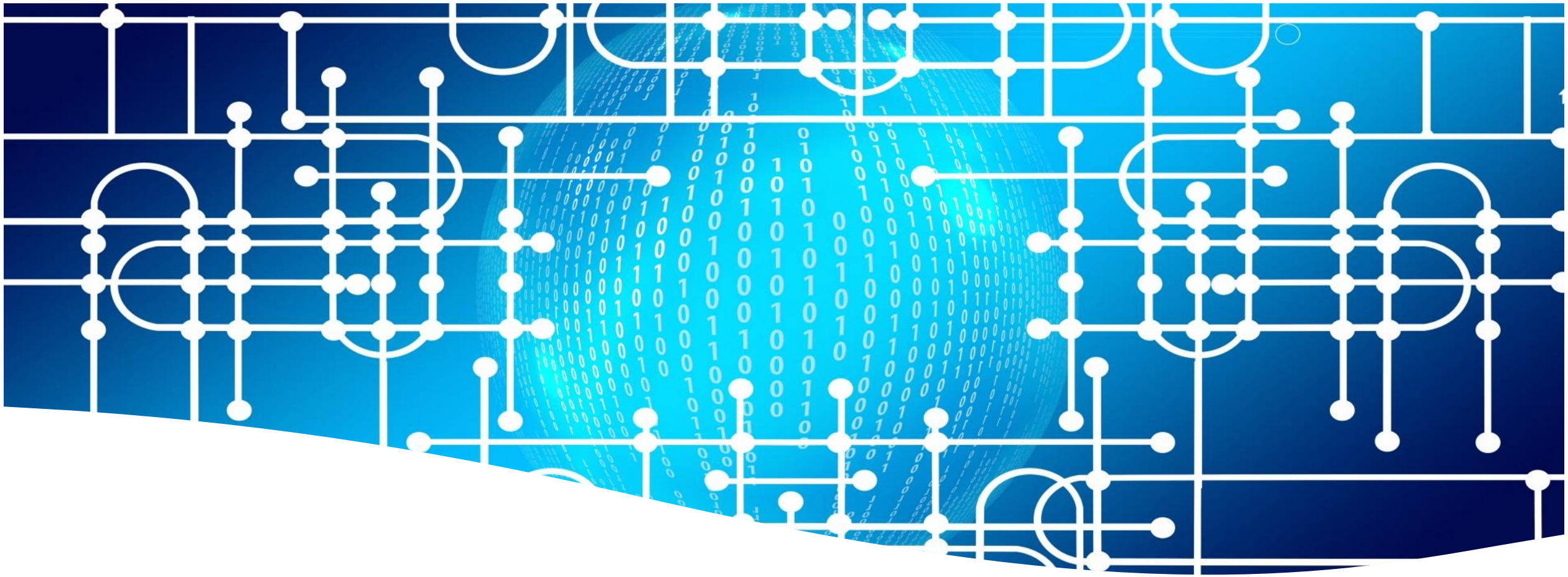
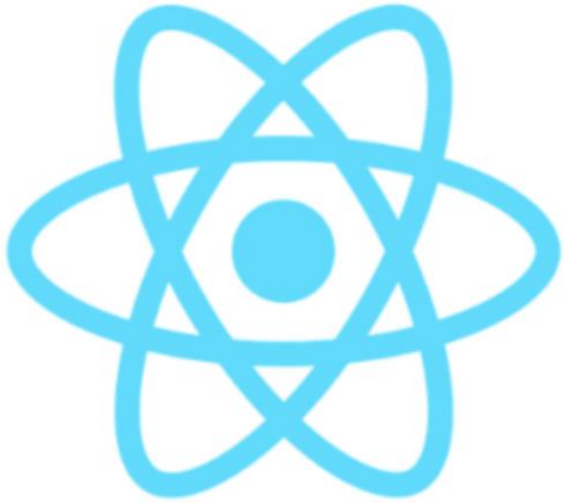


Full-stack Application Development

SPA Routing with React Router v5

Where to Find The Code and Materials?

