CONTENTS

// Tutorial //

How To Build a CRUD App with React Hooks and the Context API

Updated on March 23, 2021

React

By Ishan Manandhar

Introduction

This article will be covering the Context API (introduced in version 16.3) and React hooks (introduced in version 16.8).

The introduction of the Context API solves one major problem: prop drilling. The process of getting our data from one component to another through layers of nested deep components. React hooks allows the use of functional rather than class-based components. Where we needed to utilize a lifecycle method, we had to use a class-based approach. And we now no longer have to call `super(props)` or worry about binding methods or the `this` keyword.

In this article, you will use Context API and React hooks together to build a fully functional CRUD application that emulates a list of employees. It will read employee data, create new employees, update employee data, and delete employees. Note, that this tutorial will not be using any external API calls. For the sake of demonstration, it will be using hard-coded objects which will serve as the state.

Prerequisites

To complete this tutorial, you'll need:

- A local development environment for Node.js. Follow How to Install Node.js and Create a Local Development Environment.
- An understanding of importing, exporting, and rendering React components. You can take a look at our How To Code in React.js series.

This tutorial was verified with Node v15.3.0, `npm` v7.4.0, `react` v17.0.1, `react-router-dom` v5.2.0, `tailwindcss-cli` v0.1.2, and `tailwindcss` v2.0.2.

Step 1 — Setting Up the Project

First, start with setting up the React project using Create React App with the following command:

```
$ npx create-react-app  react-crud-employees-example
```

Copy

Navigate to the newly created project directory:

```
$ cd  react-crud-employees-example
```

Copy

Next, add `react-router-dom` as a dependency by running the following command:

```
$ npm install react-router-dom @5.2.0
```

Copy

> **Note:** For additional information on React Router, consult our React Router tutorial.

Then, navigate to the `src` directory:

```
cd src
```

Add the default build of Tailwind CSS to your project with the following command:

```
$ npx tailwindcss-cli @0.1.2  build --output tailwind.css
```

Copy

> **Note:** For additional information on Tailwind CSS, consult our Tailwind CSS tutorial.

Next, open `index.js` in your code editor and modify it to use `tailwind.css` and `BrowserRouter`:

src/index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
```

Copy

```
 import { BrowserRouter } from 'react-router-dom';
 import './tailwind.css';
import './index.css';
import App from './App';

ReactDOM.render(
   <BrowserRouter>
     <App />
   <BrowserRouter>
  document.getElementById('root')
);
```

At this point, you will have a new React project with Tailwind CSS and `react-router-dom`.

Step 2 – Building the and

First, under the `src` directory, create a new `context` directory.

In this new directory, create a new `AppReducer.js` file. This reducer will define CRUD actions like `ADD_EMPLOYEE`, `EDIT_EMPLOYEE`, and `REMOVE_EMPLOYEE`. Open this file in your code editor and add the following lines of code:

src/context/AppReducer.js

Copy

```
export default function appReducer(state, action) {
  switch (action.type) {
    case "ADD_EMPLOYEE":
      return {
        ...state,
        employees: [...state.employees, action.payload],
      };

    case "EDIT_EMPLOYEE":
      const updatedEmployee = action.payload;

      const updatedEmployees = state.employees.map((employee) => {
        if (employee.id === updatedEmployee.id) {
          return updatedEmployee;
        }
        return employee;
      });

      return {
        ...state,
        employees: updatedEmployees,
      };

    case "REMOVE_EMPLOYEE":
      return {
        ...state,
        employees: state.employees.filter(
          (employee) => employee.id !== action.payload
        ),
      };

    default:
      return state;
  }
};
```

`ADD_EMPLOYEES` will take a payload value containing new employees and return the updated employee state.

`EDIT_EMPLOYEE` will take a payload value and compare the `id` with the employees - if it finds a match, it will use the new payload values and return the updated employee state.

`REMOVE_EMPLOYEE` will take a payload value and compare the `id` with the employees - if it finds a match, it will remove that employee and return the updated employee state.

While remaining in the `context` directory, create a new `GlobalState.js` file. It will contain an initial hard-coded value to emulate employee data returned from a request. Open this file in your code editor and add the following lines of code:

src/context/GlobalState.js

