**Plural Site React:**

1. Intro
   1. The Big Picture
2. Beginner
   1. Getting Started
   2. Using React Hooks – Chapters 1-3
3. Intermediate
   1. Building Application with React and Flux
      1. Required: Chapters 1-9
      2. 10 – 11: Flux has some issues that makes Redux the better choice IMO. Is easier to use and uses the same principles as RxJs.
      3. See Redux Notes Below for details
   2. Building Application with React and Redux
      1. Required: Chapter 2 and Chapters 6 – 9. Other chapters are the same as the previous module as it is the same author.
      2. 10 – 12 : Redux Thunk asynchronous calls. I’m not sure if we really need this framework
4. Advanced
5. Extras

**NOTES:**

We will not be using classes as the latest version is moving away from classes and focusing on simple functions for hooks. This reduces code overhead and over complication of classes’ declarations for universal hook states.

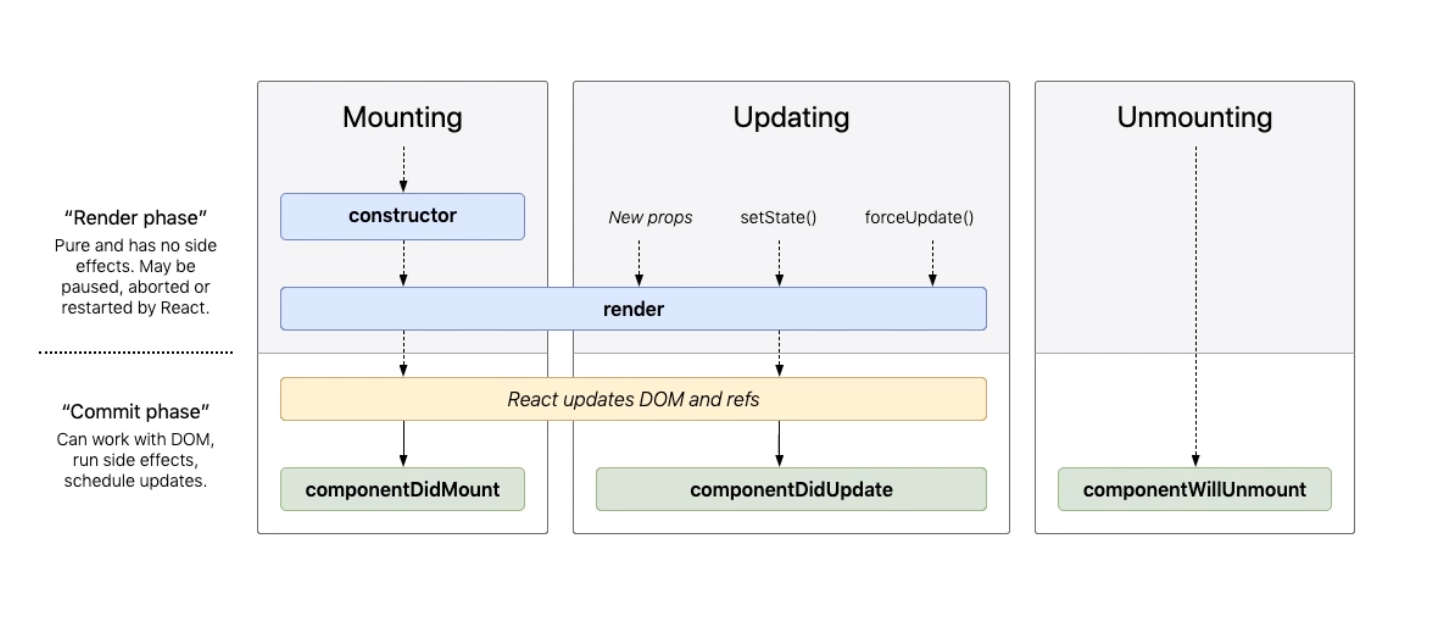
Babel Transpiles JSX into react elements.

Webpack transforms React elements into plain Javascript to run in the browser.

Typescript is now only used handle models, enums, and interfaces.

