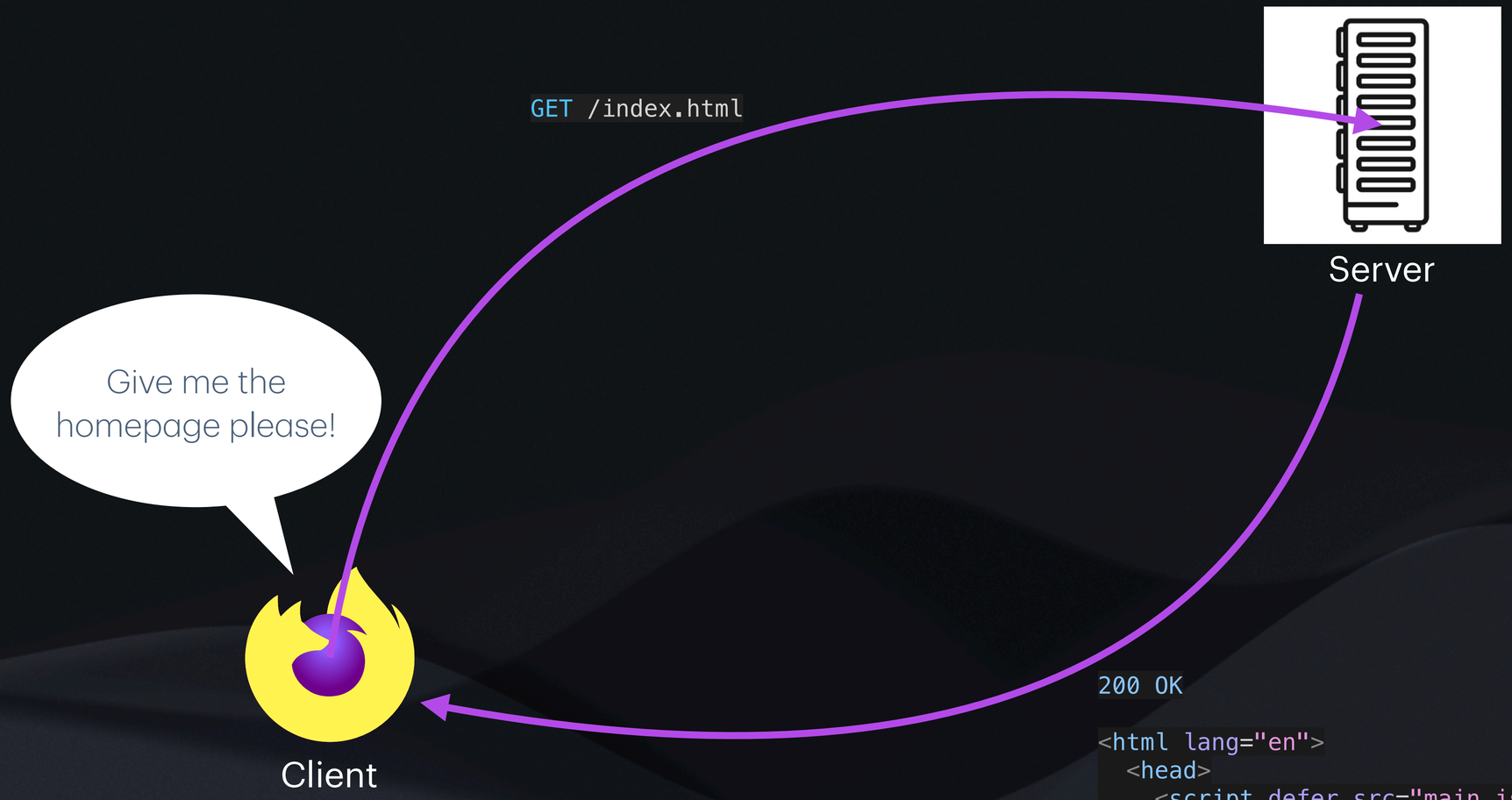
