

Practice Hands-on lab: Budget Allocation Application

Estimated Time: 60 minutes

In this React Budget Allocation app, you will learn how to break down a UI into React components. You will become familiar with state using the Context API. Furthermore, you will explore actions, reducers, and the dispatch function. You will create a code file, save it, and edit it to make changes.

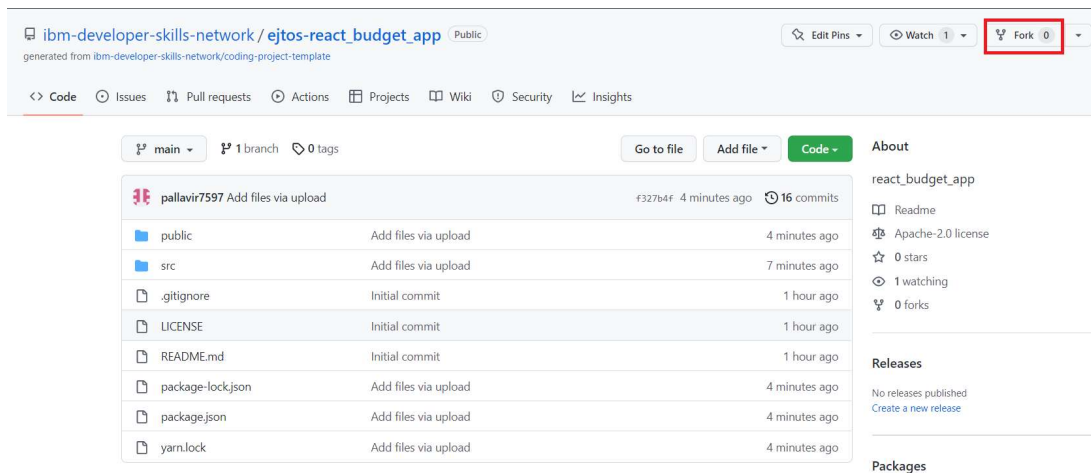
Objectives:

- Setup a React Project.
- Put the UI Components in Place.
- Render the created components and context in App.js.
- View your app on the browser.