API calls will be used standard JS instead of Typescript due to the verbose nature and setup for calls.

React phases:

* 
* componentDidMount –C# equivalent OnInit and PageLoad
* componentDidUpdate – C# equivalent PageLoad and ASPx listeners
* componentWillUnmount – Angular equivalent ngOnDestroy

Smart & Dumb Components:

* Smart – Parent Component Handles State and Logic
* Dumb – Child Component Handles Mark Up
* C# & ASP Equivalent – Smart is the page and Dumb is like ASPx simple controls except all logic and decision making is defined from the parent page.
* Single Page React:
  + Logic calls before render and return is smart type
  + Return markup is dumb

Hooks:

* It uses array destructuring
* useState
  + Const [email, setEmail] = useState(“”);
    - First element “email” is the piece of state \_\_\_\_ C# Get
    - Second element “setEmail” is the setter \_\_\_\_ C# Set
    - Declaration “useState” can set string, number, array, or an object.
* useEffect
  + Runs after each time the state is changed
  + It requires a dependency at the end like email to listen to changes
    - WARNING: If a dependency is not defined, it will run in infinite loop, and calling an API will make it crash. We will need to diligent not to cause this issue.
  + useEffect is a combination of react componentDidMount, componentDidUpdate, and componentWillunmount.
* It shares the state between components
* Rules:
  + Only call from React function components
  + Must be declared on the top level, cannot be declared in loops, statements, or other sub processes.
  + If a hook needs to be ran conditionally, put the conditional statement within the hook.
  + React tracks hook calls in order with an array

