21.4.4 Implement the Navigation Links and URL Parameters

You now have client-side routing in place, but there's no natural way for users to get to these other pages. The app needs navigation links!

With React Router, however, you can't simply use `<a>` elements. An element like `<a href="/login">` would cause the browser to refresh and make a new request to your server for `/login`. That would defeat the whole purpose of React and its single-page goodness. Instead, you can use React Router's `Link` component. This component will change the URL while staying on the same page.

Open the `Header` component located at `src/components/Header/index.js` and add the following `import` statement to the top of the file:

```
import { Link } from 'react-router-dom';
```

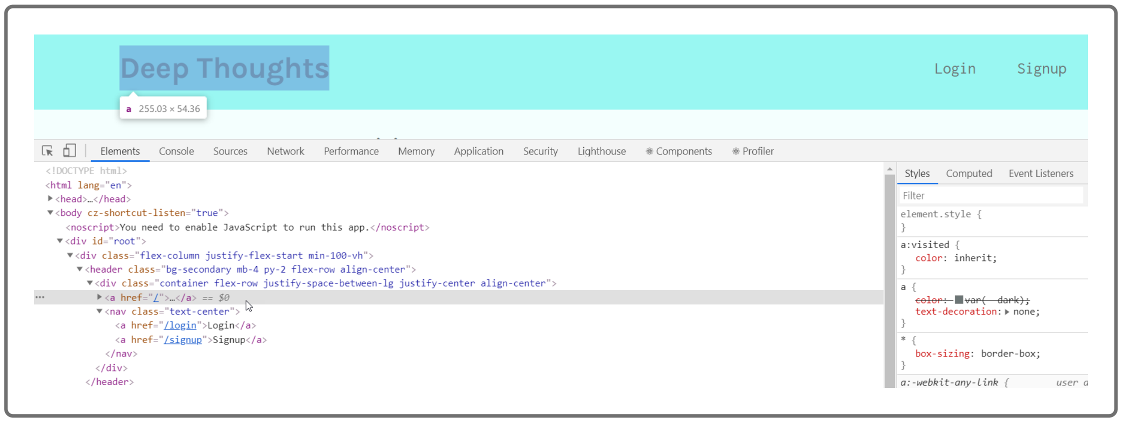
Next, update the `Header` functional component to look like the following code:

```
const Header = () => {
  return (
    <header className="bg-secondary mb-4 py-2 flex-row align-center">
      <div className="container flex-row justify-space-between-lg justify-ce
        <Link to="/">
          <h1>Deep Thoughts</h1>
        </Link>

        <nav className="text-center">
          <Link to="/login">Login</Link>
          <Link to="/signup">Signup</Link>
        </nav>
      </div>
    </header>
  );
};
```

In the preceding code, note how the syntax for the `Link` component uses a `to` attribute instead of an `href` attribute.

Switch over to the browser and use the Chrome DevTools to inspect one of the `Link` components that you just added. You'll see that in the real DOM, the `Link` becomes a regular `<a>` element. The following image demonstrates what you should see:



```
<!DOCTYPE html>
<html lang="en">
▶<head>…</head>
▼<body cz-shortcut-listen="true">
   <noscript>You need to enable JavaScript to run this app.</noscript>
  ▼<div id="root">
    ▼<div class="flex-column justify-flex-start min-100-vh">
      ▼<header class="bg-secondary mb-4 py-2 flex-row align-center">
        ▼<div class="container flex-row justify-space-between-lg justify-center align-center">
...       ▶<a href="/">…</a> == $0
          ▼<nav class="text-center">
             <a href="/login">Login</a>
             <a href="/signup">Signup</a>
           </nav>
         </div>
       </header>
```

```
element.style {
}
a:visited {
  color: inherit;
}
a {
  ~~color: var(--dark);~~
  text-decoration: ▶ none;
}
* {
  box-sizing: border-box;
}
a:-webkit-any-link {            user a
```

In React's virtual DOM, however, the `Link` component is still there, ready to handle the client-side routing for you. To see this in action, click the Login and Signup links in the header. Notice that the URL changes, but the page doesn't refresh. React simply renders a different component.

