- ⏱ 3 hours

Greenhouse Project

In this project, you will be creating a frontend application that maintains a mock greenhouse. You will display and control the climate of the greenhouse using React context. By the end of the project, you should be able to:

- Create a context, a context Provider with a dynamic value object, and a React custom wrapper hook
- Connect multiple components to a context
- Change the value of a context in a React component

Phase 0: Set Up

- Clone the repo from the `Download` link below and run `npm install`.
- Study the code in each of the files to get a feel for what is going on. You will be using a package called React Slider to visually adjust the thermometer and hygrometer.
- Start the development server by running `npm start` and take a look at what you will be working with.

Phase 1: Greenhouse Theme

In this first phase, you will connect the theme stored in the `ThemeContext` to change the look of the `Greenhouse` component.

Take a look at the code in the `src/context/ThemeContext.js` **file.**

- Notice that a new context called, `ThemeContext`, has been created.
- Also notice that a Provider for that context has been created.
- The value passed to the Provider is an object with two keys,`themeName`, and `setThemeName`.
- A custom React hook called `useTheme` which returns the value of the `ThemeContext` has also been created.
- The `themeName` state will eventually be able to switch between two modes, `"day"` or `"night"`.

Take a look at the code in the `src/index.js`**.**

- The Provider created for the `ThemeContext` wraps the entire application inside of the `Root` component.
- This gives all nested components the ability to gain access to the `ThemeContext`'s value.

Take a look at the code in `src/components/Greenhouse/Greenhouse.js`. Your job is to use the value of `ThemeContext` in the `Greenhouse` component.

- There are two images imported into the `Greenhouse` component, `dayImage` and `nightImage`.
- You want to display a different image based on the `themeName` state that is passed in ThemeContext Provider's value object. If `themeName` is `"day"`, then `dayImage` should be displayed. If the `themeName` is `"night"`, `nightImage` should be displayed.
- To use the `ThemeContext` value in this component, use the `useTheme` hook from `src/context/ThemeContext.js` to bring in the `themeName` and `setThemeName` properties and use them in the component.

Time to test in the browser.
Inside `ThemeProvider` in the `ThemeContext.js` file, you can manually change the default state. Check to see that the proper image displays when `themeName` is set to `'day'` and when it is set to `'night'`. (Remember to reload the page after changing the default.)

Congratulations, you just connected a component to a context!

Phase 2: LightSwitch

Now, you'll work on changing the value of a context in a React component.

Take a look at the code in the `LightSwitch` component (`src/components/Greenhouse/LightSwitch.js`). This component should change the `themeName` between `"day"` and `"night"`. When the `div` with the class of "on" is clicked, the `themeName` should be set to `"day"`. When the `div` with the class of "off" is clicked, the `themeName` should be set to `"night"`.
The `themeName` value should also replace the hard coded `"day"` class on the `div` with the class of "light-switch" plus the appropriate theme name based on state. Ex: `<div className={`light-switch ${themeName}`}>`.
Hint:

- Use the `useTheme` hook like you used in the previous phase to retrieve the `setThemeName` property from the context Provider value.
- Then use the `setThemeName` function to correctly change the name of the theme based on clicking the `<div>` with the `onClick` event listener.

Test in the browser.

If the image of the greenhouse changes when you click on the "DAY" or "NIGHT" text, then you successfully changed the value of the context in a React component!

Phase 3: ClimateContext

In this phase, you'll learn how to create context, a context Provider, and a React custom wrapper hook that will return the value of the newly created context.

- In the `src/context/ClimateContext.js` file, create a context called `ClimateContext`, a provider called `ClimateProvider` for that context, and a custom React hook that returns the value of `ClimateContext`.
- In the provider, create a component state variable that defines the temperature of the greenhouse.
- The temperature should be set to `50` (Number) by default.
- The value of the context Provider should be set to an object with a key to read the temperature and a key to set the temperature.

Refer to the `src/context/ThemeContext.js` file if you need a reminder on how to do any of the above.
By the way, did you remember to wrap the application with `ClimateContext` in order to make it available to nested components?

Phase 4: Thermometer

In this phase, you will connect the `Thermometer` component (`src/components/Thermometer/Thermometer.js`) to the `ClimateContext`. The thermometer should show and change the `thermometer` value in the `ClimateContext`. There is a React component called `ReactSlider` being mounted which renders a slider in the browser. When the slider value changes, the temperature value in the context should also change.

In this component, the thermometer value should be displayed on the "Actual Thermometer".

- Fill in the two props, `value` and `onAfterChange`, which are passed into `ReactSlider`.
- `value` should be set to the temperature value in the `ClimateContext`.
- In the `onAfterChange` function `val` should be set as the new temperature value in the context.

Test this out on the browser!

Phase 5: Hygrometer

In this phase, you will add more properties on the `ClimateContext` value, and use those new properties in the `Hygrometer` component (`src/components/Hygrometer/Hygrometer.js`).

- Create a new component state variable in the `ClimateContext` Provider that reads and sets the humidity of the greenhouse. Set the default humidity to `40` (Number).
- The `Hygrometer` component should show and change the humidity value in the `ClimateContext`.
- Similar to the `Thermometer` component, fill in the two props passed into `ReactSlider`--i.e., `value` and `onAfterChange`--in order to read and change the humidity value of the `ClimateContext`.

Test this out on the browser!

Phase 6: Climate Stats

- Connect the `ClimateStats` component (`src/components/Greenhouse/ClimateStats.js`) to the `ClimateContext`.
- Display the temperature and humidity values here.

Congratulations! You've connected multiple components to context, created your own context, created a provider for context with a dynamic value object, and a custom React hook to return the value object from your context.

Bonus Phase 1: Thermostat

In this bonus phase, whenever a user changes the desired temperature on the thermostat, slowly change the value of the actual temperature on a timer at a rate of 1 degree per second.

Hint: Create a component state variable that tracks the desired temperature displayed on the thermostat. Also create a `useEffect` with that state variable as a dependency and a `setTimeout` function that changes the actual temperature by 1 degree. The actual temperature should stop changing on the display when the actual temperature equals the desired temperature. Make sure to include a cleanup function in the `useEffect` that clears the timeout.

Bonus Phase 2: Hygrometer

Like the previous phase, whenever a user changes the desired humidity on the hygrometer, change the value of the actual humidity on a timer at a rate of 2 percent per second until it reaches the actual value.

Did you find this lesson helpful?
No
Yes

✓
Archive your file into a **.zip** folder and click **Submit Project** to upload.

Solutions become available after uploading.