

REACT 2

CSE5006 – LAB 5

REPOSITORY:

[HTTPS://GITHUB.COM/CSE5006/LAB-5](https://github.com/CSE5006/LAB-5)

TABLE OF CONTENTS

1. Introduction	3
1.2. Clone and Fork	3
2. React State	5
2.1. Making the Task Interactive	5
2.2. Implementing Task Deletion	11
2.3. Exercise 1	11

1. INTRODUCTION

In the previous lab, we've built a static task list app. While the app can render a list of tasks, these tasks can't be interacted with by the user. In this lab, we will make our app interactive by making use of React state, and implementing features like task completion and deletion.

React state is what makes React components interactive. It's a source of data that can change over time, triggering re-renders of components when it does. This is in contrast to props, which are unchanging over the lifespan of a component.

Let's get started.

1.2. CLONE AND FORK

1. Fork the react-task-list-interactive project from <https://github.com/CSE5006/lab-5> as your own private GitHub repository.
2. Use the Git command line tool to clone a local copy of the repository.

The following example clones the project directly

```
vboxuser@CSE5006:~/Documents$ git clone https://github.com/CSE5006/lab-5
Cloning into 'lab-5'...
remote: Enumerating objects: 57, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (45/45), done.
remote: Total 57 (delta 6), reused 49 (delta 3), pack-reused 0
Receiving objects: 100% (57/57), 319.24 KiB | 3.10 MiB/s, done.
Resolving deltas: 100% (6/6), done.
vboxuser@CSE5006:~/Documents$
```

3. Open a new terminal window and run the following command to start the server.

```
docker compose up --build
```

```
vboxuser@CSE5006:~/Documents$ cd lab-5
vboxuser@CSE5006:~/Documents/lab-5$ docker compose up --build
[+] Building 2.9s (7/7) FINISHED
=> [app internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 157B                                                  0.0s
=> [app internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                        0.0s
=> [app internal] load metadata for docker.io/library/node:18.16.1-alpin             2.7s
=> [app 1/3] FROM docker.io/library/node:18.16.1-alpine3.18@sha256:d5b2a            0.0s
=> CACHED [app 2/3] RUN mkdir -p /app                                                0.0s
=> CACHED [app 3/3] WORKDIR /app                                                      0.0s
=> [app] exporting to image                                                           0.0s
=> => exporting layers                                                                0.0s
=> => writing image sha256:f6eee69f53aa457947a04c1ab69d7aa9f3d720f201c4d          0.0s
=> => naming to docker.io/library/lab-5-app                                          0.0s
```

Wait until you see the following output

```
lab-5-app-1 |  
lab-5-app-1 | Compiled successfully!  
lab-5-app-1 |  
lab-5-app-1 | You can now view lab-4 in the browser.  
lab-5-app-1 |  
lab-5-app-1 | Local: http://localhost:3000  
lab-5-app-1 | On Your Network: http://172.19.0.2:3000  
lab-5-app-1 |  
lab-5-app-1 | Note that the development build is not optimized.  
lab-5-app-1 |  
lab-5-app-1 | To create a production build, use npm run build.  
lab-5-app-1 |  
lab-5-app-1 | webpack compiled successfully
```

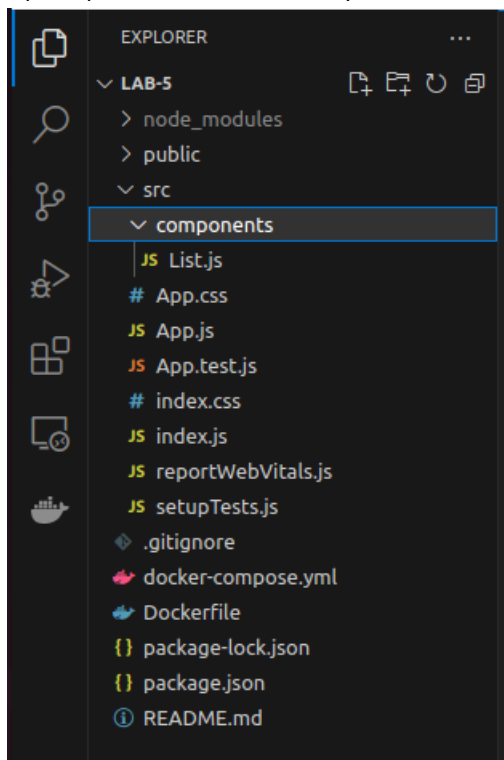
4. Point your web browser to <http://localhost:3000> in order to load the website after the container has finished starting.



