**React tutorial**

## ***What is React?***

* JavaScript library for building user interfaces --> view layer for web applications
* heart of all React applications are **components -->** self-contained module
* components inside higher-level components define structure of our app
* Element(input fields, labels or buttons) can be written as a React component.
* The form component would specify the structure of the form and include each of these interface elements inside of it.

## ***How does it work?***

* React operates on a **virtual DOM** not in browser’s **DOM**(Document object m)
* [React Virtual DOM](https://facebook.github.io/react/docs/dom-differences.html) exists entirely in-memory and represents browser's DOM
* Because of this components not written in DOM directly but in react virtual DOM ,that react turns it into DOM.

## ***JSX/ES5/ES6***

* ES5 (ECMAScript) is basically "regular JavaScript” **-->** 2009
* ES6(2015) **-->** new syntactical and functional additions **-->** new version of JS
* ES5 and ES6 [fully supported](http://kangax.github.io/compat-table/es6/) by all major browsers
* BABEL is the library to transpile ES6 into ES5
* **JavaScript eXtension**, or more commonly **JSX**, is a React extension that allows us to write JavaScript that *looks like* HTML.
* render function specifies the HTML output of our React component **-->** returns JSX **-->** then JSX is *translated* to regular JavaScript at runtime **-->**all elements specified will convert into createElement() after translation
* render() function is expected to return a virtual DOM representation of the browser DOM element(s).

## ***How to create component – FIRST REACT APP***

* ReactDOM.render(<what>,<where>) places React application on the page.
* Every component extend the React.Component
* In React we use className instead of Class in a tag
* ***Complex Components***
  + Rather build a single component, break it down multiple components
  + start looking at applications using the idea of *components*

## ***Child components***

* + component nested inside another component is *child* component.
* {/\* This is a comment in React \*/} to write comment in JSX

## ***Data-Driven-Components***

* set up components to be driven by data
* Cause its not very convenient to have to update our component's template every time we have a change in our website's data.

## **Introducing props**

* sending data using attributes or *properties* on a component.
* We can access these properties inside a component as this.props
* we can pass multiple props into a component.
* We can also create a javascript object and can pass it as props.
* Destructuring allows us to save on typing and define variables in a shorter way.

***CONSTRUCTOR***

* first line of the constructor should *always* call super(props)

## ***State***

* sometimes a component needs to be able to update it's own state
* Example - updating a timer on a stopwatch
* state is intended to be completely internal to the Component and it's children
* accessed via this.state
* Whenever the state changes (via the this.setState() function), the component will rerender.

**REACT COMPONENT LIFE CYCLE**

A React Component can go through four stages of its life as follows.

{

* **Mounting** –Birth of your component
* **Update** – Growth of your component
* **Unmount** – Death of your component

}

* **Initialization:** This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.
* **Mounting:** stage of rendering the JSX returned by the render method itself.
  + **componentWillMount() -->** invoked once before the render() function is executed
  + **componentDidMount() -->** invoked right after the component is mounted on the DOM
  + **shouldComponentUpdate()**
* **Updating:** Updating is the stage when the state of a component is updated and the application is repainted.
  + **componentWillRecieveProps() -->** invoked before a mounted component gets its props reassigned.
  + **componentWillRecieveProps() -->** invoked before a mounted component gets its props reassigned
  + **componentWillUpdate()**
  + **componentDidUpdate()**
* **Unmounting:** final step of the component lifecycle where the component is removed from the page.
  + **componentWillUnmount()**

## ***Packaging and PropTypes***

* PropTypes for the set of types I.e object or string which allow us to expose a component API.
* Usually used to check value types before passing values.
* Proptypes need to be imported first for ex: import PropTypes from ‘prop-types’;
* Syntax : Component.propTypes = { KEY : VALUE(Type of that key)}
* PropTypes.(stirng, number, bool, func, symbol, object)
* PropTypes.(array, arrayOf([type]), one of([arr]))
* Communication through parent to child component and vice versa.
* Its possible to make a prop required using **PropTypes.name.isRequired.**
* Default value for props can also be set using Component.defaultProps = {key:values}

## ***Styles***

* CSS
* Inline
* Styling libraries
* Styling library – **Radium** (To define common style outside of the react component)

## ***Interactivity***

1. USER INTERACTION
   * Event fired by browser can be catch by react function.
2. PreventDefault() function is used after calling the submit method which in return stops the default behavior of the browser.

## ***Calling parent method from child***

* We can call parent method using child with the use of props, we supposed to send method in props and just need to bind that method.

## ***Pure Components***

* Pure stateless component using normal Javascript functions
* It can replace component that only has render function
* If react knows the **props** sent to a component, then it can determine when to re-render component.
* Props in pure component could be send as same we send it in normally and catch it as args.
* Advantages of using functional components are :
  + No this keyword (I.e no binding)
  + No heavy lifting of component like constructor, state, life-cycle etc
  + Its speedier, encourages building small components, self-contained components
  + Easy to reuse,

## ***Installing packages***

* **Npm install –global create-react-app** to install react command
* **Create-react-app app\_name && cd app\_name** to create react app and getting inside it
* **Npm start** to start app in localhost 3000
* **Npm run build** will create minified, optimized version which we can upload in server.

## ***Repeating elements***

* We can use native iterators like map, forEach in jsx by binding them inside {}
* **props.children** 
  + Used to display whatever you include b/w opening and closing tags on component
  + render () {  
     return (  
     <div className='container'>  
     <Picture key={picture.id} src={picture.src}>  
     //what is placed here is passed as props.children   
     </Picture>  
     </div>  
     )  
    }
* **react.cloneELement** 
  + it takes element, sends new props to it and is able to insert children components also.
  + Syntax -: React.cloneElement( element, [props], [...children])
  + New children will replace the existing children. props will shallowly be merged,

## ***Introduction to APIs***

* **Querrying for remote data**
* **FETCH**
  + **Npm install --save whatwg-fetch**
  + To make a request to an offsite server
  + We can make a request using fetch() api,
* **Promises**
  + Used for handling asynchronous computations, It represents the eventual completion or failure of an asynchronous operation and its resulting value
  + Promise provide a method which handles the object when it is instantiated,
  + Handling value after it is known (resolved), or if unavailable (rejected)
  + It helps to associate functionalities for asynchronous operations
  + it has three states I.e. **pending, fulfilled, rejected**
  + **Chaining** is also great thing about promise,
  + **Promise example :**
  + <script>

function printName(){

var x = Math.random();

var result = x>0.5;

document.write(x+"<br>");

**return new Promise ((resolve, reject) =**> {

result? resolve("Resolved") : reject("rejected")

});

}

printName().then(result => printName()).then(result => {

document.write(result);

return true;

})

. catch(err=> document.write('The error is : '+err))

</script>

* + In the above example printName() is been called and the return value will be stored in result variable and then it will pass to the next called function after the then() and so on

## ***Displaying remote data***

* **Import whatwg-fetch;** has the fetch() method which is used to fetch data from APIs
* To handle API requests fetch() accepts URL, that URL contains the api address which returns the json data which can further be used in react application.
* Bringing remote data into our app and handling it is