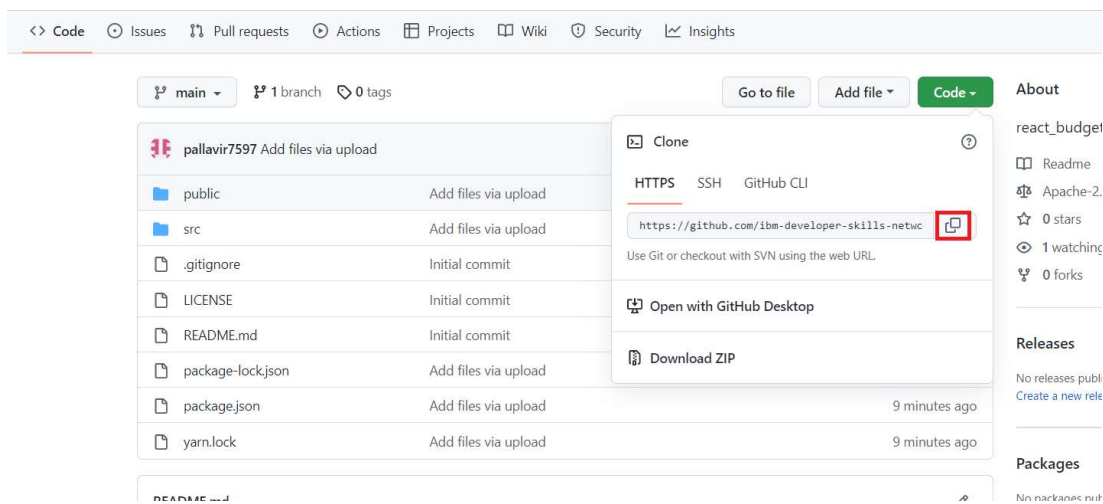
Exercise 1: Setup a React Project

Fork the Git repository to have the react code you need to start

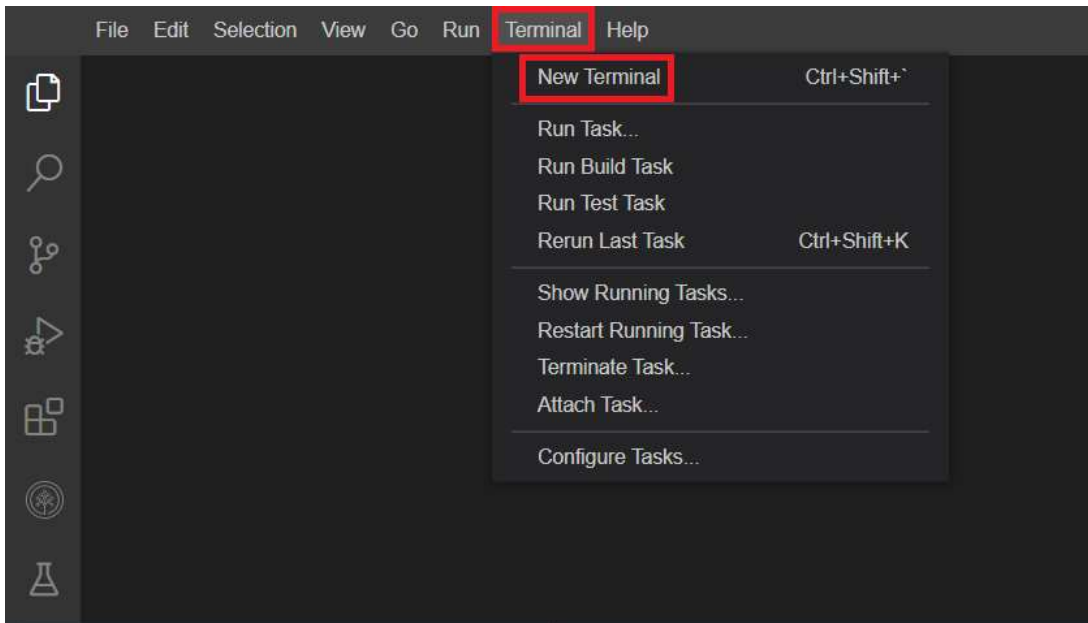
1. Go to the project repository on this [link](#) which has the partially developed code for react code.
2. Create a fork of the repository into your own. *You will need to have a github account of your own to do so.*



3. Go to your repository and copy the clone url.



4. Open a terminal window by using the menu in the editor: Terminal > New Terminal.



5. Clone the repository by running the command given below:

1. 1
1. `git clone <your_repo_name>`

Copied!

```
Problems x theia@theiadocker-pallavir /home/project x
theia@theiadocker-pallavir:/home/project$ git clone https://github.com/ibm-developer-skills-network/ejtos-react_budget_app.git
Cloning into 'ejtos-react_budget_app'...
remote: Enumerating objects: 73, done.
remote: Counting objects: 100% (73/73), done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 73 (delta 19), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (73/73), done.
theia@theiadocker-pallavir:/home/project$
```

6. This will clone the repository with budget allocation app files in your home directory in the lab environment. Check the same with the following command.

1. 1
1. `ls`

Copied! Executed!

7. Change to the project directory and check its contents.

1. 1
1. `cd ejtos-react_budget_app && ls`

Copied! Executed!

8. All packages required to be installed are listed in package.json. Run `npm install -s`, to install and save those packages.

1. 1
1. `npm install -s`

Copied! Executed!

Exercise 2: Put the UI Components in Place.

Company's Budget Allocation

Budget: £2000

Remaining: £1040

Spent so far: £960

Allocation

Department	Allocated Budget	Increase by 10	Delete
Marketing	£50	<div>+</div>	<div>✕</div>
Finance	£300	<div>+</div>	<div>✕</div>
Sales	£70	<div>+</div>	<div>✕</div>
Human Resource	£40	<div>+</div>	<div>✕</div>
IT	£500	<div>+</div>	<div>✕</div>

Change allocation

Department

Choose...

Allocation

Add

Save

The image above displays the Company’s Budget Allocation application page you will be developing in this practice project.

It has the following six components:

- Budget
- ExpenseItem
- ExpenseList
- ExpenseTotal (Spent so far)
- Remaining
- AllocationForm

All of these components will be using redux for state management through AppContext.js. You can verify the content of AppContext.js by clicking on the below button.

