

EdX and its Members use cookies and other tracking technologies for performance, analytics, and marketing purposes. By using this website, you accept this use. Learn more about these technologies in the [Privacy Policy](#). ✕



[Course](#) > [Week 3...](#) > [Home...](#) > Home...

## Homework 5 - React

In this assignment, you will develop a simple application to track lists of items using React.

The application you will be building has many components that work together in order to organize multiple lists. Such an app could theoretically be used to create a shopping list, a set of daily tasks, or so forth.

In completing this assignment, you will:

- Gain experience understanding and modifying an existing React app
- Use the Node framework to create and deploy a React app with multiple components defined in separate files
- Gain more experience in implementing callback functions in React components that affect the components' appearance
- Apply what you have learned in the lessons about the relationships between React components and how components interact

### Debugging/Error Note:

If you run into errors/bugs/don't understand the output that Codio is giving you, please post in the Discussion Forum and a TA will assist you! Please do NOT email Codio as they will not review any errors you are getting.

### Background

For this assignment, we will provide you with a React app that initially looks like this:

### Important List Tracker

What will be on your next list?

Create List

Add new lists to get started!

And you will modify it so that the user can create new lists, and then add items to those lists, so that it can for instance look like this:

## Important List Tracker

What will be on your next list?

Create List

### Dogs List

- Snoopy
- Lola
- Sprinkles

Add Dogs

Name

Submit

### Cats List

- Felidae
- Garfield
- Cat in the Hat

Add Cats

Name

Submit

### Getting Started

To complete this assignment, you will need to set up a development environment that uses Node.js and the create-react-app package using the steps below; you will likely need sudo/root access for each of these steps:

1. Install Node.js locally by downloading it from <https://nodejs.org/en/download/>
2. From the Terminal, Command Prompt, etc. update Node Package Manager by typing the command: **npm install npm -g**
3. Install the create-react-app package by typing the command: **npm install -g create-react-app**
4. Create a new folder or directory for your project, then navigate to it using Terminal, Command Prompt, etc.
5. Create the project by typing the command: **create-react-app lists**
6. This will create a "lists" directory and install the default React app there. You can start it by changing to the "lists" directory and typing the command: **npm start**



You should then be able to access **http://localhost:3000** using your web browser and see the default React app. If so, you're ready to proceed.

Start by downloading **React-lists.zip**. This contains the files you need in order to start building this app.

To install the app:

1. Extract the files from the .zip file that you downloaded
2. Copy **index.html** into the "public/" directory of your React project; this will overwrite the default index.html that is already there.
3. Copy the **seven .js files** and **App.css** into the "src/" directory of your React project; this will overwrite the default index.js, App.js and App.css that are already there

#### Having trouble setting up your environment?

You can do this assignment using the Codio grading platform, which we have configured with Node and React.

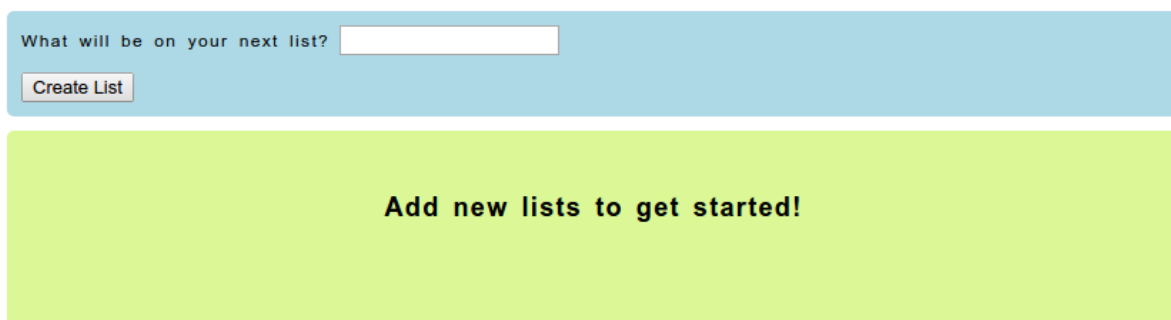
Click the "BEGIN SUBMISSION" button at the bottom of the screen, which will take you to Codio.

Once you're there, use the "Filetree" on the left side of the screen to expand the "dev" folder, then click "Instructions.md"

This will open up the file that includes instructions for completing this assignment on Codio.

If your React app is still running, you should see it refresh in the browser and you should now see a screen that looks like this:

## Important List Tracker



What will be on your next list?

Create List

**Add new lists to get started!**

If this does not happen, restart your React app as described above and then open **http://localhost:3000** in your browser.

#### Activity

The goal of this assignment is to modify the code we have provided so that the user can:

1. Create a new list
2. Add items to that list
3. Mark items by changing their color

When you are finished, the app should look something like this, depending on the user inputs:



# Important List Tracker

What will be on your next list?

Create List

Dogs List

- Snoopy
- Lola
- Sprinkles

Add Dogs

Name

Submit

Cats List

- Felidae
- Garfiled
- Cat in the Hat

Add Cats

Name

Submit

To accomplish this, you will use six React components that are organized like this:

**App**

## Important List Tracker

**AddList**

What will be on your next list?

