Redux

# What is Redux?

* A 3rd party library to manage **Global State**
* It’s a standalone library, but easy to integrate with React apps using react-redux library
* All **Global State** will be stored into one **globally accessible store,** and to update it using **actions**
* Redux is a combination of the **Context API and the useReducer** hook
  + The state is stored globally and updated with actions (useReducer)
  + As soon as the global state is updated, all consuming comonentens are re-rendered (Context Api)
* Redux has ‘two versions’
  + Classic Redux
  + Modern Redux Toolkit

# When to use Redux?

A screenshot of a computer

Description automatically generated

* Redux is PERFECT to manage
  + UI Global State (State that is not part of a outside server)

# How Redux Works (Redux Cycle)?

* Almost the same as the useReducer.
* We Dispatch an action
* **We dispatch the action to the STATE STORE**
  + The single source of truth of the global state
  + Contains one or multiple reducers
  + Why multiple reducers?
    - We should create 1 reducer / application feature
      * Cart
      * Login
      * Products
* **Instead of writing the action by hand, we create FUNCTION to do this automatically**
  + Like this, we can keep all actions in one place

# Redux 1st Practice – banking\_app

* We create the store.jsx
  + Create the initialState
  + Define the reducer function
    - We pass the INITIALSTATE as the default state
  + Use Switch for actions
    - We define the actions in the following format
      * ‘domain / action’ 🡺 ‘account/deposit’
  + Create the Redux Store
    - We use Actual Redux Feature (install redux)
    - Const store = createStore(reducer)
  + We can now dispatch actions on the store
    - Store.Dispatch({type: action type, payload: action payload})

A screen shot of a computer

Description automatically generated

* Import the whole ‘/store.jsx’ file (to run the code inside this file)
* Using store.getState()
  + We get the current state

A screen shot of a computer

Description automatically generated

* Requesting a Loan

A screen shot of a computer program

Description automatically generated

## Redux Action Creators (function to create actions)

* Not actually part of Redux, but it’s a **convention**
* Functions that do the same thing as a dispatch, but it’s a function with params

A screen shot of a computer program

Description automatically generated

* Using the Action Creators

A screen shot of a computer program

Description automatically generated

## We can create as many initialStates and Reducers as needed inside the same store file (GLOBAL SOURCE OF TRUTH FOR ALL STATE)

* After creating the reducer
* WE need to **COMBINE ALL REDUCERS into the same STORE**
  + We use the ‘combineReducers’ function from Redux
  + rootReducer = combineReducers({
    - account:accountReducer,
    - customer: customerReducer
    - })
* A screen shot of a computer code

  Description automatically generated

The output of getState will be the reducers combined

A screen shot of a computer

Description automatically generated

# Professional Redux File Structure

* Back in the early days
  + there was 1 reducers folder 🡺 1 file per reducer
  + 1 actions creators folder 🡺 1 file per action creator
* **Now,** we organize the application into **Features**
  + Feature 1 🡺 Account
    - This will contain **everything** that is **related to the ACCOUNT**
  + Feature 2 🡺 Customer
    - This will contain everything that is related to the CustomerA screenshot of a computer

      Description automatically generated
* For Redux, we create **slices**
  + **accountSlice**
    - a ‘slice’ from the store that contains the account logic
    - we export default the reducer
* A screen shot of a computer

  Description automatically generated
  + - we name export the action creators
* A screen shot of a computer

  Description automatically generated

**OR**

**A screen shot of a computer

Description automatically generated Just using export**

* + **customerSlice**
    - a ‘slice’ where all logic for customer is stored
    - we export default the reducer
    - we name export the action creators
* We **COMBINE everything** in the main store.jsx file
  + Import the reducers
  + Combine them
  + Create the unique store
  + Export the Store

A screen shot of a computer program

Description automatically generated

* Import the store and whatever action creators we need into the File

A screen shot of a computer

Description automatically generated

# Connecting our app with the Redux Store

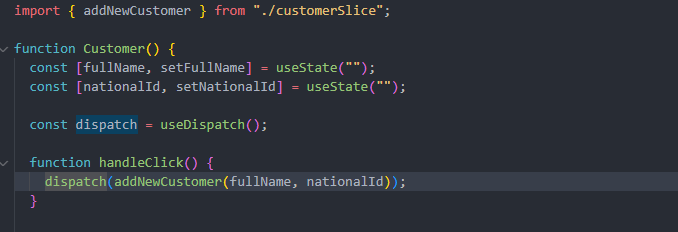
* We need to use the ‘react-redux’ package
  + Npm I react-redux
* Import the Provider from the react-redux library
  + 
* Wrap the application ( component ) with that Provider and provide the store through a prop
  + A screen shot of a computer

    Description automatically generated
* All COMPONENTS will be able to read from the store and to dispatch actions
* To read data from the Redux store, we use the **useSelector** hook
  + useSelector((store)=>store.customer)
  + A screen shot of a computer

    Description automatically generated
  + In this way, whenever the **store changes, the subscription to the store will make the component re-render**

### Dispatch Actions to the Store from within React Components

* We need to use the **useDispatch()**  hook provided by react-redux
  + Const dispatch = useDispatch();
* To dispatch an action, we also use the **action creators**
  + Dispatch(addNewCustomer(‘name’,nationalId)



# Redux Middleware

* Extend the functionality of Redux

Where to make an asynchronous API call in Redux?

