Episode-08 | Let’s Get Classy

Please make sure to follow along with the whole ”**Namaste React**" series, starting from Episode-1 and continuing through each subsequent episode. The notes are designed to provide detailed explanations of each concept along with examples to ensure thorough understanding. Each episode builds upon the knowledge gained from the previous ones, so starting from the beginning will give you a comprehensive understanding of React dvelopment.

I've got a quick tip for you. To get the most out of these notes, it's a good idea to watch **Episode-08** first. Understanding what ”**Akshay**” shares in the video will make these notes way easier to understand.

Q) How do you `create Nested Routes react-routerdom configuration` ?

> In React applications using react-router-dom, we can create nested routes by nesting our `<Route>` components inside each other within the route configuration. This allows you to define routes and components hierarchically, making it easier to manage the routing structure of your application.

Here's a step-by-step guide on how to create nested routes using react-router-dom:

1 `Install react-router-dom if we haven't already`:

```
npm install react-router-dom
```

2 `Import the necessary components from react-router-dom in your application file`:

```
import { BrowserRouter as Router, Route, Switch } from 'react
-router-dom';
```

Defining our route hierarchy by nesting components within each other. Typically, this is done within a component that acts as a layout or container for the nested routes. For example, if we have a layout component called Layout:

```
import React from 'react';
import { Route } from 'react-router-dom';

// Import your nested route components
import Home from './Home';
import About from './About';

function Layout() {
  return (
    <div>
      <h1>My App</h1>
```

```
      <Route path="/home" component={Home} />
      <Route path="/about" component={About} />
    </div>
  );
}

export default Layout;
```

⁉ In our main application file, wrap our entire application with the Router component, and use the component to render only the first matching route:

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react
-router-dom';

// Import your Layout component that defines nested routes
import Layout from './Layout';

function App() {
  return (
    <Router>
      <Switch>
        <Route path="/" component={Layout} />
      </Switch>
    </Router>
  );
}

export default App;
```

Now we have a simple example of nested routes. In this case, the Layout component defines the `/home` and `/about` routes, and these nested routes can have their own components and nested routes as well. We can continue to nest routes further by adding more `<Route>` components inside the Home and About components to create a more complex routing structure. Remember that this is

just a basic example, and we can customize our routing structure based on the requirements of our application. We can also use the exact prop on routes to ensure that only the exact path is matched if needed.

Q) Read about `createHashRouter`, `createMemoryRouter` from `React Router docs.`

1. `createHashRouter` - createHashRouter is part of the React Router library and provides routing capabilities for single-page applications (SPAs). It's commonly used for building client-side navigation within applications. Unlike traditional server-side routing, it uses the fragment identifier (hash) in the URL to manage and handle routes on the client side. This means that changes in the URL after the # symbol do not trigger a full page reload, making it suitable for SPAs.

To use createHashRouter, we typically import it from the React Router library and define our routes using Route components. Here's a basic example of how you might use it:

```
import { createHashRouter, Route } from 'react-router-dom';

const App = () => (
  <createHashRouter>
    <Route path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/contact" component={Contact} />
  </createHashRouter>
);
```

`` `` `createMemoryRouter` - createMemoryRouter is another routing component provided by React Router. Unlike createHashRouter or BrowserRouter, createMemoryRouter is not associated with the browser's URL. Instead, it allows you to create an in-memory router for testing or other scenarios where you don't want to interact with the actual browser's URL.

Here's a simple example of how to use createMemoryRouter:

```
import { createMemoryRouter, Route } from 'react-router-dom';

const App = () => (
  <createMemoryRouter>
    <Route path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/contact" component={Contact} />
  </createMemoryRouter>
);
```

In both cases, we define our application's routes within the router component and specify the components to render for each route. The choice between `createHashRouter` and `createMemoryRouter` depends on our specific use case, such as whether we're building an SPA that interacts with the browser's URL or a scenario where we need an in-memory router for testing.

Q) What is the `order of life cycle method calls in Class Based Components` ?

`Constructor` The constructor method is the first to be called when a component is created. It's where we typically initialize the component's state and bind event handlers.

`Render` The render method is responsible for rendering the component's UI. It must return a React element (typically JSX representing the component's structure.

`ComponentDidMount` This method is called immediately after the component is inserted into the DOM. It's often used for making AJAX requests, setting up subscriptions, or other one-time initializations.

`ComponentDidUpdate` This method is called after the component has been updated (re-rendered) due to changes in state or props. It's often used for side effects, like updating the DOM in response to state or prop changes.

`ComponentWillUnmount` This method is called just before the component is removed from the DOM. It's used to clean up resources or perform any necessary cleanup. For more reference React-Lifecycle-methods

Q) Why do we use `componentDidMount` ?

The `componentDidMount` lifecycle method in React class-based components is used for a specific purpose: it is called immediately after a component is inserted into the DOM (Document Object Model). This makes it a crucial point in the component's lifecycle and provides a valuable opportunity to perform various tasks that require interaction with the DOM or external data sources. Here are some common use cases for componentDidMount:

`Fetching Data` It's often used to make asynchronous requests to fetch data from APIs or external sources. This is a common scenario for components that need to display dynamic content.

`DOM Manipulation` When we need to interact with the DOM directly, such as selecting elements, setting attributes, or applying third-party libraries that require DOM elements to be present, we can safely do so in componentDidMount. This is because the component is guaranteed to be in the DOM at this point.

