|  |
| --- |
| **Redux-Toolkit-ToDoApp** |
| * npx create-react-app reduxtoolkit * cd reduxtoolkit * npm install @reduxjs/toolkit * npm install react-redux |
| Index.js  // index.js  import React from 'react';  import ReactDOM from 'react-dom';  import { Provider } from 'react-redux';  import App from './App';  import store from './store';  ReactDOM.render(  <Provider store={store}>  <App />  </Provider>,  document.getElementById('root')  ); |
| // store.js  import { configureStore } from '@reduxjs/toolkit';  import todoReducer from './todoSlice';  const store = configureStore({  reducer: {  todos: todoReducer,  },  });  export default store;  // todoSlice.js  import { createSlice } from '@reduxjs/toolkit';  const initialState = [];  const todoSlice = createSlice({    name: 'todos',  initialState,  reducers: {  addTodo: (state, action) => {  state.push(action.payload);  },  toggleTodo: (state, action) => {  const todo = state.find(todo => todo.id === action.payload);  if (todo) {  todo.completed = !todo.completed;  }  },  deleteTodo: (state, action) => {  return state.filter(todo => todo.id !== action.payload);  }  }  });  export const { addTodo, toggleTodo, deleteTodo } = todoSlice.actions;  export default todoSlice.reducer;  // App.js  import React, { useState } from 'react';  import { useDispatch } from 'react-redux';  import { addTodo } from './todoSlice';  import TodoList from './TodoList';  const App = () => {  // const todos = useSelector(state => state.todos);  const dispatch = useDispatch();  const [text, setText] = useState('');  const handleAddTodo = () => {  if (text.trim() !== '') {  dispatch(addTodo({id: Date.now(),text,completed: false}));  setText('');  }  };  return (  <div>  <input  type="text"  value={text}  onChange={e => setText(e.target.value)}  placeholder="Enter a new todo"  />  <button onClick={handleAddTodo}>Add</button>  <TodoList />  </div>  );  };  export default App;  // TodoList.js  import React from 'react';  import { useSelector, useDispatch } from 'react-redux';  import { deleteTodo } from './todoSlice';  import TodoItem from './TodoItem';  const TodoList = () => {  const todos = useSelector(state => state.todos);  const dispatch = useDispatch();  if (!todos || todos.length === 0) {  return <div>No todos to display</div>;  }  const handleDeleteTodo = id => {  dispatch(deleteTodo(id));  };  return (  <div>  {todos.map(todo => (  <div key={todo.id}>  <TodoItem todo={todo} />  <button onClick={() => handleDeleteTodo(todo.id)}>Delete</button>  </div>  ))}  </div>  );  };  export default TodoList;  // TodoItem.js  import React from 'react';  const TodoItem = ({ todo }) => {  return (  <div>  <input type="checkbox" checked={todo.completed} readOnly />  <span>{todo.text}</span>  </div>  );  };  export default TodoItem; |
| The **addTodo** function you provided is a reducer function, not the action creator itself. Reducer functions in Redux expect two parameters: **state** and **action**. Let's break down how the **dispatch(addTodo({id: Date.now(),text,completed: false}))** call works in conjunction with the **addTodo** reducer function:   1. **Dispatching the Action:**    * When you call **dispatch(addTodo({id: Date.now(),text,completed: false}))**, you are dispatching an action to the Redux store.    * The **dispatch** function is provided by Redux, and it's used to send actions to the store. 2. **Action Creation:**    * The argument passed to **dispatch** is the result of calling the **addTodo** action creator function. However, in this case, it's not explicitly defined as an action creator function in the provided code. Instead, it's directly passing an object with **{id: Date.now(), text, completed: false}**. 3. **Action Object:**    * The object **{id: Date.now(), text, completed: false}** serves as the payload for the action. It contains data necessary for the action to perform its task, such as the **id**, **text**, and **completed** properties for creating a new todo item. 4. **Reducer Function:**    * When the action is dispatched, it is passed to all of the reducer functions in the application.    * The **addTodo** reducer function is one of these reducer functions. It's responsible for updating the state based on the dispatched action. 5. **Handling the Action in the Reducer:**    * In the **addTodo** reducer function, it takes two parameters: **state** and **action**.    * **state** represents the current state of the application (specifically, the state related to todos in this case).    * **action** represents the dispatched action object.    * When the **addTodo** action is dispatched, the **addTodo** reducer function is executed.    * Inside the **addTodo** reducer function, the **action.payload**, which contains the new todo item, is pushed into the **state** array.   So, even though the **addTodo** reducer function expects **state** and **action**, the **dispatch** function is responsible for passing the action object (containing the payload) to the reducer function when it's dispatched.  Top of Form  Bottom of Form |