

EVIDEN

React JS

Basics Level 1

EVIDEN

Content Overview

01

Introduction,
Installation , Features

02

Component
development using JSX

03

Working with
Bootstrap and JQuery

04

Props, State and
Component Life Cycle

05

Hooks (useState)

06

Handling Events,

EVIDEN

Content Overview

07

Keys , List and
Conditional Rendering

08

Working with Forms,
Controlled Vs. Uncontrolled
Component

09

Performing REST API calls

EVIDEN

01 Introduction, Installation , Features

Introduction

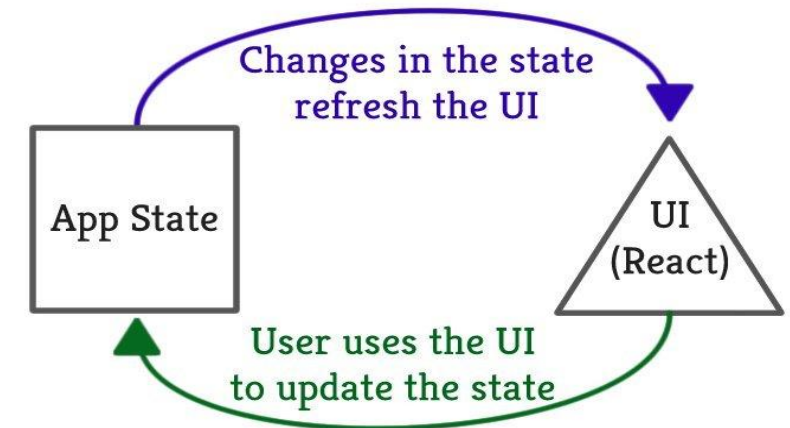
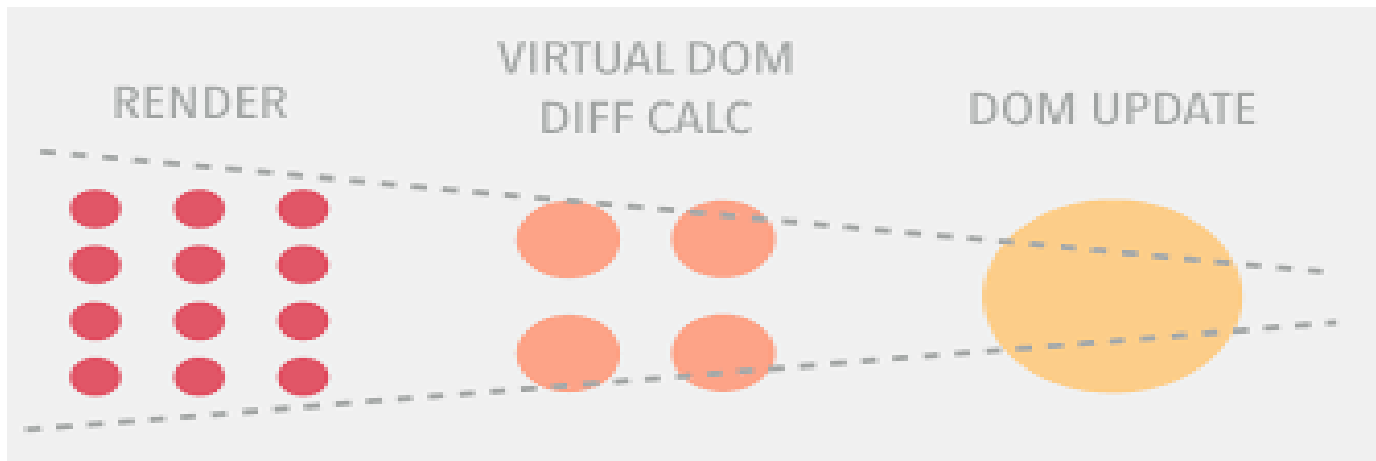
What is React?

