

REACT 1

CSE5006 – LAB 4

REPOSITORY:

[HTTPS://GITHUB.COM/CSE5006/LAB-4](https://github.com/CSE5006/LAB-4)

TABLE OF CONTENTS

1. Introduction	3
1.1. Forking and Running the Initial Application.....	3
2. Stateless Functional Components.....	6
2.1. Create your own Component.....	8
2.2. Tasks as Plain old JavaScript Objects.....	12
2.3. Using the map Function	16

1. INTRODUCTION

In this lab, we will practice programming in Facebook's React JavaScript library. React is a whole new way to write frontend JavaScript applications. It may seem a bit weird at first, but it is really nice once you get used to it and will ultimately make writing big applications easier.

You may recall from an earlier lab that in the web browser, the page is represented as a DOM (Document Object Model) and that JavaScript can manipulate the DOM to change what is displayed to the user. However, as applications grow in complexity, it can quickly become difficult to manage many different DOM manipulations as data changes. This can lead to big problems, such as the interface becoming inconsistent.

For example, let's say you are showing the logged-in username on the screen in two places. When the user logs out, you could easily forget to update one of those elements. Code gets duplicated, things get messy, and it becomes harder and harder to develop features and fix bugs as the application grows.

React fixes these problems by taking a novel approach to DOM manipulation. Instead of manually writing imperative code to manipulate the DOM, in React, you define a mapping from your data (also called the model or the store) to the view. Whenever your data model changes, React will automatically rerender the view from scratch.

While this may seem very slow and inefficient at first, React is clever in the way it handles this re-rendering. Specifically, React is able to analyze changes between subsequent renders and only apply the differences to the actual DOM. This makes the life of a frontend developer much easier. Instead of thinking about intermingled model and view manipulations scattered throughout the project, the mapping from model to view can be defined in one place.

Now, whenever the model changes, we can rest assured that the view will show the correct thing.

In React, everything is a component, and these components are linked together in a tree-like structure, similar to the DOM. Data is passed from a parent component to a child component through properties, also known as "props" for short. React also allows components to store internal information using state variables.

For more detailed information about properties and state, I recommend referring back to the lecture notes to learn more.

1.1. FORKING AND RUNNING THE INITIAL APPLICATION

1. Fork the react-task-list project from <https://github.com/CSE5006/lab-4> as your own private GitHub repository.
2. Use the Git command line tool to clone a local copy of the repository.
In this example, the remote repository is cloned directly from the source

```
vboxuser@CSE5006:~$ cd Documents
vboxuser@CSE5006:~/Documents$ git clone https://github.com/CSE5006/lab-4
Cloning into 'lab-4'...
remote: Enumerating objects: 74, done.
remote: Counting objects: 100% (74/74), done.
remote: Compressing objects: 100% (54/54), done.
remote: Total 74 (delta 15), reused 65 (delta 10), pack-reused 0
Receiving objects: 100% (74/74), 319.36 KiB | 491.00 KiB/s, done.
Resolving deltas: 100% (15/15), done.
vboxuser@CSE5006:~/Documents$
```

```
vboxuser@CSE5006:~/Documents$ cd lab-4
vboxuser@CSE5006:~/Documents/lab-4$ ls -l
total 1224
-rw-rw-r-- 1 vboxuser vboxuser 102 Aug 18 06:41 docker-compose.yml
-rw-rw-r-- 1 vboxuser vboxuser 120 Aug 18 06:41 Dockerfile
-rw-rw-r-- 1 vboxuser vboxuser 808 Aug 18 06:41 package.json
-rw-rw-r-- 1 vboxuser vboxuser 1224992 Aug 18 06:41 package-lock.json
drwxrwxr-x 2 vboxuser vboxuser 4096 Aug 18 06:41 public
-rw-rw-r-- 1 vboxuser vboxuser 3359 Aug 18 06:41 README.md
drwxrwxr-x 3 vboxuser vboxuser 4096 Aug 18 06:41 src
vboxuser@CSE5006:~/Documents/lab-4$
```

