Hands-on Lab - React Todo_List Application

Estimated Time Needed: 1 hour 30 mins

In this lab, you will be building a TODO list application using React components.You will discover how to create, view, delete, complete, modify and save to-do lists.

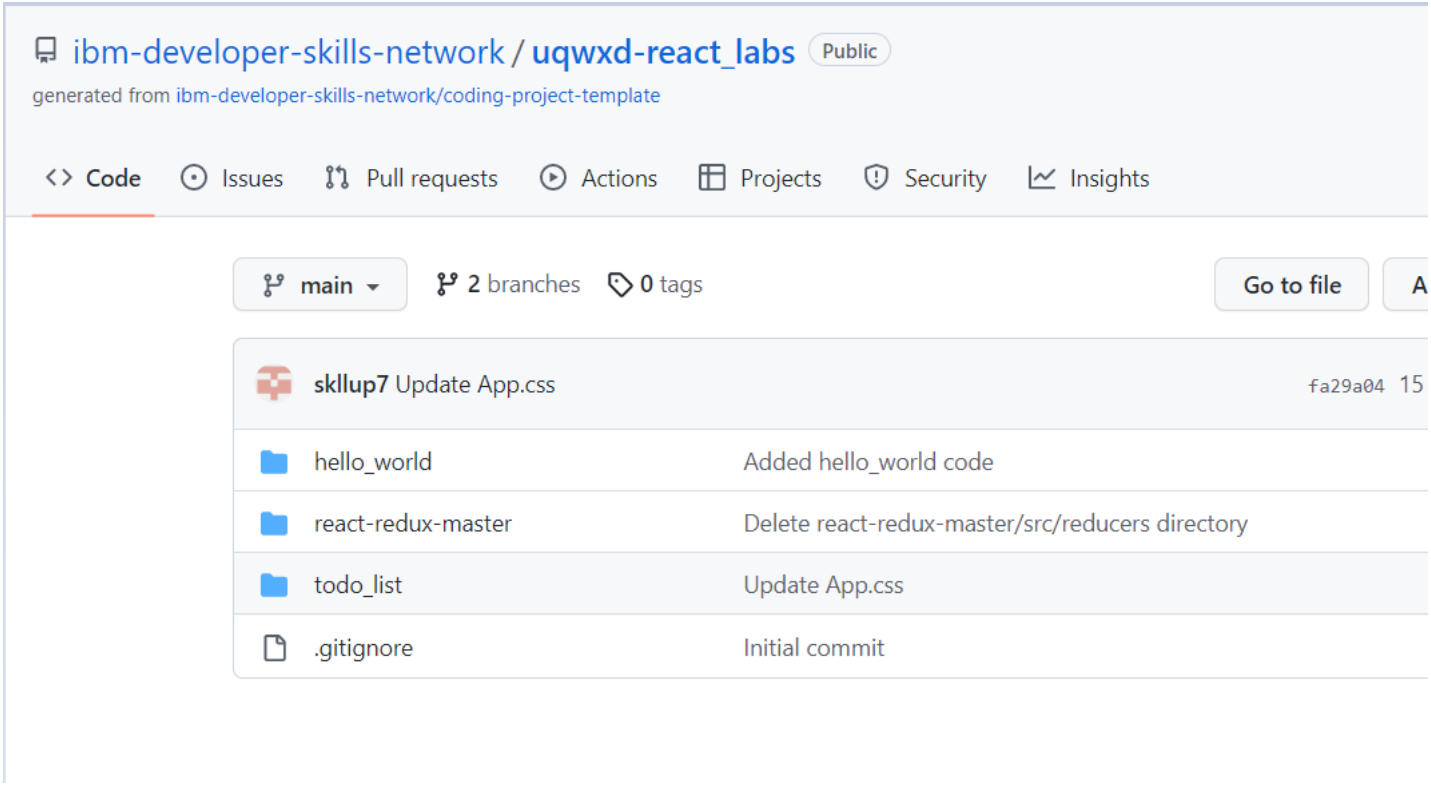
Objective:

After completing this lab, you will be able to:

