

Optional: Sample Project: The Shopping Application



Estimated Time: 90 minutes

Note: Though this lab is split into multiple parts, it is highly recommended it is all completed in one sitting as the project environment may not be persisted. If for any reason that is not possible, please push your changes to git so that the changes are copied to your remote repository and you can start where you left.

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.*

The screenshot shows the GitHub repository page for 'ibm-developer-skills-network / kduia-shopping-app'. The repository is public and was generated from 'ibm-developer-skills-network/coding-project-template'. The 'Fork' button is highlighted with a red box. The repository has 1 branch (main) and 0 tags. The file list shows the following files and their commit history:

| File | Commit Message | Time Ago |
|-------------------|------------------------|-------------|
| public | Added the shopping app | 2 hours ago |
| src | Delete .DS_Store | 1 hour ago |
| .gitignore | Initial commit | 2 hours ago |
| LICENSE | Initial commit | 2 hours ago |
| README.md | Added the shopping app | 2 hours ago |
| package-lock.json | Added the shopping app | 2 hours ago |
| package.json | Added the shopping app | 2 hours ago |

The right sidebar shows the repository details: 'shopping-app', 'Readme', 'Apache-2.0 license', '0 stars', '1 watching', '0 forks', 'Releases' (No releases published), and 'Packages'.

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

The screenshot shows the GitHub interface for a repository named 'kduia-shopping-app'. The top navigation bar includes links for Code, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. Below the navigation bar, the repository name 'kduia-shopping-app' is displayed, along with the current branch 'main' and a notification that the branch is up to date. A table of files and folders is shown, including 'public', 'src', '.gitignore', 'LICENSE', 'README.md', 'package-lock.json', and 'package.json'. The 'Code' button is highlighted, and its dropdown menu is open, showing options for cloning the repository via HTTPS, SSH, or GitHub CLI. The HTTPS URL is displayed, and the copy icon is highlighted with a red box. Other options in the menu include 'Open with GitHub Desktop' and 'Download ZIP'. The right sidebar contains information about the repository, including the README, license, stars, watching, and forking status.

Code Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

This branch is up to date with ibm-developer-skills-network/kduia-shopping-app

lavskillup Delete .DS_Store

| | |
|-------------------|------------------------------------|
| public | Added the shopping app |
| src | Delete .DS_Store |
| .gitignore | Initial commit |
| LICENSE | Initial commit |
| README.md | Added the shopping app |
| package-lock.json | Added the shopping app 3 hours ago |
| package.json | Added the shopping app 3 hours ago |

Local Codespaces New

Clone ?

HTTPS SSH GitHub CLI

[https://github.com/\[redacted\]/kduia-shopping-app](https://github.com/[redacted]/kduia-shopping-app)

