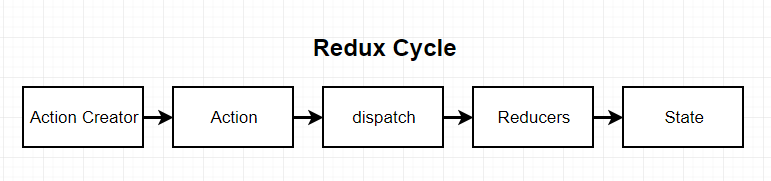
**Redux Notes**

**What is Redux?**

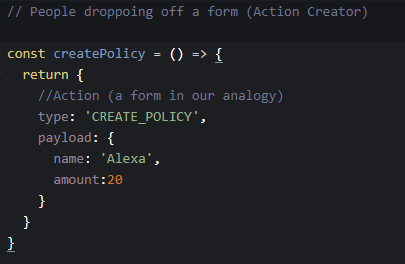
* State management library
  + Rendering content on the screen
  + Handling user interaction
  + Handling data inside applications
* Redux does not automatically detect action creators being called
* Redux does not automatically detect a function returning an object that is an ‘action’
* Makes creating complex applications easier (exponential difficulty vs linear complexity in understanding)
* Only modify states through action creators
* Not required to create a react app
* Not explicitly designed to work with React



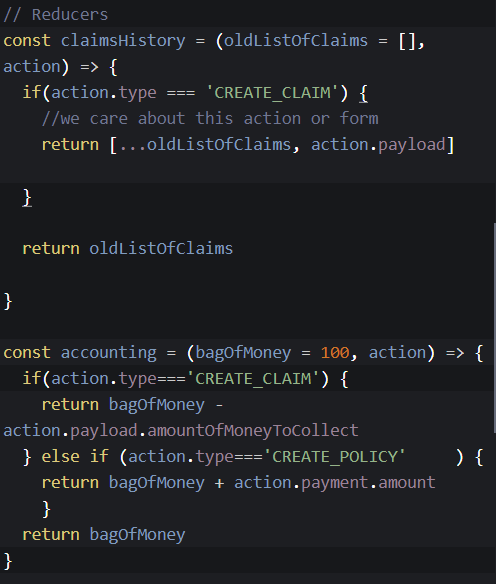
* Action creator - Function that is going to create or return a plain javascript object (as an action which has a type which describes the purpose of the action and a payload property which describes some context around the change we want to make or what the action is doing )
* Action - contains information to describe some change we want to make to the data inside of our application
* Dispatch - Dispatch function takes in an action and makes copies of that object where it passes it off to a bunch of different places (to each reducers) throughout our application
* Reducer - A function that is responsible for taking in an action and some existing amount of data, it is going to process that action and make some changes to the data (modify) and return it so that it can be centralised to some other location (take in an action, look at its type and based on the type it will then decide on how it should update its data)
* State - central repository of all information that has been created by reducers and all the information gets consolidated inside the state object, so that react application can easily reach in redux application and get access to all of the data in the application (so react app does not have to go to each separate reducer to ask for current states therefore through preventing passing of props)

