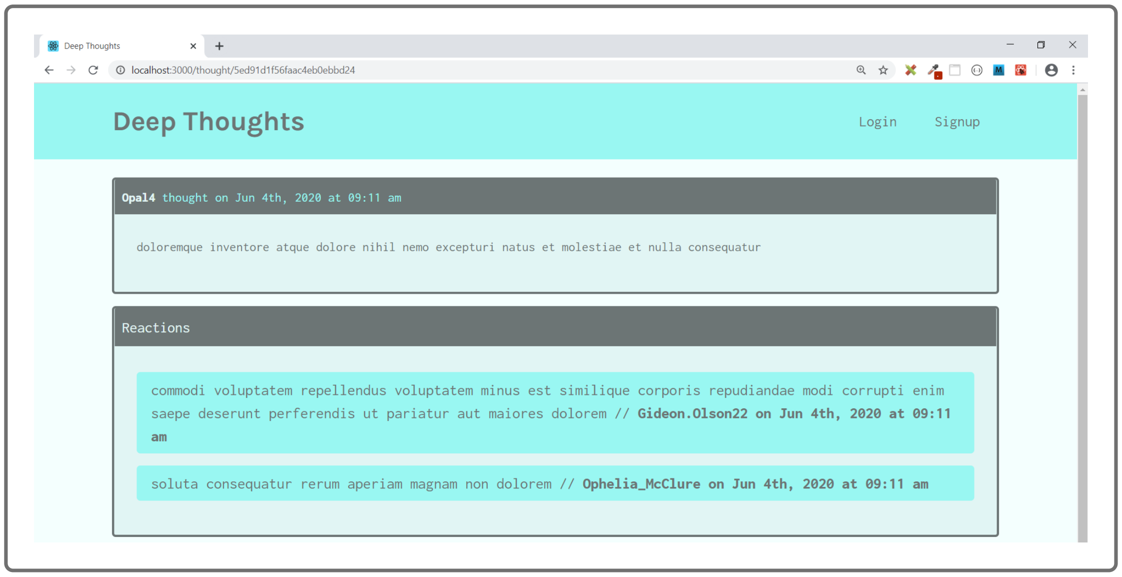
21.4.5 Build the "Single Thought" Page

Once completed, the Single Thought page will look like the following image:



Deep Thoughts

Login Signup

Opal4 thought on Jun 4th, 2020 at 09:11 am

doloremque inventore atque dolore nihil nemo excepturi natus et molestiae et nulla consequatur

Reactions

commodi voluptatem repellendus voluptatem minus est similique corporis repudiandae modi corrupti enim saepe deserunt perferendis ut pariatur aut maiores dolorem // **Gideon.Olson22 on Jun 4th, 2020 at 09:11 am**

soluta consequatur rerum aperiam magnam non dolorem // **Ophelia_McClure on Jun 4th, 2020 at 09:11 am**

All of the information displayed on this page is already available through the `thought()` query that you set up earlier using GraphQL. Now you just need to use this query in the front end of your app.

Open the `utils/queries.js` file and add the following query definition:

```
export const QUERY_THOUGHT = gql`
  query thought($id: ID!) {
    thought(_id: $id) {
      _id
      thoughtText
      createdAt
      username
      reactionCount
      reactions {
        _id
        createdAt
        username
        reactionBody
      }
    }
  }
`;
```

In the `SingleThought.js` file, import this query and the `useQuery` Hook by using the following lines of code:

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

Next, update the `SingleThought` functional component to include the following code before the `return` statement:

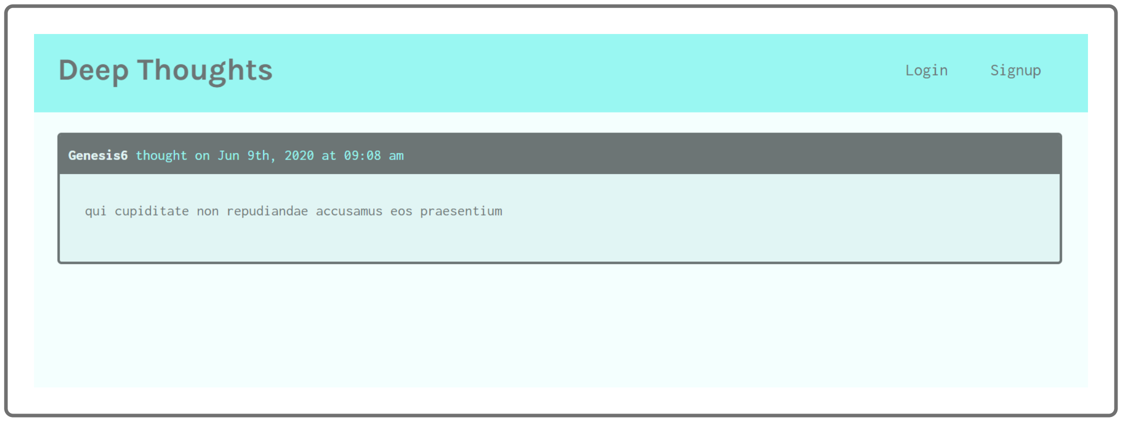
```
const { id: thoughtId } = useParams();

const { loading, data } = useQuery(QUERY_THOUGHT, {
  variables: { id: thoughtId }
});

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

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

This is similar to the query logic that you used on the homepage. The variables `loading` and `data` are destructured from the `useQuery` Hook. The `loading` variable is then used to briefly show a loading `<div>` element, and the `data` variable is used to populate a `thought` object. There is one difference, however. The `useQuery` Hook was given a second argument in the form of an object. This is how you can pass variables to queries that need them. The `id` property on the `variables` object will become the `$id` parameter in the GraphQL query.

Now that you have a `thought` object, update the JSX in the `return` statement to use its properties. The JSX should look like the following code:

```
<div>
  <div className="card mb-3">
    <p className="card-header">
      <span style={{ fontWeight: 700 }} className="text-light">
        {thought.username}
      </span>{' '}
      thought on {thought.createdAt}
    </p>
    <div className="card-body">
      <p>{thought.thoughtText}</p>
    </div>
  </div>
</div>
```

In the browser, the Single Thought page should now look like the following image:



Genesis6 thought on Jun 9th, 2020 at 09:08 am

qui cupiditate non repudiandae accusamus eos praesentium

The thought text is displaying correctly, but there are no reactions yet. Reactions are available on the `thought.reactions` property, so you would just need to map these into JSX elements. To keep the code organized and reusable, however, it would be better to create a separate component for listing reactions.

In the `src/components` directory, create a new folder called `ReactionList`. In this folder, create a new `index.js` file. Then in the `ReactionList/index.js` file, add the following code:

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

const ReactionList = ({ reactions }) => {
  return (

  );
};

export default ReactionList;
```

The `ReactionList` component will be given the `reactions` array as a prop. This array can then be mapped into a list of `<p>` elements. Each reaction also includes the author's name, which should route to the Profile page when clicked. Thus, we'll need to import the `Link` component.

In the `return` statement, add the following JSX code:

```
<div className="card mb-3">
  <div className="card-header">
    <span className="text-light">Reactions</span>
  </div>
  <div className="card-body">
    {reactions &&
      reactions.map(reaction => (
        <p className="pill mb-3" key={reaction._id}>
          {reaction.reactionBody} {'// '}
          <Link to={`/profile/${reaction.username}`} style={{ fontWeight: 70
            {reaction.username} on {reaction.createdAt}
          </Link>
        </p>
      ))}
  </div>
</div>
```

Switch back to the `SingleThought.js` file and import the new `ReactionList` component by using the following line of code:

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

Then update the JSX in the `SingleThought` functional component's `return` statement to look like the following code:

```
<div>
  <div className="card mb-3">
    <p className="card-header">
      <span style={{ fontWeight: 700 }} className="text-light">
        {thought.username}
      </span>{' '}
      thought on {thought.createdAt}
    </p>
    <div className="card-body">
      <p>{thought.thoughtText}</p>
    </div>
  </div>
  {thought.reactionCount > 0 && <ReactionList reactions={thought.reactions} 
</div>
```

The only new addition is adding the `ReactionList` component at the bottom, passing in the `reactions` array as a prop. We combined this with a `thought.reactionCount > 0` expression to prevent rendering the reactions if the array is empty.

Test the page in the browser again, making sure the reactions render and that the author's name displays the `Profile` component when clicked. It works! Now that the Single Thought page is working, we can focus on finishing out the Profile page in the next section.