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

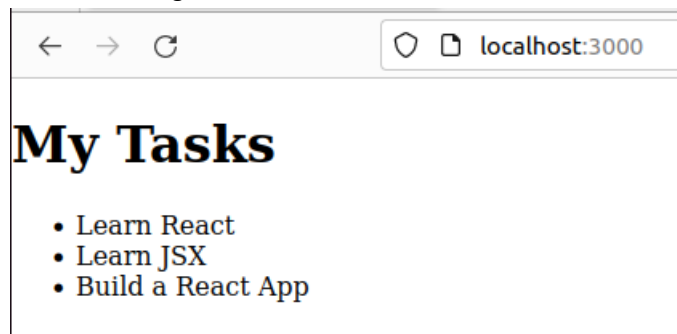
```
docker compose up --build
```

```
vboxuser@CSE5006:~/Documents/lab-4$ docker compose up --build
[+] Building 4.7s (3/6)
=> [app internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 157B 0.0s
=> [app internal] load .dockerignore 0.1s
=> => transferring context: 2B 0.0s
=> [app internal] load metadata for docker.io/library/node:18.16.1-alpin 2.9s
=> [app 1/3] FROM docker.io/library/node:18.16.1-alpine3.18@sha256:d5b2a 1.7s
=> => resolve docker.io/library/node:18.16.1-alpine3.18@sha256:d5b2a7869 0.0s
=> => sha256:560412e561fb4693d4bc6e5d197ccdfb42d1453d0e 5.24MB / 47.49MB 1.7s
=> => sha256:02735cb6c78bf40ddce33f02c68827a8ae3fdc99370db62 0B / 2.34MB 1.7s
=> => sha256:86d562f7b85533d4f8cf87a09747444f7002c74b1897f5e 450B / 450B 1.4s
=> => sha256:d5b2a7869a4016b1847986ea52098fa404421e44281 1.43kB / 1.43kB 0.0s
=> => sha256:bf6c61feabc1a1bd565065016abe77fa378500ec75e 1.16kB / 1.16kB 0.0s
=> => sha256:f85482183a4f18ec843fea14e64575a5550f153211c 6.73kB / 6.73kB 0.0s
```

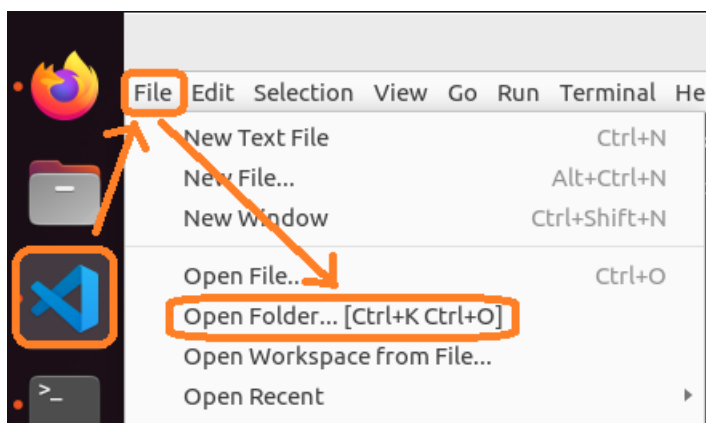
If you see any warnings, just ignore them and wait until you see the following output

