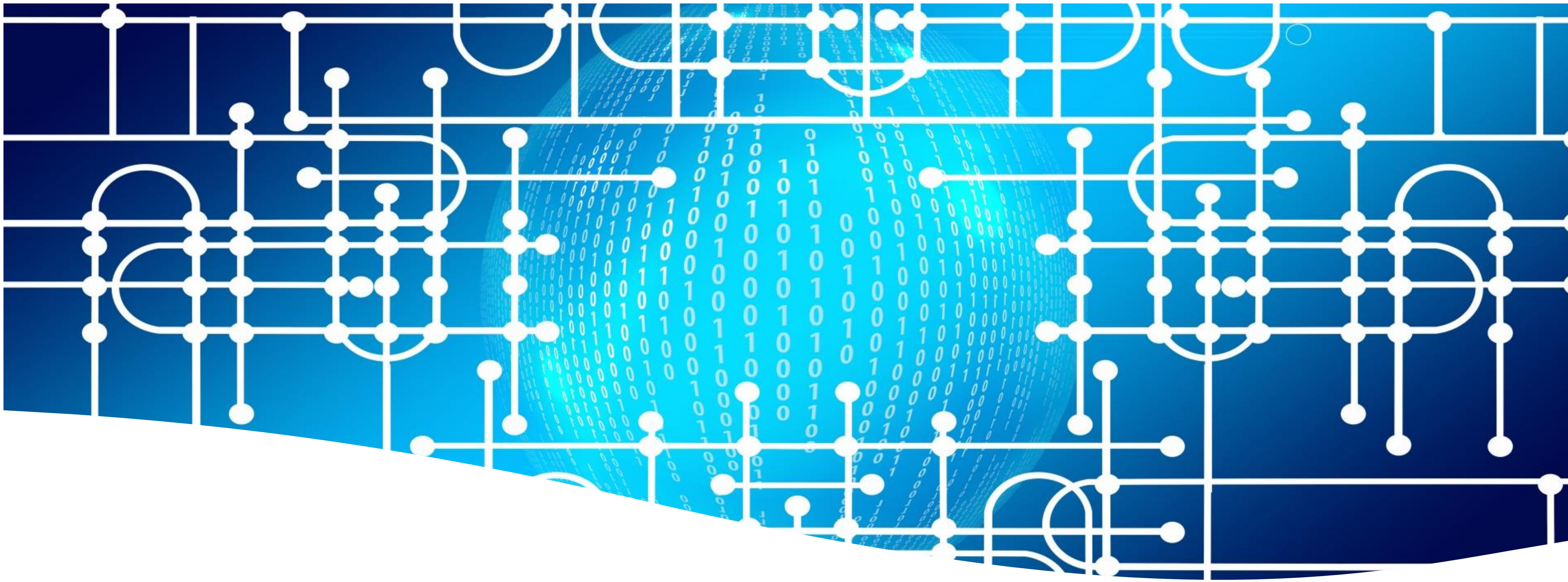


Full-stack Application Development

SPA Routing with React Router

Copyright © 2003-2022 IPT – Intellectual Products & Technologies Ltd. Licensed under CC BY NC SA 4.0 license.

Where to Find The Code and Materials?

