21 - React: Routing und Navigation mit `react-router-dom`

Einleitung

- **Themen**: In diesem Skript verwandeln wir unsere bisherige "Ein-Seiten"-Anwendung in eine vollwertige **Single-Page Application (SPA)** mit mehreren, navigierbaren Ansichten. Wir führen die Standard-Bibliothek für das Routing in React ein: `react-router-dom`.
- **Fokus**: Das grundlegende Konzept des "Client-Side Routings". Wir lernen, wie man verschiedene URL-Pfade definiert, welche React-Komponente für welchen Pfad angezeigt werden soll, wie man zwischen diesen "Seiten" navigiert und wie man dynamische Daten (wie eine ID) aus der URL ausliest.
- **Lernziele**:
  - Verstehen, warum eine SPA eine spezielle Routing-Bibliothek benötigt.
  - `react-router-dom` in einem Projekt installieren und die grundlegende Konfiguration vornehmen.
  - Die Kernkomponenten `<BrowserRouter>`, `<Routes>` und `<Route>` verstehen und anwenden.
  - Klickbare Navigationslinks mit der `<Link>`-Komponente erstellen.
  - Dynamische Routen für Detailseiten (z.B. `/rezepte/123`) definieren.
  - Den `useParams`-Hook verwenden, um dynamische Parameter aus der URL auszulesen.
  - Den `useNavigate`-Hook für programmgesteuerte Weiterleitungen (z.B. nach einem Login) nutzen.

1. Das Problem: Navigation in einer Single-Page Application

In einer traditionellen Webseite führt ein Klick auf einen Link (`<a href="/about">Über Uns</a>`) dazu, dass der Browser eine komplett neue HTML-Seite vom Server anfordert und die aktuelle Seite vollständig ersetzt wird.

In einer SPA wollen wir das **verhindern**. Unsere gesamte Anwendung läuft bereits im Browser. Ein Neuladen würde den gesamten Zustand (alle State-Variablen) unserer React-Komponenten zurücksetzen und die flüssige Benutzererfahrung zerstören.

Die Lösung: Wir benötigen **Client-Side Routing**. Eine JavaScript-Bibliothek fängt Klicks auf Links ab, ändert die URL in der Browser-Adresszeile manuell (ohne Neuladen) und sorgt dann dafür, dass React die passenden Komponenten für die neue URL anzeigt.

2. Die Lösung: `react-router-dom`

`react-router-dom` ist die De-facto-Standardbibliothek für das Routing in React-Webanwendungen.

- **Installation**: Öffne ein Terminal im Verzeichnis deines React-Projekts und führe folgenden Befehl aus:

```
# Installiert die Bibliothek und fügt sie zu den Abhängigkeiten in package.json hinzu
npm install react-router-dom
```

3. Die Grundkonfiguration (BrowserRouter, Routes, Route)

Die Einrichtung besteht aus drei zentralen Komponenten, die zusammenarbeiten.

3.1 `<BrowserRouter>` (Der "Aktivator")

- **Was er macht**: Diese Komponente aktiviert das Client-Side Routing für die gesamte Anwendung. Sie muss alle anderen Komponenten umschließen, die vom Routing betroffen sind. Sie "hört" auf Änderungen in der URL-Leiste des Browsers und stellt sicher, dass unsere App darauf reagiert.
- **Syntax und Platzierung**: Üblicherweise wird die `<App />`-Komponente in `src/main.jsx` mit dem `<BrowserRouter>` umschlossen.