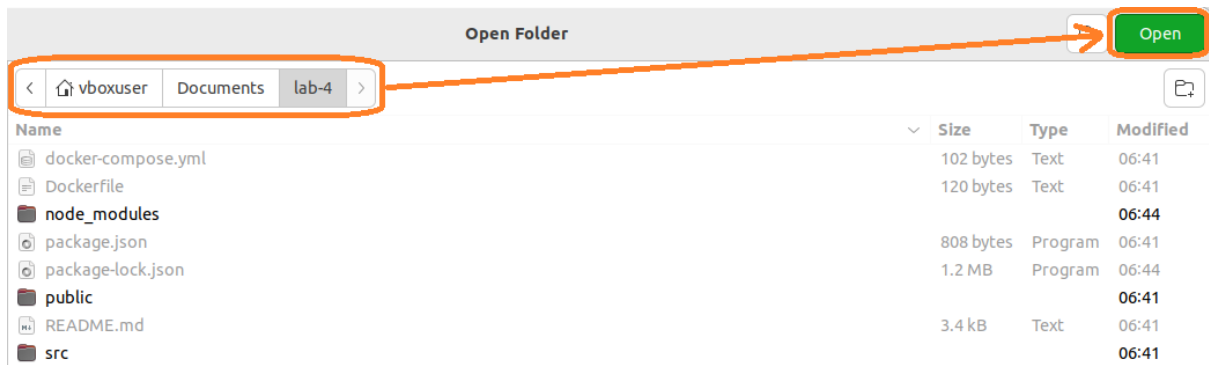
```
lab-4-app-1 | babel-preset-react-app is part of the create-react-app project, w  
hich  
lab-4-app-1 | is not maintained anymore. It is thus unlikely that this bug will  
lab-4-app-1 | ever be fixed. Add "@babel/plugin-proposal-private-property-in-ob  
ject" to  
lab-4-app-1 | your devDependencies to work around this error. This will make th  
is message  
lab-4-app-1 | go away.  
lab-4-app-1 |  
lab-4-app-1 |  
lab-4-app-1 | Compiled successfully!  
lab-4-app-1 |  
lab-4-app-1 | You can now view lab-4 in the browser.  
lab-4-app-1 |  
lab-4-app-1 |  
lab-4-app-1 | Local: http://localhost:3000  
lab-4-app-1 | On Your Network: http://172.18.0.2:3000  
lab-4-app-1 |  
lab-4-app-1 | Note that the development build is not optimized.  
lab-4-app-1 | To create a production build, use npm run build.  
lab-4-app-1 |  
lab-4-app-1 |  
lab-4-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.

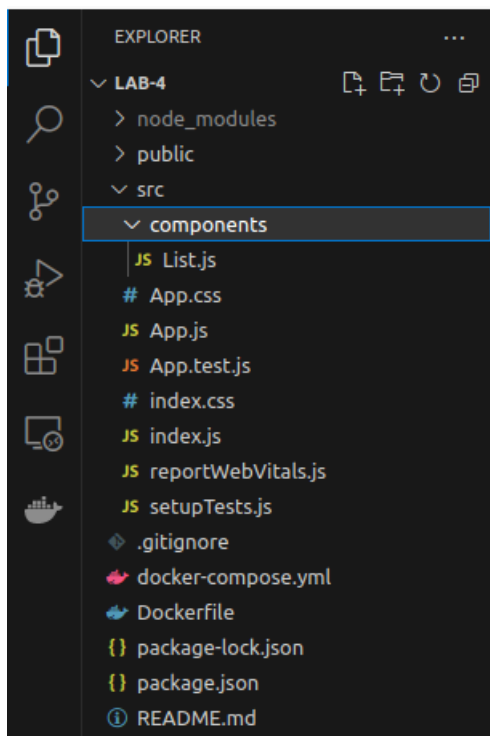


Open up the entire lab directory in Visual Studio Code.





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



They are index.js, App.js, List.js, Task.js. The index.js and App.js files contain the root React component from which all the other components are descended. You can think of it like the main function in a Java program. The List.js contains the Task List component which will display a todo list. In this lab we will create a very simple todo list with very limited functionality. In the next lab we will make it do much cooler things!

2. STATELESS FUNCTIONAL COMPONENTS

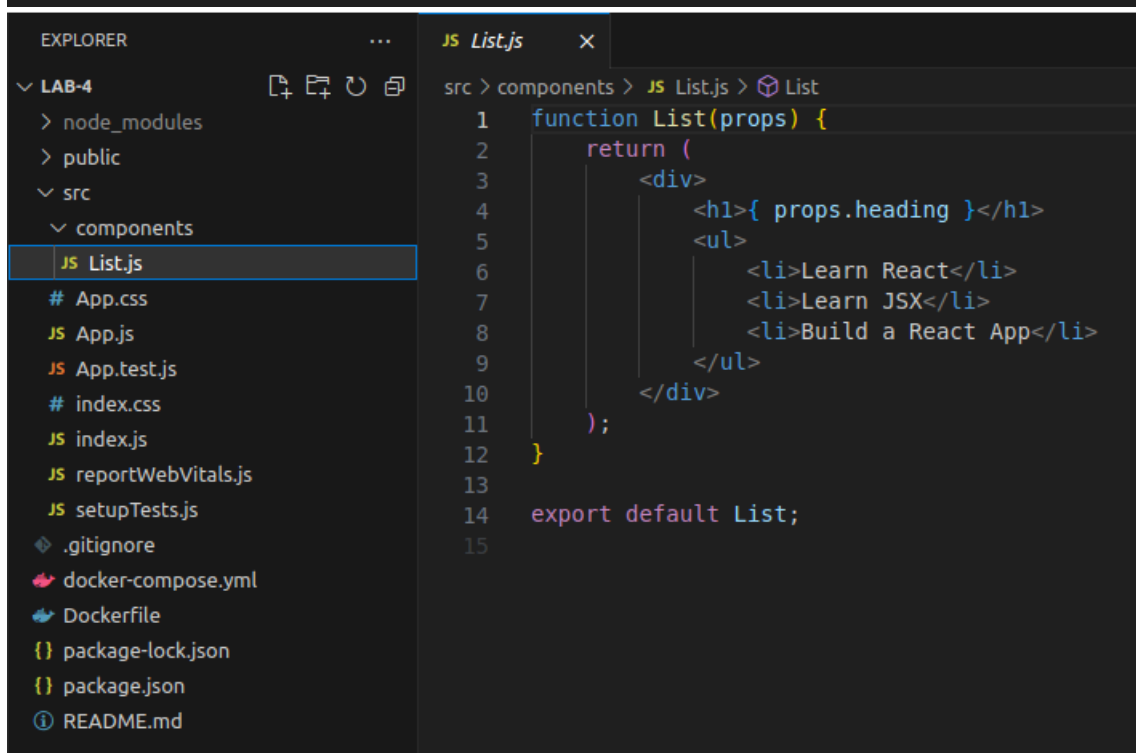
A React component is like a custom HTML tag. To begin with, we will be using the most simple type of React components: “stateless functional components”. Creating a stateless functional component is as simple as writing a JavaScript function.

Our task list application already has a component called List, which is defined in List.js.