**Redux Code**

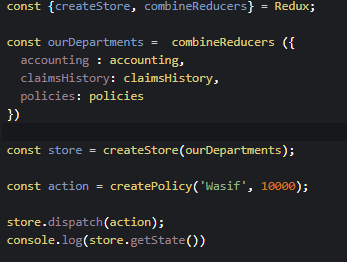
* Action Creator

****

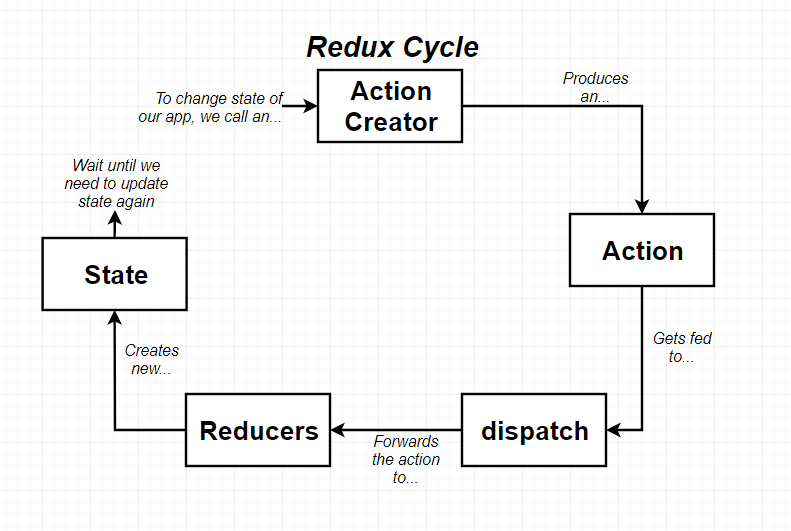
* Reducers



* Store
  + combineReducers - helper function that turns an object whose values are different reducing functions into a single reducing function to pass to createStore
  + getState - function that gets the entire assembled repository for data and get access to all of it

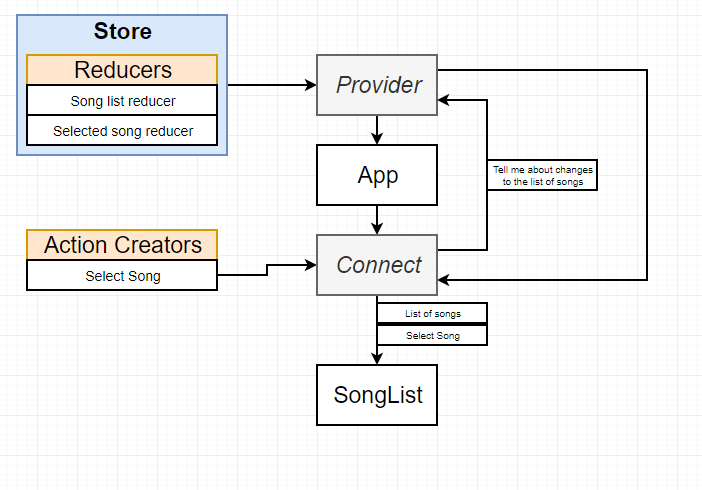


**Redux Cycle**



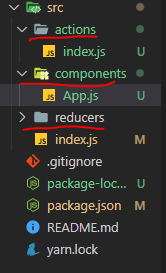
**React-Redux**

* Provider includes all the information of the application as it contains the store and reducers (passed as a reference to the redux store)
* Connect contains the configuration of the application and allows components of the application to be able to interact with the redux store
* Configure the connect tag by telling it what different pieces of state do you want from store and what action creators does it want to be connected with
* The states and action creators show up as props in the songlist component
* Connect communicates with the provider through context system not props
* Context system allows any parent component to communicate directly with any child component even if there are other components in between them such as the app component
* When it renders the songlist as a child it needs to reach back up to provider and tell it that it needs to get the list of songs that are contained within the store
* Provider will send the updated list of songs to connect
* Connect will take that list of songs and pass it as a prop to the song list components
* Action creators are not stored in store, instead call an action creator and take the action that gets returned and send it to the store.dispatch function
* Connect wants to also be able to get the select song action creator from within the songlist and is sent down to the song list component as a prop



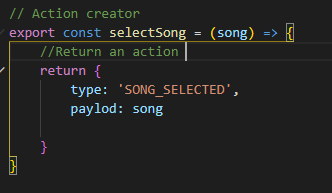
**Folder Structure**

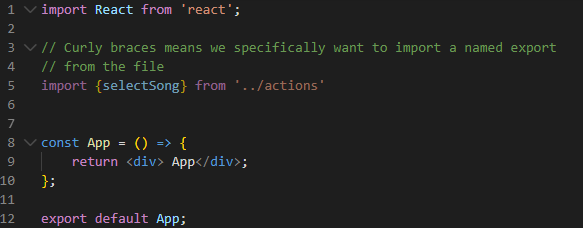
* Actions folder contains an index.js as it can be referred in app.js as a shortcut