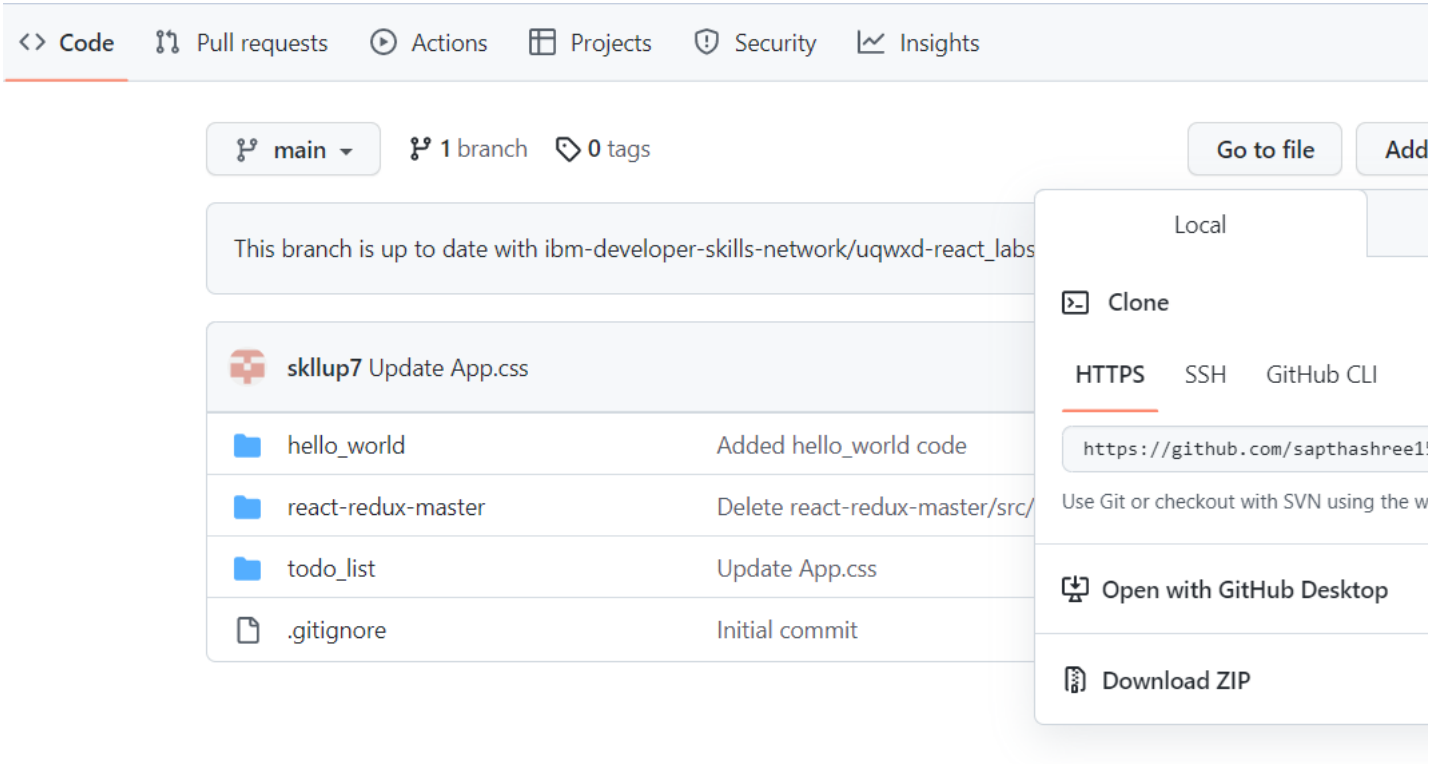
1. Use event handler for creating add todo task.
2. Delete the completed task from todo list using filter method.
3. Add the Toggle function and checkbox to check completed or not completed task.
4. Edit a added Todo task and submit it using map function.
5. Use the UseEffect hook to save new todos into localstorage.

