GOALS:

1. Understand Purpose of Reducers
2. Making API calls with Redux
3. What is the purpose of Redux Thunk
4. Workflow on porting a non-redux app to a redux app

Lecture time: 1hr

Learning time: 6hrs

App overview

* Post Title
* Post content
* Authors name

API: Json placeholder API: <https://jsonplaceholder.typicode.com/>

|  |  |  |
| --- | --- | --- |
| 1 | Redux Reducers |  |
| 2 | Redux api calls |  |
| 3 | Redux Thunk | Middleware to make network requests |
| 4 | Workflow |  |

WORK FLOW to add Redux to an application.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1 | Replace  index.js  with redux boilerplate | import React from "react";  import ReactDOM from "react-dom";  import { Provider } from "react-redux";  import { createStore, applyMiddleware } from "redux";  import thunk from "redux-thunk";  import App from "./components/App";  import reducers from "./reducers";  const store = createStore(reducers, applyMiddleware(thunk));  ReactDOM.render(  <Provider store={store}>  <App />  </Provider>,  document.querySelector("#root")  ); |
| 2 | Install packages | npm install redux react-redux redux-thunk axios |
| 3 | Reducer placeholder  Boilerplate | Create folder: reducers  Create file: index.js  import { combineReducers } from "redux";  export default combineReducers({  dummy: () => "This is a dummy reducer"  }); |
| 4 | Setup Action Creator  ( Example#1 returns an async function that will be handled by redux thunk) | Create folder: actions  Create file: index.js  **An Asynchronous Action Creator Example**  import jsonPlaceholder from "../apis/jsonPlaceholder";  **export const fetchPosts = () => async dispatch => {**  **const response = await jsonPlaceholder.get("/posts");**  **dispatch({ type: "FETCH\_POSTS", payload: response });**  **};**  **A Synchronous Action Creator Example**  **export const fetchPosts = () => {**  **const response = “Hello World!”;**  **return { type: "FETCH\_POSTS", payload: response };** |
| 5 | Call the action from a component | import React from "react";  import { connect } from "react-redux";  import { fetchPosts } from "../actions";  class PostList extends React.Component {  componentDidMount() {  this.props.fetchPosts();  }  render() {  return <div>PostList</div>;  }  }  export default connect(  null,  { fetchPosts: fetchPosts }  )(PostList); |
| 6 | Organise apis into a separate folder | Create folder apis  Create file for the api: (Example)  import axios from "axios";  export default axios.create({  baseURL: "https://jsonplaceholder.typeicode.com"  }); |
| 7 | Update Reducers | Rules for reducers (Grider 167)   1. Must return a value 2. Produces state data based on previous state data & an action 3. Has no dependency on external functions 4. Must not mutate the input state – Reason: a new return state is detected and triggers the react application to re-render. If original state is modified, and no new state is returned, React will not re-render.   Index.js  import { combineReducers } from "redux";  import postsReducer from ".postsReducer";  export default combineReducers({  **posts**: postsReducer  });  postsReducer.js  export default (state = [], action) => {  switch (action.type) {  case “FETCH\_POSTS”  return action.payload;  break;  default:  return state;  }  }; |
| 8 | MapStateToProps | import React from "react";  import { connect } from "react-redux";  import { fetchPosts } from "../actions";  class PostList extends React.Component {  componentDidMount() {  this.props.fetchPosts();  }  render() {  return <div>PostList</div>;  }  }  Const mapStateToProps = (state) => {  Return {posts:state.posts};  }  export default connect(  mapStateToProps,  { fetchPosts: fetchPosts }  )(PostList); |

First Exercise: The following to be managed by redux

cardsVisible – this is a state (managed by the reducer)

setCardsVisible – this is an action creator

getCardsVisible – this is an action creator

**Flow diagram for loading data with redux Grider 159**

|  |  |  |
| --- | --- | --- |
|  |  | **Code required in the Application** |
| 1 | Component rendered on screen |  |
| 2 | ComponentDidMount gets called | [5] Call to action creator in the component |
| 3 | Action creator makes API request | [4] Define Action creators |
| 4 | API responds with data |  |
| 5 | Action creator returns an action with fetched data on the payload | [1] Set up redux-thunk for asynchronous action creators  [6] Organise APIs into a separate folder |
| 6 | Reducer sees the action, returns the data off the payload (state is updated) | [7] Update reducers in [3] |
| 7 | New state is generated, causing React App to re-render | [8] Map state to props for the component |

**The redux cycle:**

**Action creator – Action – dispatch – Reducers - State**

**Interview question: Why can’t we mutate the State in redux? Grider 173 (2:00)**

* There is a “corner case” where this will cause trouble
* The corner case:

