**REACTJS NOTES**

1.Components are javascript like functions that accept arbitrary inputs called Props and return react elements depicting what to display or view on the screen.

2.creation of a functional component:

Function Welcome(props){

Return<h1>Hello,{props.name}</h1>;

}

3.Props are read only. All react components must act like pure functions w.r.to props.

4.keys help react to identify which items have been changed/added/removed. They must be given to elements inside the array. They must be unique.(within the siblings level but not at the global level).

5.There are two types of form components in react:

Controlled and uncontrolled components.

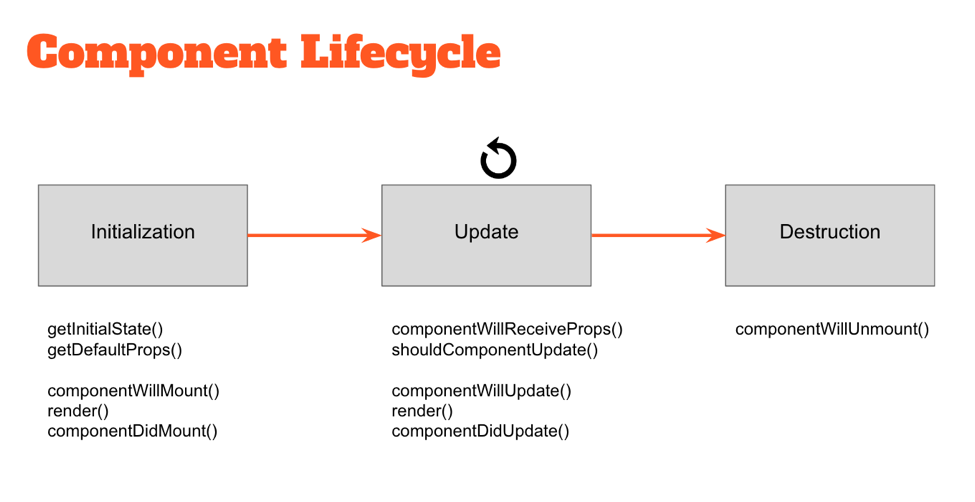
6.controlled form comp provides a value prop and doesnot maintain internal state;comp renders purely on props.

7.uncontrolled form comp are those that doesnot provide a value prop.:manages its own state.

8.composition is highly preferred to be used in react than inheritance in order to create the components in react.(specific component rendering a more generic one and configuring it with props—composition prop).

9. At the highest level, React components have lifecycle events that fall into three general categories:

1. Initialization
2. State/Property Updates
3. Destruction



essentially abstract methods — that can be utilized by any React component to more accurately manage updates.

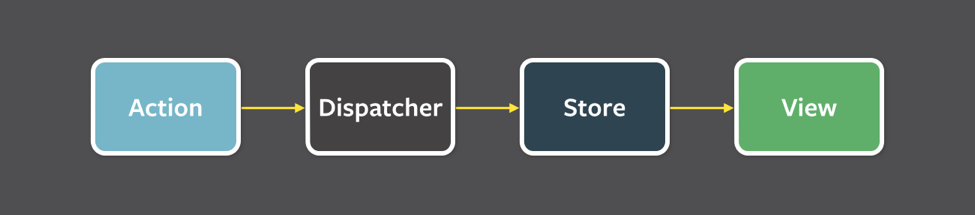
The events under “Initialization” only happen when a component is first initialized or added to the DOM. Similarly, the events under “Destruction” only happen once (when the component is removed from the DOM). However, the events under “Update” happen every time the properties or state of the component change.

JSX---JavaScript called JSX that embeds raw HTML templates inside JavaScript code. JSX code by itself cannot be read by the browser; it must be transpiled into traditional JavaScript using tools like Babel and webpack.It is declarative and reduces overall code complexity.

Flux is an architectural pattern that enforces unidirectional data flow — its core purpose is to control derived data so that multiple components can interact with that data without risking pollution.The Flux pattern is generic; In the Flux pattern, the Store is the central authority for all data; any mutations to the data must occur within the store. Changes to the Store data are subsequently broadcast to subscribing Views via events. Views then update themselves based on the new state of received data.

To request changes to any Store data, Actions may be fired. These Actions are controlled by a central Dispatcher; Actions may not occur simultaneously, ensuring that a Store only mutates data once per Action.

The strict unidirectional flow of this Flux pattern enforces data stability, reducing data-related runtime errors throughout an application.



Flux pattern complex UIs no longer suffer from cascading updates; any given React component will be able to reconstruct its state based on the data provided by the store. The flux pattern also enforces data integrity by restricting direct access to the shared data.

With a Flux architecture, the Store data is similarly shared between multiple Views. However this data can’t be directly mutated — all of the requests to update the data must pass through the Action > Dispatcher chain first, eliminating the risk of random data pollution. When updates are made to the data, it’s now much easier to locate the code requesting those changes.

UI components in AngularJS typically rely on some internal $scope to store their data. This data can be directly mutated from within the UI component or anything given access to $scope — a risky situation for any part of the component or greater application which relies on that data.

By contrast, the Flux pattern encourages the use of immutable data. Because the store is the central authority on all data, any mutations to that data must occur within the store. The risk of data pollution is greatly reduced.

Testing using react---Developers can recreate and test the state of any React component by simply updating the store — direct interactions with the UI (with tools like [Selenium](http://www.seleniumhq.org/projects/webdriver/)) are no longer necessary in many cases.

* [Redux](http://redux.js.org/): perhaps the most popular Flux library today.
* [Alt.js](http://alt.js.org/): another popular library for managing data in React applications.
* Stateless components----- (a flavor of “reusable” components) are nothing more than pure functions that render DOM based solely on the properties provided to them.
* const StatelessCmp = (props) => {
* return (
* <div className=”my-stateless-component”>
* {props.name}: {props.birthday}
* </div>
* );
* };
* // ---
* ReactDOM.render(
* <StatelessCmp name=”Art” birthday=”10/01/1980” />,
* document.getElementById(“main”)
* );
* As you can see, this component has no need for any internal state — let alone a constructor or lifecycle handlers. The output of the component is purely a function of the properties provided to it.