Set-up : Fork and Clone the repository

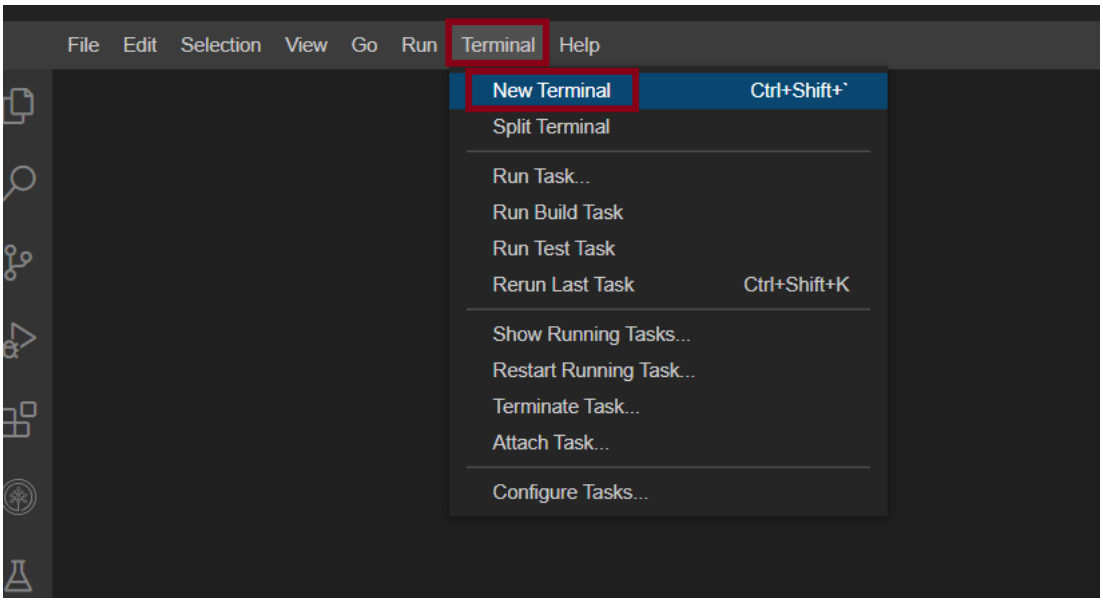
1. Go to the git repository https://github.com/ibm-developer-skills-network/uqwxd-react_labs.git that contains all the labs folder. You will be able to view todo-list for this lab which contains the starter code.
2. Create a fork of the repository into your own. You will need to have a github account of your own to do so.



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



4. In the lab environment, open a terminal window by choosing Terminal > New Terminal from the menu.



5. Change to your project folder, if you are not in the project folder already.

```
1
cd /home/project
```

Copied! Executed!

6. Clone your forked Git repository, if it doesn't already exist.

```
1
[ ! -d 'uqwxd-react_labs' ] && git clone https://github.com/<your Github username>/uqwxd-react_labs.git
```

Copied! Executed!

7. Change to the directory uqwxd-react_labs/todo_list/ to start working on the lab.

```
1
cd uqwxd-react_labs/todo_list/
```

Copied! Executed!

8. List the contents of this directory to see the artifacts for this lab.

```
1
ls
```

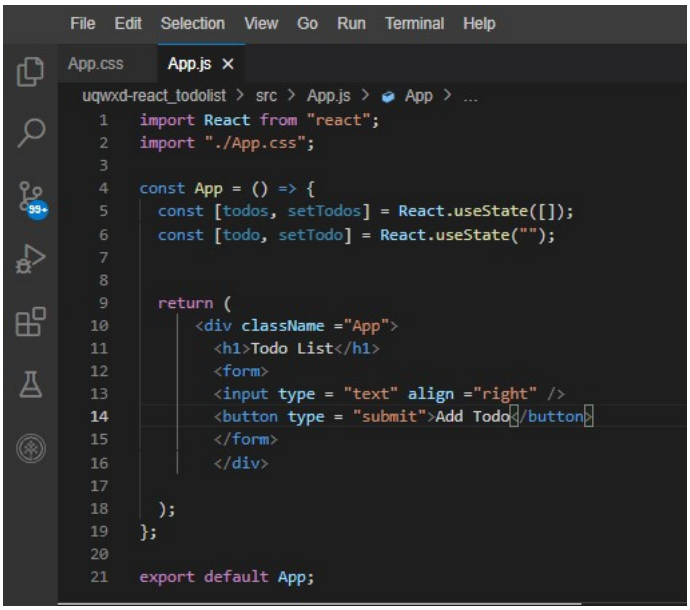
Copied! Executed!

```
theia@theiadocke          :/home/project$ cd uqwxd-react_labs/todo_list/
theia@theiadocke          :/home/project/uqwxd-react_labs/todo_list$ ls
LICENSE  package.json  package-lock.json  public  README.md  src  yarn.lock
theia@theiadocke          :/home/project/uqwxd-react_labs/todo_list$
```