- Developed by Face book in 2011
- Open Sourced in 2015 and has a large community
- **"A JavaScript Library for building User Interfaces"**
- Declarative, efficient, and flexible JavaScript library for building user interfaces
- compose complex UIs from small and isolated pieces of code called “components”.
- React implements one-way data flow which makes it easy
- Uses Flux is a pattern that helps keeping your data unidirectional.
- solve one problem: **building large applications with data that changes over time**
 - Extremely fast, because it uses a virtual DOM, and syncs only the changed parts with the underlying page

Introduction

Working of Virtual DOM

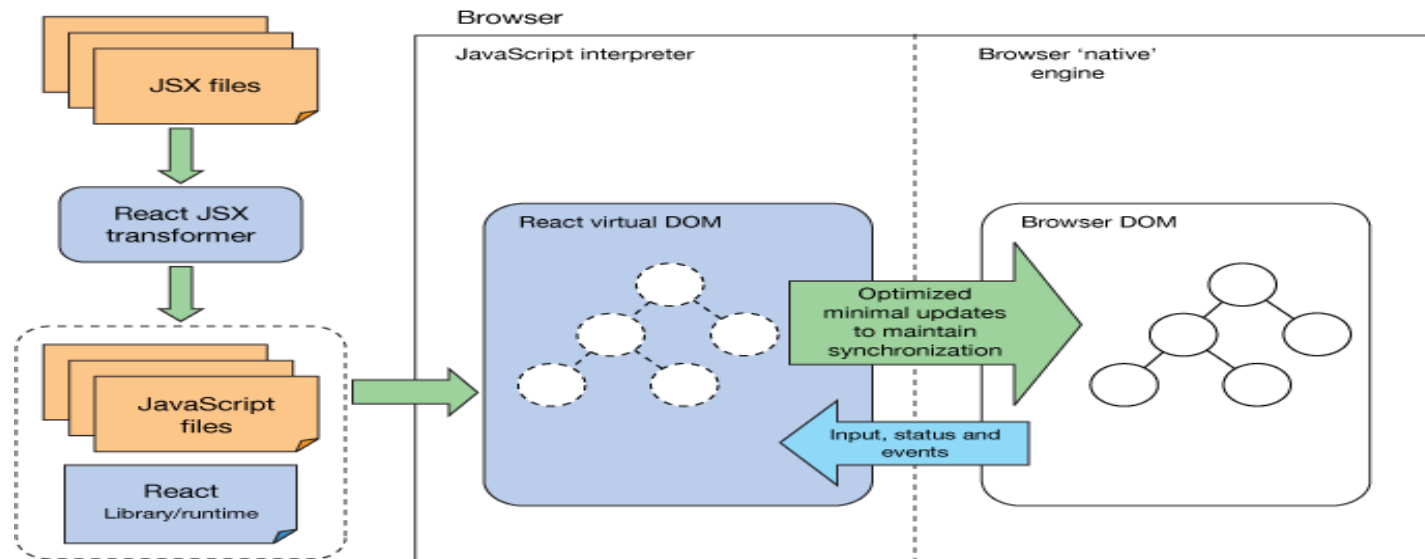
- step 1: Whenever anything may have changed, the entire UI will be re-rendered in a Virtual DOM representation.
- step 2: The difference between the previous Virtual DOM representation and the new one will be calculated.
- step 3: The real DOM will be updated with what has actually changed. This is very much like applying a patch.



Introduction

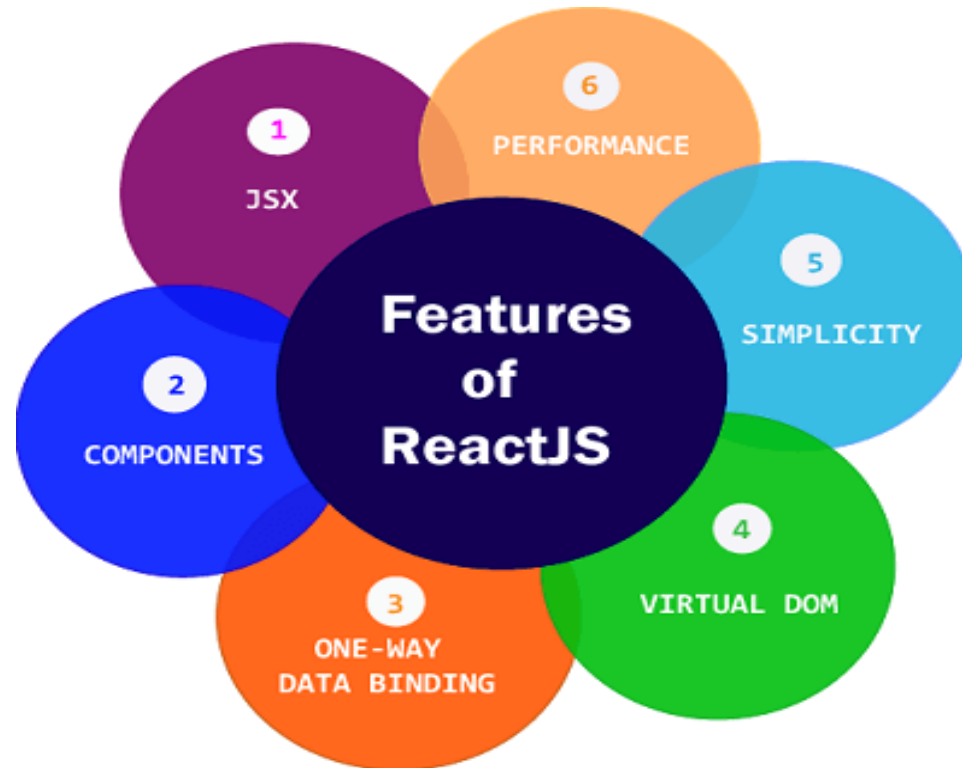
What is React JS?