- In AppContext.js you will be creating reducer, which is used to update the state, based on the action. Then you will set the initial state for the departments. You will be creating the Provider component which wraps the components you want to give access to the state.
- Here, you are adding an initial budget, creating a Provider component, setting up the useReducer hook which will hold your state, and allow you to update the state via dispatch.
- Adding a new case to the switch statement called “ADD_EXPENSE”, “RED_EXPENSE”, “DELETE_EXPENSE”.

Open AppContext.js in IDE

Details of the six components:

- In Budget.js you will be adding text and value for your budget. You will be importing app context and the useContext hook, and pass your AppContext to it - this is how a component connects to the context in order to get values from global state.
- In Remaining.js you will be importing expense and budget from context and getting the remaining value using subtraction.
- In ExpenseTotal.js you will be adding useContext and AppContext. Taking the expenses from state.
- In ExpenseList.js you will be using the map function to display the Expenseitem component.
- In Expenseitem.js you will be importing dispatch from Context, which allows you to dispatch a delete action, creating a function that gets called when the delete icon is clicked.
- In AllocationForm.js you will be creating an expense object, containing the name and the cost. This is what will get dispatched as the payload, and what you’ll use to update the state.

At start you will be creating a Budget, Remaining and Spent so far button.

1. Make changes to **Budget.js** component. Open the code, in the src/components/Budget.js. Copy the below code and paste in the Budget.js

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11

1. import React, { useContext } from 'react';
2. import { AppContext } from '../context/AppContext';
3. const Budget = () => {
```

```

4.     const { budget } = useContext(AppContext);
5.     return (
6.         <div className='alert alert-secondary'>
7.             <span>Budget: ₹{budget}</span>
8.         </div>
9.     );
10.  };
11. export default Budget;

```

Copied!

Here, you are using the Bootstrap Alert classes to give a nice gray background by adding some text and hard coding a value.

2. Make changes to App.js. Open the code, in the src/App.js. Copy the below code and paste in the App.js in the space provided. You will notice that Budget.js has been already imported for you in App.js.

```

1. 1
2. 2
3. 3
4. 4

1.           // Budget component
2.           <div className='col-sm'>
3.               <Budget />
4.           </div>

```

Copied!

3. Make sure you are in the ejtos-react_budget_app directory and run the server using the following command.

```

1. 1

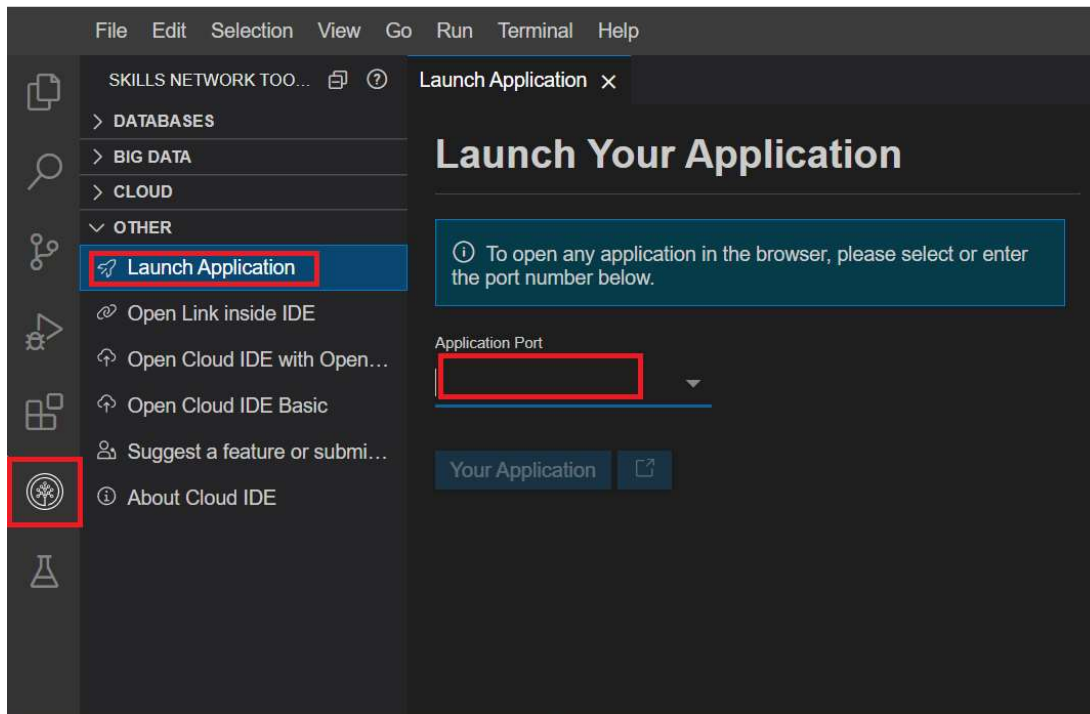
1. npm start

```

Copied! Executed!

4. Click on the button below to launch the application or click on Skills Network button on the left, it will open the “Skills Network Toolbox”. Then click **Other** then **Launch Application**. From there you should be able to enter the port 3000.

Budget Application



5. Verify your output on the browser.

Company's Budget Allocation

Budget: £2000

You can view the Budget button created in the above image.

6. Make changes to **Remaining.js** component. Open the code, in the `src/components/Remaining.js`. Copy the below code and paste in the `Remaining.js`

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15

1. import React, { useContext } from 'react';
2. import { AppContext } from '../context/AppContext';
3. const Remaining = () => {
4.   const { expenses, budget } = useContext(AppContext);
5.   const totalExpenses = expenses.reduce((total, item) => {
6.     return (total = total + item.cost);
7.   }, 0);
8.   const alertType = totalExpenses > budget ? 'alert-danger' : 'alert-success';
9.   return (
10.     <div className={`alert ${alertType}`}>
11.       <span>Remaining: £{budget - totalExpenses}</span>
12.     </div>
13.   );
14. };
15. export default Remaining;

