Lab: Todo List Application

Skills Network

Estimated time needed: 40 minutes

What you will learn

In this lab, you will learn how to use React functional components and the useState hook to make a simple to-do list application. You will learn how to manage state using the useState hook. The lab includes the basics of managing React states, taking user input, and showing dynamic lists of things.

Learning objectives

After completing this lab, you will be able to:

- Create a simple Todo List application using React and add headings and associated lists to organize tasks.

- Manage the todo items and delete specific headings and lists, provide basic CRUD (Create, Read, Update, Delete) functionality.

- Use React's useState hook to manage the state of the Todo List, including the list of headings and associated lists.

- Use React's useState hook also to render the values dynamically.

Prerequisites

- Basic knowledge of HTML
- Intermediate knowledge of JavaScript
- Basic knowledge of React functional component and state management using useState hook

Setting up the environment

1. From the menu on top of the lab, click the **Terminal** tab at the top-right of the window shown at number 1 in the given screenshot, and then click **New Terminal** as shown at number 2.
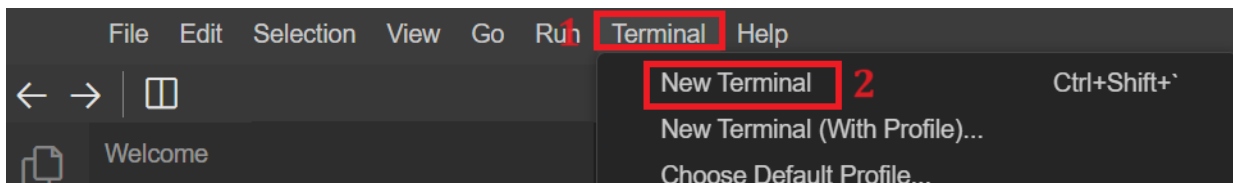
2. Now, write the following command in the terminal to clone the boiler template for this React application and hit Enter.

```
git clone https://github.com/ibm-developer-skills-network/todo_list.git
```

3. The above command will create a folder, "todo_list" under the "Project" folder. You can see the structure in the screenshot.
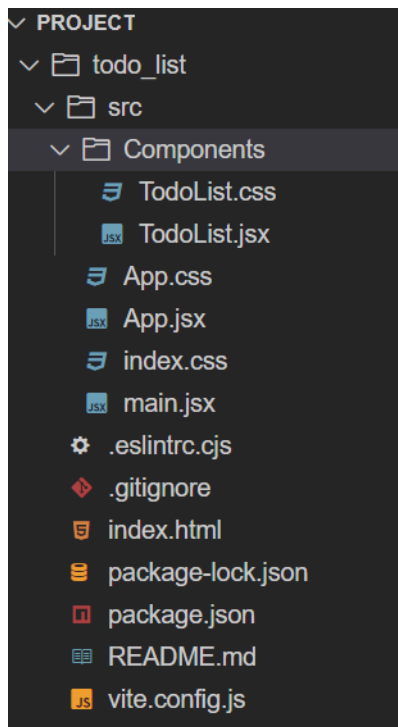
4. Next, you need to run the React application by entering the folder name in the terminal using the given command. This action will set your terminal path to run the React application within the **todo_list** folder.

```
cd todo_list
```

5. To ensure the code you have cloned is working correctly, you need to perform the following steps:

   ○ Write the given command in the terminal and hit Enter. This command will install all the necessary packages to execute the application.

   ```
   npm install
   ```

   ○ Then execute the following command to run the application and this will provide you with port number 4173.