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

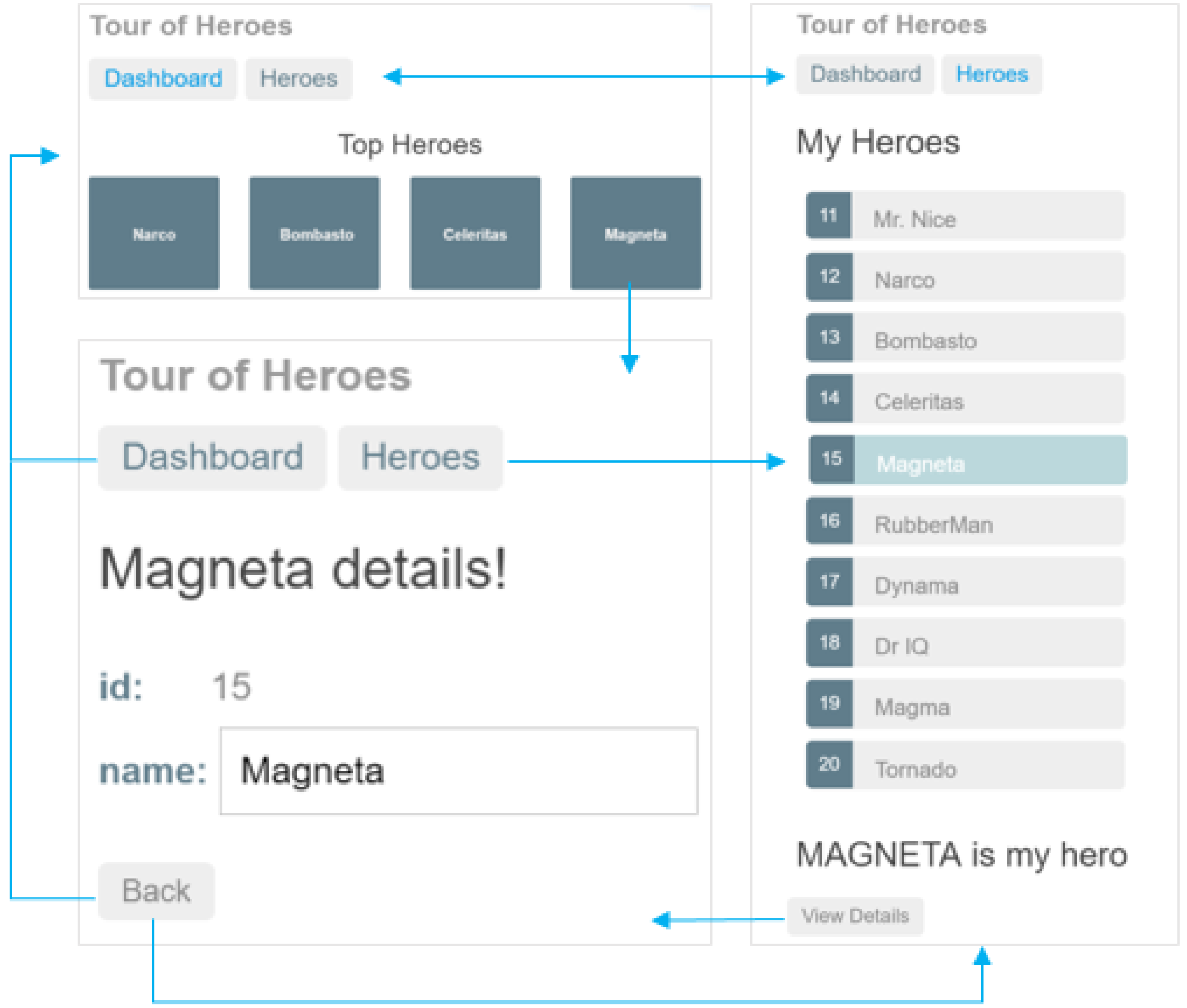
Agenda

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

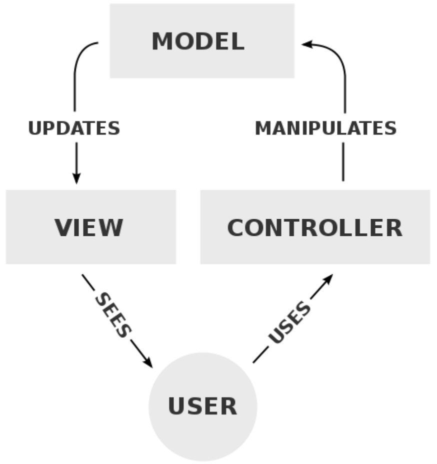
Single Page Applications (SPA)