- A front-end framework used by Facebook internally for managing user interface
 - Main function is to bind HTML elements to data and once this data changes, updating the interface so that you don't have to bind this data manually
- It is only the view layer and is all about modular, composable components
- Solve one problem: **building large applications with data that changes over time**
 - Extremely fast, because it uses a virtual DOM, and syncs only the changed parts with the underlying page



Introduction

Features



Introduction

Features

Overview

- **Advantages**

- Uses virtual DOM which is a JavaScript object. Improves apps performance, since JavaScript virtual DOM is faster than the regular DOM.
- Can be used on client and server side(With node js) also on the client side
- Component and data patterns improve readability, which helps to maintain larger

- **Limitations**

- Covers only the view layer of the app, hence you still need to choose other technologies to get a complete tooling set for development.
- Uses inline templating and JSX, which might seem awkward to some developers
- **React is a view layer library, not a framework like Backbone, Angular etc.**
- **You can't use React to build a fully-functional web app**

Introduction

Installation

- Step 1: **npm install -g create-react-app**
 - **Npx create-react-app**
- Step 2: **create-react-app reactproject**
- **NOTE:** We can combine the above two steps in a single command using **npx**. The npx is a package runner tool which comes with npm 5.2 and above version.
- **npx create-react-app reactproject**
- Start the server using the below command
npm start
- if we want to make the project for the production mode, type the following command. This command will generate the production build, which is best optimized.
- **\$ npm build**

Introduction

Installation

- NPM - Package manager
- NPX - Package runner
- `npm create-react-app my-app` executes the local `create-react-app` package from your machine, so you first have to install it globally on your system with `npm install -g create-react-app`.
- If you run `npx create-react-app my-app` and don't have `create-react-app` globally on your system, it will get downloaded and not installed globally. This is great if you don't want to pollute your system with global packages that you only run once every two months.
- Installing packages
 - `npm install --save react-bootstrap`

EVIDEN

02 Component development using JSX

Component development using JSX

What is JSX?

- JSX is an inline markup that looks like HTML and gets transformed to JavaScript
 - JSX expression starts with an HTML-like open tag, and ends with the corresponding closing tag. JSX tags support the XML self close syntax so you can optionally leave the closing tag off.
- It is a preprocessor step that adds XML syntax to JavaScript
- React can be used without JSX but JSX makes React a lot more elegant(it is a concise and familiar syntax for defining tree structures with attributes)
- It's actually a declarative syntax that's used to express the virtual DOM.
 - JSX gets interpreted and converted to virtual DOM, which gets diffed against the real DOM.
 - Rather than rewrite the whole DOM tree, only the differences get applied. That makes React renders fast.

Component development using JSX

What is JSX?

- Really just JavaScript with HTML in the render methods
 - Use className instead of class
 - Components are written in JSX
 - Helps in creating a virtual DOM by returning HTML
 - It is faster because it performs the optimization while compiling code into javascript
 - It is typesafe most of the errors can be caught during compilation
-
- Bindings:
 - Mustache syntax `{}`
 - Bind to props
-
- Note : JSX is optional

