**React :**

React is a JavaScript Library for building UI for web applications, mobiles applications with react native and desktop applications using electron.

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.

Why?

* UI state is difficult to manage because of the DOM traversal, picking the right element and putting the required data into the element is way cumbersome for enterprise applications.
* Focuses on business logic instead of DOM traversal.
* Active community, huge eco system and high performance we applications.

**Using a Build workflow**

Recommended for SPA and MPA,

Why?

* optimize our code (bundling, building, use),
* Next gen features while development (Less error prone and better readability, JSX support)
* More productive (CSS auto-prefixing, linting)

How?

* Need Dependency management NPM or Yarn (ract-dom, react, babel)
* Use bundler (for concatenating, bundling of JS and CSS, Webpack)
* Use compiler for NextGen JS (Babel + presets)
* Use development server acts as web server for local (HTTP Server)

For building this workflow as a developer we need lot of time, instead just we can refer facebook’s create-react-app github repo for the details, and follow the instruction below steps

* Npm install -g create-react-app
* Create-react-app <project-name> [for specific version of folder structure “—script-version <version-numbber(1.15)>”]
* cd <project-name>
* npm start

**Context API:**

The only issue with property binding is, there were no possibility of sharing the data between the components without having to know in between components in component tree. Consider component tree with 5 to 6 levels and the 1st component has some data to share with the 6th component. In this case property binding has to pass the data through all the levels of components, instead this by using this context API there is a possibility of sharing directly to the child. Set the state from somewhere at the tree level and utilize it at anywhere in the child hierarchy.

In react >=16.6 versions introduce a static property contextType, through this we can assign the Context to the variable before render method and use it across the component. The older versions doesn’t have that capability to do that.

Through this it will create a private variable context in that component to use the context’s state throughout the component.

**Routing:**

Routing can be done by installing using react-router-dom node module as a dependency to the project.

React-router-dom is dependent on react-routing package.

**NOTE:** These two packages were not implemented by Facebook, but we use these packages as a standard.

By default, if we specify just path in Route component, it means the path starts with. It will be useful when we have child paths. To eliminate this or if we want just to load only for the path use ‘exact’ attribute to load only for that path.

If we <a/> anchor tags for the links, react treats those as anchors and it will reload the page when we perform any click, instead we can use <Link/> component from ‘react-router-dom’ to use react’s rendering behavior.

**NOTE:** By default, all paths specified in “to” attribute will be considered as absolute paths, to make it as relative paths we have to use the router properties props.match.url.

General behavior of the router links is to show activated link with different styles, for that we have to use separate component <NavLink/>, it comes capability of configuring active class with “activeClassName” prop and styles with “activeStyles” prop. By default, the “active” class will be populated to represent the active link.

By default, the router sets some properties to the routed component which was mentioned in Route component and these properties won’t be forwarded to the components in the hierarchy. To pass the router props to any of the component in the hierarchy we can use “withRouter” HOC provided by the react-router-dom, it will bind the router properties to the component properties.

For nested routes, just create use <Route/> component under the specified route component. But remember, should not use “exact” attribute for parent route in case of nested routes, because its checks the exact route first and then renders the component, in this case it fails in the first step and never goes to the step 2 means it won’t render the component, without rendering the routing logic won’t work at all.

**NOTE:** React doesn’t render the child route’s component entirely, so be caution about the child route’s data update, we might have to use “componentDidMount” and “componentDidUpdate” lifecycle methods both together.

If we want to redirect to any page, we can use <Redirect/> component and it has two attributes one is from=”” and another is to=””. Inside a switch statement we can use from and outside just “to” and it just redirects when it is rendered, So mostly we can use it when conditional redirecting. And we can redirect the page using route’s history API to push or replace the state and it act’s accordingly.

**Router Guards:**

Provides an access to the routes