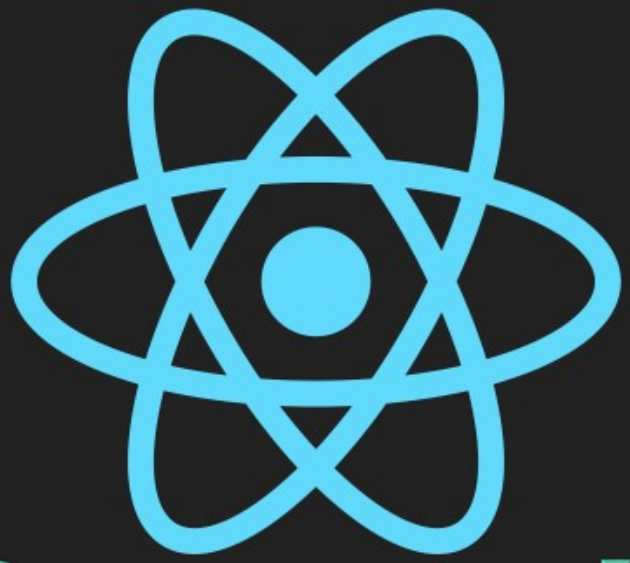
తెలుగు లో

PART - 1



React JS

Crash Course

Introduction

Applications of

Virtual DOM

#Jobs

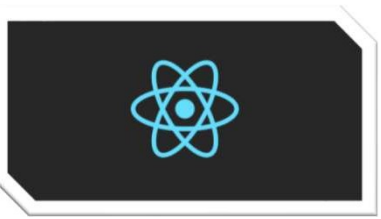
#Trendy

Create Project


node
JS
®

Contents

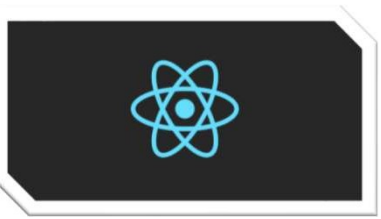
- ReactJS Introduction
- Applications of React
- Virtual DOM
- Popularity of React
- Create Project
- Display “Hello World!”



Before Proceeding to React JS...

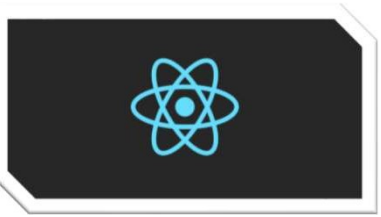
You should be familiar with the following languages.

- HTML
- CSS
- JavaScript (including ES6 features)



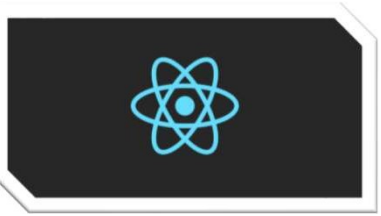
What is React JS?

- React JS is UI Library
- Developed By Facebook
- It's Open Source
- It's released on 2013
- Current Stable Version 17.0.1

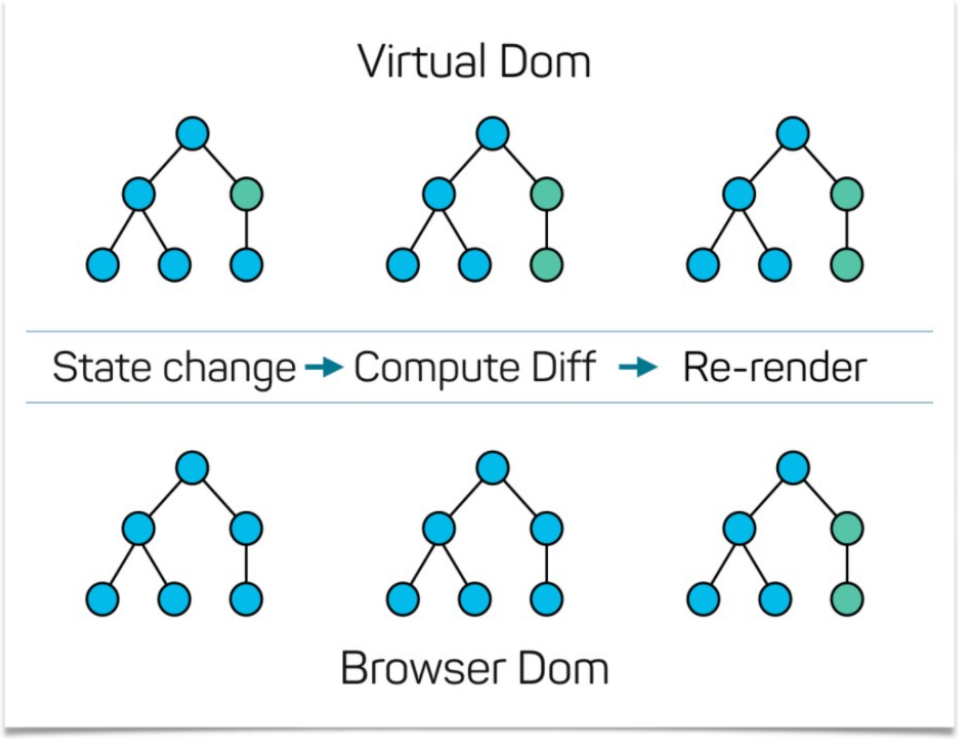


Applications of React JS

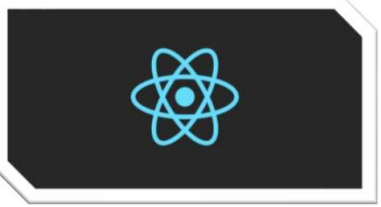
- React JS can be used to develop Single Page Web Applications.
- React Native with the similar React JS syntax can be used to develop Cross Platform Mobile Applications.
- It uses Virtual DOM for updating the DOM changes.



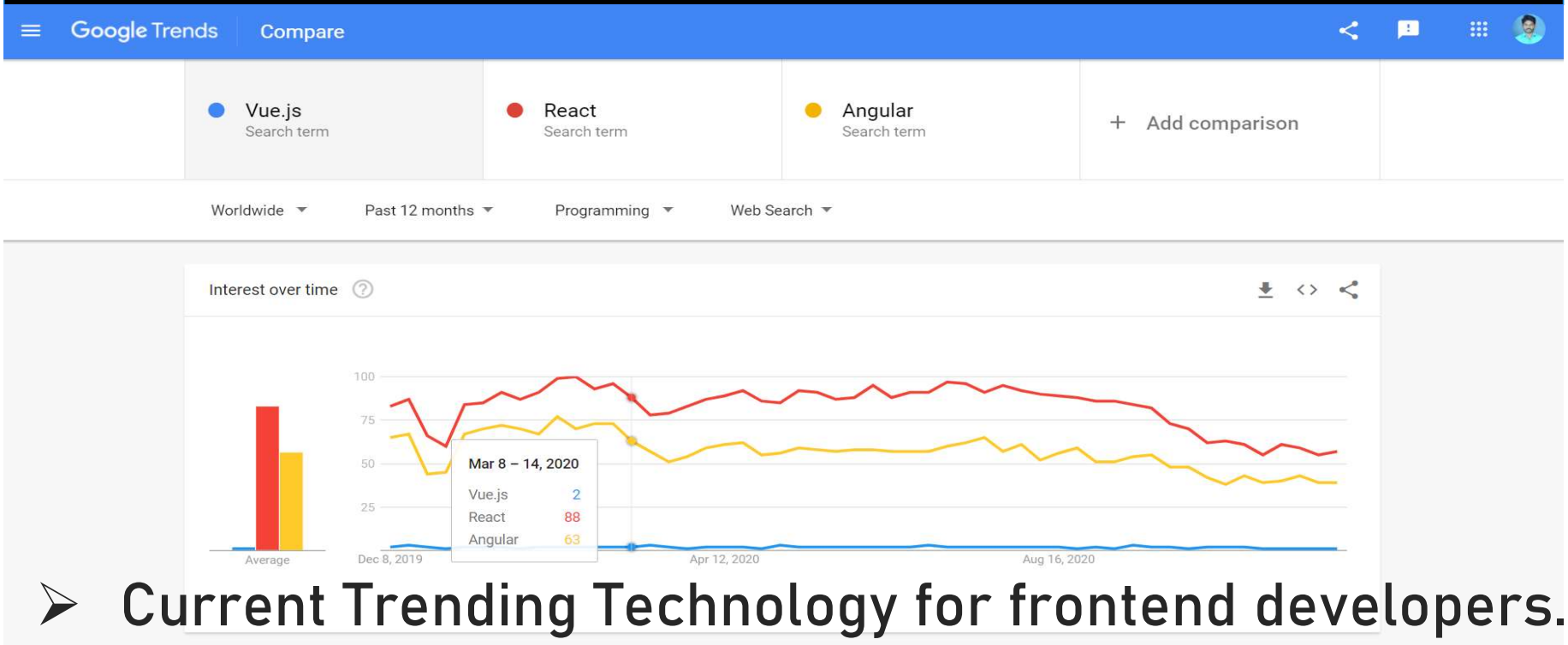
Virtual DOM



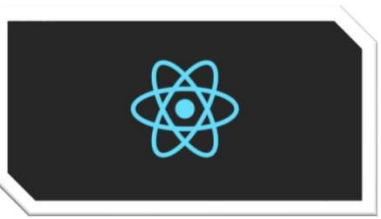
- It takes lightweight copy of Real DOM and keeps in memory and syncs with the Real DOM by using ReactDOM Library.
- It improves lot of performance.



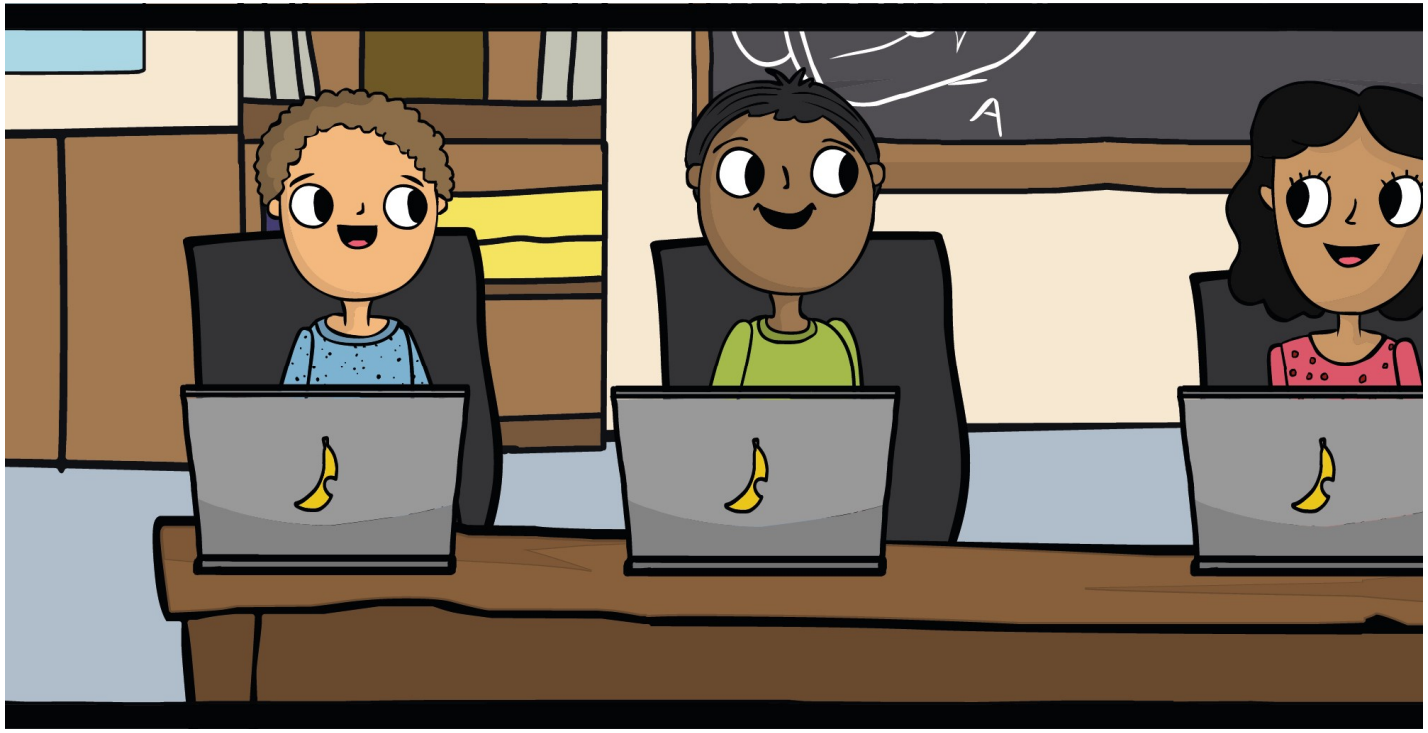
Popularity Of React JS

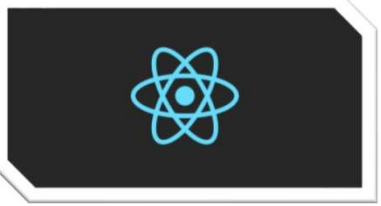


- Current Trending Technology for frontend developers.



