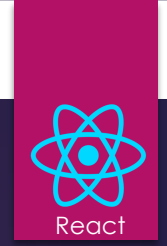
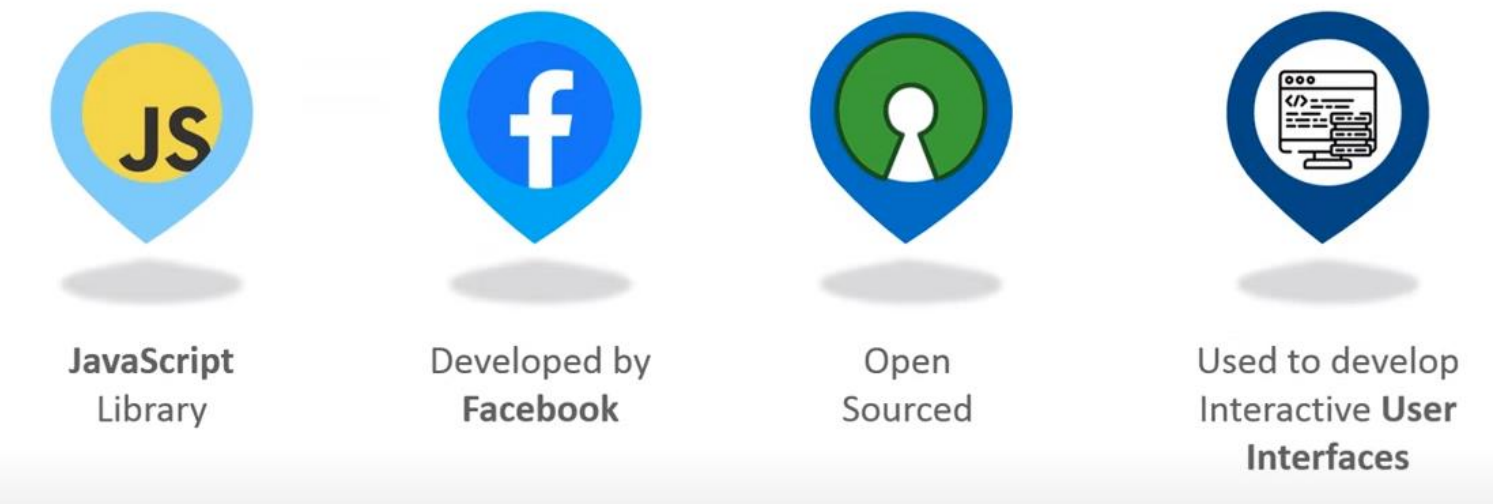
VENKAT
CORPORATE TRAINER & MOTIVATIONAL SPEAKER

Prerequisites?



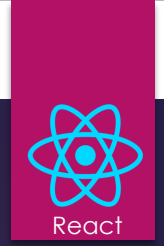
- HTML
- CSS
- JAVASCRIPT Fundamentals & OOPS
- ES.next

What is React?



- **React** was **created** by Jordan Walke, and it was open sourced in May 2013
- A JavaScript Library For Building User Interfaces
  - Most people think it is a framework but it is library
- Renders your UI and responds to events.

Framework vs Library



- Many of us will be unaware of this difference which is really important to understand during development

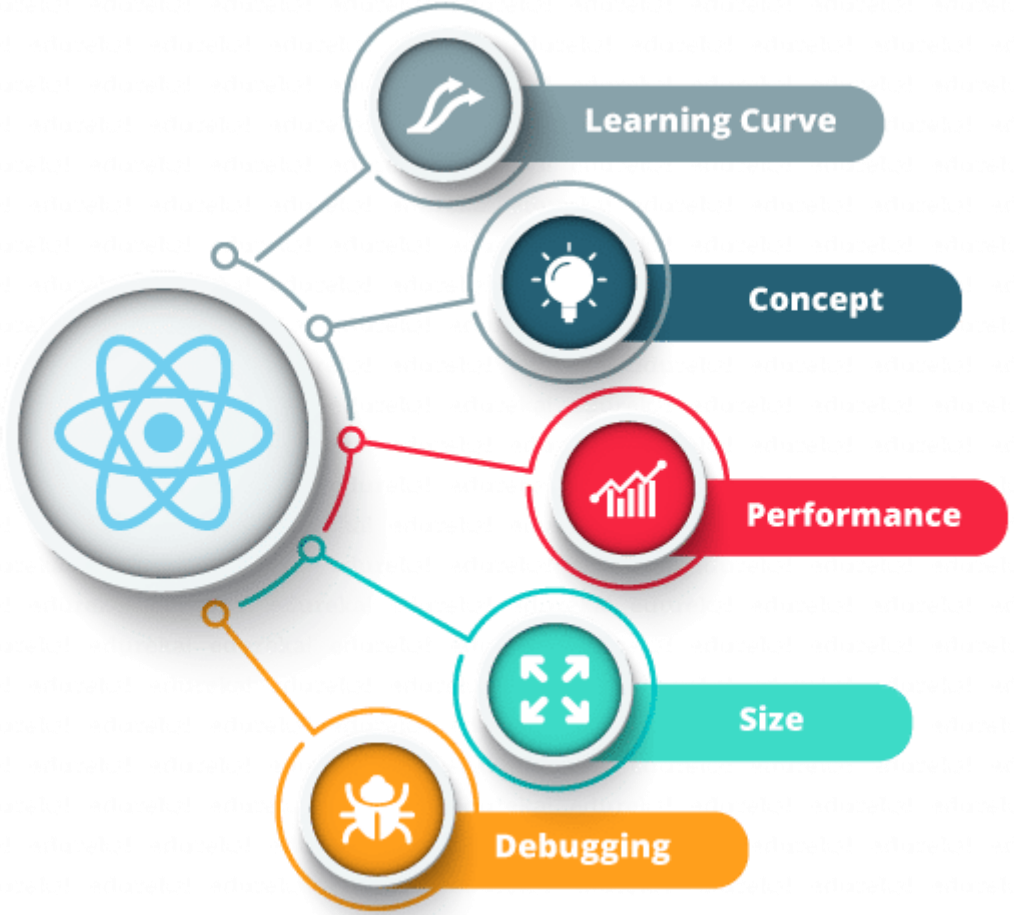
```
$(document.ready(){     // this call will be done by the jquery
// framework when document will be ready.

    function() {
        /* your code */    // our implementation inside the framework's function
    }
});



str = "Product.price"
var pos = str.lastIndexOf("."); // simply calling function of string library
```

Features of React

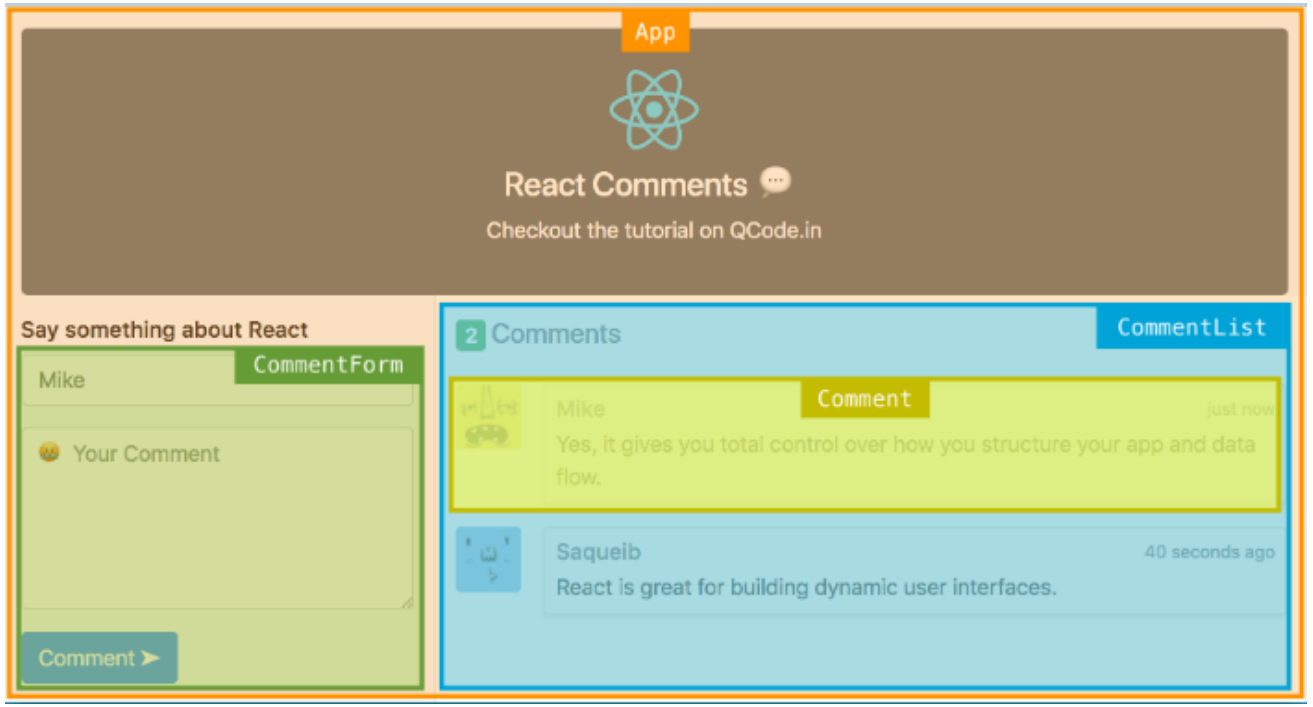


What is React?

React

- Design simple views for each state in your application
- React will efficiently update and render just the right components when your data changes.
- Declarative views make your code more predictable and easier to debug.
- Build encapsulated components that manage their own state, then compose them to make complex UIs.
- Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

React has just Component



JSX?

React Uses JSX for Writing the code.

Which is basically template syntax

like HTML but provides full power of JavaScript

```
function App() {
  return (
    <div className="App">
      <h1>Welcome to React </h1>
    </div>
  );
}
```

JSX IS OPTIONAL

- React components implement a render() method that takes input data and returns what to display.

With JSX

LIVE JSX EDITOR ☑ JSX?

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Taylor" />,
  document.getElementById('hello-example')
);
```

RESULT

Hello Taylor

Without JSX

LIVE JSX EDITOR ☐ JSX?

```
class HelloMessage extends React.Component {
  render() {
    return React.createElement(
      "div",
      null,
      "Hello ",
      this.props.name
    );
  }
}