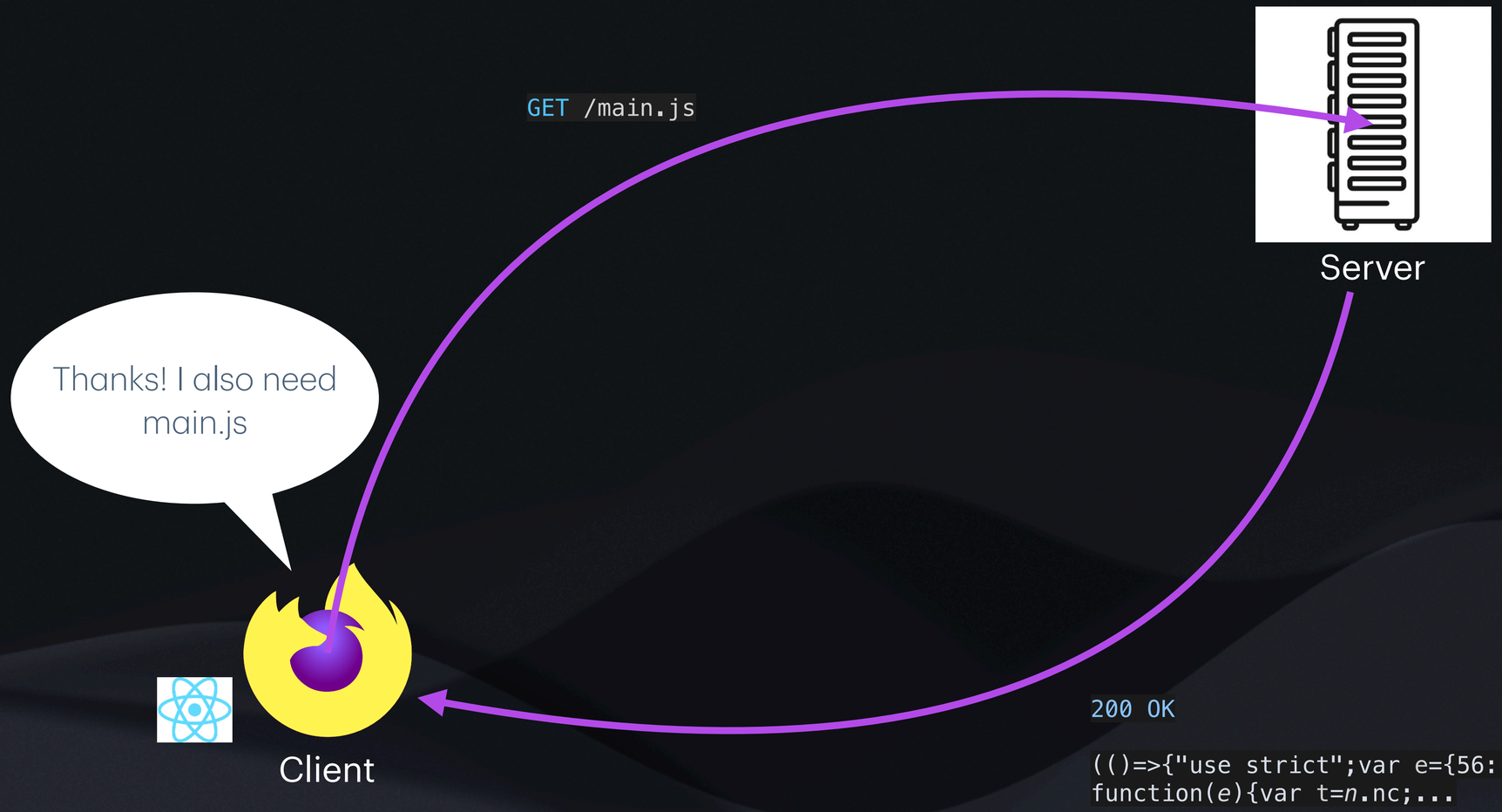
Client-Side Routing 