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Download ZIP

About shopping-app

Readme

Apache-2.0 license

0 stars

0 watching

1 fork

Releases

No releases published

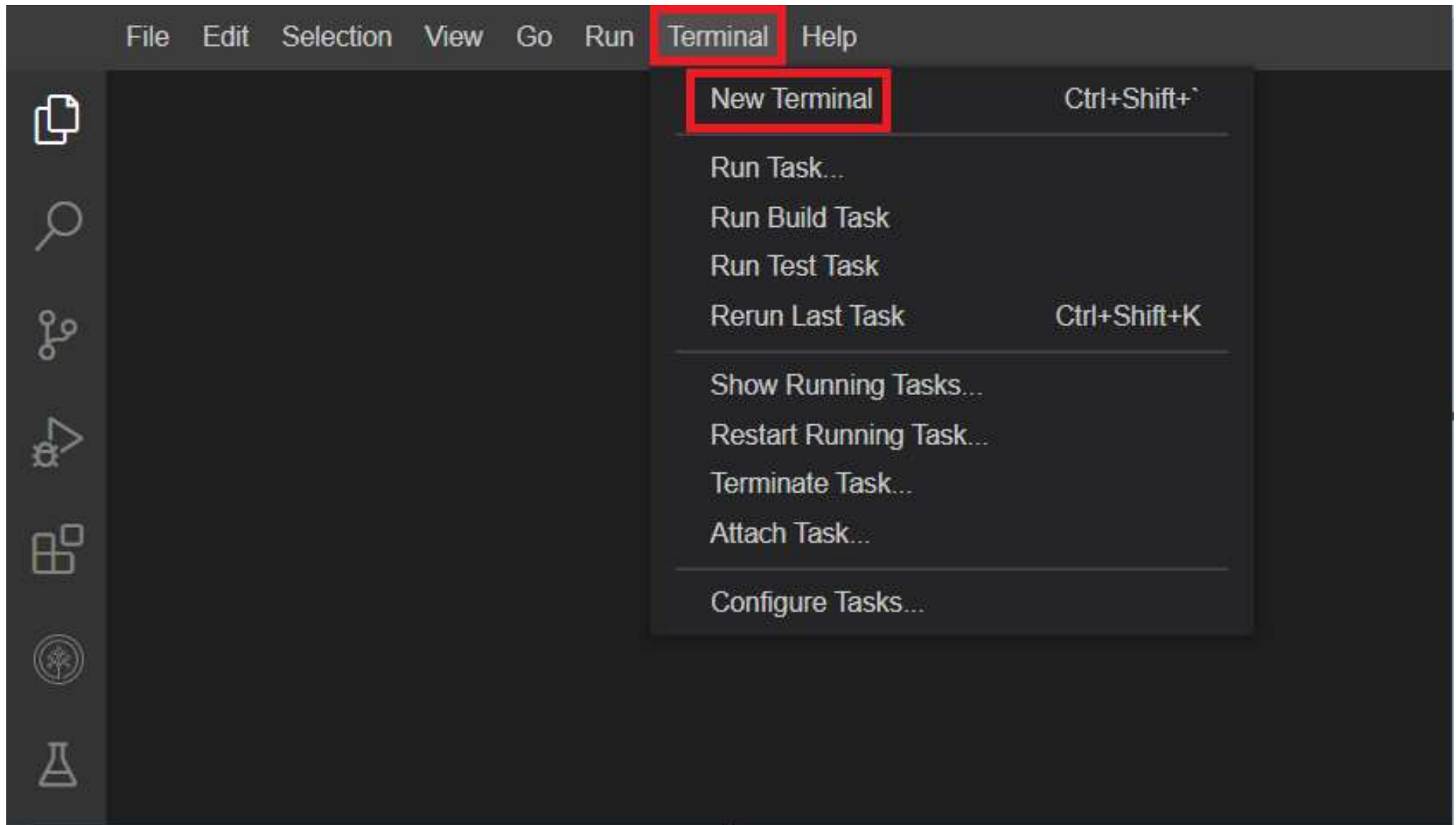
Create a new release

Packages

No packages published

Publish your first package

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!

```
Output theia@theiadocker-pallavir: /home/project x

theia@theiadocker-[-[REDACTED]:/home/project$ git clone https://github.com/[REDACTED]/kduia-shopping-app.git
Cloning into 'kduia-shopping-app'...
remote: Enumerating objects: 51, done.
remote: Counting objects: 100% (51/51), done.
remote: Compressing objects: 100% (40/40), done.
remote: Total 51 (delta 11), reused 45 (delta 8), pack-reused 0
Unpacking objects: 100% (51/51), done.
```

6. This will clone the repository with Shopping application 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 kduia-shopping-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.

Shopping App

Cart Value: ₹0

Location

India(₹)

Shopping Cart

| Items | Quantity | Unit Price | Items Price | Remove |
|------------|----------|------------|-------------|--------------|
| Shirt | 0 | ₹500 | ₹0 | <div>✕</div> |
| Jeans | 0 | ₹300 | ₹0 | <div>✕</div> |
| Dress | 0 | ₹400 | ₹0 | <div>✕</div> |
| Dinner set | 0 | ₹600 | ₹0 | <div>✕</div> |
| Bags | 0 | ₹200 | ₹0 | <div>✕</div> |

Add Items

Items

Jeans

Quantity

Add

200

Save

The image above displays the Shopping application page you will be developing in this sample project. This lab will help you in gaining the knowledge about react to build a shopping app and complete the tasks provided in final project: Budget Allocation application.