**Lists**

**Dogs List**

- Snoopy
- Lola
- Sprinkles

**ListItems**

**Add Dogs**

Name

**Cats List**

**List**

- Felidae
- Garfiled
- Cat in the Hat

**AddItem**

**Add Cats**

Name

The **App** component is the main component of the app and its render function creates two additional components:

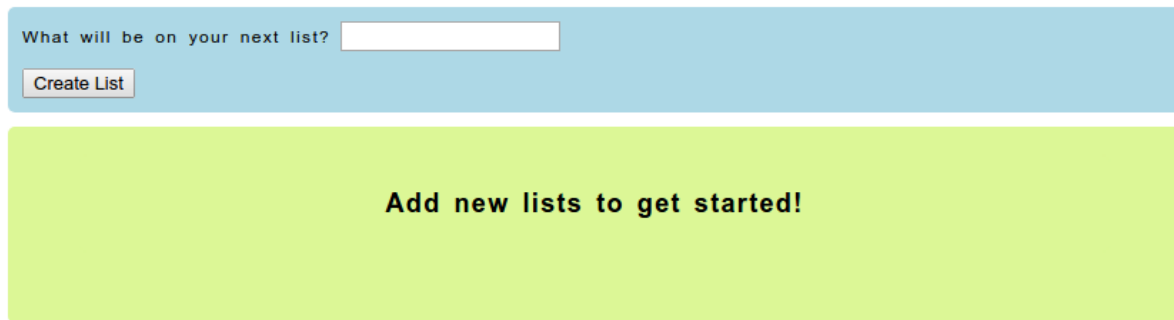
- **AddList**, which is responsible for taking user input and passing information back to the **App** component in order to create a new list
- **Lists**, which is empty when the **App** is first rendered, but contains all instances of individual **List** components, each of which contains an **AddItem** component for adding items and **ListItem** components for all items in the list

Each component is implemented in a separate file, which has the same name as the component.

Before proceeding, be sure you understand the role of each component in the app and how the components are related.

### Part 1. Creating a new List

## Important List Tracker



What will be on your next list?

Create List

**Add new lists to get started!**

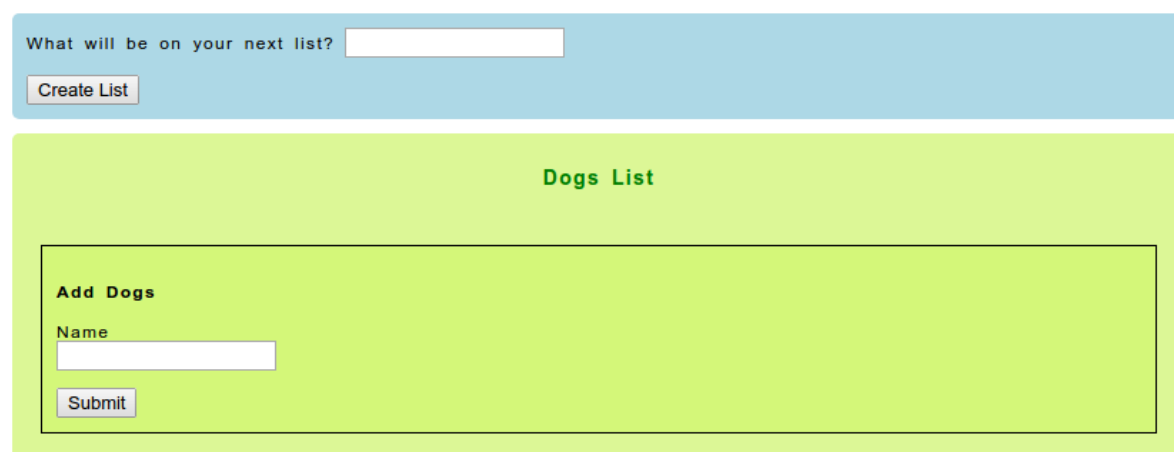
In the image above, we see the **App** component render two child components: **AddList** and **Lists**. The **AddList** component contains the text input field next to the words “What will be on your next list?” and a button labeled “Create List”. The **Lists** component by default reads “Add new lists to get started!”, but will eventually display each individual **List** component, which consists of an **AddItem** component and **ListItem** components.

Implement the **AddList.handleSubmit** and **App.handleAddList** functions and modify the components as necessary so that it is possible for the user to create a new list and see the **List** and **AddItem** components rendered in the page. In particular, the app should behave as follows:

- When the user types the name of a new list into the input field of the **AddList** component and clicks the “Create List” button, the **AddList** component should call the **AddList.handleSubmit** function, which should then update its state and then use the **App.handleAddList** function to update the state of the **App** (specifically, the “lists” and “items” variables in the state, as described in the code comments and according to the example below).
- The **App** should then re-render, and with the **App**’s state now containing a new list, the **App** should pass the necessary information from its state to the **Lists** component, which will render a new **List** component representing a single list.