Let's Start Coding...



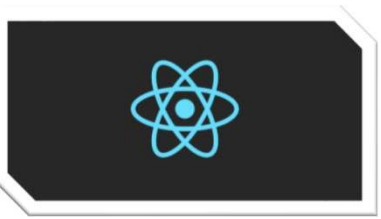


How to Create React JS Project?

- npm install create-react-app -g
- create-react-app myproject

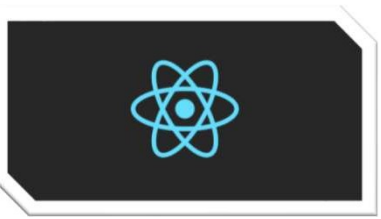
OR

- npx create-react-app myproject

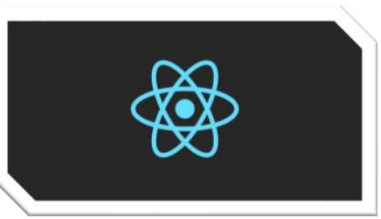


Naming Convention for Project?

- Project Name should be URL Friendly.
- It should not contain any special characters except hyphen(-) and underscore(_)
- It should not contain any Capital Letters.



Hello World!



తెలుగు లో

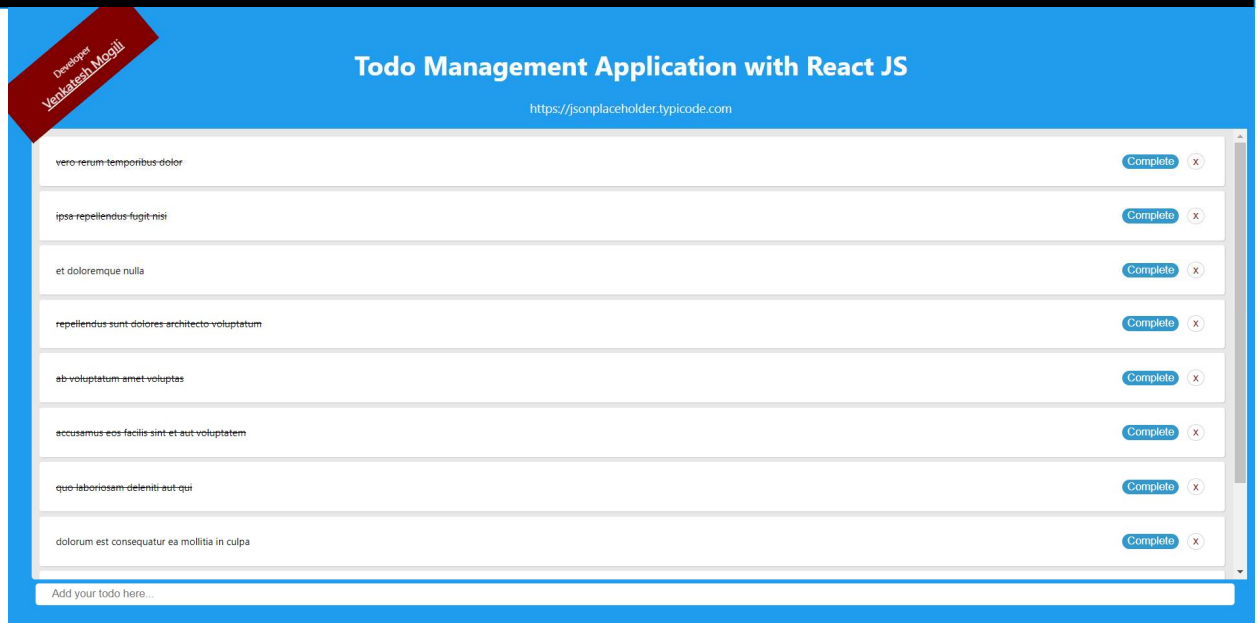
PART - 2

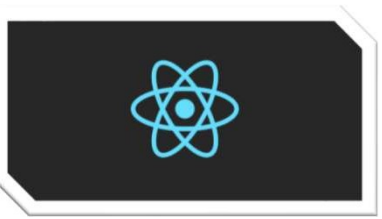
React JS

Crash Course

Contents

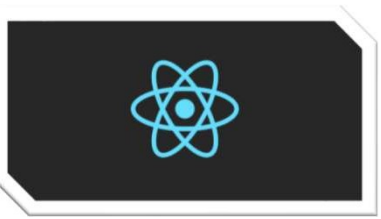
- JSX?
- Components
- Props
- State
- LifeCycle Methods
- Event Handling
- Rendering
- Lists
- Forms
- Todo Management Application





Useful VS Code Extensions

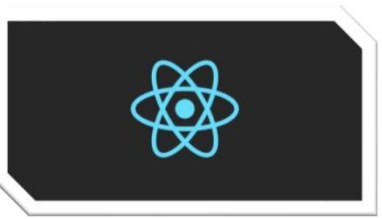
- Prettier Now
- ES7/React Snippets



What is JSX?

- JSX stands for JavaScript XML which means combination of both JavaScript and XML syntax.
- JavaScript syntax:
  - let button;
- XML syntax:
  - <button>Click here</button>

- JSX syntax:
  - let button = <button>Click here</button>;



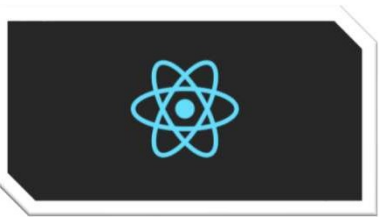
Is JSX Mandatory?

- It's recommended to use JSX for React Applications.
- It's not mandatory, We can use babel syntax instead of JSX.
- Babel syntax:
- import React from 'react';
- let button = React.createElement('button',null,'Hello');
- Equivalent syntax in JSX:
- let button = <button>Hello</button>;

HTML Tag/React Component

Children

Properties



Any extra syntax changes?

- class → className
- onclick → onClick
- onchange → onChange
- checked → defaultChecked
- selected → defaultValue
- for → htmlFor
- style="normal css" → style={ {camel case css}}

```
<div className="col-md-12"></div>
```

```
<button onClick={}>Click here</button>
```

```
<select onChange={}>Click here</select>
```

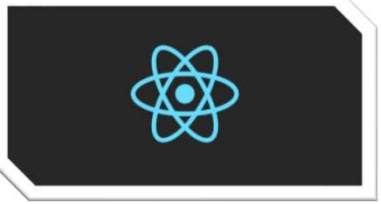
```
<input type='checkbox' defaultChecked />
```

```
<select type='checkbox' defaultValue></select>
```

```
<label htmlFor="firstname">FirstName</label>
```

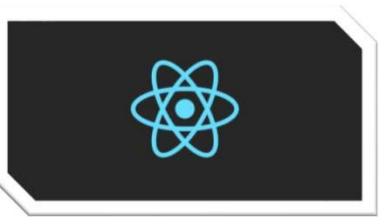
```
<button style={{backgroundColor:"blue",fontSize:"30px"}}>Stylish Button</button>
```

Stylish Button



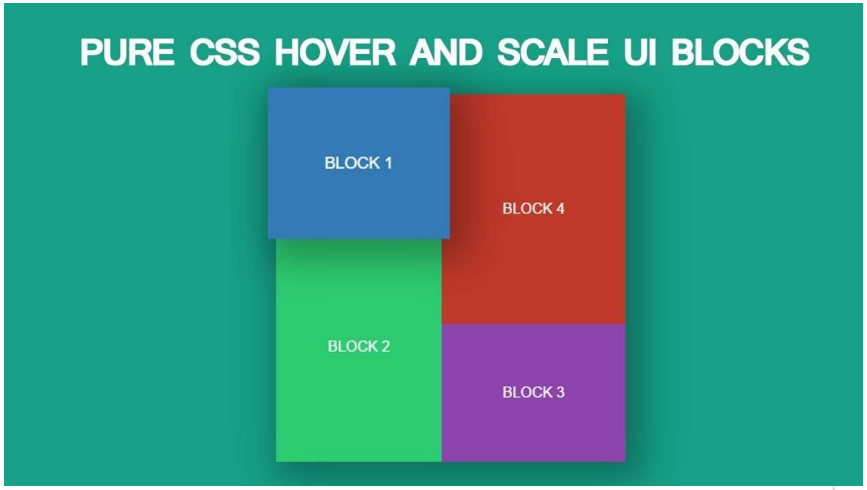
JSX Summary

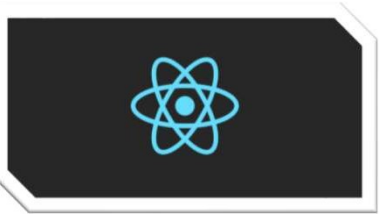
- JSX is a special syntax for creating user interface using React Library.
- JSX stands for JavaScript XML.
- It's not mandatory to use but recommended to use for learning React fast.
- Babel is the compiler for JSX Syntax.
- Few attribute changes such as className,onClick….



What is Component?

- Any UI block can be considered as a Component.
- It can be reusable. Create once use anywhere.





Types of Components in React?