```
class MyComponent extends React.Component {
  componentDidMount() {
    // Fetch data from an API
    fetch('https://api.example.com/data')
      .then(response => response.json())
      .then(data => {
        // Update the component's state with the fetched data
        this.setState({ data });
      })
      .catch(error => {
        // Handle any errors
        console.error(error);
      });
```

```
  }

  render() {
    // Render component based on state
    return (
      <div>{/* Display data from this.state.data */}</div>
    );
  }
}
```

By using `componentDidMount`, we can ensure that the data fetching or other side effects happen after the initial render and that our component interacts with the DOM or external data sources at the right time in the component's lifecycle.

Q) Why do we use `componentWillUnmount` ? Show with example.

> The `componentWillUnmount` lifecycle method in React class-based components is used to perform cleanup and teardown tasks just before a component is removed from the DOM. It's a crucial part of managing resources and subscriptions to prevent memory leaks and ensure that the component's behavior is properly cleaned up. Here's why and when we should use componentWillUnmount:

1 `Cleanup Resources` ⍰ If your component has allocated any resources, such as event listeners, subscriptions, timers, or manual DOM manipulations, it's essential to release these resources to prevent memory leaks. componentWillUnmount is the appropriate place to do this.

2 `Cancel Pending Requests` ⍰ If your component has initiated any asynchronous requests, such as AJAX calls or timers, you should cancel or clean them up to avoid unexpected behavior after the component is unmounted.

Here's an example of using componentWillUnmount to remove an event listener when a component is unmounted:

```
class MyComponent extends React.Component {
  constructor() {
    super();
    this.handleResize = this.handleResize.bind(this);
  }

  componentDidMount() {
    // Add a window resize event listener when the component
is mounted
    window.addEventListener('resize', this.handleResize);
  }

  componentWillUnmount() {
    // Remove the window resize event listener when the compo
nent is unmounted
    window.removeEventListener('resize', this.handleResize);
  }

  handleResize(event) {
    // Handle the resize event
    console.log('Window resized:', event);
  }

  render() {
    return <div>My Component</div>;
  }
}
```

In this example, the component adds a resize event listener to the window when it's mounted, and it removes that listener in the componentWillUnmount method.

This ensures that the event listener is properly cleaned up when the component is unmounted, preventing memory leaks or unexpected behavior.

By using componentWillUnmount, we can ensure that any cleanup tasks are executed reliably when the component is no longer needed, helping to maintain the integrity of our application and avoiding potential issues.

Q) (Research) Why do we use `super(props)` in constructor?

> In JavaScript, when you define a class that extends another class (inherits from a parent class), we often use the super() method with props as an argument in the constructor of the child class. This is commonly seen in React when you create class-based components. The super(props) call is used for the following reasons:

`Access to Parent Class's Constructor` ⍰ When a child class extends a parent class, the child class can have its constructor. However, if the child class has a constructor, it must call super(props) as the first statement in its constructor. This is because super(props) is used to invoke the constructor of the parent class, ensuring that the parent class's initialization is performed before the child class's constructor code is executed. It is essential to maintain the inheritance chain correctly.

`Passing Props to the Parent Constructor` ⍰ By passing props to super(props), we ensure that the props object is correctly passed to the parent class's constructor. This is important because the parent class may need to set up its properties or handle the props somehow. By calling super(props), we make the props available for the parent class's constructor to work with.

> Here's an example of how super(props) is used in a React component:

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props); // Call the constructor of the parent class
(React.Component)
    // Initialize your component's state or perform other set
up
  }

  render() {
    // Render the component based on its state and props
    return <div>{this.props.someProp}</div>;
  }
}
```

In this example, the super(props) call ensures that the `React.Component` class's constructor is called, which is necessary for React to set up the component correctly. This is especially important because React uses the props object to pass data from parent components to child components. By calling super(props), we make sure that the props are properly handled in the parent class's constructor, and we can access them in our child component.

In modern JavaScript and React, it's also common to define a constructor without explicitly calling super(props), and it will be automatically called for us. However, if we define a constructor in a child class, and the parent class has its constructor, it's a good practice to include super(props) to ensure that the parent class's constructor is invoked correctly.

