08 React.js: Router, Strict Mode und Styling

Multimedia Engineering (PS siehe 36609b)

Nils Hellwig, M.Sc.
Lehrstuhl für Medieninformatik
FAKULTÄT FÜR INFORMATIK UND DATA SCIENCE

Universität Regensburg

Quelle: https://vite.dev/guide/

React Environment

- Wenn Node.js installiert ist, kann mithilfe von Vite eine moderne React-Anwendung erstellt werden.

  npm create ermöglicht die Erstellung von Projekt-Templates, u.a. um eine React-Anwendung namens my-react-app zu erstellen:

  ```
  $ npm create vite@latest <Name der neuen Anwendung> -- --template react
  ```

- Danach ins Projektverzeichnis wechseln und Abhängigkeiten installieren:

  ```
  $ cd <Name der neuen Anwendung>  $ npm install
  ```

- Die Anwendung kann dann über einen lokalen Development-Server (standardmäßig unter localhost:5173) gestartet werden mit: `$ npm run dev`

Das erste -- trennt **npm**-Argumente von den Argumenten für **Vite**
--template react ist ein **Vite-internes Argument**, das das React-Template auswählt.
Ohne das doppelte -- würde npm den Parameter falsch interpretieren.

React Environment

- Entwickelt man mit npm run dev, läuft lokal ein Web-Server, man hat Zugriff auf ein paar Entwicklungstools – das ist gut zum Testen.
- Aber für das Veröffentlichen im Internet brauchst du:

  (1) schnellen, kompakten Code

  (2) keine Entwickler-Tools (z.B. Tests etc.)
- Ein Build-Prozess (`npm run build`) erstellt fertige HTML-, JS- und CSS-Dateien im Ordner `dist/`

React JSX: Expressions

- Mit JSX können JavaScript-Ausdrücke innerhalb geschweifter Klammern { } geschrieben werden
- Eine Expression kann eine React-Variable, eine Eigenschaft oder ein anderer JavaScript-Ausdruck sein
- Beispiel:

```
const myElement = <h1>React is {5 + 5} times better with JSX</h1>;
```

React JSX: Expressions

- Das Einfügen eines großen JSX-Blocks ist ebenfalls möglich:

```
const myElement = (
  <div>
    <p>Apples</p>
    <p>Bananas</p>
    <p>Cherries</p>
  </div>
);
```

- Sollen wie hier zwei Absätze geschrieben werden, muss man sie in ein übergeordnetes Element (hier – Element) einfügen.

React Components: Functional Components

- Beispiel:

```
function Car() {
    return <h2>Hi, I am a Car!</h2>;
}
```

React Hooks: useState

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

function Counter() {
    const [counter, setCounter] = useState(0);
    return (
        <>
            <h1>Counter: {counter}!</h1>
            <button type="button" onClick={() => setCounter(counter+1)}>Add +1</button>
        </>
    )
}
```

React Hooks: useEffect

1. **Keine Dependencies übergeben**

```
useEffect(() => {
    // Runs on every render
});
```

2. **Ein leeres Array**

```
useEffect(() => {
    // Runs only on the first render
}, []);
```

3. **State Werte**

```
useEffect(() => {
   // Runs on the first render and any time any dependency value changes
}, [state]);
```

```
import { useState, createContext, useContext } from
"react"; import ReactDOM from "react-dom/client";

const UserContext = createContext();

function Component1() {
    const [user, setUser] = useState("Jesse Hall");
    return (
        <UserContext.Provider value={user}>
            <h1>{`Hello ${user}!`}</h1>
            <Component2 />
        </UserContext.Provider>
    );
}

function Component2() {
    return (
        <>
            <h1>Component 2</h1>
            <Component3 />
        </>
    );
}
```

```
function Component3() {
    return (
        <>
            <h1>Component 3</h1>
            <Component4 />
        </>
    );
}

function Component4() {
    return (
        <>
            <h1>Component 4</h1>
            <Component5 />
        </>
    );
}

function Component5() {
    const user = useContext(UserContext);

    return (
        <>
            <h1>Component 5</h1>
            <h2>{`Hello ${user} again!`}</h2>
        </>
    );
}
```

React Router

- `vite/react` umfasst standardmäßig keine Schnittstelle für Page-Routing
- Der „React Router“ ist dafür die beliebteste Lösung
- Installation des Routers:

```
npm i react-router-dom
```

https://reactrouter.com/start/declarative/routing

React Router: Demo