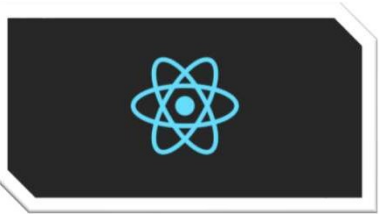
- There are 2 types of components available in React.
- 1.Functional Component
- 2.Class Component

```jsx
import React from 'react'

export default function Button() {
    return (
        <div>
            <button>Submit</button>
        </div>
    )
}
```

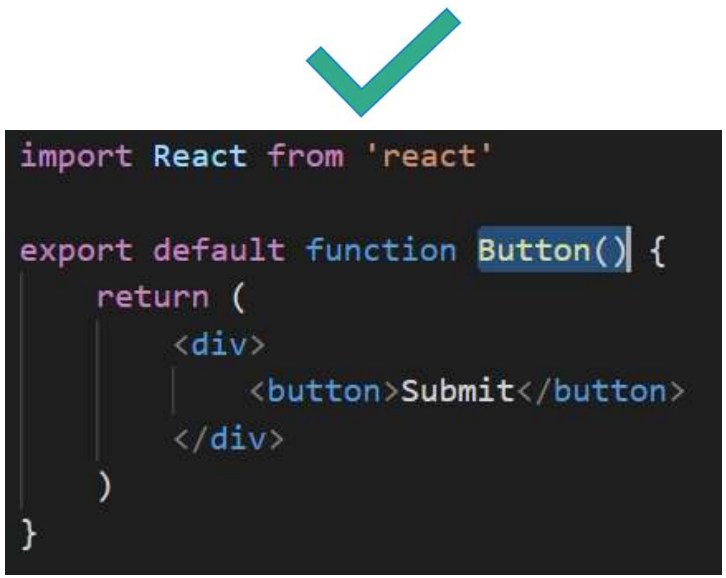
```jsx
import React, { Component } from 'react'

export default class Button extends Component {
    render() {
        return (
            <div>
                <button>Submit</button>
            </div>
        )
    }
}
```



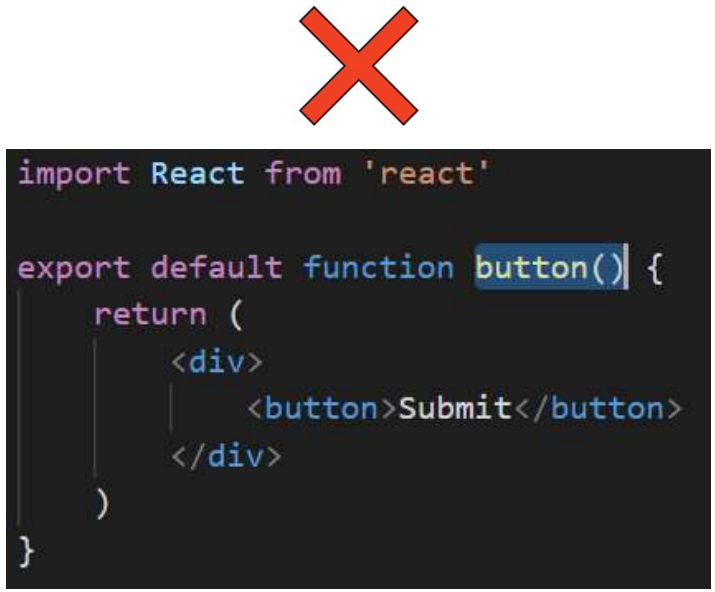
Component Naming Convention?

- React Component Name should start with Capital Letter.



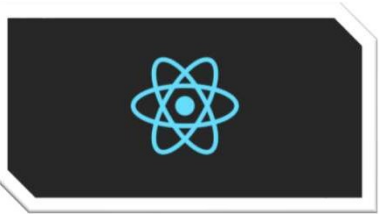
```jsx
import React from 'react'

export default function Button() {
    return (
        <div>
            <button>Submit</button>
        </div>
    )
}
```



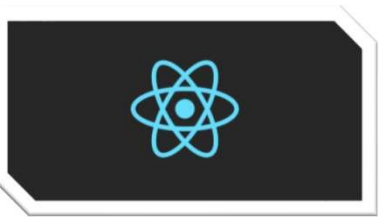
```jsx
import React from 'react'

export default function button() {
    return (
        <div>
            <button>Submit</button>
        </div>
    )
}
```



Components Summary

- Any UI block can be considered as a Component.
- It can be reusable. Create once use anywhere.
- There are 2 types of components available in React.
  1.Functional Components & 2.Class Components
- React Component Name should start with Capital Letter.



Props in React?

- Props full form is Properties, which means simply attributes.
- Props will be used to pass the data between the components.
- We can pass any valid JavaScript expression as prop.

App Component → Button Component

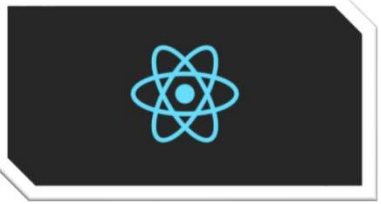
```jsx
import React from 'react';
import Button from './components/Button';

function App() {
    return (
        <div>
            <Button title='Click here' />
        </div>
    );
}
export default App;
```

```jsx
import React from 'react';

export default function Button(props) {
    return (
        <div>
            <button>{props.title}</button>
        </div>
    );
}
```

Output: Click here



Props in React?

- Data can be passed in one way only. Unidirectional or top-down.
- From the below example, title is passed from App component, but Button component won't know whether it's value is typed manually or passed from state variable or passed from it's parent component.

App Component → Button Component

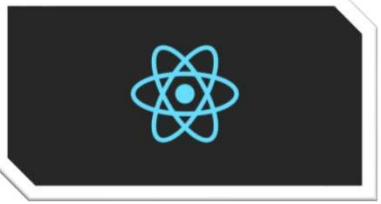
```jsx
import React from 'react';
import Button from './components/Button';

function App() {
    return (
        <div>
            <Button title='Click here' />
        </div>
    );
}
export default App;
```

```jsx
import React from 'react';

export default function Button(props) {
    return (
        <div>
            <button>{props.title}</button>
        </div>
    );
}
```

Output: Click here



State in React?

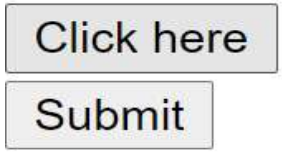
- State is a local variable for the component.
- Any state in one component won't be accessible to other components, which means by default states are encapsulated in components.

App Component

```
import React, { useState } from 'react';
import Button from './components/Button';

function App() {
    const [ title ] = useState('Click here');
    return (
        <div>
            <button>{title}</button>
            <Button />
        </div>
    );
}
export default App;
```

Output:



Button Component

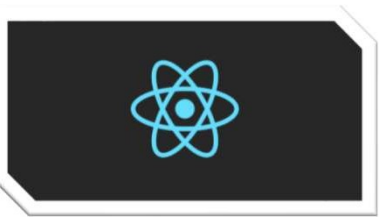
```
import React, { Component } from 'react';

export default class Button extends Component {
    state = {
        title: 'Submit',
    };
    render() {
        return (
            <div>
                <button>{this.state.title}</button>
            </div>
        );
    }
}
```

Output:

Props and State Summary

- Properties are nothing but attributes, which can be used to pass the data from one component and another component.
- Any valid JavaScript expression can be passed a prop.
- State is local variable of the component, which is by default encapsulated to it's own component.
- setState method can be used to update the state values.



తెలుగు లో

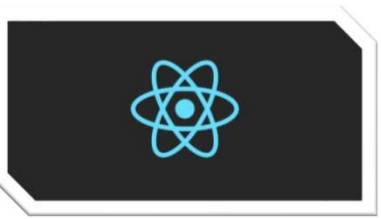
PART - 3

React JS

Crash Course

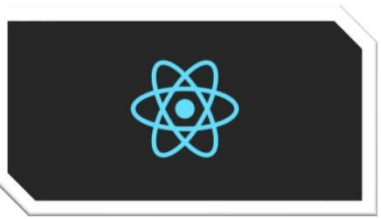
Contents

- Project Structure
- Styles
- LifeCycle Methods
- Event Handling
- Rendering
- Lists
- Forms



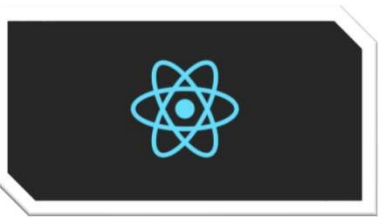
Contents

- ✓ Project Structure
- ➢ **Styles**
- ➢ LifeCycle Methods
- ➢ Event Handling
- ➢ Rendering
- ➢ Lists
- ➢ Forms



Contents

- ✓ Project Structure
- ✓ Styles
- ➢ **LifeCycle Methods**
- ➢ Event Handling
- ➢ Rendering
- ➢ Lists
- ➢ Forms



LifeCycle methods in Class Component?

React component has three lifecycle methods.

- ✓ 1.Mounting (Putting elements into the DOM)
- ✓ 2.Updating (when component is updated)
- ✓ 3.Unmounting (when component is removed from the DOM)

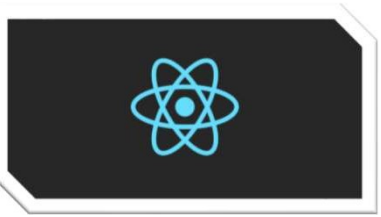
I) Mounting has the below order of execution.

1.constructor() →optional

2.getDerivedStateFromProps() →optional

3.render() →Mandatory

4.componentDidMount() →optional



LifeCycle methods in Class Component?

II) Updating has the below order of execution.

1.getDerivedStateFromProps() →optional

2.shouldComponentUpdate() →optional

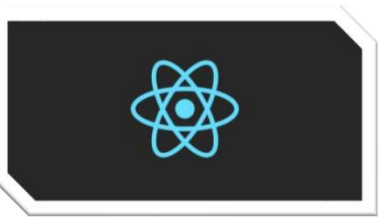
3.render() →Mandatory

4.getSnapshotBeforeUpdate() →optional

5.componentDidUpdate() →optional

III) Unmounting will run the below lifecycle method.

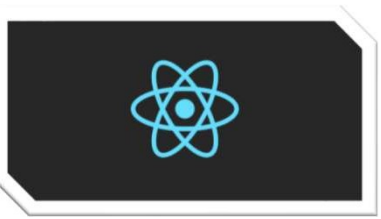
1.componentWillUnmount()



LifeCycle methods in Functional Component?

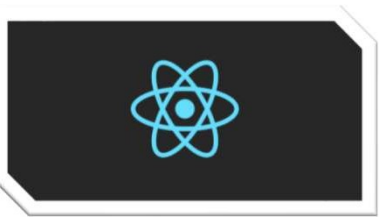
- The below useEffect hook will be used for all the three lifecycle phases.

```
useEffect(()=>{
},[ ])
```



Contents

- ✓ Project Structure
- ✓ Styles
- ✓ LifeCycle Methods
- ➢ **Event Handling**
- ➢ Rendering
- ➢ Lists
- ➢ Forms

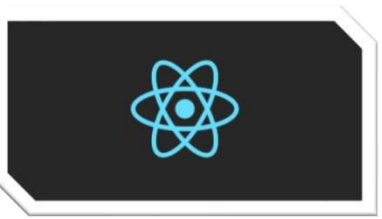


Event Handling

- When user performs any action, it should be considered as an event.
- onClick button, onChange dropdown values, onCheck checkbox etc.,

```
import React, { Component } from 'react';

export default class Button extends Component {
    constructor(props) {
        super(props);
        this.state = {
            title: 'Submit',
        };
        this.submit = this.submit.bind(this);
    }
    submit() {
        console.log('Submitted');
    }
    render() {
        return (
            <div>
                <button onClick={this.submit}>{this.state.title}</button>
            </div>
        );
    }
}
```

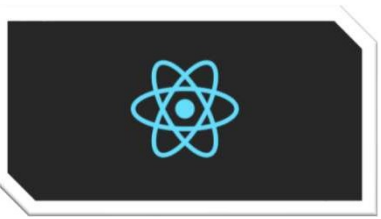


Event Handling

- Inline Binding

```
import React, { Component } from 'react';

export default class Button extends Component {
    constructor(props) {
        super(props);
        this.state = {
            title: 'Submit',
        };
    }
    submit() {
        console.log('Submitted');
    }
    render() {
        return (
            <div>
                <button onClick={this.submit.bind(this)}>{this.state.title}</button>
            </div>
        );
    }
}
```

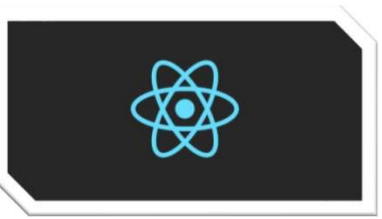


Event Handling

- ES6 Arrow Function

```
import React, { Component } from 'react';

export default class Button extends Component {
    constructor(props) {
        super(props);
        this.state = {
            title: 'Submit',
        };
    }
    submit = () => {
        console.log('Submitted');
    };
    render() {
        return (
            <div>
                <button onClick={this.submit}>{this.state.title}</button>
            </div>
        );
    }
}
```

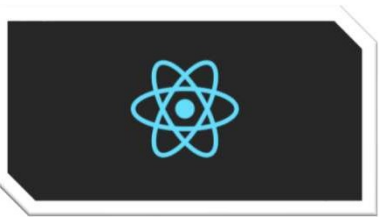


Event Handling

- ES6 Inline Arrow Function

```
import React, { Component } from 'react';

export default class Button extends Component {
    constructor(props) {
        super(props);
        this.state = {
            title: 'Submit',
        };
    }
    submit() {
        console.log('Submitted');
    }
    render() {
        return (
            <div>
                <button onClick={() => this.submit()}>{this.state.title}</button>
            </div>
        );
    }
}
```

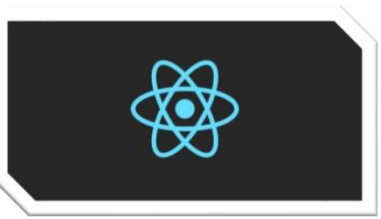


Event Handling

- Passing the arguments to the function from the event.