```
import React, { createContext, useReducer } from 'react';

import appReducer from './AppReducer';

const initialState = {
  employees: [
    {
      id: 1,
      name: "Sammy",
      location: "DigitalOcean",
      designation: "Shark"
    }
  ]
};
```

Copy

```
export const GlobalContext = createContext(initialState);

export const GlobalProvider = ({ children }) => {
  const [state, dispatch] = useReducer(appReducer, initialState);

  function addEmployee(employee) {
    dispatch({
      type: "ADD_EMPLOYEE",
      payload: employee
    });
  }

  function editEmployee(employee) {
    dispatch({
      type: "EDIT_EMPLOYEE",
      payload: employee
    });
  }

  function removeEmployee(id) {
    dispatch({
      type: "REMOVE_EMPLOYEE",
      payload: id
    });
  }

  return (
    <GlobalContext.Provider
      value={{
        employees: state.employees,
        addEmployee,
        editEmployee,
        removeEmployee
      }}
    >
      {children}
    </GlobalContext.Provider>
  );
};
```

This code adds some functionality to dispatch an action that goes into the reducer file to switch upon the case that corresponds to each action.

At this point, you should have a React application with `AppReducer.js` and `GlobalState.js`.

Let’s create an `EmployeeList` component to verify that the application is in working order. Navigate to the `src` directory and create a new `components` directory. In that directory, create a new `EmployeeList.js` file and add the following code:

src/components/EmployeeList.js

```
import React, { useContext } from 'react';

import { GlobalContext } from '../context/GlobalState';

export const EmployeeList = () => {
  const { employees } = useContext(GlobalContext);
  return (
    <React.Fragment>
      {employees.length > 0 ? (
        <React.Fragment>
          {employees.map((employee) => (
            <div
              className="flex items-center bg-gray-100 mb-10 shadow"
              key={employee.id}
            >
              <div className="flex-auto text-left px-4 py-2 m-2">
                <p className="text-gray-900 leading-none">
                  {employee.name}
                </p>
                <p className="text-gray-600">
                  {employee.designation}
                </p>
                <span className="inline-block text-sm font-semibold mt-1">
                  {employee.location}
                </span>
              </div>
            </div>
          ))}
        </React.Fragment>
      ) : (
        <p className="text-center bg-gray-100 text-gray-500 py-5">No data.</p>
      )}
    </React.Fragment>
  );
};
```

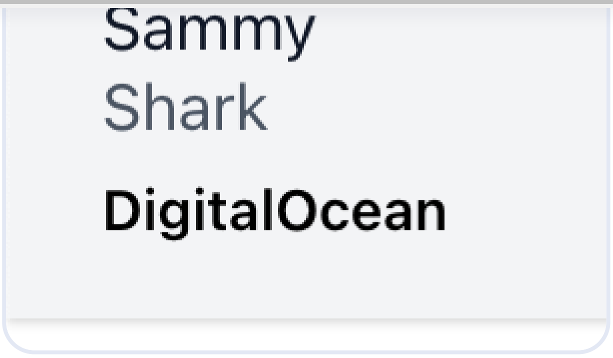
This code will display the `employee.name`, `employee.designation`, and `employee.location` for all `employees`.

Next, open `App.js` in your code editor. And add `EmployeeList` and `GlobalProvider`.

src/App.js

```
 import { EmployeeList } from './components/EmployeeList';

 import { GlobalProvider } from './context/GlobalState';

function App() {
  return (
     <GlobalProvider>
      <div className="App">
         <EmployeeList />
      </div>
     </GlobalProvider>
  );
}
```

Copy



The `EmployeeList` component will display the hard-coded values that were established in `GlobalState.js`.

Step 3 – Building the and Components

In this step, you will build the components to supporting creating a new employee and updating an existing employee.

Now, navigate back to the `components` directory. Create a new `AddEmployee.js` file. This will serve as the `AddEmployee` component which will include an `onSubmit` handler to push the values of the form field into the state:

src/components/AddEmployee.js

Copy