**Interview question: What’s wrong with fetch posts? Grider 162**

**Common problem with Redux Action Creator**

**Asynchronous action creator – Common Pitfall**

|  |  |
| --- | --- |
| **Good Example** | **Bad Example** |
| import jsonPlaceholder from "../apis/jsonPlaceholder";  export const fetchPosts = () => async dispatch => {  const response = await jsonPlaceholder.get("/posts");  dispatch({ type: "FETCH\_POSTS", payload: response });  }; | import jsonPlaceholder from "../apis/jsonPlaceholder";  export **const fetchPosts = async() => {**  **const response = await jsonPlaceholder.get("/posts");**  **return{ type: "FETCH\_POSTS", payload: response };**  **};**  **Error: (1) actions must be plain objects**  **(2) Use custom middleware for async actions** |
|  | **Reason for error 1**  **The compiler with ASYNC and AWAIT syntax causes return to occur at: jsonPlaceholder.get("/posts"); -**   * **This is a REQUEST object it is NOT a plain object.** |
|  | **Alternative method to illustrate error 2 – pass promise to the reducer (bad example #2):**  import jsonPlaceholder from "../apis/jsonPlaceholder";  export **const fetchPosts = () => {**  **const promise = jsonPlaceholder.get("/posts");**  **return{ type: "FETCH\_POSTS", payload: response };**  **};**  **Promise: is an object. An event is going to happen in the future.**  **The redux cycle does not wait for the promise to deliver the data.** |
|  |  |

|  |  |
| --- | --- |
| **Synchronous Action Creators** | **Asynchronous Action Creators: grider 164** |
| export const fetchPosts = () => {  const response = “Hello World!”;  return { type: "FETCH\_POSTS", payload: response };   1. Happens immediately 2. Returns { type, payload } object | **export const fetchPosts = () => {**  **return async function(dispatch, getState) {**  **const response = await jsonPlaceholder.get(‘/posts’);**  **dispatch({type: ‘FETCH\_POSTS’, payload: response})**  **};**  **}**  **REFACTOR ROUND 1:**  **export const fetchPosts = () => {**  **return async (dispatch) => {**  **const response = await jsonPlaceholder.get(‘/posts’);**  **dispatch({type: ‘FETCH\_POSTS’, payload: response})**  **};**  **}**  **REFACTOR ROUND 2**  **export const fetchPosts = () => async (dispatch) => {**  **const response = await jsonPlaceholder.get(‘/posts’);**  **dispatch({type: ‘FETCH\_POSTS’, payload: response})**  **};**  **Refer Grider 166** |
| Returns an action object | **With Redux Thunk, an action creator can return an action object or a Function.** |

**A middleware is required.**

**What is middleware – Redux Thunk**

|  |  |
| --- | --- |
| **1** | **Action creator – Action – dispatch – [MIDDLEWARE] - Reducers - State** |
| **2** | **Is called with every single action being dispatched** |
| **3** | **Will process take the action creator output and act on it.** |
| **4** | **Redux Thunk: If action creator returns an Action Object, do nothing. If action creator returns a function. Call the function** |
| **5** | **The Function passed into the middleware:**   1. **Has two arguments: dispatch, getState** 2. **Performs an async action** 3. **Manually dispatch the action when (2) is completed** |
| **6** | **Applying the middleware:**  **Index.js (top level of react application)**  import thunk from "redux-thunk";  const store = createStore(reducers, applyMiddleware(thunk)); |
| **7** | Construct the Asynchronous Action Creator with Redux-Thunk  import jsonPlaceholder from "../apis/jsonPlaceholder";  export const fetchPosts = () => async dispatch => {  const response = await jsonPlaceholder.get("/posts");  dispatch({ type: "FETCH\_POSTS", payload: response });  }; |

**Important exercise for Reducers – Grider 174 @0:41**

**Given that the reducer has an existing state, how do we generate the new state?**

|  |  |  |
| --- | --- | --- |
|  | **Bad** | **Good** |
| **Remove item from array** |  | **State.filter(e=> e !== ‘hi’)** |
| **Add item to array** |  | **[…State, ‘hi’]** |
| **Replace item in array** |  | **State.map(e=>e===’hi’?’bye’:e)** |
| **Updating a property in an object** |  | **{…state, name:”Sam”}** |
| **Add property to an object** |  | **{…state, name:”Sam”}** |
| **Remove property from an object** |  | **\_omit(state, “name”)**  **Must install lodash library**  **Otherwise**  **{…state, Name: undefined}** |