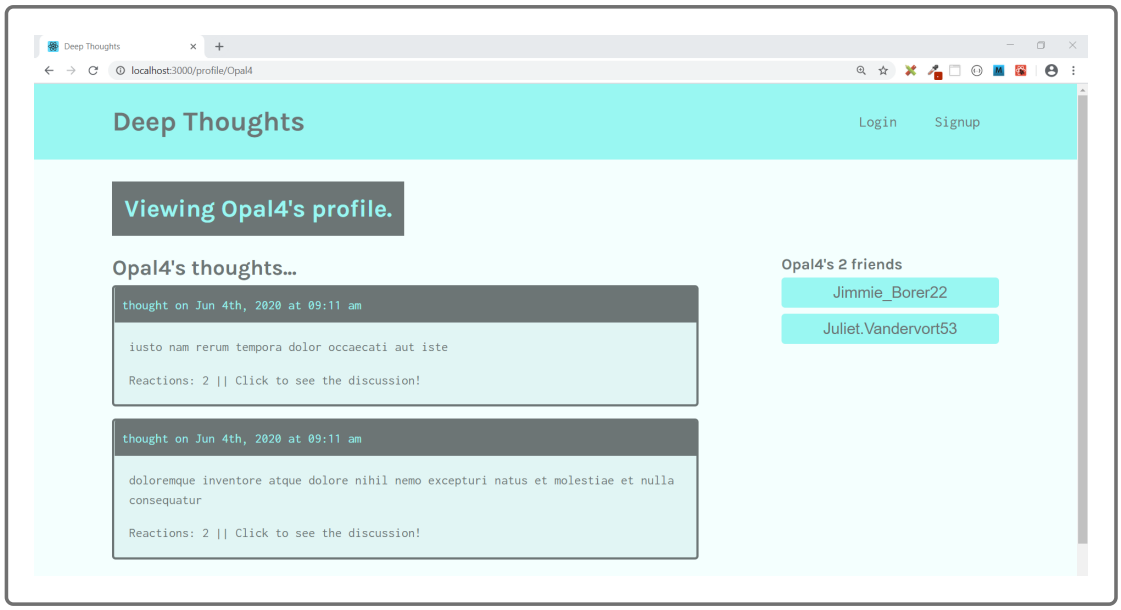
21.4.6 Build the Profile Page

Populating the Profile page will be very similar to the Single Thought page. For reference, the completed page will look like the following image:



The data on this page is a combination of the user's information, thoughts, and friends. Fortunately, you've already written a `user()` query in GraphQL that consolidates these data points. We just need to set up the front end to use it.

In the `utils/queries.js` file, add the following code:

```
export const QUERY_USER = gql`
  query user($username: String!) {
    user(username: $username) {
      _id
      username
      email
      friendCount
      friends {
        _id
        username
      }
      thoughts {
        _id
        thoughtText
        createdAt
        reactionCount
      }
    }
  }
`;
```

In the `Profile.js` file, add the following `import` statements:

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

import ThoughtList from '../components/ThoughtList';

import { useQuery } from '@apollo/react-hooks';
import { QUERY_USER } from '../utils/queries';
```

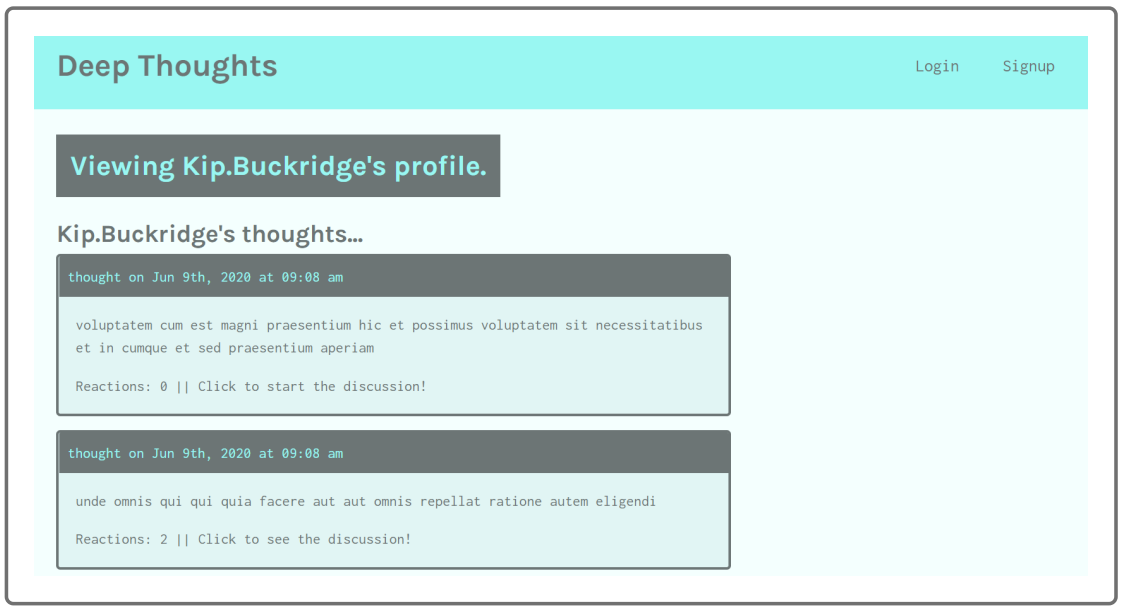
Next, update the `Profile` functional component to look like the following code:

```
const Profile = () => {
  const { username: userParam } = useParams();

  const { loading, data } = useQuery(QUERY_USER, {
    variables: { username: userParam }
```

```
  });

  const user = data?.user || {};

  if (loading) {
    return <div>Loading...</div>;
  }

  return (
    <div>
      <div className="flex-row mb-3">
        <h2 className="bg-dark text-secondary p-3 display-inline-block">
          Viewing {user.username}'s profile.
        </h2>
      </div>

      <div className="flex-row justify-space-between mb-3">
        <div className="col-12 mb-3 col-lg-8">
          <ThoughtList thoughts={user.thoughts} title={`${user.username}'s t
        </div>
      </div>
    </div>
  );
};
```

Again, this is very similar to the logic in `SingleThought.js`. The `useParams` Hook retrieves the username from the URL, which is then passed to the `useQuery` Hook. The `user` object that is created afterwards is used to populate the JSX. This includes passing props to the `ThoughtList` component to render a list of thoughts unique to this user.

Test the page in the browser. The Profile page should now look like the following image:



As you can see from the image, the user's thoughts are displaying, but we are missing their friend list. We will most likely need to render friends in other areas of the app, so populating them in a separate component would be best.

In the `src/components` directory, create a new folder called `FriendList`. In this folder, create a new `index.js` file.

We plan on passing three props to the `FriendList` component: the username whose friends these belong to, the friend count, and the actual array of friends. With this data, we can display a different message if the user has no friends. Otherwise, we can map the friends into elements that link to their profiles.

In the `FriendList/index.js` file, write the following code:

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

const FriendList = ({ friendCount, username, friends }) => {
  if (!friends || !friends.length) {
    return <p className="bg-dark text-light p-3">{username}, make some frien
  }

  return (
```

```
    <div>
      <h5>
        {username}'s {friendCount} {friendCount === 1 ? 'friend' : 'friends'
      </h5>
      {friends.map(friend => (
        <button className="btn w-100 display-block mb-2" key={friend._id}>
          <Link to={`/profile/${friend.username}`}>{friend.username}</Link>
        </button>
      ))}
    </div>
  );
};

export default FriendList;
```

Revisit the `Profile.js` file and add the following `import` statement:

```
import FriendList from '../components/FriendList';
```

Then update the second `<div className="flex-row">` element in the `Profile` functional component to look like the following code:

```
<div className="flex-row justify-space-between mb-3">
  <div className="col-12 mb-3 col-lg-8">
    <ThoughtList thoughts={user.thoughts} title={`${user.username}'s thought
  </div>

  <div className="col-12 col-lg-3 mb-3">
    <FriendList
      username={user.username}
      friendCount={user.friendCount}
      friends={user.friends}
    />
  </div>
</div>
```

Test it out in the browser. The friend list will appear next to the user's thoughts. Make sure to click on the friend names to test routing to other Profile pages. Once everything looks good, save your work with Git and merge the feature branch into `develop`. You can also close the GitHub issue at this point.

Before moving on to the next lesson, take a moment for a quick knowledge check:

Given the following React Router setup, which component would be rendered for the route `/post` ?

```
<Switch>
  <Route exact path="/" component={Home} />
  <Route exact path="/post/:id" component={Post} />
  <Route component={NoMatch} />
</Switch>
```

- Home
- Post
- NoMatch ✔

Response-specific feedback
Yes, without a `?` , the route for the `Post` component must be `/post/something` .

For routing, why should you use React Router's `Link` component instead of the `<a>` element?

- The `<a>` element causes the page to refresh. ✔
- The `<a>` element can't have click handlers attached to it.
- The `Link` component has extra styling options that can be added.
- The `Link` component renders faster.

Response-specific feedback
Yes, the browser treats the `<a>` element as a new request for server-side resources, and we don't want the page to refresh when using React Router.

Finish ▶