Next, you'll need to update the thoughts listed on the homepage to render the `SingleThought` component when the thought text is clicked and the `Profile` component when the author name is clicked. Open the `ThoughtList` component located at `src/components/ThoughtList/index.js` and add the following `import` statement to the top of the file:

```
import { Link } from 'react-router-dom';
```

Then in the `ThoughtList` functional component's `return` statement, update the `<p className="card-header">` element to the following code:

```
<p className="card-header">
  <Link
    to={`/profile/${thought.username}`}
    style={{ fontWeight: 700 }}
    className="text-light"
  >
    {thought.username}
  </Link>{' '}
  thought on {thought.createdAt}
</p>
```

Also update the `<div className="card-body">` element to the following code:

```
<div className="card-body">
  <Link to={`/thought/${thought._id}`}>
    <p>{thought.thoughtText}</p>
    <p className="mb-0">
      Reactions: {thought.reactionCount} || Click to{' '}
      {thought.reactionCount ? 'see' : 'start'} the discussion!
    </p>
  </Link>
</div>
```

In both instances, we've added an extra path to the end of the `Link` component. For example, `/profile/${thought.username}` would become `/profile/Zoe66` for that particular user. Test this out by navigating to the homepage and clicking on a thought's author name or text content. Unfortunately, you'll see that the `NoMatch` component gets rendered instead of `Profile` or `SingleThought`.

Why is that? In `App.js`, we specified that the `/profile` route must be exactly `/profile` for the `Profile` component to render. Alas, `/profile/Zoe66` or any `/profile/${username}` URL counts as a completely different route. Fortunately, React Router supports URL parameters. This is very similar to using parameters in Express.js on the back end.

REWIND

An Express.js route with a parameter would look like the following code:

```
app.get('/api/animals/:id', (req, res) => {
  console.log(req.params.id);
});
```

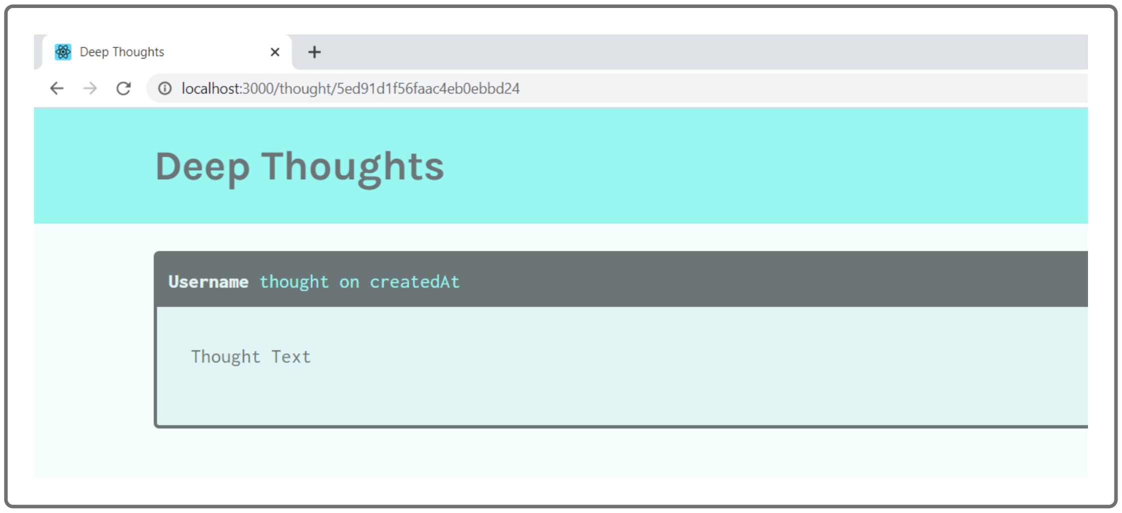
In this example, `:id` is the parameter, meaning requests to `/api/animals/1` or `/api/animals/100` would fall under the same route.