https://github.com/iproduct/fullstack-typescript-react



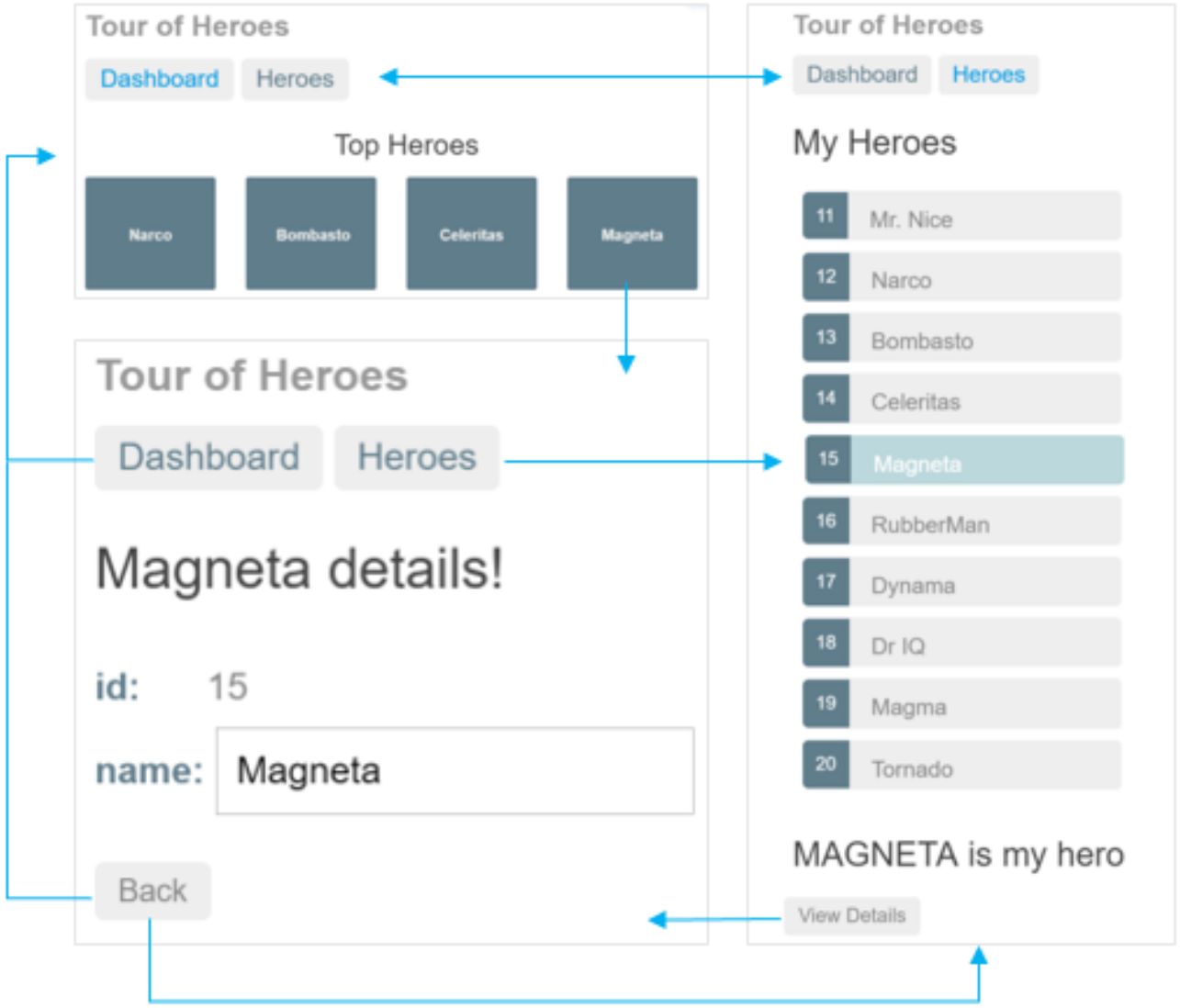
Agenda

1. Single Page Applications (SPA)
2. Why SPA?
3. Hierarchical Routing
4. SPA with Multiple Router Outlets
5. Basic Routing using React Router v5
6. Nested Routing & Params using Router v5
7. React Router Dynamic Configuration
8. Site Navigation using Router
9. Programmatic Navigation using Router
10. Using *withRouter* Decorator (HOC)
11. Login Demo with Redirection

Contemporary Web Applications

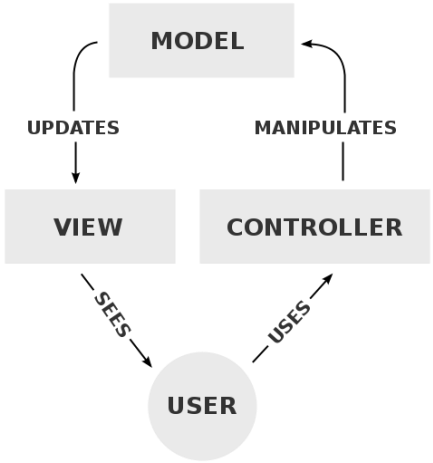
- Provide better **User Experience (UX)** by:
  - more interactive
  - loading and reacting faster in response (or even anticipation) of user's moves
  - able to work offline
  - supporting multiple devices and screen resolutions (responsive design)
  - are following design metaphors consistently (e.g.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