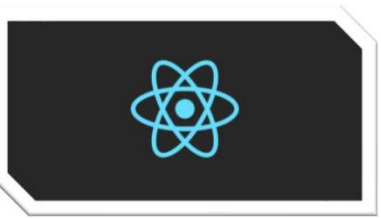
```
import React, { Component } from 'react';

export default class Button extends Component {
    constructor(props) {
        super(props);
        this.state = {
            title: 'Submit',
        };
    }
    submit(e) {
        console.log('Submitted', e);
    }
    render() {
        return (
            <div>
                <button onClick={(e) => this.submit(e)}>{this.state.title}</button>
            </div>
        );
    }
}
```



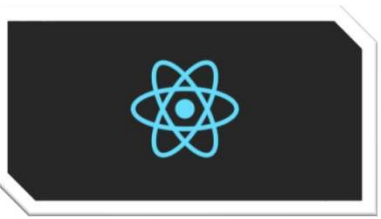
LifeCycle methods and Event Handling Summary

- React has 3 lifecycle methods, which are Mounting, Updating, Unmounting and render method is mandatory to display the output in DOM.
- Any event can be performed by user action such as onClick,onChange etc.,



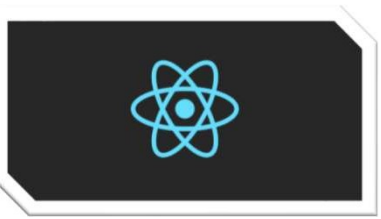
LifeCycle methods and Event Handling Summary

Break



Contents

- ✓ Project Structure
- ✓ Styles
- ✓ LifeCycle Methods
- ✓ Event Handling
- ➢ **Rendering**
- ➢ Lists
- ➢ Forms



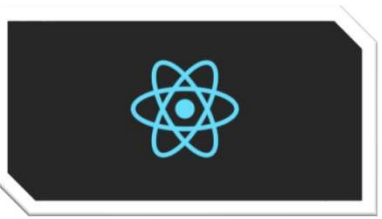
Conditional Rendering

JavaScript conditions will be executed like below.

```
If(a && b && c){
        console.log("output");
}
```

- If a is true, then only b will be executed, similarly c.
- We can use the same condition for elements whichever we want to display, we will write a condition before it.
- Example:

```
{this.state.user && <h1>{this.state.username}</h1>}
```



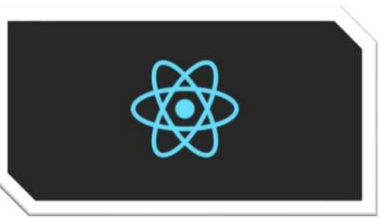
Using Ternary Operator

JavaScript Ternary operation looks like below code.

a?"display":"won't display";

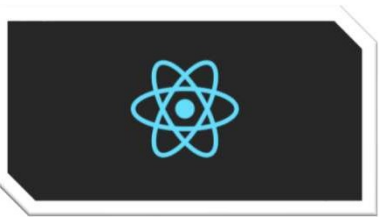
- If a is true, then it will display, otherwise it won't display.
- We can use the same operation for elements whichever we want to display, we will write the ternary operator before it. If condition is false, we should pass null or '' (empty string) or any false value.
- Example:

{this.state.user?<h1>{this.state.username}</h1>:null}



Contents

- ✓ Project Structure
- ✓ Styles
- ✓ LifeCycle Methods
- ✓ Event Handling
- ✓ Rendering
- ➢ **Lists**
- ➢ Forms

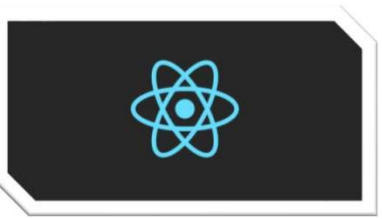


Displaying List of Items

JavaScript ES6 has map function which iterates the loop and returns the new values.

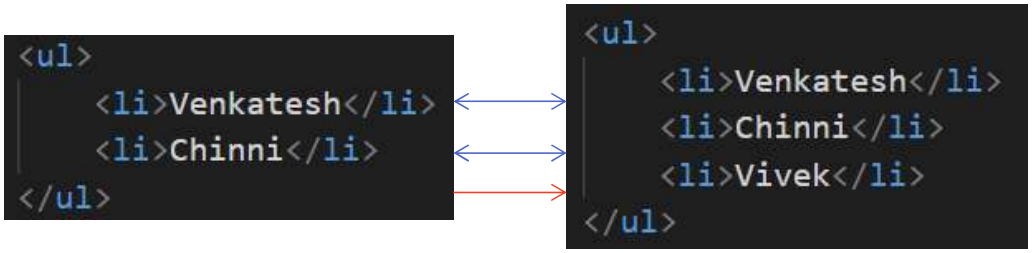
- Example:

```
this.state={
        users : [{name:"Venkatesh"},{name:"Chinni"}];
    }
{this.state.users.map((user)=>(
        <p>{user.name}</p>
        )
)}
```

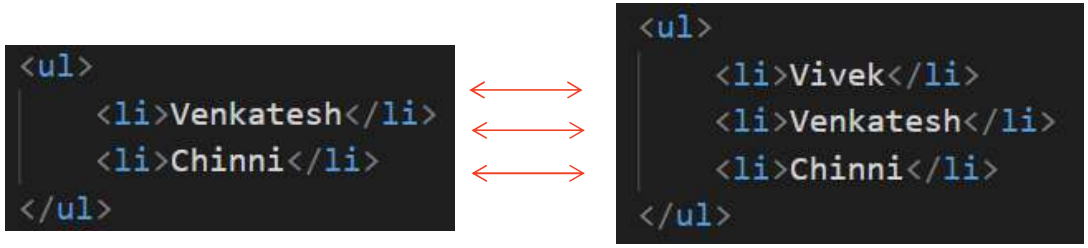


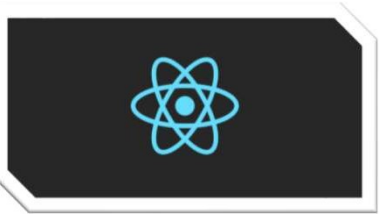
List without keys

Add Item to end



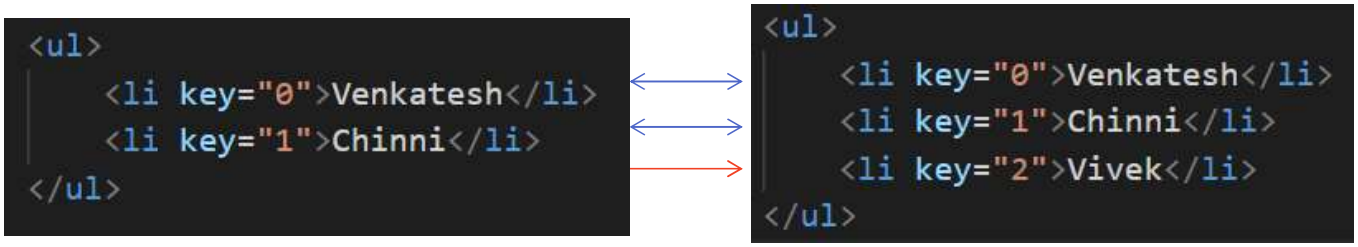
Add Item to start



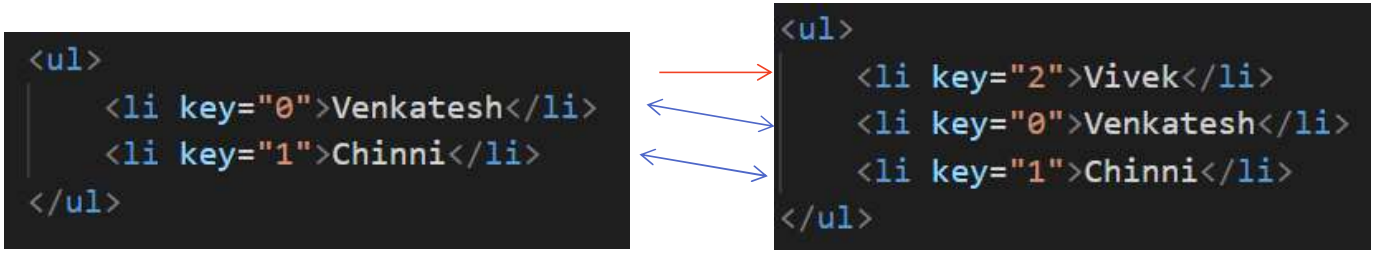


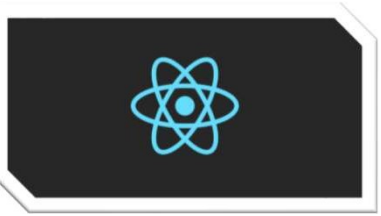
List with keys

Add Item to end



Add Item to start



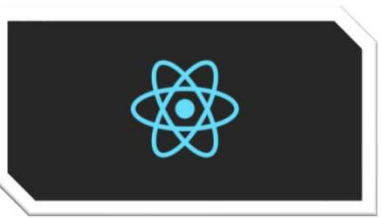


Displaying List of Items

JavaScript ES6 has map function which iterates the loop and returns the new values.

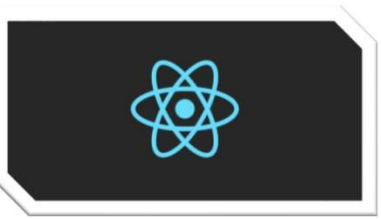
- Example:

```
this.state={
    users : [{name:"Venkatesh"},{name:"Chinni"}];
 }
{this.state.users.map((user,index)=>(
   <p key={index}>{user.name}</p>
   )
 )}
```



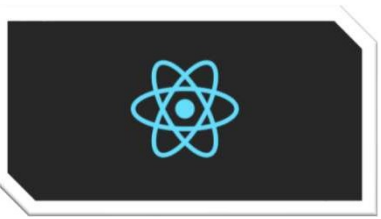
Contents

- ✓ Project Structure
- ✓ Styles
- ✓ LifeCycle Methods
- ✓ Event Handling
- ✓ Rendering
- ✓ Lists
- ➢ **Forms**



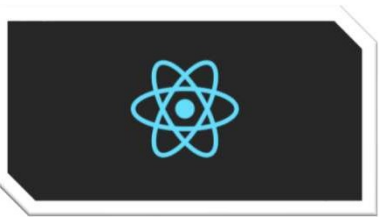
Forms

- Forms can be used as we use in html.
- When we do onSubmit, we can prevent reload the page by using event.preventDefault() which won't reload the page and output will be displayed.



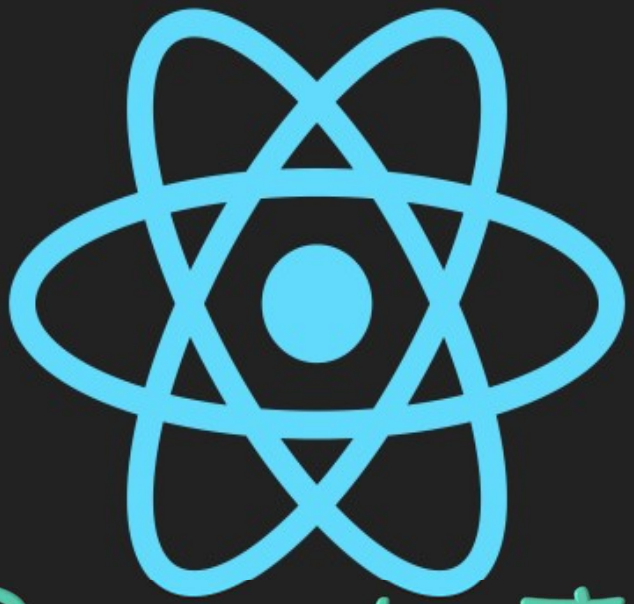
Lifting State Up

- When we implement individual components with their own functionalities, there can a situation to use those functionalities in it's parent component or in it's sibling component.
- In this case, we will completely move the functionalities to it's parent component. Which is called State Lifting.



