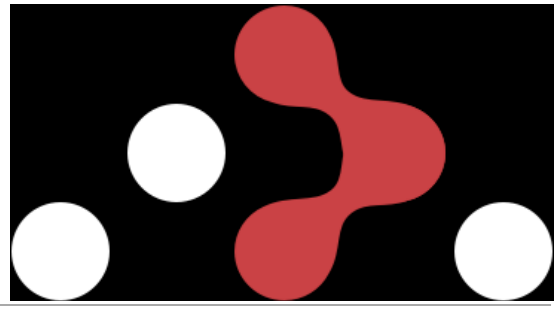
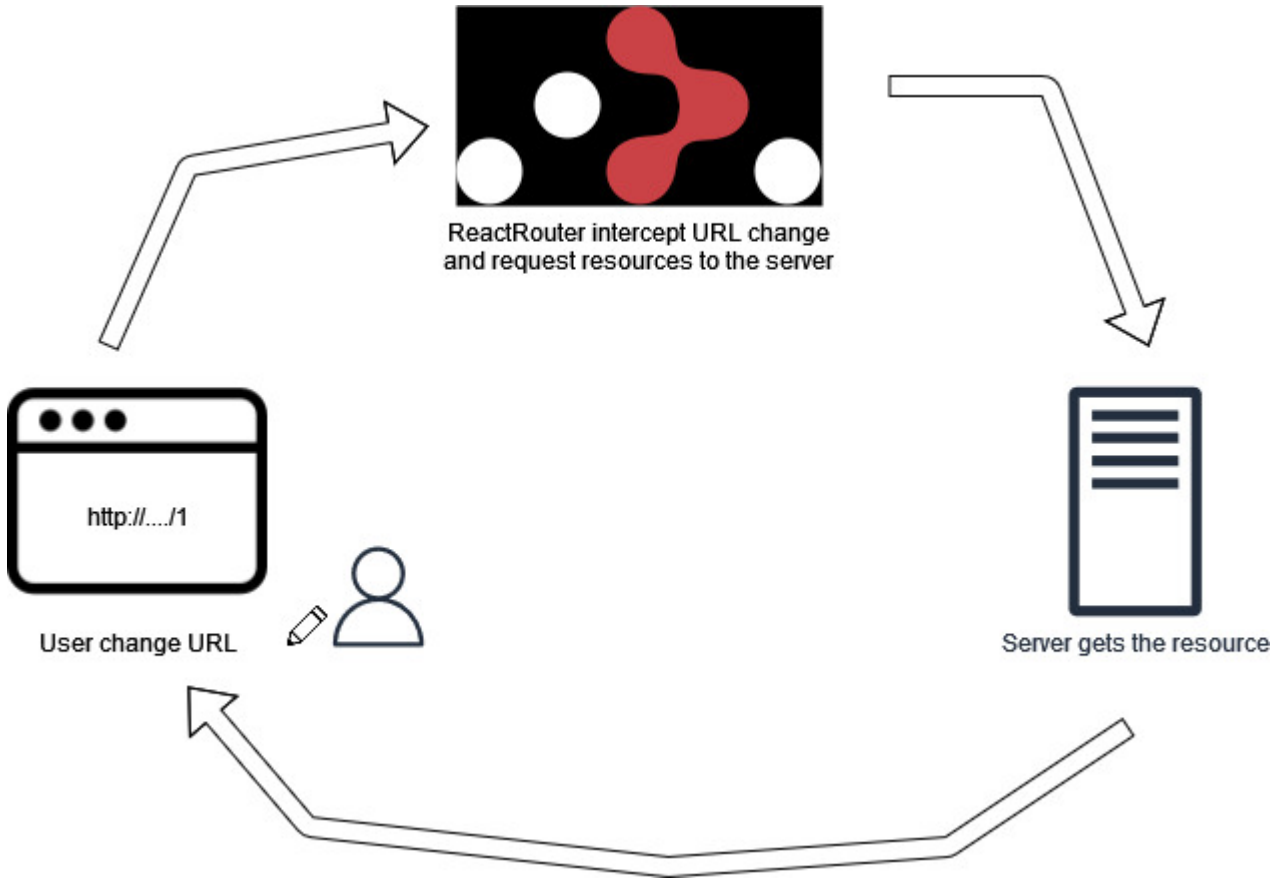
ReactJS Routing