Installing and running the server for React Application

The structure of the project folder and the starter code for To-Do list is shown in the below screenshot.

1. In the file Explorer go to App.js file under the src folder and view it. It would appear like this.



2. When you initialize the variable for the input field using the useState Hook, you define a getter, to get the value of the state and a setter, to set the value of the state. In the code above, **todos** is the state, and **setTodos** is the function that updates the state value. Similarly you have another state **todo** with its own setter.

```
1
2
```

```
const [todos, setTodos] = React.useState([]);
const [todo, setTodo] = React.useState("");
```

Copied!

3. In the terminal, ensure you are in the **uqwxd-react_labs/todo_list** directory and run the following command to install all the packages that are required for running the server.

```
1
```

```
npm install
```

Copied! Executed!

This will install all the required packages as defined in packages.json.

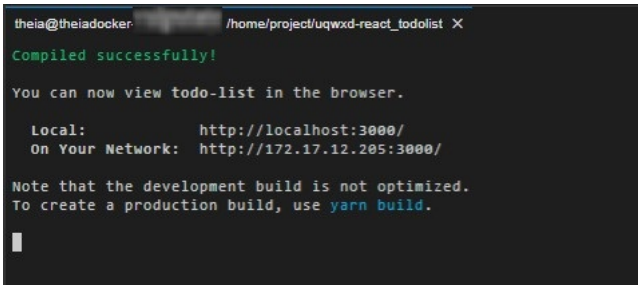
4. Start the server using the below command in the terminal.

```
1
```