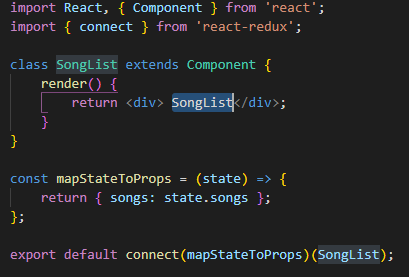
**Coding Tips**

* Action Creator below can be exported as a named function such as “selectSong” which can be imported using curly braces





* Connect function the first set returns a connect function (function to connect) and the second set (Songlist) contains a component that is the second function call

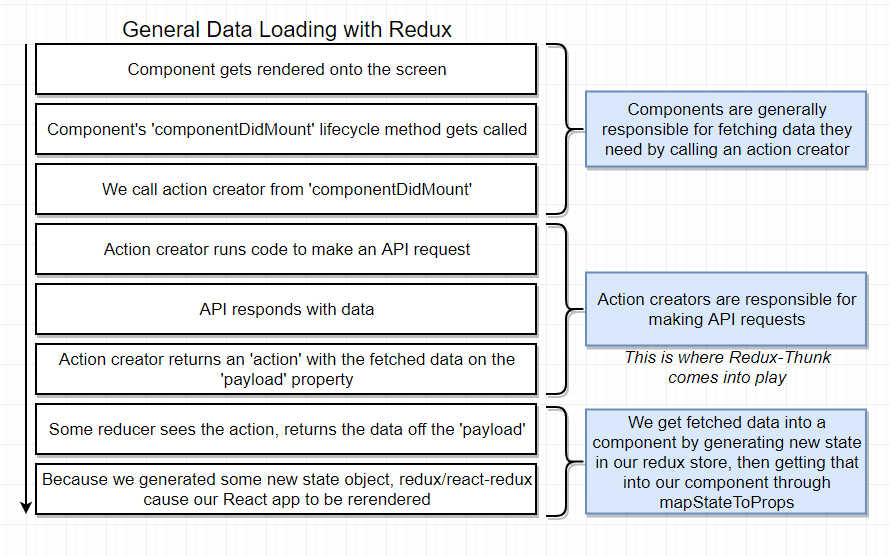


* mapStateToProps function - Take first argument of state object and going to run a calculation/computation that will cause the data to show up as props inside the component.
  + It will return an object that will show up as props inside the component

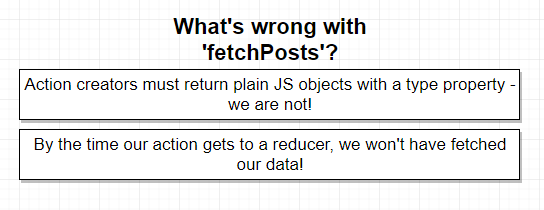
**Review of Dependencies**

* Redux - The redux library for state management
* React-redux - Integration layer between react and redux
* Axios - Helps make network requests (better than fetch function as there is a lot of manual holding you have to do with fetch)
* Redux-thunk - Middleware (functions that will change the behaviour of the redux store, add new features and behaviours ) to help us make network requests in a redux application