NAVIGATE THROUGH COMPONENTS AND URLS

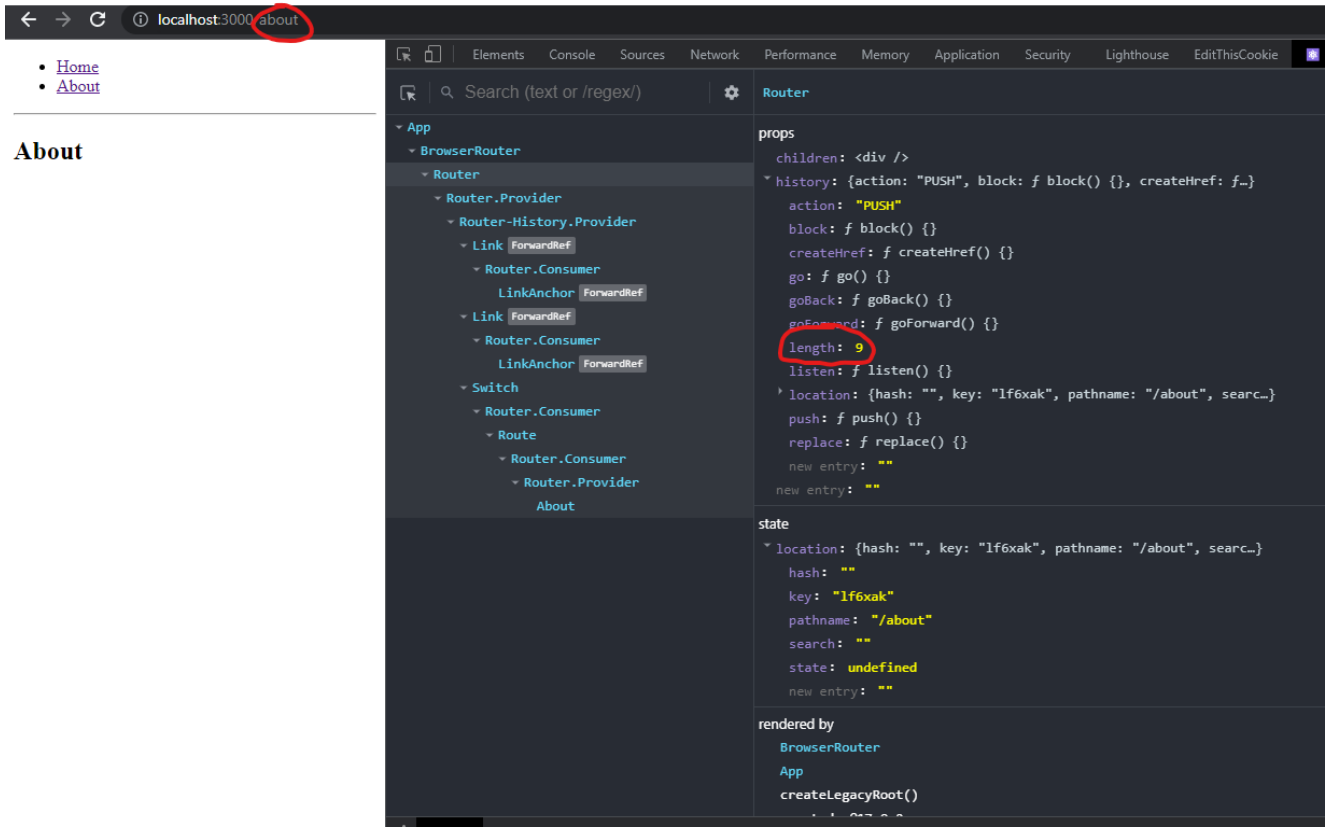
What’s routing

- Routing is the **ability to move between different parts of an application**, when a user clicks an element (link, button, etc…) or when a URL is entered
- Routing is **not native in React**, as per many other function you need an external library. We’ll use React-Router
- Routing depends both on React, to render and make component interactable, and the server who is serving the website to get the correct resource for the URL



Routers

- This library include 3 routers, the main component handling the route change
  - **BrowserRouter** → uses HTML5 history API
  - **HashRouter** → uses a portion of URL hashed
  - MemoryRouter → keeps the history of the URLs in memory, used in native apps
- Each time the **URL change a record in history is added** this is used to keep track of the current location and re-render the application with appropriate component