   ```
   npm run preview
   ```

6. To view your React application, click the Skills Network icon on the left (refer to number 1). This action will open the **SKILLS NETWORK TOOLBOX**. Next, click **Launch Application** (refer to number 2). Enter the port number **4173** in **Application Port** (refer to number 3) and click [↗].
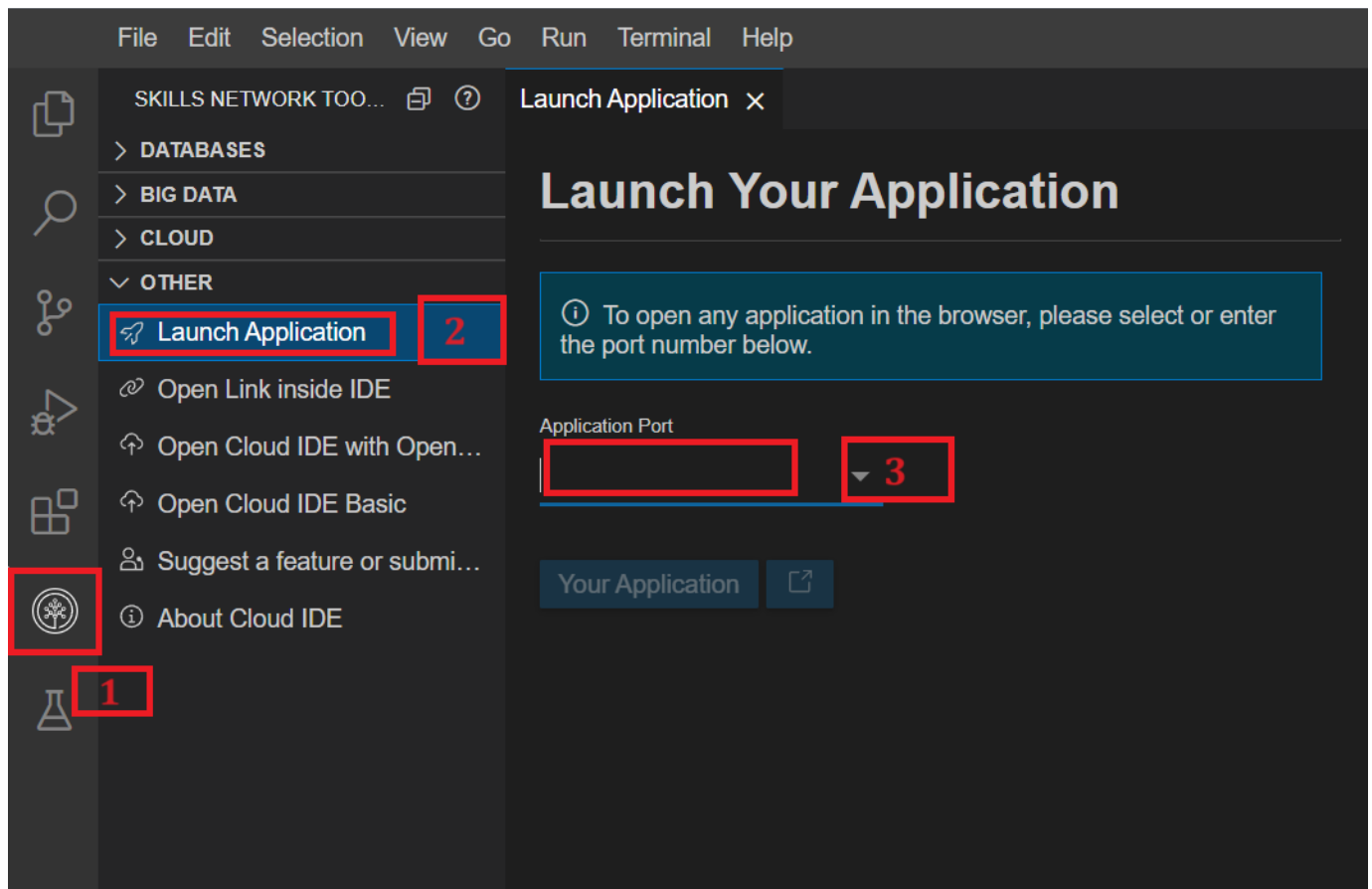
7. The output will display as shown in the given screenshot.
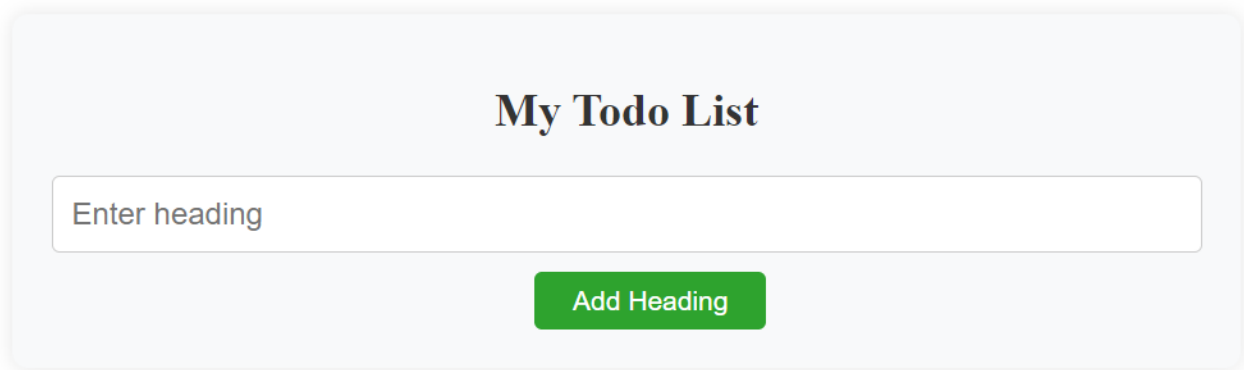
Setting the initial state