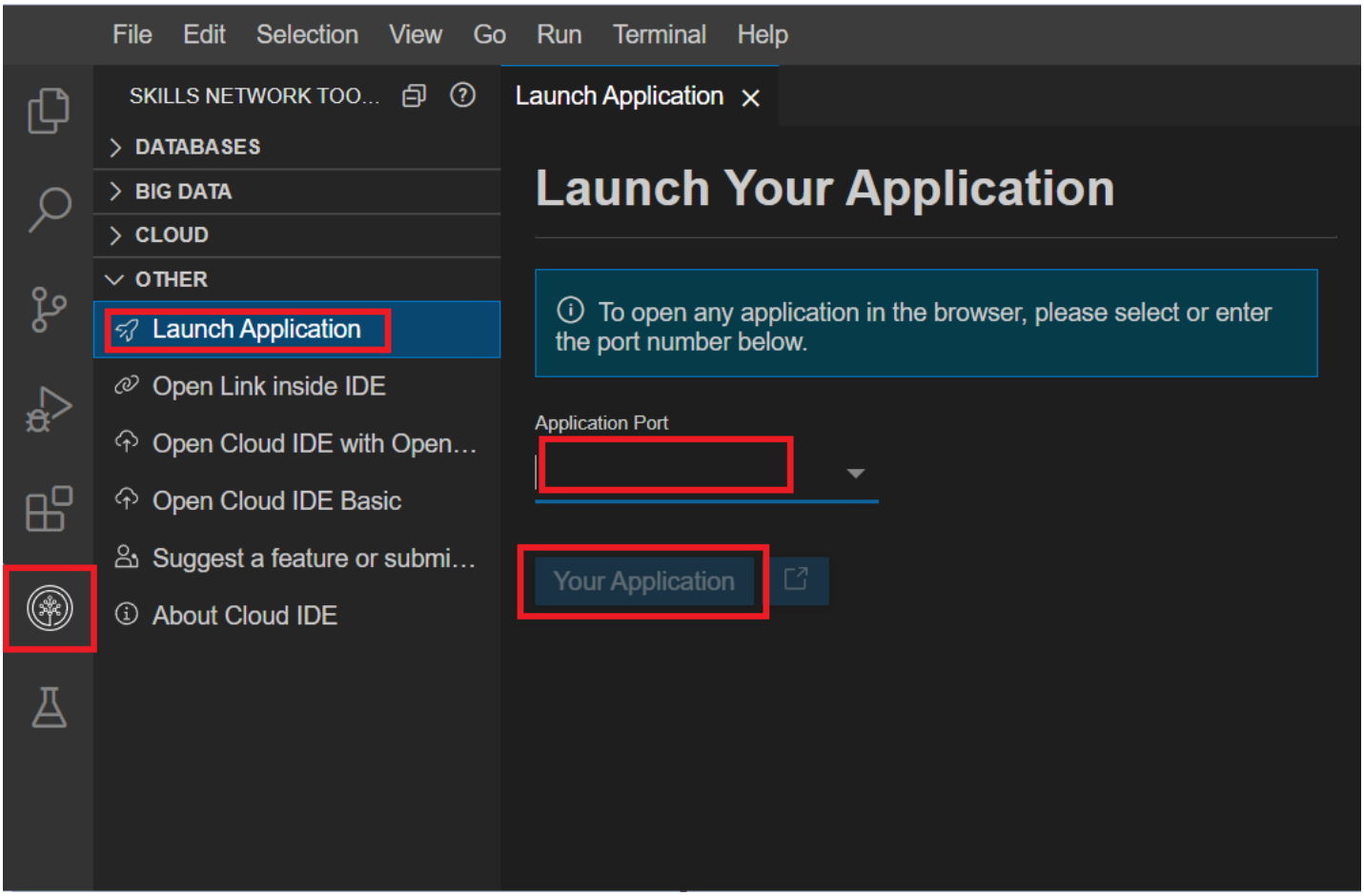
```
npm start
```

Copied! Executed!

You will see this output indicating that the server is running.



5. To verify that the server is running, click on the Skills Network button on the left to open the Skills Network Toolbox. Then click **Other**. Choose Launch Application and enter the port number 3000 on which the server is running and click the launch icon.



The todoapp UI will appear on the browser as seen in the image below.

6. To stop the server, go to the terminal in the lab environment and press Ctrl+c to stop the server.

Exercise 1: Create and Implement code to add task.

1. In **App.js** file there is a placeholder where you need to add the handlesubmit function. Now, let us give application the power to add a new task for our to-do list app. You will add a handleSubmit function that can handle newTodo items and add the task to the list. The user input is validated to ensure the input is non-empty and doesn't have preceeding or succeeding spaces.

1. 1
2. 2

```
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
function handleSubmit(e) {
    e.preventDefault();

    const newTodo = {
      id: new Date().getTime(),
      text: todo.trim(),
      completed: false,
    };
    if (newTodo.text.length > 0 ) {
        setTodos([...todos].concat(newTodo));
        setTodo("");

    } else {

        alert("Enter Valid Task");
        setTodo("");
    }
  }
```

Copied!

2. On submitting the form, the task is added to the todo array. The code uses the map to iterate through the todo array, and renders each task as a list item. Using useState, this component registers a state — value — and a function for updating it — setTodo. The handleSubmit handler will prevent the default action that would normally be taken on the form and add a new Task using the latest value that is in the input field.

Paste the below code inside the return function.