ReactDOM.render(React.createElement(HelloMessage, { name:
"Taylor" }), document.getElementById('hello-example'));
```

RESULT

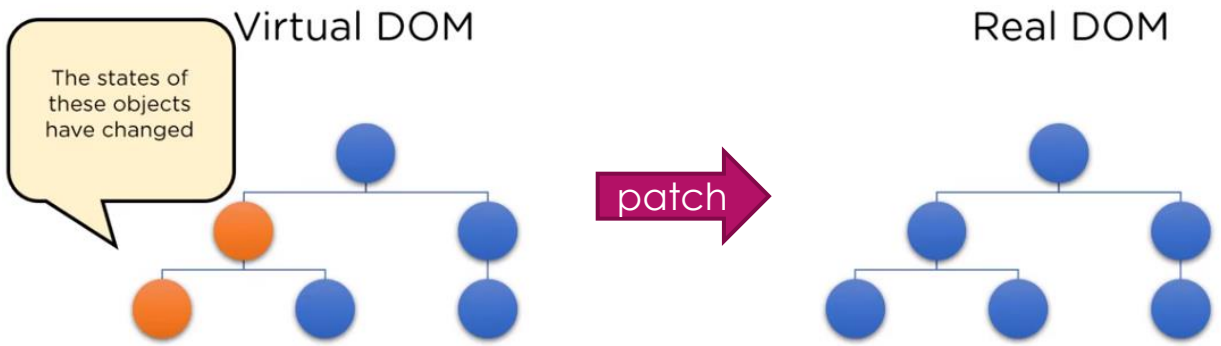
Hello Taylor

Source: https://reactjs.org/

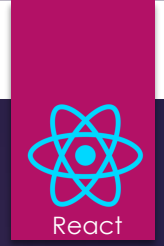
Virtual DOM

React

- The virtual DOM (VDOM) is a programming concept where an ideal, or "virtual", representation of a UI is kept in memory and synced with the "real" DOM by a library such as ReactDOM.
- This process is called reconciliation.
- The virtual DOM is a concept implemented by libraries in JavaScript on top of browser APIs.



React Fiber?



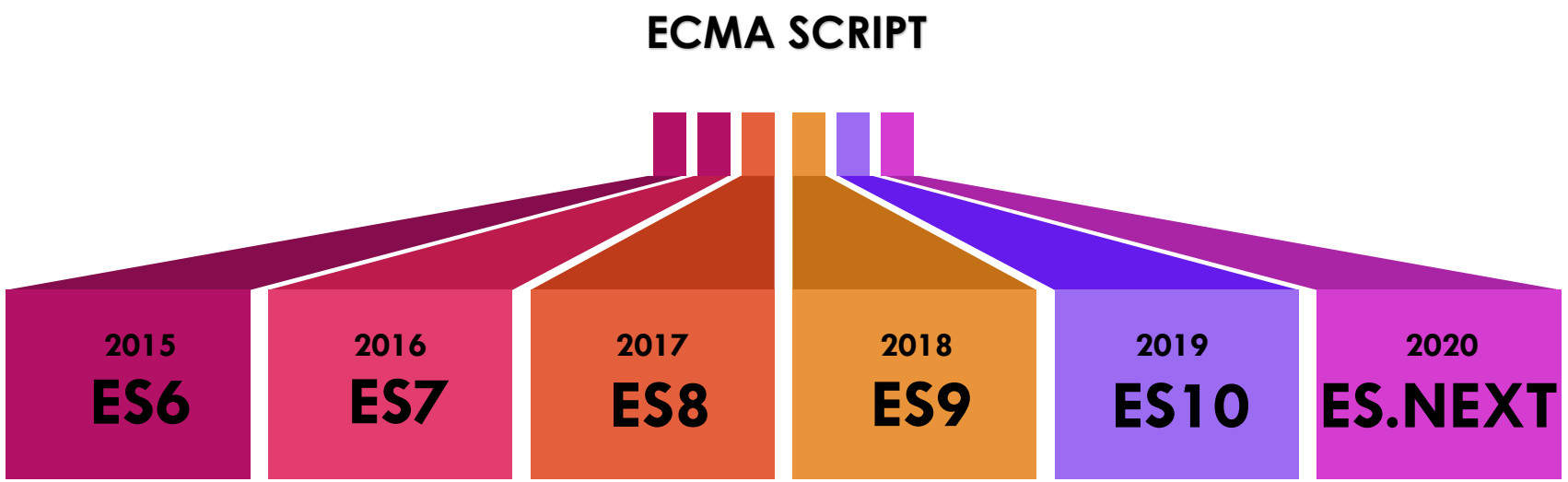
- Fiber is the new reconciliation engine in React 16.
- Its main goal is to enable incremental rendering of the virtual DOM
- The goal of React Fiber is to increase its suitability for areas like

  animation,
  layout,
  and
  gestures.

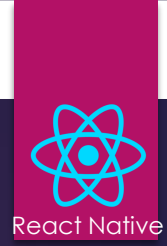
  Its headline feature is **incremental rendering**: the ability to split rendering work into chunks and spread it out over multiple frames.


ES
.next

ES6 to ES.next

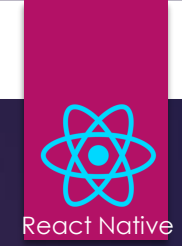


ES6



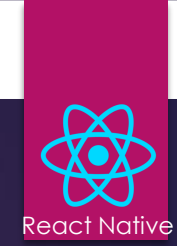
- Arrows
- Classes
- Enhanced object literals
- Template strings
- Destructuring
- Default + rest + spread
- Let + const
- Iterators + for..Of
- Generators
- Unicode
- Modules
- Module loaders
- Map + set + weakmap + weakset
- Proxies
- Symbols
- Subclassable built-ins
- Promises
- Math + number + string + array + object apis
- Binary and octal literals
- Reflect api
- Tail calls

ES7



- Array.prototype.includes()
- Exponentiation operator (**)

ES8



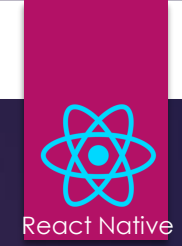
Major Features

- Async Functions (Brian Terlson)
- Shared memory and atomics (Lars T. Hansen)

Minor Features

- Object.values/Object.entries (Jordan Harband)
- String padding (Jordan Harband, Rick Waldron)
- Object.getOwnPropertyDescriptors() (Jordan Harband, Andrea Giammarchi)
- Trailing commas in function parameter lists and calls (Jeff Morrison)

ES9



Major Features

- Asynchronous Iteration (Domenic Denicola, Kevin Smith)
- Rest/Spread Properties (Sebastian Markbåge)

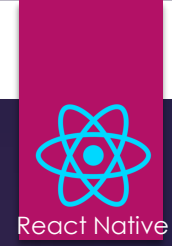
Minor Features

- RegExp named capture groups (Gorkem Yakin, Daniel Ehrenberg)
- RegExp Unicode Property Escapes (Mathias Bynens)
- RegExp Lookbehind Assertions (Gorkem Yakin, Nozomu Katō, Daniel Ehrenberg)
- s (dotAll) flag for regular expressions (Mathias Bynens)

Other Features

- Promise.prototype.finally() (Jordan Harband)
- Template Literal Revision (Tim Disney)

ES10 / next



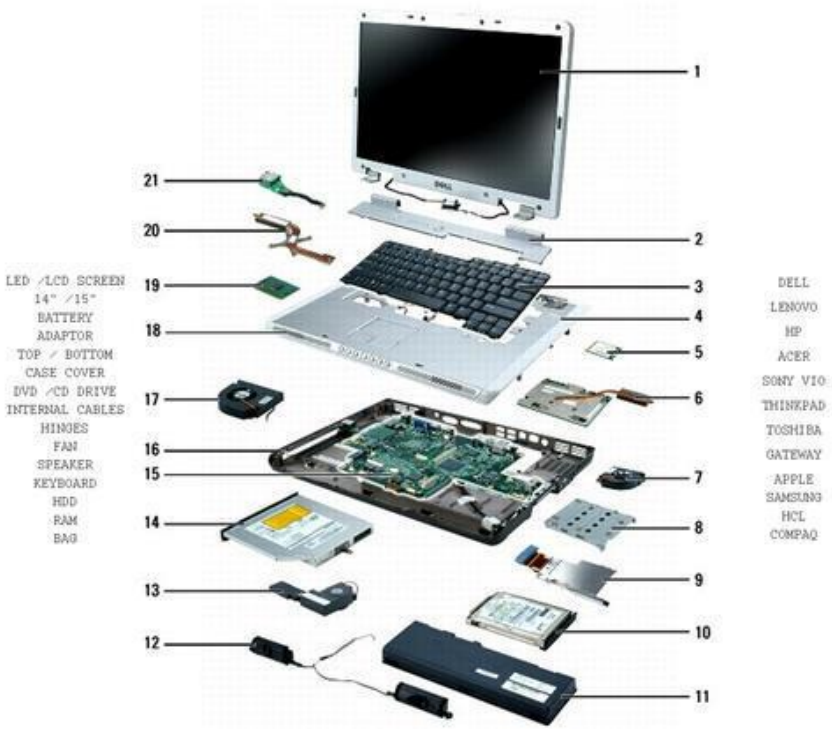
- Bigint
  - const b = 1n; // append n to create a BigInt
- string.prototype.matchAll()

Upcoming Js Proposals

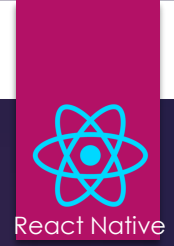
https://github.com/tc39/proposals

PRIVATE AND PROTECTED

- *Internal interface* – methods and properties, accessible from other methods of the class, but not from the outside.
- *External interface* – methods and properties, accessible also from outside the class.



Javascript CRUD App



- *Pure JS*
- *Classes*
- *Dom Manipulation*
- *Local Storage*
- *Arrow Functions*

Module

- Module is just a file, one script is one module
- Modules can load each other and use special directives export and import to interchange functionality
- Call function of one module from another one
  - Export: keyword labels variables and functions that should be accessible from outside the current module.
  - Import: allows the import of functionality from other modules.

```
//module.js

export function fn(){
        ……
}
```

```
//main.js

import { fn } from ‘./module.js’;

alert(fn)     // function…
fn(‘user’)    // Hello, user
```

Map

- Map is a collection of key : value data items. Just like Object
- Difference is it Allows any type of the kays
- Unlike objects keys are not converted to strings

  - new Map() – creates the map.
  - map.set(key, value) – stores the value by the key.
  - map.get(key) – returns the value by the key, undefined if key doesn't exist in map.
  - map.has(key) – returns true if the key exists, false otherwise.
  - map.delete(key) – removes the value by the key.
  - map.clear() – removes everything from the map.
  - map.size() – returns the current element count.

Set

- A Set is a special type collection – “set of values” (without keys)
- Each value may occur only once (Unique)

  - new Set(iterable) – creates the set, and if an iterable object is provided (usually an array), copies values from it into the set.
  - set.add(value) – adds a value, returns the set itself.
  - set.delete(value) – removes the value, returns true if value existed at the moment of the call, otherwise false.
  - set.has(value) – returns true if the value exists in the set, otherwise false.
  - set.clear() – removes everything from the set.
  - set.size – is the elements count.

Generator

- Regular functions return only one, single value (or nothing).
- Generators can return ("yield") multiple values, one after another, on-demand.
- They work great with iterables, allowing to create data streams with ease.