```

function List(props) {
  return (
    <div>
      <h1>{ props.heading }</h1>
      <ul>
        <li>Learn React</li>
        <li>Learn JSX</li>
        <li>Build a React App</li>
      </ul>
    </div>
  );
}

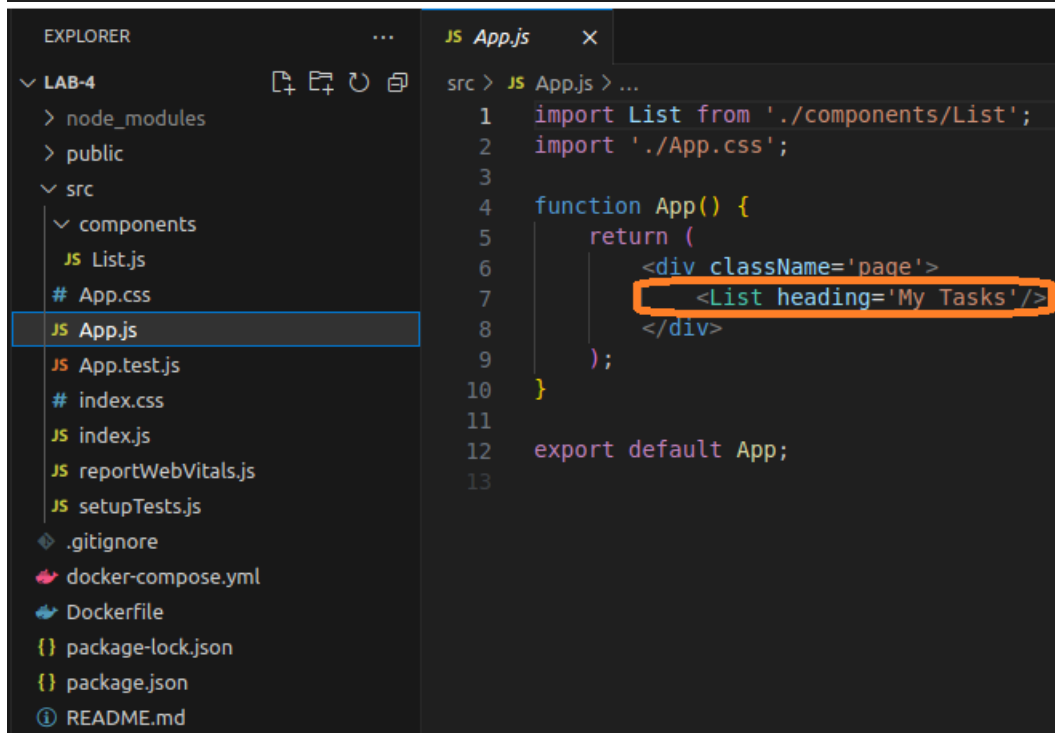
```



As you can see, the List is just a rendering function that accepts a single argument, "props", and returns JSX code that will be rendered as part of the web page. It consists of an `` unordered list containing three hard-coded `` list items. The curly brackets in JSX allow us to insert snippets of JavaScript code, in this case, to access the value of "props.heading".


Once List has been defined, it can be used in the JSX of other components like a normal HTML tag. In our application, this happens in the Root component found in App.js.

