

<WA/>

2024

# React Router

Applications have more than one page...

Fulvio Corno

Luigi De Russis

Enrico Masala



# Outline

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



Full Stack React, chapter “Routing”

React Handbook, chapter “React Router”

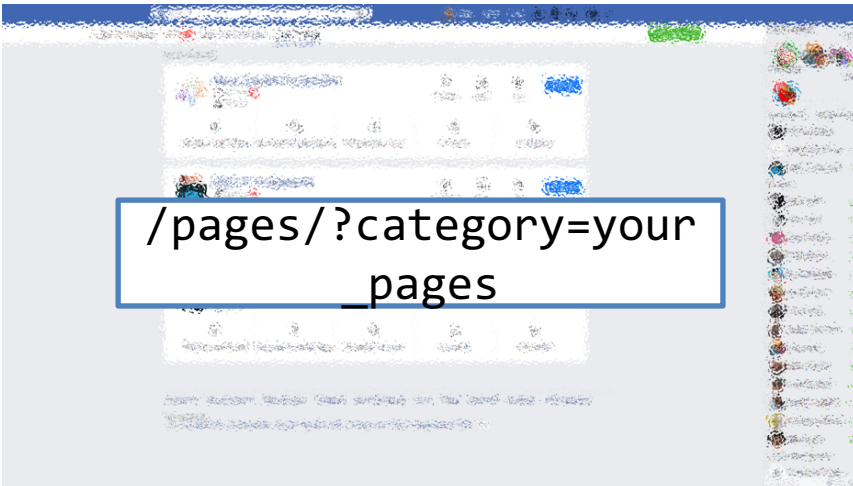
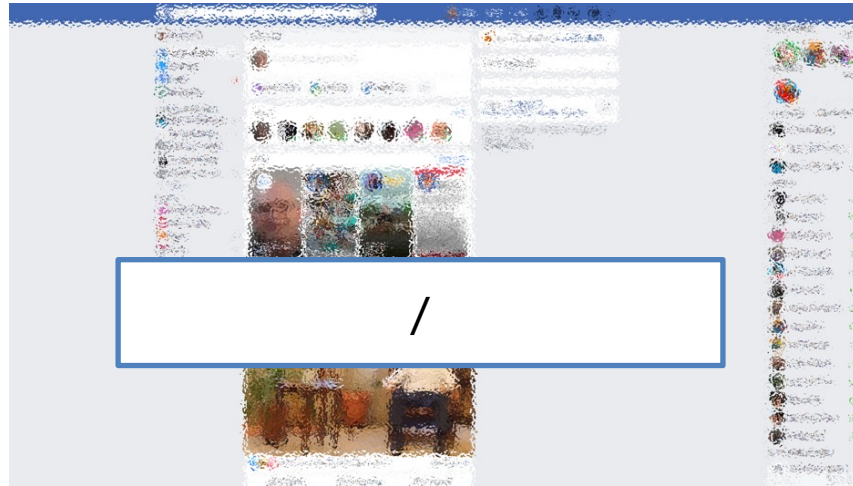
Multi-page Single Page Applications

# 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
- Avoiding to re-load 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

- Main list / detail view
- Logged in / 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 item IDs, referrers, ...
- URLs can be used for navigation
  - Deep Linking
- Back and Forward

Special configuration:

- 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 content is then queried by the App to customize the render

Example URLs on facebook.com:

```
/
/profile.name
/profile.name
/posts/12341232124
22123
/pagename
/pages/?category=y
our_pages
```





<https://reactrouter.com/>

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

<https://www.robinwieruch.de/react-router/>

Full Stack React, chapter “Routing”

React Handbook, chapter “React Router”

React as an HTTP 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

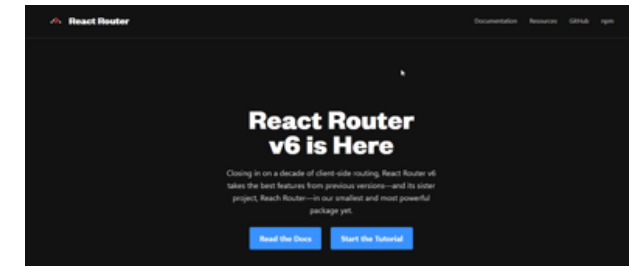
- React does not contain a specific router functionality
  - Different router libraries are available
  - The most frequently adopted is **react-router**
    - Now version 6.23
  - `npm install react-router-dom`
    - Will also install react-router as a dependency



<https://reactrouter.com/>

<https://github.com/remix-run/react-router>

- `react-router` contains most of the core functionality of React Router including the route matching algorithm and most of the core components and hooks
- `react-router-dom` includes everything from `react-router` and adds a few DOM-specific APIs, including `<BrowserRouter>`, `<HashRouter>`, and `<Link>`
- `react-router-native` includes everything from `react-router` and adds a few APIs that are specific to React Native, including `<NativeRouter>` and a native version of `<Link>`