Single Page Apps (now with multiple pages!)

Client-Side Routing

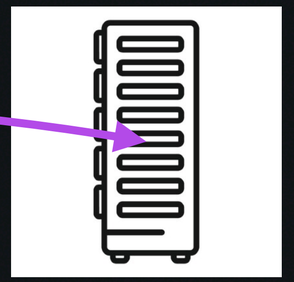


Client-Side Routing



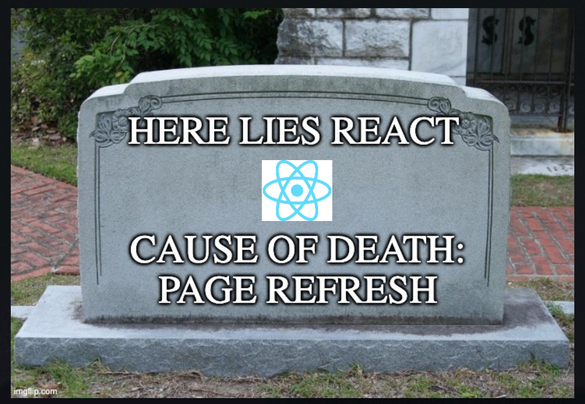
Client-Side Routing

Sure, here's a new HTML page!



Server

```
GET /about.html
```



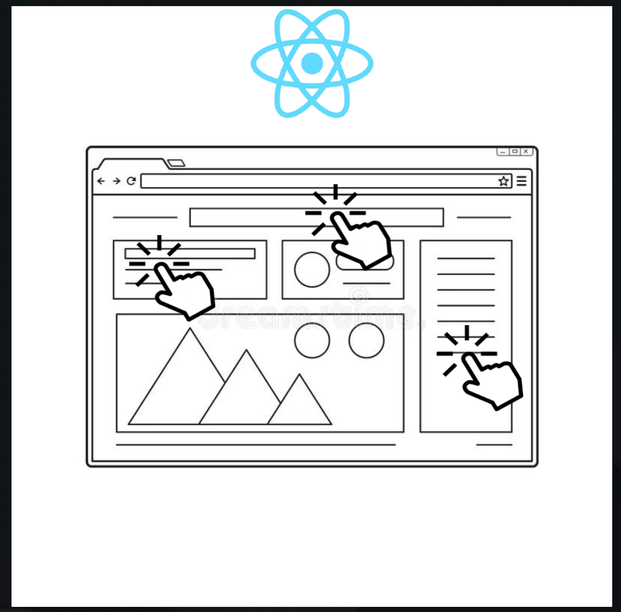
My user clicked "About" on the navbar

Client

```
200 OK

<html lang="en">
  <head>
    <script defer src="main.js"></script>
...
```

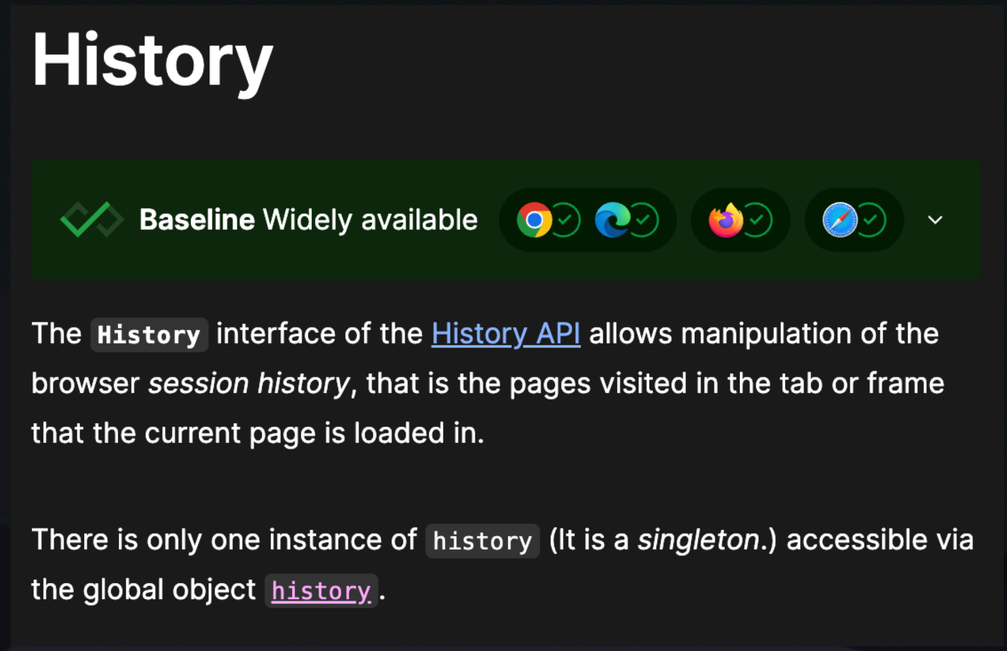
Client-Side Routing



- React is designed to be a “single page application” (SPA)
- Every time we reload the page, we get a new HTML and bundled JS file (e.g. main.js or bundle.js)
- How do we prevent this?

history & location

- Browsers have many ways to interact with the url in the address bar
- Maybe we can intercept the default behavior to prevent page reloads whenever the user clicks on a link?
- And if the user loads the page from a different route than “/”, we can load the correct view?



History

Baseline Widely available

The `History` interface of the History API allows manipulation of the browser *session history*, that is the pages visited in the tab or frame that the current page is loaded in.

There is only one instance of `history` (It is a *singleton*.) accessible via the global object `history`.