```
<List heading='My Tasks' />
```



```
src > JS App.js > ...
1  import List from './components/List';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className='page'>
7        <List heading='My Tasks' />
8      </div>
9    );
10 }
11
12 export default App;
13
```

The attributes of the tag (in this case, there's just one, "heading") are combined into an object by React and become the "props" argument of the stateless functional component. So when it's time for React to render our application, it will automatically call `List({ heading: "My task list" })` to determine what elements should be inserted into the DOM.



```
src > JS App.js > ...
1  import List from './components/List';
2  import './App.css';
3
4  function App() {
5    return (
6      <div className='page'>
7        <List heading='My Tasks' />
8      </div>
9    );
10 }
11
12 export default App;
13
```

```
src > components > JS List.js > List
1  function List(props) {
2    return (
3      <div>
4        <h1>{ props.heading }</h1>
5        <ul>
6          <li>Learn React</li>
7          <li>Learn JSX</li>
8          <li>Build a React App</li>
9        </ul>
10      </div>
11    );
12 }
13
14 export default List;
15
```

For the List component, **props.heading** becomes the text of the `<h1>` heading tag.

2.1. CREATE YOUR OWN COMPONENT

Now let's create a new component called Task, which we will put in the file List.js for convenience. We want to pass a description prop from the List component to the Task component. The Task component will then display the description with a bullet point using the **HTML** list item tag, ``.

1. Add a new stateless functional component in List.js above List. Call the component Task. Remember that a component is implemented as a function.


```
JS List.js 1 ●
src > components > JS List.js > ...
1 function Task(props) {
2   return (
3
4   );
5 }
6
```

You may see a red line underneath the closing bracket to indicate an error in your code. This is normal as a return function require something here. We will fix this error soon.

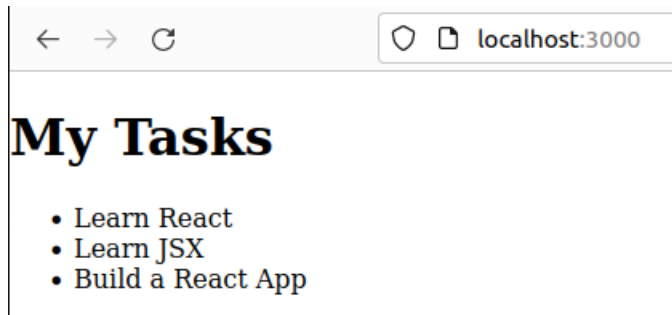
2. We want the Task component to render as a `` tag. The text inside the `` tag should depend on the value of the "description" prop. Reusing the List component code should help here.
3. Inside the List component change the three hard coded `` elements to Task components that have the appropriate description strings passed in as the description prop. Recall that React components are used like normal HTML tags in JSX, and that props are passed in like HTML attributes (ie `<Task propName1="propValue1" />`).

```
JS List.js M X JS App.js
src > components > JS List.js > ...
1 function Task(props) {
2   return (
3     <li>{props.description}</li>
4   );
5 }
6
7 function List(props) {
8   return (
9     <div>
10       <h1>{ props.heading }</h1>
11       <ul>
12         <Task description="Learn React"/>
13         <li>Learn JSX</li>
14         <li>Build a React App</li>
15       </ul>
16     </div>
17   );
18 }
19
20 export default List;
21
```

```
JS List.js M X JS App.js
src > components > JS List.js > ...
1 function Task(props)
2   return (
3     <li>{props.description}</li>
4   );
5 }
6
7 function List(props) {
8   return (
9     <div>
10       <h1>{ props.heading }</h1>
11       <ul>
12         <Task description="Learn React" />
13         <li>Learn JSX</li>
14         <li>Build a React App</li>
15       </ul>
16     </div>
17   );
18 }
19
20 export default List;
21
```

Now you see that the content of the list is considered as a dynamic component, created using the “Task” function.

4. Now test the program by reloading the web page. If you followed the steps, and the output still appears as before, then your code is probably correct.



5. Using the same principle. Replace all hard-coded list with the Task component.