My Tasks

- Learn React ☒
- Learn JSX ☐
- Build a React App ☐

Open up the entire lab directory in Visual Studio Code.



The files that we will be working on are under the components directory.

2. REACT STATE

State in React is a way for components to maintain and change their own data. This makes it possible to create components that respond to user interactions.

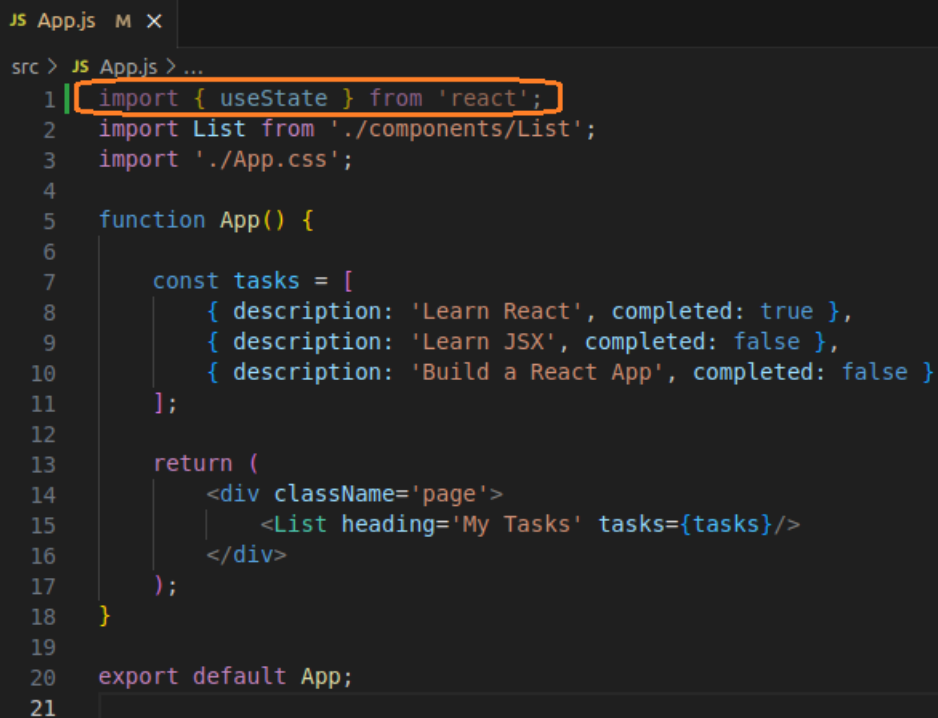
In our task list app, we want to be able to mark tasks as completed and remove them from the list. To make these features work, we will use React state.

2.1. MAKING THE TASK INTERACTIVE

We're going to update our Root component so that the completed status can be toggled.

1. First, import the `useState` hook from React at the top of your `App.js` file with:

```
import { useState } from 'react';
```



```
JS App.js M X
src > JS App.js > ...
1 | import { useState } from 'react';
2 | import List from './components/List';
3 | import './App.css';
4 |
5 | function App() {
6 |
7 |   const tasks = [
8 |     { description: 'Learn React', completed: true },
9 |     { description: 'Learn JSX', completed: false },
10 |    { description: 'Build a React App', completed: false }
11 |  ];
12 |
13 |   return (
14 |     <div className='page'>
15 |       <List heading='My Tasks' tasks={tasks}/>
16 |     </div>
17 |   );
18 | }
19 |
20 | export default App;
21 |
```

2. Inside the Root component, replace the tasks const with the following:

```
const [tasks, setTasks] = useState([
  { id: 1, description: 'Learn React', completed: true },
  { id: 2, description: 'Learn JSX', completed: false },
  { id: 3, description: 'Build a React App', completed: false }
]);
```

Also pass setTasks to List as a prop. Please note that we also add “id” property to every task. We will use the id later on as the component identifier.

```
JS App.js M ●
src > JS App.js > ...
1 | import { useState } from 'react';
2 | import List from './components/List';
3 | import './App.css';
4 |
5 | function App() {
6 |
7 |     const [tasks, setTasks] = useState([
8 |       { id:1, description: 'Learn React', completed: true },
9 |       { id:2, description: 'Learn JSX', completed: false },
10 |      { id:3, description: 'Build a React App', completed: false }
11 |    ]);
12 |
13 |     return (
14 |       <div className='page'>
15 |         <List heading='My Tasks' tasks={tasks} setTasks={setTasks}/>
16 |       </div>
17 |     );
18 |   }
19 |
20 |   export default App;
```

Now we're going to make it so that clicking on the checkbox changes the completed state.