Beispiel `src/main.jsx`:

```
// src/main.jsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';
import { BrowserRouter } from 'react-router-dom'; // Importieren
```

```
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    {/* Wir umschließen die gesamte App mit dem BrowserRouter */}
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

3.2 `<Routes>` und `<Route>` (Die "Wegweiser")

- **`<Routes>` (Der "Weichensteller")**: Diese Komponente agiert als Container für alle unsere Routen-Definitionen. Sie schaut sich die aktuelle URL an und rendert die **erste** `<Route>`, deren `path` übereinstimmt.
- **`<Route>` (Die "Streckendefinition")**: Diese Komponente definiert eine einzelne Regel nach dem Muster: "Wenn die URL diesem `path` entspricht, dann rendere dieses `element`."

Beispiel für eine einfache Routen-Konfiguration in `src/App.jsx`: Zuerst erstellen wir ein paar einfache "Seiten"-Komponenten zum Anzeigen.

`src/pages/HomePage.jsx`:

```
function HomePage() {
  return <h1>Willkommen auf der Startseite!</h1>;
}
export default HomePage;
```

`src/pages/AboutPage.jsx`:

```
function AboutPage() {
  return <h1>Über Uns</h1>;
}
export default AboutPage;
```

Jetzt konfigurieren wir die Routen in `src/App.jsx`:

```
// src/App.jsx
import { Routes, Route } from 'react-router-dom'; // Importieren
import HomePage from './pages/HomePage';
import AboutPage from './pages/AboutPage';

function App() {
  return (
    <div>
      {/* Hier könnte eine globale Navigation stehen */}
      <h1>Meine Anwendung</h1>

      {/* Der Routes-Container bestimmt, welche Seite angezeigt wird */}
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/about" element={<AboutPage />} />
      </Routes>

      {/* Hier könnte ein globaler Footer stehen */}
    </div>
  );
}
export default App;
```

Wenn man jetzt im Browser `/` aufruft, wird die `HomePage` angezeigt. Ruft man `/about` auf, wird die `AboutPage` angezeigt.

4. Zwischen Seiten navigieren mit `<Link>`

Um dem Benutzer zu ermöglichen, zwischen den Seiten zu wechseln, verwenden wir die `<Link>`-Komponente anstelle eines normalen `<a>`-Tags.

- **Was sie macht**: Die `<Link>`-Komponente rendert im Hintergrund einen normalen `<a>`-Tag, aber sie verhindert dessen Standardverhalten (das Neuladen der Seite). Stattdessen aktualisiert sie nur die URL in der Browser-Leiste und überlässt es `<Routes>`, die richtige Komponente anzuzeigen.
- **Syntax**: Sie verwendet das `to`-Attribut anstelle von `href`.

Beispiel: Eine Navigationskomponente `src/components/Navigation.jsx`:

```
// src/components/Navigation.jsx
import { Link } from 'react-router-dom';

function Navigation() {
  return (
    <nav>
      <ul>
        <li><Link to="/">Startseite</Link></li>
        <li><Link to="/about">Über Uns</Link></li>
      </ul>
    </nav>
  );
}
export default Navigation;
```

Diese Komponente kann dann in `App.jsx` über den `<Routes>` platziert werden, um auf jeder "Seite" sichtbar zu sein.

5. Dynamische Routen mit URL-Parametern (`useParams`)

Wir wollen nicht nur statische Seiten, sondern auch Detailseiten für spezifische Objekte, z.B. `/rezepte/123`.

- **Route mit Platzhalter definieren**: In der `<Route>`-Definition verwenden wir einen Doppelpunkt `:`, um einen dynamischen Teil zu markieren.

  ```
  // in App.jsx
  <Route path="/rezepte/:recipeId" element={<RecipeDetailPage />} />
  ```

  Dieser Pfad passt auf `/rezepte/1`, `/rezepte/42` oder `/rezepte/irgendwas`.

- **Parameter auslesen mit dem `useParams`-Hook**: Innerhalb der Komponente, die von der dynamischen Route gerendert wird (hier `RecipeDetailPage`), können wir den `useParams`-Hook verwenden, um auf den Wert des Parameters zuzugreifen.

Beispiel `src/pages/RecipeDetailPage.jsx`:

```
// src/pages/RecipeDetailPage.jsx
import { useParams } from 'react-router-dom';
import { useState, useEffect } from 'react';

function RecipeDetailPage() {
  // useParams gibt ein Objekt zurück, z.B. { recipeId: '123' }
  const { recipeId } = useParams();

  const [recipe, setRecipe] = useState(null);

