- ⏱ 2 hours

Context Practice

In this exercise, you will practice using React contexts while gaining familiarity with the Jest testing framework for React. By the end, you should be able to:

- Create a custom React context-wrapper hook
- Connect a React component to a context
- Change the content of a React component dynamically based on context
- Change the context from a React component
- Run Jest tests and use any error messages to help debug your code

Resources

You can use any of the following resources during this practice:

- React Router documentation
- React documentation
- Mozilla Developer Network
- Any of your previous projects or assessments

Tests

This practice contains tests to help you complete the phases below. When run, they will report to you when expectations of *functionality* are met. It is still up to you to make sure that your code does not produce errors in the console or that it is not poorly formatted.

The tests are grouped by phase. Use these test groupings to know when each phase is complete from a functionality perspective.

The tests are located in _src/__tests___. Do not modify the files in that directory.

There is also a file named *src/setupTests.js*. Do not modify or delete that file either. It sets up the testing environment.

Recommendations/Requirements

It is recommended that you consistently review the output of the build in your Terminal or browser console to make sure that you do not have any warnings about things like unused variables.

After each phase, the application should render in the browser without errors.

Do not change the location of files in the *src/components* folder. The tests expect the existence of those files.
When instructed to do something like "Use the `Blah` component in the `render` method", that implies that you may need to import something like the `Blah` component. In most cases, you will *not* be instructed to import something; rather, you should use your knowledge of how JavaScript modules work to figure that out on your own.

Phase 0: Getting the app running

This starter contains an application generated by Create React App using the `@appacademy/react-v17` template and providing `react-router-dom`. It contains some styling for you in the *index.css* file for CSS classes that you will apply during the steps so that things are not rendered with the default browser layout.

It also contains the tests for the components that you will create.

The React application is already wrapped in the `BrowserRouter` component from `react-router-dom` in *src/index.js*.
After cloning, run `npm install` to install the dependencies.
You can run your tests with `npm test`. Follow the on-screen instructions to quit the test runner. If you want to run your tests only once, use the following command.

```
CI=true npm test
```

To run the tests for the step you're on, run `npm test -- <file>` where `<file>` is the test file you're working on.

Here's the list of test files to run individually:

- `npm test 1-SelectedCoffeeBean`
- `npm test 2-SetCoffeeBean`

Type *Control+C* to stop the test runner.
You can also run all the tests at once by simply running `npm test`.
To run the application, use the command `npm start`.

macOS

If you are on macOS, this should also open a Google Chrome window to see your application.

Windows

If you are on Windows, you can run `npm start` from the Terminal Bash prompt in Visual Studio Code which should open your browser for you.
Warning: make sure the browser that it opens is Google Chrome. If you don't have Google Chrome as your default browser, it will open Microsoft Internet Explorer. For the purposes of this class, you must use Google Chrome. Running `npm start` in the Ubuntu window will only start the application; in that case, you will have to open a browser window and navigate to http://localhost:3000/ yourself (by clicking on that link, for example).

Phase 1: SelectedCoffeeBean

In this phase, you will be filling in the code for the `SelectedCoffeeBean` component which will show the name of the selected coffee bean extracted from a context.
Take a look at the context and provider created in *src/context/CoffeeContext.js*. The value of the `CoffeeContext` has a `coffeeBean` that should be used in the `SelectedCoffeeBean` component.
Hint: You may want to create a custom hook to return the values of the `CoffeeContext`.
To test this phase, run `npm test 1-SelectedCoffeeBean`.
Render the `SelectedCoffeeBean` component inside of the `App` component. Inside of the `SelectedCoffeeBean` component, render a `div` as the outermost element of the component with a class of `selected-coffee`. Inside of this `div`, render an `h2` element with exactly the text of the name of the selected coffee bean.

Phase 2: SetCoffeeBean

In the previous phase, you used the `CoffeeContext`'s `coffeeBean` value. In this phase you will set the `coffeeBean` value of the context in the `SetCoffeeBean` component.
To test this phase, run `npm test 2-SetCoffeeBean`.
Do not change the class names and existing attributes for any of the elements rendered in the `SetCoffeeBean` component. The tests rely on these to be present.
Render the `SetCoffeeBean` component inside of the `App` component. Pass in the array of coffee beans exported from the *src/mockData/coffeeBeans.json* mock data file as a prop to the `SetCoffeeBean` called `coffeeBeans`.
A `select` element with `option`s has been rendered for you in the `SetCoffeeBean` component. Make the `select` a *controlled input* that changes the value of the `coffeeBean` in the `CoffeeContext` using the `setCoffeeBeanId` function from the `CoffeeContext`. Finally, initialize the `select`'s `value` attribute to the id of the initially selected coffee bean.
If done correctly, you should see the name of the coffee bean in the `SelectedCoffeeBean` component change when you select a new option.

Wrapping up

Make sure that you pass all the tests by running `npm test`.
Did you find this lesson helpful?
No
Yes