It has the following five components:

- CartValue
- ExpenseItem
- ExpenseList
- ItemSelected
- Location

All of these components will be using redux for state management through `AppContext.js`. You can open the `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 Shopping Cart. You will be creating the Provider component which wraps the components you want to give access to the state.

[Open AppContext.js in IDE](#)

1. Make changes to **AppContext.js** component. Open the code, in the `src/context/AppContext.js`. Copy the below code and paste in the `AppContext.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
82. 82
83. 83
84. 84
85. 85
86. 86
87. 87
88. 88
89. 89
90. 90
91. 91
92. 92
93. 93
94. 94
95. 95
96. 96
97. 97
98. 98
99. 99
100. 100

1. import React, { createContext, useReducer } from 'react';
2.
3. // 5. The reducer - this is used to update the state, based on the action
4. export const AppReducer = (state, action) => {
```

```

5.   let new_expenses = [];
6.   switch (action.type) {
7.     case 'ADD_QUANTITY':
8.       let updatedqty = false;
9.       state.expenses.map((expense)=>{
10.        if(expense.name === action.payload.name) {
11.          expense.quantity = expense.quantity + action.payload.quantity;
12.          updatedqty = true;
13.        }
14.        new_expenses.push(expense);
15.        return true;
16.      })
17.      state.expenses = new_expenses;
18.      action.type = "DONE";
19.      return {
20.        ...state,
21.      };
22.
23.     case 'RED_QUANTITY':
24.       state.expenses.map((expense)=>{
25.        if(expense.name === action.payload.name) {
26.          expense.quantity = expense.quantity - action.payload.quantity;
27.        }
28.        expense.quantity = expense.quantity < 0 ? 0: expense.quantity;
29.        new_expenses.push(expense);
30.        return true;
31.      })
32.      state.expenses = new_expenses;
33.      action.type = "DONE";
34.      return {
35.        ...state,
36.      };
37.     case 'DELETE_ITEM':
38.       state.expenses.map((expense)=>{
39.        if(expense.name === action.payload.name) {
40.          expense.quantity = 0;
41.        }
42.        new_expenses.push(expense);
43.        return true;
44.      })
45.      state.expenses = new_expenses;
46.      action.type = "DONE";
47.      return {
48.        ...state,
49.      };
50.     case 'CHG_LOCATION':
51.       action.type = "DONE";
52.       state.Location = action.payload;
53.       return {
54.        ...state
55.      }
56.
57.     default:
58.       return state;
59.   }
60. };
61.
62. // 1. Sets the initial state when the app loads
63. const initialState = {
64.   expenses: [
65.     { id: "Shirt", name: 'Shirt', quantity: 0, unitprice: 500 },
66.     { id: "Jeans", name: 'Jeans', quantity: 0, unitprice: 300 },
67.     { id: "Dress", name: 'Dress', quantity: 0, unitprice: 400 },
68.     { id: "Dinner set", name: 'Dinner set', quantity: 0, unitprice: 600 },
69.     { id: "Bags", name: 'Bags', quantity: 0, unitprice: 200 },
70.   ],
71.   Location: 'E'
72. };
73.
74. // 2. Creates the context this is the thing our components import and use to get the state
75. export const AppContext = createContext();
76.
77. // 3. Provider component - wraps the components we want to give access to the state
78. // Accepts the children, which are the nested(wrapped) components
79. export const AppProvider = (props) => {
80.   // 4. Sets up the app state. takes a reducer, and an initial state
81.   const [state, dispatch] = useReducer(AppReducer, initialState);
82.
83.   const totalExpenses = state.expenses.reduce((total, item) => {
84.     return (total = total + (item.unitprice*item.quantity));
85.   }, 0);
86.   state.CartValue = totalExpenses;
87.
88.   return (
89.     <AppContext.Provider
90.       value={{
91.         expenses: state.expenses,
92.         CartValue: state.CartValue,
93.         dispatch,
94.         Location: state.Location
95.       }}
96.     >
97.     {props.children}
98.   </AppContext.Provider>
99. );

```