```
import React, { useState, useContext } from 'react';
import { Link, useHistory } from 'react-router-dom';

import { GlobalContext } from '../context/GlobalState';

export const AddEmployee = () => {
  let history = useHistory();

  const { addEmployee, employees } = useContext(GlobalContext);

  const [name, setName] = useState("");
  const [location, setLocation] = useState("");
  const [designation, setDesignation] = useState("");

  const onSubmit = (e) => {
    e.preventDefault();
    const newEmployee = {
      id: employees.length + 1,
      name,
      location,
      designation,
    };
    addEmployee(newEmployee);
    history.push("/");
  };

  return (
    <React.Fragment>
      <div className="w-full max-w-sm container mt-20 mx-auto">
        <form onSubmit={onSubmit}>
          <div className="w-full mb-5">
            <label
              className="block uppercase tracking-wide text-gray-700 text-xs font-bo
              htmlFor="name"
            >
              Name of employee
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3 text
              value={name}
              onChange={(e) => setName(e.target.value)}
              type="text"
              placeholder="Enter name"
            />
          </div>
          <div className="w-full mb-5">
            <label
              className="block uppercase tracking-wide text-gray-700 text-xs font-bo
              htmlFor="location"
            >
              Location
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3 text
              value={location}
              onChange={(e) => setLocation(e.target.value)}
              type="text"
              placeholder="Enter location"
            />
          </div>
          <div className="w-full mb-5">
            <label
              className="block uppercase tracking-wide text-gray-700 text-xs font-bo
              htmlFor="designation"
            >
              Designation
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3 text
              value={designation}
              onChange={(e) => setDesignation(e.target.value)}
              type="text"
              placeholder="Enter designation"
            />
          </div>
          <div className="flex items-center justify-between">
            <button className="mt-5 bg-green-400 w-full hover:bg-green-500 text-whit
              Add Employee
            </button>
          </div>
          <div className="text-center mt-4 text-gray-500">
            <Link to="/">Cancel</Link>
          </div>
        </form>
      </div>
    </React.Fragment>
    );
  };
```

In this code `setName`, `setLocation`, and `setDesignation` will take the current values that users enter into the form fields. These values will be wrapped in a new constant, `newEmployee`, with a unique `id` (adding one to the total length). Then, the route will be changed to the main screen which will display the updated list of employees - including the newly added employee.

The `AddEmployee` component imported `GlobalState` and , one of the built-in React Hooks, giving functional components easy access to our context.

The `employees` object, `removeEmployee`, and `editEmployees` were imported from the `GlobalState.js` file.

While still in the `components` directory, create a new `EditEmployee.js` file. This will serve as the `editEmployee` component which will include functionality for editing the existing objects from the state:

src/components/EditEmployee.js

```
import React, { useState, useContext, useEffect } from 'react';
import { useHistory, Link } from 'react-router-dom';

import { GlobalContext } from '../context/GlobalState';

export const EditEmployee = (route) => {
  let history = useHistory();

  const { employees, editEmployee } = useContext(GlobalContext);

  const [selectedUser, setSelectedUser] = useState({
    id: null,
    name: "",
    designation: "",
    location: "",
  });

  const currentUserId = route.match.params.id;

  useEffect(() => {
    const employeeId = currentUserId;
    const selectedUser = employees.find(
      (currentEmployeeTraversal) => currentEmployeeTraversal.id === parseInt(employe
```

Copy

```
    );
    setSelectedUser(selectedUser);
  }, [currentUserId, employees]);

  const onSubmit = (e) => {
    e.preventDefault();
    editEmployee(selectedUser);
    history.push("/");
  };

  const handleOnChange = (userKey, newValue) =>
    setSelectedUser({ ...selectedUser, [userKey]: newValue });

  if (!selectedUser || !selectedUser.id) {
    return <div>Invalid Employee ID.</div>;
  }