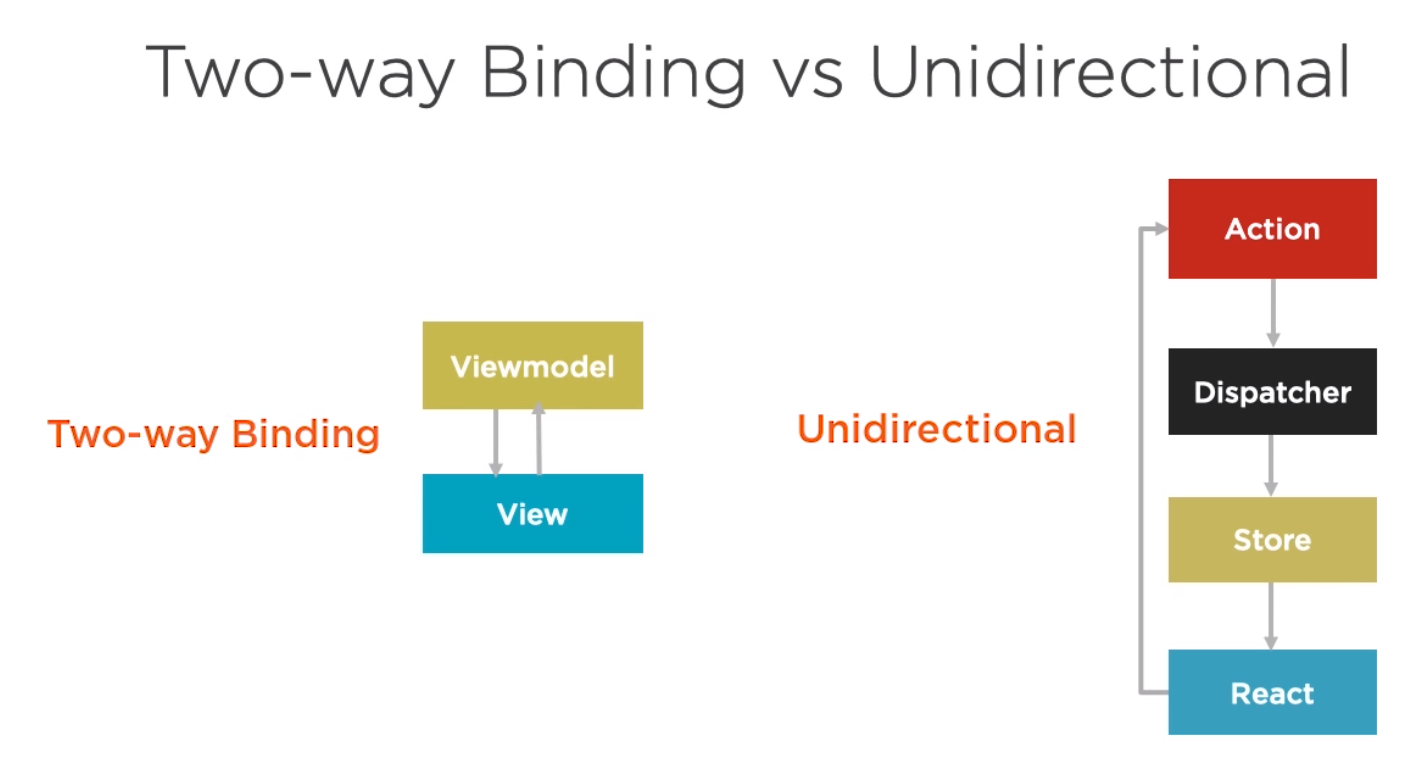
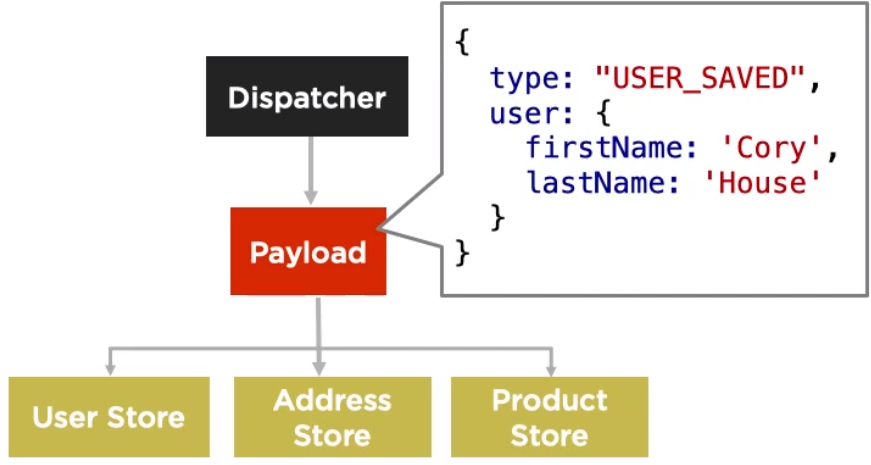
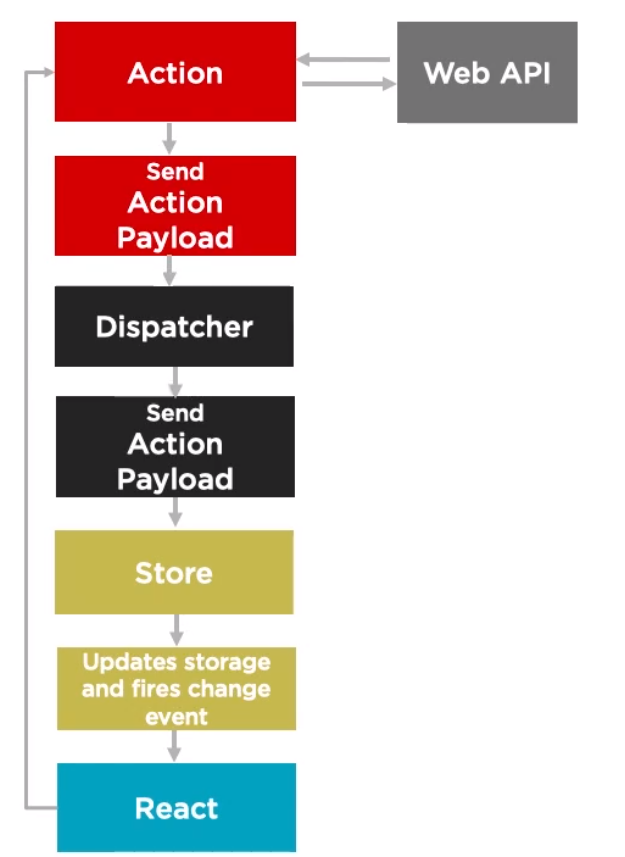
Router:

* HashRouter – Preappends routing with hashes?? Why ??
* BrowserRouter – Standard HTML5 routing
* MemoryRouting – For ReactNative(Mobile) and testing
* For the project BrowserRouter will be used

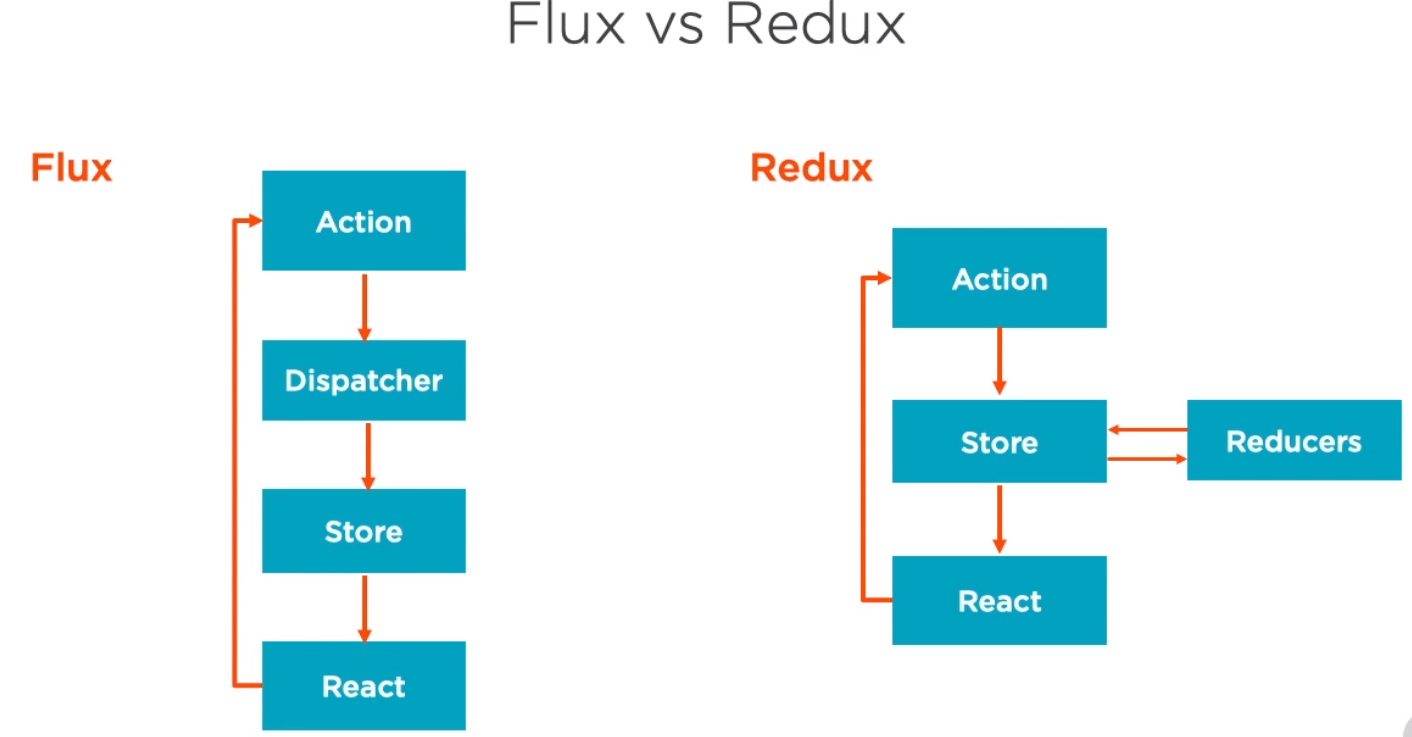
Forms:

* Inputs are controlled components and elements always match a value assigned to a prop.
* Every input type control requires a change handler.
* We should always work with controlled components.