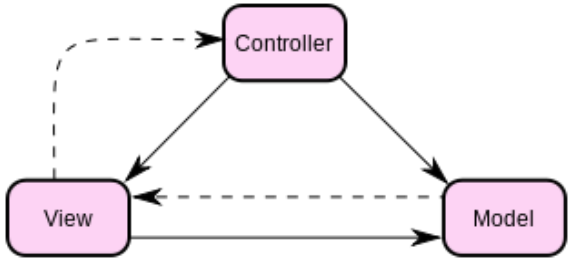
Single Page Applications (SPA)



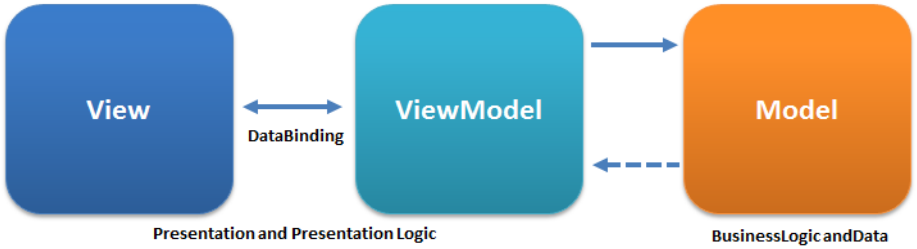
MVC Comes in Different Flavors

- MVC

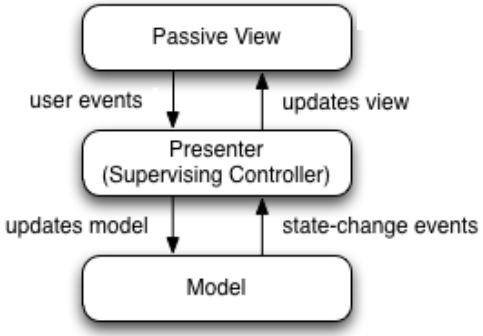




- MVVM



- MVP



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 3 steps

1) Setting up a build system – *npm, webpack, gulp* are common choices, *babel, typescript, JSX/TSX, CSS preprocessors (SASS, SCSS, LESS), jasmine, karma, protractor, live servers* ...

2) Designing front-end architecture components – *views & layouts* + *view models* (presentation data models) + *presentation logic* (event handling, messaging) + *routing paths* (essential for SPA)