```
function* generateSequence(){
          yield 1;
          yield 2;
          return 3;
}
```

- Generator functions behave differently from regular ones. When such function is called, it doesn't run its code. Instead it returns a special object, called "generator object", to manage the execution.

Virtual Dom


Model
User: Allen
Id: 101
Loc: USA
View
User: Max
Id:101
Loc: USA
User: Allen
Id:101
Loc: USA
DOM

Advantages

Application's performance is increased

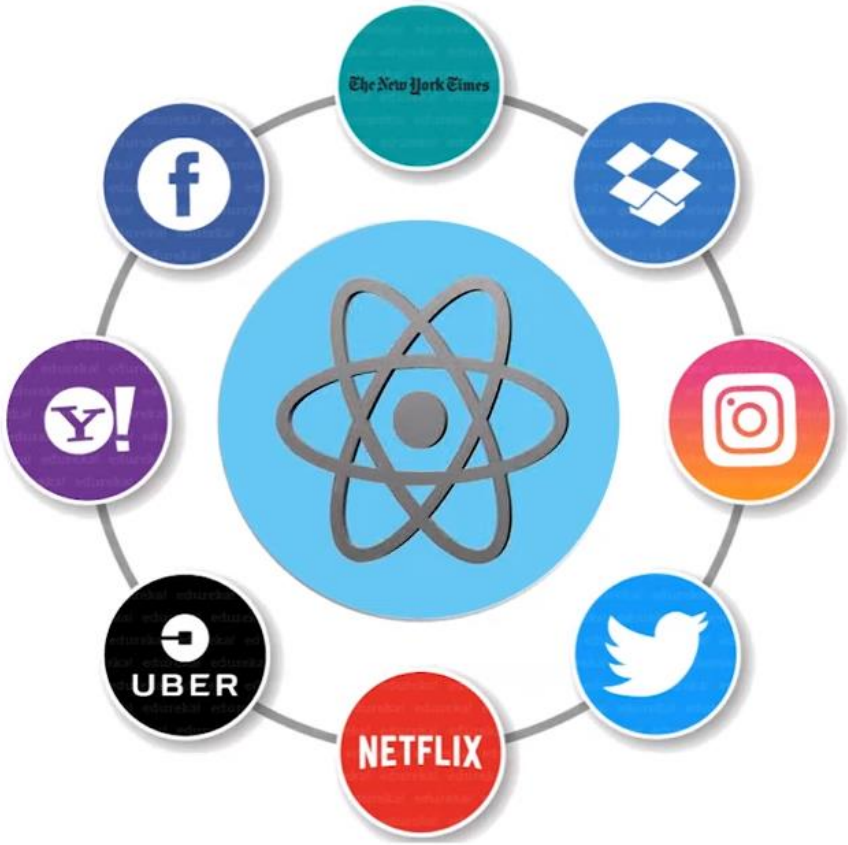
Used on Client as well as Server Side

Readability is Improved

METEOR

Easily used with other frameworks

Applications Using React

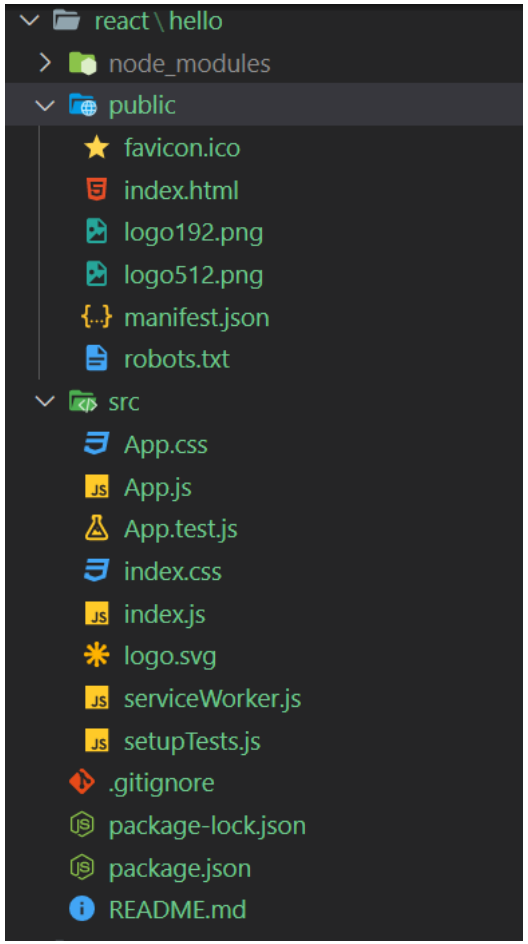


Environment Setup

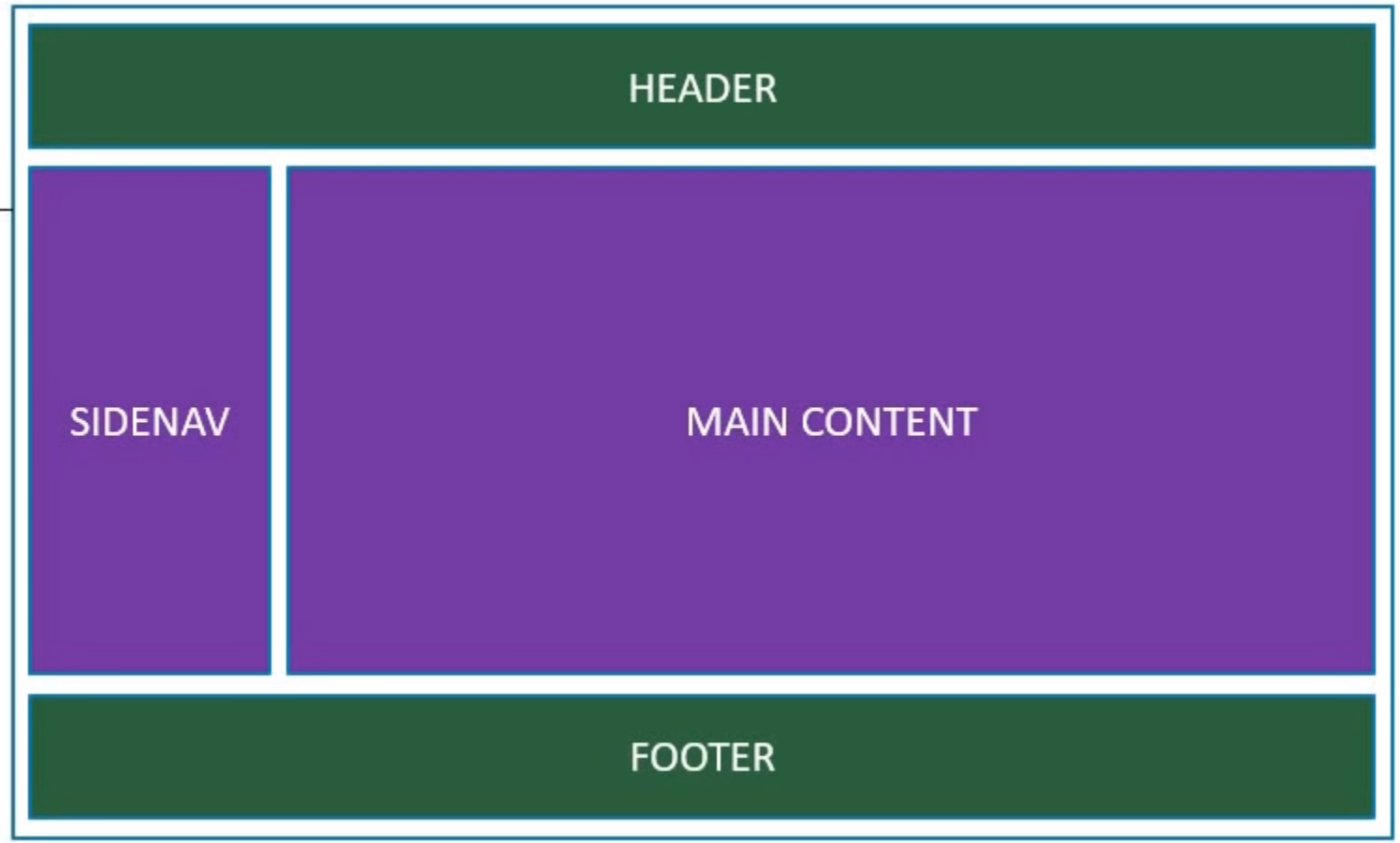
- Install Node
- Install react globally
  - npm i create-react-app –g
- Create app
  - npx create-react-app AppName
  - cd appFolder
  - npm start

Npx	Npm
npx create-react-app <appname>	npm create-react-app -g
npm package runner	create-react-app <appname>

Folder Structure



Components



- Components describes a part of user interface
- They are re usable and can be integrated in other components

Component Types

Stateless Functional Component

JavaScript Functions

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

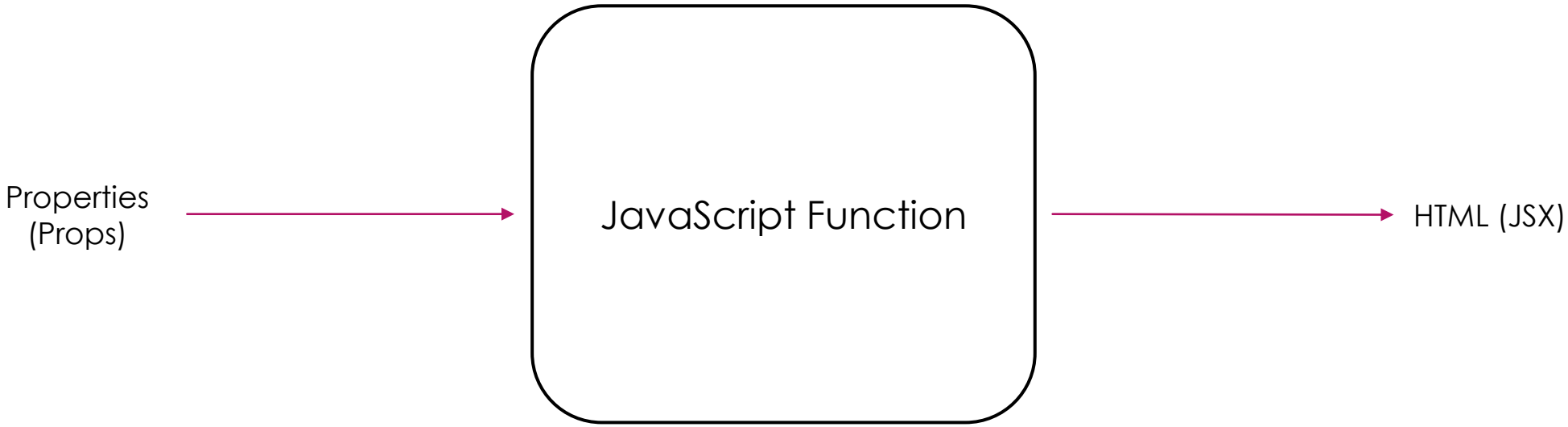
Stateful Class Component

Class extending Component class

Render method returning HTML

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Functional Components



Class Components

Functional vs Class Components

Simple functions

Use Func components as much as possible

Absence of ‘this’ keyword

Solution without using state

Mainly responsible for the UI

Stateless/ Dumb/ Presentational

More feature rich

Maintain their own private data - state

Complex UI logic

Provide lifecycle hooks

Stateful/ Smart/ Container

Hooks (2018: React 16.7 Alpha)

- No Breaking changes
- Completely opt-in & 100% backwards-compatible
- What ever we'have learned so far in this series still holds good
- Using state, lifecycle methods and 'this' binding
- After understanding state, event binding and lifecycle hooks in class components

JSX

JavaScript XML (JSX) – Extension to the JavaScript language syntax.

Write XML-like code for elements and components.

JSX tags have a tag name, attributes, and children.

JSX is not a necessity to write React applications.

JSX makes your react code simpler and elegant.

JSX ultimately transpiles to pure JavaScript which is understood by the browsers.

JSX Differences

Class -> className

for -> htmlFor

camelCase property naming convention

- onclick -> onClick
- tabindex -> tabIndex

https://github.com/facebook/react/issues/

https://github.com/facebook/react/issues/13525

Props vs State

props	state
props get passed to the component	state is managed within the component
Function parameters	Variables declared in the function body
props are immutable	state can be changed
props – Functional Components this.props – Class Components	useState Hook – Functional Components this.state – Class Components

return vs return()

```
return
a + b;
```

is transformed by ASI into:

```
return;
a + b;
```

To avoid this problem (to prevent ASI), you could use parentheses:

```
return (
        a + b
);
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Automatic_semicolon_insertion

setState()

- Always make use of setSTate and never modify state directly.
- Code has to be executed after the state has been updated? Place that code in the call back function which is the second argument to the setState method
- When you have to update state based on the previous state value, pass in a function as an argument instead of the regular object.

Destruction

- Always make use of setSTate and never modify state directly.
- Code has to be executed after the state has been updated? Place that code in the call back function which is the second argument to the setState method
- When you have to update state based on the previous state value, pass in a function as an argument instead of the regular object.

Binding Events

- **Approach 1:** Binding events in render method
- **Approach 2:** Binding events using arrow
- **Approach 3:** Binding events in class constructor
- **Approach 4:** Binding Events in class property as Arrow functions

Today Topics

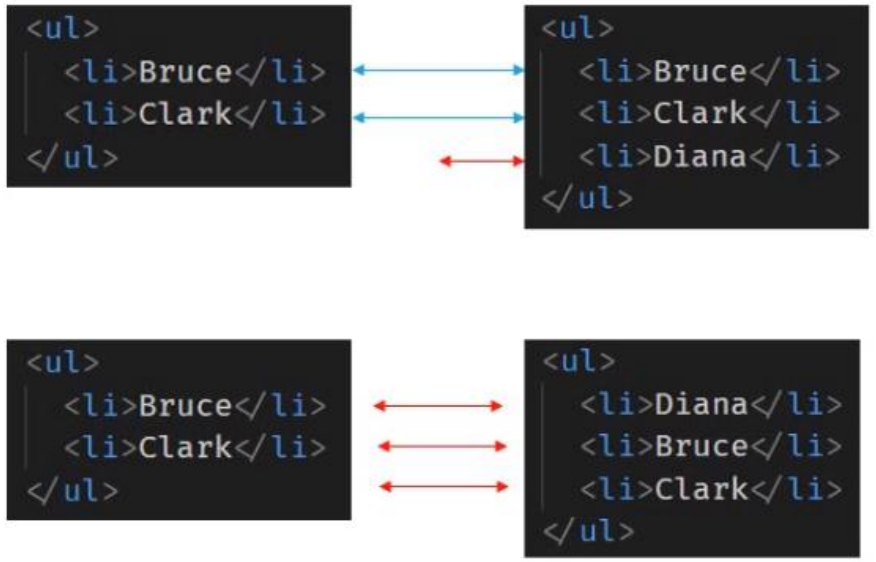
- CONDITIONAL RENDERING
- LIST RENDERING
- LISTS AND KEYS
- INDEX AS KEY PATTERN
- STYLING AND CSS
- BASICS OF FORM HANDLING
- COMPONENT MOUNTING LIFE CYCLE METHODS
- COMPONENT UPDATE LIFE CYCLE METHODS
- FRAGMENTS
- PURE COMPONENTS
- MEMO COMPONENTS
- REFS WITH CLASS COMPONENTS