React vs Angular

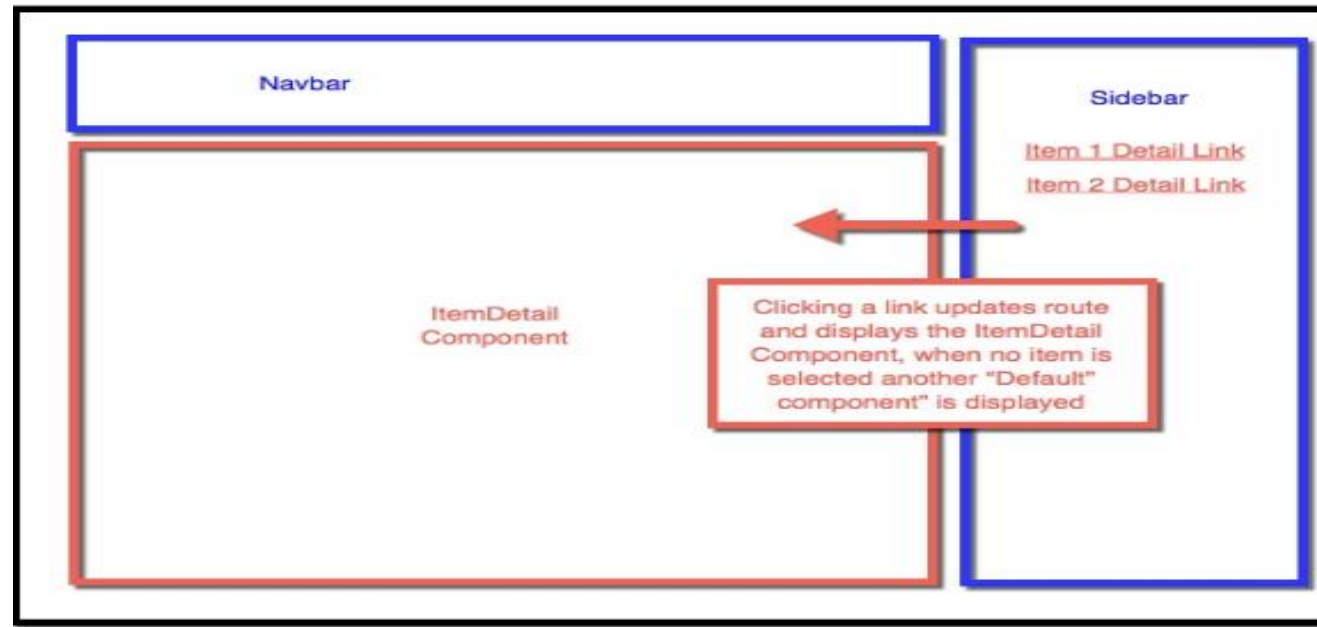
Technology	AngularJS	React
Developer	Google	Facebook
Technology type	Full-fledged MVC framework written in JavaScript	JavaScript library (View in MVC; requires Flux to implement architecture)
Concept	Brings JavaScript into HTML Works with the real DOM Client-side rendering	Brings HTML into JavaScript Works with the virtual DOM Server-side rendering
Data binding	Two-way data binding	One-way data binding
Dependencies	Manages dependencies automatically	Requires additional tools to manage dependencies
Language	JavaScript + HTML	JavaScript + JSX
Last version	AngularJS 1.6.0 RC2	React 16
Suits best	Best for SPAs that update single view at a time	Best for SPAs that update many views at a time

React

Components

- **Components**

- React is all about modular, composable components
 - Components are just like functions(Simple functions that take in “props” and “state”)
- React encourages reusable components which can be integrated into your project
- Example: A login system can be broken down into one root component with several child components.



Types of Components

- There are two types of Components
 - 1. Functional
 - --do not maintain state
 - --static data
 - 2. class based
 - --data changes over time
 - --dynamic data
 - --maintain the state data

EVIDEN

03 Adding Bootstrap to React Application

Adding Bootstrap to React Application

- Refer to the online documentation
 - <https://react-bootstrap.github.io/getting-started/introduction>

Step 1: From the command prompt move to your project folder and install the bootstrap package

- Adding bootstrap
 - D:\nodeprojects\reactlevel1>npm install react-bootstrap bootstrap
- Step 2: Adding stylesheet in index.html before </head>

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
integrity="sha384-Vkoo8x4CGsO3+Hhvx8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
crossorigin="anonymous" />
```
- Step 3: Now lets try applying style on a button
 - In App.js import the required Class eg Button

```
import { Button } from 'react-bootstrap';
```
 - Add some style on the Button in the div element using the variant