Q) (Research) Why can't we have the `callback function` of `useEffect` `async` ?

A⍰ In React, the `useEffect` hook is designed to handle side effects in functional components. It's a powerful and flexible tool for managing asynchronous operations, such as data fetching, API calls, and more. However, useEffect itself cannot directly accept an async callback function. This is because useEffect expects its callback function to return either nothing (i.e., undefined) or a cleanup function, and it doesn't work well with Promises returned from async functions. There are a few reasons for this:

`Return Value Expectation` The primary purpose of the useEffect callback function is to handle side effects and perform cleanup. React expects us to either return nothing (i.e., undefined) from the callback or return a cleanup function. An async function returns a Promise, and it doesn't fit well with this expected behavior.

`Execution Order and Timing` With async functions, we might not have fine-grained control over the execution order of the asynchronous code and the cleanup code. React relies on the returned cleanup function to handle cleanup when the component is unmounted or when the dependencies specified in the useEffect dependency array change. If you return a Promise, React doesn't know when or how to handle cleanup.

To work with async operations within a useEffect, we can use the following pattern:

```
useEffect(() => {
  const fetchData = async () => {
    try {
      // Perform asynchronous operations
      const result = await someAsyncOperation();
      // Update the state with the result
      setState(result);
    } catch (error) {
      // Handle errors
      console.error(error);
    }
  };

  fetchData(); // Call the async function

  return () => {
    // Cleanup code, if necessary
    // This function will be called when the component unmoun
ts or when dependencies change

  };
}, [/* dependency array */]);
```

In this pattern, we define an async function within the useEffect callback, perform our asynchronous operations, and then call that function. Additionally, we return a cleanup function from the useEffect to handle any necessary cleanup tasks when the component unmounts or when specified dependencies change.

By using this approach, we can effectively manage asynchronous operations with useEffect while adhering to React's expectations for the callback function's return value.

Self-Notes

Class based components

Class based components is the older way to create component in react if we are building the project today never need to use class-based component.

In the about us section will create the class-based components where fetch the data from GitHub Apis and showcases team member information.

First will make the component functional based component and then will convert in into the **functional based component –**

Basic syntax-

```
const User = () => {
   return (

      <div className="user-card">
         <h2>Name: Pragati</h2>
         <h3>Location: Niwari</h3>
         <h4>Contact: Pragatikhard@gmail.com</h4>
      </div>
   )
}

export default User;
```

Class based component-

Basic syntax –

```
import React from 'react';
class UserClass extends React.Component {
   render() {
      return (
         <div className="user-card">
            <h2>Name: Pragati</h2>
            <h3>Location: Niwari</h3>
            <h4>Contact: Pragatikhard@gmail.com</h4>
         </div>
      )
   }
}

export default UserClass;
```

extends React.Component will make react know this is the class based component so the react tracking it.

render () will return a piece of JSX which will be display to the UI.

Compare between functional and class-based components –

Functional component – It's a function that return some piece of JSX.

Class component – It's a class which extends React.Component and it has a render method which return some piece of JSX and the JSX convert into html and render on to the web page.

What is the React.Component?

React.Component is the class which is given us to the react and the component class (UserClass) inherit the property from it and it's come from import React from 'react';

Pass the props in functional based component -

So do it like this and receive it like this –

```
About.js M
src > components > About.js > About
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
    return (
        <div>
            <h1>About</h1>
            <h2>This is the Namaste React
            Web Series</h2>
            <User name={"Pragati Khard
            (Function)"}/>

            <UserClass name={"Pragati Khard
            (Class)"}/>
        </div>
    )
}

export default About;
```

```
User.js U
src > components > User.js > User
const User = (props) => {
    return (

        <div className="user-card">
            <h2>Name: {props.name}</h2>
            <h3>Location: Niwari</h3>
            <h4>Contact: Pragatikhard@gmail.
            com</h4>
        </div>
    )
}

export default User;
```

OR Destructuring on the fly-

```
About.js M
src > components > About.js > About
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
    return (
        <div>
            <h1>About</h1>
            <h2>This is the Namaste React
            Web Series</h2>
            <User name={"Pragati Khard
            (Function)"}/>

            <UserClass name={"Pragati Khard
            (Class)"}/>
        </div>
    )
}

export default About;
```

```
User.js U
src > components > User.js > User
const User = ({name}) => {
    return (

        <div className="user-card">
            <h2>Name: {name}</h2>
            <h3>Location: Niwari</h3>
            <h4>Contact: Pragatikhard@gmail.
            com</h4>
        </div>
    )
}

export default User;
```

<u>Pass the props in class based component -</u>

Constructors receive the props and we will have to write super(props). If we will not write the super(props) we will get the error.