```
100. };
Copied!
```

- In `AppContext.js` you are setting the initial state of `Expenses` and `Location`. You can see how the items, their respective unit price are all added to `Expenses`. Here, you are adding an initial expenses, 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_QUANTITY`”, “`RED_QUANTITY`”, “`DELETE_ITEM`”, “`CHG_LOCATION`”.

2. Make changes to **CartValue.js** component. You can open the `CartValue.js` by clicking on the below button.

```
Open CartValue.js in IDE
```

Copy the below code and paste in the `CartValue.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

1. import React, { useContext } from 'react';
2. import { AppContext } from '../context/AppContext';
3.
4. const CartValue = () => {
5.   const { expenses, Location } = useContext(AppContext);
6.   const totalExpenses = expenses.reduce((total, item) => {
7.     return (total += (item.unitprice * item.quantity));
8.   }, 0);
9.
10.   return (
11.     <div className='alert alert-primary'>
12.       <span>Cart Value: {Location}{totalExpenses}</span>
13.     </div>
14.   );
15. };
16.
17. export default CartValue;
Copied!
```

- Here, you are importing `AppContext` from your `Context`. Import 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. The `Bootstrap Alert` classes are used to give a nice gray background by adding some text and hard coding a value.
- Now if you change the budget in `AppContext` and reload your browser, you will see the budget updates on the UI.

3. Make changes to **ExpenseList.js** component. You can open the `ExpenseList.js` by clicking on the below button.

```
Open ExpenseList.js in IDE
```

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
28. 28
```



```

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">Items</th>
13.          <th scope="col">Quantity</th>
14.          <th scope="col">Unit Price</th>
15.          <th scope="col">Items Price</th>
16.          <th scope="col">Remove</th>
17.        </tr>
18.      </thead>
19.      <tbody>
20.        {expenses.map((expense) => (
21.          <ExpenseItem id={expense.id} key={expense.id} name={expense.name} quantity={expense.quantity} unitprice={expense.unitprice} />
22.        ))}
23.      </tbody>
24.    </table>
25.  );
26. };
27.
28. export default ExpenseList;

```

Copied!

- In ExpenseList you are importing your AppContext and useContext hook like before. Here, you are creating a list, using the map function to iterate over the expenses, and displaying an ExpenseItem component.

4. Make changes to **ExpenseItem.js** component. You can open the ExpenseItem.js by clicking on the below button.

Open **ExpenseItem.js** in IDE

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

```

```

1. import React, { useContext } from 'react';
2. import { AppContext } from '../context/AppContext';
3. import { FaTimesCircle } from 'react-icons/fa';
4.
5. const ExpenseItem = (props) => {
6.   const { dispatch, Location } = useContext(AppContext);
7.
8.   const handleDeleteItem = () => {
9.     const item = {
10.       name: props.name,
11.     };
12.
13.     dispatch({
14.       type: 'DELETE_ITEM',
15.       payload: item,
16.     });
17.   };
18.
19.   return (
20.     <tr>
21.       <td>{props.name}</td>
22.       <td>{props.quantity}</td>

```

```
24.         <td>{Location}{parseInt(props.unitprice)}</td>
25.         <td>{Location}{parseInt(props.quantity)*parseInt(props.unitprice)}</td>
26.         <td><FaTimesCircle size='2.2em' color="red" onClick={handleDeleteItem}></FaTimesCircle></td>
27.       </tr>
28.     );
29.   };
30.
31. export default ExpenseItem;
```

Copied!