```
JS List.js M X JS App.js
src > components > JS List.js > ...
1  function Task(props) {
2      return (
3          <li>{props.description}</li>
4      );
5  }
6
7  function List(props) {
8      return (
9          <div>
10             <h1>{ props.heading }</h1>
11             <ul>
12                 <Task description="Learn React"/>
13                 <Task description="Learn JSX"/>
14                 <Task description="Build a React App"/>
15             </ul>
16         </div>
17     );
18 }
19
20 export default List;
21
```

6. Now, let's add a checkbox on the list item, as an indication whether a task has been done or not. By default, the task has been completed. You can use a "checkbox" object in the Task component.

```
JS List.js M X JS App.js
src > components > JS List.js > ...
1 function Task(props) {
2   return (
3     <li>{props.description}<input type="checkbox" checked="false" /></li>
4   );
5 }
6
7 function List(props) {
8   return (
9     <div>
10      <h1>{ props.heading }</h1>
11      <ul>
12        <Task description="Learn React"/>
13        <Task description="Learn JSX"/>
14        <Task description="Build a React App"/>
15      </ul>
16    </div>
17  );
18 }
19
20 export default List;
21
```

My Tasks

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

2.2. TASKS AS PLAIN OLD JAVASCRIPT OBJECTS

1. Now let's really do it properly. Go to the App.js file and then create an array of objects. Each object will be a task item. Each object will have three properties, id, description, and completed. Write the following code inside the Root function, at the top.

```
const tasks = [
  { description: 'Learn React', completed: true },
  { description: 'Learn JSX', completed: false },
  { description: 'Build a React App', completed: false }
];
```

2. Pass the above tasks array into the List component via a property called tasks. Don't forget to use curly braces, since the property value is not a text string.

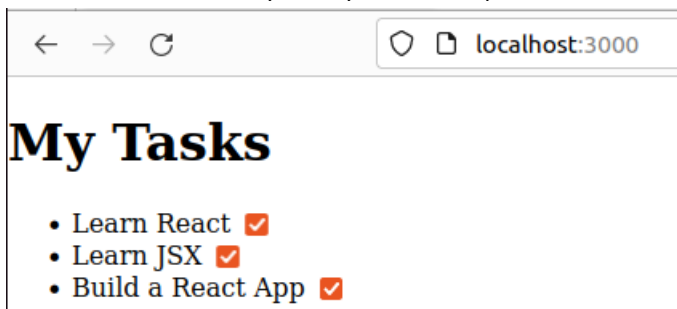
```
JS List.js M JS App.js M X
src > JS App.js > ...
1 import List from './components/List';
2 import './App.css';
3
4 const tasks = [
5   {description: 'Learn React', completed:true},
6   {description: 'Learn JSX', completed:false},
7   {description: 'Build a React App', completed:false}
8 ];
9
10 function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

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

3. Now go to the List.js and use the tasks array from the App.js that was passed from the Root component.

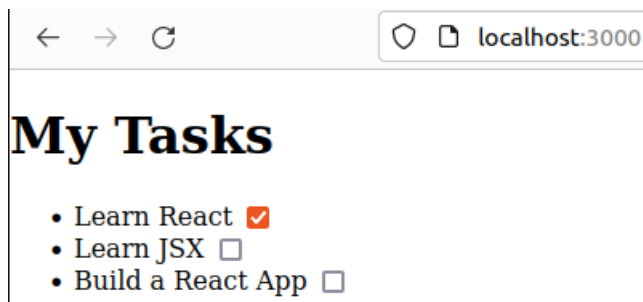
```
JS List.js M • JS App.js M
src > components > JS List.js > ...
1 function Task(props) {
2   return (
3     <li>{props.description} <input type="checkbox" checked="false" /></li>
4   );
5 }
6
7 function List(props) {
8   return (
9     <div>
10      <h1>{ props.heading }</h1>
11      <ul>
12        <Task description={props.tasks[0].description}/>
13        <Task description={props.tasks[1].description}/>
14        <Task description={props.tasks[2].description}/>
15      </ul>
16    </div>
17  );
18 }
19
20
21 export default List;
22
```

Check the browser, and you may see the output like this.