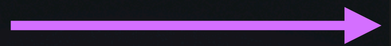
developer.mozilla.org/en-US/docs/Web/API/History

history & location

User loads the page at “/support-us”

```
location.pathname
```

User clicks a link that goes to “/about”

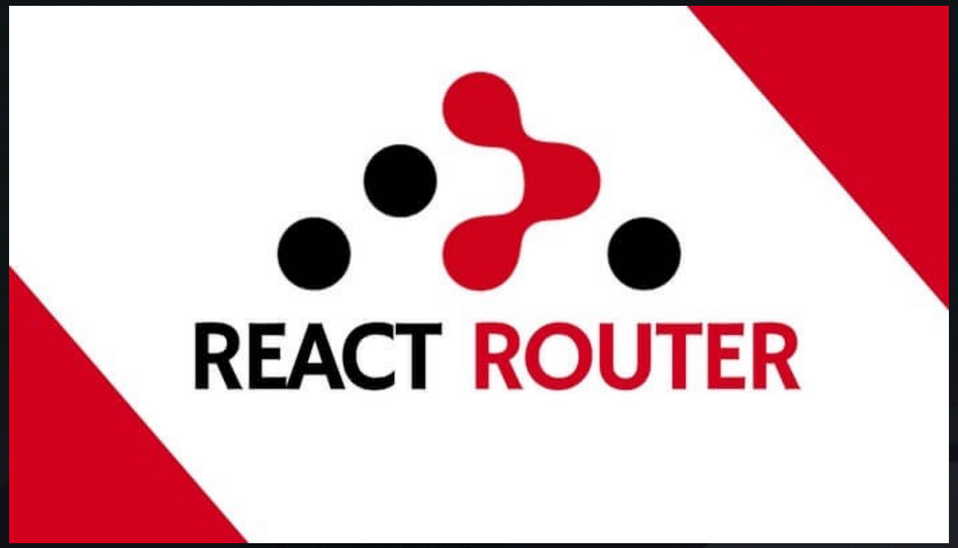


```
const url = new URL(location);
url.pathname = "/about";
history.pushState({}, "", url);
```

User clicks a “previous page” button

```
history.back()
```

React-Router v7



reactrouter.com/home

React-Router v7

- Allows us to create routes that only exist on our front-end (React)
- When the user clicks links, React-Router takes care of managing history and location
- When the user first visits our page, React-Router sends them the appropriate components

Setting Up Your Router

```
import { BrowserRouter, Routes } from “react-router";

...

const root = createRoot(document.getElementById("root"));
root.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
);
```

- React-Router has three different “modes”: Declarative, Data, Framework.
  - Let’s use Declarative for now
- React-Router exports **`BrowserRouter`**, which must be at the root of our application
  - Uses something called “Context” under the hood

Setting Up Your Routes

```
const App = () => {
  ...
  return (
    <div>
      <NavBar />
      <Routes>
        <Route path="/my-account" element={<MyAccount />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="/" element={<Homepage />} />
      </Routes>
    </div>
  );
};
```

- **<Routes>** let's you declare a series of **<Route>** components
- **<Route>** takes a path and an element. When the path matches, it renders the element
- It goes *in order*, and renders the first match it finds

🚨🚨🚨 These are not Express routes!

Setting Up Your Routes

```
<Routes>
  <Route path="/my-account" element={<MyAccount />} />
  <Route path="/about" element={<About />} />
  <Route path="/contact" element={<Contact />} />
  <Route path="/ducks">
    <Route exact path="/" element={<DuckList />} />
    <Route path=":id" element={<DuckDetails />} />
    <Route path="new" element={<AddDuck />} />
  </Route>
  <Route path="/" element={<Homepage />} />
</Routes>
```

/ducks => DuckList
/ducks/3 => DuckDetails for 3
/ducks/new => AddDuck

- Routes can be nested
- Routes can take parameters
- You can tell React-Router to only match exactly that route
  - Default: Fuzzy Matching

Link & NavLink

```
<Link to="/ducks">All Ducks</Link>
<Link to="/ducks/new">Add Duck</Link>
<Link to={`/ducks/${duckId}`}>Duck Details for {duckId}</Link>
```

- Links are a tags that don’t reload the page
- Pass them a destination and they’ll direct the user there
  - Triggers a change in location, which renders a new Route

Link & NavLink

```
<NavLink
  to="/ducks/new"
  className={({ isActive, isPending }) =>
    isPending ? "pending" : isActive ? "active" : ""
  }
>
  Add Duck
</NavLink>
```

- NavLinks are just like Links, except it’s easier to style them “active”
- React-Router will automatically give them the “active” class when they are active

useNavigate

```
import { useNavigate } from "react-router";

const AddDuck = () => {
  let navigate = useNavigate();

  const handleSubmit = async (event) => {
    event.preventDefault();
    ...
    navigate("/");
  }
...
```

- useNavigate is a hook, just like useState and useEffect
- Useful when we want to make the user go directly to a new route
  - e.g. after submitting a form

Client-Side Routing Practice