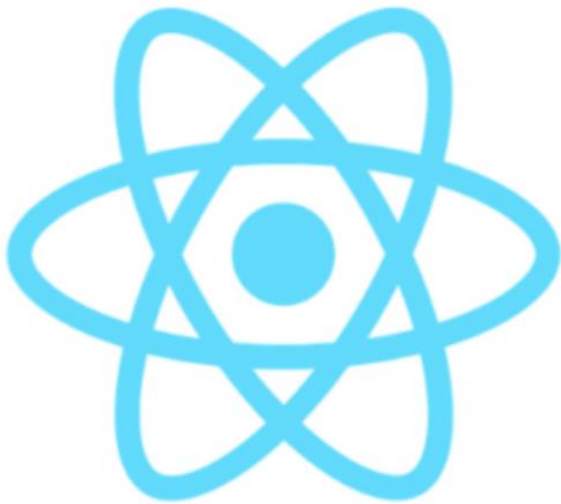


Full-stack Application Development

SPA Routing with React Router

Where to Find The Code and Materials?

<https://github.com/iproduct/react-typescript-academy-2022>



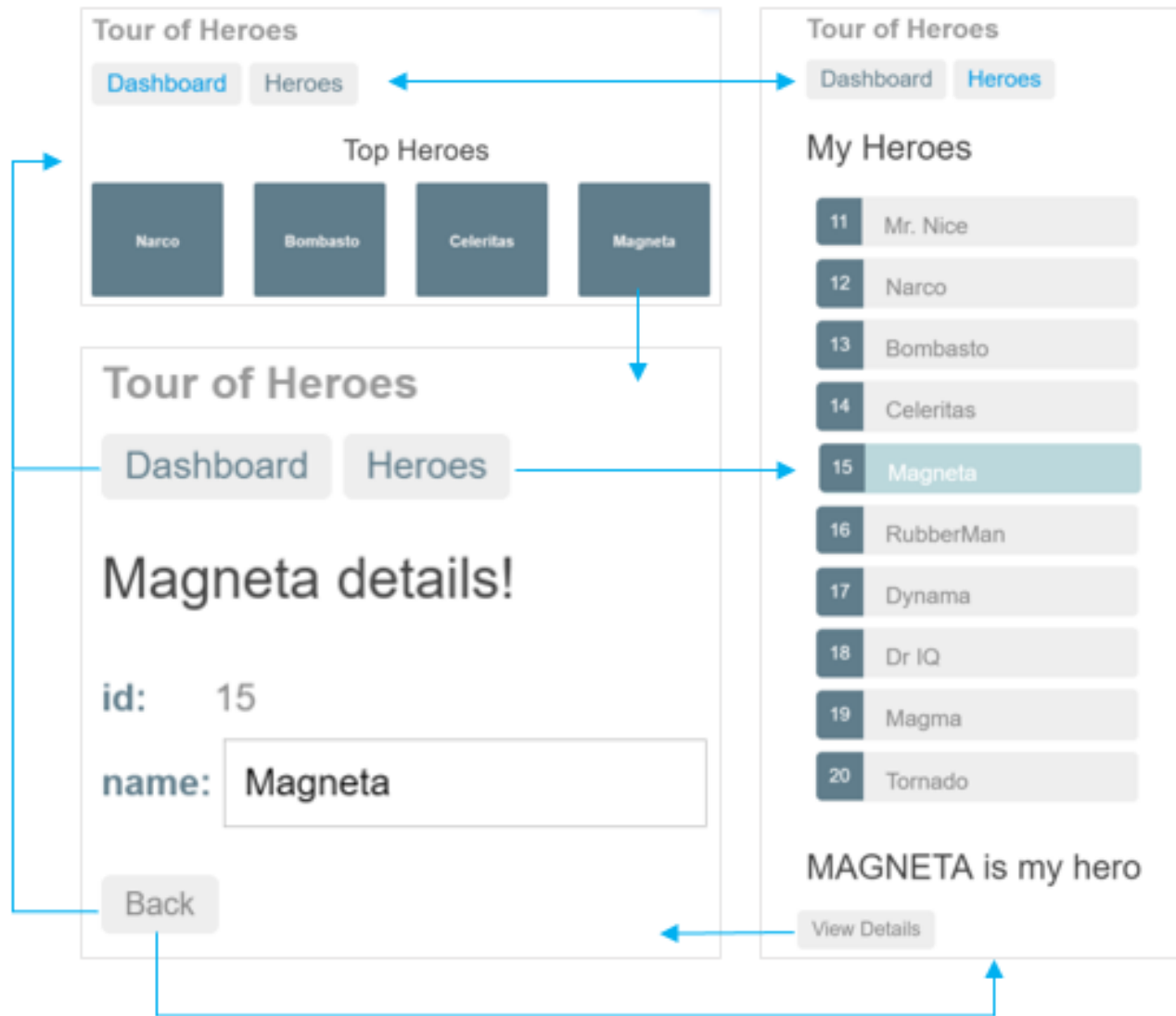
Agenda

1. Single Page Applications (SPA)
2. Why SPA?
3. Hierarchical Routing
4. Router Outlets
5. Basic Routing using React Router v6.4+
6. Nested Routing & Params using Router v6.4+
7. React Router Configuration
8. Site Navigation using Router
9. Programmatic Navigation using Router
10. Login Demo with Redirection

Contemporary Web Applications

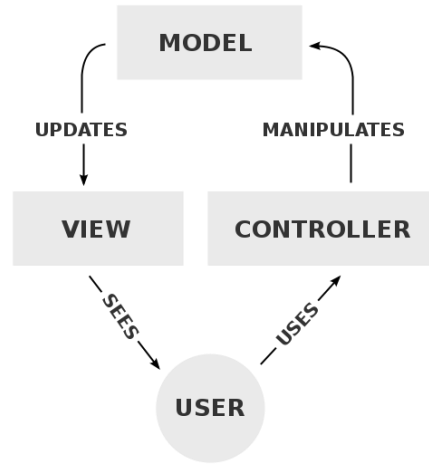
- Provide better **User Experience (UX)** by:
 - more interactive
 - loading and reacting faster in response (or even anticipation) of user's moves
 - able to work offline
 - supporting multiple devices and screen resolutions (responsive design)
 - are following design metaphors consistently (e.g. **Google Material Design - MD**)
 - looking more like desktop application than static web page

Single Page Applications (SPA)

