React JS 17

By

Anand Kulkarni

anand.pune38@gmail.com

Contents

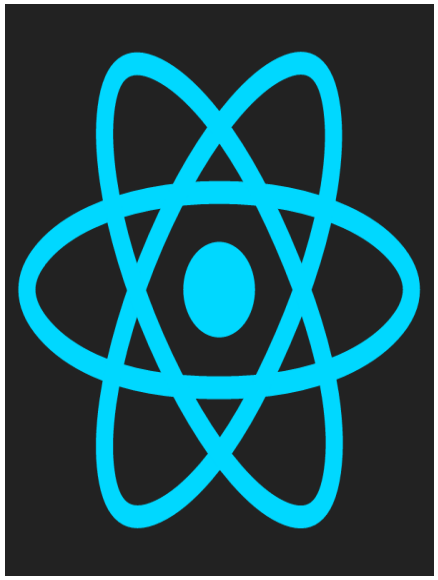
Module	Topic
Module 1	Introduction to React
Module 2	Developing first React application
Module 3	React components
Module 4	React component with properties
Module 5	State of a component
Module 6	Component life cycle
Module 7	Component event handling
Module 8	Component references
Module 9	Component children handling

What is React?

- React is an open source javascript library for building user interfaces.
- It is maintained by Facebook, Instagram and a community of individual developers and corporations.
- React helps us to implement Single Page Application (SPA). Here, react takes care of only 'View' part.
- React builds a view using several reusable components.
- React component encompass both logic & view inside.
- React's homepage is located at http://facebook.github.io/react/
- React is presently being used by Facebook, Instagram, Netflix, Alibaba, Yahoo, E-Bay, Khan-Academy, AirBnB, Sony, Atlassian etc.