1. Next, navigate to the **TodoList.jsx** file located in the **Components** folder of the **src** directory in your cloned **todo_list** folder.

2. The basic structure of this component will be as shown in the screenshot.

```
import React, { useState } from 'react';
import './TodoList.css';

const TodoList = () => {
  return (
    <>
      <div className="todo-container">
        <h1 className="title">My Todo List</h1>
        <div className="input-container">
          <input
            type="text"
            className="heading-input"
            placeholder="Enter heading"


          />
          <button className="add-list-button">Add Heading</button>
        </div>
      </div>
      <div className="todo_main">

      </div>
    </>
  );
};

export default TodoList;
```

- The above code represents the basic structure of a todo list component in a React application. It includes a container for the todo list `<div className="todo-container">` with a title "My Todo List" and an input container `<div className="input-container">` containing an input field for entering todo headings and a button labeled "Add Heading".

- There is a main section `<div className="todo_main">` intended to display the todo items. You need to apply the functionality for handling user input, managing todo items, and dynamically rendering them, which would be implemented using React state and event handling.

- You need to initialize the following three states:

  - `todos`: To represent an array of todo items. Initialize it with an empty array `[]`, indicating that there are no todo items initially.
  - `headingInput`: To represent the value entered by user into an input field for adding a new heading for a todo item. Initialize it as an empty string `''`.
  - `listInputs`: Initialize listInputs as an empty object `{}`. This state will hold the value of input fields for each todo item individually.

    ```
    const [todos, setTodos] = useState([]);
    const [headingInput, setHeadingInput] = useState('');
    const [listInputs, setListInputs] = useState({});
    ```

Note: Include above code before `return` of component.

Implement Add Heading Functionality:

Add Heading Functionality for Todo List

- At first, for the similar todo tasks you will add the specific heading such as `Grocery Items`. Under this heading, list todo items will come like milk, butter, and bread.

- For this one `input` tag has already been provided to you with class name `heading-input`. To get the heading from this input box, you need to create a function named as **handleAddTodo** that will be triggered on clicking the **Add Heading** button.

- Check if `headingInput` is empty. If not empty, create a new todo object with the heading from `headingInput` and an empty array for lists.

- Then append this todo object to the todo array and clear `headingInput` by setting it to an empty string.

```
const handleAddTodo = () => {
  if (headingInput.trim() !== '') {
    setTodos([...todos, { heading: headingInput, lists: [] }]);
    setHeadingInput('');
  }
};
```

In the above code:

- `const handleAddTodo = () => { ... }`: Declares a constant named `handleAddTodo` and assigns it an arrow function.

- `if (headingInput.trim() !== '') { ... }`: Checks if the `headingInput` variable, a piece of text input from the user, is empty after trimming any whitespace characters from the beginning and end. This condition ensures that the user has entered some content before proceeding.