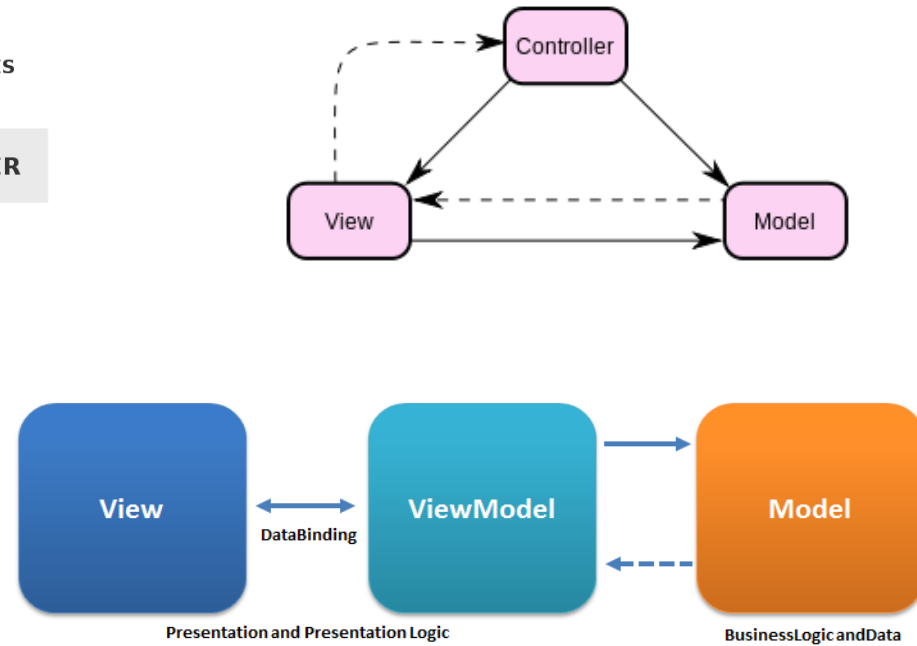


MVC Comes in Different Flavors

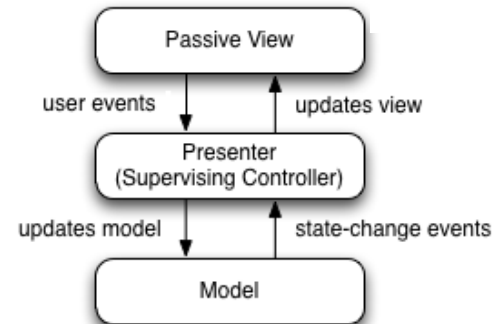