We always use this keyword inside your class so we can accessing the props inside your class.

```
About.js M
src > components > About.js > About
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
    return (
        <div>
            <h1>About</h1>
            <h2>This is the Namaste React
            Web Series</h2>
            <User name={"Pragati Khard
            (Function)"}/>

            <UserClass name={"Pragati
            Khard (Class)"}/>
        </div>
    )
}

export default About;
```

```
UserClass.js U
src > components > UserClass.js > ...
import React from 'react';

class UserClass extends React.Component {
    constructor(props) {
        super(props)

        console.log(props)
    }
    render() {
        return (
            <div className="user-card">
                {/* <h2>Name: Pragati</h2> */}
                <h2>Name: {this.props.name}</
                h2>
                <h3>Location: Niwari</h3>
                <h4>Contact:
                Pragatikhard@gmail.com</h4>
            </div>
        )
    }
}

export default UserClass;
```

In class based component you will received the props inside the constructor and will use this.props.name, whenever class is initialize/ created constructor is called

Another example-

```
About.js M
src > components > About.js > default
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
    return (
        <div>
            <h1>About</h1>
            <h2>This is the Namaste React
            Web Series</h2>
            <User name={"Pragati Khard
            (Function)"}/>

            <UserClass name={"Pragati Khard
            (Class)"} location={"M.P"}/>
        </div>
    )
}

export default About;
```

```
UserClass.js U
src > components > UserClass.js > UserClass > render
import React from 'react';

class UserClass extends React.Component {
    constructor(props) {
        super(props)

        console.log(props)
    }
    render() {
        return (
            <div className="user-card">
                {/* <h2>Name: Pragati</h2>
                */}
                <h2>Name: {this.props.name}
                </h2>
                <h3>Location: {this.props.
                location}</h3>
                <h4>Contact:
                Pragatikhard@gmail.com</h4>
            </div>
        )
    }
}

export default UserClass;
```

We can destruct it –

```
JS About.js M
src > components > JS About.js > default
import User from "./User";
import UserClass from "./UserClass";

const About = () => {
    return (
        <div>
            <h1>About</h1>
            <h2>This is the Namaste React
            Web Series</h2>
            <User name={"Pragati Khard
            (Function)"}/>

            <UserClass name={"Pragati Khard
            (Class)"} location={"M.P"}/>
        </div>
    )
}

export default About;
```

```
JS UserClass.js U
src > components > JS UserClass.js > UserClass > render
import React from 'react';

class UserClass extends React.Component {
    constructor(props) {
        super(props)

        console.log(props)
    }
    render() {
        const {name,location } = this.props;
        return (
            <div className="user-card">
                {/* <h2>Name: Pragati</h2>
                */}
                <h2>Name: {name}</h2>
                <h3>Location: {location}</
                h3>
                <h4>Contact:
                Pragatikhard@gmail.com</h4>
            </div>
        )
    }
}

export default UserClass;
```

<u>Create the state variable in functional based component-</u>

In functional based component for creating the state variable we are using hook known as useState.

```
JS User.js U
src > components > JS User.js > default
import { useState } from "react";

const User = ({ name }) => {
    const [count] = useState(0);
    return (

        <div className="user-card">
            <h1>Count - {count}</h1>
            <h2>Name: {name}</h2>
            <h3>Location: Niwari</h3>
            <h4>Contact: Pragatikhard@gmail.com</h4>
        </div>
    )
}

export default User;
```

<u>Create the state variable in class based component-</u>

There is no hooks inside the class based component

State is created whenever the class instants is created means when we are says loading the class-based component on a web page that means I am creating the instants of a class.

And whenever we are creating the instant of the class , the constructor is called and that's is the best place to receive a props and this is the best place to create the state variables.

In class based component we are use this.state and this state is the big whole object which contains state variables.

```
JS UserClass.js U ×
src > components > JS UserClass.js > UserClass > constructor
1   import React from 'react';
2
3   class UserClass extends React.Component {
4       constructor(props) {
5           super(props)
6
7           this.state = {
8               count: 0,
9           }
10      }
11      render() {
12          const {name,location } = this.props;
13          return (
14              <div className="user-card">
15                  <h1>Count: {this.state.count}</h1>
16                  <h2>Name: {name}</h2>
17                  <h3>Location: {location}</h3>
18                  <h4>Contact: Pragatikhard@gmail.
                    com</h4>
19              </div>
20          )
21      }
22  }
23
24  export default UserClass;
```