```
<Button variant="primary">Hello World Primary</Button>
```
 - ```
<Button variant="success">Hello World Success</Button>
```
  - ```
<Button variant="danger">Hello World Danger</Button>
```

EVIDEN

04 Props, State and Component Life Cycle

Props

- **React** allows us to pass information to a Component using something called **props** (stands for properties).
- **Props** are basically kind of global variable or object.
- We can pass props to any component as we declare attributes for any HTML tag. Have a look at the below code snippet:
`<DemoComponent sampleProp = "HelloProp" />`
- We can access any props inside from the component's class to which the props is passed. The props can be accessed as shown below:
 - `this.props.propName`;
- The 'this.props' is a kind of global object which stores all of a components props. The *propName*, that is the names of props are keys of this object.
- We can use the *propType* for validating any data we are receiving from props.
- Demo

- **Component Properties**

- Component properties are known as props
 - Props are the "arguments" to your components
 - React encourages reusable components which can be integrated into your project
- attributes are available in our component as **this.props** and can be used in our render method to render dynamic data and any nested elements as **this.props.children**

- **React** components has a built-in **state** object.
- The **state** object is where you store property values that belongs to the component.
- When the **state** object changes, the component re-renders.
- The state object is initialized in the constructor
 - **class Car extends React.Component { constructor(props) {**
 - **super(props);**
 - **this.state = {brand: "Ford"}; }**
 - **render() { return (<div> <h1>My Car</h1> </div>);**
 - **}**
 - **}**

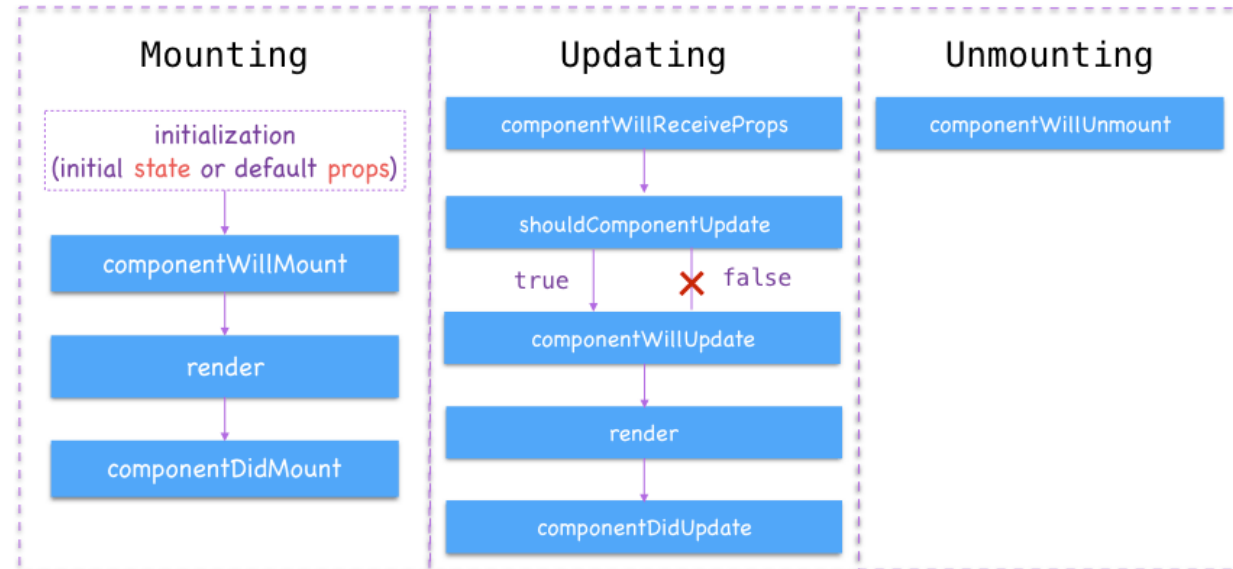
Component life cycle

- **Component Lifecycle**

- The render method is the only required spec for creating a component, but there are several lifecycle methods & specs
- They notify us of important milestones in our component's life, and we can use these notifications to simply pay attention or change what our component is about to do

- Can be categorized into 4 phases:

- Initialization
- Mounting
- Updation
- Unmounting



Components Lifecycle

- **Components Lifecycle**

- **Initialization**

- The initialization phase is where we define defaults and initial values for **this.props** and **this.state**
 - **getDefaultProps():**
 - method is called once and cached before any instance of the component are created.
 - method returns an object which properties values will be set on **this.props** if that prop is not specified by the parent component.
 - **getInitialState():**
 - method is also invoked once, right before the mounting phase.
 - return value of this method will be used as initial value of **this.state** and should be an object.

Components Lifecycle

- **Components Lifecycle**

- **Mounting**

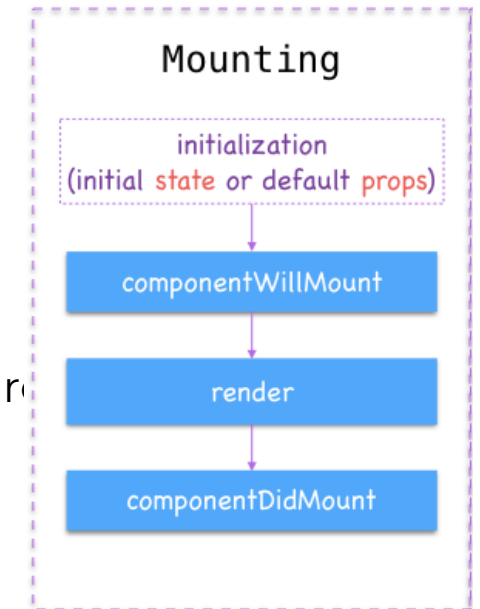
- **componentWillMount()**

- invoked once and immediately before the initial rendering occurs, before React inserts the component into the DOM

- **Note:** calling **this.setState()** within this method will not trigger a re-render

- **componentDidMount()**

- last step in the Birth/Mount life cycle phase
 - method is called once all our children Elements and our Component instances are mounted onto the DOM
 - is executed after the first render only on the client side. This is where AJAX requests DOM or state updates should occur.



Component life cycle

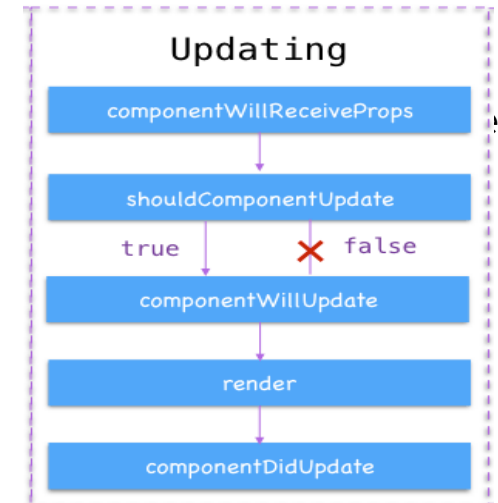
- **Components Lifecycle**

- **Updation**

- component updating phase is when an user trigger updations like state value trough a function executed on user events like onClick, onChange etc triggers a set of component state/props change usually and the component occurs updating the DOM accordingly

- **componentWillReceiveProps()**

- method is called when props are passed to the Component instance.
 - Here we could extract the new props and allows us to check and see if new props are coming in and we can make choices based on the data
 - Method can be skipped if the Update is triggered by just a state change.



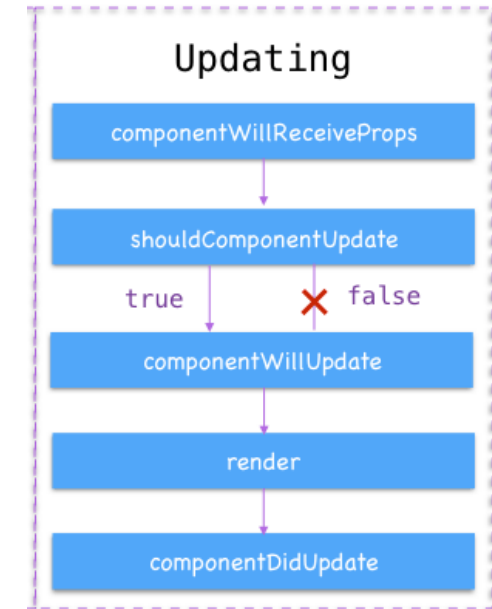
Components Lifecycle

- **Components Lifecycle**

- **Updating**

- **shouldComponentUpdate()**

- allows your Component to exit the Update life cycle if there is no reason to apply a new render.