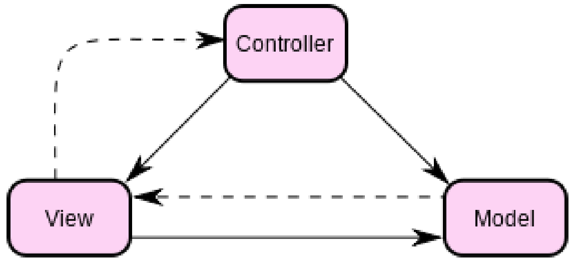


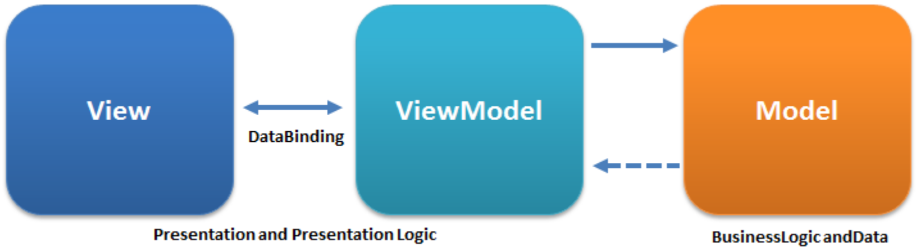
Source: Angular 2 Tutorial: Routing https://angular.io/docs/ts/latest/tutorial/toh-pt5.html
License: CC BY 4.0.

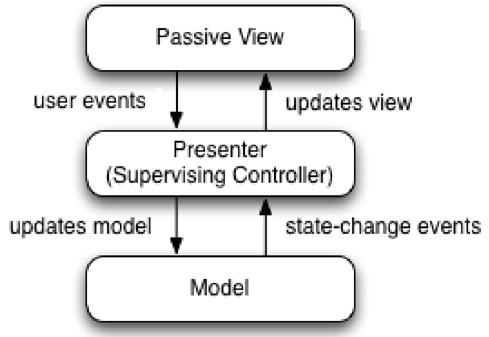
MVC Comes in Different Flavors

- MVC
- MVVM
- MVP









Sources: https://en.wikipedia.org/wiki/Model_View_ViewModel#/media/File:MVVMPattern.png,
https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter#/media/File:Model_View_Presenter_GUI_Design_Pattern.png
License: CC BY-SA 3.0, Authors:Ugaya40, Daniel.Cardenas

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 + snapshops for crawlers (SEO)
- Code reuse – REST endopints are general purpose
- Supporting multiple platforms (Web, iOS, Android) → React Native

Developing Sinagle 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)