4. You can see that the completion status is hardcoded. Now, we want to make sure that the completion status is loaded from the initialisation array in App.js. Pass the completion status through the property of Task, named completed.

```
JS List.js M X JS App.js M
src > components > JS List.js > ...
1 function Task(props) {
2   return (
3     <li>{props.description} <input type="checkbox" checked={props.completed} /></li>
4   );
5 }
6
7 function List(props) {
8   return (
9     <div>
10      <h1>{ props.heading }</h1>
11      <ul>
12        <Task description={props.tasks[0].description} completed={props.tasks[0].completed}/>
13        <Task description={props.tasks[1].description} completed={props.tasks[1].completed}/>
14        <Task description={props.tasks[2].description} completed={props.tasks[2].completed}/>
15      </ul>
16    </div>
17  );
18 }
19
20
21 export default List;
```



Now, as you can see the completion status matches the initial configuration in App.js

5. Now go to List.js and simplify the code by using Loop. It's not practical to have numerous repetition like this in our code. To do this, inside the List function (at the top) create an array of Task components using the tasks prop that was passed in from the Root component. In this example, we named it "taskArray". Hint: use a for loop to go through each element of tasks, create a Task component for that element, and add that component to a new array (the task component array).
6. Now replace the three Task components in the JSX with the task component array you have created. Ensure that you enclose the array in {}.

```

JS List.js M X JS App.js M
src > components > JS List.js > Task
1 function Task(props) {
2   return (
3     <li>{props.description} <input type="checkbox" checked={props.completed} /></li>
4   );
5 }
6
7 function List(props) {
8   var taskArray = [];
9   for (let i=0; i<props.tasks.length; i++)
10    { let desc = props.tasks[i].description;
11      let stat= props.tasks[i].completed;
12      taskArray.push(<Task description={desc} completed={stat}/>)
13    }
14
15   return (
16     <div>
17       <h1>{ props.heading }</h1>
18       <ul>
19         {taskArray}
20       </ul>
21     </div>
22   );
23 }
24
25 export default List;

```

7. Finally let's make use of the "completed" property that has been passed into the Task component. Copy the following code inside the Task function, just before the closing tag:

```
<input type="checkbox" checked={props.completed} readOnly />
```

8. The above checkbox is readOnly and therefore designed only to show you the completed status, not change it.

```
JS List.js M X JS App.js M
src > components > JS List.js > ...
1 function Task(props) {
2   return (
3     <li>{props.description} <input type="checkbox" checked={props.completed} readonly/></li>
4   );
5 }
6
7 function List(props) {
8   var taskArray = [];
9   for (let i=0;i<props.tasks.length;i++)
10    { let desc = props.tasks[i].description;
11      let stat= props.tasks[i].completed;
12      taskArray.push(<Task description={desc} completed={stat}/>)
13    }
14
15   return (
16     <div>
17       <h1>{ props.heading }</h1>
18       <ul>
19         {taskArray}
20       </ul>
21     </div>
22   );
23 }
24
25 export default List;
```

← → ↻ 🔒 localhost:3000

My Tasks

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

2.3. USING THE MAP FUNCTION

Now let's make our code look really nice by removing the for loop in List.

1. Delete the for loop and array we added inside List earlier.
2. Now replace the part of the JSX with the following:

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



```
return (
  <div>
    <h1>{ props.heading }</h1>
    <ul>
      { /* { tasks } */
      { props.tasks.map(task =>
        <Task
          description={task.description}
          completed={task.completed} /> ) }
      }
    </ul>
  </div>
);

export default List;
```

← → ↻ 🔒 localhost:3000

My Tasks

- Learn React ☒
 - Learn JSX ☐
 - Build a React App ☐
3. The above code takes the entire List array and then converts each element into a Task component, which is then placed into a new array of components. This array of components is used to provide the items within the tag.

Congratulations, you've made a static React app!