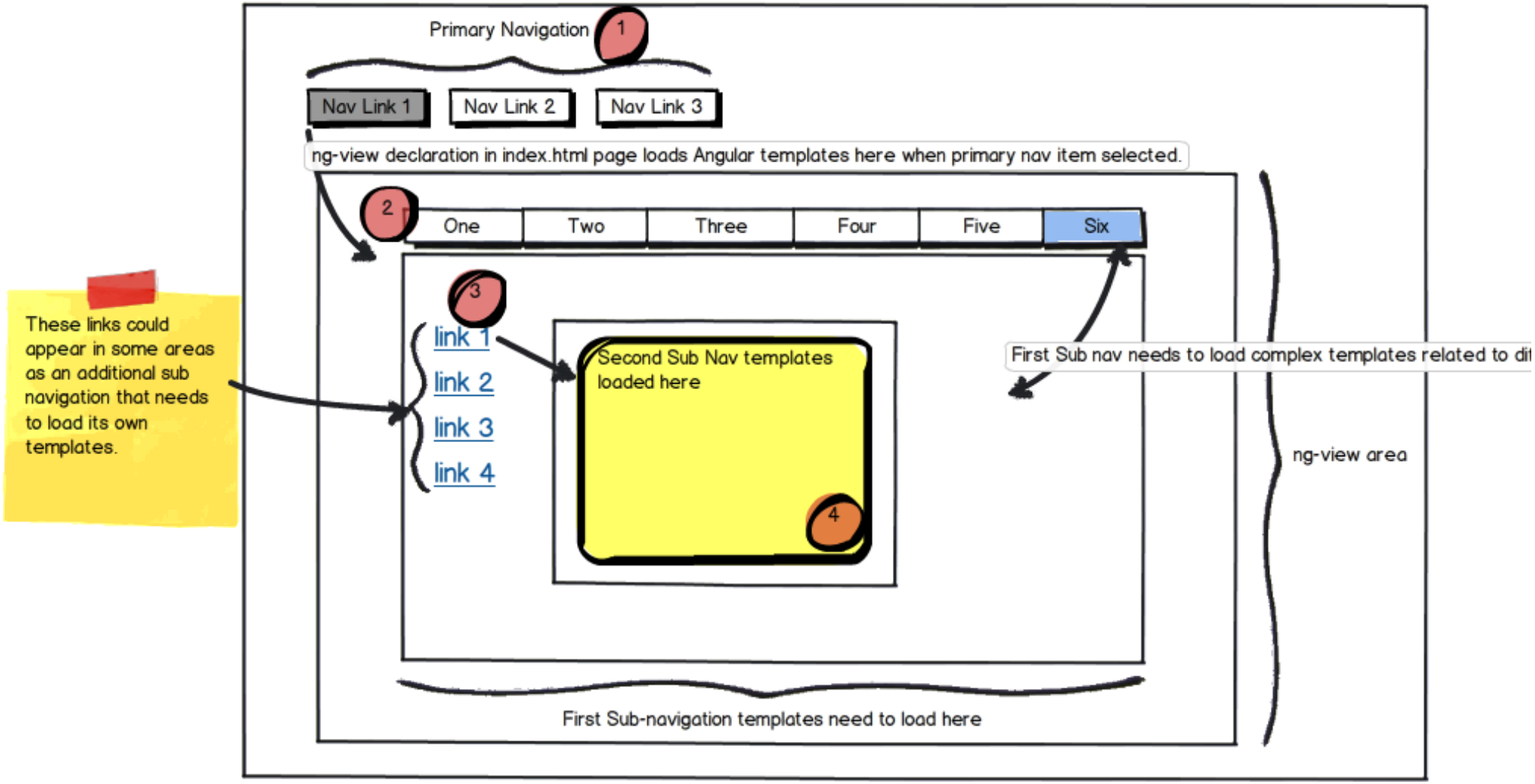
3) Better to use component model to boost productivity and maintainability.

