

<WA1/>
<AW1/>
2021

React Router

Applications have more than one page...

Fulvio Corno
Luigi De Russis
Enrico Masala



alamy stock photo

F5GY26
www.alamy.com



POLITECNICO
DI TORINO



Outline

- Objective and problems
- A Solution, the React way: React Router



Full Stack React, chapter “Routing”

React Handbook, chapter “React Router”

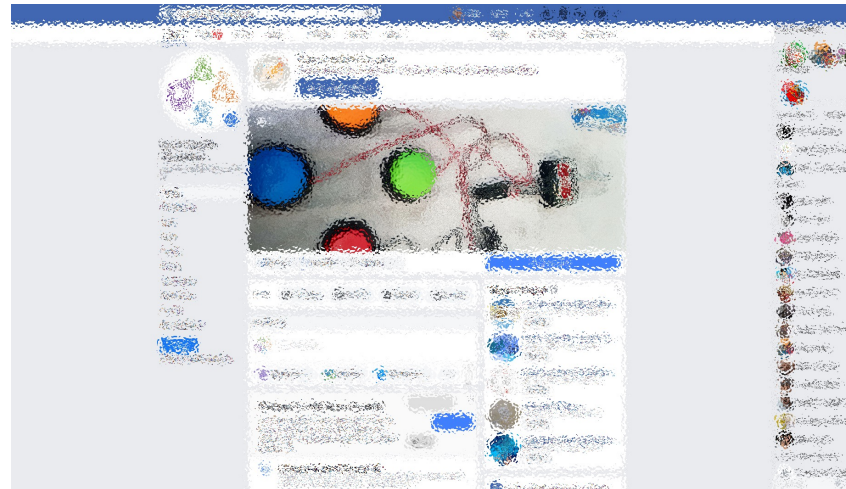
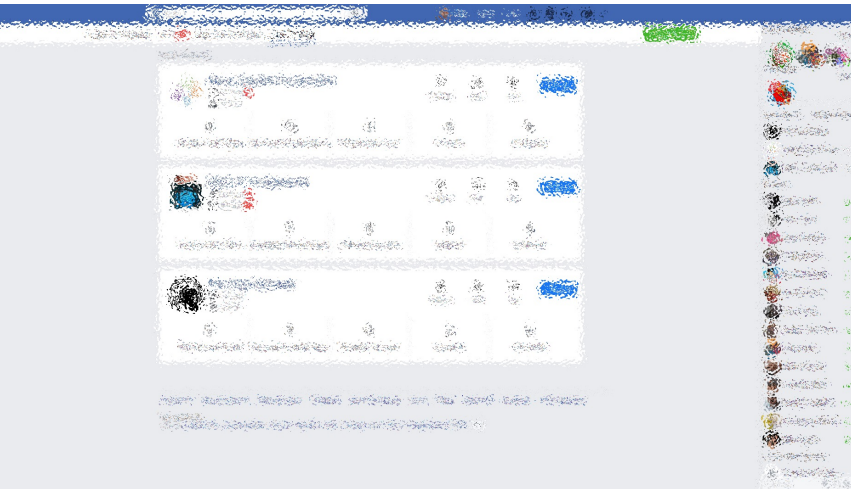
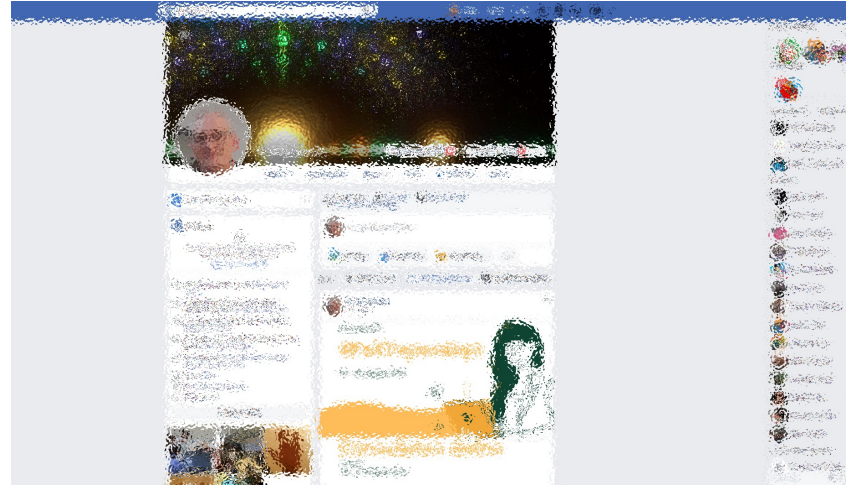
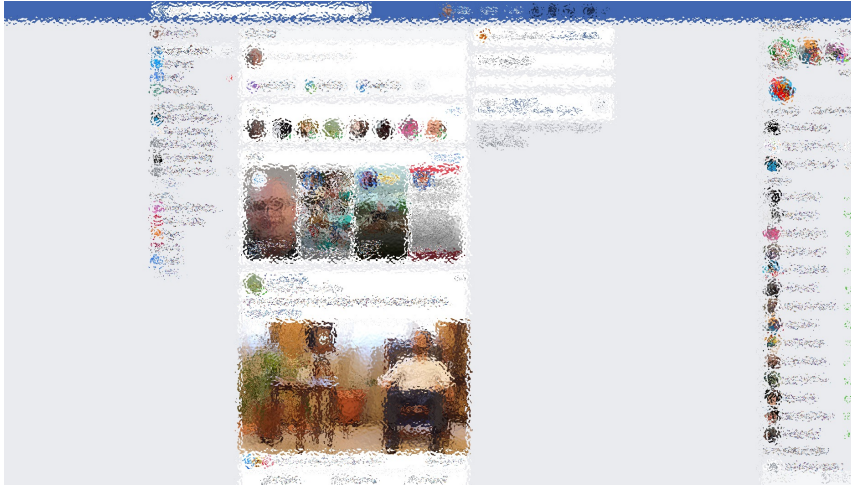
Multi-page Single Page Applications

