Avoid Props Drilling with Context

In this lesson, we will introduce context to our Bank Application to avoid the props drilling.

One easy solution is to look at the `Root` component where we began passing props down and finding a way to introduce a context object in there.

Going off that solution, we could create a context object with no initial default values above the `Root` class declaration:

```
const { Provider, Consumer } = createContext()
class Root extends Component {
  state = {
    loggedInUser: null
  }

  ...
}
```

Then we could wrap the main `App` component around the `Provider` with a value prop.

```
class Root extends Component {
  state = {
    loggedInUser: null
  }
  ...
  render () {
    ...
    return loggedInUser ? (
      <Provider value={this.state.loggedInUser}>
        <App loggedInUser={loggedInUser} />
      </Provider>
    )
...
```

Initially, the `Provider` `value` prop will be `null`, but as soon as a user logs in and the `state` is updated in `Root`, the `Provider` will also receive the current `loggedInUser`.

With this done, we can import the `Consumer` wherever we want and do away

with passing props needlessly down the component tree.

For example, here's the `Consumer` used in the `Greeting` component:

```
import { Consumer } from '../Root'
const Greeting = () => {
  return (
    <Consumer>
      {user => <p>Welcome, {user.name}! </p>}
    </Consumer>
  )
}
```

We could go ahead and do the same everywhere else we've passed the `loggedInUser` prop needlessly. The app works just as before, only we got rid of passing down `loggedInUser` over and over again.

In the next lesson, we'll centralize the implementation details of the user state and provider in one place.