We can Destructuring it like this –

```
import React from 'react';

class UserClass extends React.Component {
    constructor(props) {
        super(props)

        this.state = {
            count: 0,
        }
    }
    render() {
        const { name, location } = this.props;
        const { count } = this.state;
        return (
            <div className="user-card">
                <h1>Count: {count}</h1>
                <h2>Name: {name}</h2>
                <h3>Location: {location}</h3>
                <h4>Contact: Pragatikhard@gmail.com</h4>
            </div>
        )
    }
}
```

<u>Create two state variables inside the functional and class component</u>

Functional component -

```
JS User.js  U ×
src > components > JS User.js > User
import { useState } from "react";

const User = ({ name }) => {
    const [count] = useState(0);
    const [count2] = useState(1);
    return (

        <div className="user-card">
            <h1>Count - {count}</h1>
            <h1>Count - {count2}</h1>
            <h2>Name: {name}</h2>
            <h3>Location: Niwari</h3>
            <h4>Contact: Pragatikhard@gmail.com</h4>
        </div>
    )
}

export default User;
```

Can’t do in this way-

```
JS User.js U    JS UserClass.js U ×
src > components > JS UserClass.js > UserClass > constructor
1   import React from 'react';
2
3   class UserClass extends React.Component {
4       constructor(props) {
5           super(props)
6
7           this.state = {
8               count: 0,
9           }
10          this.state = {
11              count1: 1,
12          }
13      }
14      render() {
15          const { name, location } = this.props;
16          const { count, count1 } = this.state;
17          return (
18              <div className="user-card">
19                  <h1>Count: {count}</h1>
20                  <h1>Count: {count1}</h1>
21                  <h2>Name: {name}</h2>
22                  <h3>Location: {location}</h3>
23                  <h4>Contact: Pragatikhard@gmail.com</h4>
24              </div>
25          )
```

Class component –

```
JS UserClass.js U ×
src > components > JS UserClass.js > UserClass > constructor
1   import React from 'react';
2
3   class UserClass extends React.Component {
4       constructor(props) {
5           super(props)
6
7           this.state = {
8               count: 0,
9               count1: 1,
10          }
11      }
12      render() {
13          const { name, location } = this.props;
14          const { count, count1 } = this.state;
15          return (
16              <div className="user-card">
17                  <h1>Count: {count}</h1>
18                  <h1>Count: {count1}</h1>
19                  <h2>Name: {name}</h2>
20                  <h3>Location: {location}</h3>
21                  <h4>Contact: Pragatikhard@gmail.com</h4>
22              </div>
23          )
24      }
25  }
```

In class component state is the big object which will contains all the state variables. So we will create inside the this.state= {}

<u>Update the variable in functional and class based components –</u>

Functional component-

```
JS User.js  U ×
src > components > JS User.js > ...
import { useState } from "react";

const User = ({ name }) => {
    const [count, setCount] = useState(0);
    const [count2] = useState(1);
    return (

        <div className="user-card">
            <h1>Count - {count}</h1>
            <h1>Count - {count2}</h1>
            <h2>Name: {name}</h2>
            <h3>Location: Niwari</h3>
            <h4>Contact: Pragatikhard@gmail.com</h4>
        </div>
    )
}

export default User;
```

Class component-

Can't do in this way-

It will not work cause never update state variables directly.

```
JS User.js U    JS UserClass.js U ×
src > components > JS UserClass.js > UserClass > render

class UserClass extends React.Component {
    constructor(props) {
        super(props)

        this.state = {
            count: 0,
            count1: 1,
        }
    }
    render() {
        const { name, location } = this.props;
        const { count, count1 } = this.state;
        return (
            <div className="user-card">
                <h1>Count: {count}</h1>
                <button onClick={()=>{
                    this.state.count =   this.state.count+1
                }}>Count Increase</button>
                <h1>Count: {count1}</h1>
                <h2>Name: {name}</h2>
                <h3>Location: {location}</h3>
                <h4>Contact: Pragatikhard@gmail.com</h4>
            </div>
        )
    }
}
```

How do I update then

React gaves you access to an function called as this.setState , and we can use it anywhere inside the class. Inside the setState we will pass an object and this object contains the updated variable of state variables

```
User.js U    JS UserClass.js U ×
rc > components > JS UserClass.js > UserClass > render

class UserClass extends React.Component {
    constructor(props) {
        super(props)

        this.state = {
            count: 0,
            count1: 1,
        }
    }
    render() {
        const { name, location } = this.props;
        const { count, count1 } = this.state;
        return (
            <div className="user-card">
                <h1>Count: {count}</h1>
                <button onClick={() => {
                    // NEVER UPDATE STATE VARIABLES DIRECTLY
                    //    this.state.count =   this.state.count+1
                  this.setState({
                    count: this.state.count+1,
                  })

                }}>Count Increase</button>
                <h1>Count: {count1}</h1>
                <h2>Name: {name}</h2>
                <h3>Location: {location}</h3>
                <h4>Contact: Pragatikhard@gmail.com</h4>
            </div>
        )
```