OBJECTIVES AND PROBLEMS

Supporting Complex Web Applications

- Switching between many different page layouts
- Managing the flow of navigation across a set of “pages”
- Maintaining the default web navigation conventions (back, forward, bookmarks, ...)
- Allowing URLs to convey information
- Allowing re-loading KBs of JavaScript at every page change
- Keeping the state across page changes
- ...

Example



- Different layout and contents
- Some common parts
- No “page reload”
- URL changes accordingly

Some Use Cases

- Master list / detail view
- Logged / Unlogged pages
- Sidebar navigation
- Modal content
- Main Contents vs. User Profile vs. Setting vs. ...

Using URLs for Navigation State

- URLs determine the *type* of the page or the *section* of the website
 - Changing page \Leftrightarrow Changing the URL
- URLs also *embed information* about the item IDs, referrers, categories, filters, etc.
- URLs can be shared/saved/bookmarked, and they are sufficient for rebuilding the whole exact page
 - Deep Linking
- Back and Forward buttons navigate the URL history

Example URLs on facebook.com:

/

/profile.name

/profile.name

/posts/12341232124
22123

/pagename

/pages/?category=y
our_pages

Using URLs for Navigation State

- URLs determine the *type* of the page or the *section* of the website
 - Changing page \Leftrightarrow Changing the URL
- URLs also *embed information* about the idem IDs, referrers, ...
- URLs can be used to identify the page and provide sufficient information to the application
 - Deep Linking
- Back and Forward navigation

➤ With any URL, the React application will **always return the same page** (`index.html/index.js`) that will load and mount the same App

➤ The URL is queried by the App to customize the render

Example URLs on facebook.com:

/

/profile.name

/profile.name

/posts/12341232124
22123

/pagename

/pages/?category=our_pages



<https://reacttraining.com/react-router/>

<https://flaviocopes.com/react-router/>

Full Stack React, chapter “Routing”

React Handbook, chapter “React Router”

React as a REST Client

THE REACT ROUTER

React-Router

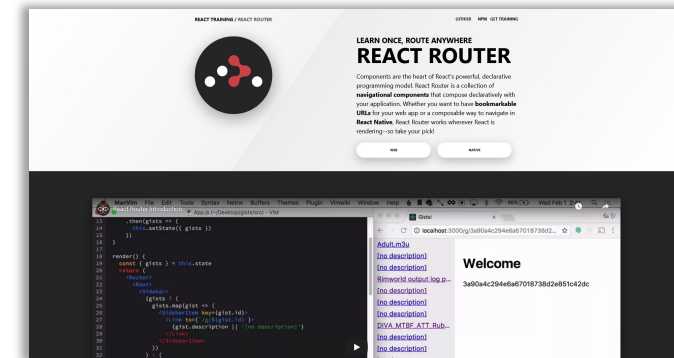
- The problems associated with multi-page navigation and URL management are usually handled by *router* libraries
- A JavaScript Router manages
 - Modifying the location of the app (the URL)
 - Determining what React components to render at a given location
- In principle, whenever the user clicks on a new URL
 - We prevent the browser from fetching the next page
 - We instruct the React app to switch in & out components

