

<WA1/>  
<AW1/>  
2025

# React Router

Applications have more than one page...

Fulvio Corno

Luigi De Russis



# Outline

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



Full Stack React, chapter “Routing”

React Handbook, chapter “React Router”

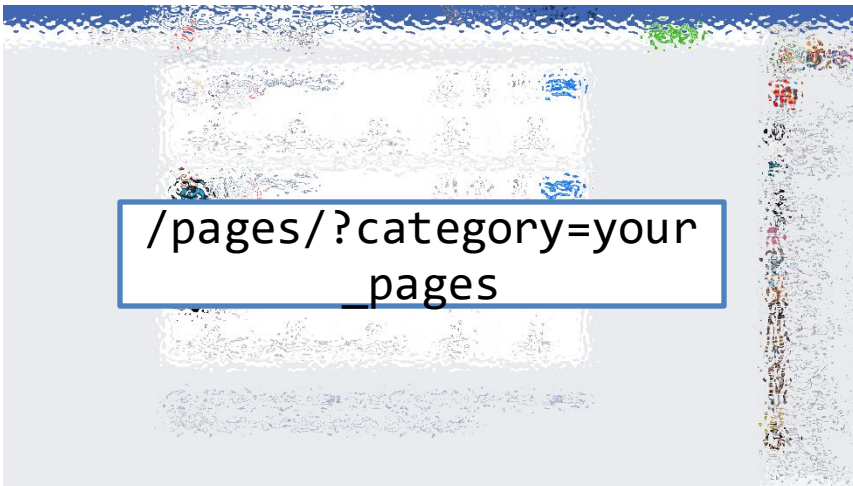
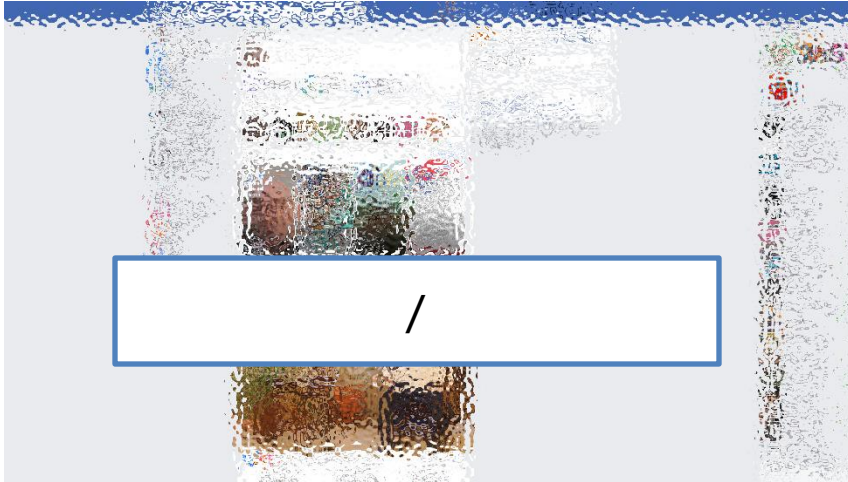
Multi-page Single Page Applications

# 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 / 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, etc.
- URLs can be configured to be sufficient to determine the page
  - Deep Linking
- Back and Forward navigation

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/1234123212422123
/pagename
/pages/?category=your_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 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

- React does not contain a specific router functionality
  - Different router libraries are available
- A commonly adopted one is **react-router**
  - Current version 7.x
  - `npm install react-router`



<https://reactrouter.com/>

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



# 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 `useNavigate`, `useParams`, `useSearchParams`
- The **whole** application is **wrapped** in a `<Router>`-like container

# Overview of React Router

`<Router>`

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

`</Router>`

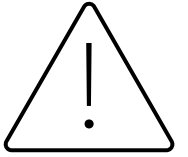
`'/about'`

`<Router>`

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

`</Router>`

# Routers



- Routers can be initialized in three ways, or “modes”
  1. Declarative
  2. Data
  3. Framework
- Features available in each mode are *additive*
  - moving from Declarative to Data to Framework adds more features at the cost of architectural control
- In the course, we will use the **Declarative** mode
  - enables basic routing features and fundamental APIs

# Types of Routers in Declarative Mode

- `<BrowserRouter>` uses normal `URLs` and the HTML5 Location API
  - ➡ – **Recommended** for modern browsers
  - Requires *some server configuration*
  - `import { BrowserRouter } from 'react-router' ;`
- `<HashRouter>` uses `#` in the URL
  - Compatible with older browsers
  - Requires no config on the server
  - **Not recommended**, unless for compatibility reasons

# Types of Routers in Declarative Mode

- `<BrowserRouter>` uses normal URLs and the HTML5 Location API



- **Recommended** for modern browsers

- Requires *some server configuration*

- `import { BrowserRouter } from 'react-router-dom'`

- `<HashRouter>` uses '#' in the URL

- Compatible with older browsers

- Requires no config on the server

- **Not recommended**, unless for compatibility

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!**

# Wrapping `<App>` with a Router

```
import { StrictMode } from 'react';  
import { createRoot } from 'react-dom/client';  
import { BrowserRouter } from 'react-router';  
import App from './App.jsx';
```

```
createRoot(document.getElementById('root')).render(  
  <StrictMode>  
    <BrowserRouter>  
      <App />  
    </BrowserRouter>  
  </StrictMode>,  
)
```

Add the highlighted lines to  
main.jsx

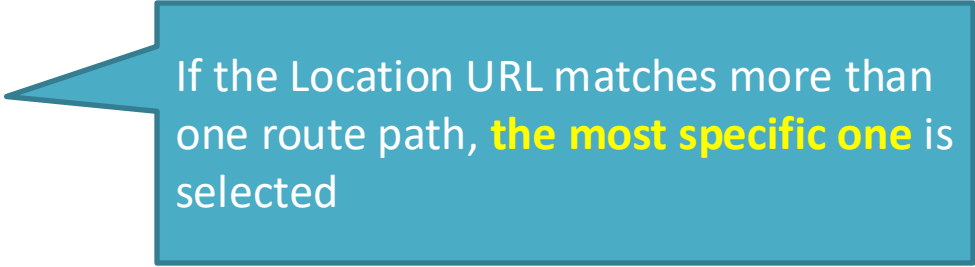
# Selective Render

- 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*

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

# 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

<https://reactrouter.com/api/components/Outlet>

# 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 doesn't have clicked 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
  - It can be used for a “Not Found” page, for example

# 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 main 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 main 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”  
(without `useActionState`)

`<form action='...'>`

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

# 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>`
- Child 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() {  
  let { invoiceId } = useParams();  
  return <h1>Invoice {invoiceId}</h1>;  
}
```

```
function Invoice() {  
  let 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., `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`
- ```
let [params, setParams] = useSearchParams();
```

  - `params` is a standard `URLSearchParams` object, <https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams>
  - `setParams` receives an object of { key: value } pairs that will replace the current parameters

# Summary: react-router

- Wrap `main.jsx` in `<BrowserRouter>`
- Routing and rendering:
  - `<Routes>`
  - `<Route path= element= />`
  - `<Outlet/>`
- Navigation:
  - `<Link to= >...</Link>`
  - `<NavLink to= >...</NavLink>`
  - `useNavigate()`
- 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/>