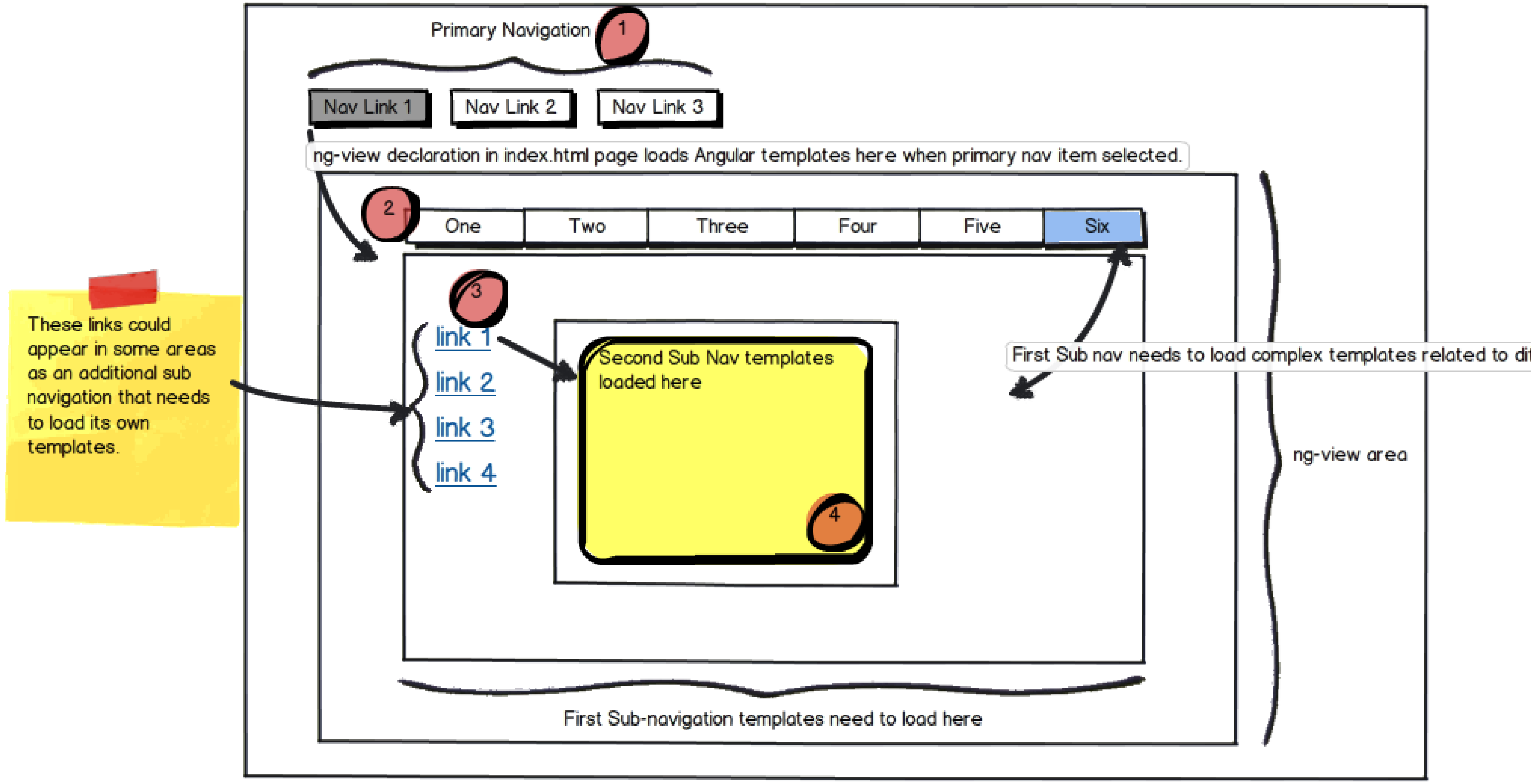
3)Better to use component model to boost productivity and maintainability.