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

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

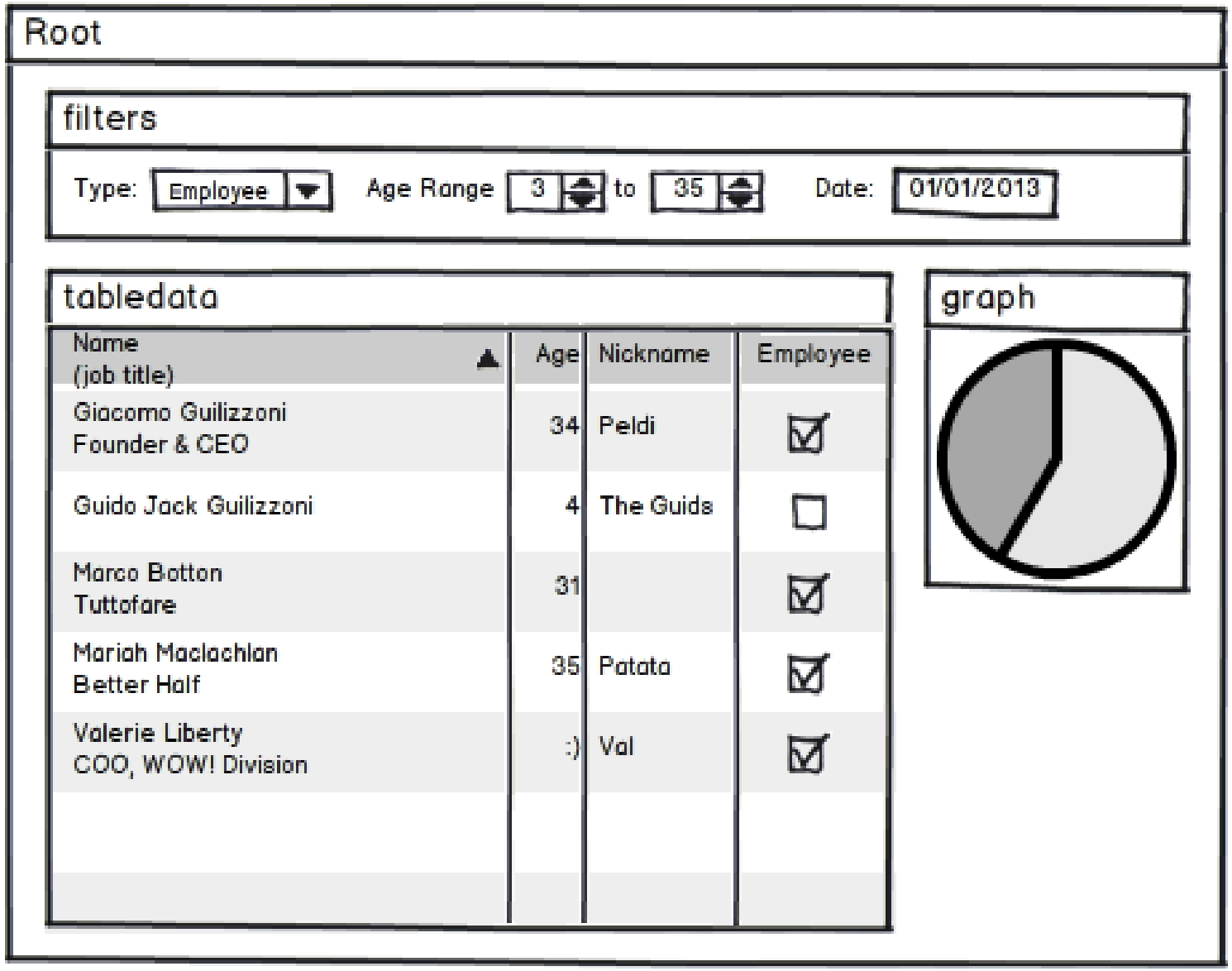
6) sitemap → router config

Hierarchical Routing



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

SPA with Multiple Router Outlets



Getting Started with React Router v5

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

  npm install -g create-react-app

  create-react-app demo-app

  cd demo-app

- Install *react-router-dom*:

  npm install react-router-dom

- Implement routing in *src/App.js*

Basic Routing using React Router v5 (1)

```
import React from 'react';

import { BrowserRouter as Router, Switch, Route, Link } from "react-router-dom";

export default function App() {
      return (
          <Router>
              <div>
                  <nav>
                      <ul>
                          <li>
                              <Link to="/">Home</Link>
                          </li>
                          <li>
                              <Link to="/about">About</Link>
                          </li>
                          <li>
                              <Link to="/topics">Topics</Link>
                          </li>
                      </ul>
                  </nav>

                  (- continues -)
```

Basic Routing using React Router v5 (2)

```
                {/* A <Switch> looks through its children <Route>s and
              renders the first one that matches the current URL. */}
                  <Switch>
                      <Route path="/about">
                          <About/>
                      </Route>
                      <Route path="/topics">
                          <Topics/>
                      </Route>
                      <Route exact path="/">
                          <Home/>
                      </Route>
                  </Switch>
              </div>
          </Router>
      );
}

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

Nested Routing & Params using Router v5 (3)

```
function Topics() {
      let match = useRouteMatch();

      return (
          <div>
              <h2>Topics</h2>

              <ul>
                  <li>
                      <Link to={`${match.url}/components`}>Components

                    </Link>
                  </li>
                  <li>
                      <Link to={`${match.url}/props-v-state`}>
                          Props v. State
                      </Link>
                  </li>
              </ul>

         (- continues -)
```

Nested Routing & Params using Router v5 (4)

```
            {/* The Topics page has its own <Switch> with more routes
            that build on the /topics URL path. You can think of the
            2nd <Route> here as an "index" page for all topics, or
            the page that is shown when no topic is selected */}
              <Switch>
                  <Route path={`${match.path}/:topicId`}>
                      <Topic />
                  </Route>
                  <Route exact path={match.path}>
                      <h3>Please select a topic.</h3>
                  </Route>
              </Switch>
          </div>
    );
}

function Topic() {
    let { topicId } = useParams();
    return <h3>Requested topic ID: {topicId}</h3>;
}
```

React Router Dynamic Configuration

```
<Route path="/" component={Base} />
<Route path="/home" component={Home} />
<Route path="/intro"
       render={() => <div>How to start using this app</div>} />
<Route path="/repos" component={Repos} />
<Route path="/topics" component={Topics} />
<Route path="/about" component={About} />
<Route path="/show-location" component={ShowTheLocation} />
```