React is basically updating , react is re-rendering the component and it just changing the portion of html

<u>How we will update count two variable in the same class component -</u>

Can't do in this way-

```
3   class UserClass extends React.Component {
4       constructor(props) {
9           count1: 1,
10      }
11  }
12  render() {
13      const { name, location } = this.props;
14      const { count, count1 } = this.state;
15      return (
16          <div className="user-card">
17              <h1>Count: {count}</h1>
18              <button onClick={() => {
19                  // NEVER UPDATE STATE VARIABLES DIRECTLY
20                  //    this.state.count =   this.state.count+1
21                this.setState({
22                  count: this.state.count+1,
23                })
24                this.setState({
25                  count2: this.state.count2+1,
26                })
27
28              }}>Count Increase</button>
29              <h1>Count: {count1}</h1>
30              <h2>Name: {name}</h2>
31              <h3>Location: {location}</h3>
32              <h4>Contact: Pragatikhard@gmail.com</h4>
```

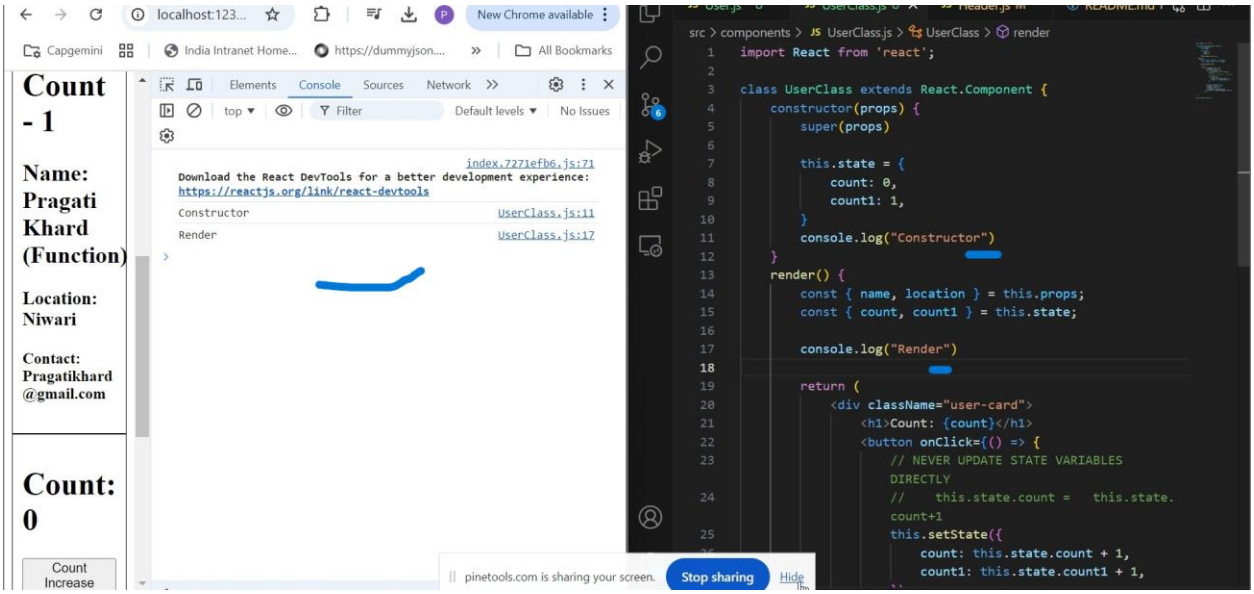
```
class UserClass extends React.Component {
    constructor(props) {
        count: 0,
        count1: 1,
    }
}
render() {
    const { name, location } = this.props;
    const { count, count1 } = this.state;
    return (
        <div className="user-card">
            <h1>Count: {count}</h1>
            <button onClick={() => {
                // NEVER UPDATE STATE VARIABLES DIRECTLY
                //    this.state.count =   this.state.count+1
                this.setState({
                    count: this.state.count + 1,
                    count1: this.state.count1 + 1,
                })


            }}>Count Increase</button>
            <h1>Count: {count1}</h1>
            <h2>Name: {name}</h2>
            <h3>Location: {location}</h3>
            <h4>Contact: Pragatikhard@gmail.com</h4>
        </div>
    )
}
```

LifeCycle methods –

Here we will learn how the class-based component mounting on the web page/ how it is put on the web page.
Loading , mounting is one of the same thing .

Suppose about us is the parent component and userClass inside of the parent component . so when you load the about us component on the webpage it goes line by line and when it see it's the class based component (userClass) so it start the class based component so that time the new instant is created so that time constructor will call. Once the constructor is called then render is called.