* We **cannot make** it inside the Store
  + The reducers are **pure functions**
* **In Middleware!!!**

## What is Middleware?

* It’s a function that sits between the dispatch and the store
* It’s allowing us to run code after dispatching, but before reaching the reducer in the store
* **Perfect for:**
  + Asynchronous calls
  + API calls, timers, logging
  + The place for side Effects
* WE are using a 3rd party middleware library called – **Thunk**

## Making an API call using Thunk.

* We Deposit Money into a foreign currency and using an API call, we will convert the deposited money into USD

### Steps to use Middleware

1. Install Middleware Thunk
   1. Npm I redux-thunk
2. Import the Middleware Thunk into our store
   1. When creating the store, use applyMiddleware function from Redux
   2. Inside the applyMiddleware, we pass the ‘thunk’ import

A screen shot of a computer program

Description automatically generated

1. Now we can use the thunk
   1. In the Action Creator, instead of returning the ‘action’
   2. We return a function

A screen shot of a computer

Description automatically generated

* 1. When Redux sees this function, it will know that it’s a THUNK
  2. Redux knows that that function is what we want to run BEFORE accessing the store
  3. The function gets as args the ‘dispatch’ and the ‘currentstate’

A screen shot of a computer

Description automatically generated

* 1. We are first making the API call, and then we are returning the DISPATCH of the same event, with the new result

A screen shot of a computer program

Description automatically generated

# Redux DevTools

1. Install the Google Chrome Extension
   1. Redux DevTools
2. Install the devTools library
   1. Npm I @redux-devtools/extension
3. Import {composeWithDevTools} named imoort into the store
4. Wrap the applyMiddleware(thunk) function in the compooseWithDevTools

A screen shot of a computer code

Description automatically generated

1. Now we have the ‘Redux’ available in the DevTools
2. WE have access to multiple tools now

A screenshot of a computer

Description automatically generated

# Modern REDUX – Redux Toolkit (RTK)

## What is Redux Toolkit?

* Modern Way to write redux
* Preferred way by the Redux Team
* Forces everyone to use the Redux Best Practices
* Both ways are 100% compatible with one another
* Lot less code to achieve the same result
* Gives us **3 big things:**
  + We can write code that ‘mutates’ state inside reducers
    - Will be converted to immutable logic by **Immer** library
  + **Automatically** creates **action creators**
  + **Automatically** sets up **Thunk Middleware** and **DevTools**

## How to use RTK?

1. Install redux toolkit
   1. Npm I @reduxjs/toolkit
2. Instead of the createStore, we will use the **configureStore**
   1. import { configureStore } from "@reduxjs/toolkit";
   2. configureStore will automatically
      1. create the action creators
      2. inject the Thunk Middleware
      3. install the DevTools
3. RTK has build in the **createSlice** function
   1. **3 Big** benefits
      1. Automatically create action creators
      2. Make writing the reducers a LOT easier (no switch and default default)
      3. Mutate state inside Reducer
   2. We create the **SLICE**
      1. The name
      2. The initial State
      3. Reducers ( one for each action )
         1. Deposit(currState, action)
            1. We mutate the state directly, not with {…currState}

A screen shot of a computer

Description automatically generated

* 1. We export what we need from the new slice

A screen shot of a computer

Description automatically generated

* 1. We use the actions in the same way as for classic Redux

A screen shot of a computer code

Description automatically generated

* 1. **If we want to send multiple params through the action argument, we have to prepare the action creator to receive multiple params like this**

**A screen shot of a computer screen

Description automatically generated**

## Thunk in RTK

1. We can use the previous Action Creator, THE ONE THAT WE HAVE WRITTEN MANUALLY
2. The only thing we must be careful is for the **type**: to have the exact shape
   1. Account/deposit
3. We must delete the export of the named ‘deposit’ action

A computer screen shot of text

Description automatically generated

# Context API + useReducer vs Redux

## Context API + useReducer

* Already in React
* No additional packages needed
* Easy to setup in a **single context**
* For multiple contexts, we might get into **Provider Hell** 
  + Many many context providers into the App.js
* No built-in Mechanism for API
* Performance optimization is a **pain**

## Redux

* Needs additional packages (larger bundle size)
* More work for initial setup
* Once setup, **easy to add additional state ‘slices’**
* **Middleware + Thunk**  for async operations
* Many Optimization is **out of the box**

A screenshot of a web page

Description automatically generated