There are many flux implementations available in the market but the redux has been the most popular one in the libraries.

* All the states of the application are stored in an single state which acts as an single store.
* To setup the redux application, we require less boilerplate than any other application.
* All the components are automatically subscribed to the redux store. So, we don’t have to attach create emitter to be subscribed to the dispatchers.
* The redux can also be used in the server-side rendering of the application.
* Redux uses the immutable store, which has a number of benefits with the application
* It also contains the hot reloading feature which helps to instantly see the changes without using the client- state.
* Time-Travel debugging which will help us to go backward and forward to test and debug the application.
* The redux is also very light to handle.

In the flux, we have to write a large amount of boilerplate and redux just handles the all the boilerplate and provide with solution but we don’t get to see the redux in detail working.

In the redux, when an action occurs, the dispatcher is going to notify the store whether there are any actions have occurred.

When the store changes, any components which are listening to the store for the data automatically change. The user interacts with the UI and the unidirectional flow starts all over again.

Think of action as deleting the user,when there is an change in the UI. The dispatcher is going to notify the change to the store and the store handles the state and deletes the specific user. The react just reacts to the UI.

Action:

* Actions could be as creating the user, deleting the user and working with the user.
* They are triggered by the user interactions
* The actions act as an payload for the dispatcher.
* Action creators are also known as the dispatch helper.

The actions can be triggered in two ways and they are when there is an change in the UI and on the page load, on errors during calls to the server.

The payload is similar to an object, it contains an type and data inside it. The names of the object can vary and they don’t have to be the same.



Dispatcher:

* All the data flows through the dispatcher, it acts an like an central hub for the application and there is only one dispatcher for the whole application.
* It holds a list of callbacks
* Stores register with the dispatcher, so they can be notified if the state changes.
* The dispatcher invokes the callbacks that have been registered with it and then broadcasts the payload that is received from the action.
* This delivers the actions to the relevant store.
* Each action updates specific stores based on the callbacks that are registered with it

We are going to save all the actions in an constants file

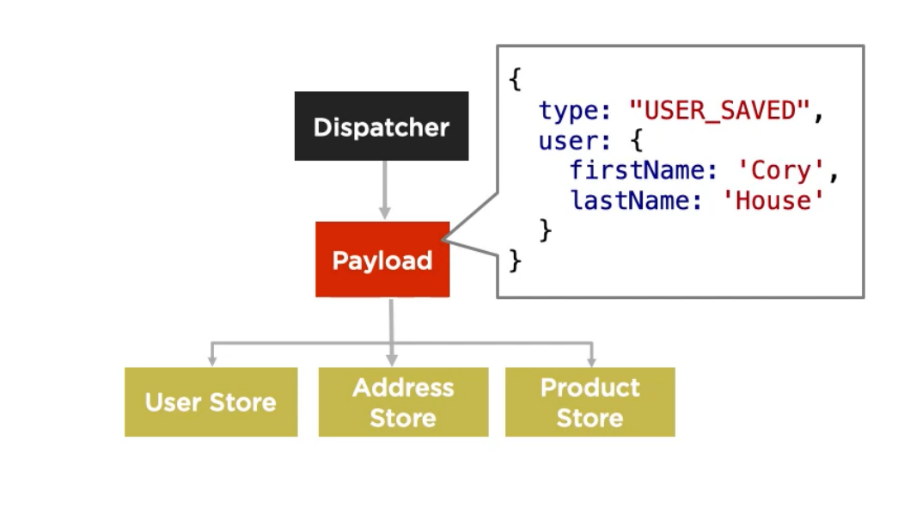
Store:

* The store is where all the data has been saved, the state logic and the data retrieval methods
* The store is not an model but it contains models.
* We can either have an single store or many
* The stores are updated because they have the register callbacks with dispatchers.
* The application should never be registered with the dispatcher. It should only be registered with the store
* The flux uses the node’s event emitter. This allows our store to both listen and broadcast the events to the application
* Only the store can update the data
* When the store is updated, they emit a change event so react can get the data

The structure of a store:

* Extend EventEmitter //So the store can emit the events to the react application.
* We expose methods for adding the data and removing the data(addChangeListener & removeChangeListener)
* EmitChange// This is used for emitting the change.