Constructor - when the class load constructor is load and once the constructor is called then the render is called. First constructor then renders



What happen when the parent is class based component – So now here we are convert the about us page which is the functional based converted into the class based component.

```
// import React from "react";
// OR for import the React.Component
import { Component } from "react";
import User from "./User";
import UserClass from "./UserClass";

// Class based component
class About extends Component {
    constructor(props) {
        super(props)
        console.log("Parent Constructor")
    }
    render() {
        console.log("Parent Render")
        return (
            <div>
                <h1>About</h1>
                <h2>This is the Namaste React Web Series</h2>
                <User name={"Pragati Khard (Function)"} />

                <UserClass name={"Pragati Khard (Class)"}
                location={"M.P"} />
            </div>
        )
    }
}

// Functional based component
// const About = () => {
```

```
import React from 'react';

class UserClass extends React.Component {
    constructor(props) {
        super(props)

        this.state = {
            count: 0,
            count1: 1,
        }
        console.log("Child Constructor")
    }
    render() {
        const { name, location } = this.props;
        const { count, count1 } = this.state;

        console.log("Child Render")

        return (
            <div className="user-card">
                <h1>Count: {count}</h1>
                <button onClick={() => {
                    // NEVER UPDATE STATE VARIABLES DIRECTLY
                    //    this.state.count =   this.state.
                    count+1
                    this.setState({
                        count: this.state.count + 1,
                        count1: this.state.count1 + 1,
                    })
```

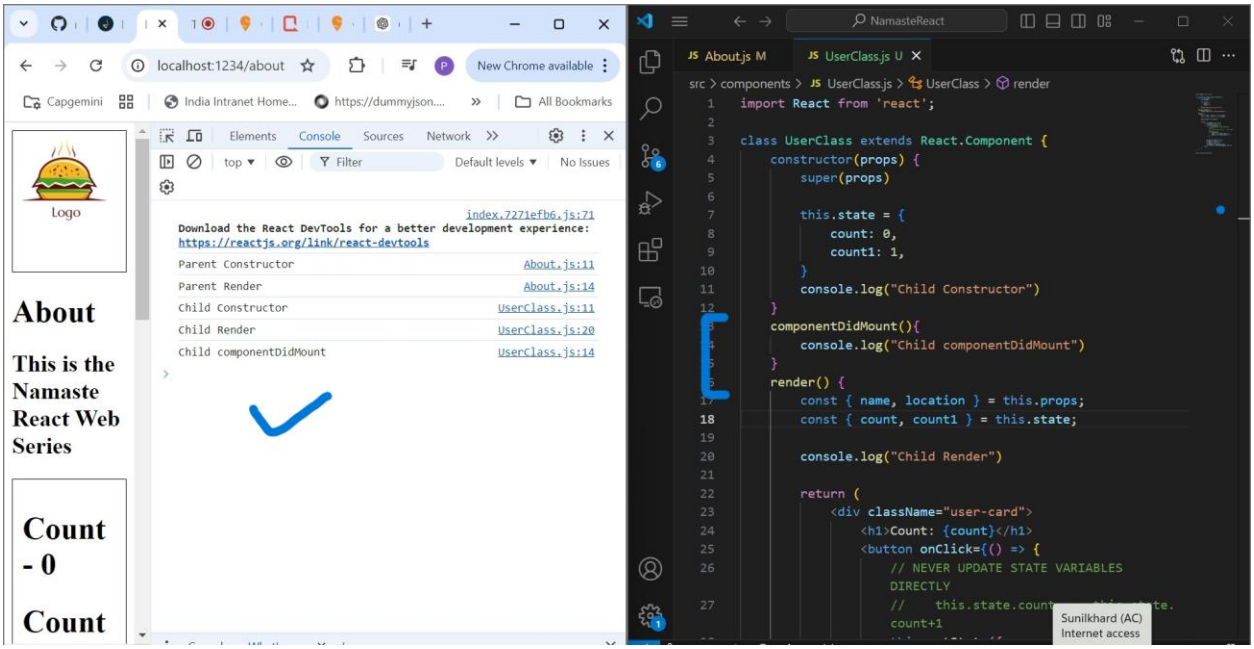
```
index.7271efb6.js:71
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
Parent Constructor                                  About.js:11
Parent Render                                       About.js:14
Child Constructor                                   UserClass.js:11
Child Render                                        UserClass.js:17
```