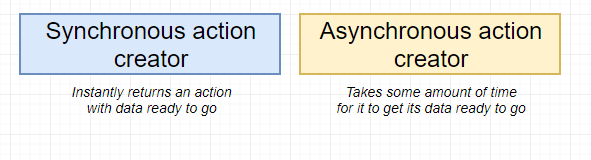
**General Data Loading with Redux**



**What’s wrong with fetchPosts?**

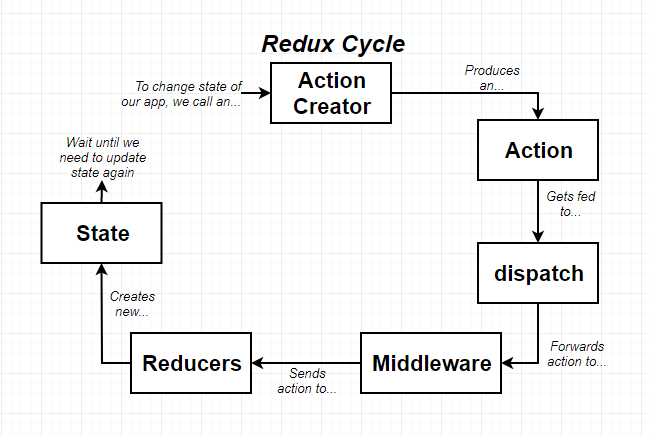
****

**Async and sync code**

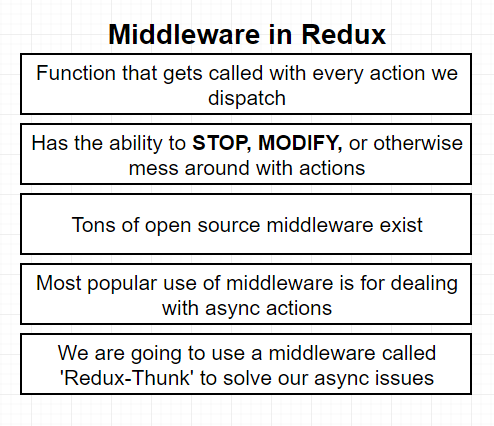
****

* Asynchronous action creators require middleware (as action creators to reducer process takes milliseconds and would process it before receiving a request for data)
* Problems with asynchronus action creator is that it causes to return a request object instead of an action