4)End-to-end application design – front-end: wireframes → views,

5)data entities & data streams → service API clients and models design,

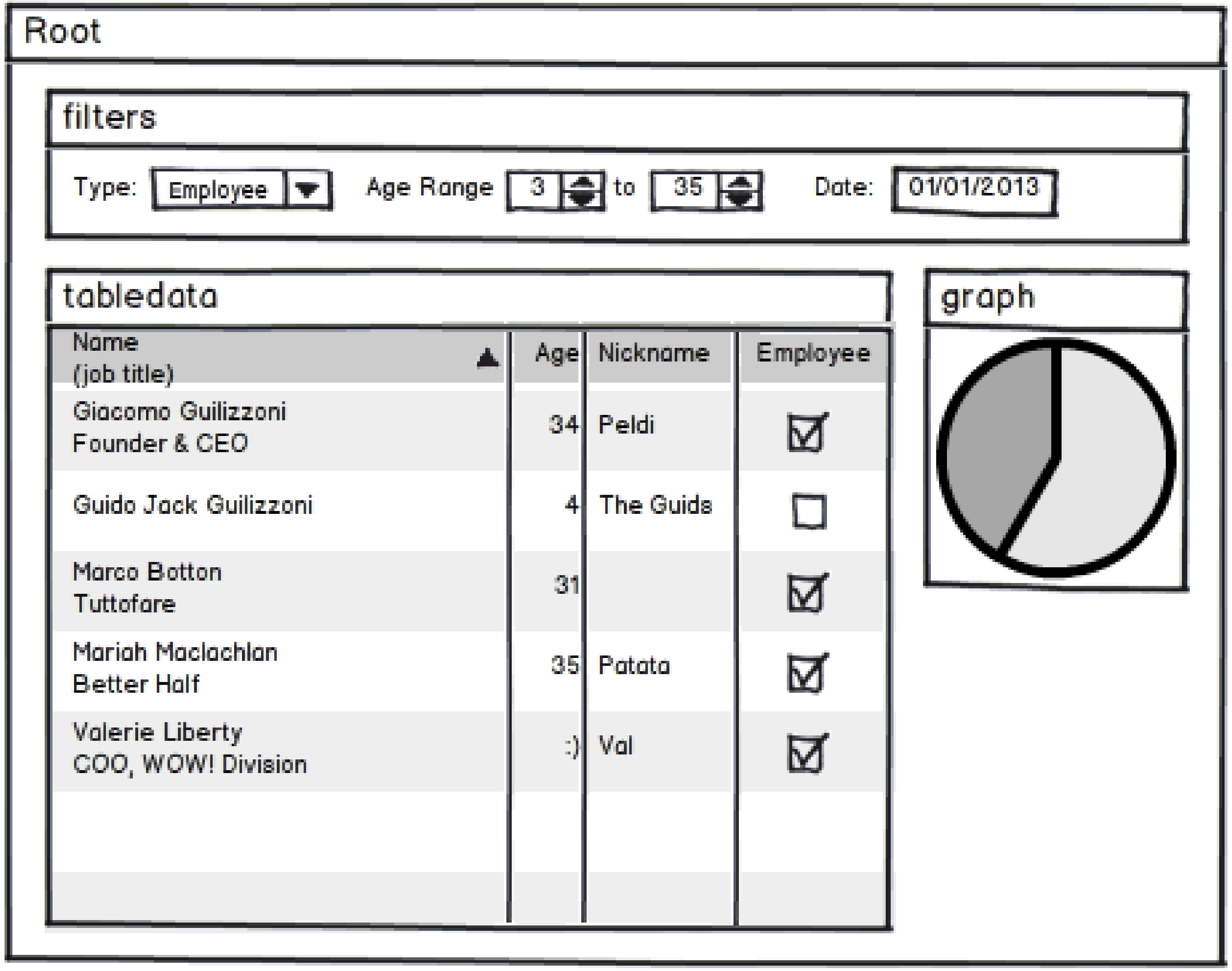
6)sitemap → router config

Hierarchical Routing


Primary Navigation
1
Nav Link 1
Nav Link 2
Nav Link 3
ng-view declaration in index.html page loads Angular templates here when primary nav item selected.
2
One
Two
Three
Four
Five
Six
3
link 1
link 2
link 3
link 4
Second Sub Nav templates loaded here
4
First Sub nav needs to load complex templates related to di
ng-view area
First Sub-navigation templates need to load here
These links could appear in some areas as an additional sub navigation that needs to load its own templates.

Source: http://stackoverflow.com/questions/12863663/complex-nesting-of-partials-and-templates
Author: PhillipKregg

SPA with Multiple Router Outlets



Name (job title)	Age	Nickname	Employee
Giacomo Guilizzoni Founder & CEO	34	Peldi	☑
Guido Jack Guilizzoni	4	The Guids	☐
Marco Botton Tuttofare	31		☑
Mariah Maclachlan Better Half	35	Patata	☑
Valerie Liberty COO, WOW! Division	:)	Val	☑

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.