```

Copied!

Here, you are using the `reduce` function to get a total of all the costs, assigning this to a variable and displaying the variable in your JSX. Now whenever the user adds an expense, this causes the state to update, which will cause all components connected to the context to re-render and update themselves with new values.

7. Make changes to `App.js`. Open the code, in the `src/App.js`. Copy the below code and paste in the `App.js` in the space provided. Import the component in the appropriate place using `import Remaining from './components/Remaining'`;

```

1. 1
2. 2
3. 3
4. 4

1.           //Remaining component
2.           <div className='col-sm'>
3.             <Remaining />
4.           </div>

```

Copied!

8. Refresh the React app page launched in step 4 and verify the output.

Company's Budget Allocation

Budget: £2000

Remaining: £1000

9. Make changes to **ExpenseTotal.js** component. Open the code, in the `src/components/ExpenseTotal.js`. Copy the below code and paste in the `ExpenseTotal.js`

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14

1. import React, { useContext } from 'react';
2. import { AppContext } from '../context/AppContext';
3. const ExpenseTotal = () => {
4.   const { expenses } = useContext(AppContext);
5.   const totalExpenses = expenses.reduce((total, item) => {
6.     return (total += item.cost);
7.   }, 0);
8.   return (
9.     <div className='alert alert-primary'>
10.      <span>Spent so far: £{totalExpenses}</span>
11.    </div>
12.  );
13. };
14. export default ExpenseTotal;

```

Copied!

Here, you are using the `reduce` function to get a total of all the costs, assigning this to a variable and displaying the variable in your JSX. Now whenever the user adds an expense, this causes the state to update, which will cause all components connected to the context to re-render and update themselves with new values.

10. Make changes to `App.js`. Open the code, in the `src/App.js`. Copy the below code and paste in the `App.js` in the space provided. Import the component in the appropriate place using `import ExpenseTotal from './components/ExpenseTotal';`