- MVC



- MVVM



- MVP



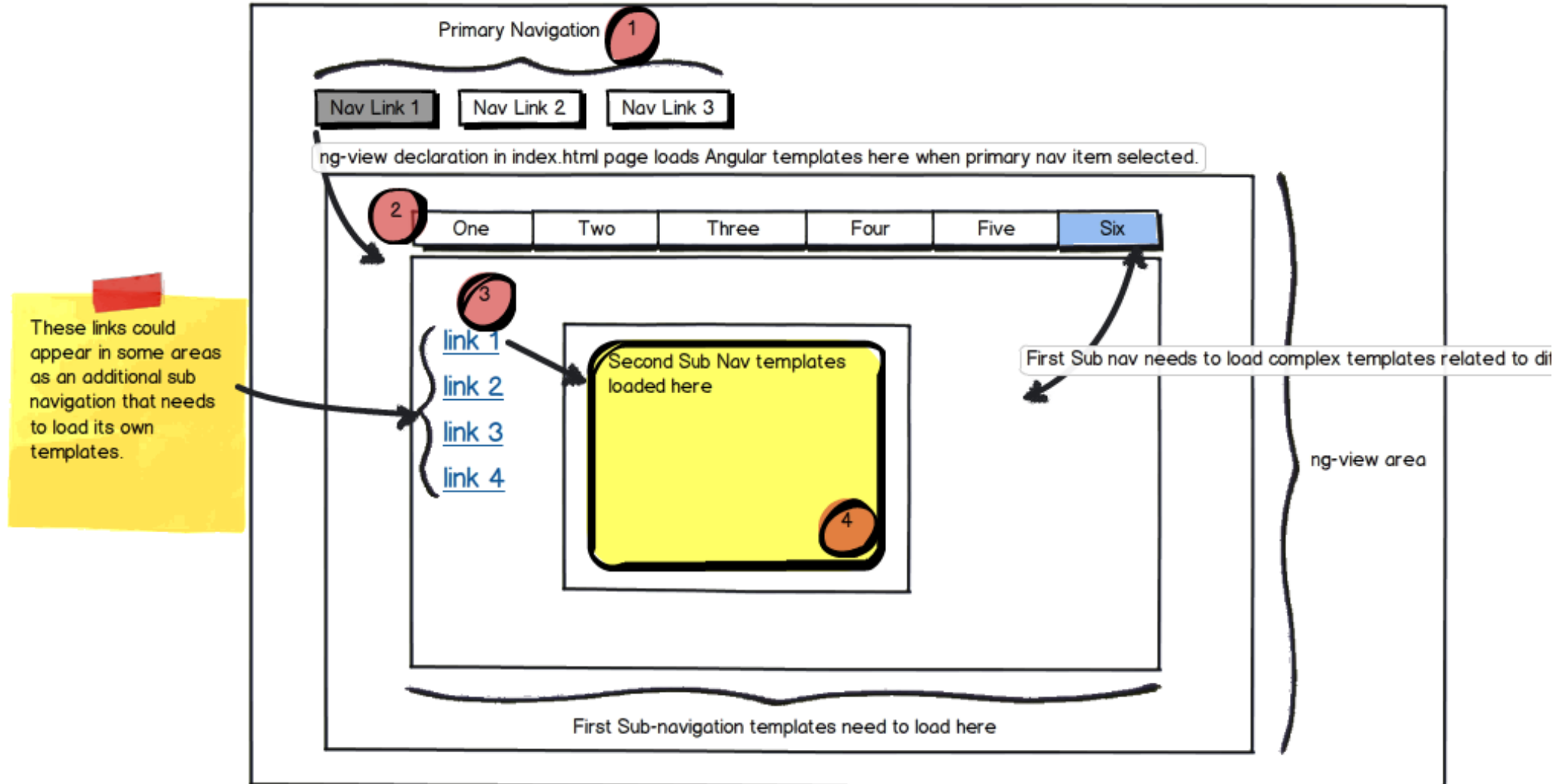
Why SPA?

