## **HANDLE INPUTS AND RENDER LISTS**

### **Introduction**

Before we go any further, we’ll dive into an assignment that will test our knowledge thus far. You should attempt to do the assignment on your own first. You can then proceed further along in this lesson to see the walkthrough of how we approached it. Don’t be discouraged if you find yourself stuck. The previous material should equip you to solve or search for the right things. However, once you’ve given it an honest effort (you’d only be cheating yourself if you didn’t!), continue with the lesson to achieve enlightenment on how things can be done.

If you feel uneasy about what we’ve learned so far, go back and review the concepts of state and props from the previous lessons.

### **Learning Outcomes**

* Understanding how to render lists in React
* Understanding how to handle input fields and forms in React.

### **Assignment**

### **Do-It-Yourself Guide**

Our application will be made of two components, App and Overview. Your application should render an input field and a submit button. With the submit button, you can add the content from your input to a “tasks array” that is managed in state. (We will use class components for this example because we haven’t introduced hooks in this section yet). Finally, for each task in the tasks array, an HTML list element should be rendered.

1. Run npx create-react-app task-app, cd into your project and open it. You can delete everything in the return statement of the App component and just return an empty div. You can also delete all of the boilerplate create-react-app provides and just leave index.js and App.js in the src directory. Just make sure to clean up the import statements and the serviceWorker in the two remaining files. If you aren’t familiar with what code or files we are referring to, go back to the first lesson of this section.
2. Create a components folder in your src directory and create a file for your component called Overview.js. Overview.js should just render our tasks, while App.js is going to handle the input field with the logic.
3. With the intended functionality explained, it’s your turn to implement this React app. You can do it. You are not required to style this assignment unless you wish to, the focus is on using event handlers and dealing with forms with state.
4. Quick tip: Use the JavaScript function map to map over your tasks array. You will need to provide a unique key to each item (read the warning, you’ll know it when you see it in the console!). And there is a difference between handling input fields in plain JavaScript and in React. If you face a problem with it, attempt to figure it out on your own using tools like the documentation, StackOverflow, Google, or experimenting. But fear not, we will again provide an overview of our solution below.

### **Detailed Guide**

1. Let’s get started! First, run npx create-react-app task-app in your terminal, and open the project in your text editor.
2. Delete all files in the src directory and just leave index.js and App.js.
3. Open your App.js file in your src directory and make sure it looks like this.

// App.js import React, { Component } from "react"; class App extends Component { render() { return <div>Hello World</div>; } } export default App;

1. Make sure to clean the index.js as well. It should look something similar to this:

// Index.js import React from "react"; import ReactDOM from "react-dom"; import App from "./App"; ReactDOM.render( <React.StrictMode> <App /> </React.StrictMode>, document.getElementById("root") );

1. We may style the application using “vanilla” CSS to make it look a little bit nicer. At this point, you should be familiar with the basics of CSS from the previous material. If you intend on styling the application, we highly recommend that you use your own styling instead of using CSS frameworks that are beyond the current scope of the curriculum. Please refer to both [MDN](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics) and [React](https://reactjs.org/docs/faq-styling.html) documentation if you wish to know more about how to style React applications with CSS. If you are following along with this tutorial and do not wish to style the application, you can skip this step.
2. Go back to your src directory and create a new folder called components with a file named Overview.js. This and our App.js file will be the main parts of the project. In Overview.js, we will display all our tasks, while the App component in App.js will contain all the logic and manage state. Don’t forget to capitalize the names of your components. It doesn’t change their functionality, but it is a widely accepted “best practice”.
3. Finally, let’s write some code. To begin, in our App.js file, our class component should look like this.

// App.js import React, { Component } from "react"; class App extends Component { constructor() { super(); this.state = { task: { text: ''}, tasks: [], }; } render() { const { task, tasks } = this.state; return ( <div> <form> <label htmlFor="taskInput">Enter task</label> <input type="text" id="taskInput"/> <button type="submit"> Add Task </button> </form> </div> ); } } export default App;

We created the skeleton of our component. First, we imported React and Component from “react”, then we initialized the constructor. In the constructor, we defined state with:

this.state = { task: { text: '' }, tasks: [], };

We assigned task to an object and task.text to an empty string, this will be the state handling what we type in our input field. And tasks will initially be set to an empty array. Later, we will include all of our tasks here.

Also, inside the render function, we destructured our state in order to make our code look cleaner when using it.

After that, we render a form element with an input and a button element.

Now, let’s have a look at what our application looks like. Run npm start in your terminal to open up the application in the browser. You should now see an input field with a label and a submit button. When you click the button, nothing happens and the page only refreshes.

Let’s add some functionality to it. Go back to your App.js component and add the following two functions. Make sure to add those functions between your constructor and the render method.

handleChange = (e) => { this.setState({ task : { text: e.target.value, } }); }; onSubmitTask = (e) => { e.preventDefault(); this.setState({ tasks: this.state.tasks.concat(this.state.task), task: { text: '' }, }); };

Naturally, if we do not invoke those functions nothing will change in our application. So let’s call them. The handleChange function will be our onChange handler for our input field. It sets the current task in state to whatever we type in our input field. The onSubmitTask function will be our onSubmit handler for our form element. The onSubmit handler of the form should be invoked by clicking the button.

In the onSubmitTask function, we first call e.preventDefault() because we don’t want the default behavior of refreshing the form anytime we submit it. After that, we modify state.