React-Router



<https://reactrouter.com/>
<https://github.com/ReactTraining/react-router>

- React does not contain a specific router functionality
 - Different router libraries are available
 - The most frequently adopted is **react-router**
 - `npm install react-router-dom`



Package	Version	Docs	Description
<code>react-router</code>	npm v5.2.0	API Docs site API Docs markdown	The core of React Router
<code>react-router-dom</code>	npm v5.2.0	API Docs site API Docs markdown	DOM bindings for React Router
<code>react-router-native</code>	npm v5.2.0	API Docs site API Docs markdown	React Native bindings for React Router
<code>react-router-config</code>	npm v5.1.1	API Docs readme	Static route config helpers

Features

- Connects React app navigation with the browser's native navigation features
- Selectively shows components according to the current routes
 - Rules matching URL fragments
- Easy to integrate and understand; it uses normal React components («it's just React»)
 - Links to new pages are handled by `<Link>`, `<NavLink>` and `<Redirect>`
 - For determining that to render we use `<Route>` and `<Switch>`
 - The whole application is wrapped in a `<Router>` container

Overview of React-Router

`<Router>`

```
<Link to="/">Home</Link>
<Link to="/about">About</Link>
<Link to="/dash">Dashboard</Link>
```

`</Router>`

`'/about'`



`<Router>`

```
<Switch>
  <Route exact path="/">
    <Home />
  </Route>
  <Route path="/about">
    <About />
  </Route>
  <Route path="/dashboard">
    <Dashboard />
  </Route>
</Switch>
```

`</Router>`

<Router>

- Different routers are available: <BrowserRouter>, <HashRouter>, <MemoryRouter>, <NativeRouter>, <StaticRouter>
- BrowserRouter uses normal URLs and the HTML5 Location API
 - Recommended for modern browsers
 - Requires *some server configuration*
 - `import { BrowserRouter as Router } from 'react-router-dom' ;`
- HashRouter uses '#' in the URL
 - Compatible with older browsers
 - Requires no config on the server
- Must wrap the entire App

<Router>

- Different routers are available: <BrowserRouter>, <HashRouter>, <MemoryRouter>, <NativeRouter>, <StaticRouter>
- BrowserRouter uses normal URLs and the HTML5 Location API
 - Recommended for modern browsers
 - Requires *some server configuration*
 - `import { BrowserRouter as Router } from 'react-router-dom';`
- HashRouter uses '#' in the URL
 - Compatible with older browsers
 - Requires no config on the server
- Must wrap the entire App

Not needed with the React Development Server.

When served as a static bundle, all paths must be mapped to index.html:

```
app.use(express.static('build'));  
  
app.get('/*', function (req, res) {  
  res.sendFile('build/index.html');  
});
```

More on this -> next weeks!

<https://create-react-app.dev/docs/deployment/#serving-apps-with-client-side-routing>

Selective Render

- Content wrapped in `<Route>` will be rendered only if the URL path matches the specification
 - `path = '/fragment'` uses regexp to check if the URL matches
 - `component = {MyComponent}` renders the specified component if the path matches

```
<Router>
  <div>
    <Route exact path="/" component={Home} />
    <Route path="/news" component={NewsFeed} />
  </div>
</Router>
```

Route matching methods


- **path** = regular expression matched against the URL
 - If path is missing, then the URL always matches
- Options
 - **exact**: revert to exact string comparison (no regexp)
 - **strict**: if the pattern has a trailing / , then the URL must have a trailing /
 - **sensitive**: the match becomes case-sensitive (default: insensitive)

Dynamic Routes

- Routes may have **parametric** segments, with the **:name** syntax in the path specification
 - `<Route exact path="/post/:id" component={Post} />`
 - The 'id' part will be available as `match.params.id`

```
<Route exact path="/post/:id" render={({match}) => (  
  <Post post={posts.find(  
    p => p.id === match.params.id)} />  
)} />
```

Route render methods

- `<Route component={MyComponent}/>`
 - If path matches, render MyComponent
 - May also specify `<MyComponent>` by *nesting* it inside `<Route>`
- `<Route render={ () => <C1><C2/></C1> } />` 
- `<Route children={ ({match}) => <C1><C2/></C1> } />`
 - Always render the result of the function (e.g., JSX expression)
 - Useful if the expression internally self-customizes according to match status
- In all cases, the component or the function receives 3 props
 - `match`: the matching status of the route
 - `location`: the current browser location (URL)
 - `history`: a reference to a history object wrapping browser's history