- Page does not flicker – seamless (or even animated) transitions
- Less data transferred – responses are cached
- Only raw data, not markup
- Features can be loaded on demand (lazy) or in background
- Most page processing happens on the client offloading the server: REST data services + snapshots for crawlers (SEO)
- Code reuse – REST endpoints are general purpose
- Supporting multiple platforms (Web, iOS, Android) → React Native

Developing Single Page Apps (SPA) in steps

- 1) Setting up a build system – *npm, webpack, etc.*
- 2) Designing front-end architecture components – *views & layouts + view models* (presentation data models) + *presentation logic* (event handling, messaging) + *routing paths* (essential for SPA)
- 3) Better to use component model to boost productivity and maintainability.
- 4) End-to-end application design – front-end: wireframes → views,
- 5) data entities & data streams → service API clients and models design,
- 6) **sitemap → router config**

Hierarchical Routing



SPA with Multiple Router Outlets

Root

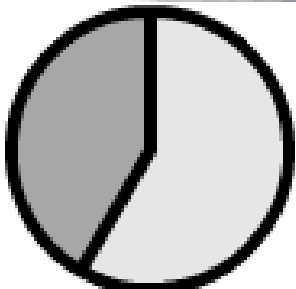
filters

Type: Age Range to Date:

tabledata

Name (job title) ▲	Age	Nickname	Employee
Giacomo Guilizzoni Founder & CEO	34	Peldi	<input checked="" type="checkbox"/>
Guido Jack Guilizzoni	4	The Guida	<input type="checkbox"/>
Marco Botton Tuttofare	31		<input checked="" type="checkbox"/>
Mariah MacLachlan Better Half	35	Potata	<input checked="" type="checkbox"/>
Valerie Liberty COO, WOW! Division	:)	Val	<input checked="" type="checkbox"/>

graph



Getting Started with React Router v6.4+

- Create new project using *create-react-app*:

```
npx create-react-app demo-app --template=typescript  
cd demo-app
```

- Install *react-router-dom*:

```
npm install react-router-dom    OR  
yarn add react-router-dom
```

- Implement routing in *src/App.tsx*

Basic Routing using React Router v6.4+

```
import React from "react";
import { createRoot } from "react-dom/client";
import {
  createBrowserRouter,
  RouterProvider,
  Route,
  Link,
} from "react-router-dom";

const router = createBrowserRouter([
  {
    path: "/",
    element: (
      <div>
        <h1>Hello World</h1>
        <Link to="about">About Us</Link>
      </div>
    ),
  },
  {
    path: "about",
    element: <div>About</div>,
  },
]);

createRoot(document.getElementById("root")!).render(
  <RouterProvider router={router} />
);
```

React Router Main Concepts I

<https://reactrouter.com/en/main/start/concepts>

- **URL** - The URL in the address bar. A lot of people use the term "URL" and "route" interchangeably, but this is not a route in React Router, it's just a URL.
- **Location** - This is a React Router specific object that is based on the built-in browser's `window.location` object. It represents "where the user is at". It's mostly an object representation of the URL but has a bit more to it than that.
- **Location State** - A value that persists with a location that **isn't encoded in the URL**. Much like hash or search params (data encoded in the URL), but stored invisibly in the browser's memory.