The following line does the magic.

tasks: this.state.tasks.concat(this.state.task),

It adds the task (whatever is in our input field when we submit the form) to our tasks array. Later we can map over this array to display all the tasks we submitted. Make sure that you DON’T directly assign state. That is also the reason we don’t use the push method here. It would give us an error.

After that, we set task in state to the initial object with task.text as an empty string, because we want our input field to be empty, in order to add another task.

We still haven’t invoked those functions yet, so let’s do that.

In your App.js component in your render method, add an onChange handler to your input element like so:

<input onChange={this.handleChange} value={task.text} type="text" id="taskInput" />

Notice that we also have to specify the value attribute for React input elements. In this case we want the value of the input field to be what we saved in our task object in state under the text property.

And also add the onSubmitTask function to our form element like so:

<form onSubmit={this.onSubmitTask}> {/\* Leave all your code. Just add the onSubmit handler to the form element, or as an onClick handler to the submit button, as you prefer \*/} </form>

If you add an onSubmit handler to the form, your button must be of type="submit", otherwise it won’t work. Alternatively, you can add an onClick event to the button which calls the onSubmitTask function.

Great, if you run your application now with npm start (or refresh the browser if it’s still running), you will still see no changes, except that the page doesn’t refresh when you submit something. That’s because we haven’t displayed anything yet. Let’s do that now.

Go to your Overview.js file in the components folder and add the following code:

// Overview.js import React from "react"; const Overview = (props) => { const { tasks } = props; return ( <ul> {tasks.map((task) => { return <li>{task.text}</li>; })} </ul> ); }; export default Overview;

It takes the tasks from the props and maps over it. For each task it will then display a li element with the content of tasks. When checking out the application in the browser we can see we received an error message which says that a unique key is required. React always requires you to add a unique key to each element when you map over a list. In real world projects you often use database ids as unique keys, however in this project we are not using a database, so let’s install a package that provides us with unique ids.

Run npm install uniqid in your project folder. Uniqid is a package which creates unique ids based on the current time, the process and the machine name. Once this is done, we just have to include it like this:

// App.js import React, { Component } from "react"; import uniqid from "uniqid"; class App extends Component { constructor() { super(); this.state = { task: { text: '', id: uniqid() }, tasks: [], }; } handleChange = (e) => { this.setState({ task: { text: e.target.value, id: this.state.task.id, }, }); }; onSubmitTask = (e) => { e.preventDefault(); this.setState({ tasks: this.state.tasks.concat(this.state.task), task: { text: '', id: uniqid() }, }); };

// Overview.js import React from "react"; const Overview = (props) => { const { tasks } = props; return ( <ul> {tasks.map((task) => { return <li key={task.id}>{task.text}</li>; })} </ul> ); }; export default Overview;

Almost done, the only thing we need to do is import our Overview component to our App.js file and add it in our render method, while passing down the tasks array as props.

Add this line to the top of your App.js file, right below where we import React.

import Overview from "./components/Overview";

And then add the Overview component to your render method in App.js. Add this line of code right before the last closing div, and right after the closing form tag in App.js.

<Overview tasks={tasks} />

Here we go, run npm start (or refresh) one last time. If you’ve done everything right, you should now be able to type a task into the input field and click submit to display it right below the input field. Feel free to play around a little bit and maybe change or style it as you like.

Your finished files should look like this:

// App.js import React, { Component } from "react"; import Overview from "./components/Overview"; import uniqid from "uniqid"; class App extends Component { constructor() { super(); this.state = { task: { text: '', id: uniqid() }, tasks: [], }; } handleChange = (e) => { this.setState({ task: { text: e.target.value, id: this.state.task.id, }, }); }; onSubmitTask = (e) => { e.preventDefault(); this.setState({ tasks: this.state.tasks.concat(this.state.task), task: { text: '', id: uniqid() }, }); }; render() { const { task, tasks } = this.state; return ( <div> <form onSubmit={this.onSubmitTask}> <label htmlFor="taskInput">Enter task</label> <input onChange={this.handleChange} value={task.text} type="text" id="taskInput" /> <button type="submit">Add Task</button> </form> <Overview tasks={tasks} /> </div> ); } } export default App;

// Overview.js import React from "react"; const Overview = (props) => { const { tasks } = props; return ( <ul> {tasks.map((task) => { return <li key={task.id}>{task.text}</li>; })} </ul> ); }; export default Overview;

### **Optional Tasks / Ideas to play around**

Here are a few optional tasks for you to practice. Try them out, if you can’t solve them, continue with the curriculum and make sure to come back later to give them another try and see how you advanced.

### **Easy**

1. Instead of displaying unordered list items, manage the amount of tasks in state and let each task display its number. Yes, you could also do that with a simple ordered list, but where’s the fun in that? Try using state.
2. Implement a delete button for each task. The delete button should remove the specific task from the state array. Don’t forget to never directly assign state. If you want you can use [Font Awesome](https://fontawesome.com/) for the icon. Tip: You can remove each task by its id, but for that you have to save the task id together with the task title.

### **Hard:**

1. Implement an edit button for each task. When you press the edit button, this specific task should become changeable, and the edit button should change to a resubmit button.

### **Additional Resources**

1. The sections on [Handling Events](https://reactjs.org/docs/handling-events.html) and [Conditional Rendering](https://reactjs.org/docs/conditional-rendering.html) from the React documentation are helpful if you are stuck!