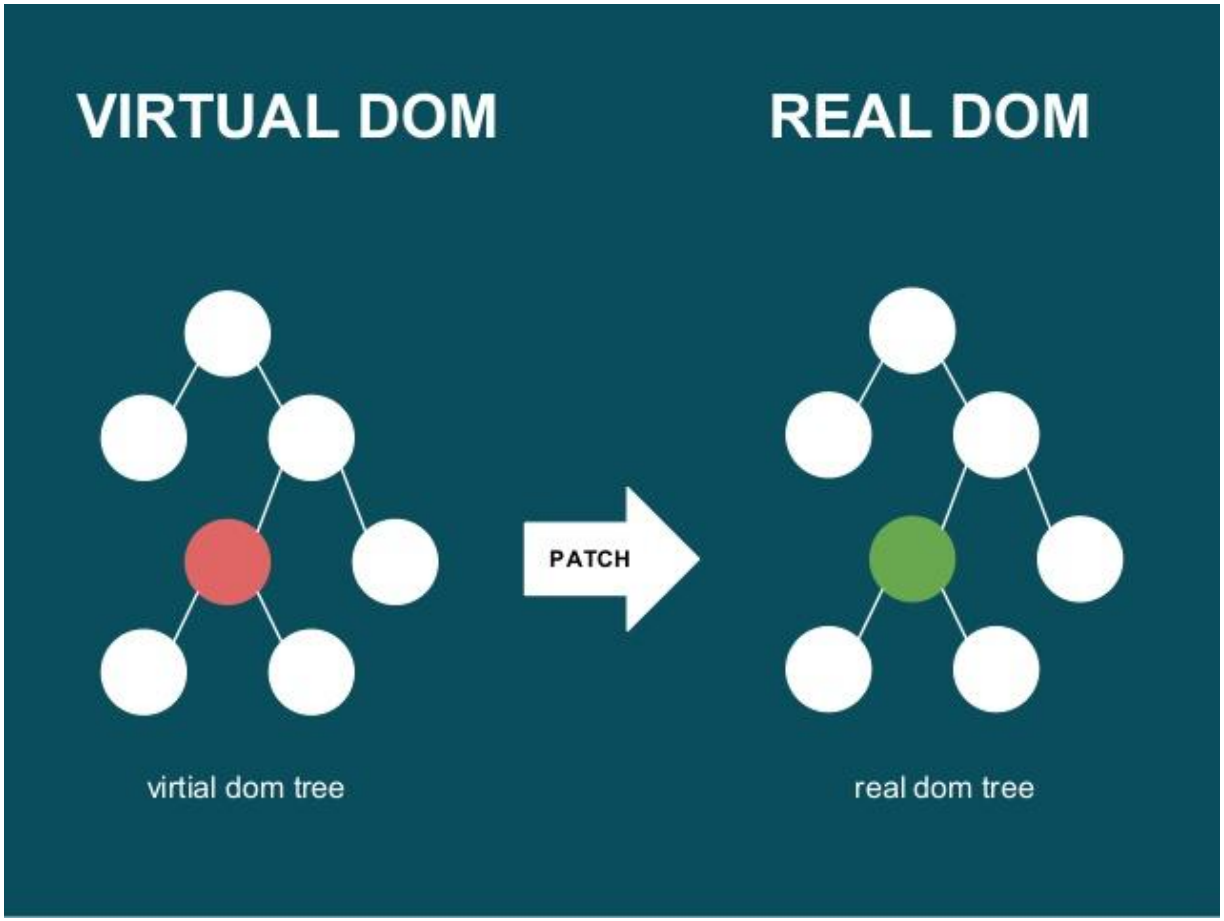
React History

- React was created by Jordan Walke, a software engineer at Facebook.
- Jordan used XHP, an HTML component framework for PHP to develop React.
- React was initially released in March 2013.



Why React?

React provides efficient UI rendering using virtual DOM. Hence, any application that demands frequent UI changes should be implemented using React.



Setup React Environment

Install Node.js (https://nodejs.org/en/download/)

Install ‘Visual Studio Code’

(https://code.visualstudio.com/download)

Developing first React application

Developing first React application

Please run below commands on your console to create a react application ‘my-app’:

1. npx create-react-app my-app
2. cd my-app
3. npm start
4. Run http://localhost:3000 in the browser.

•*‘npx’ is a package runner tool that comes with npm installation itself.*

•*If ‘npx’ command is not working then run below 2 commands:*

npm install -g create-react-app

create-react-app my-app

Developing first React application

5) Create ‘components’ directory in *‘src’* & write hello.js inside it:

```
import React from 'react';

export class HelloComponent extends React.Component {

    render() {

        return <h1>Hello first react app...</h1>

    }

}
```

Developing first React application

6) Finally import the HelloComponent in App.js:

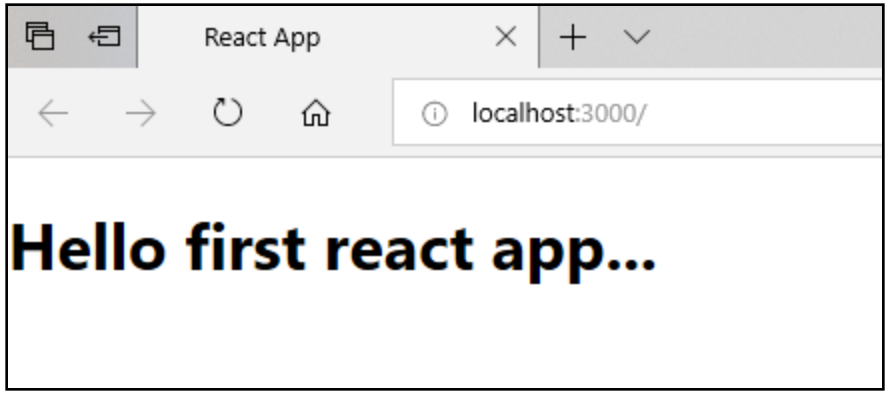
```
import React from 'react';

import { HelloComponent } from './components/hello.js'


export default function App() {

    return (<HelloComponent />);

}
```



Introduction to JSX

- JSX stands for JavaScript XML.
- JSX is fairly close to JavaScript especially in its statements and expressions.
- JSX gives better performance that JavaScript because JSX uses 'inline expansion'.
- You can use React without using JSX. But JSX makes it easy to build React components. It reduces the amount of code required to write.
- JSX looks like HTML markup. JSX syntax is simple and concise and it's very easy to visualize the components that are getting built.
- JSX code is transformed into JavaScript using a transpiler called ‘Babel’. Refer https://babeljs.io/repl/

React components

React components

- React component should either extend the class React.Component or say createReactClass({}).
- React component must have render() method where you describe the GUI you wish to render.
- React component tag name must start with capital letter.

