22. Redux Thunk and Mini-Project

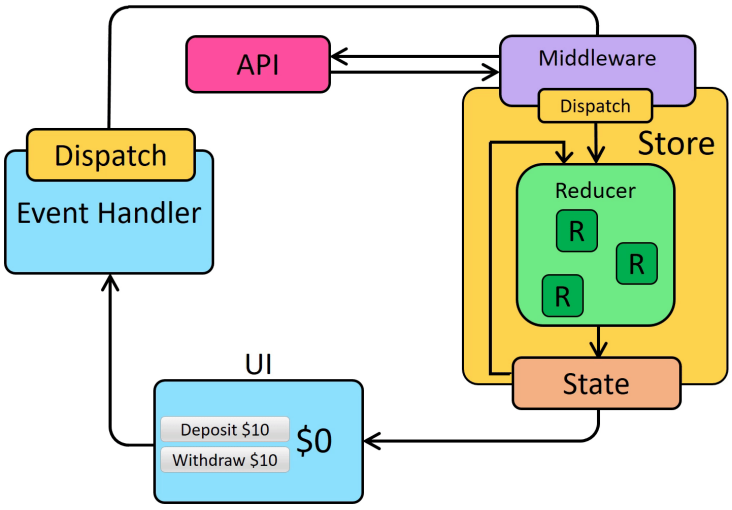
Table of Content

1. Using Middleware with Redux

By itself, a Redux store doesn't know anything about async logic. It only knows how to synchronously dispatch actions, update the state by calling the root reducer function, and notify the UI that something has changed. Any asynchronicity has to happen outside the store.

But, what if you want to have async logic interact with the store by dispatching or checking the current store state? That's where Redux middleware come in. They extend the store, and allow you to:

- Execute extra logic when any action is dispatched (such as logging the action and state)
- Pause, modify, delay, replace, or halt dispatched actions
- Write extra code that has access to `dispatch` and `getState`
- Teach `dispatch` how to accept other values besides plain action objects, such as functions and promises, by intercepting them and dispatching real action objects instead



https://d33wubrfki0l68.cloudfront.net/08d01ed85246d3ece01963408572f3f6dfb49d41/4bc12/assets/images/reduxasyncdataflowdiagram-d97ff38a0f4da0f327163170ccc13e80.gif

2. Redux Thunk

There are many kinds of async middleware for Redux, and each lets you write your logic using different syntax. The most common async middleware is redux-thunk, which lets you write plain functions that may contain async logic directly.

Redux Toolkit's `configureStore` function automatically sets up the thunk middleware by default, and it is recommend using thunks as the standard approach for writing async logic with Redux.

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit'

const initialState = {
  posts: [],
  status: 'idle',
  error: null
}

export const fetchPosts = createAsyncThunk('posts/fetchPosts', async () => {
  const response = await client.get('/fakeApi/posts')
  return response.data
})
```

and now inside the createSlice() method, we can use like this:

```
const postsSlice = createSlice({
  name: 'posts',
  initialState,
  reducers: {
    // omit existing reducers here
  },
  extraReducers: function (builder) {
    builder
      .addCase(fetchPosts.pending, (state, action) => {
        state.status = 'loading'
      })
      .addCase(fetchPosts.fulfilled, (state, action) => {
        state.status = 'succeeded'
        // Add any fetched posts to the array
        state.posts = state.posts.concat(action.payload)
      })
      .addCase(fetchPosts.rejected, (state, action) => {
        state.status = 'failed'
        state.error = action.error.message
      })
  }
})
```

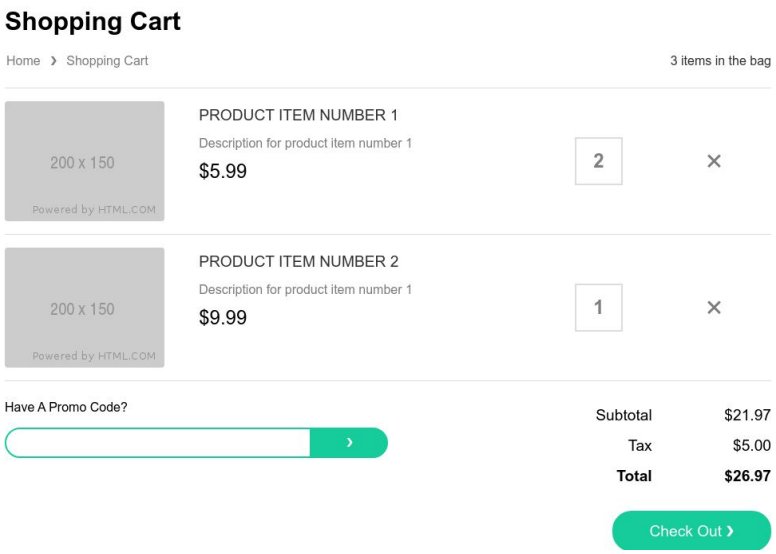
▼ **Dispatch the async Thunk from Components**

Inside your components, you can write:

```
const postStatus = useSelector(state => state.posts.status)
useEffect(() => {
    if (postStatus === 'idle') {
      dispatch(fetchPosts())
    }
  }, [postStatus, dispatch])
```

Assignments

1. Build a Cart page to display each Cart Item



2. Build the recipe application using the https://spoonacular.com/food-api