1. Update the map function call in List.js to the following:

```

{ props.tasks.map(task =>
  <Task
    setTasks={props.setTasks}
    id={task.id}
    description={task.description}
    completed={task.completed}
  />
) }

```

```

8
9  function List(props) {
10
11    return (
12      <div>
13        <h1>{ props.heading }</h1>
14        <ul>
15          { props.tasks.map(task =>
16            <Task
17              setTasks={props.setTasks}
18              id={task.id}
19              description={task.description}
20              completed={task.completed}
21            />)
22          }
23        </ul>
24      </div>
25    );
26  }
27
28  export default List;
29

```

2. Update the checkbox in your Task component to remove the readOnly attribute and add an onChange attribute. The onChange attribute should be a function that calls **setCompleted** with the opposite of the current completed state:


```

function Task(props) {

  console.log(props);

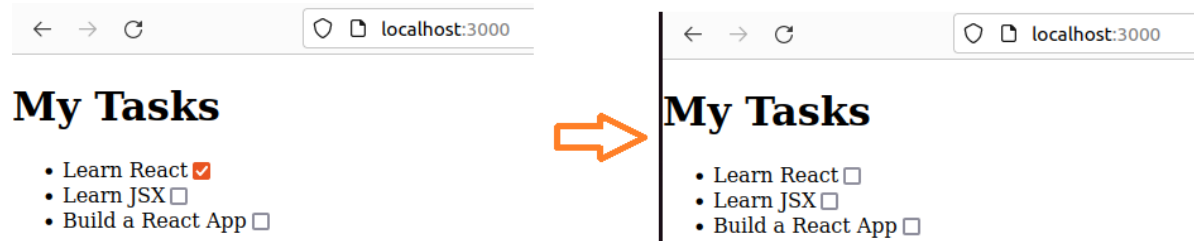
  function onChange() {
    // Find the task we want to update and update it
    props.setTasks(tasks => tasks.map(task => {
      if (task.id === props.id) {
        return {
          id: task.id,
          description: task.description,
          completed: !task.completed
        };
      } else {
        return task;
      }
    }));
  }

  return (
    <li>
      { props.description }
      <input
        type="checkbox"
        checked={props.completed}
        onChange={onChange}
      />
    </li>
  );
}

```

```
JS App.js M JS List.js M X
src > components > JS List.js > ...
1 import { useState } from 'react';
2
3 function Task(props) {
4   console.log(props);
5   function onChange() {
6     // Find the task we want to update and update it
7     props.setTasks(tasks => tasks.map(task => {
8       if (task.id === props.id) {
9         return {
10           id: task.id,
11           description: task.description,
12           completed: !task.completed
13         };
14       } else {
15         return task;
16       }
17     }));
18   }
19
20   return (
21     <li>{ props.description }
22     <input
23       type="checkbox"
24       checked={props.completed}
25       onChange={onChange}
26     />
27   </li>
28 );
29 }
30
```

Try clicking the checkbox next to one of the tasks! It should now be able to toggle.



2.2. IMPLEMENTING TASK DELETION

The next piece of interactivity we're going to add to our app is the ability to delete tasks.

1. Inside the Task component, we can create a function that deletes our task from the list of tasks. Like this:

```
function onClick() {  
  // Find the task we want to delete and remove it  
  props.setTasks(tasks => tasks.filter(task => task.id !== props.id));  
}
```

This function will filter to get all task where the id is not the same as the props.id

2. Then we can add a button that deletes the task. Put this button just after the opening tag:

```
<button type="button" onClick={onClick}>X</button>
```

```
3 function Task(props) {  
4   console.log(props);  
5   function onChange() {  
6     // Find the task we want to update and update it  
7     props.setTasks(tasks => tasks.map(task => {  
8       if (task.id === props.id) {  
9         return {  
10          id: task.id,  
11          description: task.description,  
12          completed: !task.completed  
13        };  
14      } else {  
15        return task;  
16      }  
17    }));  
18  }  
19  
20  function onClick() {  
21    props.setTasks(tasks => tasks.filter(task => task.id !== props.id));  
22  }  
23}
```

Wow, that was simple, your button should now delete the task, try it!



