04-06 React Router

Obbiettivi e problema:

- Switchare tra diversi layout
- Gestire un flow di navigazione
- Mantenere convenzioni di default nella navigazione
- Mantenere lo stato anche quando si cambia pagina

Casi d'uso:

- Lista principale, dettagli della vista
- Pagina diversa se loggata e unlogagta
- Navigazione sidebar
- Contenuti modali

Gli url possono contenere delle informazioni:

- Url determina il ripo della pagina o la sezione del sito
- Ha informazioni su ID item, categorie, filtri
- URL può essere salvato, condiviso, bookmarked
- Si può fare back e Forward

ReactRouter:

- Libreria intercetta navigazione
- Applica layout che ci aspettiamo

REACT ROUTER

– npm install **react-router-dom**

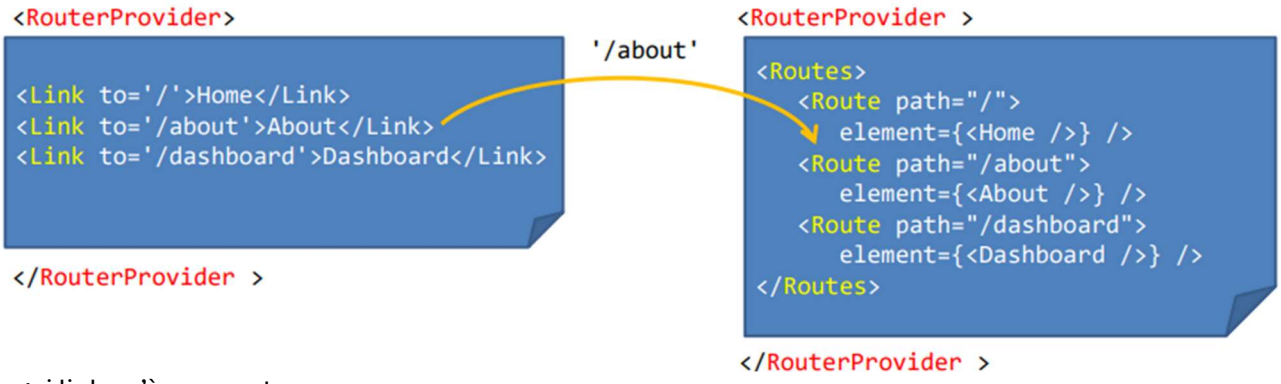
- Will also install `react-router` as a dependency

- `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>`

Permette di poter connettere le applicazioni di react a feature

Componenti:

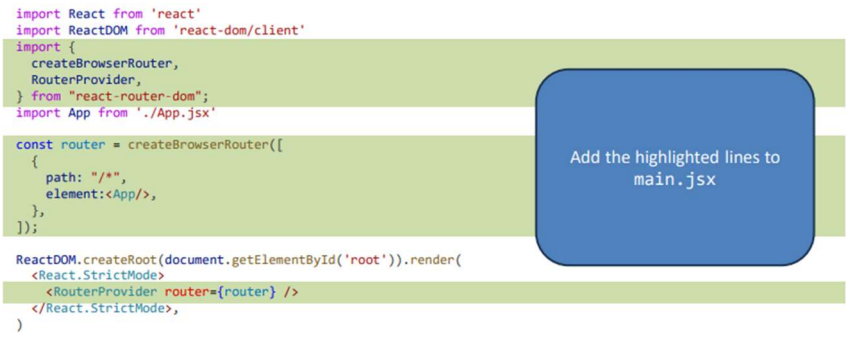
- Link per nuove pagine, gestiti con:
  - Link
  - Navlink
  - Navigate
- Determinare cosa renderizzare:
  - Route
  - Routes
- Hooks:
  - useNavigate
  - useParams
  - useSearchParams



Per ogni link , c'è una route

Tipi di routers:

- **createBrowserRouter**: crea delle route in modo raccomandato
- **createHashRouter**: url con # → vecchio



SelectiveRender:

- ogni possibile layout alternativo è mappato da una route
- libreria sceglie il alyout in base a quello che matcha di più

- 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>
```

Un **path è composto da elementi**:

- statici
- dinamici
- **star** → matcha tutto ciò che potrebbe andare bene

- 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:

Le route possono seguire il **layout gerarchico** dell'interfaccia dei componenti:

- path concatenato
- route parent navigherà, ricorsivamente, attraverso tutti i percorsi corrispondenti
- Verrà eseguito il rendering di tutti gli elementi del percorso nel percorso corrispondente migliore

Special routes:

- **INDEX route**
  - pagina di default all'apertura

```
<Route index element={<Home />} />
```

  - Un percorso figlio senza percorso che viene visualizzato nell'outlet del padre all'URL del padre
  - Casi d'uso:
    - Matchano quando un percorso padre corrisponde, ma nessuno degli altri figli corrisponde.
    - Sono il percorso figlio di default per un percorso padre.
    - Vengono visualizzati quando l'utente non ha ancora fatto clic su uno degli elementi in un elenco di navigazione
- **Layout route:**
  - route senza path
  - utile per wrappare un content layout
- **No Match Route**:
  - route che ha una sola stella "*"
  - matcha solo quando non c'è nulla

primo accesso layout

- se scriviamo outlet → home
- se scriviamo /about → about
- se scriviamo una cosa che non esiste → *

```
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>
  );
}
```

Navigazione:

Changing the location URL will re render the Router, and all Routes will be evaluated

- **Link to**
- **useNavigate**() → ritonra una funzionae che triggera navigazione

```
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>
  );
}
```

Navlink: molto comodo nelle navbar → si porta dietro la classe active

- `<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

Route dinamiche: possono essere inseriti dei parametri nella route;
Per poter usare questo valore dobbiamo usare usePsrams()

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

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

Esempio:

```
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: attributo del DOM che ci permette di passare informazioni quando ci spostiamo da una pagina ad un'altra.
Oggetto viene convertito in stringa → quindi va riconvertito in ciò che mi serve.

```
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;
```

→ recupero informazione che avevamo salvato

Query parameters: si recuperano con metodo useSearchParams()

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

- searchParams is a URL.searchParams object
https://developer.mozilla.org/en-
US/docs/Web/API/URL/searchParams
You may access each parameter with
searchParams.get('date')
searchParams.get('filter')


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

- Wrap `main.jsx` in `createBrowserRouter`
- 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`)

⚠ Warning ⚠

Never use a "plain hyperlink" `<a>`
Never use a "form submission" `<form action='…'>`

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