తెలుగు లో

Part-4



React JS

Crash Course

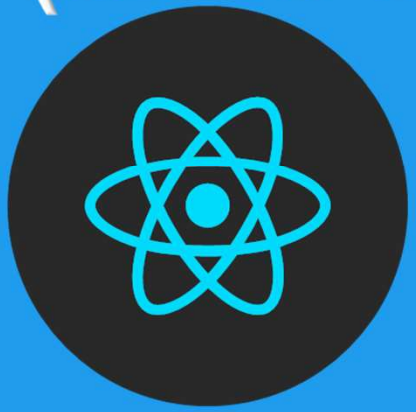
File Upload using React & Cloudinary

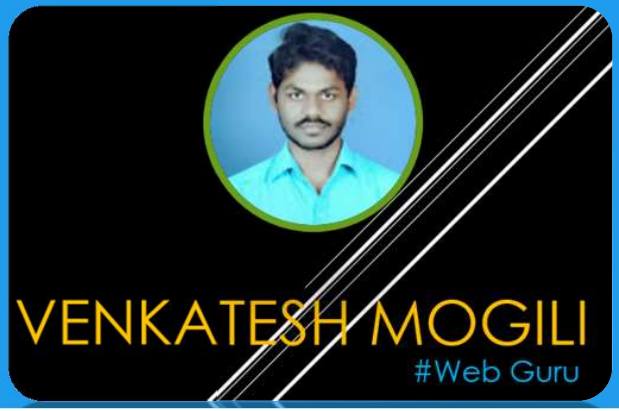
Choose Files No file chosen

Upload

File Upload (Multi and Single)

తెలుగు లో





Venkatesh Mogili

#WebGuru

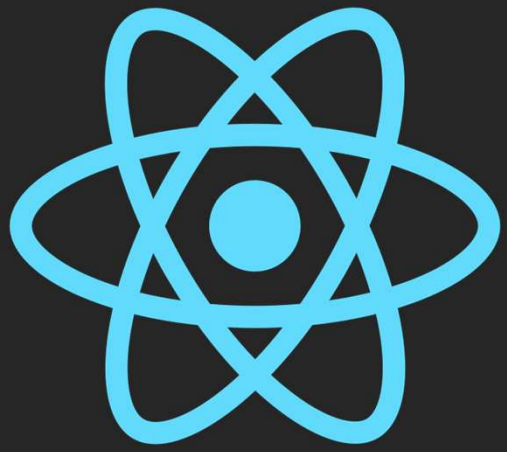
React JS

Crash Course

తెలుగు లో

PART – 5

Routing



Contents

- **Routing Introduction**
- Routers
- Route Matchers
- Route Navigation
- Routing with Hooks
- Authentication with route
- Lazy Loading/Code splitting Components
- Example Scenarios
- Summary

Prerequisites and Project Setup

- ✓ Before learning Routing concept, better to have basic knowledge on React concepts like props, components, children, JSX syntax.

- ➢ Create a react project.

  npx create-react-app routing-crash-course

- ➢ Install react-router-dom package.

  npm i react-router-dom

- ➢ Install Bootstrap and import in index.css file

  npm i bootstrap@next

```
@import 'bootstrap/dist/css/bootstrap.min.css';
```

Routing

- Routing means simply navigating from one URL to another URL without refreshing the application every time.

Primary Components:

There are three primary categories of components in React Router:

1) **routers**, like <BrowserRouter> and <HashRouter>
2) **route matchers**, like <Route> and <Switch>
3) **navigation**, like <Link>, <NavLink>, and <Redirect>

- All of the components that you use in a web application should be imported from **react-router-dom**

- **react-router-dom (for web)**
- **react-router-native (for mobile)**

Contents

- ✓ Routing Introduction
- ➢ **Routers**
- ➢ **Route Matchers**
- ➢ **Route Navigation**
- ➢ Routing with Hooks
- ➢ Authentication with route
- ➢ Lazy Loading/Code splitting Components
- ➢ Example Scenarios
- ➢ Summary

Browser Router VS Hash Router

1) Browser Router:

When we have big production-ready applications which serve backend, it is recommended to use <BrowserRouter>

Example: http://localhost:3000/about

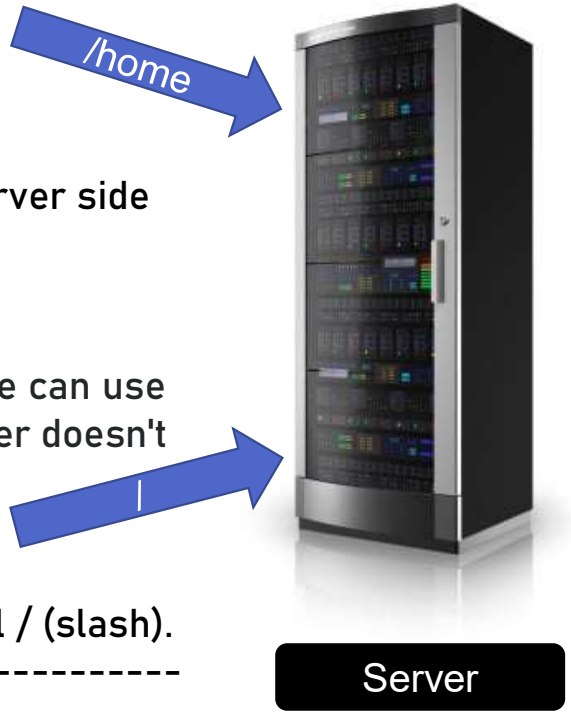
As server supports normal url system, it will take the complete path for server side api calls for GET requests.

2) Hash Router:

When we have small client side applications which doesn't need backend we can use <HashRouter> because when we use hashes in the URL/location bar browser doesn't make a server request.

Example: http://localhost:3000/#/about

Server will ignore about as it appended after #, server only takes the url till / (slash).



---

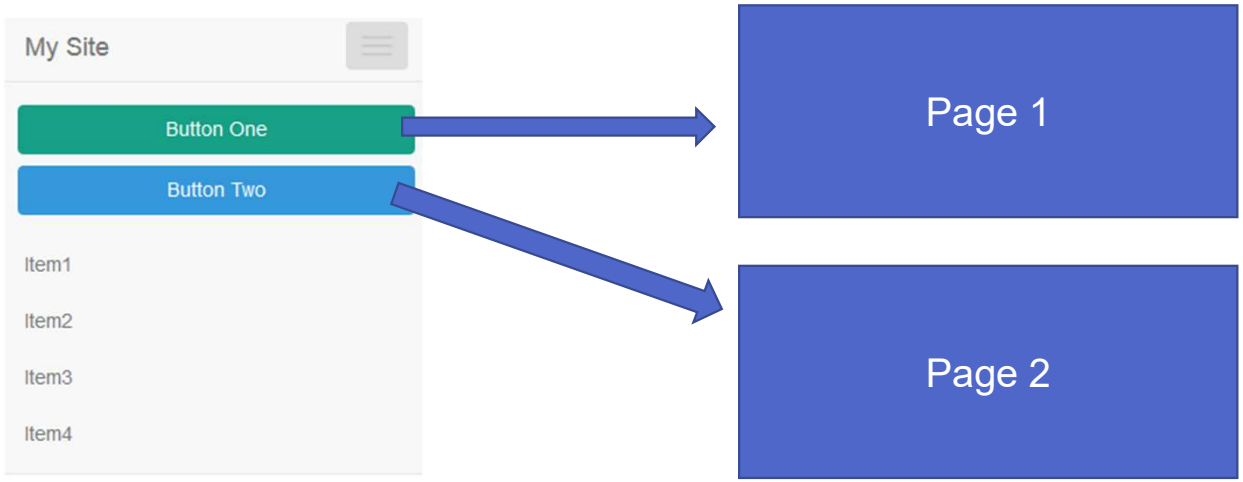
forceRefresh can imitate the normal url functionality.
basename can be used to give a global app url for our application.
getUserConfirmation Prompt before moving to another page can be checked

Switch

Switch:

- When a <Switch> is rendered, it searches through its children <Route> elements to find one whose path matches the current URL.
- When it finds one, it renders that <Route> and ignores all others.
- This means that you should put <Route>s with more specific (typically longer) paths **before** less-specific ones.
- If no <Route> matches, the <Switch> renders nothing (null).

Example:
/projects/:id
/projects
/about
/home
/dashboard



Route

Route:

***path** - (/projects, /home, /dashboard etc.,)

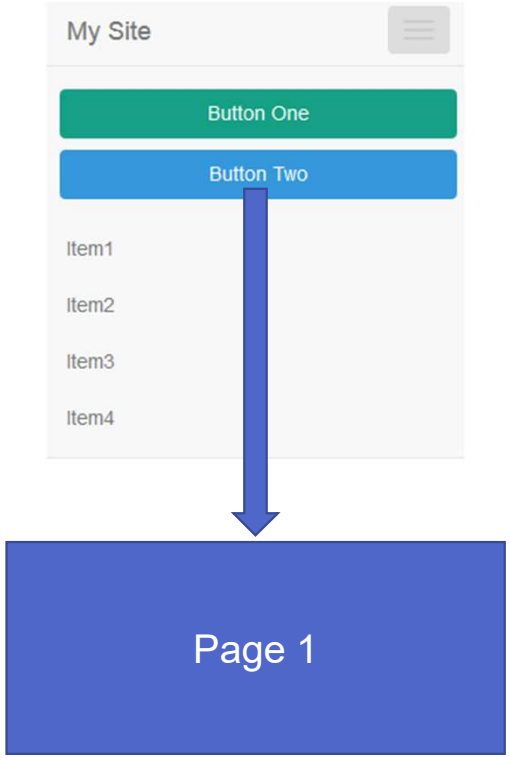
Every route has props (match, location, history)
component – we can pass component to render for particular route. (it will create a new React component whenever we render it)

render – we can use inline rendering for particular route. (it won't create new react component, instead it will take previous route props).

children – we can pass JSX elements to render for particular route. (similar to render but it works even if there is no match).

children>component>render

exact – to match exact url.(browser location. Works for **/about** also **/about/)**
strict – to match exact url including slash. (works for /about/ but not /about)
sensitive – to match url with case sensitive. (works for /About but not /about)

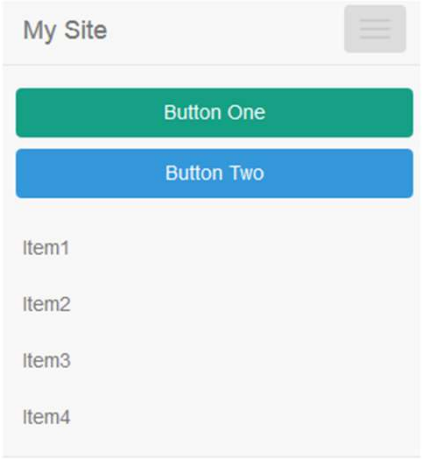


Navigation

Navigation:

It used to change navigations by using components like <Link>, <NavLink> and <Redirect>

<Link>: uses **to** prop to navigate to specified path.
<NavLink>: it will do the same like Link but also having extra feature like active link or not.
<Redirect>: it redirects to specified path.