  // useEffect, um die Daten für dieses spezifische Rezept zu laden
  useEffect(() => {
    // API-Anfrage an z.B. /api/recipes/123/
```

```
    fetch(`http://127.0.0.1:8000/api/recipes/${recipeId}/`)
      .then(res => res.json())
      .then(data => setRecipe(data));
  }, [recipeId]); // Effekt neu ausführen, wenn sich die recipeId ändert

  if (!recipe) {
    return <p>Lade Rezept...</p>;
  }

  return (
    <div>
      <h1>{recipe.title}</h1>
      <p>{recipe.description}</p>
    </div>
  );
}
export default RecipeDetailPage;
```

6. Programmatische Navigation mit `useNavigate`

Manchmal möchten wir den Benutzer per Code auf eine andere Seite weiterleiten, z.B. nach einer erfolgreichen Formular-Absendung.

- **Der `useNavigate`-Hook**: Dieser Hook gibt eine Funktion zurück, die wir aufrufen können, um zu einem neuen Pfad zu navigieren.

Beispiel in einer `AddRecipeForm`-Komponente:

```
import { useNavigate } from 'react-router-dom';

function AddRecipeForm() {
  const navigate = useNavigate(); // Hook aufrufen, um die navigate-Funktion zu erhalten

  async function handleSubmit(event) {
    event.preventDefault();
    // ... Logik zum Senden der POST-Anfrage an die API ...

    // Annahme: API gibt das neu erstellte Rezept zurück
    // const newRecipe = await response.json();

    // Nach erfolgreicher Erstellung zur Detailseite des neuen Rezepts weiterleiten
    // navigate(`/rezepte/${newRecipe.id}`);

    // Oder einfach zurück zur Liste
    navigate('/rezepte');
  }

  return <form onSubmit={handleSubmit}> {/* ... Formular ... */} </form>;
}
```

Fazit

- **Client-Side Routing**: Verhindert das Neuladen der Seite und ermöglicht ein flüssiges SPA-Erlebnis.
- **`react-router-dom`**: Die Standardlösung für Routing in React.
- **Kernkomponenten**: `<BrowserRouter>` aktiviert das Routing, `<Routes>` wählt die passende Route aus, und `<Route>` definiert eine Regel (`path` -> `element`).
- **`<Link>`**: Die React-konforme Alternative zu `<a>`-Tags für die Navigation.
- **Hooks für dynamisches Routing**:
  - `useParams`: Zum Auslesen von Parametern aus der URL (z.B. IDs).
  - `useNavigate`: Für programmgesteuerte Weiterleitungen.

Cheat Sheet

- **Installation**:

```
npm install react-router-dom
```

- **Setup (`main.jsx`)**:

```
import { BrowserRouter } from 'react-router-dom';
// <BrowserRouter><App /></BrowserRouter>
```

- **Routen definieren (`App.jsx`)**:

```
import { Routes, Route } from 'react-router-dom';
// <Routes>
//   <Route path="/" element={<HomePage />} />
//   <Route path="/about" element={<AboutPage />} />
//   <Route path="/items/:itemId" element={<ItemDetailPage />} />
// </Routes>
```

- **Navigieren (`Link`)**:

```
import { Link } from 'react-router-dom';
// <Link to="/about">Über Uns</Link>
// <Link to={`/items/${item.id}`}>Details</Link>
```

- **Hooks**:
  - **Parameter auslesen**: `import { useParams } from 'react-router-dom'; const { itemId } = useParams();`
  - **Weiterleiten**: `import { useNavigate } from 'react-router-dom'; const navigate = useNavigate(); navigate('/new-path');`

Übungsaufgaben

Erstelle eine kleine "Mini-Blog"-Anwendung mit drei Seiten.

1. **Komponenten und Routen erstellen**:
   - Erstelle drei einfache Seiten-Komponenten: `BlogListPage`, `PostDetailPage` und `AuthorProfilePage`.
   - Richte in `App.jsx` die Routen ein:
     - `/blog` soll `BlogListPage` anzeigen.
     - `/posts/:postId` soll `PostDetailPage` anzeigen.
     - `/authors/:authorName` soll `AuthorProfilePage` anzeigen.
2. **Navigation hinzufügen**:
   - Erstelle eine `Navigation`-Komponente mit `<Link>`-Tags, die zur Blog-Liste (`/blog`) und zu einem Beispiel-Autorenprofil (`/authors/Max`) verlinken.
3. **Parameter auslesen**:
   - In `PostDetailPage`, verwende `useParams`, um die `postId` aus der URL auszulesen und anzuzeigen ("Details für Post mit ID: ...").
   - In `AuthorProfilePage`, verwende `useParams`, um den `authorName` auszulesen und anzuzeigen ("Profil von Autor: ...").
4. **Listen-Seite mit Links**:
   - In `BlogListPage`, erstelle eine (hartcodierte) Liste von Blog-Posts (z.B. `{id: 1, title: 'Mein erster Post'}, {id: 2, title: 'React ist toll'}`).
   - Iteriere über die Liste und rendere für jeden Post einen `<Link>`, der zur jeweiligen Detailseite verlinkt (z.B. `<Link to={/posts/${post.id}}>...</Link>`).

Schüler-Projekt (Eigenständig): Community Recipe Sharing Platform

Wir bauen die Navigation in unsere Rezept-Plattform ein, sodass Benutzer zwischen der Listen- und einer neuen Detailansicht wechseln können.

Aufgabe:

1. `react-router-dom` **installieren**:

   ```
   npm install react-router-dom
   ```

2. `BrowserRouter` **in** `main.jsx` **einrichten**: Umschließe deine `<App />`-Komponente mit dem `<BrowserRouter>`.
3. **Seiten-Komponenten erstellen**:
   - Benenne deine bestehende `RecipeList`-Komponente (oder die Komponente, die die Liste anzeigt) in `pages/RecipeListPage.jsx` um.
   - Erstelle eine neue "Seiten"-Komponente `pages/RecipeDetailPage.jsx`. Diese wird die Details eines einzelnen Rezepts anzeigen.
   - Erstelle eine weitere "Seiten"-Komponente `pages/AddRecipePage.jsx`, die dein `AddRecipeForm` enthält.
4. **Routen in** `App.jsx` **definieren**:
   - Importiere `Routes`, `Route` und deine neuen Seiten-Komponenten.
   - Definiere die folgenden Routen innerhalb von `<Routes>`:
     - `path="/"` soll die `RecipeListPage` rendern.
     - `path="/rezepte/neu"` soll die `AddRecipePage` rendern.
     - `path="/rezepte/:recipeId"` soll die `RecipeDetailPage` rendern.
5. `RecipeListPage` **anpassen**:
   - Ändere die `<li>`-Elemente in deiner Rezeptliste so, dass der Titel des Rezepts ein `<Link>` ist, der zur korrekten Detailseite führt.

     ```
     // Beispiel
     <Link to={`/rezepte/${recipe.id}`}>{recipe.title}</Link>
     ```

6. `RecipeDetailPage`**-Komponente implementieren**:
   - In `RecipeDetailPage`, verwende den `useParams`-Hook, um die `recipeId` aus der URL zu extrahieren.
   - Verwende `useEffect` und `useState`, um die Daten für **nur dieses eine Rezept** von deiner API abzurufen (vom Endpunkt `/api/recipes/{recipeId}/`).
   - Zeige alle Details des Rezepts an (Titel, Beschreibung, Bild etc.).
7. **Navigation einrichten**:
   - Erstelle eine `Navigation`-Komponente mit `<Link>`-Tags, die auf die Startseite (`/`) und die "Rezept hinzufügen"-Seite (`/rezepte/neu`) verweisen.
   - Binde diese `Navigation`-Komponente in `App.jsx` ein, sodass sie auf allen Seiten sichtbar ist.
8. **(Optional) Weiterleitung nach Erstellung**:
   - Passe die `AddRecipeForm`-Komponente an. Importiere `useNavigate` und leite den Benutzer nach erfolgreicher Erstellung eines Rezepts zur Detailseite des neuen Rezepts oder zurück zur Startseite weiter.