  return (
    <React.Fragment>
      <div className="w-full max-w-sm container mt-20 mx-auto">
        <form onSubmit={onSubmit}>
          <div className="w-full mb-5">
            <label
              className="block uppercase tracking-wide text-gray-700 text-xs font-bo
              htmlFor="name"
            >
              Name of employee
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3 text
              value={selectedUser.name}
              onChange={(e) => handleOnChange("name", e.target.value)}
              type="text"
              placeholder="Enter name"
            />
          </div>
          <div className="w-full  mb-5">
            <label
              className="block uppercase tracking-wide text-gray-700 text-xs font-bo
              htmlFor="location"
            >
              Location
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3 text
              value={selectedUser.location}
              onChange={(e) => handleOnChange("location", e.target.value)}
              type="text"
              placeholder="Enter location"
            />
          </div>
          <div className="w-full  mb-5">
            <label
              className="block uppercase tracking-wide text-gray-700 text-xs font-bo
              htmlFor="designation"
            >
              Designation
            </label>
            <input
              className="shadow appearance-none border rounded w-full py-2 px-3 text
              value={selectedUser.designation}
              onChange={(e) => handleOnChange("designation", e.target.value)}
              type="text"
              placeholder="Enter designation"
            />
          </div>
          <div className="flex items-center justify-between">
            <button className="block mt-5 bg-green-400 w-full hover:bg-green-500 tex
              Edit Employee
            </button>
          </div>
          <div className="text-center mt-4 text-gray-500">
            <Link to="/">Cancel</Link>
          </div>
        </form>
      </div>
    </React.Fragment>
  );
};
```

This code uses the hook, which is invoked when the component is mounted. Inside this hook, the current route parameter will be compared with the same parameter in the `employees` object from the state.

`onChange` event listeners are triggered when the user makes a change to the form fields. The `userKey` and the `newValue` are passed to `setSelectedUser`. `selectedUser` is spread and `userKey` is set as the key and `newValue` is set as the value.

Step 4 — Setting Up Routes

In this step, you will update the `EmployeeList` to link to the `AddEmployee` and `EditEmployee` components.

Revisit `EmployeeList.js` and modify it to use `Link` and `removeEmployee`:

src/components/EmployeeList.js

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

import { GlobalContext } from '../context/GlobalState';

export const EmployeeList = () => {
  const { employees,  removeEmployee  } = useContext(GlobalContext);
  return (
    <React.Fragment>
      {employees.length > 0 ? (
        <React.Fragment>
          {employees.map((employee) => (
            <div
              className="flex items-center bg-gray-100 mb-10 shadow"
              key={employee.id}
            >
              <div className="flex-auto text-left px-4 py-2 m-2">
                <p className="text-gray-900 leading-none">
                  {employee.name}
                </p>
                <p className="text-gray-600">
                  {employee.designation}
                </p>
                <span className="inline-block text-sm font-semibold mt-1">
                  {employee.location}
                </span>
              </div>
               <div className="flex-auto text-right px-4 py-2 m-2">
                <Link
                  to={`/edit/${employee.id}` }
                  title="Edit Employee "
                >
                  <div className="bg-gray-300 hover:bg-gray-400 text-gray-800 font-
                    <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" 
                  < /div>
                </Link>
                <button
                  onClick={() => removeEmployee(employee.id) }
                  className="block bg-gray-300 hover:bg-gray-400 text-gray-800 font-
```

```
                    title="Remove Employee "
                  >
                    <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" vi
                  </button>
                </div>
              </div>
            ))}
          </React.Fragment>
        ) : (
          <p className="text-center bg-gray-100 text-gray-500 py-5">No data.</p>
        )}
      </React.Fragment>
    );
  };
```

This code will add two icons next to the employee information. The pencil and paper icon represents “Edit” and links to the `EditEmployee` component. The trashbin icon represents “Remove” and clicking on it will fire `removeEmployee`.

Next, you will create two new components - `Heading` and `Home` - to display the `EmployeeList` component and provide users with access to the `AddEmployee` component.

In the `components` directory, create a new `Heading.js` file:

src/components/Heading.js

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

export const Heading = () => {
  return (
    <div>
      <div className="flex items-center mt-24 mb-10">
        <div className="flex-grow text-left px-4 py-2 m-2">
          <h5 className="text-gray-900 font-bold text-xl">Employee Listing</h5>
        </div>
        <div className="flex-grow text-right px-4 py-2 m-2">
          <Link to="/add">
            <button className="bg-green-400 hover:bg-green-500 text-white font-semib
              <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox
              <span className="pl-2">Add Employee</span>
            </button>
          </Link>
        </div>
      </div>
    </div>
  );
};
```

In the `components` directory, create a new `Home.js` file:

src/components/Home.js