Link, NavLink, Redirect

Link:

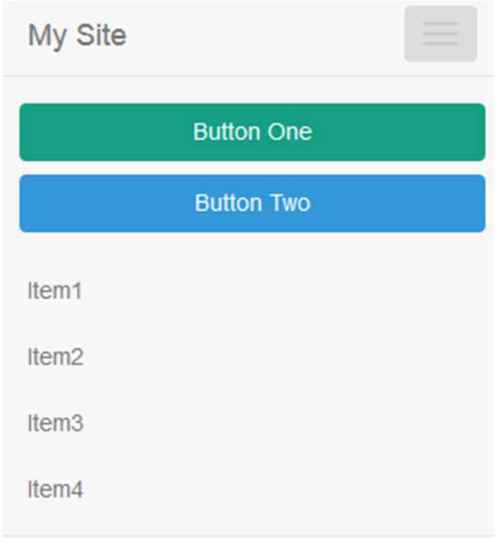
to - string (/home)
to - object ({to:"/home"})
to - function (location=>({...location,pathname:"/home"})

NavLink:

to – string
activeClassName – className
activeStyle – object styling.
exact – style applies only if url matches exactly
strict - style applies only if url matches exactly including slash (/)

Redirect:

to - string (/home)
to - object ({pathname:"/home"})
from – string



Contents

- ✓ Routing Introduction
- ✓ Routers
- ✓ Route Matchers
- ✓ Route Navigation
- ➢ **Routing with Hooks**
- ➢ Authentication with route
- ➢ Lazy Loading/Code splitting Components
- ➢ Example Scenarios
- ➢ Summary

Using Hooks

We can use hooks to do the similar things which can be done with Switch and Route.
useHistory, useLocation, useParams, useRouteMatch

1.useHistory: it handles push, pop etc., history functionality. (history.push("/home"))

2.useLocation: we can get location details by using this. (location.pathname)

3.useParams: it provides the params of the matched url. ({id})

4.useRouteMatch: we can do whatever things can be done with <Route> by using this hook.

Contents

- ✓ Routing Introduction
- ✓ Routers
- ✓ Route Matchers
- ✓ Route Navigation
- ✓ Routing with Hooks
- ➢ **Authentication with route**
- ➢ **Lazy Loading/Code splitting Components**
- ➢ **Example Scenarios**
- ➢ Summary

Example Scenarios

1. Default Homepage if url has only slash.
2. 404 Not Found! (if none of the paths are matching)
3. url parameters (by using useParams())
4. Auth redirects based on conditions.
5. Redirect from old site url to new site url
6. Nested routing
7. Sidebar routing
8. Query parameters

Contents

- ✓ Routing Introduction
- ✓ Routers
- ✓ Route Matchers
- ✓ Route Navigation
- ✓ Routing with Hooks
- ✓ Authentication with route
- ✓ Lazy Loading/Code splitting Components
- ✓ Example Scenarios
- ➢ **Summary**

Summary

1. Routing is very useful while developing single page applications or any kind of web/mobile applications which has lot of page transitions from one page to another.

2. Primary components in react-router-dom package are three.
    1. Routers
    2. Route matchers
    3. Navigation

3. We can use hooks to do the same functionalities as Switch, Route components.

4. We can use routing for many applications for many scenarios.

Exercise and References

Exercise:

1. Display either login or logout by checking with user data stored in local storage.

2. Try to add routing to todo-app which we built in this series. Or make a new app with any of your idea and add routing to it along with authentication.

References:

https://www.freecodecamp.org/news/how-to-optimize-react-applications-with-lazy-loading-232183e02768/

https://reactjs.org/docs/code-splitting.html

https://reactrouter.com/web/guides/quick-start

Sharing is caring

SUBSCRIBE

LIKE

SHARE

COMMENT

Venkatesh Mogili

#WebGuru

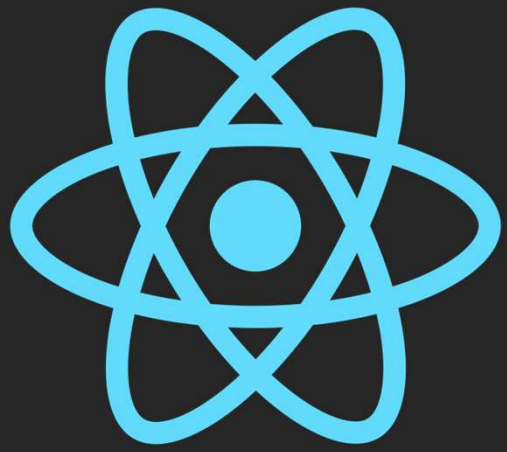
React JS

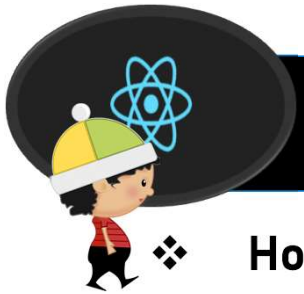
Crash Course

తెలుగు లో

PART – 6

Hooks





Contents

- ❖ **Hooks Introduction**
- ➢ Rules Of Hooks
- ➢ Basic Hooks:
- ➢ useState, useEffect, useContext, customHooks
- ➢ Additional Hooks:
- ➢ useReducer,
- ➢ useMemo,
- ➢ useCallback,
- ➢ useRef,
- ➢ useImperativeHandle,
- ➢ useLayoutEffect,
- ➢ useDebugValue
- ➢ Summary

Introduction

- Hooks are a new addition in **React 16.8 version.**

1) **Hooks don't replace your knowledge of React concepts.** Instead, Hooks provide a more direct API to the React concepts you already know: props, state, context, refs, and lifecycle.

2) **100% backwards-compatible.** Hooks don't contain any breaking changes.

3) **There are no plans to remove classes from React.**

- It does not work inside classes.

Rules of Hooks

- Hooks are JavaScript functions, but they impose two additional rules:

1. Only call Hooks **at the top level**. Don't call Hooks inside loops, conditions, or nested functions.

2. Only call Hooks **from React function components**. Don't call Hooks from regular JavaScript functions.

✓ one other valid place to call Hooks — your own custom Hooks.

Contents

- ✓ Hooks Introduction
- ✓ Rules Of Hooks
- ➢ **Basic Hooks:**
- ➢ **useState, useEffect, useContext, customHooks**
- ➢ Additional Hooks:
- ➢ useReducer,
- ➢ useMemo,
- ➢ useCallback,
- ➢ useRef,
- ➢ useImperativeHandle,
- ➢ useLayoutEffect,
- ➢ useDebugValue
- ➢ Summary

useState

❖ Let's understand Object and Array De-structuring before hooks.

1) **useState** returns a pair: **the current state value** and a **function** that lets you update it.

2) It's similar to this.setState in a class, except it doesn't merge the old and new state together.

3) We can use single and many useState hooks.

4) Hooks execute one after another in a sequence.

useEffect

- ❖ useEffect, adds the ability to perform side effects from a function component.

1) It serves the same purpose as componentDidMount, componentDidUpdate, and componentWillUnmount in React classes, but unified into a single API

2) By default, React runs the effects after every render — including the first render.

3) Just like with useState, you can use more than a single effect in a component.

Custom Hooks

- Custom Hooks are more of a convention than a feature. If a function's name starts with "use" and it calls other Hooks, we say it is a custom Hook.

- The state of each component is completely independent. Hooks are a way to reuse stateful logic, not state itself. In fact, each call to a Hook has a completely isolated state — so you can even use the same custom Hook twice in one component.

useContext

- useContext hook makes it easy to pass data throughout your app without manually passing props down the tree.

- Context can make a nice simple alternative to Redux when your data is simple or your app is small.

Contents

- ✓ Hooks Introduction
- ✓ Rules Of Hooks
- ✓ Basic Hooks:
- ✓ useState, useEffect, useContext, customHooks
- ➢ **Additional Hooks:**
- ➢ **useReducer,**
- ➢ **useMemo,**
- ➢ **useCallback,**
- ➢ **useRef,**
- ➢ **useImperativeHandle,**
- ➢ **useLayoutEffect,**
- ➢ **useDebugValue**
- ➢ Summary

useReducer

- ✓ convenient for dealing with more complex state changes in React components.
- ✓ useReducer borrows some theory from **Redux**, namely the concepts of reducers, action, and dispatch.

useMemo

❖ Returns a **memoized value**.

1) Whenever a huge computation is involved, then we can use useMemo hook.

2) Whenever there is no need of re-rendering If the respective state is not changed.

useCallback

❖ Returns a **memoized callback** instead of a single value like useMemo.

1) We use the result as a function, so that if it is required, we can pass parameters to it for doing operations.

2) Both useMemo and useCallback are equivalent but with the small difference.

useRef

- ✓ The useRef Hook is a function that returns a mutable ref object whose .current property is initialized with the passed argument (initialValue). The returned object will persist for the full lifetime of the component.

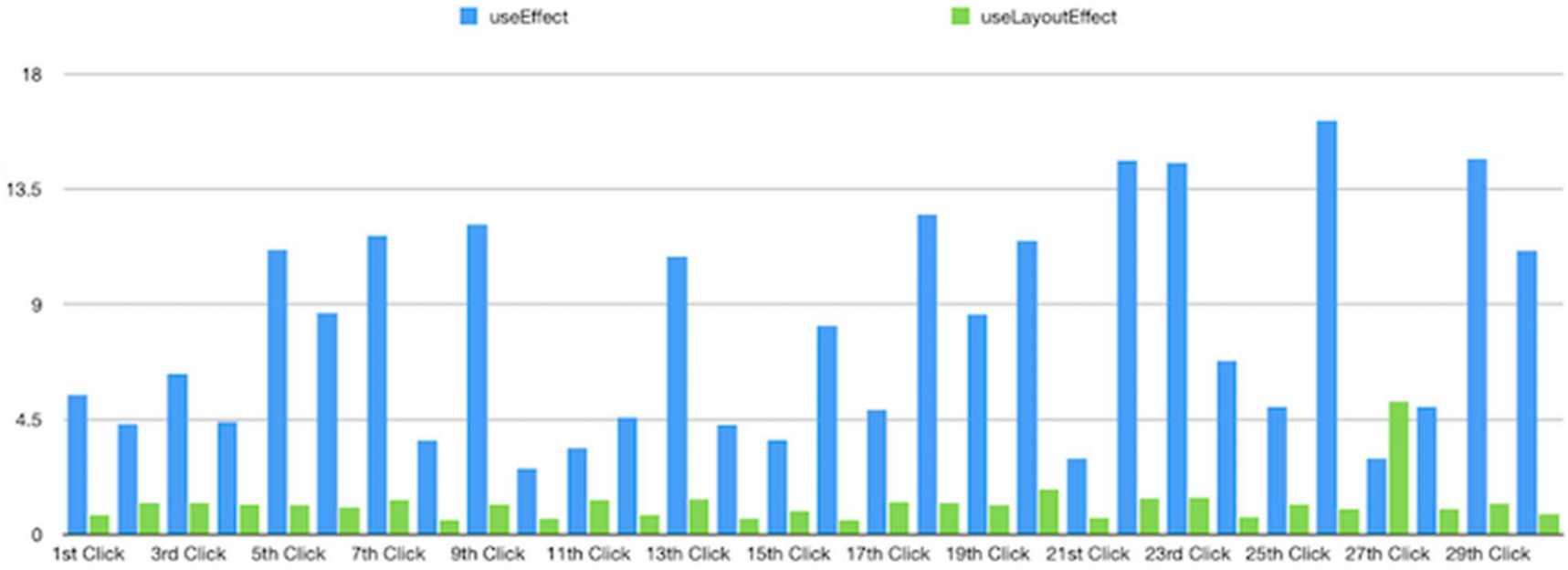
- ✓ A ref created with useRef will be created only when the component has been mounted and preserved for the full lifecycle.

useImperativeHandle

- ✓ **useImperativeHandle** customizes the instance value that is exposed to parent components when using ref.

useLayoutEffect

- ✓ Updates scheduled inside useLayoutEffect will be flushed synchronously, before the browser has a chance to paint.



✨ useDebugValue

- ✓ **useDebugValue** can be used to display a label for custom hooks in React DevTools.