- `setTodos([...todos, { heading: headingInput, lists: [] }]);`: If the condition in the if statement is met, this line updates the state variable `todos`. It spreads the existing todos array (todos) into a new array using the spread syntax (…todos) and appends a new object to it. The new object contains a `heading` property set to the value of `headingInput` and a `lists` property initialized as an empty array.

- `setHeadingInput('');`: After adding a new todo item, this line clears the `headingInput` state variable, resetting the text input field for the user to enter a new todo item heading.

Note: Include above code after initializing the three states.

- To implement the "Add Todo" functionality, changes need to be made inside the div tag with classname `input-container` where the input field and the button for adding the heading are located.
- The input field and the button element in the div tag need to be updated to incorporate the functionality to add a new todo item on button click.

```
<div className="input-container">{/* Input field to enter a new heading */}
  <input
    type="text"
    className="heading-input"// CSS class for styling
    placeholder="Enter heading"// Text shown when input is empty
    value={headingInput}
    onChange={(e) => {setHeadingInput(e.target.value);}} // Add onChange event handler to update headingInput state
  />
  {/* Button to add the entered heading to the todo list */}
  <button className="add-list-button" onClick={handleAddTodo}>Add Heading</button>
</div>
```

In the input element:

- `value` attribute is set to {headingInput}, which binds the value of the 'input' field to the 'headingInput' state variable.

- onChange: This is an event handler in React that triggers when the value of an input element changes.

  - (e) => {…}: This is an arrow function, which is a concise way to define a function in JavaScript. It takes an event (e) as an argument, which represents the event that triggered the handler.

  - {setHeadingInput(e.target.value);}: This is the body of the arrow function. `setHeadingInput` is a function that updates the state variable `headingInput` with the value entered in the input field. `e.target.value` retrieves the current value of the input element that triggered the event, and `setHeadingInput` is called with this value to update the state.

In the button element:

- `onClick` event handler is added to trigger the `handleAddTodo` function on button click. This function adds a new todo item under the entered heading.

Display Todo Heading

- To display the heading of each todo item, you need to iterate over the todos array and render the heading within the JSX.
- You need to update the JSX portion where each todo item is rendered inside div with class name `todo-card` to display the heading.
- You need to include this code within the div tag with class name `todo_main`.

```
{todos.map((todo, index) => ( // Iterate over each todo item in the todos array
    <div key={index} className="todo-card">
```

```
        <div className="heading_todo">
      {/* Display the heading text of the current todo item */}
          <h3>{todo.heading}</h3> {/* Display the heading here */}
      {/* Button to delete the current heading by passing its index */}
          <button className="delete-button-heading" onClick={() => handleDeleteTodo(index)}>Delete Heading </button>
        </div>
      </div>
    ))}
```

- In the code:

  - `Mapping over Todos Array: todos.map((todo, index) => ...):` Maps over the todos array, which contains todo items.The map() function executes the specified function for each todo item in the array.
  - `Rendering Todo Item: <div key={index} className="todo-card"> ... </div>:` For each todo item, a div element with the class `todo-card` is rendered. The key attribute is set to index to identify each to-do item within the list uniquely.
  - `Displaying Todo Heading: <h3>{todo.heading}</h3>:` Within each todo-card div, the heading of the current todo item is displayed using an `<h3>` element. The heading text is retrieved from the heading property of the todo object.
  - `Deleting Todo Item: <button className="delete-button-heading" onClick={() => handleDeleteTodo(index)}>Delete Heading </button>:` Each todo item is accompanied by a "Delete Heading" button. When clicked, this button triggers the `handleDeleteTodo` function, passing the index of the current todo item as an argument. The index allows the function to identify and delete the corresponding todo item from the todos array.

- Now save the `TodoList.jsx` component and re-run the application to check the working of the code. If your application is already open then just refresh the webpage.

- Now enter the heading such as `Grocery Item` in the input box and click on **Add Heading** button. You will look output similar to given screenshot.

Implement Add List Functionality

In this task you will now include the todo list items under specific headings. To do so follow the given steps.