First React Component

```
//app.js

import React from 'react'

class App extends React.Component {

        render() {

                return <h1>Hello World!!!</h1> //JSX code

        }

}

export default App
```

Rendering React Component

```
//app.js

import React from 'react' //ES6 feature

import {render} from 'react-dom' //ES6

import HelloComponent from './components/hello.js‘ //importing hello.js
export default function App() {
   return (<HelloComponent />);
}
```

React Component with properties

```
class HelloComponent extends React.Component {
        render() {
                return <h1>Hello {this.props.fname} {this.props.lname}!!!</h1>
        }
}
//app.js
export default function App() {
   return (< HelloComponent fname='Tom' lname='Cruise'  />);
}
```

Note: React also provides us *'this.props.children'* to retrieve all children of your tag.

Handling default properties

Using ‘defaultProps’ attribute you can specify the default values of properties.

```
class HelloComponent extends React.Component {
        render() {
                return <h1>Hello {this.props.fname} {this.props.lname}!!!</h1
        }
}
HelloComponent.defaultProps = { lname: ‘Bayross’ };
//app.js
export default function App() {
   return (< HelloComponent fname=‘Tom’ lname=‘Cruise’  />);
}
```

Output: Hello Tom Cruise

Handling property types

React provides a way to validate the props using PropTypes. This is extremely useful to ensure that the components are used correctly.

```
import PropTypes from 'prop-types';

HelloComponent.propTypes = {

        fname: PropTypes.string,

        lname: PropTypes.string

};

//app.js

<HelloComponent fname='Tom' lname='Cruise' /> //OK

<HelloComponent fname='Tom' lname=25 /> //Error
```

Possible property types

- PropTypes.string
- PropTypes.array
- PropTypes.number
- PropTypes.bool
- PropTypes.func
- PropTypes.object
- PropTypes.any
- PropTypes.element

Facts about property types

- If the property value is not matching with specified property type, then does not throw any error rather it only shows warning message in browser’s console.
- 'propTypes' are only checked in development. Their job is to just check that all the assumptions that we are making about our components are being met.

Facts about property types

- If you want to implement a custom logic for property validation then do the following:

```
HelloComponent.propTypes = {
        fname: function(props, propName, componentName) {
        if (props[propName].length < 3)
                return Error('First name must be at least 3
characters long');
        }};
```

State of a component

- Every component can have its own state in React.
- The main difference between state and props is that props are passed to the component from the parent component; whereas, state is something that is internal to the component.
- Props are passed when a component gets instantiated. State is something that can change over time. Therefore, changes in state affect the rendering of components.
- One can assign the value of props to the state.