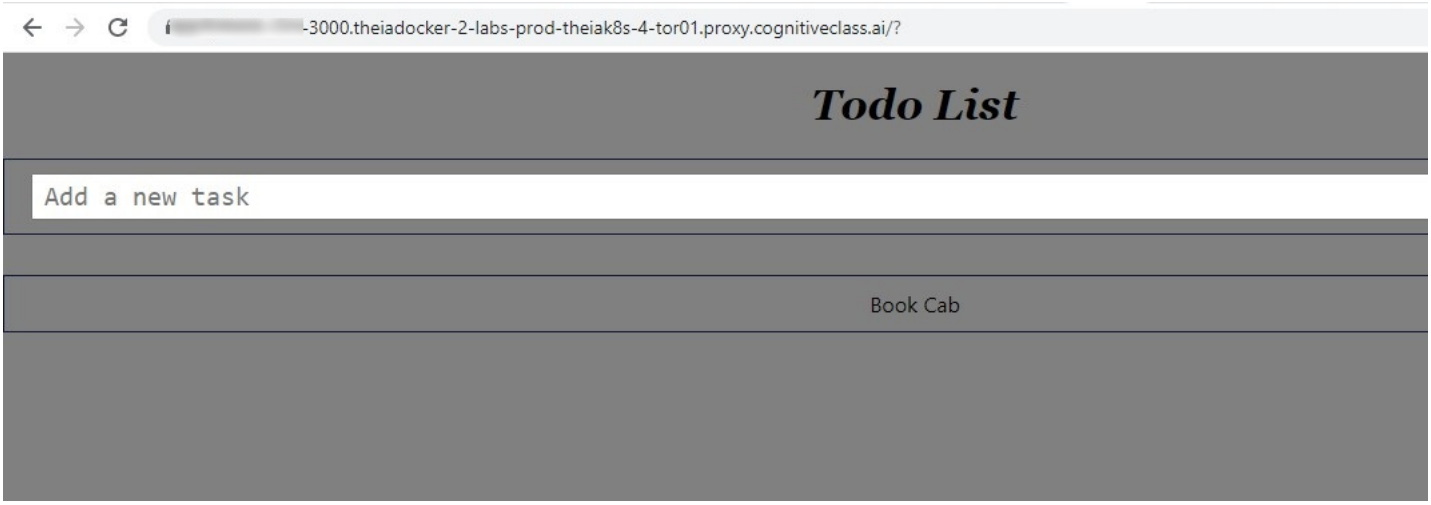
```
1
2
3
4
5
6
7
8
9
10
11
```

```
<h1>Todo List</h1>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          onChange={(e) => setTodo(e.target.value)}
          placeholder="Add a new task"
          value={todo}
        />
            <button type="submit">Add Todo</button>
        </form>
        {todos.map((todo) => <div>{todo.text}</div>)}
```

Copied!

▶ Click here to view the complete code.

3.Start the server and enter the valid task inside the input box where you can see “Add a new task “. You will see the ouput as below:



Exercise 2: Delete a completed task from the list.

1. Delete task can be handled in many ways. Now write the code using the filter method that will be applied when a button is clicked. It filters out the task to be deleted and returns the rest of the tasks. Placeholder is added in the App.js file where you need to add the delete to do function.

▶ Click here to view the code.

2. Add a button to delete the task.

▶ Click here to view the code.

3. Start the server, add the task to-do list, and then try to delete it by pressing the delete button. The task must be removed from the list.

Exercise 3: Adding Checkbox and Toggle function.

1. You will now add a checkbox to mark task completion. Create a new function, **toggleComplete** function that uses the map method to iterate through the task and mark them complete inside App.js.

```
1
2
3
4
5
6
7
8
9
```

```
function toggleComplete(id) {
    let updatedTodos = [...todos].map((todo) => {
      if (todo.id === id) {
        todo.completed = !todo.completed;
      }
      return todo;
    });
    setTodos(updatedTodos);
  }
```

Copied!

Code to add the checkbox above the delete button.

```
1
```

```
<input type="checkbox" id="completed" checked={todo.completed} onChange={() => toggleComplete(todo.id)}/>
```

Copied!

2. Start the server, add the task to-do list, and then use the checkbox if the task is completed.

Exercise 4: Edit a added Todo task and submit it.

1. In the previous exercise, you added the ability to toggle the checkbox for completed task in todos. Now add the functionality to edit the task.
2. You need to add two more states to implement the editing functionality.

```
1
2
```

```
const [todoEditing, setTodoEditing] = React.useState(null);
const [editingText, setEditingText] = React.useState("");
```

Copied!