```
const Repos = (props) => {
    return (
      <div>
        <h2>Repos</h2>
        <Route path="/repos/:userName/:repoName" component={Repo} />
      </div>
    );
};
```

Hierarchical navigation

Site Navigation using Router

```
<ul className="main-menu">
  <li><Link to="/home">Home</Link></li>
  <li><Link to="/intro">Intro</Link></li>
  <li><Link to="/repos">Repos</Link></li>
  <li><Link to="/topics">Topics</Link></li>
  <li><Link to="/show-location">Show the Location</Link></li>
  <li><Link to="/about">About</Link></li>
  <form className="navbar-form navbar-right" role="search"
        onSubmit={this.handleSerch}>
    <input type="text" placeholder="userName" /> / {' '}
    <input type="text" placeholder="repo" /> {' '}
    <button type="submit" className="btn btn-default">Go</button>
  </form>
</ul>
```

Programmatic Navigation using Router

```
ReactDOM.render(
    <Router >
        <App />
    </Router>,
    document.getElementById('root')
);
```

```
handleSearch = (event) => {
    event.preventDefault();
    const userName = event.target.elements[0].value;
    const repo = event.target.elements[1].value;
    const path = `/repos/${userName}/${repo}`;
    console.log(path);
    console.log(this.context);
    // this.context.router.history.push(path);
    this.props.history.push(path);
}
```

Using *withRouter* Decorator (HOC)

```
import React from 'react';
import { withRouter } from 'react-router-dom';

@withRouter
export default class ShowTheLocation extends React.Component {
  render() {
    const { match, location, history } = this.props;
    return (
      <div>
        <div>You are now at {location.pathname}</div>
        <div>The match is: {JSON.stringify(match)}</div>
        <div>The history contains: {JSON.stringify(history)}</div>
      </div>
    )
  }
}
```

Using *withRouter* Decorator (HOC)

```
import React from 'react';
import { withRouter } from 'react-router-dom';

class ShowTheLocation extends React.Component {
  render() {
    const { match, location, history } = this.props;
    return (
      <div>
        <div>You are now at {location.pathname}</div>
        <div>The match is: {JSON.stringify(match)}</div>
        <div>The history contains: {JSON.stringify(history)}</div>
      </div>
    )
  }
}
export default withRouter(ShowTheLocation);
```

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.

Login Demo with Redirection (1)

```
export default function AuthExample() {
    return (
        <Router>
            <div>
                <AuthButton />
                <ul>
                    <li><Link to="/public">Public Page</Link></li>
                    <li><Link to="/protected">Protected Page</Link></li>
                </ul>
                <Switch>
                    <Route path="/public"><PublicPage /></Route>
                    <Route path="/login"><LoginPage /></Route>
                    <PrivateRoute path="/protected">
                        <ProtectedPage />
                    </PrivateRoute>
                </Switch>
            </div>
        </Router>
    );
}
```

Login Demo with Redirection (2)

```
const fakeAuth = {
      isAuthenticated: false,
      authenticate(cb) {
          fakeAuth.isAuthenticated = true; setTimeout(cb, 100); },
      signout(cb) {
          fakeAuth.isAuthenticated = false; setTimeout(cb, 100);}
  };

  function AuthButton() {
      let history = useHistory();
      return fakeAuth.isAuthenticated ? (
          <p>Welcome!{" "}
              <button onClick={() => {
                  fakeAuth.signout(() => history.push("/"));
              }}>Sign out</button>
          </p>)
          : (<p>You are not logged in.</p>);
  }
```

Login Demo with Redirection (3)

```
// A wrapper for <Route> that redirects to the login screen if you're not yet authenticated.
function PrivateRoute({ children, ...rest }) {
    return (
        <Route {...rest}
            render={({ location }) =>
                fakeAuth.isAuthenticated ? (
                    children
                ) : (
                    <Redirect to={{
                    pathname: "/login",
                    state: { from: location }
                  }}/>
                )
            }
        />
    );
}
```

Login Demo with Redirection (4)

```
function PublicPage() { return <h3>Public</h3>; }

function ProtectedPage() { return <h3>Protected</h3>; }

function LoginPage() {
    let history = useHistory();
    let location = useLocation();
    let { from } = location.state || { from: { pathname: "/" } };
    let login = () => {
        fakeAuth.authenticate(() => {
            history.replace(from);
        });
    };
    return (
        <div>
            <p>You must log in to view the page at {from.pathname}</p>
            <button onClick={login}>Log in</button>
        </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