Add Form in JSX

- To add list and display list you need to include an input box for user to enter the list name and a button to add the list.

- The list should display only after including the adding section. Therefore, you need to include one input box with one button.

- Include the following code after the div tag with the class name `heading_todo` as child of div with class name `todo-card` so that it will appear only after adding the heading.

```
        <div className='add_list'>
        {/* Input field for adding a new item under a specific heading */}
                    <input
                      type="text"
                      className="list-input"
                      placeholder="Add List"
                      value={listInputs[index] || ''}// Use the value from listInputs array based on the current heading index
                      onChange={(e) => handleListInputChange(index, e.target.value)}/>
        {/* Button to add the list item to the corresponding heading */}
                      <button className="add-list-button" onClick={() => handleAddList(index)}>Add List</button>
                  </div>
```

- The above JSX code snippet represents:

  - A form element for adding a new list item within a todo item. It consists of an input field `<input>` for entering the text of the new list item, with its value bound to the `listInput` state variable.

  - The input element has an `onChange` event handler. When the value of the input changes, the event (e) is captured, and the `handleListInputChange` function is called with the current index and the new value obtained from e.target.value. This updates the listInputs state object, ensuring that the input for each todo item's list is tracked individually.

  - In addition, there is a `<button>` labeled "Add List" that triggers the `handleAddList` function with the current index value as its parameter on click, to add the entered list item to the todo item at the specified index in the todos array.

handleAddList() Function To Add Todo Items

To implement the functionality to add list inside the heading, you need to perform the following steps:

- Create a function named as `handleAddList` that takes the index of the todo item as a parameter.

- Check if `listInputs[index]` is not empty after trimming any leading or trailing whitespace. If not empty, create a copy of the todos array.

- Push the new list item from `listInputs[index]` into the lists array of the todo item at the specified index.

- Update the state with the modified todos array and clear `listInputs[index]` by setting it to an empty string.

```
    // Function to handle adding a new list item to a specific todo heading
    const handleAddList = (index) => {
        // Check if the input for the given index is not empty or just whitespace
        if (listInputs[index] && listInputs[index].trim() !== '') {
            const newTodos = [...todos]; // Create a copy of the current todos array
            newTodos[index].lists.push(listInputs[index]); // Add the new list item to the corresponding heading's list
            setTodos(newTodos); // Update the todos state with the new list item
            setListInputs({ ...listInputs, [index]: '' }); // Clear the input field for that index
        }
    };
    // Function to update list input value for a specific heading index
    const handleListInputChange = (index, value) => {
        setListInputs({ ...listInputs, [index]: value }); // Update the listInputs state for the corresponding index
    };
```

  - `const handleAddList = (index) => { ... }`: Declares a constant named `handleAddList` and assigns it an arrow function that takes an index parameter.

  - if (listInputs[index] && listInputs[index].trim() !== '') { … }: Checks if the listInputs[index] variable, which is a piece of text input from the user for adding a new list item, is not empty after trimming any leading or trailing whitespace. This condition ensures that the user has entered some content before proceeding.

  - `const newTodos = [...todos];`: Creates a shallow copy of the todos array using the spread syntax (``…todos``). The copy is made to avoid directly mutating the state, which is a best practice in React.

  - `newTodos[index].lists.push(listInput);`: Accesses the todo item at the specified index in the newTodos array and pushes the value of listInputs[index] into its lists array. This push assumes that each todo item has a lists property, an array containing the items within that todo.

  - `setTodos(newTodos);`: After updating the newTodos array with the new list item, the setTodos function, a state updater function provided by React's useState hook, updates the state variable todos with the modified array.

  - `setListInput('');`: Finally, this function resets the listInput state variable, clearing the text input field for adding new list items.

The handleListInputChange function:

- const handleListInputChange = (index, value) => { ... }: Declares a constant named handleListInputChange and assigns it an arrow function that takes two parameters: the index of the todo item and the new value of the list input.

- setListInputs({ ...listInputs, [index]: value });: This function updates the listInputs state object with the new value for the input at the specified index, ensuring that each todo item's list input is tracked individually.

