



```
now give crud implementation forredux toolkit in ts
import { useState, useCallback } from "react";
import axios from "axios";

const useApi = (baseUrl) => {
  const [data, setData] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const fetchData = useCallback(async (endpoint, method = "GET",
    body = null, headers = {}) => {
      setLoading(true);
      setError(null);

      try {
        const response = await axios({
          url: `${baseUrl}${endpoint}`,
          method,
          data: body,
          headers,
        });
        setData(response.data);
      } catch (err) {
        setError(err.message || "An error occurred");
      } finally {
        setLoading(false);
      }
    }, [baseUrl]);

  return { data, loading, error, fetchData };
};

export default useApi;
```



To implement CRUD operations using Redux Toolkit in TypeScript with the custom `useApi` hook you provided, follow the steps below. We'll integrate Redux Toolkit for state management while leveraging `useApi` for API calls.

### Step 1: Install Dependencies

Install Redux Toolkit and React-Redux if not already installed:

```
bash
```

[Copy code](#)

```
npm install @reduxjs/toolkit react-redux
```

### Step 2: Create a Slice for the To-Do Items

Create a `todosSlice.ts` file for handling the state of To-Do items.

```
typescript
```

[Copy code](#)

```
import { createSlice, PayloadAction, createAsyncThunk } from "@reduxjs/toolkit";
import { RootState } from "../store";
import useApi from "../../hooks/useApi";

const { fetchData } = useApi("https://api.example.com");

// Define the To-Do interface
export interface Todo {
  id: string;
  title: string;
  completed: boolean;
}

// Define the slice state
interface TodoState {
  todos: Todo[];
  loading: boolean;
  error: string | null;
}

// Initial state
const initialState: TodoState = {
  todos: [],
  loading: false,
  error: null,
};

// Thunks for async CRUD operations
```

```

// thanks for async CRUD operations
export const fetchTodos = createAsyncThunk("todos/fetchTodos", async () => {
  const response = await fetchData("/todos");
  return response;
});

export const addTodo = createAsyncThunk(
  "todos/addTodo",
  async (newTodo: Omit<Todo, "id">) => {
    const response = await fetchData("/todos", "POST", newTodo);
    return response;
  }
);

export const updateTodo = createAsyncThunk(
  "todos/updateTodo",
  async (updatedTodo: Todo) => {
    const response = await fetchData(`/todos/${updatedTodo.id}`, "PUT", updatedTodo);
    return response;
  }
);

export const deleteTodo = createAsyncThunk("todos/deleteTodo", async (id: string) => {
  await fetchData(`/todos/${id}`, "DELETE");
  return id;
});

// Create the slice
const todosSlice = createSlice({
  name: "todos",
  initialState,
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchTodos.pending, (state) => {
        state.loading = true;
        state.error = null;
      })
      .addCase(fetchTodos.fulfilled, (state, action: PayloadAction<Todo[]>) => {
        state.loading = false;
        state.todos = action.payload;
      })
      .addCase(fetchTodos.rejected, (state, action) => {
        state.loading = false;
        state.error = action.error.message || "Failed to fetch todos";
      })
      .addCase(addTodo.fulfilled, (state, action: PayloadAction<Todo>) => {
        state.todos.push(action.payload);
      })
      .addCase(updateTodo.fulfilled, (state, action: PayloadAction<Todo>) => {
        const index = state.todos.findIndex((todo) => todo.id === action.payload.id);
        if (index !== -1) {
          state.todos[index] = action.payload;
        }
      })
      .addCase(deleteTodo.fulfilled, (state, action: PayloadAction<string>) => {
        state.todos = state.todos.filter((todo) => todo.id !== action.payload);
      });
  },
});

export default todosSlice.reducer;

// Selector
export const selectTodos = (state: RootState) => state.todos;

```

### Step 3: Configure the Store

Create a `store.ts` file to configure the Redux store.

```

typescript Copy code

import { configureStore } from "@reduxjs/toolkit";
import todoReducer from "../slices/todoSlice";

export const store = configureStore({
  reducer: {
    todos: todoReducer,
  },
});

// RootState type
export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;

```

## Step 4: Provide the Store in Your App

Wrap your application with the `Provider` from React-Redux in `main.tsx`:

```
tsx Copy code

import React from "react";
import ReactDOM from "react-dom/client";
import { Provider } from "react-redux";
import { store } from "../store";
import App from "../App";

ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);
```

## Step 5: Use in a Component

Consume the Redux state and dispatch actions in your To-Do component.

```
tsx Copy code

import React, { useEffect } from "react";
import { useSelector, useDispatch } from "react-redux";
import { AppDispatch, RootState } from "../store";
import { fetchTodos, addTodo, updateTodo, deleteTodo } from "../slices/todoslice";

const TodoApp: React.FC = () => {
  const { todos, loading, error } = useSelector((state: RootState) => state.todos);
  const dispatch = useDispatch<AppDispatch>();

  useEffect(() => {
    dispatch(fetchTodos());
  }, [dispatch]);

  const handleAddTodo = () => {
    dispatch(addTodo({ title: "New To-Do", completed: false }));
  };

  const handleUpdateTodo = (id: string) => {
    dispatch(updateTodo({ id, title: "Updated To-Do", completed: true }));
  };

  const handleDeleteTodo = (id: string) => {
    dispatch(deleteTodo(id));
  };

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error: {error}</p>;

  return (
    <div>
      <h1>To-Do List</h1>
      <button onClick={handleAddTodo}>Add To-Do</button>
      <ul>
        {todos.map((todo) => (
          <li key={todo.id}>
            {todo.title} - {todo.completed ? "Completed" : "Pending"}
            <button onClick={() => handleUpdateTodo(todo.id)}>Update</button>
            <button onClick={() => handleDeleteTodo(todo.id)}>Delete</button>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default TodoApp;
```

This implementation connects your `useApi` hook with Redux Toolkit for CRUD operations while ensuring TypeScript type safety.

🔊 📄 🌐 🔄 📌

Message ChatGPT