Conditional Rendering

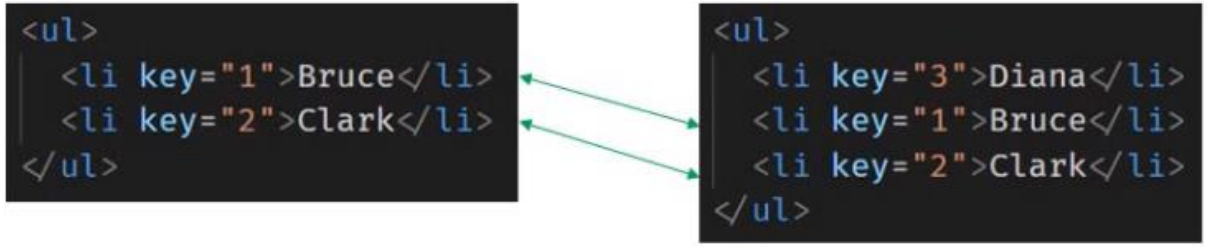
- When we work with components we should show or hide few html elements
  - If/else
  - Element Variables
  - Ternary Operator
  - Short Circuit Operator

List and Keys

- List Without Key



- List with Key



List and Keys

- A “key” is a special string attribute you need to include when creating of elements
- Keys give the elements stable identity
- Keys help react identity which item have changed, are added, or are removed
- Help in efficient update of the user interface

Index as Anti-Pattern

https://codepen.io/vsrao/pen/vYNPzWP

Index as Key

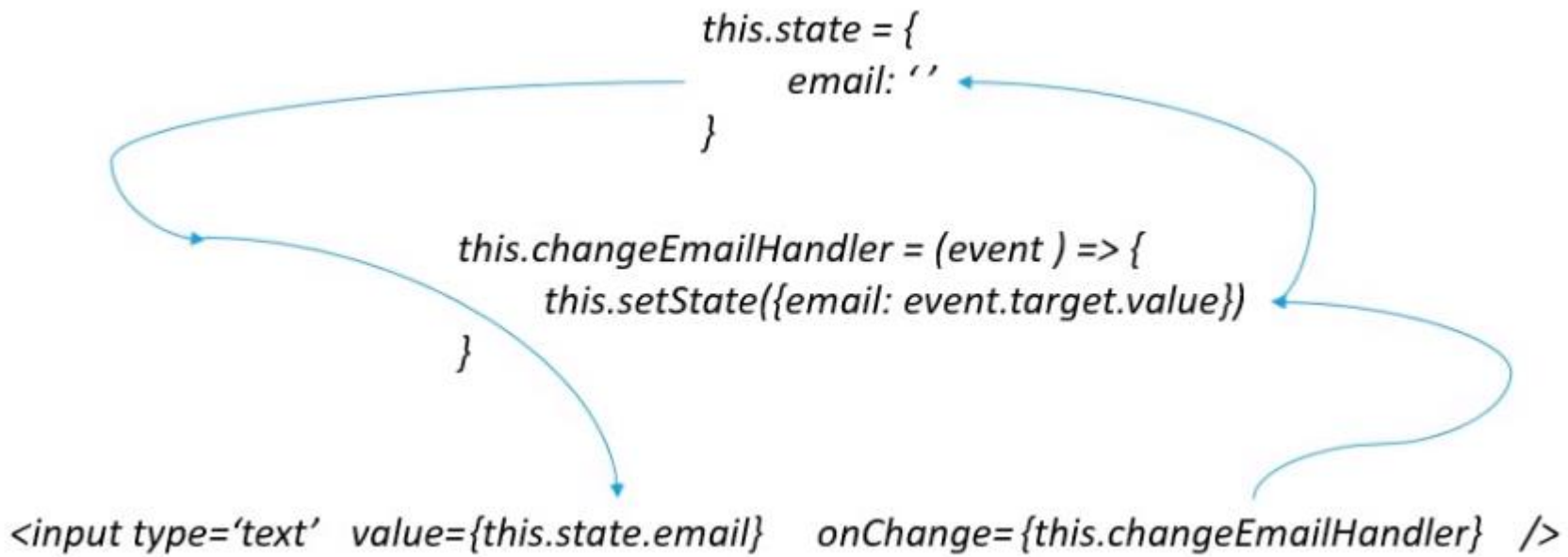
- When Index as a key
  - The items in your list do not has a unique id
  - The list is a static list and will not change
  - The list will never be rendered or filtered

https://codepen.io/vsrao/pen/vYNPzWP

Basic Styling React Components

- Css Style Sheet
- Inline Styling
- Css Modules
- Css in Libraries (Styled Components)

Basics of Forms Handling



Component Life Cycle

Mounting	When an instance of a component is being created and inserted into the DOM
Updating	When a component is being re-rendered as a result of changes to either its props or state
Unmounting	When a component is being removed from the DOM
Error Handling	When there is an error during rendering, in a lifecycle method, or in the constructor of any child component

Component Life Cycle Methods

Mounting	*constructor, static getDerivedStateFromProps, render* and *componentDidMount*
Updating	*static getDerivedStateFromProps, shouldComponentUpdate, render, getSnapshotBeforeUpdate* and *componentDidUpdate*
Unmounting	*componentWillUnmount*
Error Handling	*static getDerivedStateFromError* and *componentDidCatch*

Mounting Life Cycle Methods

constructor(props)

A special function that will get called whenever a new component is created.

Initializing state
Binding the event handlers

Do not cause side effects. Ex: HTTP requests

super(props)
Directly overwrite this.state

Mounting Life Cycle Methods

constructor(props)

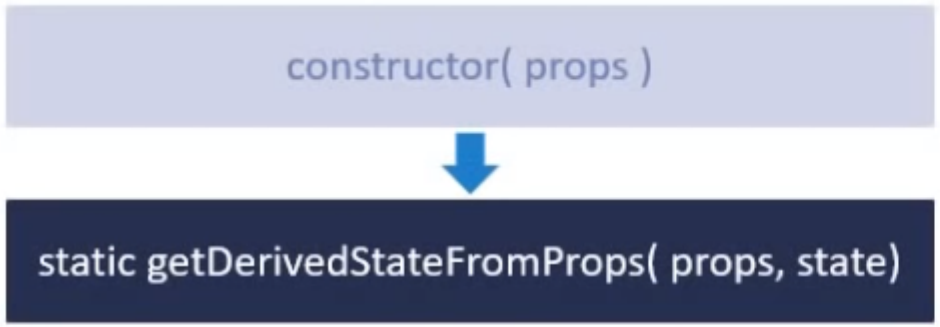
A special function that will get called whenever a new component is created.

Initializing state
Binding the event handlers

Do not cause side effects. Ex: HTTP requests

super(props)
Directly overwrite this.state

Mounting Life Cycle Methods

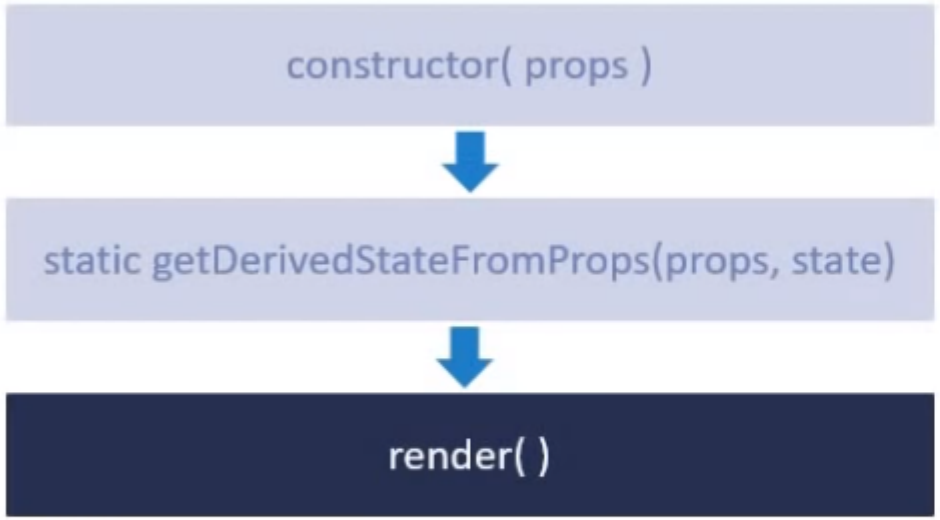


When the state of the component depends on changes in props over time.

Set the state

Do not cause side effects. Ex: HTTP requests

Mounting Life Cycle Methods



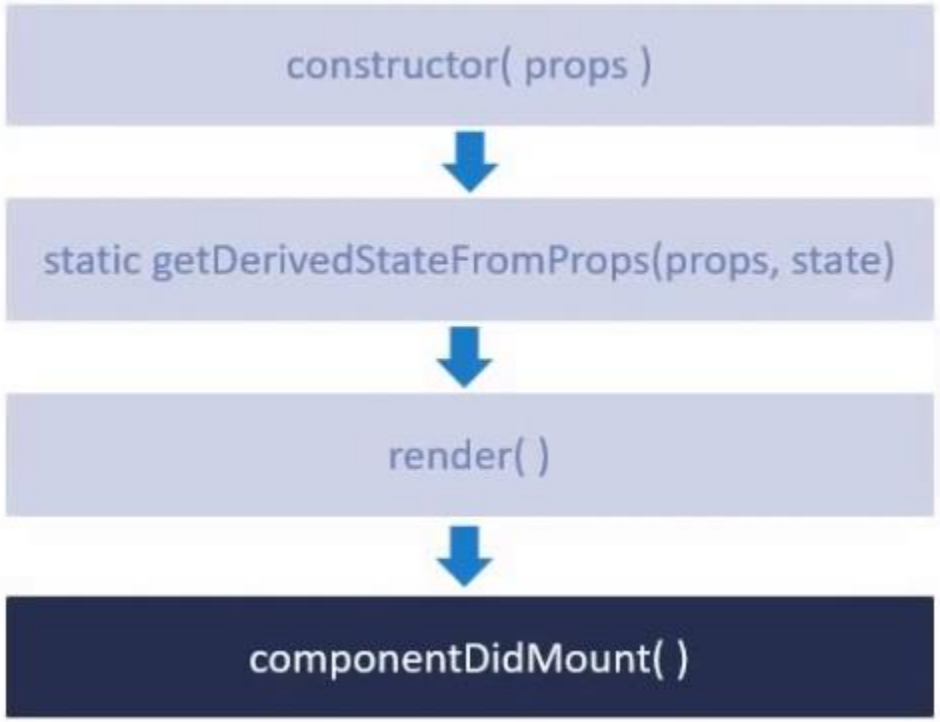
Only required method

Read props & state and return JSX

Do not change state or interact with DOM or make ajax calls.

Children components lifecycle methods are also executed.

Mounting Life Cycle Methods



Invoked immediately after a component and all its children components have been rendered to the DOM.

Cause side effects. Ex: Interact with the DOM or perform any ajax calls to load data.