- In ExpenseItem you are importing dispatch from Context, which allows you to dispatch a delete action. You are creating a function that gets called when the delete icon is clicked.

5. Make changes to **ItemSelected.js** component. You can open the ItemSelected.js by clicking on the below button.

Open **ItemSelected.js** in IDE

Copy the below code and paste in the ItemSelected.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

1. import React, { useContext, useState } from 'react';
2. import { AppContext } from '../context/AppContext';
3.
4. const ItemSelected = (props) => {
5.   const { dispatch } = useContext(AppContext);
6.
7.   const [name, setName] = useState('');
8.   const [quantity, setQuantity] = useState('');
9.   const [action, setAction] = useState('');
10.
11.
12.   const submitEvent = () => {
13.
14.     const item = {
15.       name: name,
16.       quantity: parseInt(quantity),
17.     };
18.
19.     if(action === "Reduce") {
20.       dispatch({
21.         type: 'RED_QUANTITY',
22.         payload: item,
23.       });
24.     } else {
25.       dispatch({
26.         type: 'ADD_QUANTITY',
27.         payload: item,
28.       });
29.     }
30.   };
31.
32.   return (
33.     <div>
34.       <div className='row'>
35.
36.         <div className="input-group mb-3" style={{ marginLeft: '2rem' }}>
37.           <div className="input-group-prepend">
38.             <label className="input-group-text" htmlFor="inputGroupSelect01">Items</label>
39.           </div>
40.           <select className="custom-select" id="inputGroupSelect01" onChange={(event) => setName(event.target.value)}>
41.             <option defaultValue>Choose...</option>
42.             <option value="Shirt" name="Shirt"> Shirt</option>
43.             <option value="Dress" name="Dress">Dress</option>
44.             <option value="Jeans" name="Jeans">Jeans</option>
45.             <option value="Dinner set" name="Dinner set">Dinner set</option>
46.             <option value="Bags" name="Bags">Bags</option>
47.           </select>
48.
49.           <div className="input-group-prepend" style={{ marginLeft: '2rem' }}>
50.             <label className="input-group-text" htmlFor="inputGroupSelect02">Quantity</label>
51.           </div>
52.           <select className="custom-select" id="inputGroupSelect02" onChange={(event) => setAction(event.target.value)}>
53.             <option defaultValue value="Add" name="Add">Add</option>
54.             <option value="Reduce" name="Reduce">Reduce</option>
55.           </select>
56.           <span className="eco" style={{ marginLeft: '2rem', marginRight: '8px' }}></span>
57.
58.           <input
59.             required='required'
60.             type='number'
61.             id='cost'
62.             value={quantity}
63.             style={{size: 10}}
64.             onChange={(event) => setQuantity(event.target.value)}>
65.           </input>
66.
67.           <button className="btn btn-primary" onClick={submitEvent} style={{ marginLeft: '2rem' }}>
68.             Save
69.           </button>
70.         </div>
71.       </div>
72.     </div>
73.   );
74. };
75. };
76.
77. export default ItemSelected;
```

Copied!

- In ItemSelected, you are importing AppContext and useContext as usual and getting dispatch from your global state. You are creating an event to reduce or add quantity. You are dispatching an action, with a type and your payload. The type tells the reducer how to update the state.

6. Make changes to **Location.js** component. You can open the Location.js by clicking on the below button.

Open **Location.js** in IDE

Copy the below code and paste in the Location.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

1. import React, { useContext } from 'react';
2. import { AppContext } from '../context/AppContext';
3.
4. const Location = () => {
5.   const {dispatch} = useContext(AppContext);
6.
7.   const changeLocation = (val)=>{
8.     dispatch({
9.       type: 'CHG_LOCATION',
10.      payload: val,
11.    })
12.  }
13.
14.  return (
15.    <div className='alert alert-secondary'> Location {
16.      <select name="location" id="location" onChange={event=>changeLocation(event.target.value)}>
17.        <option value="£">Uk(£)</option>
18.        <option value="₹">India(₹)</option>
19.        <option value="€">Europe(€)</option>
20.        <option value="CAD">Canada(CAD)</option>
21.      </select>
22.    }
23.  </div>
24.  );
25. };
26. };
27.
28. export default Location;
```

Copied!