# 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
  - Links to new pages are handled by `<Link>`, `<NavLink>`, and `<Navigate>`
  - To determine what must be rendered we use `<Route>` and `<Routes>`
  - Defines hooks `useRoute`, `useNavigate`, `useSearchParams`
- The whole application is **wrapped** in a `<BrowserRouter>` container

# Overview of React-Router

`<Router>`

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

`</Router>`

`'/about'`



`<Router>`

```
<Routes>
  <Route path="/"
    element={<Home />} />
  <Route path="/about">
    element={<About />} />
  <Route path="/dashboard">
    element={<Dashboard />} />
</Routes>
```

`</Router>`

# Routers



- Routers can be defined as **functions** or **components**
- Routers' functions are quite new in React Router (since version 6.4) and not yet fully supported in the entire React ecosystem
  - e.g., they do not work in React Native
- We will use *Router Components* in the course

In v6.4, new routers were introduced that support the new data APIs:



- `<BrowserRouter>`
- `<HashRouter>`
- `<MemoryRouter>`
- `<StaticRouter>`

The following routers do not support the data APIs:

- `<BrowserRouter>`
- `<MemoryRouter>`
- `<HashRouter>`
- `<NativeRouter>`
- `<StaticRouter>`

# <Router>

- Different routers are available:  
<BrowserRouter>, <HashRouter>, <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
  - **Not recommended**, unless for compatibility reasons
- Must wrap the entire App



# <Router>

- Different routers are available:  
<BrowserRouter>, <HashRouter>, <StaticRouter>

- ➔ • **BrowserRouter** uses normal URLs and the HTML5 Location API

- **Recommended** for modern browsers
- Requires *some server configuration*
- `import { BrowserRouter as Router`

- **HashRouter** uses '#' in the URL

- Compatible with older browsers
- Requires no config on the server
- **Not recommended**, unless for compatibility

- Must wrap the entire App

Not needed with the React Development Server.

When served as a static bundle from a “normal” web server, all paths must be mapped to index.html:

```
app.use(express.static('build'));

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

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

# Selective Render

- Alternative versions of a component content must be wrapped in `<Routes>`
  - Each alternative is represented by a `Route`
  - The route with the “most specific” match will be rendered
- Each `<Route>` specifies the URL path matching requirement
  - `path = '/fragment'` check if the URL matches the fragment
  - `element = {<JSXelement/>}` renders the specified JSX fragment if the path is *the best match*

```
<Router>
  <Routes>
    <Route path="/" element={<Home/>} />
    <Route path="/news" element={<NewsFeed/>} />
  </Routes>
</Router>
```


# Route matching Methods

- **path** = string matched against the URL
- A path is made of different URL 'segments' (separated by /)
  - **Static** segment → e.g., users
  - **Dynamic** segment → e.g., :userId
  - **Star** segment → \*
- Examples:
  - /users/:userId
  - /docs/\*
  - /
  - /contact-us
- Options
  - **caseSensitive**: the match becomes case-sensitive (default: insensitive)
    - changing the default is not recommended



If the Location URL matches more than one route path, **the most specific one** is selected

# Nesting Routes

- Routes may follow the layout hierarchy of the interface components
- It is possible to **nest** a `<Route>` **inside** another `<Route>` component
  - The paths will be **concatenated**
  - The parent `<Routes>` will browse, recursively, through all matching paths
  - **All** route elements in the **best matching path** will be rendered
- The matching children will be rendered inside the `<Outlet>` component in the parent's render tree
  - `<Outlet>` specifies “**where**” the matching children should be rendered
  -  If you forget `<Outlet>`, the children will *not* display

# Example

```
function App() {  
  return (  
    <div>  
      <h1>Basic Example</h1>  
      <Routes>  
        <Route path="/" element={<Layout />}>  
          <Route path="about" element={<About />} />  
          <Route path="dashboard"  
            element={<Dashboard />} />  
        </Route>  
      </Routes>  
    </div>  
  );  
}
```

```
function Layout() {  
  return (  
    <div>  
      <nav>... main navigation menu ...</nav>  
      <hr />  
      <Outlet />  
    </div>  
  );  
}
```

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

# Special Routes (1/2)

- **Index** route
  - `<Route index element={<Home />} />`
  - A child route with **no path** that renders in the parent's outlet at the parent's URL
  - Use cases:
    - They match when a parent route matches but none of the other children match.
    - They are the default child route for a parent route.
    - They render when the user did not click one of the items in a navigation list yet.

# Special Routes (2/2)

- **Layout** route
  - A route without path will always be matched
  - Useful to “wrap” with a common layout its children’s routes
- **“No Match”** route
  - Special case: `path="*"`
  - Will match only when no other routes do



# Example

```
function App() {
  return (
    <div>
      <h1>Basic Example</h1>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="about" element={<About />} />
          <Route path="dashboard" element={<Dashboard />} />
          <Route path="*" element={<NoMatch />} />
        </Route>
      </Routes>
    </div>
  );
}
```