 - Whenever a component's state changes, React re-renders the component and all its children. Often times, the component and its children barely change yet we need to render everything causing an inefficient code rendering.
 - method checks the current props and state, compares it to the next props and state and then returns true if they are different, or false if they are the same. Based on the Boolean value, rendering can be avoided when it isn't needed



Components Lifecycle

- **Components Lifecycle**

- **Updating**

- **componentWillUpdate()**

- is called just before rendering.

- method is passed in two arguments: nextProps and nextState
 - **method** is called every time a re-render is required, such as when **this.setState()** is called. Unlike **componentWillMount()** we get next props and state.
 - this method is called before **render()**. Because we have not rendered yet, our Component's access to the Native UI (DOM, etc.) will reflect the old rendered UI. Is executed before rendering, on both the server and the client side.

- **componentDidUpdate()**

- Useful when an operation needs to happen after the DOM is updated and the update queue is emptied
 - most common use case would be, updating the DOM in response to prop or state changes.



Components Lifecycle

- **Components Lifecycle**

- **Unmounting**

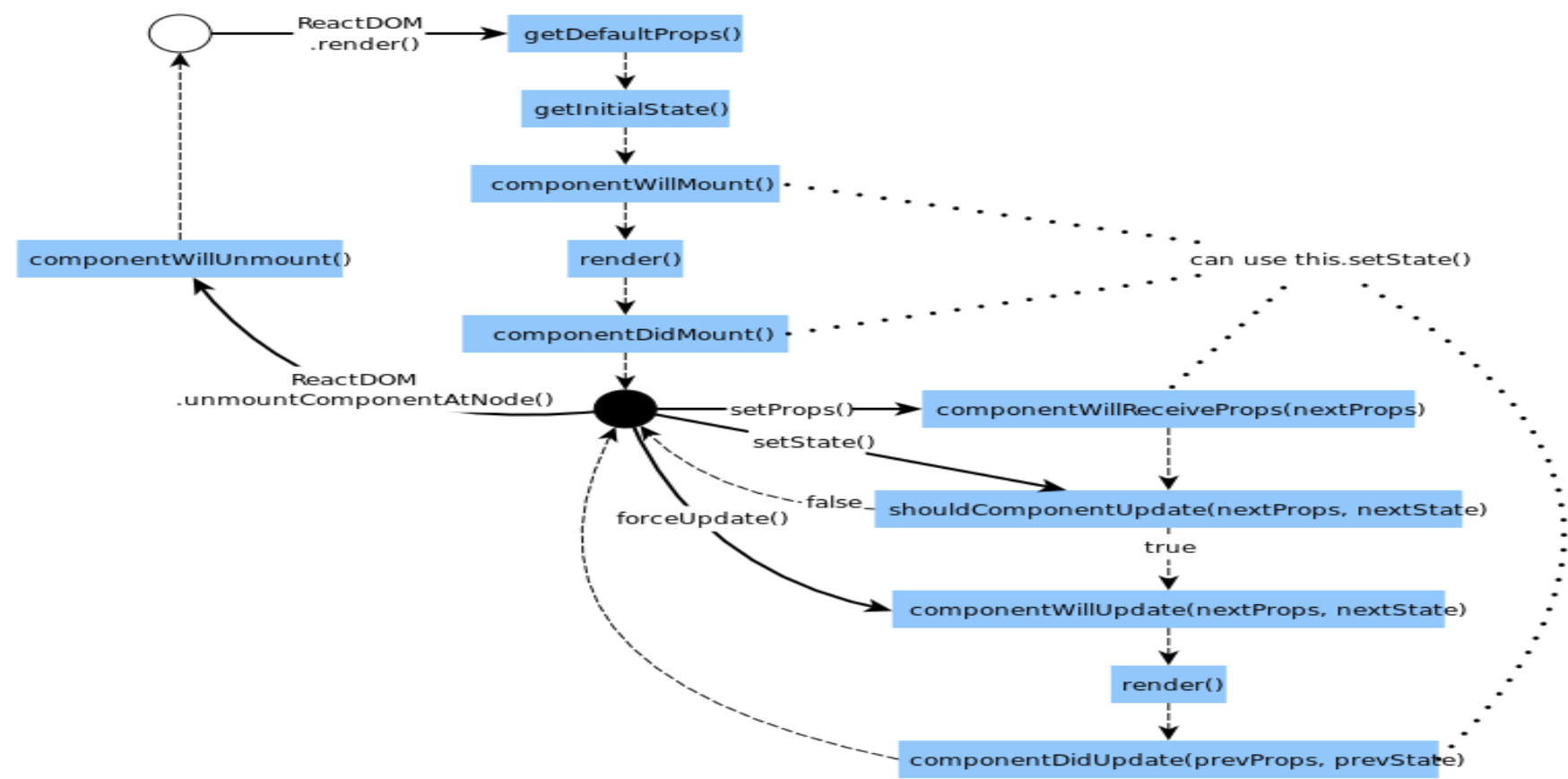
- react component undergoes the Unmounting Phase, while the component is deleted/removed from the DOM, and any last updates and functions to be executed while the component is being removed shall be done in the Unmounting phase

- **componentWillUnmount()**

- component is almost done and is going to be removed/deleted. Maybe forever or conditionally like show and remove.
 - Here you can cancel any outgoing network requests, or remove all event listeners associated with the component.
 - Basically, clean up anything to do that solely involves the component.



Complete flow



EVIDEN

05 Handling Events



Events

- **Steps in Event Handling**

- Step 1 Bind the event

this.increase = this.increase.bind(this);

- Step 2: Implement the event handler where we react to the event

- **Event Handler:**