Updating Life Cycle Methods

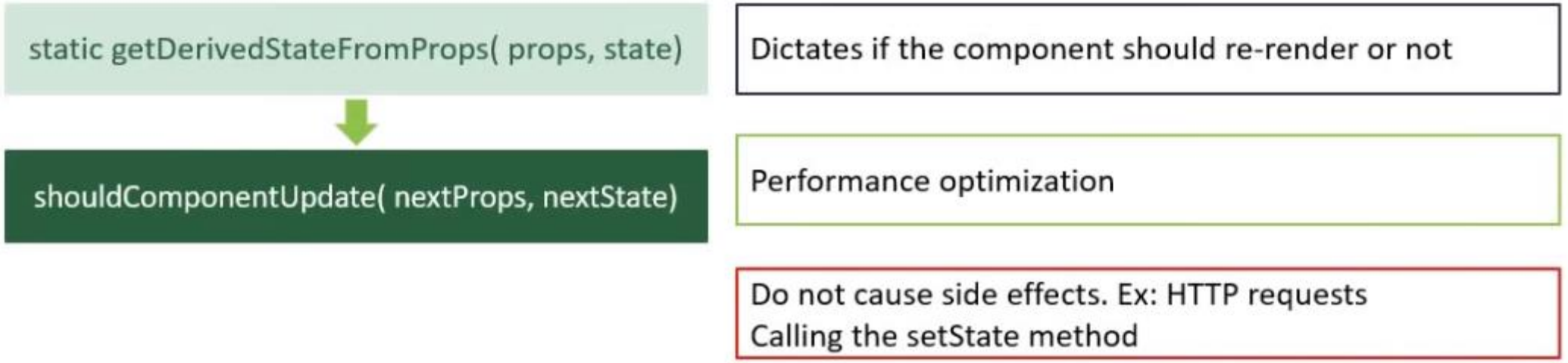
static getDerivedStateFromProps(props, state)

Method is called every time a component is re-rendered

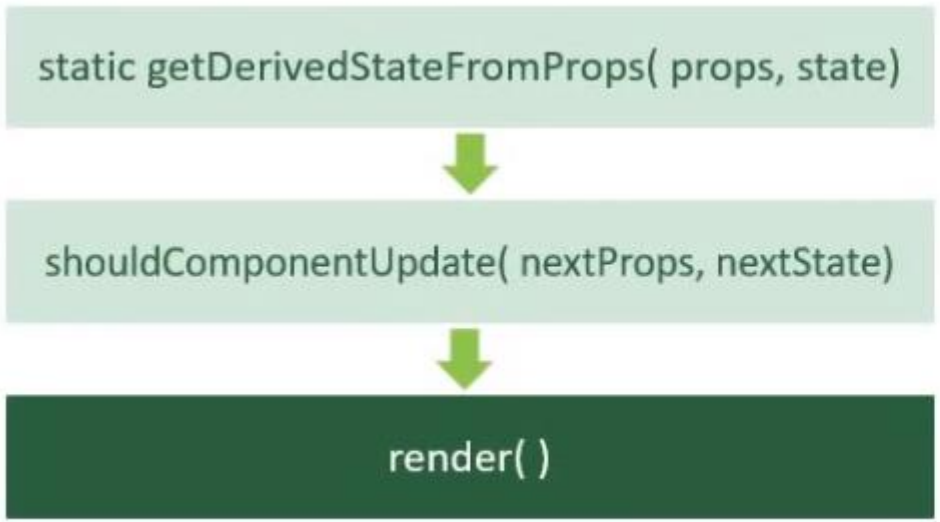
Set the state

Do not cause side effects. Ex: HTTP requests

Updating Life Cycle Methods



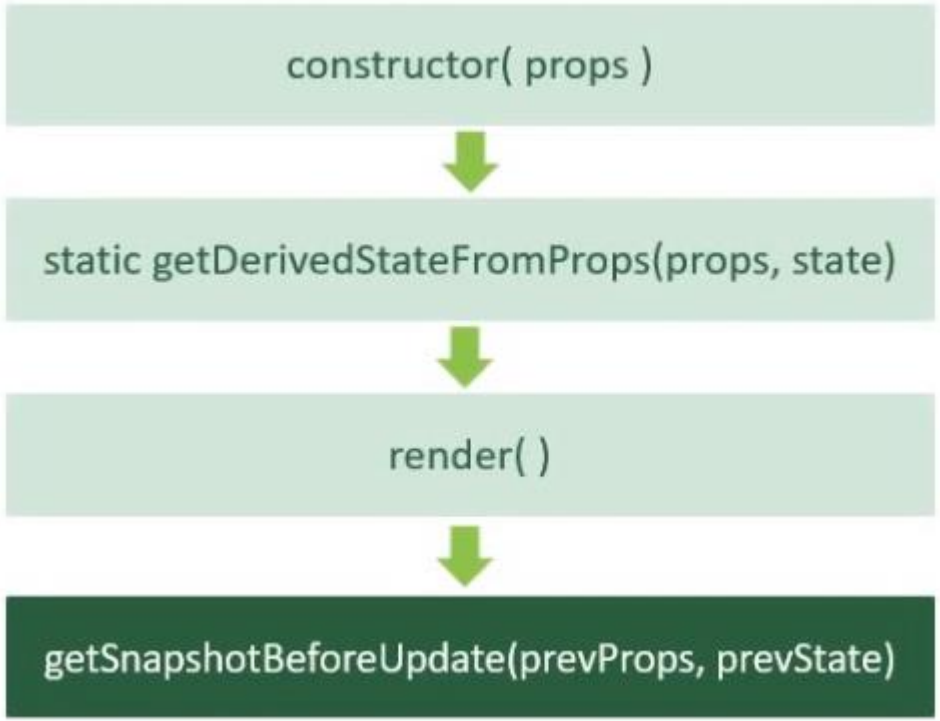
Updating Life Cycle Methods



Only required method

Read props & state and return JSX

Updating Life Cycle Methods

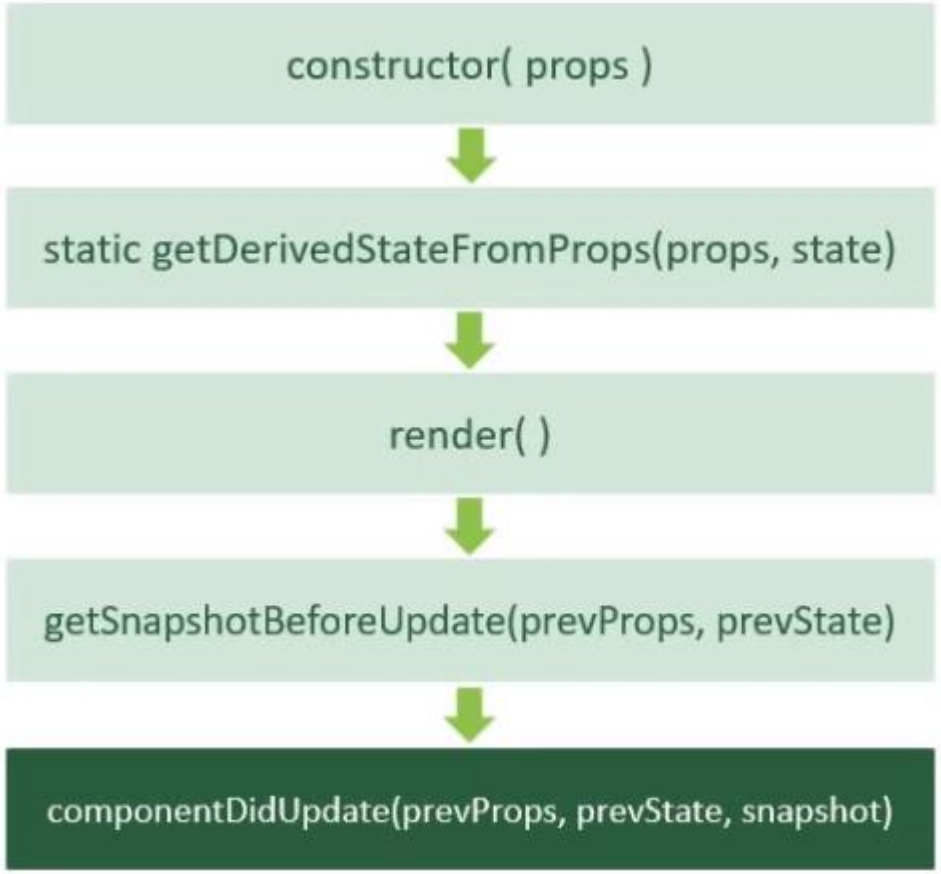


Called right before the changes from the virtual DOM are to be reflected in the DOM

Capture some information from the DOM

Method will either return null or return a value. Returned value will be passed as the third parameter to the next method.

Updating Life Cycle Methods



Called after the render is finished in the re-render cycles

Cause side effects

Unmounting Phase Method

componentWillUnmount( )

Method is invoked immediately before a component is unmounted and destroyed.

Cancelling any network requests, removing event handlers, cancelling any subscriptions and also invalidating timers.

Do not call the setState method.

Error Handline Phase

static getDerivedStateFromError(error)

componentDidCatch(error, info)

When there is an error either during rendering, in a lifecycle method, or in the constructor of any child component.

Today Topics

- FRAGMENTS
- PURE COMPONENTS
- MEMO COMPONENTS
- REFS
- REFS WITH CLASS COMPONENTS
- FORWARDING REFS
- PORTALS
- ERROR BOUNDARY
- HIGH ORDER COMPONENTS (HOC)

Pure Components

Regular Component

A regular component does not implement the *shouldComponentUpdate* method. It always returns true by default.

Pure Component

A pure component on the other hand implements *shouldComponentUpdate* with a shallow props and state comparison.

Shallow Comparison

Primitive Types
a (SC) b returns true if a and b have the same value and same type
Ex: string ‘project’ (SC) string ‘project’ returns true

Complex Types
a (SC) b returns true if a and b refers same object

```
var a = [1,2,3];
var b = [1,2,3];
var c = a;

var ab_eq = (a === b); // false
var ac_eq = (a === c); // true
```

```
var a = { x: 1, y: 2 };
var b = { x: 1, y: 2 };
var c = a;

var ab_eq = (a === b); // false
var ac_eq = (a === c); // true
```

Conclusion Pure Components

We can create a component by extending the PureComponent class.

A PureComponent implements the *shouldComponentUpdate* lifecycle method by performing a shallow comparison on the props and state of the component.

If there is no difference, the component is not re-rendered – performance boost.

It is a good idea to ensure that all the children components are also pure to avoid unexpected behaviour.

Never mutate the state. Always return a new object that reflects the new state.

Note: Its always better to use regular components unless you see performance issue

Memo (16.6)

- To Achieve the same features as Pure components in Functional Components Memo approach helps

Refs

- Refs helps to access directly DOM Nodes with React
- Use case :
  - We have to focus input on Document loaded like google
- Refs With Class Component
  - Click on parent element, child element should get focus
- Forwarding Refs
  - Helps us to pass refs automatically to its children's

Portals

- Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.

https://codepen.io/vsrao/pen/bGVPMGK

https://codesandbox.io/s/nostalgic-borg-1f5wl?file=/src/index.js

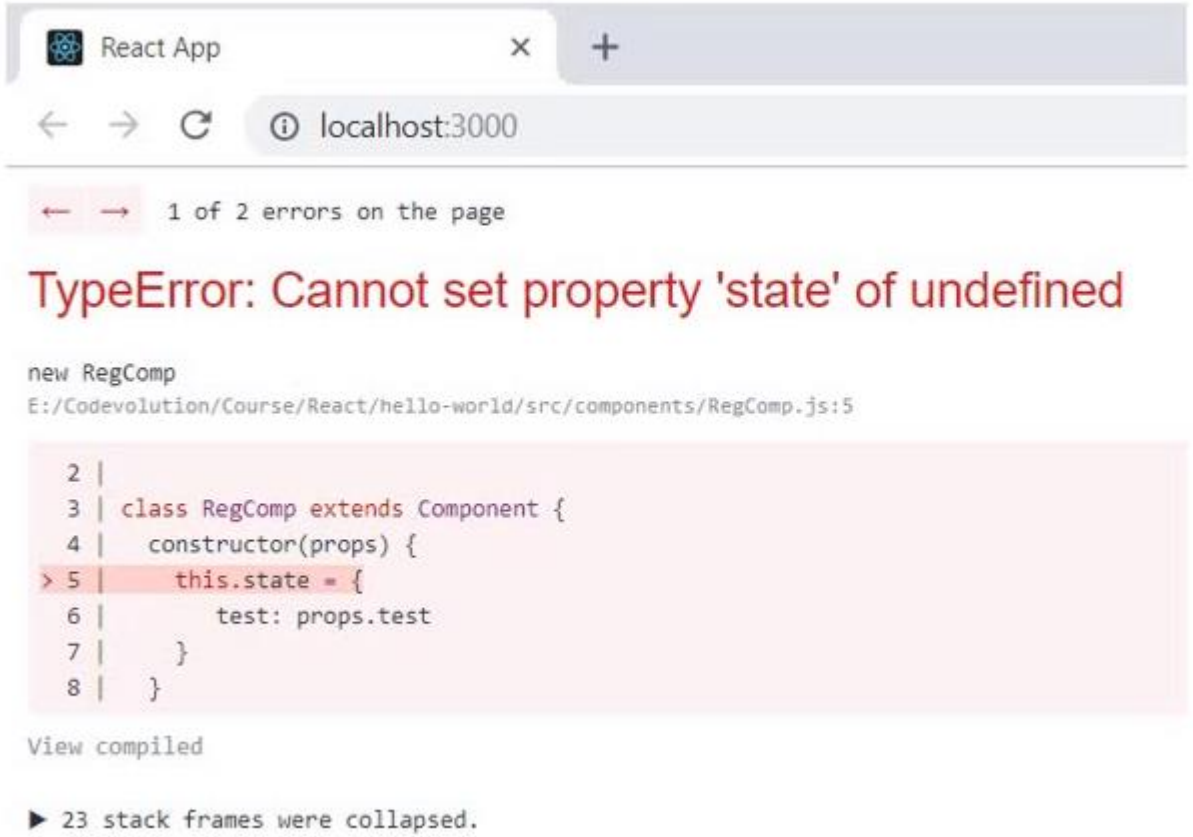
Error Handling Phase

static getDerivedStateFromError(error)

componentDidCatch(error, info)

When there is an error either during rendering, in a lifecycle method, or in the constructor of any child component.