The dispatcher tells the type of action and based on the action; the store is going to handle the payload. We can also update the stores in an specific order.

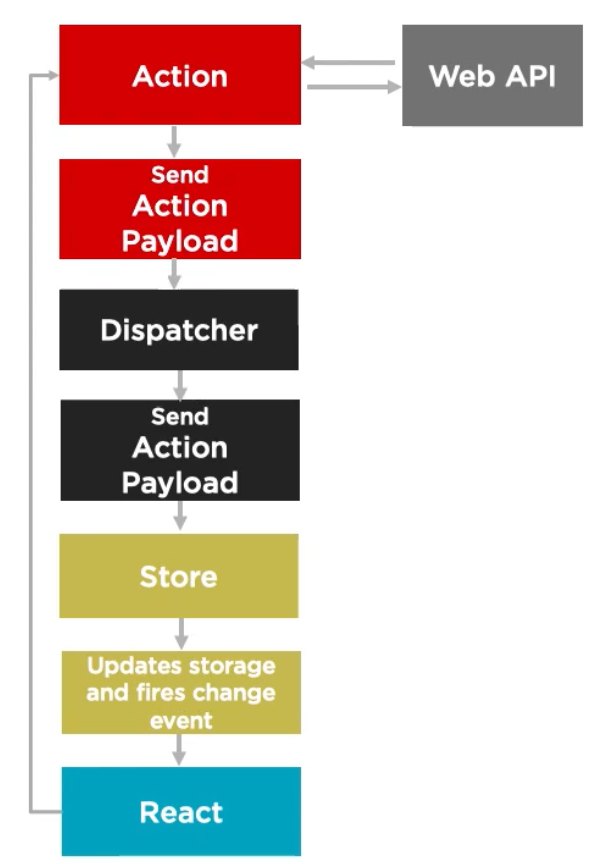


Controller View:

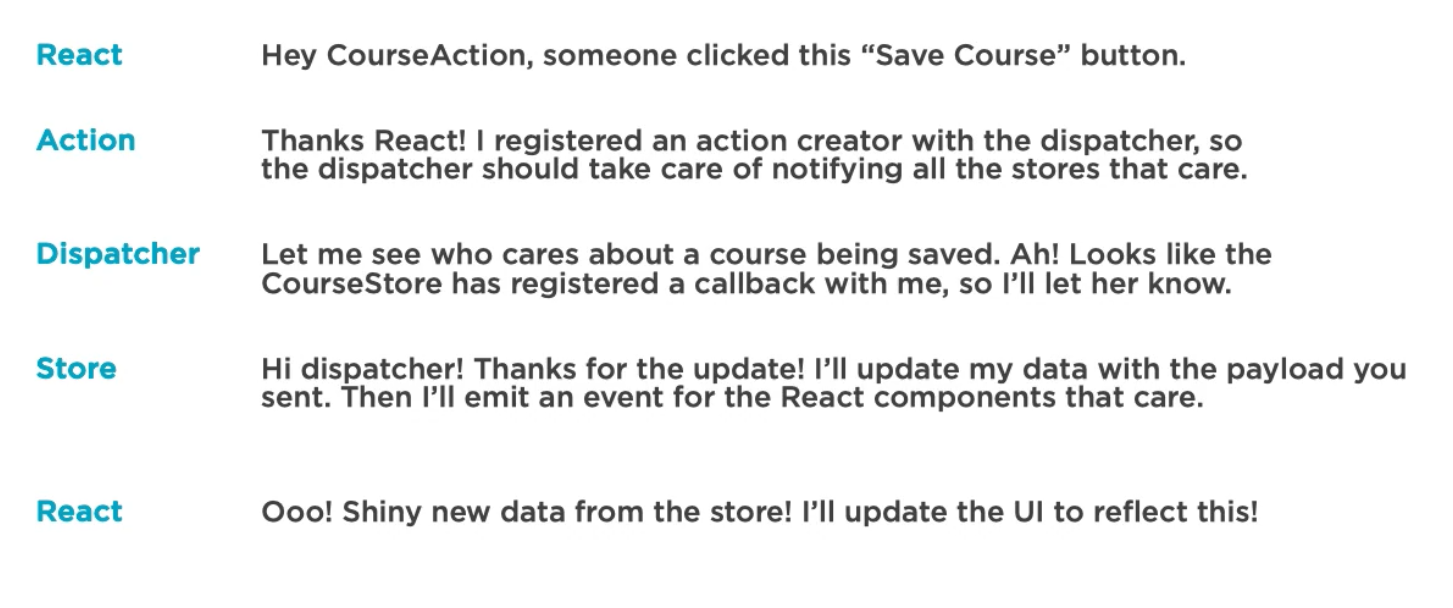
* The controller view is like an root view and it is going to handle the data which will sent to the child views.
* The controller view are the ones who interact with the stores
* When the stores receive an update, the update will be sent to the controller view and it has to send the update to the child view.
* Child components are updated by the help of the props.
* The controller view holds the data in the state and sends the data as the props to the children.