**Middleware**

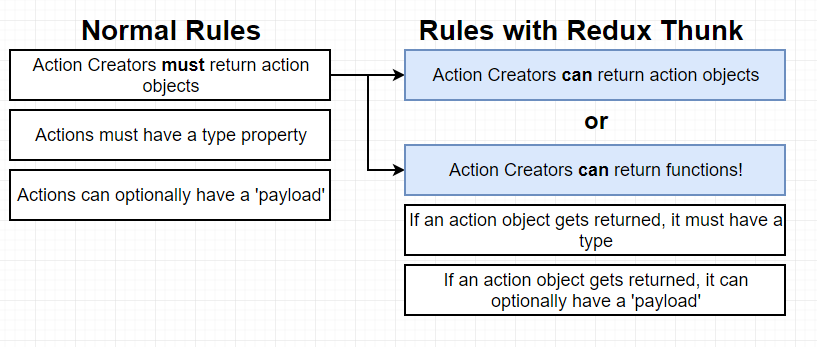


**Middleware in Redux**

****

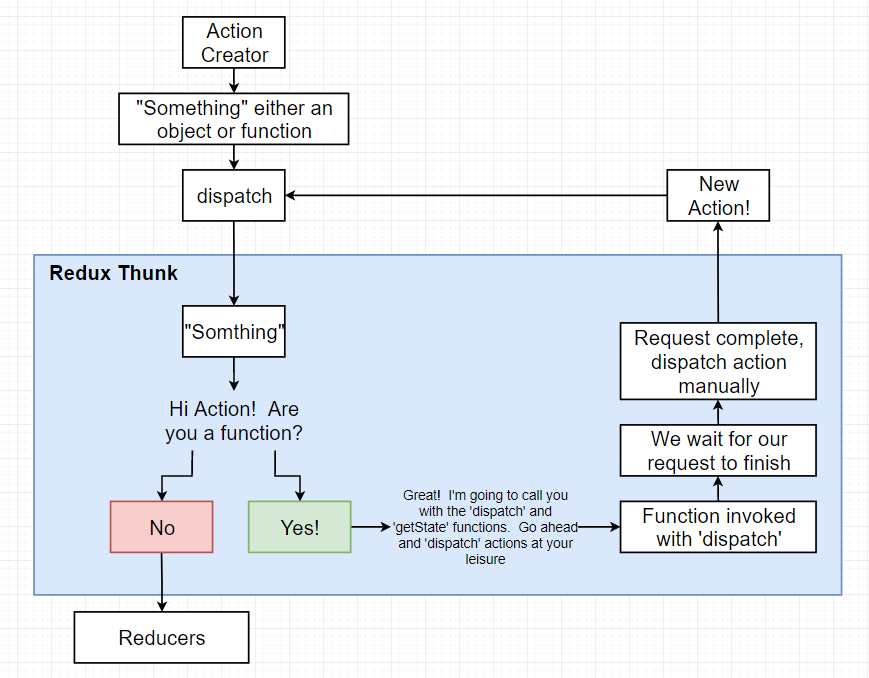
**Redux Thunk**

* Allows for the use of async and await

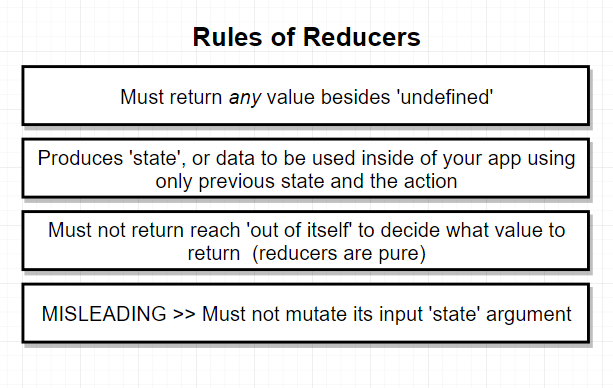
****

**Redux Thunk behind the scenes**

* First calls an action creator
* Something returns an object or function
* Will take that something and pass it to the dispatch function
* Dispatch function then sends it to the middleware (redux thunk)
* Inside of redux thunk it will take a look at that something and consider whether it is a function or object
  + If it is an object it will pass it to the reducer
  + If it is a function it will invoke it and pass it to the dispatch (unlimited power to initiate changes to the data on the redux side of the app) and getState (can be called on a redux store and return all of the data inside the store therefore read and access data) as functions
* Waits to receive a response from the request or when the request is complete (Typicode API)
* Request is complete and dispatches action manually at some point of time in the future
* New action gets created (plain object)
* Flows back into the dispatch
* Dispatch sends it to redux thunk
* Since it is an object it will send it to all of the reducers