Flux:

* Unidirectional data flow that is battle tested and supported by Facebook.
* Two-way binding is what Angular and Vue uses.
* 
* 3 Parts:
  + Action
    - Triggered by the user interface and server
    - Pass payload to a dispatcher
  + Dispatcher
    - Central Hub - Only one per app
    - Singleton registry
    - Holds list of callbacks
    - Sends actions to stores
    - Requires Constants for actions
  + Store
    - Holds state, logic, and data
    - Not a model – Contains Models
    - Uses Node’s EventEmitter
    - Structure:
      * Extend Event Emitter
      * AddChangeListener and RemoveChangeListeners
      * Emit Change
      * 
* 
* Need to be used at the Parent component.
* **\*\*\* AVOID NESTING CONTROLLER VIEWS – Keep it dumb for presentation layer \*\*\***
  + That is the problem with MWP Angular with our states are being updated in Parent, child, and sub child components slowing down the system.
* Requires event listener objects… Damn!

Redux:

* Unidirectional
* Same actions
* Except it has only one store vs Flux’s many stores.
* 
* Redux doesn’t need a dispatcher; it is reliant on reducers which are pure functions.
* Flux data is mutable while Redux data is immutable that brings a whole new copy to the store
* Redux separate store and change logic responsibilities
* Simplifies the ability to use a hierarchy of reducers.
* No need for the event emitter patter \*\*YES!\*\*

Redux Continued:

* When object has an array, it does not automatically copy over.
  + Const user = {name:”Bacon”, credentials: { type: ‘Agent’, key: ‘BlahBlah’ }}
  + Const userCopy = { …user }; Credentials was not copied, just referenced.
  + Const userCopy = { …user, credentials: { …user.credentials} }; Copied over
  + Only clone the objects or sub objects that change. Otherwise the entire object will kick off other listeners slowing the application.
    - Uga Buga – DEEP CLONING BAD, SUB-OBJECT CLONE GOOD.
* Arrays
  + Avoid push, pop, reverse as it will kick off all the reducers.
  + Use map, filter, reduce, concate, or spread to properly return the new array.
* There are libraries that help with this issue but I think we should not use those custom libraries.
* Reducers:
  + 1 Store, Multiple Reducers
  + Slice of state – handles small parts of the store
  + Each action can handle multiple reducers and each reducer can handle multiple actions
  + Forbidden Laws
    - Mutate arguments
    - Perform API calls or side effect updates
    - Call non-pure functions that does B
* Provider:
  + Only call once at the top level and wrapped around the app component.
    - <Provider><Root/></Provider>
* Connect
  + Wrapped around the component level
    - Export default connect (passToProps, dispatchToProps)(PageName);
      * First function is what state to pass
      * Second function is what action to pass
    - Benefits:
      * No Manual Subscriptions, YES!!!
      * Clearly declared exactly what state should update the component
      * Performance – simply maps what components need to rerender upon a change.
    - Both are optional
  + Only map multiple use stores like User is required in multiple places.
* 8 Initial steps to setup Redux:
  1. Create Action
  2. Create Reducer
  3. Create Root Reducer
  4. Configure Store
  5. Instantiate Store
  6. Connect Component
  7. Pass props via connect
  8. Dispatch action
* 4 Steps to use a feature
  1. Create Action
  2. Enhance Reducer
  3. Connect Component
  4. Dispatch Action

Redux Async Libraries:

* Redux-thunk
* Redux-promise
* Redux-oberserable
* Redux-saga

Not sure if this is really need in our project. To simply confusion with our complex application, we should only update the store based on the page vs inside an API call.

For example, there are times when a Redux store is required to remain static like the enrollment process tabs, and let hooks handle the benefit selection process to handle logic. When a tab is added to the process, that logic should be handled at the page to update the Redux store. MWP is currently updating the store blindly causing many hits just because it can, and I find this redundant and inefficient waste of resources.

Testing:

* Jest – FaceBook supported Framework
* Mocha
* Jasmine
* Tape

Helper Libraries

* React Test Utils
* Enzyme
* React Testing Library