Nested Routes

```
createBrowserRouter(
    createRoutesFromElements(
        <Route path="/" element={<Root />}>
            <Route path="contact" element={<Contact />} />
            <Route
                path="dashboard"
                element={<Dashboard />}
                loader={({ request }) =>
                    fetch("/api/dashboard.json", {
                        signal: request.signal,
                    })
                }
            />
            <Route element={<AuthLayout />}>
                <Route
                    path="login"
                    element={<Login />}
                    loader={redirectIfUser}
                />
                <Route path="logout" action={logoutUser}/>
            </Route>
        </Route>
    ));
```

Dynamic Segments: <Route path="projects/:projectId/tasks/:taskId" />

```
// If the current location is /projects/abc/tasks/3
<Route
    // sent to loaders
    loader={({ params }) => {
        params.projectId; // abc
        params.taskId; // 3
    }}
    // and actions
    action={({ params }) => {
        params.projectId; // abc
        params.taskId; // 3
    }}
    element={<Task />}
/>;

function Task() {
    // returned from `useParams`
    const params = useParams();
    params.projectId; // abc
    params.taskId; // 3
}
function Random() {
    const match = useMatch("/projects/:projectId/tasks/:taskId");
    match.params.projectId; // abc
    match.params.taskId; // 3
}
```

Active Links

```
<NavLink
    style={({isActive, isPending}) => {
        return {
            color: isActive ? "red" : "inherit",
        };
    }}
    className={({isActive, isPending}) => {
        return isActive ? "active" : isPending ? "pending" : "";
    }}
/>

//You can also useMatch for any other "active" indication outside of links.

function SomeComp() {
    const match = useMatch("/messages");
    return <li className={Boolean(match) ? "active" : ""}/>;
}
```

Data Loading

```
<Route
  path="/"
  loader={async ({ request }) => {
    // loaders can be async functions
    const res = await fetch("/api/user.json", {
      signal: request.signal,
    });
    const user = await res.json();
    return user;
  }}
  element={<Root />}
>
  <Route
    path=":teamId"
    // loaders understand Fetch Responses and will automatically
    // unwrap the res.json(), so you can simply return a fetch
    loader={({ params }) => {
      return fetch(`/api/teams/${params.teamId}`);
    }}
    element={<Team />}
  >
    <Route
      path=":gameId"
      loader={({ params }) => {
        // of course you can use any data store
        return fakeSdk.getTeam(params.gameId);
      }}
      element={<Game />}
    />
  </Route>
</Route>
```

Data Mutations

```
<Form action="/project/new">
    <label>
        Project title
        <br />
        <input type="text" name="title" />
    </label>

    <label>
        Target Finish Date
        <br />
        <input type="date" name="due" />
    </label>
</Form>
```

```
<Route
    path="project/new"
    action={async ({ request }) => {
        const formData = await request.formData();
        const newProject = await createProject({
            title: formData.get("title"),
            due: formData.get("due"),
        });
        return redirect(`/projects/${newProject.id}`);
    }}
/>
```

Redirects

```
<Route
  path="dashboard"
  loader={async () => {
    const user = await fake.getUser();
    if (!user) {
      // if you know you can't render the route, you can
      // throw a redirect to stop executing code here,
      // sending the user to a new route
      throw redirect("/login");
    }

    // otherwise continue
    const stats = await fake.getDashboardStats();
    return { user, stats };
  }}
/>
<Route
  path="project/new"
  action={async ({ request }) => {
    const data = await request.formData();
    const newProject = await createProject(data);
    // it's common to redirect after actions complete,
    // sending the user to the new record
    return redirect(`/projects/${newProject.id}`);
  }}
/>
```

Pending Navigation UI

```
function Root() {
    const navigation = useNavigation();
    return (
        <div>
            {navigation.state === "loading" && <GlobalSpinner />}
            <FakeSidebar />
            <Outlet />
            <FakeFooter />
        </div>
    );
}
```

Skeleton UI with <Suspense> : https://reactrouter.com/en/main/guides/deferred

```
async function loader({ params }) {
    const packageLocationPromise = getPackageLocation(
        params.packageId
    );

    return defer({
        packageLocation: packageLocationPromise,
    });
}
```

```
export default function PackageRoute() {
    const data = useLoaderData();

    return (
        <main>
            <h1>Let's locate your package</h1>
            <React.Suspense
                fallback={<p>Loading package location...</p>}
            >
                <Await
                    resolve={data.packageLocation}
                    errorElement={
                        <p>Error loading package location!</p>
                    }
                >
                    {(packageLocation) => (
                        <p>
                            Your package is at {packageLocation.latitude}{" "}
                            lat and {packageLocation.longitude} long.
                        </p>
                    )}
                </Await>
            </React.Suspense>
        </main>
    );
}
```