```
function Layout() {
  return (
    <div>
      <nav>... main navigation menu ...</nav>
      <hr />
      <Outlet />
    </div>
  );
}
```

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

# Navigation

- Changing the location URL will re-render the Router, and all Routes will be evaluated
- Two options:
  - `<Link to= >` creates a router-aware hyperlink (activated by user clicks)
  - `useNavigate()` returns a function to trigger navigation (useful inside event handlers)

# Navigation

- Changing the location URL will re-render the Router, and all Routes will be evaluated
- Two options:
  - `<Link to= >` creates a router-aware hyperlink (activated by user clicks)
  - `useNavigate()` returns a function to trigger navigation (useful inside event handlers)

## ⚠ Warning ⚠

**Never** use a “plain hyperlink” `<a>`

**Never** use a “form submission”  
`<form action='...'>`

They will reload the whole application  
(and destroy the current state)

This is **NOT ALLOWED** in a SPA

# Examples

```
function Home() {  
  return (  
    <div>  
      <h1>Home</h1>  
      <nav>  
        <Link to="/">Home</Link>  
        {" "  
        <Link to="about">About</Link>  
      </nav>  
    </div>  
  );  
}
```

```
function Invoices() {  
  const navigate = useNavigate();  
  return (  
    <div>  
      <NewInvoiceForm  
        onSubmit={(event) => {  
          const newInvoice = create(event.target);  
          navigate(`/invoices/${newInvoice.id}`);  
        }}  
      />  
    </div>  
  );  
}
```

All paths are relative,  
unless they start with /

# Active Navigation

- When creating menus or navigation elements, it is useful to see which item is the currently selected one
- `<NavLink>` behaves like `<Link>`, but knows whether it is “active”
  - It adds the “`active`” class to the rendered link (to be customized with CSS)
  - You may create a **callback** in `className={}` that receives the `isActive` status and decides which class to apply
  - You may create a **callback** in `style={}` that receives the `isActive` status and decides which CSS style(s) to apply

# Dynamic Routes

- Routes may have **parametric** segments, with the **:name** syntax in the path specification
  - `<Route path="/post/:id" element={<Post/>} />`
  - The 'id' part will be available to the element through the **useParams()** hook

```
<Route  
  path="/post/:id"  
  element={<Post/>} />
```

```
function Post(props) {  
  const {id} = useParams();  
  ...  
}
```

# Dynamic Routes

- Routes may have **parametric** segments, with the **:name** syntax in the path specification
  - `<Route path="/post/:id" element={<Post/>} />`
  - The 'id' part will be available to the element through the **useParams()** hook

```
<Route  
  path="/post/:id"  
  element={<Post
```

- useParams returns an object of key/value pairs of the dynamic params from the current URL that were matched by the `<Route path>`
- Children routes inherit all params from their parent routes

```
function Post(props) {  
  const {id} = useParams();  
  ...  
}
```



# Example

```
function App() {  
  return (  
    <Routes>  
      <Route  
        path="/invoices/:invoiceId"  
        element={<Invoice />}  
      />  
    </Routes>  
  );  
}
```

Matches a URL like  
`/invoices/1234`

```
function Invoice() {  
  const { invoiceId } = useParams();  
  return <h1>Invoice {invoiceId}</h1>;  
}
```

```
function Invoice() {  
  const params = useParams();  
  return <h1>Invoice {params.invoiceId}</h1>;  
}
```

# Location State: Passing Information Among Pages

- When navigating, it is possible to **pass** some **information** to the next page, thanks to the `location.state` BOM attribute
  - Alternative to dynamic URLs
- The value may be retrieved with `useLocation()` on the next page
  - **Beware:** objects are serialized as strings, **avoid passing 'complex' objects** (e.g., DO NOT pass `dayjs` objects)



```
const navigate = useNavigate() ;

// go to URL and send information
navigate( url, {state: userData} ) ;
```

```
<Link to={url}
      state={userData} >
  . . .
</Link>
```



```
const location = useLocation();
const userData = location.state;
```

# Exploiting Search Parameters

- A URL may contain some “query search parameters”
  - `/products?sort=date&filter=valid`
- `useSearchParams()` allows you to read and modify the query string portion of the location
  - Returns the current version of the parameter, and a function to modify them
  - Behaves like `useState`

```
const [searchParams, setSearchParams] =  
  useSearchParams();
```

- `searchParams` is a `URLSearchParams` object

<https://developer.mozilla.org/en-US/docs/Web/API/URL/searchParams>

You may access each parameter with  
`searchParams.get('sort')`  
`searchParams.get('filter')`

- `setSearchParams` receives an object of `{ key: value }` pairs that will replace the current parameters

# Summary: react-router-dom

- Wrap app in `<BrowserRouter>`
- Routing and rendering:
  - `<Routes>`
  - `<Route path= element= />`
  - `<Outlet/>`
- Navigation:
  - `<Link to= >...</Link>`
  - `<NavLink to= >...</NavLink>`
  - `useNavigate()` or `<Navigate>`
- Parameters
  - `useParams()` for Dynamic Routes
  - `useSearchParams()` for URL query strings (after “?”)
  - `useLocation()` for retrieving location state (set by navigate)

# 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/>