Route match object

- With `component={}` you have `props.match` inside the component
- With `render={}` or `children={}`, you have `({match}) => ()` in the function
- `match` is composed by
 - `params` (*object*) Key/value pairs corresponding to the dynamic segments of the path
 - `isExact` (*boolean*) true if the entire URL was matched (no trailing characters)
 - `path` (*string*) The path pattern used to match. Useful for building nested `<Route>`s
 - `url` (*string*) The matched portion of the URL. Useful for building nested `<Link>`s
- Note: with `children`, `match` may be `null` (`null` will be passed to the render function)

<https://reacttraining.com/react-router/web/api/match>

Hooks

- The three routing props, together with the route's parametric segment, are available as **hooks**
 - useHistory()
 - useLocation()
 - useParams()
 - useRouteMatch()
- useRouteMatch is useful for accessing the **match** data without actually rendering a <Route>

```
const history = useHistory();
history.push('/home');
// navigate to '/home'

const location = useLocation();
console.log(location.pathname);
// e.g., /blog

const { slug } = useParams();
console.log(slug);
// if <Route path="/blog/:slug">
// and the URL is "/blog/3"
// it will print "3"
```

<Switch>

- General rule: **all** <Route>s whose path matches the URL are rendered
 - by default, Route is *inclusive*
- Sometimes, we want to **render only one**, of a group of Routes
- <Switch> may include many <Route> (or <Redirect>), and will render only *the first child* that matches
 - Routes included in Switch are *exclusive*
 - Always start with the **most restrictive** rules

```
<Switch>
  <Route exact path="/">
    <Home />
  </Route>
  <Route path="/about">
    <About />
  </Route>
  <Route path="/:user">
    would also match /about
    <User />
  </Route>
  <Route> no path: always matches
    <NoMatch />
  </Route>
</Switch>
```

<Link>

- The **Link** component is used to trigger new routes
 - Don't use <a> links
- Attribute **to**={} specifies the target URL
 - As a string
 - As an object {pathname, search, hash, state}
 - As a function returning one of the above
- **replace** overwrites (rather than adding) the URL in the history
- Will generate a DOM <a> component
 - Extra attributes are forwarded to the <a>

```
<Link to={'/dashboard'}>Dashboard</Link>  
<Link to={'/about'}>About</Link>
```

Link Destination Object

- `<Link to={object}/>`, with the object composed of:
 - `pathname`: A string representing the path to link to
 - `search`: A string representation of query parameters (useful for dynamically generated parameters)
 - `hash`: A hash to put in the URL, e.g., `#a-hash` (not used with `BrowserRouter`)
 - `state`: State to persist to the location (useful to initialize the state after the route has been followed)

Passing State Among Pages

- If you need to pass information that will be available whenever the app returns to a specific location, you can include it in `to={object}`
- Alternative to pass information as param in the URL
- Available as `location.state` in the target `<Route>`

```
<Link to={{  
  pathname: "/update",  
  state: { examCode: code }  
}}>Update</Link>
```

```
<Route path="/update"  
  render={({location}) =>  
  <ExamForm  
    examCode={location.state.examCode}/>  
  }/>
```

Tips

- `location.state` can be accessed also via `useLocation()` hook
- `location.state` may not be set if the URL is erroneously invoked or directly loaded: double check it is correctly set before use

```
<Route path="/update"
  render={() => <ExamForm ... /> }/>
```

```
function ExamForm(props) => {
  const location = useLocation();
  const examCode = location.state ?
    location.state.examCode : ''
}
```

```
<Route path="/update"
  render={({location}) =>
  <ExamForm
    examCode=location.state ?
    location.state.examCode : ''>/>
  }/>
```


<NavLink>

- A special version of the <Link> that will add styling attributes to the rendered element *when it matches the current URL*
- Useful for automatically highlighting *the current item* in a menu
 - `activeClassName` (string): the class to give the element when it is active (default: `'active'`). Added to `className`
 - `activeStyle` (object): the styles to apply to the element when it is active

```
<NavLink
  to={` ${albumsPathname}/${album.id}`}
  activeClassName='active'
  className='item'
  key={album.id}
>${album.name}</NavLink>
```

<Redirect>

- When rendered, forces the navigation to a new location
- Used to “programmatically” force a location change
 - In event handlers, you often need to “jump” to a given page
 - Might use `history.push`
 - Easier way: set a state property that will cause a render of a `<Redirect>`