```
import React from "react";
import { Heading } from "./Heading";
import { EmployeeList } from "./EmployeeList";

export const Home = () => {
  return (
    <React.Fragment>
      <div className="container mx-auto">
        <h3 className="text-center text-3xl mt-20 text-base leading-8 text-black fon
          CRUD with React Context API and Hooks
        </h3>
        <Heading />
        <EmployeeList />
      </div>
    </React.Fragment>
  );
};
```

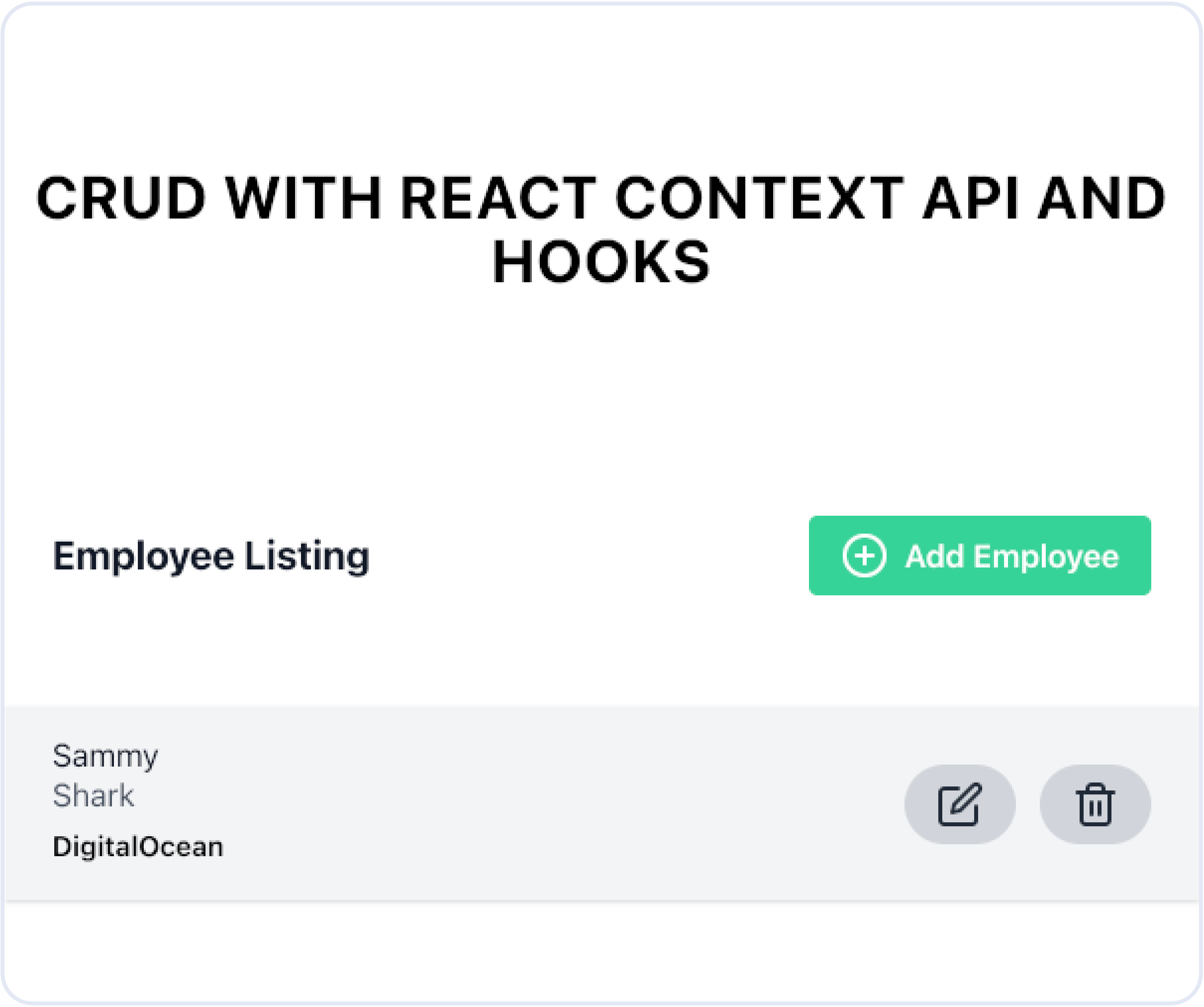
Revisit `App.js` and import `Route` and `Switch` from `react-router-dom`. Assign the `Home`, `AddeEmployee` and `EditEmployee` components to each route:

src/App.js

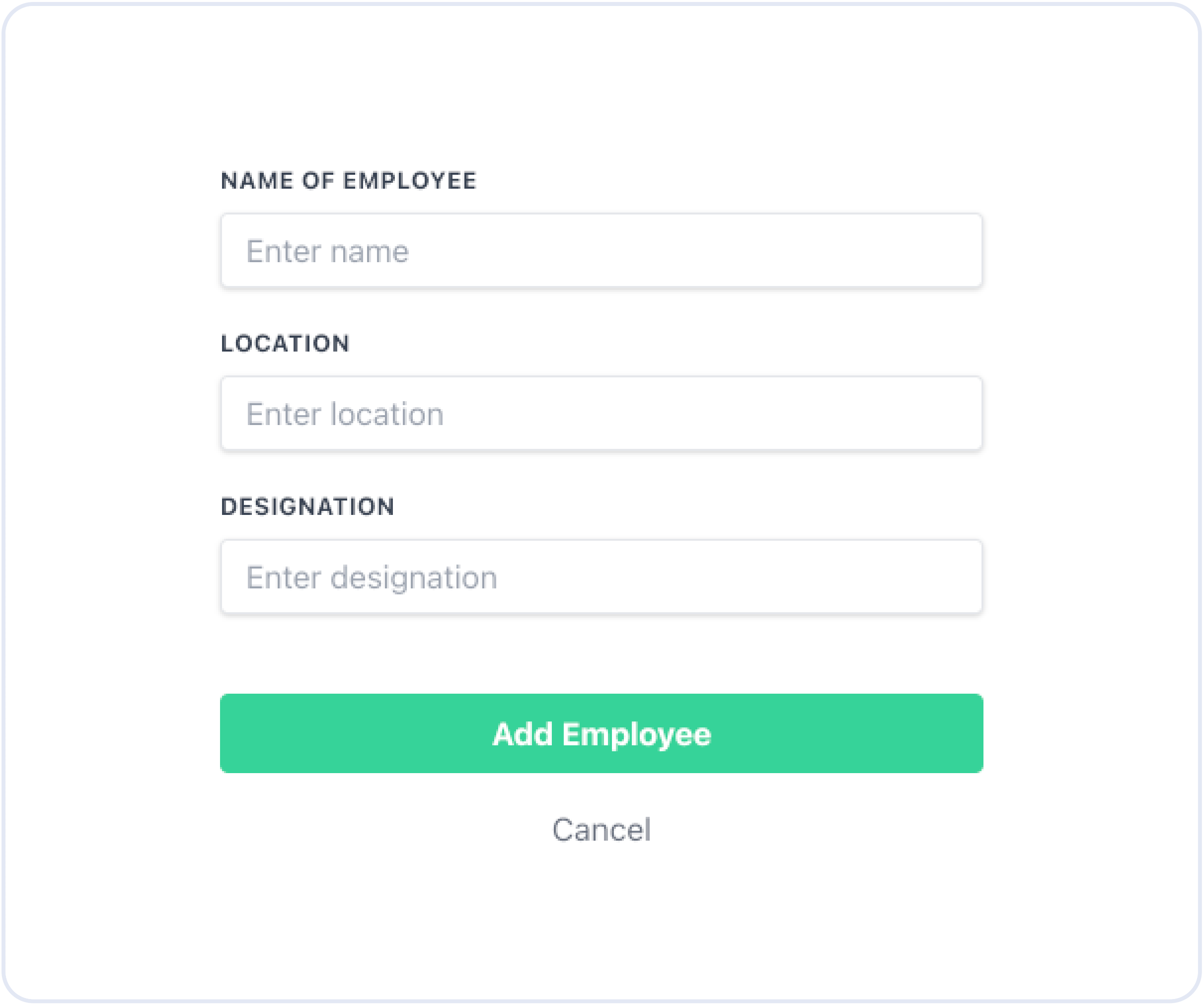
```
 import { Route, Switch } from 'react-router-dom';