Contents

- ✓ Hooks Introduction
- ✓ Rules Of Hooks
- ✓ Basic Hooks:
- ✓ useState, useEffect, useContext, customHooks
- ✓ Additional Hooks:
- ✓ useReducer,
- ✓ useMemo,
- ✓ useCallback,
- ✓ useRef,
- ✓ useImperativeHandle,
- ✓ useLayoutEffect,
- ✓ useDebugValue
- ➢ **Summary**

Summary

- ✓ Hooks are introduced in React 16.8 version.
- ✓ We have basic hooks such as useState,useEffect, useContext etc., and additional hooks and customHooks that we can create by our own.
- ✓ We can use hooks in functional components only.
- ✓ We can't use in class components.
- ✓ We should follow 2 rules to use hooks:
  1. Only call Hooks **at the top level**
  2. Only call Hooks **from React function components**
  3. We can call in **custom hooks as well**

Sharing is caring

SUBSCRIBE

LIKE

SHARE

COMMENT

Venkatesh Mogili

#WebGuru

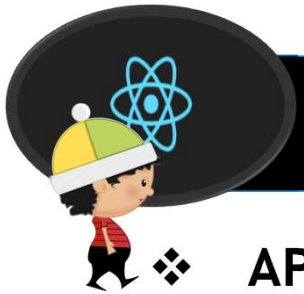
తెలుగు లో

PART – 7

API Calls, Import Export, JEST Unit Testing

React JS

Crash Course



Contents

- ❖ **API Calls Introduction**
  - ➢ Fetch
  - ➢ Axios
  - ➢ Different ways of Import and Export functionality in React.
  - ➢ JEST unit testing.

Introduction

- **API** – Application Programming Interface (simply takes request and gives response).
- **API Methods** – GET,POST,PUT,DELETE etc.,
- **Headers** – Authorization, Content-Type:application/json, multipart/form-data etc.,
- **API Body** – Data that we want to send through the request.
- **API Response** – Which we get as response either non-json format or json format.
- **API Response with catch** block if any network error occurs.

- What is API explained: https://www.youtube.com/watch?v=TA_17YOpcjg
- How to create APIs: https://www.youtube.com/watch?v=5zywriW3q8Y

Contents

- ✓ API Calls Introduction
- ➢ **Fetch**
- ➢ Axios
- ➢ Different ways of Import and Export functionality in React.
- ➢ JEST unit testing.

Fetch

- The Fetch API provides an **interface for fetching resources** (including across the network). It provides a more powerful and flexible feature set.
- Fetch provides a generic definition of **Request** and **Response** objects (and other things involved with network requests). This will allow them to be used wherever they are needed in the future.
- The fetch() method takes **one mandatory argument**, the path to the resource you want to fetch. It **returns a Promise** that resolves to the **Response** to that request, whether it is **successful or not.**
- **fetch()** won't support in node.js, **node-fetch** can be used as alternative to fetch.

❖ Reference: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

Contents

- ✓ API Calls Introduction
- ✓ Fetch
- ➢ **Axios**
- ➢ Different ways of Import and Export functionality in React.
- ➢ JEST unit testing.

Axios

- Promise based HTTP client for the **browser and node.js.**
- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Automatic transforms for JSON data
- Cancel requests

Reference: https://www.npmjs.com/package/axios

Contents

- ✓ API Calls Introduction
- ✓ Fetch
- ✓ Axios
- ➢ **Different ways of Import and Export functionality in React.**
- ➢ JEST unit testing.

Import and Export

- module.exports = { // Define your exports here. } and require('module')

1) Importing default export

2) Importing named values.

3) Importing a combination of Default Exports and Named Values.

4) Importing with Absolute paths.

Contents

- ✓ API Calls Introduction
- ✓ Fetch
- ✓ Axios
- ✓ Different ways of Import and Export functionality in React.
- ➢ **JEST unit testing.**

Unit Testing

- You can test React components similar to testing other JavaScript code.

There are a few ways to test React components. Broadly, they divide into two categories:

1) **Rendering component trees** in a simplified test environment and asserting on their output.
2) **Running a complete app** in a realistic browser environment (also known as "end-to-end" tests)

- Jest, React Testing Library are available for testing.
- **Jest** is a JavaScript test runner that lets you access the DOM via jsdom. While jsdom is only an approximation of how the browser works, it is often good enough for testing React components.

Jest Setup

1) npm install --save-dev jest babel-jest @babel/core @babel/preset-env @babel/preset-react

2) babel.config.js: (in root directory)

```
module.exports = { presets: [ "@babel/preset-env", "@babel/preset-react" ], };
```

3) npm install --save-dev enzyme @wojtekmaj/enzyme-adapter-react-17

4) setupTests.js: (in src directory)

```
import { configure } from "enzyme";
import Adapter from "@wojtekmaj/enzyme-adapter-react-17";
configure({ adapter: new Adapter() });
```

5) npm install --save-dev identity-obj-proxy

```
module.exports = { moduleNameMapper: { "\\.(css|less|scss)$": "identity-obj-proxy", }, };
```

Note: If you get errors ./App.css kind of imports, then you can install the identity-obj-proxy to solve it.

Contents

- ✓ API Calls Introduction
- ✓ Fetch
- ✓ Axios
- ✓ Different ways of Import and Export functionality in React.
- ✓ JEST unit testing.
- ➢ **Summary**

Summary

- We have seen different ways to call APIs in react using fetch and axios.
- We have understood how many different types of imports and exports available in react.
- Unit Testing using JEST package.

Sharing is caring

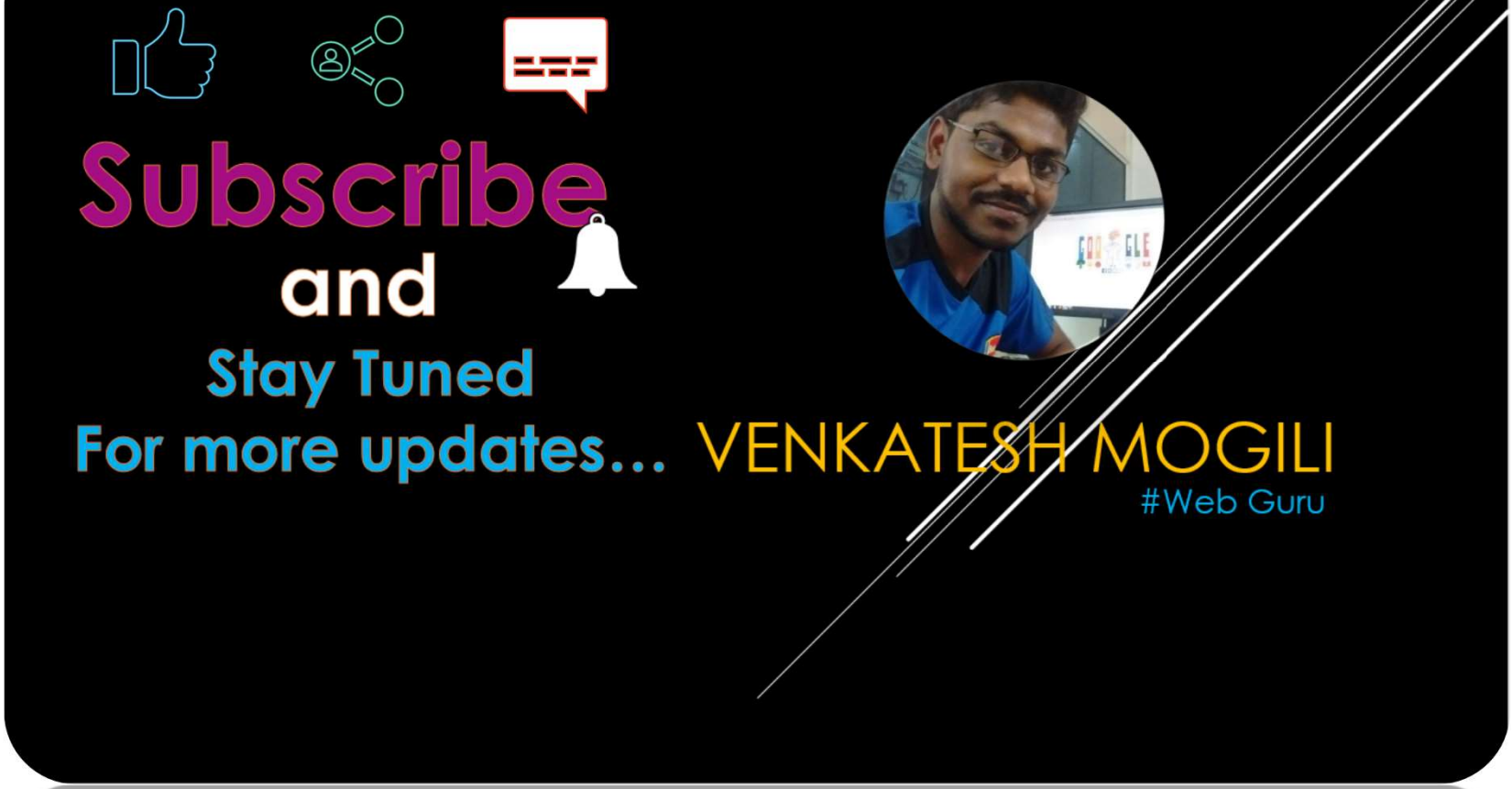
SUBSCRIBE

LIKE

SHARE

COMMENT

Welcome back



Venkatesh Mogili

#WebGuru

తెలుగు లో

PART – 8

Deploying React Apps

GitHub

Heroku

GoDaddy

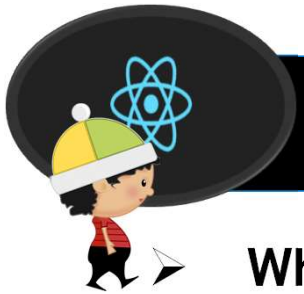
Netlify

Firebase

AWS

React JS

Crash Course



Contents

- **What is Deployment?**
- Deploying in Free Service Providers:

1) GitHub Pages
2) Netlify
3) Heroku
4) Firebase

- Automatic Deployment
- Deploying in Paid/Trail Service Providers:
- AWS/GoDaddy/Any other service with Apache Server (xampp)

Deployment

- ✓ Deployment is nothing but uploading our build files into production which is server (apache or nodejs or cloud servers).
- ✓ We usually call it as hosting. Hosting means simply using some memory in given hosting service provider.
- ✓ We usually call our local output as Development and global output as Production/Live.
- ✓ In real world, we can simply think hosting as one dedicated laptop/desktop which stores our files and will serve through out the internet by connecting to different domains/ip's.

Contents

- ✓ **What is Deployment?**
- ➢ **Deploying in Free Service Providers:**

1) GitHub Pages
2) Netlify

3) Heroku
4) Firebase

- ➢ Automatic Deployment
- ➢ Deploying in Paid/Trail Service Providers:
- ➢ AWS/GoDaddy/Any other service with Apache Server (xampp)

Build Before Deploying

Before deploying your application, you should build your entire application as a small bundle which can be supported by browsers and servers.

- Run below command to build your application.
  - npm run build
- Run below command to install serve package as global. Which can be used to serve production files in local system, but won't support for https as in local http only supports.
  - npm i serve –g
- Serve the production build files using below command.
  - serve –s build
  - Add "homepage":"." in package.json file for giving relative path to every link.

Deploying in Free Service Providers

There are many free service providers for hosting our files.
SignUp and SignIn are common for any online services

1. Github Pages

- git add .
  git commit –m “first commit”
  git push origin master
- Go to settings->Github Pages->Choose **master** branch and Save.
- Access your deployed site

2. Netlify

- Choose Sites tab and Drag and Drop your build folder.
- Access your deployed site

Contents

- ✓ What is Deployment?
- ✓ Deploying in Free Service Providers:
  1) GitHub Pages
  2) Netlify
  3) Heroku
  4) Firebase
- ➢ Automatic Deployment
- ➢ Deploying in Paid/Trail Service Providers:
- ➢ AWS/GoDaddy/Any other service with Apache Server (xampp)

GitHub

AWS

Deploying in Free Service Providers Continued..