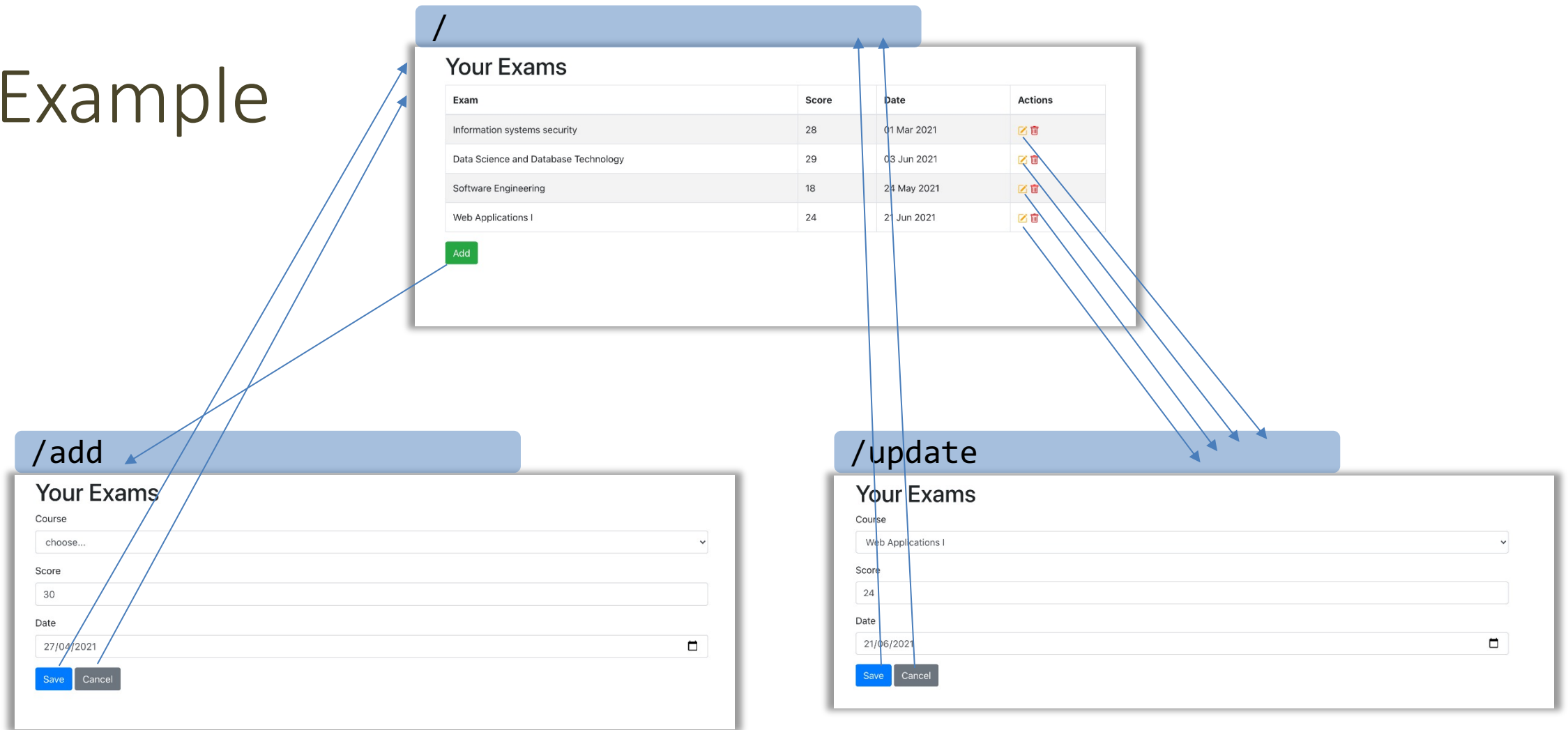
```
const [submitted, setSubmitted] =
  useState(false) ;

handleSubmit = (ev) => {
  ev.preventDefault();
  setSubmitted(true);
}

if (submitted)
  return <Redirect to="/" />;
else
  return ...
```

<https://tylermcginnis.com/react-router-programmatically-navigate/>

Example



https://github.com/polito-WA1-AW1-2021/react-scores/tree/with_router



License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
 - **Share** — copy and redistribute the material in any medium or format
 - **Adapt** — remix, transform, and build upon the material
 - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
 - **Attribution** — You must give [appropriate credit](#), provide a link to the license, and [indicate if changes were made](#). You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
 - **NonCommercial** — You may not use the material for [commercial purposes](#).
 - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the [same license](#) as the original.
 - **No additional restrictions** — You may not apply legal terms or [technological measures](#) that legally restrict others from doing anything the license permits.
- <https://creativecommons.org/licenses/by-nc-sa/4.0/>