import { GlobalProvider } from './context/GlobalState';

 import { Home } from './components/Home';
 import { AddEmployee } from './components/AddEmployee';
 import { EditEmployee } from './components/EditEmployee';

function App() {
  return (
      <GlobalProvider>
        <div className="App">
          <Switch>
            <Route path="/" component={Home} exact />
            <Route path="/add" component={AddEmployee} exact />
            <Route path="/edit/:id" component={EditEmployee} exact />
          </Switch>
        </div>
      </GlobalProvider>
    );
  }

export default App;
```

Compile the app and observe it in your browser.

You will be routed to the `Home` component with the `Heading` and `EmployeeList` components:

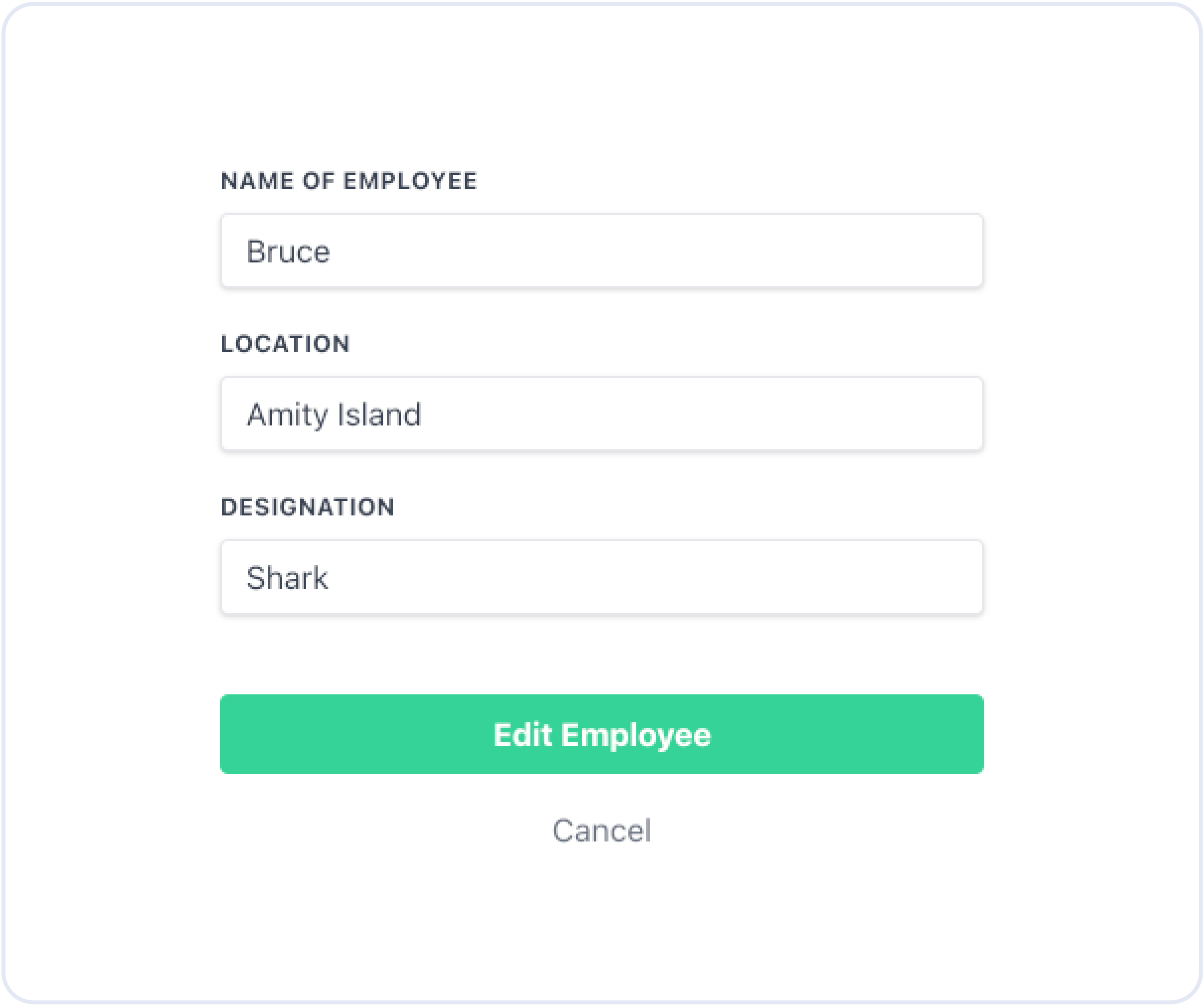


Click on the **Add Employee** link. You will be routed to the `AddEmployee` component:



After submitting information for a new employee, you will be routed back to the `Home` component and it will now list the new employee.

Click on the **Edit Employee** link. You will be routed to the `EditEmployee` component:


NAME OF EMPLOYEE

Bruce

LOCATION

Amity Island

DESIGNATION

Shark

After modifying the information for the employee, you will be routed back to the `Home` component and it will now list the new employee with the updated details.

Conclusion

In this article, you used Context API and React hooks together to build a fully functional CRUD application.

If you'd like to learn more about React, take a look at our How To Code in React.js series, or check out our React topic page for exercises and programming projects.

About the authors

Ishan Manandhar Author

Still looking for an answer? Ask a question

Search for more help

Was this helpful? Yes No

Comments

7 Comments

Leave a comment...

This textbox defaults to using `Markdown` to format your answer.

You can type `!ref` in this text area to quickly search our full set of tutorials, documentation & marketplace offerings and insert the link!

Sign In or Sign Up to Comment

suhailvs • May 16, 2022

there is a bug in `index.js`. here is the fixed

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import './tailwind.css';
import './index.css';
import App from './App';

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById('root')
);
```