Display Todo List in JSX

- To display the list, you need to iterate todo.lists inside <ul> tag and include user input in the <li> tag.

```
<ul>
  {/* Iterate over each list item inside the current todo */}
  {todo.lists.map((list, listIndex) => (
    <li key={listIndex} className='todo_inside_list'>
      {/* Display the text content of the list item */}
      <p>{list}</p>
    </li>
  ))}
</ul>
```

- This JSX code snippet renders a list of items within a todo item. It utilizes the map function to iterate over the lists array of the todo object (representing a todo item).
- For each item in the lists array, it generates a <li> element with a unique key attribute set to listIndex to ensure proper rendering and performance optimization.
- Inside each <li> element, it displays the list item' content wrapped in a <p> element.

*Note:Include above code before <div> with class name **add_list***

- Check the output by re running the application again. At first you will see output according to given screenshot.
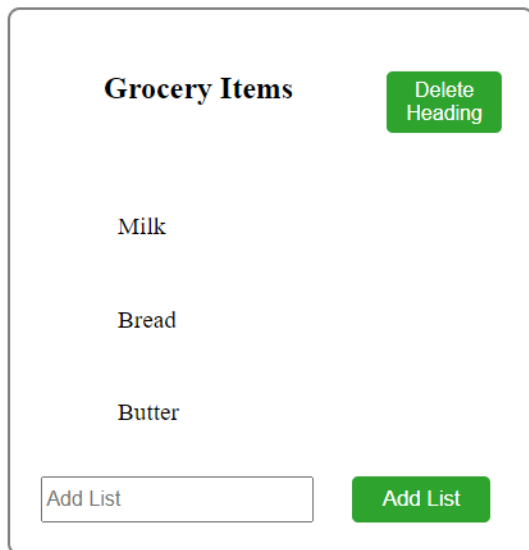

- Then write your grocery list one after another and click on **Add List** button. Then you items will display as below:

Delete Heading With Todo List

- To delete the heading section, the entire todo list will also be deleted. Suppose if a user has entered the heading as **Grocery List** and all items under this heading have been added, then the user should be able to delete the entire list from their application interface.

- To delete the list create a function with name `handleDeleteTodo` and apply logic to delete the element.

```
 const handleDeleteTodo = (index) => {
// Create a shallow copy of the current todos array
const newTodos = [...todos];
// Remove the todo at the specified index
newTodos.splice(index, 1);
// Update the state with the new array (without the deleted todo)
setTodos(newTodos);
 };
```

- The function `handleDeleteTodo` is designed to remove a todo item from the todos array at a specific index in the above code as follows:

    - `const handleDeleteTodo = (index) => { ... }`: Declares a constant named `handleDeleteTodo` that has an arrow function which takes an `index` parameter, indicating the index of the todo item to be deleted.

    - `const newTodos = [...todos];`: Creates a shallow copy of the todos array using the spread syntax (…todos). This step is crucial to avoid directly mutating the original state.

    - `newTodos.splice(index, 1);`: The splice method is called on the newTodos array to remove one element at the specified index.

    - `setTodos(newTodos);`: Finally, the setTodos function, provided by React's useState hook, is called with the updated `newTodos` array as an argument. This updates the state variable `todos` with the modified array, removing the todo item specified by the index from the UI and re-rendering the component accordingly.

Note: Include above code before the `return` of component.

- Now you need to include `onClick` event in the button with class name **delete-button-heading**

```
<button className="delete-button-heading" onClick={handleDeleteTodo}>Delete Heading</button>
```