React Router Main Concepts II

- **History Stack** - As the user navigates, the browser keeps track of each location in a stack. If you click and hold the back button in a browser you can see the browser's history stack right there.
- **Client Side Routing (CSR)** - A plain HTML document can link to other documents and the browser handles the history stack itself. Client Side Routing enables developers to manipulate the browser history stack **without making a document request to the server**.
- **History** - An object that allows React Router to subscribe to changes in the URL as well as providing APIs to manipulate the browser history stack programmatically.

React Router Main Concepts III

- **History Action** - One of **POP**, **PUSH**, or **REPLACE**. Users can arrive at a URL for one of these three reasons. A push when a new entry is added to the history stack (typically a link click or the programmer forced a navigation). A replace is similar except it replaces the current entry on the stack instead of pushing a new one. Finally, a pop happens when the user clicks the back or forward buttons in the browser chrome.
- **Segment** - The parts of a URL or path pattern between the **/** characters. For example, **"/users/123"** has two segments.
- **Path Pattern** - These look like URLs but can have special characters for matching URLs to routes, like dynamic segments (**"/users/:userId"**) or star segments (**"/docs/*"**). They aren't URLs, they're patterns that React Router will match.

React Router Main Concepts IV

- **Dynamic Segment** - A segment of a path pattern that is dynamic, meaning it can match any values in the segment. For example the pattern `/users/:userId` will match URLs like `/users/123`
- **URL Params** - The parsed values from the URL that matched a dynamic segment.
- **Router** - Stateful, top-level component that makes all the other components and hooks work.
- **Route Config** - A tree of routes objects that will be ranked and matched (**with nesting**) against the current location to create a branch of route matches.

React Router Main Concepts V

- **Route** - An object or Route Element typically with a shape of { path, element } or <Route path element>. The path is a path pattern. When the path pattern matches the current URL, the element will be rendered.
- **Route Element** - Or <Route>. This element's props are read to create a route by <Routes>, but otherwise does nothing.
- **Nested Routes** - Because routes can have children and each route defines a portion of the URL through segments, a single URL can match multiple routes in a nested "branch" of the tree. This enables automatic layout nesting through outlet, relative links, and more.

React Router Main Concepts VI

- **Relative links** - Links that **don't start with /** will **inherit the closest route in which they are rendered**. This makes it easy to link to deeper URLs without having to know and build up the entire path.
- **Match** - An object that holds information when a route matches the URL, like the **url params** and **pathname** that matched.
- **Matches** - An **array of routes** (or branch of the **route config**) that **matches the current location**. This structure enables **nested routes**.
- **Parent Route** - A route with child routes.
- **Outlet** - A component that renders the next match in a set of matches.
- **Index Route** - A child route with no path that renders in the parent's outlet at the parent's URL.

React Router Main Concepts VII

- **Layout Route** - A parent route without a path, used exclusively for grouping child routes inside a specific layout.

Simple Navigation Using <Link />

```
<nav>
  <ul>
    <li>
      <Link to='/'>Home</Link>
    </li>
    <li>
      <Link to={`contacts/1`}>Your Name</Link>
    </li>
    <li>
      <Link to={`contacts/2`}>Your Friend</Link>
    </li>
    <li>
      <Link to='/about'>About</Link>
    </li>
  </ul>
</nav>
```

Better Navigation Using <NavLink /> and Active Css Class

```
<nav>
  <ul>
    <li>
      <NavLink to="/" end>
        {{{ isActive }} => (
          <span
            className={
              isActive ? 'active' : undefined
            }>Home</span>
        )}
      </NavLink>
    </li>
    <li>
      <NavLink to="/posts">
        {{{ isActive }} => (
          <span
            className={
              isActive ? 'active' : undefined
            }>Blog Posts</span>
        )}
      </NavLink>
    </li>
```

```
    <li>
      <NavLink to="/contacts/1">
        {{{ isActive }} => (
          <span
            className={
              isActive ? 'active' : undefined
            }>Your Name</span>
        )}
      </NavLink>
    </li>
    <li>
      <NavLink to="/contacts/2">
        {{{ isActive }} => (
          <span
            className={
              isActive ? 'active' : undefined
            }>Your Friend</span>
        )}
      </NavLink>
    </li>
  </ul>
</nav>
```