Reply

tronganhnguyen222 • June 29, 2021

Hi, I am doing a get list and delete data in react-context that following your video and I got a problem is when I click on my removeUser(i,id) an error comes out with a message removeUser is not function although I have declare a value for that: const {user,removeUser} = useContext(GlobalContext);

Reply

viniciusfesil • March 11, 2021

For those who searched the `Home.js` file and did not find code:

```
import React, { Fragment } from 'react';
import { Heading } from './Heading';
import { Employeelist } from './Employeelist';
```

Copy

```
export const Home = () => {
    return (
        <Fragment>
            <div className="App">
                <div className="container mx-auto">
                    <h3 className="text-center  text-3xl mt-20 text-base le
                    <Heading />
                    <Employeelist />
                </div>
            </div>
        </Fragment>
    )
}
```

Reply

alimrandev • November 25, 2020

Hi! This is a nice tutorial you are shared with us. You have explained it very well. I will definitely try to do this app. Thanks for sharing

Reply

RishabhMalik • August 22, 2020

Hey there, I guess you missed Home.js …

Reply

mutanyi • June 24, 2020

npx create-react-app react-contextAPI

error: Cannot create a project named “react-contextAPI” because of npm naming restrictions:

Show replies Reply

jvlg2002 • June 11, 2020

This comment has been deleted

Try DigitalOcean for free

Click below to sign up and get **$200 of credit** to try our products over 60 days!

Sign up

Popular Topics

Ubuntu

Linux Basics

JavaScript

Python

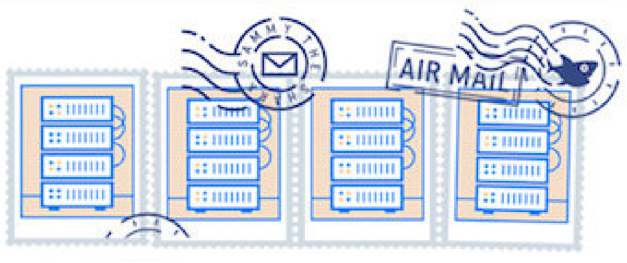
MySQL

Docker

Kubernetes

All tutorials →

Talk to an expert →



Get our biweekly newsletter

Sign up for Infrastructure as a Newsletter.

Sign up →

Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →

Become a contributor

You get paid; we donate to tech nonprofits.

Learn more →

Featured on Community

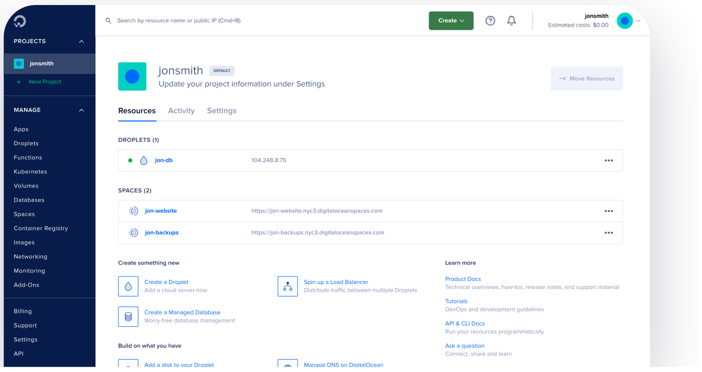
Kubernetes Course Learn Python 3 Machine Learning in Python

Getting started with Go Intro to Kubernetes

DigitalOcean Products

Cloudways Virtual Machines Managed Databases Managed Kubernetes

Block Storage Object Storage Marketplace VPC Load Balancers



Company

Products

Community

Solutions

Contact