Error in App



- When error occurs in app react loads fall back UI. Even the error is one of the children component all other component render get impacted

Error Boundary

A class component that implements either one or both of the lifecycle methods ***getDerivedStateFromError*** or ***componentDidCatch*** becomes an **error boundary**.

The static method *getDerivedStateFromError* method is used to render a fallback UI after an error is thrown and the *componentDidCatch* method is used to log the error information.

Summary of Error Boundary

Error boundaries are React components that catch JavaScript error in their child component tree, log those errors, and display a fall-back UI.

A class component becomes an Error Boundary by defining either or both of getDerivedStateFromError and componentDidCatch lifecycle methods.

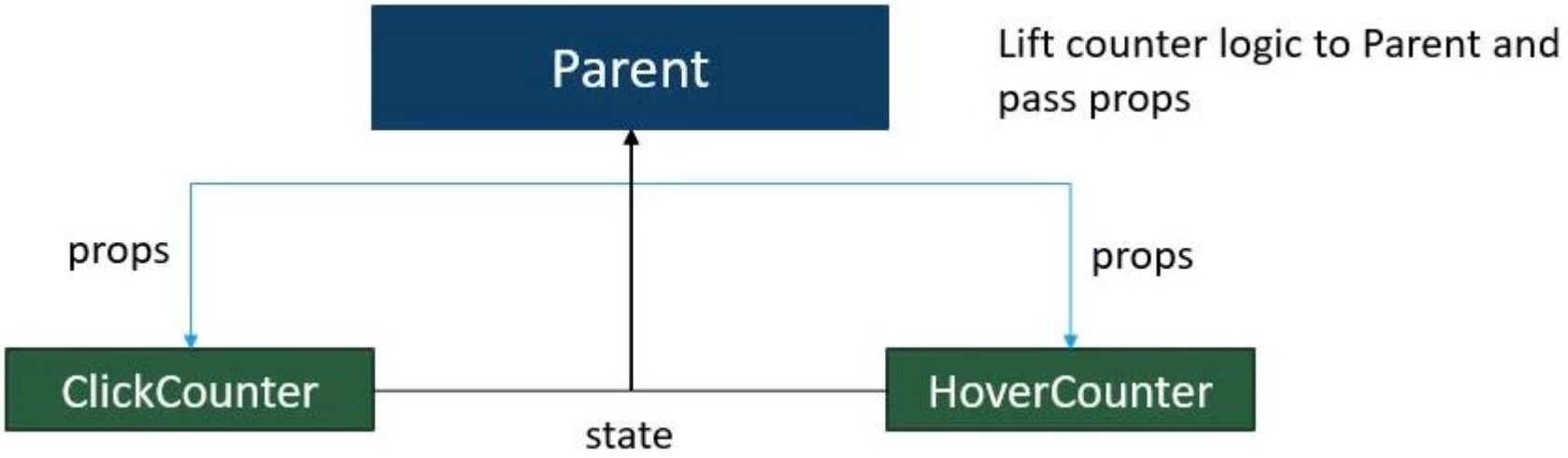
The placement of the Error Boundary also matters as it controls if the entire app should have the fall-back UI or just the component causing the problem.

Provide a way to gracefully handle error in application code.

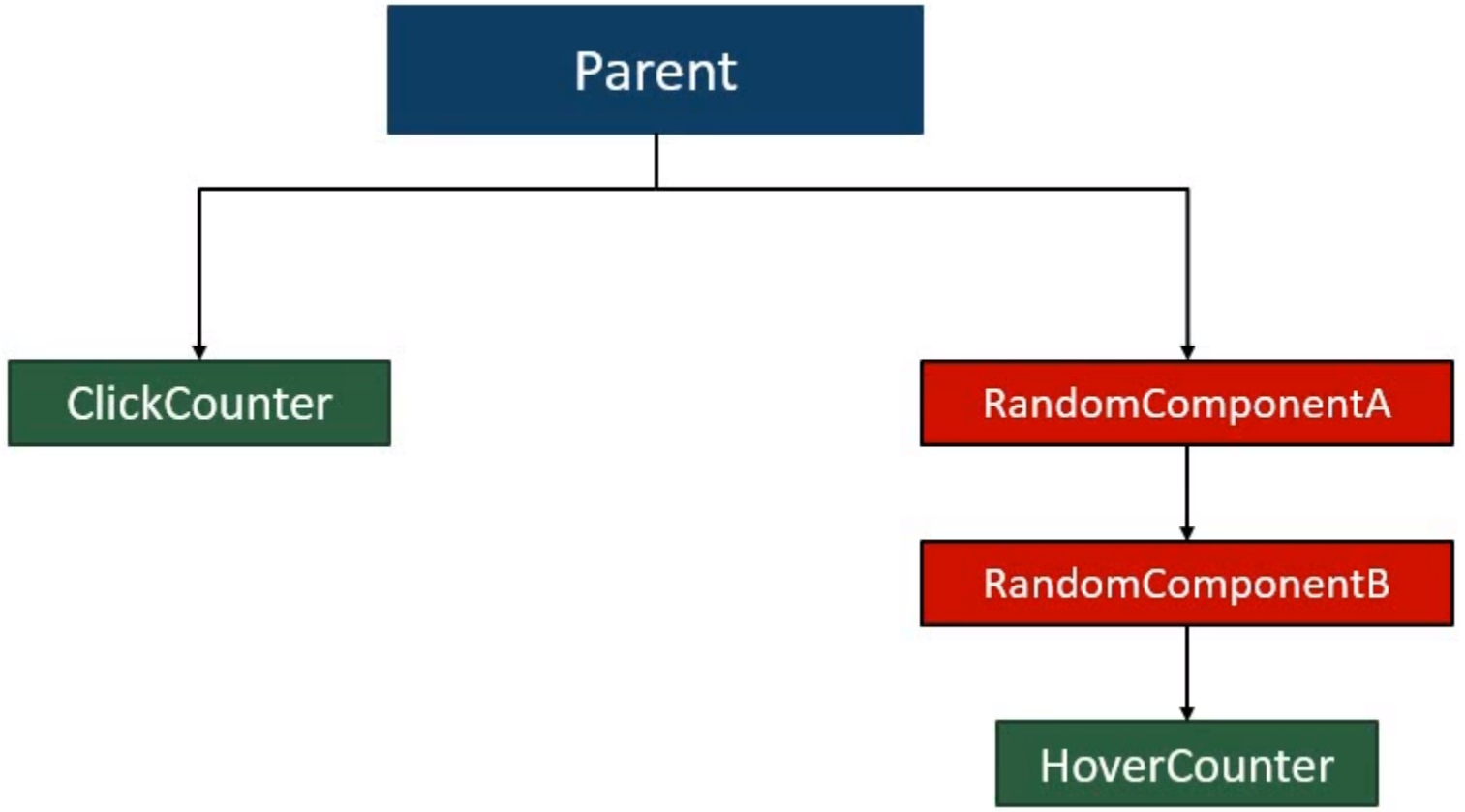
Today Topics

- HIGH ORDER COMPONENTS (HOC)
- RENDER PROPS
- CONTEXT
- HTTP
- HTTP GET
- HTTP POST
- HOOKS
- USESTATE
- USESTATE WITH PREVIOUS STATE
- USESTATE WITH OBJECT
- USESTATE WITH ARRAY
- USE EFFECT HOOK
- USE EFFECT RENDER
- USE EFFECT AFTER RENDER
- CONDITIONALLY RUN EFFECTS
- RUN EFFECTS ONLY ONCE
- USE EFFECT WITH CLEANUP
- USE EFFECT WITH INCORRECT DEPENDENCY