3. Heroku

- Create nodejs app and Install Heroku cli software
- Click Deploy tab and follow the commands.
- git add .
  git commit –am “updated”
  git push heroku master
- Access your deployed site

4. Firebase

- Create firebase project and Install firebase-tools globally using npm command (npm i firebase-tools –g)
- firebase login, firebase init and firebase deploy.
- Access your deployed site

Contents

- ✓ What is Deployment?
- ✓ Deploying in Free Service Providers:
- 1) GitHub Pages
- 2) Netlify
- 3) Heroku
- 4) Firebase
- ➢ **Automatic Deployment**
- ➢ Deploying in Paid/Trail Service Providers:
- ➢ AWS/GoDaddy/Any other service with Apache Server (xampp)

Automatic Deployment with Git

Any service which should support continuous deployment, should be connected to Git.

- ✓ GitHub Pages
  By default if you push any changes to the build folder, it will be deployed automatically.
  Access your site 👍
- ✓ Netlify (New site from Git)
  Choose the repository to deploy and Click deploy.
  Access your site 👍
- ✓ Heroku (It will work if you upgrade) ☹
  Under Deploy tab->Deployment method should be GitHub
  Choose the repository and connect and Click Enable Automatic Deploys.

Automatic Deployment with Git Continued...

✓ Firebase
firebase init, choose hosting, use existing or new project, select app

1. What do you want to use as your public directory? build
2. Configure as a single-page app (rewrite all urls to /index.html)? Yes
3. **Set up automatic builds and deploys with GitHub?** Yes
4. File build/index.html already exists. Overwrite? No
5. For which GitHub repository would you like to set up a GitHub workflow? (format:user/repository) VenkateshMogili/routing-app
6. Set up the workflow to run a build script before every deploy? Yes
7. What script should be run before every deploy? npm run build
8. Set up automatic deployment to your site's live channel when a PR is merged? Yes
9. What is the name of the GitHub branch associated with your site's live channel? master

✓ Access your site

Contents

- ✓ What is Deployment?
- ✓ Deploying in Free Service Providers:
- 1) GitHub Pages
- 2) Netlify
- 3) Heroku
- 4) Firebase
- ✓ Automatic Deployment
- ➢ **Deploying in Paid/Trail Service Providers:**
- ➢ AWS/GoDaddy/Any other service with Apache Server (xampp)

GitHub

AWS

Deploying in Paid/Trail Service Providers

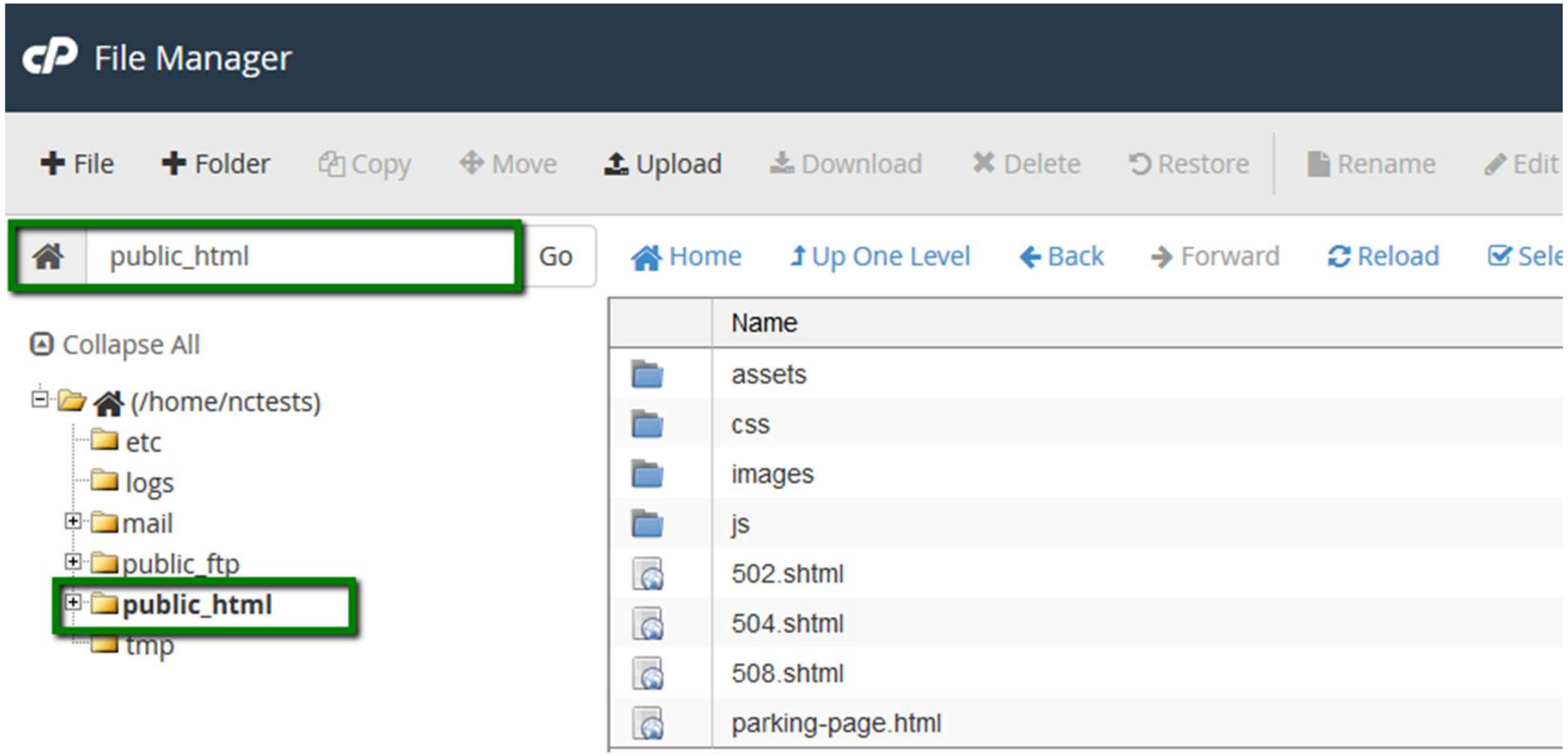
- ✓ AWS (SignUp and SignIn to AWS Console)
- ✓ Create EC2 Instance as free-trail by clicking Launch Instances->
- ✓ select **Windows**/Ubuntu 64-bit->
- ✓ choose free tier->add storage(30gb is for free)->review and launch->
- ✓ create a key pair and download pem file and create instance.
- ✓ Click connect->RDP Client->Download RDP file->Get password->browse downloaded .pem file->decrypt password. Then you will get password to connect with Remote Desktop.
- ✓ Install xampp software->Open xampp control panel->Start apache server.
- ✓ Upload build files in htdocs folder.
- ✓ Allow firewall for domain, private and public profiles in windows firewall settings.
- ✓ Access your deployed site 👍

AWS

Deploying in Paid/Trail Service Providers

- ✓ GoDaddy (SignUp and SignIn)
- ✓ Click on Manage for the particular domain
- ✓ Open File Manager->Open public_html folder and copy the files from the build folder instead of build folder as public_html folder acts as build.
- ✓ Access your deployed site

Deploying in Paid/Trail Service Providers



Contents

- ✓ What is Deployment?
- ✓ Deploying in Free Service Providers:
- ✓ GitHub Pages
- ✓ Netlify
- ✓ Heroku
- ✓ Firebase
- ✓ Automatic Deployment
- ✓ Deploying in Paid/Trail Service Providers:
- ✓ AWS/GoDaddy/Any other service with Apache Server (xampp)

Summary

- ✓ Deployment means simply hosting our build files in cloud/internet which can be accessed through out the world.
- ✓ We have seen how to deploy in Github,Netlify,Heroku,Firebase, AWS and GoDaddy service providers.
- ✓ We have seen automatic deployment by connecting with Git repository.

Sharing is caring

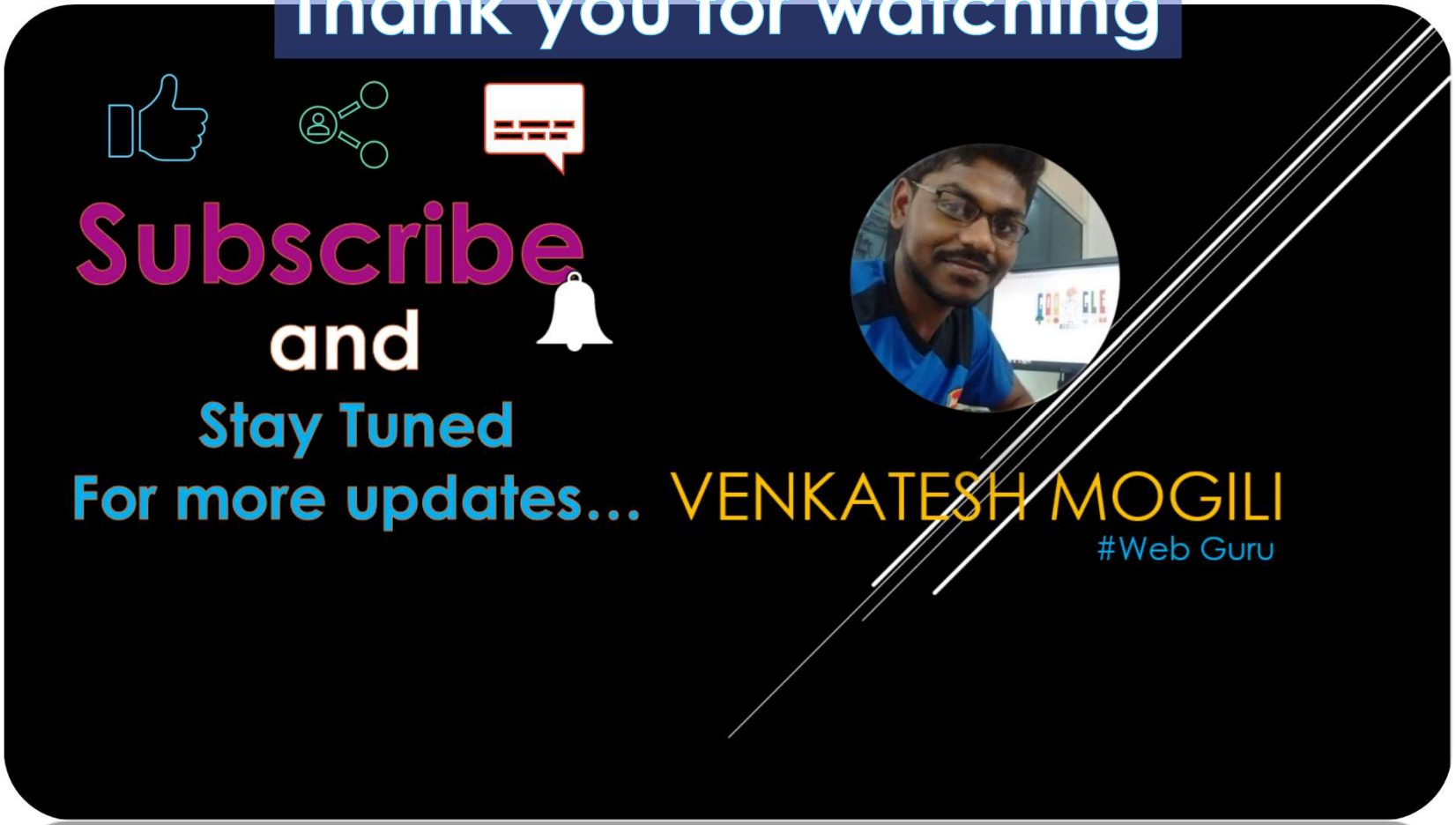
SUBSCRIBE

LIKE

SHARE

COMMENT

Thank you for watching



తెలుగు లో

PART – 1

Next Course

Venkatesh Mogili

#WebGuru

node JS

Crash Course