```
import { useState } from "react";
import reactLogo from "./assets/react.svg";
import viteLogo from "/vite.svg";
import "./App.css";
import Home from "./components/Home";
import About from "./components/About";
import Contact from "./components/Contact";
import { Route, Routes } from "react-router-dom";

function App() {
  return (
   <div>
     <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
     </Routes>
   </div>
  );
}

export default App;
```

React Router: <Link/>

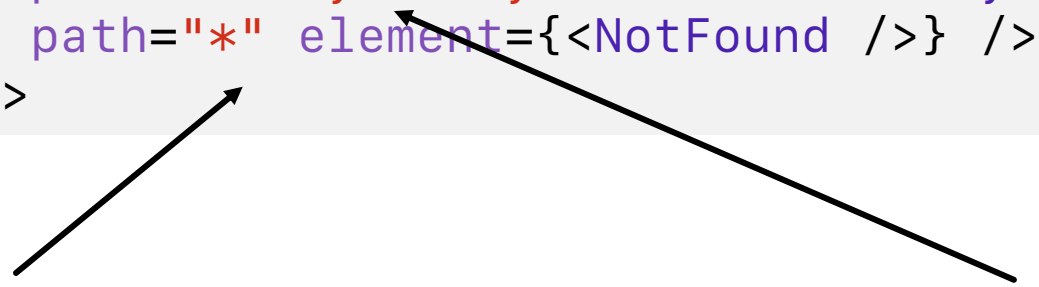
- Ersetzt <a>-Tags für clientseitige Navigation falls der Router genutzt wird.
- Link **verhindert das Neuladen** der Seite, was bei klassischen `<a href="...">` passiert.
- States bleiben erhalten wenn Seite gewechselt wird (siehe Beispiel auf GRIPS/GitHub)

```jsx
import { Link } from "react-router-dom";

function Navigation() {
  return (
    <nav>
      <Link to="/">Startseite</Link>
      <Link to="/kontakt">Kontakt</Link>
    </nav>
  );
}
```

React Router: 404 Page / Individueller Pfad

```
<Routes>
  <Route path="/" element={<Home />} />
  <Route path="/about" element={<About />} />
  <Route path="/contact" element={<Contact />} />
  <Route path="/city/:city" element={<City />} />
  <Route path="*" element={<NotFound />} />
</Routes>
```



Die path="*"-Route fängt **alle Pfade ab**, die nicht vorher definiert wurden.

„:“ definiert einen **dynamischen Pfad** in deiner React Router-Konfiguration. Sie wird aktiv, wenn ein Nutzer eine URL wie /city/berlin oder /city/muenchen aufruft.

React Komponenten mit CSS stylen

- Inline Styling

```
<h1 style={{color: "red"}}>Hello Style!</h1>
```

Da das Inline-CSS in einem JavaScript-Objekt geschrieben wird, müssen Eigenschaften mit Bindestrich-Trennzeichen, wie

z.B. `background-color`, in Camel-Case-Syntax geschrieben werden: `backgroundColor`

- CSS Stylesheet: CSS-Deklarationen können in eine separate CSS-Datei gespeichert werden und importiert werden

```
import './App.css';
const Header = () => {
    return (
       <>
        <h1>Hello Style!</h1>
        <p>Add a little style!.</p>
       </>
    );
}
```

React Conventions

- PascalCase für Komponentennamen und Dateinamen von Komponenten
- camelCase für Funktionen (und deren Dateinamen)

React StrictMode

```
<React.StrictMode>
   <App />
</React.StrictMode >
```

- Der Strict Mode sorgt dafür, dass potenziell unsichere Praktiken und Fehlerquellen im Code erkannt werden. So können Sicherheitslücken vermieden und die Gesamtsicherheit der Anwendung erhöht werden
- Prüfungen im Strict Mode werden nur im Entwicklungsmodus durchgeführt, sie haben keine Auswirkungen auf den Produktions-Build
- ⚠ **Wichtig zu wissen:** `useEffect` wird **falls der StrictMode verwendet wird** doppelt aufgerufen, um Fehler zu erkennen
- Liste der Funktionalitäten des Strict Modes: https://reactjs.org/docs/strict-mode.html

https://app.parteivortrag.de/

https://github.com/markusbink/basisdokument-implementierung

Weiterführende Literatur / Quellen

- Offizielle Dokumentation zu React: https://reactjs.org/docs/getting-started.html
- Ausführliches React Tutorial mit anschaulichen Beispielen: https://www.w3schools.com/react
- Mehr zu React naming conventions: https://www.upbeatcode.com/react/react-naming-conventions/
- Popularität von React: https://trends.google.com/trends/explore?date=all&q=React,Vue%20js,Angular