The image below shows what the app should now look like when the user adds a “Dogs” list.

## Important List Tracker



What will be on your next list?

Create List

**Dogs List**

**Add Dogs**

Name

Submit

At this point, the **App**’s state should look like this:

```
{
```



```
items: {Dogs: []},  
lists: ["Dogs"]  
}
```

## Part 2. Adding ListItems to a List

Now implement the **AddItem.handleSubmit** and **App.handleAddItem** functions and modify the components as necessary so that it is possible for the user to use the AddItem component to add an item to a list and see the ListItem component rendered in the page. In particular, the app should behave as follows:

- When the user enters a value in the input field of the AddItem component and clicks the "Submit" button, the AddItem component should call the **AddItem.handleSubmit** function, which should update the component's state and then use the **App.handleAddItem** function to update the state of the App (specifically, the "items" variable in the state, as described in the code comments and according to the example below).
- The App should then re-render. It should pass the necessary information from its state to the Lists component, which will then pass it to its List components, which can then render each ListItem with the appropriate values.

Below is an image depicting the results of adding two lists – one for Dogs and one for Cats – and adding three items to each.

## Important List Tracker

What will be on your next list?

Create List

### Dogs List

- Snoopy
- Lola
- Sprinkles

Add Dogs

Name

Submit

### Cats List

- Felidae
- Garfiled
- Cat in the Hat

Add Cats

Name

Submit

At this point, the App's state should look like this:

```
{
  items:
    {Dogs: [{name: "Snoopy"}, {name: "Lola"}, {name: "Sprinkles"}],
     Cats: [{name: "Felidae"}, {name: "Garfield"}, {name: "Cat in the Hat"}] },
  lists: ["Dogs", "Cats"]
}
```

### Part 3. Marking a ListItem

Certain lists, such as To-Dos or grocery lists, would be greatly improved by the ability to “mark” an item as complete or purchased.

Implement the **ListItem.handleClick** function so that it is invoked when the user clicks on the content in the ListItem, i.e., within the `<span>` element. The function should alternate the color of the text in the ListItem between black and gray.

The image below depicts what would happen if you added Milk, Bread, and Eggs to a Grocery list and clicked on the Milk and Bread ListItems:



#### One more important note:

Although you may add props to the components, please do not change the names of the ones that we have provided, as these prop names need to be consistent for grading. In particular, please do not change the “name” prop for the List, and be sure that each ListItem has an “item” prop that is an object that has a “name” property.

Also, please do not change the “id”, “type”, or “ref” attribute of any HTML/JSEX element created in the components’ “render” functions, as these will be used for grading.

#### Helpful Hints

Review the past few lessons on developing React apps, particularly focusing on the examples in which one component renders another, and where one component is able to pass a function as a prop to another component.

In Parts 1 and 2, we’ve used the “ref” attribute of some of the HTML elements in order to allow React to access these elements and their values. For instance in AddList.js we have:

```
<input type='text' ref='id' id='newID'>
```

Your React code can then access this HTML element’s value using: `this.refs.id.value`

Also in Parts 1 and 2, you may want to update the state of a component and then invoke a function after doing so. However, if you do this:

```
this.setState( { name: value } );
someFunction(this.state);
```

You may see that `someFunction` is invoked *before* the state has been updated. Fortunately, you can pass a callback function to `setState` that will be invoked *after* the state is updated. like this:





```
this.setState( { name: value }, function() {  
  someFunction(this.state);  
});
```

Keep in mind that although you are using Node to deploy your React app, and even though your browser is accessing the app via a web server, all of the JavaScript is running in your browser. So if you use `console.log` to debug your app, the messages will appear in the browser's JavaScript console, and not in the Terminal/Command Prompt window where you ran "npm start".

Last, in numerous places in the code we've provided, you'll see props that are specified like this:

```
key={uuidv4() }
```

This is a way of specifying a "universally unique identifier," which is used by React internally. You should not need to modify any of this code.

### **Before you submit**

Please be sure that:

- You have not changed the names of any of the props that we have specified in the code we distributed
- You have not changed the "id", "type", or "ref" attribute of any HTML/JSX element created in the components' "render" functions
- You have not created any additional files and all of your React code is in the .js files that we have provided

### **Assessment**

Your submission will be assessed using automatic grading scripts that will check that the app works correctly for various inputs and events. Your score is determined by the percentage of these tests that "pass," i.e. that produce the correct behavior for the specified input or event.

To submit your assignment, click the "BEGIN SUBMISSION" button below and follow the instructions in the Codio Readme file. You need to upload the six React component .js files for grading (**AddItem.js**, **AddList.js**, **App.js**, **List.js**, **ListItem.js**, and **Lists.js**), but you do not need to submit index.js or any of the files from the "public/" folder.

---

## Homework #5 Submission (External resource) (100.0 / 100.0 points)

This link will take you to Codio so that you may submit this assignment.

**BEGIN SUBMISSION** 

Learn About Verified Certificates

© All Rights Reserved