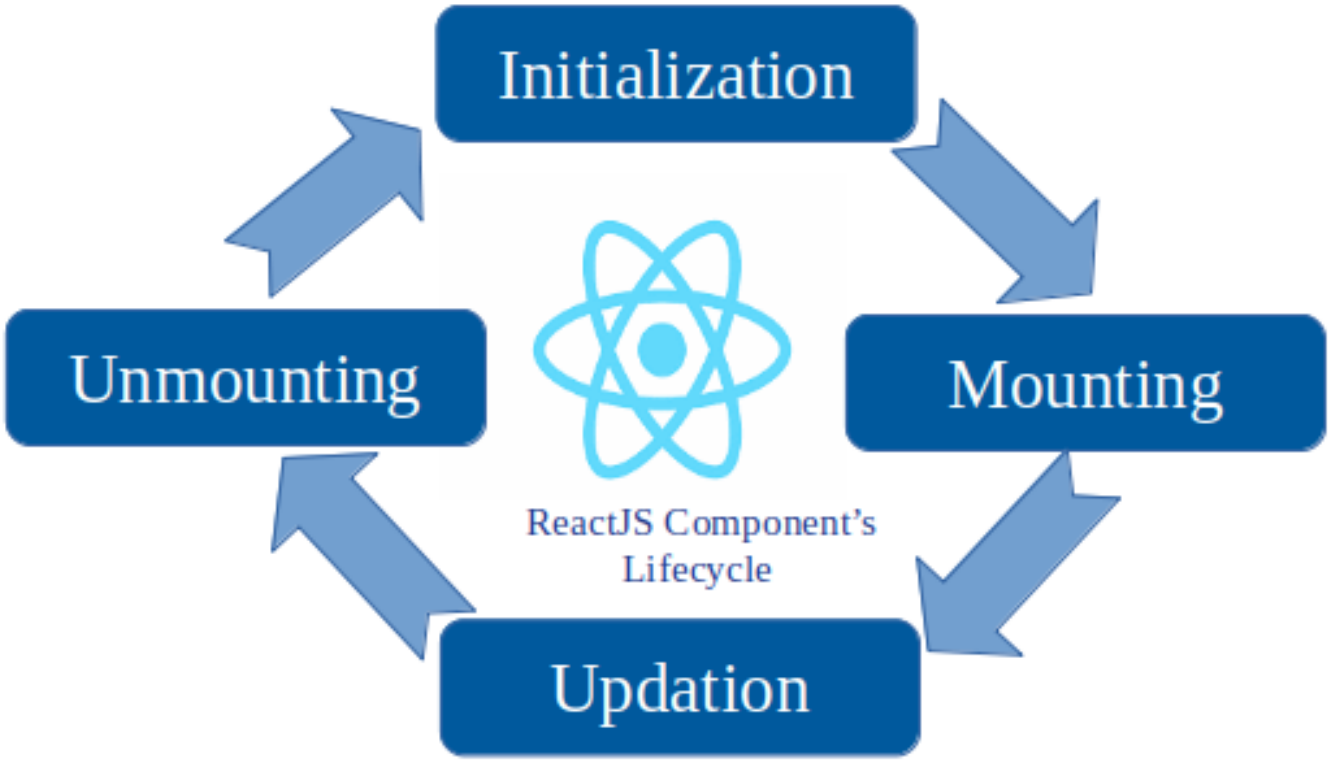
How to set component's State?

```
class App extends React.Component {
        constructor(props) {
                super(props);
                this.state = { count: 5, //setting up component's state
                        fname: this.props.fname //setting up component's
state using props
                };
        }
        incrementCount() {
                this.setState({count: this.state.count + 1}); //updating the state
        }
        render() {
                return <h1>Count = {this.state.count}!!!</h1> //Accessing the
state
        }
}
```