Skeleton UI with <Suspense> - Example 2

```
<Route
  path="issue/:issueId"
  element={<Issue />}
  loader={async ({ params }) => {
    // these are promises, but *not* awaited
    const comments = fake.getIssueComments(params.issueId);
    const history = fake.getIssueHistory(params.issueId);
    // the issue, however, *is* awaited
    const issue = await fake.getIssue(params.issueId);

    // defer enables suspense for the un-awaited promises
    return defer({ issue, comments, history });
  }}
/>;
```

```
function Issue() {
  const { issue, history, comments } = useLoaderData();
  return (
    <div>
      <IssueDescription issue={issue} />

      {/* Suspense provides the placeholder fallback */}
      <Suspense fallback={<IssueHistorySkeleton />}>
        {/* Await manages the deferred data (promise) */}
        <Await resolve={history}>
          {/* this calls back when the data is resolved */}
          {(resolvedHistory) => (
            <IssueHistory history={resolvedHistory} />
          )}
        </Await>
      </Suspense>

      <Suspense fallback={<IssueCommentsSkeleton />}>
        <Await resolve={comments}>
          {/* ... or you can use hooks to access the data */}
          <IssueComments />
        </Await>
      </Suspense>
    </div>
  );
}

function IssueComments() {
  const comments = useAsyncValue();
  return <div>{/* ... */}</div>;
}
```

Basic Routing using React Router v6 - I

```
function Layout() {
  return (
    <div>
      {/* A "layout route" is a good place to put markup you want to
          share across all the pages on your site, like navigation. */}
      <nav>
        <ul>
          <li>
            <Link to="/">Home</Link>
          </li>
          <li>
            <Link to="/about">About</Link>
          </li>
          <li>
            <Link to="/dashboard">Dashboard</Link>
          </li>
          <li>
            <Link to="/nothing-here">Nothing Here</Link>
          </li>
        </ul>
      </nav>
      <hr />
```

Basic Routing using React Router v6 - II

```
      {/* An <Outlet> renders whatever child route is currently active,
          so you can think about this <Outlet> as a placeholder for
          the child routes we defined above. */}
      <Outlet />
    </div>
  );
}

function Home() {
  return (
    <div>
      <h2>Home</h2>
    </div>
  );
}

function About() {
  return (
    <div>
      <h2>About</h2>
    </div>
  );
}
```

Basic Routing using React Router v6 - III

```
function Dashboard() {
  return (
    <div>
      <h2>Dashboard</h2>
    </div>
  );
}

function NoMatch() {
  return (
    <div>
      <h2>Nothing to see here!</h2>
      <p>
        <Link to="/">Go to the home page</Link>
      </p>
    </div>
  );
}
```

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 />,
      }]
    }]
  }]);
```

Using <Form /> with action and method=GET => loader

```
<Form action="/posts" id="search-form" role="search">
  <input
    id="q"
    aria-label="Search contacts"
    placeholder="Search"
    type="search"
    name="q"
  />
  <button type="submit">Search</button>
  <div
    id="search-spinner"
    aria-hidden
    hidden={true}
  />
  <div
    className="sr-only"
    aria-live="polite"
  ></div>
</Form>
```

Using <Form /> with implicit action and method POST / PUT / DELETE

```
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>
          <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>
```

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>
        </>
    )
}
```

Login Demo with Redirection

- There are 3 pages:
  - **public page** (demonstrating the public part of a web site)
  - **protected page** (demonstrating the private part of web site)
  - **login page**
- In order to see the protected page, you must login first. Upon login success, you will be redirected automatically to the required protected page.
- If you click the back button, would you expect to go back to the login page? No! You're already logged in. Going back, you should see the page you visited *before* logging in - the public page.

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