```
increase(e) {  
  this.setState({  
    count: this.state.count + 1  
  });  
}
```

Step 3: Listen for the event on the Control.

<button onClick={this.increase} style={buttonStyle}>+</button>

EVIDEN

06 Hooks (useState)

Hooks

- **Hooks** are the new feature introduced in the **React** 16.8 version.
- It allows you to use state and other **React** features without writing a class.
- **Hooks** are the functions which "**hook** into" **React** state and lifecycle features from function components
- It does not work inside classes.
- Hooks are backward-compatible, which means it does not contain any breaking changes. Also, it does not replace your knowledge of React concepts.
- React Hooks installation
 - \$ npm install react@16.8.0-alpha.1 --save
 - \$ npm install react-dom@16.8.0-alpha.1 --save
- Hook uses useState() functional component for setting and retrieving state.
- Demo

EVIDEN

07 Keys , List and Conditional Rendering

Working with keys

- React **keys** are useful when working with dynamically created components
- Also used when when your lists are altered by the users.
- Setting the **key** value will keep your components uniquely identified after the change.
- Code snippet:

```
render() {  
  return (  
    <div>  
      <div>  
        {this.state.data.map((dynamicComponent, i) => <Content  
          key = {i} componentData = {dynamicComponent}/>)}  
      </div>  
    </div>  
  );  
}
```

Conditional Rendering

- In **React**, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state of your application.
- **Conditional** rendering in **React** works the same way conditions work in JavaScript.
- There are different methods to implement Conditional rendering
 - Using if..else
 - Using element variable
 - Using switch statement
 - Ternary operators
 - Using Enhanced JSX
- Demo

EVIDEN

08 Forms, Controlled Vs. Uncontrolled Component

Forms

- In HTML form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input.
- In React, mutable state is typically kept in the state property of components, and only updated with setState()
- An input form element whose value is controlled by React in this way is called a “controlled component”.
- Demo 1: Handling input Text in a Form
- Demo 2: Handling a select control in the form

SN	Controlled	Uncontrolled
1.	It does not maintain its internal state.	It maintains its internal states.
2.	Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
3.	It accepts its current value as a prop.	It uses a ref for their current values.
4.	It allows validation control.	It does not allow validation control.
5.	It has better control over the form elements and data.	It has limited control over the form elements and data.

Controlled Vs. Uncontrolled Component

- A controlled component is bound to a value, and its changes will be handled in code by using **event-based callbacks**.
- It is similar to the traditional HTML form inputs.

SN	Controlled	Uncontrolled
1.	It does not maintain its internal state.	It maintains its internal states.
2.	Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
3.	It accepts its current value as a prop.	It uses a ref for their current values.
4.	It allows validation control.	It does not allow validation control.
5.	It has better control over the form elements and data.	It has limited control over the form elements and data.

EVIDEN

09 Performing REST API Call

REST call

- Axios is among the most popular promise based HTTP clients for both browsers and Node.js.
- Npm install axios
- Get method is used to perform the get request.
 - `axios.get(`https://jsonplaceholder.typicode.com/users`)`
- In Axios, you can create POST requests using the `.post()` method and passing the data as a second parameter
 - **`axios.post(url[, data[, config]])`**
- `.delete()` method is used to perform a delete request.
 - `axios.delete(url[, config])`
 - **Demo**

EVIDEN

Questions



The background features a large, abstract graphic on the right side consisting of several interlocking, three-dimensional loops or ribbons in various shades of blue and teal, creating a complex, knot-like structure.

EVIDEN

Thank you!

For more information, please contact us.

Confidential information owned by BULL SAS, to be used by the recipient only.
This document, or any part of it, may not be reproduced, copied, circulated
and/or distributed nor quoted without prior written approval from BULL SAS.

© BULL SAS – For internal use