Lab 3: Using Hooks to Make a Task List (with localStorage)

Step 1: Setting Up the Context and Reducer

How the Hooks will be used in this application:

`useReducer` Hook:

> Manages the state of the to-do list with actions (ADD_TODO, TOGGLE_TODO, REMOVE_TODO).
>
> Provides a dispatch function to send actions to the reducer.

`useContext` Hook:

> Accesses the to-do list state and dispatch function from the context in the TodoList component.

`useRef` Hook:

> Creates a reference to the input element to get its value and reset it after adding a new to-do.

`useEffect` Hook:

Logs the updated to-do list to the console whenever the state changes.

This breakdown should help you understand how the different parts of the application work together and how various hooks are used to manage state and effects in a functional component-based approach.

1. TodoContext.js

1. First, let's create a context for our to-do application. This context will allow us to share the to-do list state and dispatch function across different components.

```
import React from 'react';

const TodoContext = React.createContext();

export default TodoContext;
```

2. todoReducer.js

1. Next, we create a reducer function to manage the state of our to-do list. The reducer function will handle adding, toggling, and removing to-dos.

```
const todoReducer = (state, action) => {
  switch (action.type) {
    case 'ADD_TODO':
      return [...state, { id: Date.now(), text:
action.payload, completed: false }];
    case 'TOGGLE_TODO':
      return state.map(todo =>
        todo.id === action.payload ? { ...todo,
completed: !todo.completed } : todo
      );
    case 'REMOVE_TODO':
      return state.filter(todo => todo.id !==
action.payload);
    default:
      return state;
  }
};

export default todoReducer;
```

Step 2: Creating the Main Application Component

3. App.js

1. In the main application component, we will use the `useReducer` hook to manage the to-do list state and provide it to the rest of the application using the `TodoContext.Provider`.

```
import React, { useReducer, useRef, useEffect } from
'react';
import TodoContext from './TodoContext';
import todoReducer from './todoReducer';
import TodoList from './TodoList';
import './App.css';

const App = () => {
  const [state, dispatch] = useReducer(todoReducer, [], ()
=> {
    const localData = localStorage.getItem('todos');
```

```
    return localData ? JSON.parse(localData) : [];
  });

  const inputRef = useRef(null);

  const handleAddTodo = () => {
    if (inputRef.current.value) {
      dispatch({ type: 'ADD_TODO', payload:
inputRef.current.value });
      inputRef.current.value = '';
    }
  };

  useEffect(() => {
    localStorage.setItem('todos', JSON.stringify(state));
  }, [state]);

  return (
    <TodoContext.Provider value={{ state, dispatch }}>
      <div className="App">
        <h1>Todo List</h1>
        <input ref={inputRef} type="text" placeholder="Add
a new task" />
        <button onClick={handleAddTodo}>Add</button>
        <TodoList />
      </div>
    </TodoContext.Provider>
  );
};

export default App;
```

App.css:

```
.App {
  font-family: 'Arial', sans-serif;
  background-color: #f7f7f7;
  padding: 20px;
  max-width: 500px;
  margin: 40px auto;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  text-align: center;
```

```
}

/* Title styling */
h1 {
  color: #333;
  margin-bottom: 20px;
}

/* Input field styling */
input[type="text"] {
  width: 70%;
  padding: 10px;
  margin-right: 10px;
  border: 1px solid #ddd;
  border-radius: 5px;
  box-shadow: inset 0 1px 3px rgba(0, 0, 0, 0.1);
}

/* Button styling */
button {
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  background-color: #007bff;
  color: white;
  font-size: 14px;
  cursor: pointer;
  transition: background-color 0.3s ease;
}

button:hover {
  background-color: #0056b3;
}

/* List styling */
ul {
  list-style: none;
  padding: 0;
}

li {
  background-color: white;
  margin: 10px 0;
```

```
  padding: 15px;
  border-radius: 5px;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
  display: flex;
  justify-content: space-between;
  align-items: center;
}

/* Completed task styling */
li.completed {
  text-decoration: line-through;
  color: #888;
}

/* Individual button styling within list items */
li button {
  background-color: transparent;
  border: none;
  cursor: pointer;
  color: #007bff;
  margin-left: 10px;
}

li button:hover {
  color: #0056b3;
}
```

Step 3: Creating the TodoList Component

TodoList.js

The TodoList component will consume the context and render the list of to-dos. It will use the `useContext` and `useEffect` hooks.

```
import React, { useContext, useEffect } from 'react';
import TodoContext from './TodoContext';
import TodoItem from './TodoItem';

const TodoList = () => {
  // Use context to get state and dispatch
```

```
  const { state, dispatch } = useContext(TodoContext);

  // Log state changes
  useEffect(() => {
    console.log('Todo list updated:', state);
  }, [state]);

  return (
    <ul>
      {state.map(todo => (
        <TodoItem key={todo.id} todo={todo}
dispatch={dispatch} />
      ))}
    </ul>
  );
};

export default TodoList;
```

Step 4: Creating the TodoItem Component

4. TodoItem.js

1. The TodoItem component represents a single to-do item and provides buttons to toggle its completion status or remove it.

```
import React from 'react';

const TodoItem = ({ todo, dispatch }) => {
       return (
         <li style={{ textDecoration: todo.completed ?
     'line-through' : 'none' }}>
      {todo.text}
      <button onClick={() => dispatch({ type:
'TOGGLE_TODO', payload: todo.id })}>
        {todo.completed ? 'Undo' : 'Complete'}
      </button>
      <button onClick={() => dispatch({ type:
'REMOVE_TODO', payload: todo.id })}>
        Remove
      </button>
    </li>
```

```
  );
};

export default TodoItem;
```

Putting It All Together

Finally, ensure all components and context are imported and used correctly in your project.

App.js: The main entry point where the TodoContext.Provider wraps the entire application.

TodoList.js: Consumes the context to display the list of to-dos and uses useEffect to log changes.

TodoItem.js: Represents individual to-do items and handles toggling and removing to-dos.

TodoContext.js: Provides a context for sharing the to-do list state and dispatch function.

todoReducer.js: Contains the reducer function to manage the to-do list state.

Running the Application

To run the application, use the following command in your project directory:

```
npm start
```