```

1. 1
2. 2
3. 3
4. 4

1.           //ExpenseTotal component
2.           <div className='col-sm'>
3.             <ExpenseTotal />
4.           </div>

```

Copied!

11. Refresh the React app page launched in step 4 and verify the output.

Company's Budget Allocation

Budget: £2000

Remaining: £1000

Spent so far: £1000

12. Make changes to **ExpenseList.js** component. Open the code, in the `src/components/ExpenseList.js`. Copy the below code and paste in the `ExpenseList.js`.

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27

1. import React, { useContext } from 'react';
2. import ExpenseItem from './ExpenseItem';
3. import { AppContext } from '../context/AppContext';
4.
5. const ExpenseList = () => {
6.   const { expenses } = useContext(AppContext);
7.
8.   return (
9.     <table className='table'>
10.       <thead className="thead-light">
11.         <tr>
12.           <th scope="col">Department</th>
13.           <th scope="col">Allocated Budget</th>
14.           <th scope="col">Increase by 10</th>
15.           <th scope="col">Delete</th>
16.         </tr>
17.       </thead>
18.       <tbody>
19.         {expenses.map((expense) => (
20.           <ExpenseItem id={expense.id} key={expense.id} name={expense.name} cost={expense.cost} />
21.         ))}
22.       </tbody>
23.     </table>
24.   );
25. };
26.
27. export default ExpenseList;

```

Copied!

Here, you are creating a list, using the `map` function to iterate over the expenses, and displaying an `ExpenseItem` component.

13. Make changes to **ExpenseItem.js** component. Open the code, in the `src/components/ExpenseItem.js`. Copy the below code and paste in the `ExpenseItem.js`.

```

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38

```

```

1. import React, { useContext } from 'react';
2. import { TiDelete } from 'react-icons/ti';
3. import { AppContext } from '../context/AppContext';
4.
5. const ExpenseItem = (props) => {
6.   const { dispatch } = useContext(AppContext);
7.
8.   const handleDeleteExpense = () => {
9.     dispatch({
10.       type: 'DELETE_EXPENSE',
11.       payload: props.id,
12.     });
13.   };
14.
15.   const increaseAllocation = (name) => {
16.     const expense = {
17.       name: name,
18.       cost: 10,
19.     };
20.
21.     dispatch({
22.       type: 'ADD_EXPENSE',
23.       payload: expense
24.     });
25.   }
26. }
27.
28. return (
29.   <tr>
30.     <td>{props.name}</td>
31.     <td>£{props.cost}</td>
32.     <td><button onClick={event=> increaseAllocation(props.name)}>+</button></td>
33.     <td><TiDelete size='1.5em' onClick={handleDeleteExpense}></TiDelete></td>
34.   </tr>
35. );
36. };
37.
38. export default ExpenseItem;

```

Copied!

Here, you are dispatching an action. Your action contains the type (so the reducer knows how to update the state) and the payload. In this case you are passing the ID of this expense (which you get from props when you rendered the ExpenseList).

14. Make changes to **AllocationForm.js** component. Open the code, in the `src/components/AllocationForm.js`. Copy the below code and paste in the `AllocationForm.js`.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
46. 46
47. 47
48. 48
49. 49
50. 50
51. 51
52. 52
53. 53
54. 54
55. 55
56. 56
57. 57
58. 58
59. 59
60. 60
61. 61
62. 62
63. 63
64. 64
65. 65
66. 66
67. 67
68. 68
69. 69
70. 70
71. 71
72. 72
73. 73
74. 74
75. 75
76. 76
77. 77
78. 78
79. 79
80. 80
81. 81
```