3. Now add the submitEdit function in the App.js which will help you to submit the task of todo list using the map function.

▶ Click here to view the code

4. You will be allowing edits only in the edit mode. Check for the appropriate mode, then display the editing form where one can edit the to-do list and submit it back to the list using the ternary operator. The editing form has a couple of additional buttons so the user can control between submitting an edit and deleting the task.

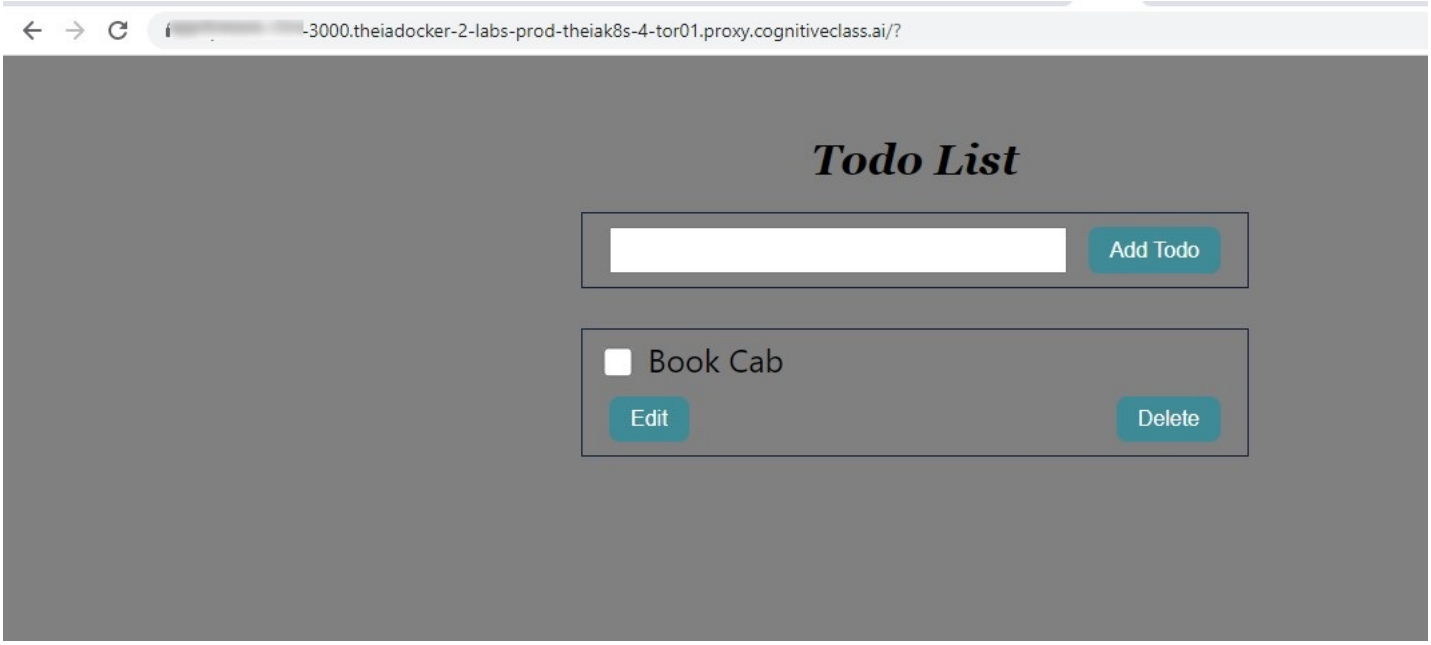
```
1
2
3
4
5
6
7
8
9
10
11
12
```

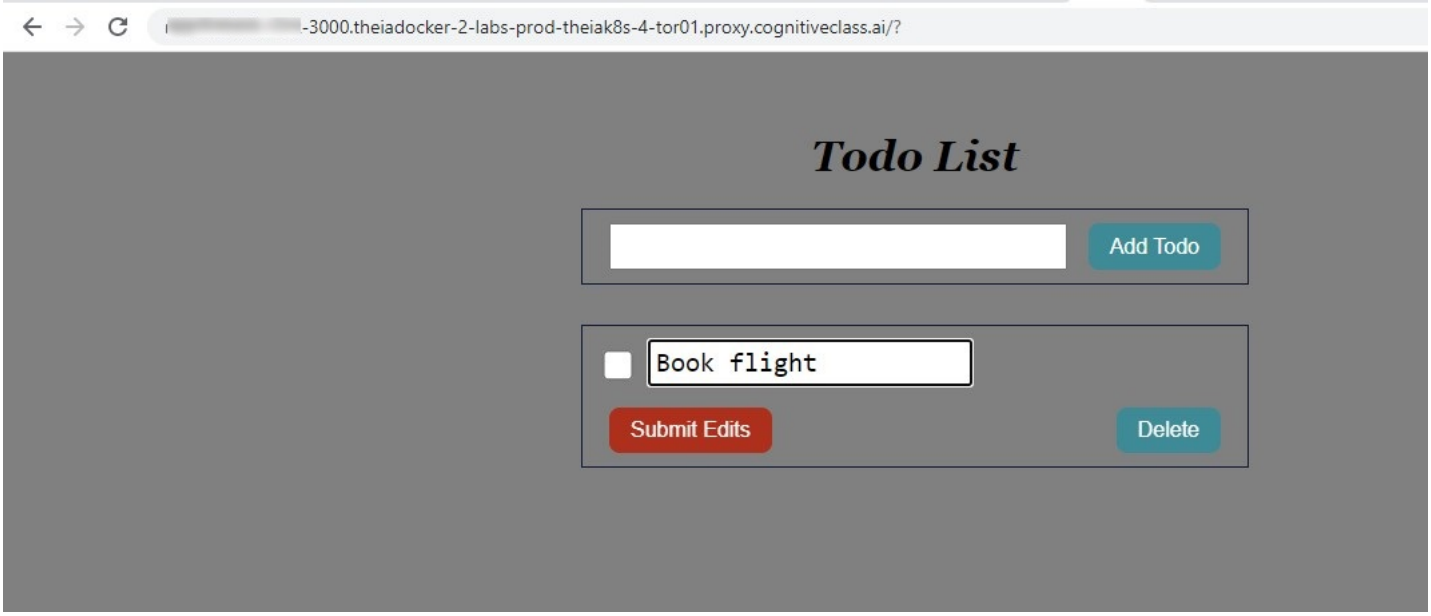
```
{todo.id === todoEditing ? (
<input type="text" onChange={(e) => setEditingText(e.target.value)} />
) : (
<div>{todo.text}</div>
)}
</div>
<div className="todo-actions">
{todo.id === todoEditing ? (
<button onClick={() => submitEdits(todo.id)}>Submit Edits</button>
) : (
<button onClick={() => setTodoEditing(todo.id)}>Edit</button>
)}
```

Copied!

▶ Click here to view the complete code.

5. Start the server, add the task to-do list and edit it using the edit button.





Exercise 5: Adding the useEffect hook.

1. You will add the useEffect hook to your application. This useEffect hook will be responsible to save new todos into localstorage.

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

```
React.useEffect(() => {
    const json = localStorage.getItem("todos");
    const loadedTodos = JSON.parse(json);
    if (loadedTodos) {
      setTodos(loadedTodos);
    }
  }, []);

  React.useEffect(() => {
    if([todos].length > 0) {
        const json = JSON.stringify(todos);
        localStorage.setItem("todos", json);
    }
  }, [todos]);
```

Copied!

The wholistic view

1. After putting everything together, the final code for To-Do list is as follows.

▶ Click here to view the complete code

2. Please commit and push all the changes to your forked Github repo.

   Note: Please refer to this lab, if you need any help in committing and pushing changes to your repo.

Congratulations! You have completed the lab for creating To-Do list Application using React.

Summary

In this lab, you used React to build a to-do list application that allows you to view the list of tasks, add tasks, edit tasks, and delete tasks.

Author(s)

Sapthashree K S

Changelog

Date	Version	Changed by	Change Description
2022-10-28	1.0	Sapthashree K S	Initial version created based videos
2023-01-31	1.1	Sapthashree K S	Instructions and screenshots updated