln

Jetzt fügen wir eine Funktion hinzu, um zu prüfen, ob jemand gewonnen hat:

```
function calculateWinner(squares) : any | null  {  no usages
    const lines : (...)[]  = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8],
        [0, 3, 6], [1, 4, 7], [2, 5, 8],
        [0, 4, 8], [2, 4, 6]
    ];
    for (let [a : number , b : number , c : number ] of lines) {
        if (squares[a] && squares[a] === squares[b] && squares[a] === squares[c]) {
            return squares[a];
        }
    }
    return null;
}
```

Jetzt nutzen wir die Funktion in Board.jsx:

```
function Board() {  no usages
    const [squares : any[] , setSquares] = useState(Array( arrayLength: 9).fill( value: null));
    const [xIsNext : boolean , setXIsNext] = useState( initialState: true);

    function handleClick(index) : void  {  Show usages
        if (squares[index] || calculateWinner(squares)) return;
        const nextSquares : any[]  = squares.slice();
        nextSquares[index] = xIsNext ? "X" : "O";
        setSquares(nextSquares);
        setXIsNext(!xIsNext);
    }

    const winner = calculateWinner(squares);
    let status : string  = winner ? `Gewinner: ${winner}` : `Nächster Spieler: ${xIsNext ? "X" : "O"}`;

    return (
        <div>
            <div className="status">{status}</div>
            <div className="board-row">
                {[0, 1, 2].map(i : number  => (
                    <Square key={i} value={squares[i]} onSquareClick={() : void  => handleClick(i)} />
                ))}
            </div>
            <div className="board-row">
                {[3, 4, 5].map(i : number  => (
                    <Square key={i} value={squares[i]} onSquareClick={() : void  => handleClick(i)} />
                ))}
            </div>
            <div className="board-row">
                {[6, 7, 8].map(i : number  => (
                    <Square key={i} value={squares[i]} onSquareClick={() : void  => handleClick(i)} />
                ))}
            </div>
        </div>
    );
}
```

Damit ist unser Tic-Tac-Toe-Spiel voll funktionsfähig!