► Click here for the solution code of TodoList.jsx

```jsx
import React, { useState } from 'react';
import './TodoList.css';
const TodoList = () => {
    // State to store all todo sections (each with a heading and associated lists)
    const [todos, setTodos] = useState([]);
    // State to manage the current heading input
    const [headingInput, setHeadingInput] = useState('');
    // State to manage each input field for the nested list items by heading index
    const [listInputs, setListInputs] = useState({});
    // Function to add a new todo heading (if input is not empty)
    const handleAddTodo = () => {
        if (headingInput.trim() !== '') {
            // Append new todo with empty list array
            setTodos([...todos, { heading: headingInput, lists: [] }]);
            setHeadingInput(''); // Clear the input field
        }
    };
    // Function to delete a todo section based on index
    const handleDeleteTodo = (index) => {
        const newTodos = [...todos];        // Create a copy of current todos
        newTodos.splice(index, 1);          // Remove the selected heading
        setTodos(newTodos);                 // Update state with modified list
    };
    // Function to add a new list item under a specific heading
    const handleAddList = (index) => {
        if (listInputs[index] && listInputs[index].trim() !== '') {
            const newTodos = [...todos];                        // Copy current todos
            newTodos[index].lists.push(listInputs[index]);      // Add list to the right section
            setTodos(newTodos);                                 // Update state
            setListInputs({ ...listInputs, [index]: '' });      // Clear list input for that section
        }
    };
    // Function to handle change in list input field for a specific section
    const handleListInputChange = (index, value) => {
        setListInputs({ ...listInputs, [index]: value }); // Track input for each heading index
    };
    return (
        <>
            {/* Input section to add a new heading */}
            <div className="todo-container">
                <h1 className="title">My Todo List</h1>
                <div className="input-container">
                    <input
                        type="text"
                        className="heading-input"
                        placeholder="Enter heading"
                        value={headingInput}
                        onChange={(e) => setHeadingInput(e.target.value)} // Update heading input value
                    />
                    <button className="add-list-button" onClick={handleAddTodo}>
                        Add Heading
                    </button>
                </div>
            </div>
            {/* Main section displaying all todos */}
            <div className="todo_main">
                {todos.map((todo, index) => (
                    <div key={index} className="todo-card">
                        <div className="heading_todo">
                            <h3>{todo.heading}</h3> {/* Display heading */}
                            <button
                                className="delete-button-heading"
                                onClick={() => handleDeleteTodo(index)}
                            >
                                Delete Heading
                            </button>
                        </div>
                        {/* Render all list items under this heading */}
                        <ul>
                            {todo.lists.map((list, listIndex) => (
                                <li key={listIndex} className='todo_inside_list'>
                                    <p>{list}</p> {/* Display individual list item */}
                                </li>
                            ))}
                        </ul>
                        {/* Input section to add list item under this heading */}
                        <div className='add_list'>
                            <input
                                type="text"
                                className="list-input"
                                placeholder="Add List"
                                value={listInputs[index] || ''} // Maintain controlled input
                                onChange={(e) => handleListInputChange(index, e.target.value)} // Update list input value
                            />
                            <button
                                className="add-list-button"
                                onClick={() => handleAddList(index)}
                            >
                                Add List
                            </button>
                        </div>
```

```
                    </div>
                ))}
            </div>
        </>
    );
};
export default TodoList;
```

Check the output

1. Stop the execution of the React application in the terminal by performing `ctrl+c` to quit.

2. Then, write the given command in the terminal and hit Enter.

   ```
   npm run preview
   ```

3. To view your React application, refresh the already opened webpage for the React application on your browser. If it is not open, then click the Skills Network icon on the left panel. This action will open the "SKILLS NETWORK TOOLBOX." Next, select "Launch Application". Enter the port number **4173** in "Application Port" and click .

4. The output will display as per the given screenshot after adding multiple headings and multiple todo lists inside heading.

Note- To see the latest changes, you need to execute `npm run preview` again in the terminal.

Congratulations! You have created a ToDo list React application!

Conclusion

- In this lab, you have learned how to manage state in a React functional component using the useState hook. State variables such as todos, headingInput, and listInput are used to maintain the state of the Todo List and user input fields.

- You have learned to render headings and associated lists based on the state maintained by React's useState hook.

- You have used code that implements event handling functions to add new headings and lists, and to handle changes in input fields. These functions update the state of the Todo List, triggering re-renders to reflect the changes in the UI.

- You have created a Todo List component that is well-structured, with separate sections for input fields, heading display, list display, and add list functionality. This modular structure enhances code readability and maintainability.

Author(s)

Richa Arora