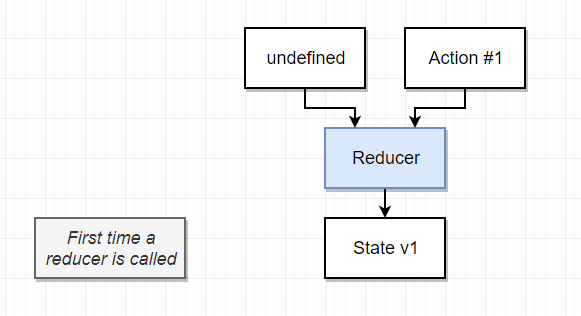
****

**Rules of reducers**

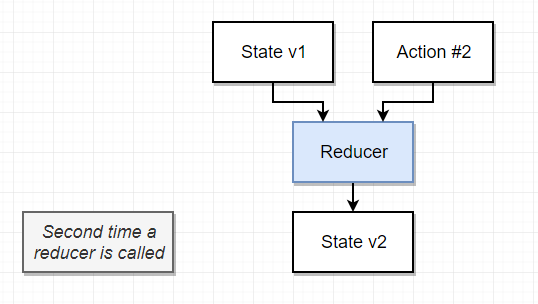
****

**Reducer**

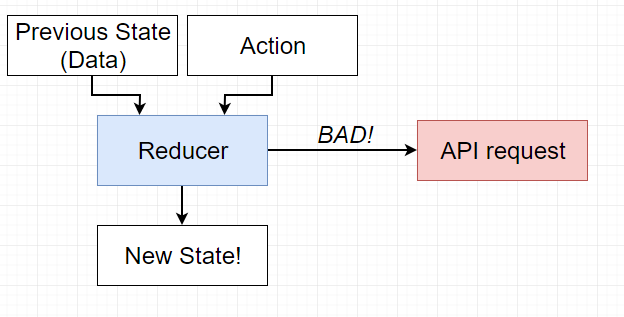
* Starting up the application, each reducer will be called automatically one time
* It is an automatic invocation that allows reducers to specify default state value
* First argument will be as undefined and second argument will be an object (action #1)
* Up to the reducer to take the two arguments and return an initial state value

****

* Second time a reducer is called, the first argument will be equal to state v1 (whatever it was the last time it ran so previous state) but only action object gets changed with type and payload

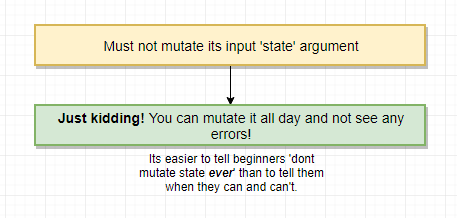
****

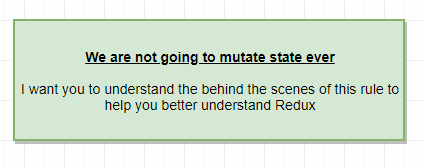
* Cannot reach out of the function for a reducer for example cannot make an API request

****

**Mutate states**

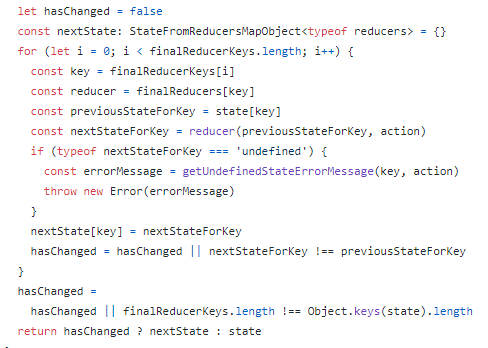
* Mutate means to change the contents (adding, removing or updating elements)
  + Objects and arrays are mutable
  + Strings and numbers are immutable (cannot be changed)





* Mutating state argument, application will not work the way you would expect it to
* Reason being it is suggested you don’t change is because if you accidentally return the same value that is passed to the reducer, even if it is modified and say return state at the bottom and it is still the same object or array whether modified or not, redux is going to redux will state that there has been no difference here as the object or array is in the same memory and therefore no updates to any data inside the application and see no updated content appear on the screen

**CombinedReducers Function in Redux Explained**



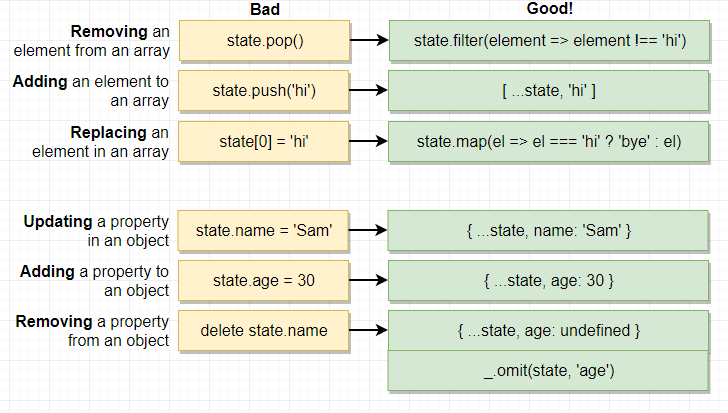
* Within combinedreducers function (from the official redux github repo), each time an action gets dispatched, this block of code will run and send the action around to all the different reducers inside the application
* The code block checks to see after running each reducer if any reducer returned a brand new array, object or value for a number or a string.
  + If they do then redux is going to return a brand new result from all of your reducers
  + Else if you’re reducers returned no new value it is going to return the old states or the old result from each of your reducers
* Redux returns old state value it will not notify rest of the application the any of the data has changed
* If you do have a new state, and have changed something from one of your reduceres and you instead return next state here, redux is going to look at that object and notify that there have been new states processed from all the reducers and it is going to notify the rest of the application that there is a new state available and therefore cause the react application to re-render