Higher Order Components



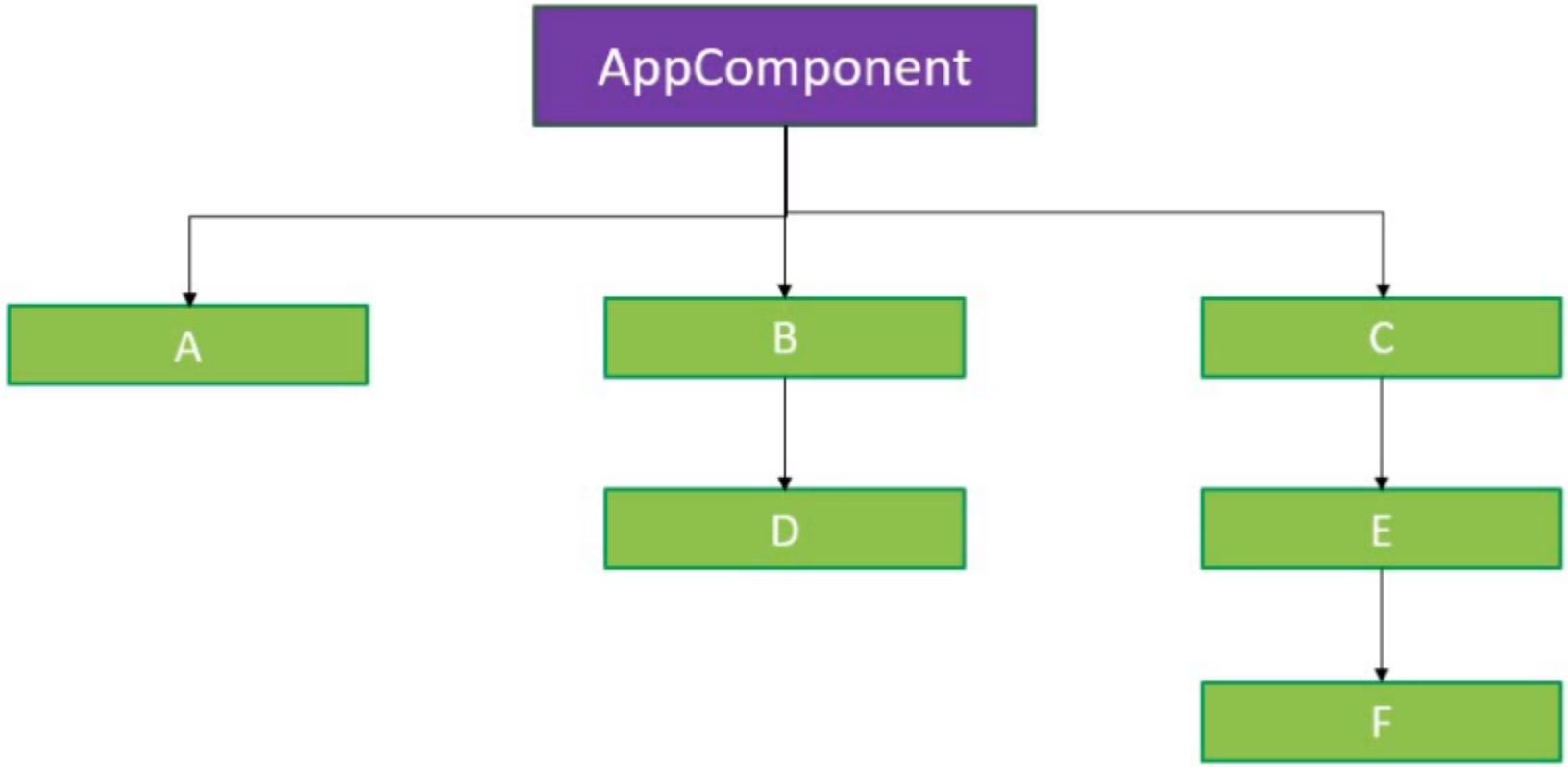
Higher Order Components



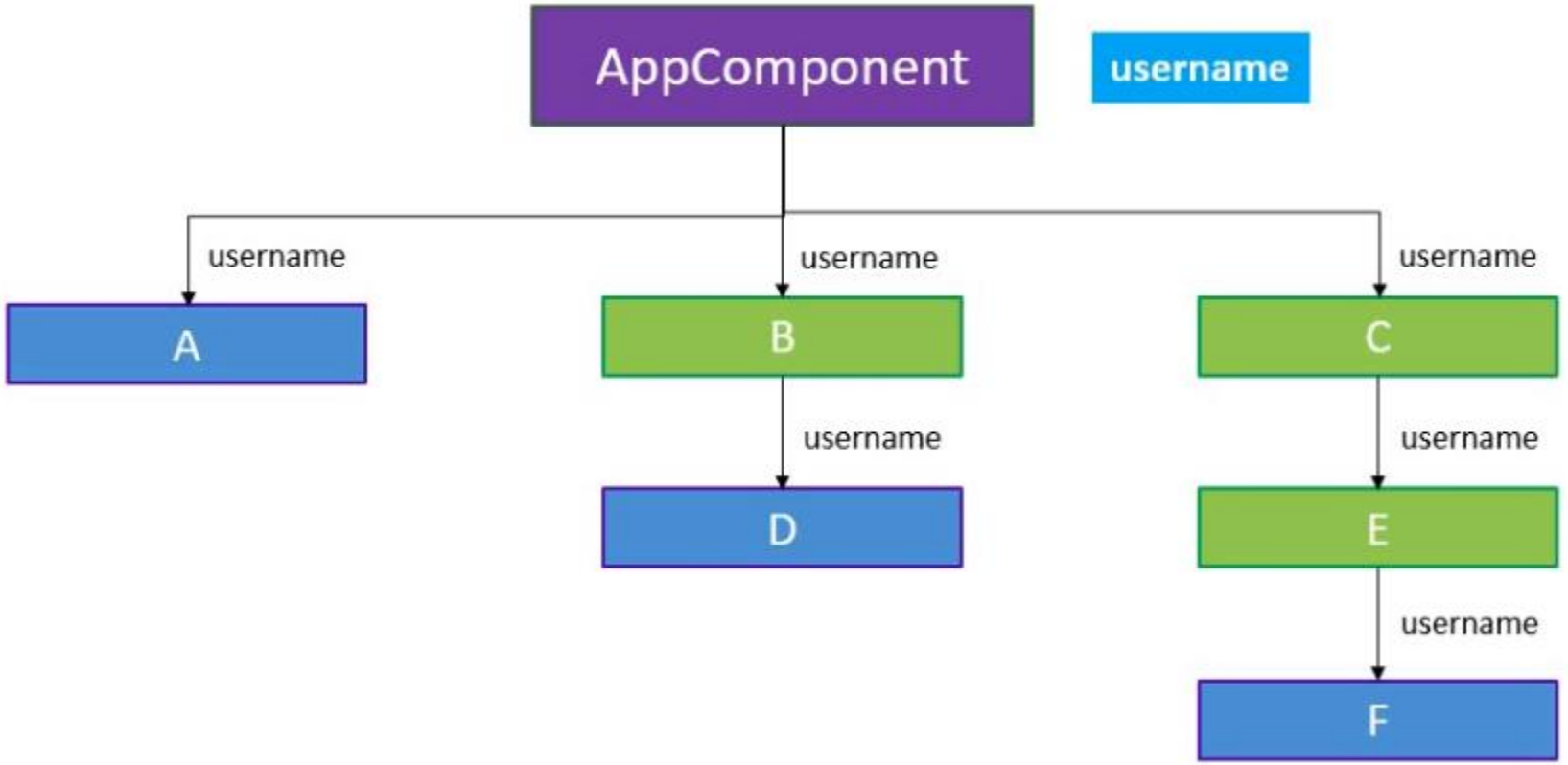
Render Prop

- The term “Render Prop” refers to a technique for sharing code between React Components using prop whose value is a function

Context

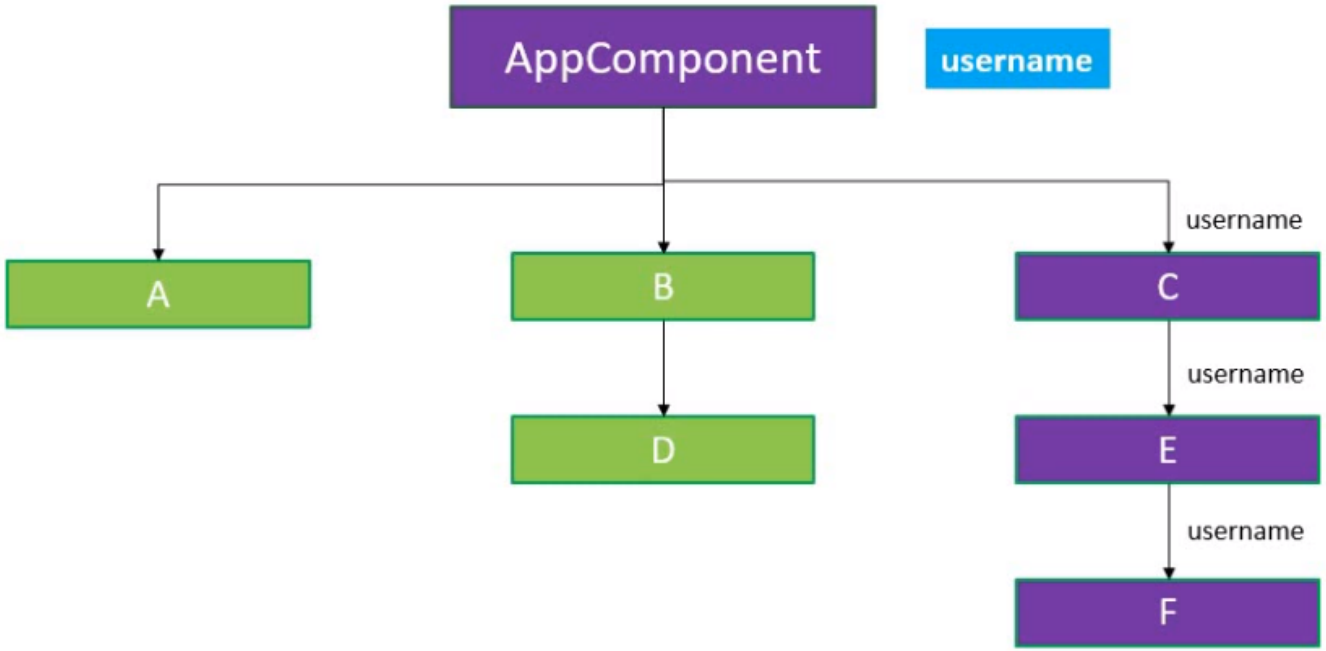


Context


AppComponent
username
username
username
username
A
B
C
username
username
D
E
username
F

Context

- Context provides a way to pass data through the component tree without having to pass props down manually at every level



Context

- Achieve context in 3 simple steps
  - Create the Context
  - Provide Context Value
  - Consume the text value

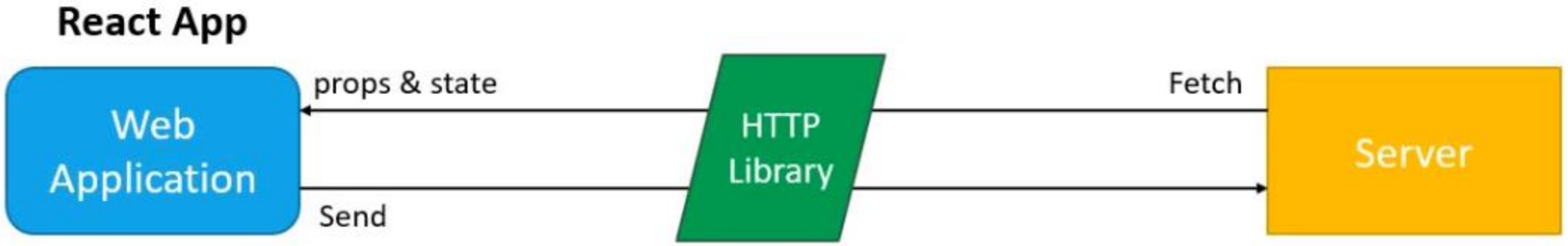
Context Type?

- Why not context type as simple?
  - Limitation 1: This works only with class components
  - Limitation 2: can use only one context

Multiple Contexts

```
function Content() {
  return (
    <ThemeContext.Consumer>
      {theme => (
        <UserContext.Consumer>
          {user => (
            <ProfilePage user={user} theme={theme} />
          )}
        </UserContext.Consumer>
      )}
    </ThemeContext.Consumer>
  );
}
```

HTTP



- React is a Library It doesn't have ability to fetch data from server
- Http Libraries we can use to fetch the data from server
- Axios
  - npm install axios

HOOKS

Prerequisites

- Basics of React
- Functional and Class Components, Props, State, etc

What are Hooks?

- Hooks are a new feature addition in React Version 16.8 which allow you to use React features without having to write a class
  - Ex: state of component
- Hooks don't work inside classes

Why Hooks?

Understand how ***this*** keyword works in JavaScript

Remember to bind event handlers in class components

Classes don't minify very well and make hot reloading very unreliable
There is no particular way to reuse stateful component logic

HOC and render props patterns do address this problem

Makes the code harder to follow

There is need a to share stateful logic in a better way
Create components for complex scenarios such as data fetching and subscribing to events

Related code is not organized in one place

Ex: Data fetching - In componentDidMount and componentDidUpdate

Ex: Event listeners – In componentDidMount and componentWillUnmount

Because of stateful logic – Cannot break components into smaller ones

Main Points of Hooks

React version 16.8 or higher

Completely opt in

Hooks don’t contain any breaking changes and the release is 100% backwards-compatible.

Classes won’t be removed from React

Can’t use Hooks inside of a class component

Hooks don’t replace your existing knowledge of React concepts

Instead, Hooks provide a more direct API to the React concepts you already know

Today Topics

- USESTATE WITH ARRAY
- USE EFFECT HOOK
- USE EFFECT RENDER
- USE EFFECT AFTER RENDER
- CONDITIONALLY RUN EFFECTS
- RUN EFFECTS ONLY ONCE
- USE EFFECT WITH CLEANUP
- USE EFFECT WITH INCORRECT DEPENDENCY

Summary of useState

- In classes, the state always an object
- With the useState hook, the state doesn't have to be an object
- The useState hook returns an array with 2 elements
- The First element is the current value of the state, and the second element is a state setter function
- New state value depends on the previous state value? You can pass a function to the setter function
- When dealing with objects or arrays, always make sure to spread your state variable and then call the setter function

useEffect

- Update the Document Title to the current counter value

```
componentDidMount() {
  document.title = `You clicked ${this.state.count} times`;
}

componentDidUpdate() {
  document.title = `You clicked ${this.state.count} times`;
}
```

Use Effect : Use case

- Timer

```
componentDidMount() {
  this.interval = setInterval(this.tick, 1000)
}
componentWillUnmount() {
  clearInterval(this.interval)
}
```

Use Effect , Issue 1

- To avoid Combine the two sets of unrelated code to group together?

```
componentDidMount() {
  document.title = `You clicked ${this.state.count} times`;
  this.interval = setInterval(this.tick, 1000)
}
componentDidUpdate() {
  document.title = `You clicked ${this.state.count} times`;
}
componentWillUnmount() {
  clearInterval(this.interval)
}
```

Use Effect ?

- The effect hook helps us to solve the problem what we discussed just now
- It is close replacement for **componentDidMount, ComponentDidUpdate and componentWillUnMount**

Multiple useEffects

```
function FriendStatusWithCounter(props) {
  const [count, setCount] = useState(0);
  useEffect(() => {
    document.title = `You clicked ${count} times`;
  });

  const [isOnline, setIsOnline] = useState(null);
  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
  });
  // ...
}
```

https://reactjs.org/docs/hooks-effect.html

useReducer

- useReducer is a hook that is used for state management
- It is an alternative to useState
- What's the difference between useState and useReducer?
- useState is built using useReducer
- When to useReducer vs useState?