```

1. import React, { useContext, useState } from 'react';
2. import { AppContext } from '../context/AppContext';
3.
4. const AllocationForm = (props) => {
5.   const { dispatch, remaining } = useContext(AppContext);
6.
7.   const [name, setName] = useState('');
8.   const [cost, setCost] = useState('');
9.   const [action, setAction] = useState('');
10.
11.   const submitEvent = () => {
12.
13.     if(cost > remaining) {
14.       alert("The value cannot exceed remaining funds £"+remaining);
15.       setCost("");
16.       return;
17.     }
18.
19.     const expense = {
20.       name: name,
21.       cost: parseInt(cost),
22.     };
23.     if(action === "Reduce") {
24.       dispatch({
25.         type: 'RED_EXPENSE',
26.         payload: expense,
27.       });
28.     } else {
29.       dispatch({
30.         type: 'ADD_EXPENSE',
31.         payload: expense,
32.       });
33.     }
34.   };
35.
36.   return (
37.     <div>
38.       <div className='row'>
39.
40.         <div className="input-group mb-3" style={{ marginLeft: '2rem' }}>
41.           <div className="input-group-prepend">
42.             <label className="input-group-text" htmlFor="inputGroupSelect01">Department</label>
43.           </div>
44.           <select className="custom-select" id="inputGroupSelect01" onChange={(event) => setName(event.target.value)}>
45.             <option defaultValue>Choose...</option>
46.             <option value="Marketing" name="marketing"> Marketing</option>
47.             <option value="Sales" name="sales">Sales</option>
48.             <option value="Finance" name="finance">Finance</option>
49.             <option value="HR" name="hr">HR</option>
50.             <option value="IT" name="it">IT</option>
51.             <option value="Admin" name="admin">Admin</option>
52.           </select>
53.
54.           <div className="input-group-prepend" style={{ marginLeft: '2rem' }}>
55.             <label className="input-group-text" htmlFor="inputGroupSelect02">Allocation</label>
56.           </div>
57.           <select className="custom-select" id="inputGroupSelect02" onChange={(event) => setAction(event.target.value)}>
58.             <option defaultValue value="Add" name="Add">Add</option>
59.             <option value="Reduce" name="Reduce">Reduce</option>
60.           </select>
61.
62.           <input
63.             required='required'
64.             type='number'
65.             id='cost'
66.             value={cost}
67.             style={{ marginLeft: '2rem' , size: 10}}
68.             onChange={(event) => setCost(event.target.value)}>
69.         </input>
70.
71.         <button className="btn btn-primary" onClick={submitEvent} style={{ marginLeft: '2rem' }}>
72.           Save
73.         </button>
74.       </div>
75.     </div>
76.
77.   </div>
78. );
79. };
80.
81. export default AllocationForm;

```

Copied!

Here, you are adding form tags, adding a label/input for name, cost and action field, and adding values for various departments.

Exercise 3: Render the created components and context in App.js

1. Make changes to **App.js**. Open the code, in the `src/App.js`. You need to add the code in the container div to import and add the components created above.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35

1. import React from 'react';
2.
3. import 'bootstrap/dist/css/bootstrap.min.css';
4.
5. //Code to import Budget.js
6. import Budget from './components/Budget';
7.
8. // Add code to import the other components here under
9.
10.
11. import { AppProvider } from './context/AppContext';
12. const App = () => {
13.   return (
14.     <AppProvider>
15.       <div className='container'>
16.         <h1 className='mt-3'>Company's Budget Allocation</h1>
17.         <div className='row mt-3'>
18.           /* Add Budget component here under */
19.
20.           /* Add Remaining component here under */
21.
22.           /* Add ExpenseTotal component here under */
23.
24.           /* Add ExpenseList component here under */
25.
26.           /* Add ExpenseItem component here under */
27.
28.           /* Add AllocationForm component here under */
29.
30.         </div>
31.       </div>
32.     </AppProvider>
33.   );
34. };
35. export default App;
```

Copied!

► [Click here to view the hint](#)

► Click here to view the solution

Here, you are importing different components,adding a bootstrap container that helps you center your App on the page

- Adding a title
- Adding a Bootstrap row
- Adding a column within the row for each of your components so far
- Imported and Rendered the AllocationForm
- Imported AppProvider and Nested components in the AppProvider element.

2. Refresh the React app page launched and verify the output.

Company's Budget Allocation

Budget: £2000

Remaining: £1040

Spent so far: £960

Allocation

Department	Allocated Budget	Increase by 10	Delete
Marketing	£50	<div>+</div>	<div>✕</div>
Finance	£300	<div>+</div>	<div>✕</div>
Sales	£70	<div>+</div>	<div>✕</div>
Human Resource	£40	<div>+</div>	<div>✕</div>
IT	£500	<div>+</div>	<div>✕</div>

Change allocation

Department

Choose... ▾

Allocation

Add ▾

Save

Commit and push your local code to your remote git repository

This git pushed code will be cloned and used in final project.

Please refer to this lab [Working with git in the Theia lab environment](#)

Congratulations! You have completed this practice lab and know how to create components of a budget allocation application.

Author(s)

Pallavi Rai

Change Log

Date	Version	Changed by	Change Description
2022-08-31	1.0	Pallavi Rai	Initial version created

© IBM Corporation 2022. All rights reserved.