Data Router Feature of React Router v6.4+

```
const router = createBrowserRouter([
  {
    path: "/",
    element: <RootPage />,
    errorElement: <ErrorPage />,
    children: [{
      errorElement: <ErrorPage />,
      children: [{
        index: true,
        element: <HomePage />,
      }, {
        path: "contacts/:contactId",
        element: <ContactPage />,
      },
    ]
  },
  {
    path: "posts",
    loader: postsLoader,
    element: <PostsPage />,
    children: [{
      errorElement: <ErrorPage />,
      path: ":postId",
      action: postAction,
      loader: postLoader,
      element: <PostPage />,
    }],
  },
  {
    path: "*",
    element: <ErrorPage />,
  },
]);
```

```
{
  path: "posts",
  loader: postsLoader,
  element: <PostsPage />,
  children: [{
    errorElement: <ErrorPage />,
    path: ":postId",
    action: postAction,
    loader: postLoader,
    element: <PostPage />,
  }],
}, {
  path: "*",
  element: <ErrorPage />,
},
]);
```

Loaders = Read

```
export async function postsLoader({ request }: LoaderFunctionArgs) {  
  const url = new URL(request.url);  
  const q = url.searchParams.get('q');  
  if (q) {  
    return PostsApi.findByTitleLike(q);  
  } else {  
    return PostsApi.findAll();  
  }  
}
```

```
export function postLoader({ params }: LoaderFunctionArgs ) {  
  if (params.postId) {  
    return PostsApi.findById(+params.postId);  
  } else {  
    throw new Error(`Invalid or missing post ID`);  
  }  
}
```

Actions = Create, Update, Delete

```
export async function postAction({ request, params }: ActionFunctionArgs) {  
  if (request.method === 'DELETE') {  
    params.postId && await PostsApi.deleteById(+params.postId);  
    return redirect('/posts');  
  } else if (request.method === 'POST') {  
    let formData = await request.formData();  
    let favorite = formData.get('favorite');  
    console.log(favorite);  
    if (favorite !== null && params.postId) {  
      return PostsApi.patchById(+params.postId, {favorite: (favorite ? favorite === 'false': undefined)});  
    }  
  }  
}
```


useLoaderData hook

```
import React, { useEffect, useState } from 'react'
import { Outlet, useLoaderData } from 'react-router-dom'
import PostList from '../components/PostList'
import { Post } from '../model/posts'
import { PostsApi } from '../service/rest-api-client'
```

```
export const PostsPage = () => {
  // const [posts, setPosts] = useState<Post[]>([])
  // useEffect(() => {
  //   PostsApi.findAll().then(posts => {
  //     setPosts(posts)
  //   })
  // }, [])
  const posts = useLoaderData() as Post[];

  return (
    <>
      <PostList posts={posts} filter={undefined} onDeletePost={() => { }} onEditPost={() => { }} />
      <div className="PostsPage-article">
        <Outlet />
      </div>
    </>
  )
}
```

Using `<Form />` and `action`

```
export default function PostPage() {
  const post = useLoaderData() as Post;

  return (
    <div id="contact">
      <div>
        <img className="PostPage-img"
          key={post?.id}
          src={post?.imageUrl}
          alt="contact avatar"
        />
      </div>

      <div>
        <h1>
          {post?.id}:
          {post?.title}
          {post && <Favorite post={post} />}
        </h1>

        {post?.content && <p>{post?.content}</p>}
      </div>
    </div>
  );
}
```

```
<div>
  <Form method="put">
    <button type="submit">Edit</button>
  </Form>
  <Form
    method="delete"
    onSubmit={(event) => {
      if (
        // eslint-disable-next-line no-restricted-globals
        !confirm(
          "Please confirm you want to delete this record."
        )
      ) {
        event.preventDefault();
      }
    }}
  >
    <button type="submit">Delete</button>
  </Form>
</div>
</div>
```

Thank's for Your Attention!



Trayan Iliev

IPT – Intellectual Products & Technologies

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>