Routes

- The Route component is one of the most important, it **renders** the appropriate component when the **current location matches the route's path**
- **Path is a *prop*** in Route component, every URL starting with that path will be handled by this route. E.g.
  - /products
  - /products/1
  - /products/…
- **To avoid path wildcard** Route accept the **exact** prop

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

Routes render methods

- There are 3 ways to tell the Route what has to be rendered
  - Route component props → use the React.createElement function
  - Route render function → useful for in-line rendering
  - Route children function → this is the recommended method

```
<Route path="/user/:userName">
  <User />
</Route>
{/*
```

```
"/}
<Route exact path="/" component={Home} />
<Route
```

```
<Route
  render={() => {
    return (
      <NoMatch>
        <Home />
      </NoMatch>
    );
  }}
></Route>
```

Switch

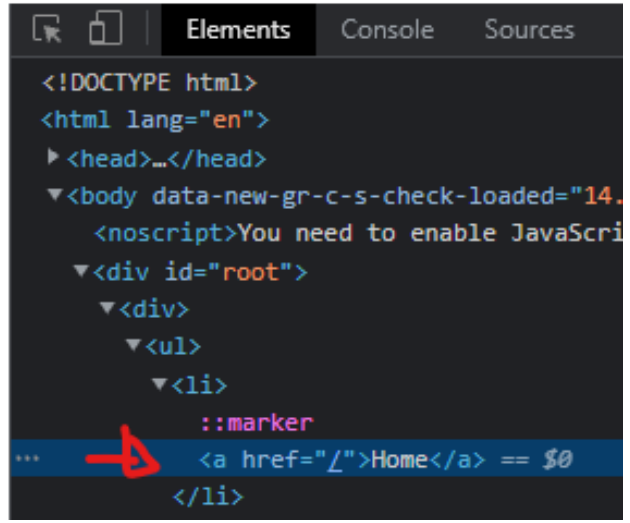
- The switch component is used to wrap multiple Route component and it picks only the first matching Route (comparing the path) that matches among its children
- The switch component **renders a route exclusively**. In contrast every route that matches the location renders *inclusively*.

```
<Switch>
  <Route path="/product">
    <Product id={0} />
  </Route>
  <Route path="/:user">
    <User />
  </Route>
  <Route>
    <NoMatch />
  </Route>
</Switch>
```

Link

- The Link component provides declarative, **accessible navigation around the application**
- The link component render an hyperlink
- Rather than the classic HTML links, **Link doesn't reload the page but changes the UI**

```
<ul>
  <li>
    <Link to="/">Home</Link>
  </li>
  <li>
    <Link to="/about">About</Link>
  </li>
</ul>
```



Dynamic & Nested paths

- Each time a page change a **match object** is created. Match object contains
  - The url → the matched part of the URL
  - The path → the route's path
  - The parameters → the parameters passed to the url
- Sometimes you don't know all the possible routes for your application up front
- In this case it helps to have a **dynamic router** that is able to generate routes as needed
- In some scenarios you might need to create **nested routes**, this is possible using the **match component**
  - /category/food
  - /category/cinema
  - /category/…

```
<BrowserRouter>
  <Switch>
    <Route path="/category">
      <Category />
    </Route>
  </Switch>
</BrowserRouter>
```

```
<ul>
  {CATEGORIES.map((cat) => (
    <li key={cat}>
      {/* N.B. if you use the match object you don't need to write the '/' at the beginning of the path. */}
      <Link to={`${match.url}/${cat}`}>{cat}</Link>
    </li>
  ))}
</ul>
```

```
<Switch>
  <Route path={`${match.url}/:categoryName`}>
    <CategoryDetails />
  </Route>
</Switch>
```

```
function CategoryDetails() {
  let match = useRouteMatch();
  let params = match.params;
  let categoryName = params.categoryName;
  // or
  // let { categoryName } = useParams();

  return (
    <div>
      This is the category <b>{categoryName}</b> with url <b>{match.url}</b> and
      defined in route with path <b>{match.path}</b>
    </div>
  );
}
```

Private Resource

- We can define private route by creating components that check the application state
- In our PrivateRoute component we need to check if the user is logged in otherwise redirect to the /login page
- React route provides a component <Redirect …/> able to redirect the user to a desired component

```
<Switch>
  <Route path="/public">
    <PublicPage />
  </Route>
  <Route path="/login">
    <LoginPage />
  </Route>
  <PrivateRoute path="/protected">
    <ProtectedPage />
  </PrivateRoute>
</Switch>
```