In `App.js`, update the following `Route` components to include the following parameters:

```
<Route exact path="/profile/:username?" component={Profile} />
<Route exact path="/thought/:id" component={SingleThought} />
```

The `?` means this parameter is optional, so `/profile` and `/profile/myUsername` will both render the `Profile` component. Later on, we'll set up `/profile` to display the logged-in user's information.

Save your changes and test the app in the browser again. Clicking on a thought's text content should display the `SingleThought` component, as the following image shows:



Deep Thoughts

Username thought on createdAt

Thought Text

The `SingleThought` component displays hardcoded data for now, but we'll update this component to be more dynamic. Notice how the URL includes the ID of the thought being viewed. We can grab that ID and query the database for the thought's full information.

How would you access the ID from the URL, though? You could use something like `document.location` to parse out the ID, but React Router has a better method built in.

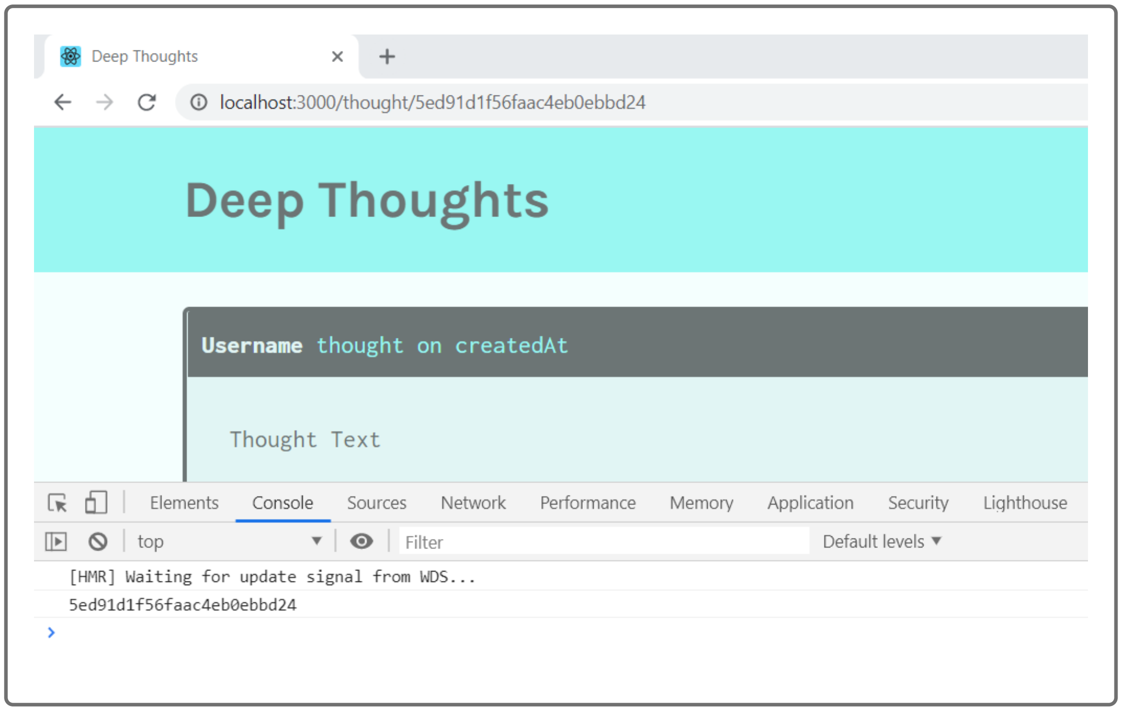
In the `SingleThought.js` file, use the following code to import a new React Hook:

```
import { useParams } from 'react-router-dom';
```

In the `SingleThought` functional component, add the following lines before the `return` statement:

```
const { id: thoughtId } = useParams();
console.log(thoughtId);
```

Open the DevTools console and navigate to the Single Thought page. The following image shows what you should see:



```
[HMR] Waiting for update signal from WDS...
5ed91d1f56faac4eb0ebbd24
```

As you can see, the ID that was logged matches the ID from the URL. In the next section, we'll use that ID to retrieve more information about the given thought and populate the Single Thought page with real data.