**Equality of Arrays and Objects**

* === - equal to
* !== - not equal to
* Primitive values (numbers and strings) are value comparisons
* Though with arrays and objects using equal operators, javascript is checking to see if it is a reference to the exact same array in memory but not the contents of the array

**Good/bad ways to mutate states in Reducer**

* The bad is because it mutates the objects / arrays
* As we want to always return brand new arrays / objects
* Setting undefined is not the best way recommended for removing properties in an object, as the key value pair still exists
* Use lo-dash for omitting a property in object as it creates a new object / array without the property

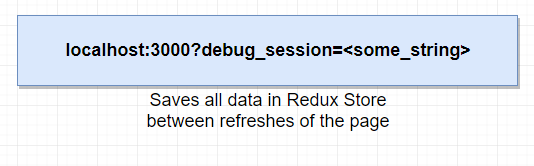


**Memoized**

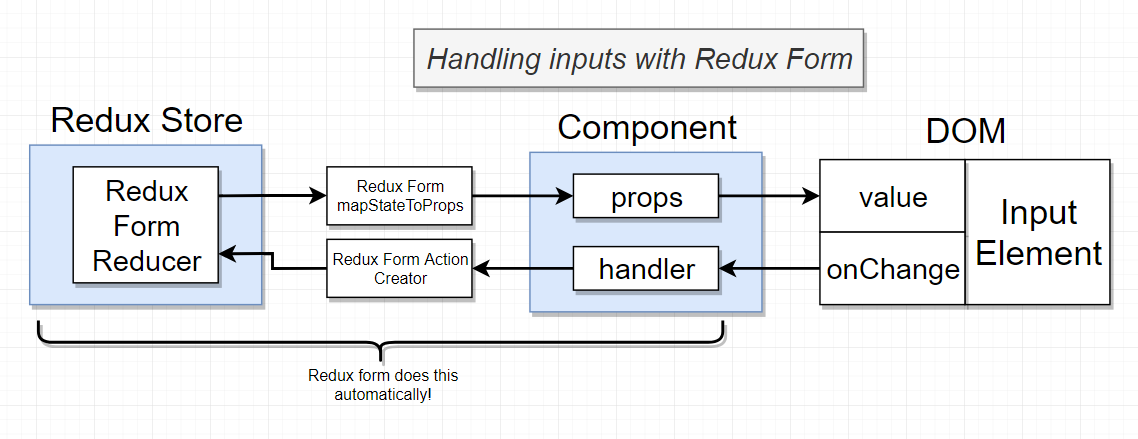
* Memoized from the lo-dash library gets a function and wraps it up with a bunch of code and returns a new copy of the function (same behaviour of the function that it copies)
* It only runs the function one time and when it is called in the future, the function won’t run but it will return the value of the function
* Due to redux thunk it will invoke the function and make a network request
* However, if you want to refetch a user (data modified / updated), cannot do it with memoized as it fetches each user one time in the application, would have to declare another action creator that has the same logic but does not apply that memoization effect

**Redux Tools**

* Redux Dev tools - allows you to view the state of your application and be able to jump back in time



**Redux Handling Forms**



* Use validate function to check for user inputs / values within a form