2.3. EXERCISE 1

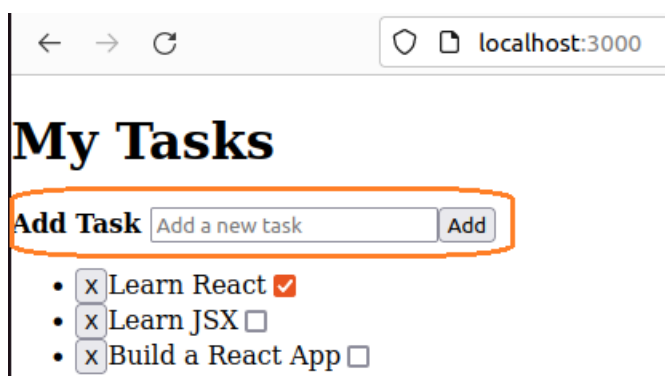
Next, we're going to allow users to add new tasks to the task list.

1. Inside the List component, create an **input** field and a **button** with the label "Add Task".
2. Declare a new state variable, **newTask**, and the corresponding setter function, **setNewTask**, using the **useState** hook. The initial state should be an empty string.
3. The **input** field should have a value attribute of **newTask** and an **onChange** attribute. The **onChange** attribute should be a function that calls **setNewTask** with the current value of the **input** field.
4. The "Add Task" button should have an **onClick** attribute. The **onClick** attribute should be a function that adds a new task to the tasks array by calling **setTasks**.

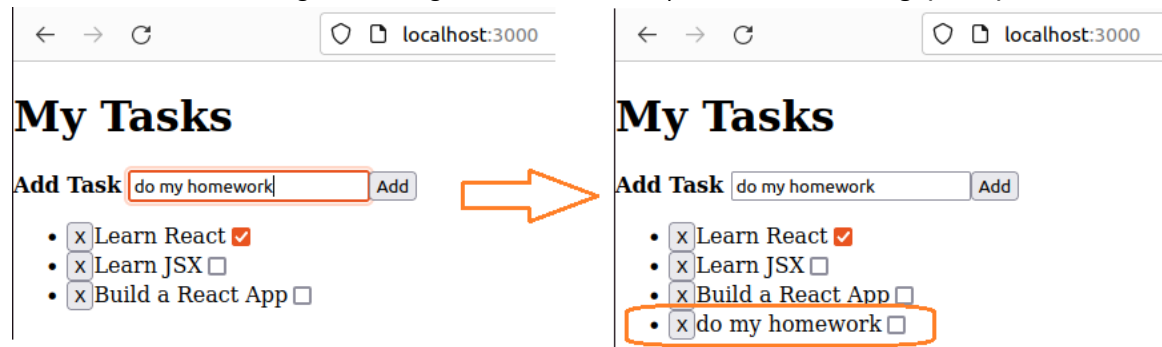
```

37 function List(props) {
38
39   const [newTask, setNewTask] = useState(""); 2
40   function onChange(event) 3
41   {
42     setNewTask(event.target.value);
43   }
44
45   function onClick() 4
46   {
47     props.setTasks(tasks => [...tasks, { id: tasks.length+1,
48                                           description: newTask,
49                                           completed: false }]);
50   }
51
52   return (
53     <div>
54       <h1>{ props.heading }</h1> 1
55       <b>Add Task </b><input type="text" 3
56                           placeholder="Add a new task"
57                           onChange={onChange}/>
58       <button type="button" onClick={onClick}>Add</button> 4
59       <ul>
60         { props.tasks.map(task =>
61           <Task
62             setTasks={props.setTasks}
63             id={task.id}
64             description={task.description}
65             completed={task.completed}
66           />)
67         }
68       </ul>
69     </div>
70   );
71 }
72
73 export default List;

```



Now, users should be able to add new tasks to the task list! **Hint:** you can pass an “event” to the onChange function and use event.target.value to get the value of the input. **function onChange(event).**



Congratulations! You've now created an interactive React app! Test it by checking off tasks, and see that their completed status changes. Try deleting tasks and see them disappear from the list. This app now has state and responds to user interactions!