Example:

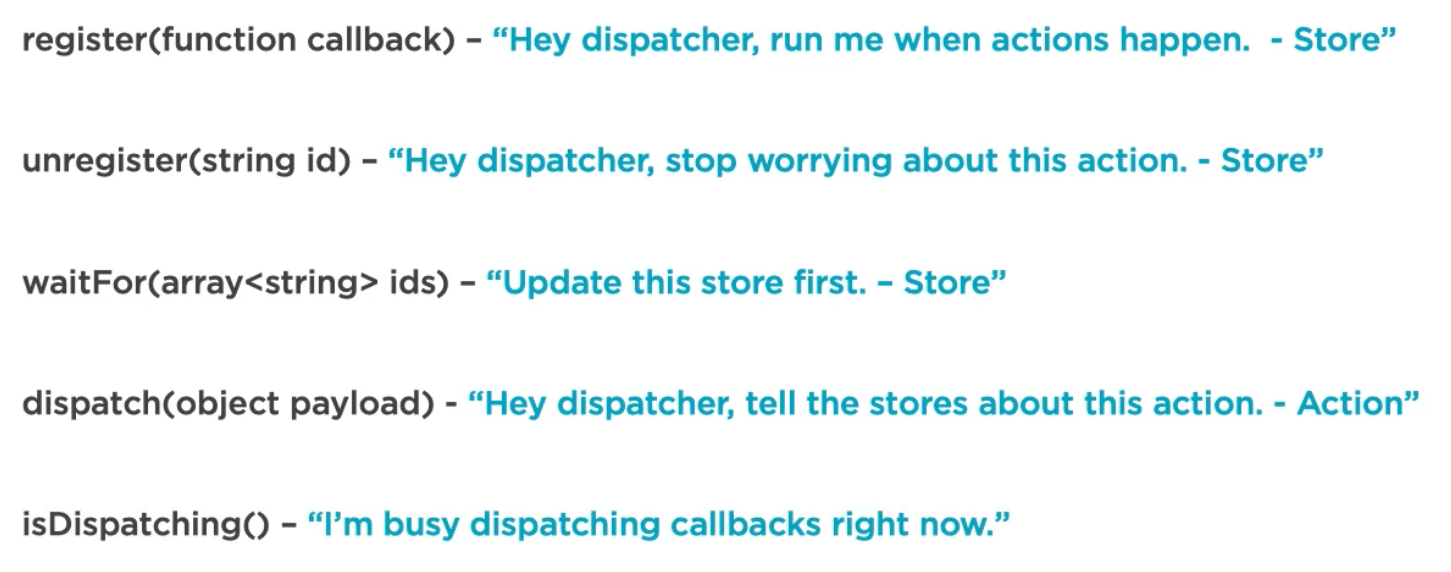
* First, the user is going to save the data which will create an action.
* The action with the type and payload will be sent to the dispatcher
* The actions also talk with the web api to fetch and get the data
* The dispatcher is going to check for any sort of registered call back for the action type
* Based on the action type, the dispatcher is going to send the payload to the specific store.
* The store is going to be updated based on the payload received to it.
* Once the store is updated, It updates the storage and fires the change event
* As the data is changed, the react is going to render itself.
* This process is going to be kept on repeated



A chat with the flux :



How does the flux API work:



Flux is not an publish-subscribe model:

* Every payload is sent to all the registered callbacks
* Callbacks can wait for other callbacks.

dwd