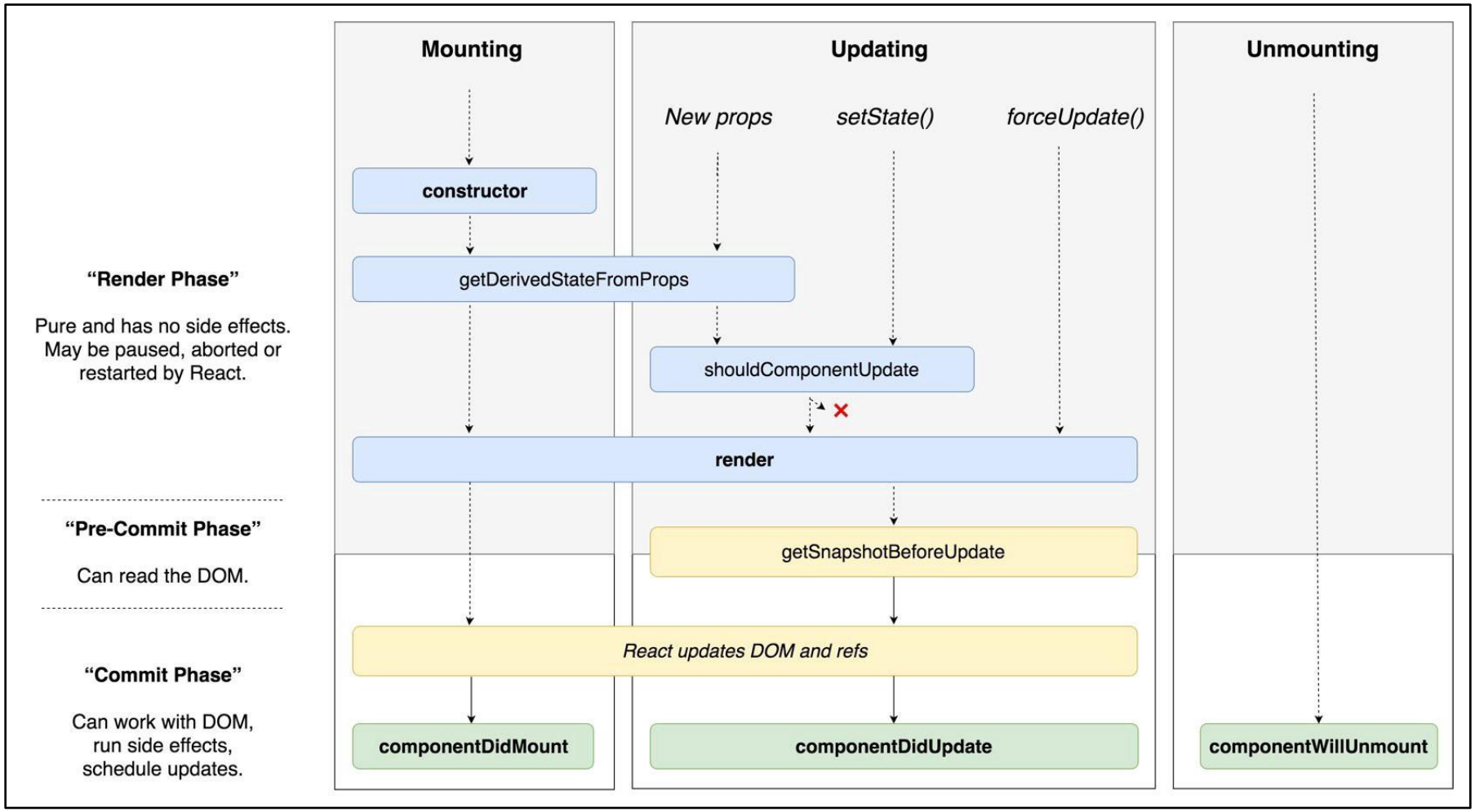
State vs Props

Sr.No.	Props	State
1.	Props are immutable. They should not be updated by the component to which they are passed.	State is mutable. State can and will change depending on the interactions with the outer world.
2.	Props are passed by parent component to child component.	State is something internal and private to the component.

Component Life Cycle



Component Life Cycle methods



Component Life Cycle methods

- **static getDerivedStateFromProps(nextProps, prevState)**

  It ensures that the state and props are in sync with each other.

- **render()**

  The render method is responsible for the actual component display.

- **componentDidMount()**

  This method is invoked immediately after mounting the component and only once after the component is rendered.

- **boolean shouldComponentUpdate(nextProps, nextState)**

  This method decides whether the component should be re-rendered or not.

- **getSnapshotBeforeUpdate(prevProps, prevState)**

  This method gets called immediately before the DOM is updated.

Component Life Cycle methods

- **componentDidUpdate(prevProps, prevState)**

  It is called just after a component is re-rendered.

- **componentWillUnmount()**

  This is called when the component is unmounted from the body. We can use this to release the resources, perform cleanups, unset any timers, and so on.

Component Event Handling

```
class App extends React.Component {
        action() {          alert("Hi " + this.props.fname);   }
        render() {
                return(<div>I am {this.props.fname}!!
                                        <a href="#"
onClick={this.action.bind(this)}>Who am I?</a>
                                </div>);
        }
}
```

Component References

Component references give us a handle to refer to an element of a component.

```
class App extends React.Component {
        action() {          this.refs.destinationTextArea.value =
this.refs.sourceTextArea.value;      }
        render() {                  return(<div>
                                    <textarea ref="sourceTextArea"
onChange={this.action.bind(this)}></textarea>
                                    <textarea
ref="destinationTextArea"></textarea>
                                    </div>
                );
        } }
```

Component Children Handling

Component children are handled using '*this.props.children*' attribute.

```
class App extends React.Component {
        render() {
                return(<div>

        <h1><center>{this.props.children}</center></h1>
                                </div>
                );
        }
}
```

Thank you!!