- In Location.js, you are importing AppContext and adding changeLication class to change the location along with its currencies. When you change the Location, currencies will be updated at all the required places.

Excercise 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

1. import React from 'react';
2. import 'bootstrap/dist/css/bootstrap.min.css';
3. import { AppProvider } from '../context/AppContext';
4. import CartValue from './components/CartValue';
5. import ExpenseList from './components/ExpenseList';
6. import ItemSelected from './components/ItemSelected';
7. import Location from './components/Location';
8.
9. function App() {
10.  return (
11.    <div className="App">
12.      </div>
13.    );
14.  }
```

```
14. }
15.
16. export default App;
```

Copied!

- Click here to view the hint
- ▼ Click here to view the solution

```
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

1. import React from 'react';
2. import 'bootstrap/dist/css/bootstrap.min.css';
3.
4.
5. import { AppProvider } from './context/AppContext';
6. import CartValue from './components/CartValue';
7. import ExpenseList from './components/ExpenseList';
8. import ItemSelected from './components/ItemSelected';
9. import Location from './components/Location';
10.
11. const App = () => {
12.   return (
13.     <AppProvider>
14.       <div className='container'>
15.         <h1 className='mt-3'>Shopping App</h1>
16.         <div className='row mt-3'>
17.           <div className='col-sm'>
18.             <CartValue />
19.           </div>
20.           <div className='col-sm'>
21.             <Location />
22.           </div>
23.         </div>
24.         <h3 className='mt-3'>Shopping Cart</h3>
25.         <div className='row ' >
26.           <div className='col-sm'>
27.             <ExpenseList />
28.           </div>
29.         </div>
30.         <h3 className='mt-3'>Add Items</h3>
31.         <div className='row mt-3'>
32.           <div className='col-sm'>
33.             <ItemSelected/>
34.           </div>
35.         </div>
36.       </div>
37.     </AppProvider>
38.   );
39. };
40. export default App;
```

Copied!

- Here, you are importing different components,adding a bootstrap container that helps you center your App on the page. Importing your AppProvider and nested your components in the AppProvider element.

Exercise 4: Launch and view your react app on the browser

1. Make sure you are in the `kdu1a-shopping-app` directory and run the server using the following command.

```
1. 1
1. npm start
```

Copied!

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

The screenshot shows the Cloud IDE interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar contains a list of icons and menu items. The 'Launch Application' menu item is highlighted with a red box. The main area displays the 'Launch Your Application' dialog box, which includes a message about opening applications in the browser, an 'Application Port' input field, and a 'Your Application' button.

File Edit Selection View Go Run Terminal Help

SKILLS NETWORK TOO... ?

> DATABASES

> BIG DATA

> CLOUD

✓ OTHER

Launch Application

Open Link inside IDE

Open Cloud IDE with Open...

Open Cloud IDE Basic

Suggest a feature or submi...

About Cloud IDE

Launch Application x

Launch Your Application

To open any application in the browser, please select or enter the port number below.

Application Port

Your Application

Verify your output on the browser.

Shopping App

Cart Value: ₹0

Location

Shopping Cart

| Items | Quantity | Unit Price | Items Price | Remove |
|------------|----------|------------|-------------|---|
| Shirt | 0 | ₹500 | ₹0 |  |
| Jeans | 0 | ₹300 | ₹0 |  |
| Dress | 0 | ₹400 | ₹0 |  |
| Dinner set | 0 | ₹600 | ₹0 |  |
| Bags | 0 | ₹200 | ₹0 |  |

Add Items

Items

Jeans

▼

Quantity

Add

▼

Save

Congratulations! You have completed this sample shopping application lab and know how to create components of a Shopping 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.