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

Hooks so far

- useState : state
- useEffect : sideEffects
- useContext : context API
- useReducer : reducers

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

Reducer vs useReducer

reduce in JavaScript	useReducer in React
array.**reduce**(***reducer***, initialValue)	**useReducer**(***reducer***, initialState)
singleValue = ***reducer***(accumulator, itemValue)	newState = ***reducer***(currentState, action)
reduce method returns a single value	**useReducer** returns a pair of values. [newState, dispatch]

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

useReducer Summary

useReducer is a hook that is used for state management in React

useReducer is related to reducer functions

useReducer(reducer, initialState)

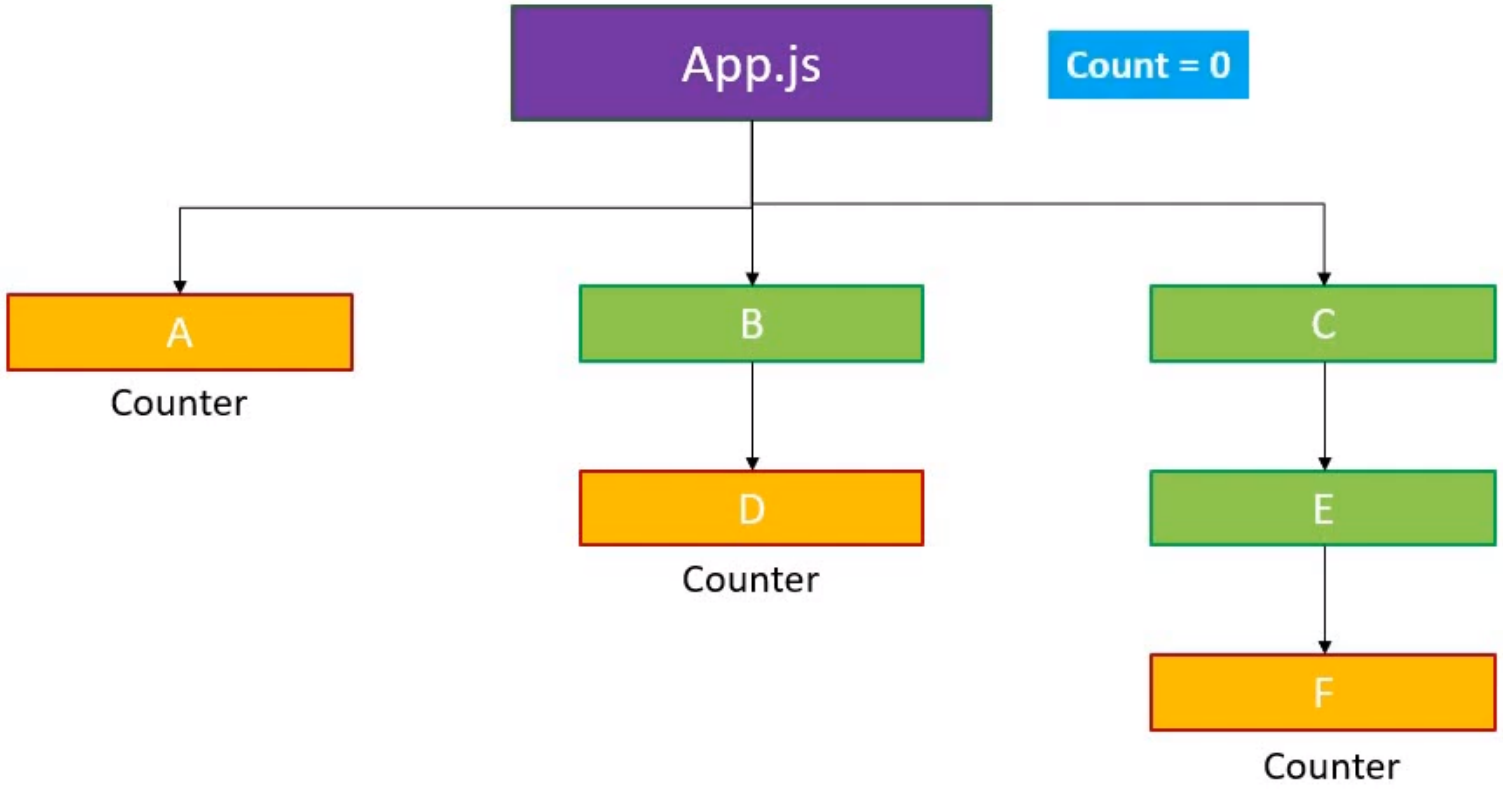
reducer(currentState, action)

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

useReducer with contextAPI

- useReducer – Local State Management
- Share state between components – Global State Management
- useReducer + useContext

useReducer with contextAPI



useState vs useReducer

Scenario	useState	useReducer
Type of state	Number, String, Boolean	Object or Array
Number of state transitions	One or two	Too many
Related state transitions?	No	Yes
Business logic	No business logic	Complex business logic
Local vs global	Local	Global

useCallback

What?

useCallback is a hook that will return a memoized version of the callback function that only changes if one of the dependencies has changed.

Why?

It is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

How?

https://kentcdodds.com/blog/usememo-and-usecallback

Hooks so far

- useState : Define State
- useEffect : Life Cycle
- useContext : To Create Context
- useReducer : Alternative to useState
- useCallback : returns meoized callback
- useMemo : returns meoized value
- useRef : mutable ref object
- Custom Hook?

https://reactjs.org/docs/hooks-reference.html

Custom Hooks

A custom Hook basically a JavaScript function whose name starts with “use”.

A custom Hook can also call Hooks if required

Why?

Share logic – alternative to HOC’s and Render Props

How to create custom Hooks?

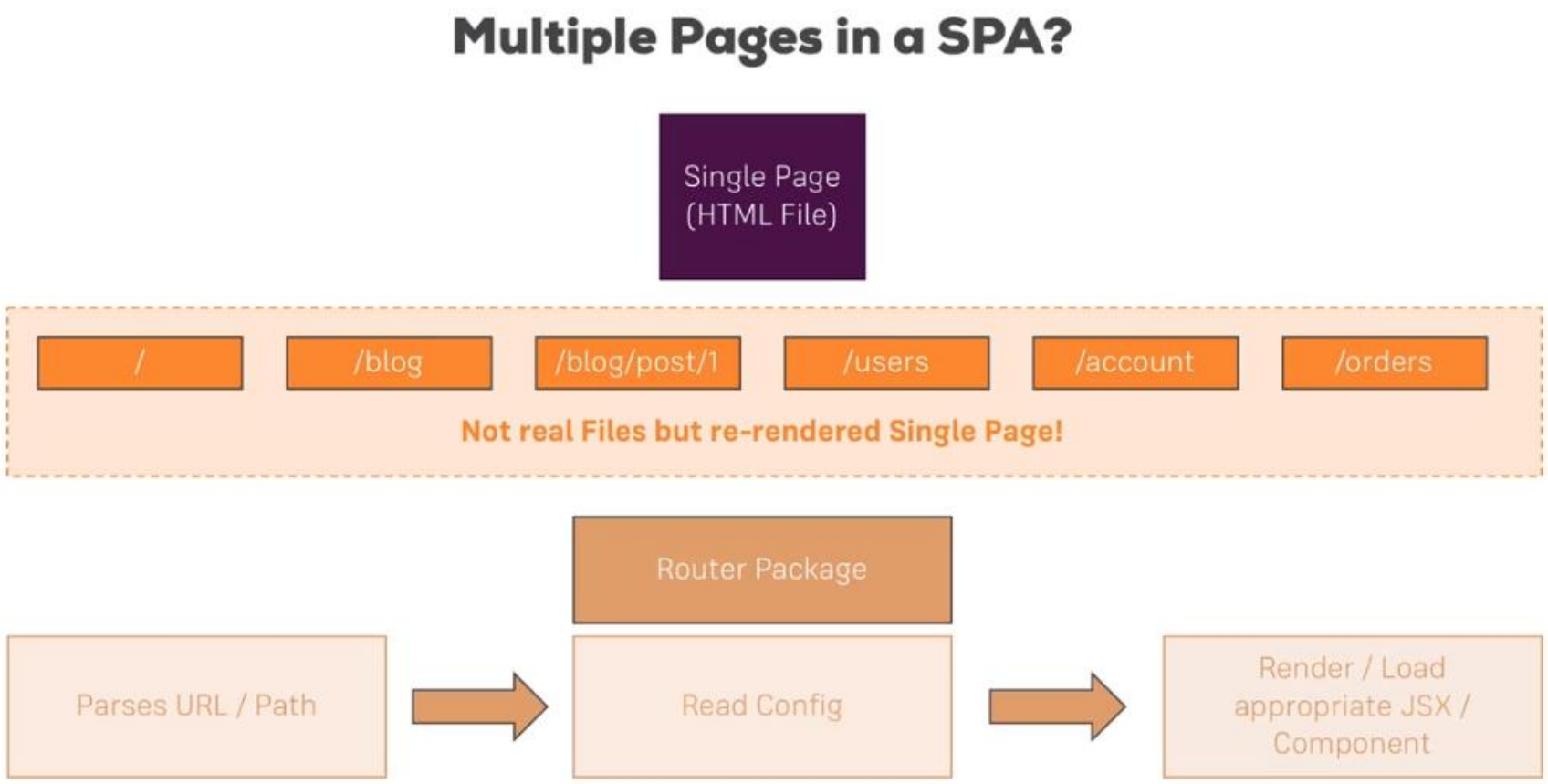
Routes

- React is simple component creation library
- Multiple pages are there in any application
- How do we switch urls?

npm i –save react-router

npm i –save react-router-dom

Routing Package



JEST Testing Library

What is Testing?

In order to accomplish a sophisticated testing toolset, you might have to address the following **test types** for your application.

Unit Test:

- A unit test examines each small piece of your code. You may think of it as testing primitive components in their life cycles. This is often the simplest and least expensive testing option.

Integration Test:

- If you have lots of composed components, you may want to test how they interact with each other. You can do this by mocking your endpoints as part of an integration test. This can be costlier and more complex than unit testing.

End-to-end Test (E2E Test):

- When it comes time to test the whole system with real data to see whether everything works as expected, end-to-end testing is your best bet.

Testing Libraries

- Jest
- Jasmine
- React-testing-library
- Enzyme
- Cypress
- Puppeteer

Jest & Jasmine

	Jest	Jasmine
Maintainers	Facebook	Pivotal labs
Github	29K ★ 4.1K forks	14.6K ★ 2.2K forks
Easy to adapt in-project	★★★★★ (CRA supports natively)	★★★ (requires configuration)
Test runner	✅	✅
Test framework	✅	✅
Documentation	★★★★★	★★★★
Snapshot testing	✅	❌
Code coverage support	✅	❌
Parallelization	✅	❌
DOM manupulation	✅	❌
Asserts	★★★★★	★★★★★
Async function test	✅	✅
Mocking functions	✅	✅ (with some limitations)

React-testing-library vs Enzyme

	`react-testing-library`	Enzyme
Github	10.1K ★ 530 forks	18.3K ★ 2K forks
Shallow render	❌	✅
Full-DOM rendering	❌	✅
Fire events	✅	✅
Exported APIs	★★★★★ (easy to use)	★★★★ (complex)
Easy to adapt project	✅	❌ (requires setup/adapters)
Testing without implementation detail	✅	❌

Cypress - Puppeteer

	Cypress	Puppeteer
Github	17.1K ★ 918 forks	51.1K ★ 5.6K forks
Documentation	★★★★★	★★★★★
Simple usage	★★★★★	★★★
Browser supports	★★★★	★ (Only Chromium)
Parallelization	✅	✅ (with testing framework)
Learning curve	❤️	😢
Price	Freemium	Free

What is JEST?

Jest is a testing framework created and maintained by Facebook. If you build your React application with create-react-app, you can start using Jest with zero config. Just add react-test-renderer and @testing-library/react library to conduct snapshot and DOM testing.

- Jest is the testing framework used at Facebook to test React components and is adopted by Uber, Airbnb and other teams.
- The React community, therefore, recommends Jest as the React testing framework of choice.
- With over 16M downloads a week, Jest is probably the most popular testing framework for React.

Summary

- As you can see, each testing method, library, and framework brings its own advantages and shortfalls
- Depending on the use case and types of data you wish to analyze you can choose framework
- react-testing-library is the most valuable and logical choice for unit and integration tests.
- JEST & ENZYME combination meets most features to test your app as per the community forums

Why is JEST?

- Zero configuration testing platform
- Conduct snapshot, parallelization, and async method tests
- Mock your functions, including third-party node_module libraries
- Execute myriad assertion methods
- View code coverage report

Why JEST

Zero configuration

Jest aims to work out of the box, config free, on most JavaScript projects.

Snapshots

Make tests which keep track of large objects with ease. Snapshots live either alongside your tests, or embedded inline.

Isolated

Tests are parallelized by running them in their own processes to maximize performance.

Great Api

From it to expect - Jest has the entire toolkit in one place. Well documented, well maintained, well good.

THANK YOU

Corporate Trainer & Motivational Speaker
9035351965