How it works, how it’s loaded and mount on the DOM-

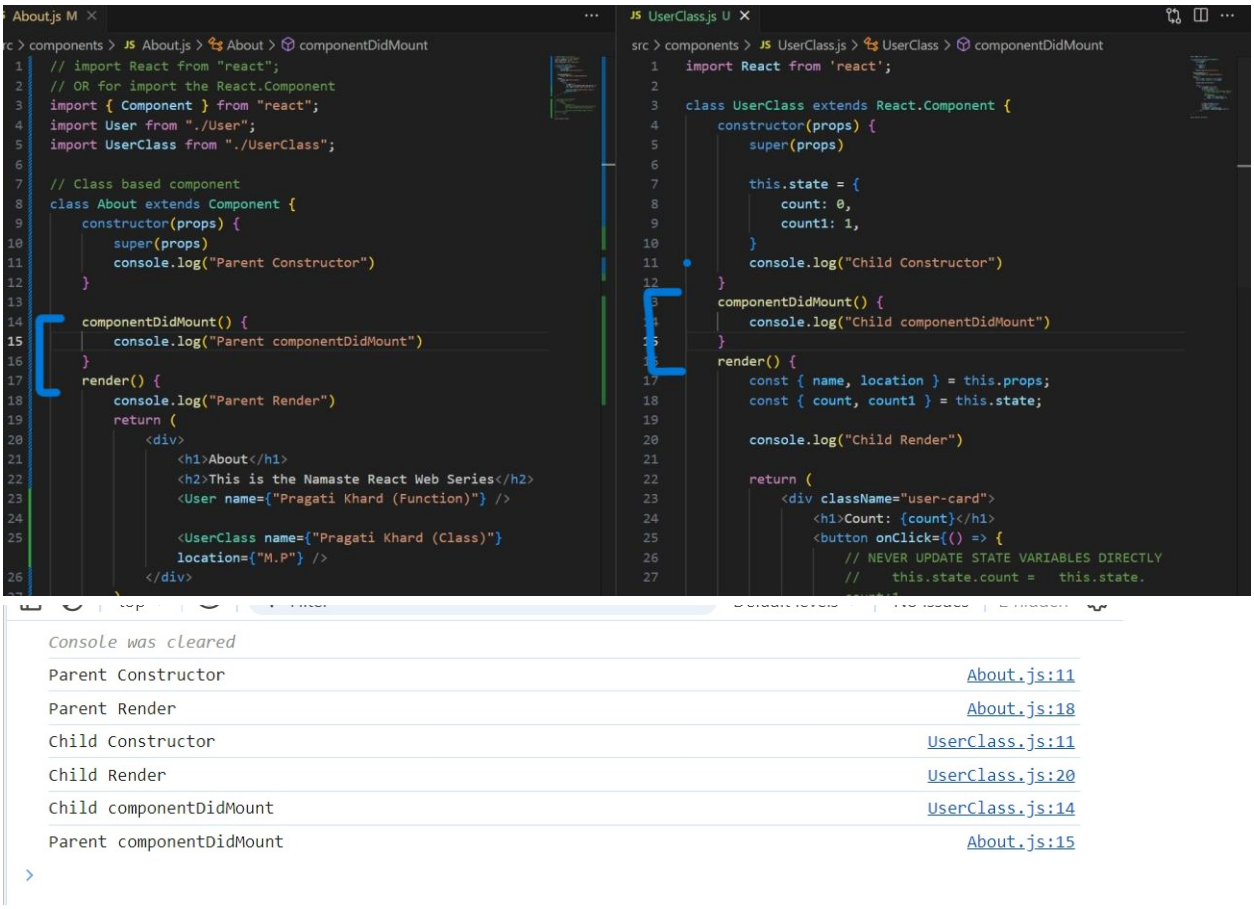
When the about component is loaded its a parent component so first of all about component is initialized , a new instant of class is created so first constructor of the parent is called then the render of parents is called then its go to the children it again trigger the life cycle and constructor the child will call and then render of the child is called.

Class based component have some other methods -

<u>componentDidMount</u> – It will call when the component mount on the web page.
So first the constructor will call then render then componentDidMount method, this is the order.



Suppose we had componentDidMount inside my parent component.



When the component is loaded first of all parent constructor will call then the render method will call, when its rendering it will see the child then it will trigger the lifecycle methods of child then it goes to the child then child constructor will be call then the render of child is called now there is no more component inside of it once the rendering is finished then the componentDidMount of child will call then the child component mount successfully once the child will mount successfully then the parent componentDidMount will be called that's how the lifecycle method works.

componentDidMount is used for Api call. But Question is why we are using inside of componentDidMount?

Take back to the functional component How we are make API calls in functional component, we make API call using useEffect() but why do we do inside useEffect . We do that because first we load our component once the component loaded basics details then make the API call and filled the details, So the react components load very fast. Otherwise, the component not render and keep on waiting to come from API. React will quickly render it then make the API call and fill the data.
In class-based component we will quickly render it then make the API call and fill the data.so that is the reason make the API call inside the componentDidMount().

Trickey question-
Previously we had parent and have one child, what if we have multiple children. Lets see how the lifecycle will works.

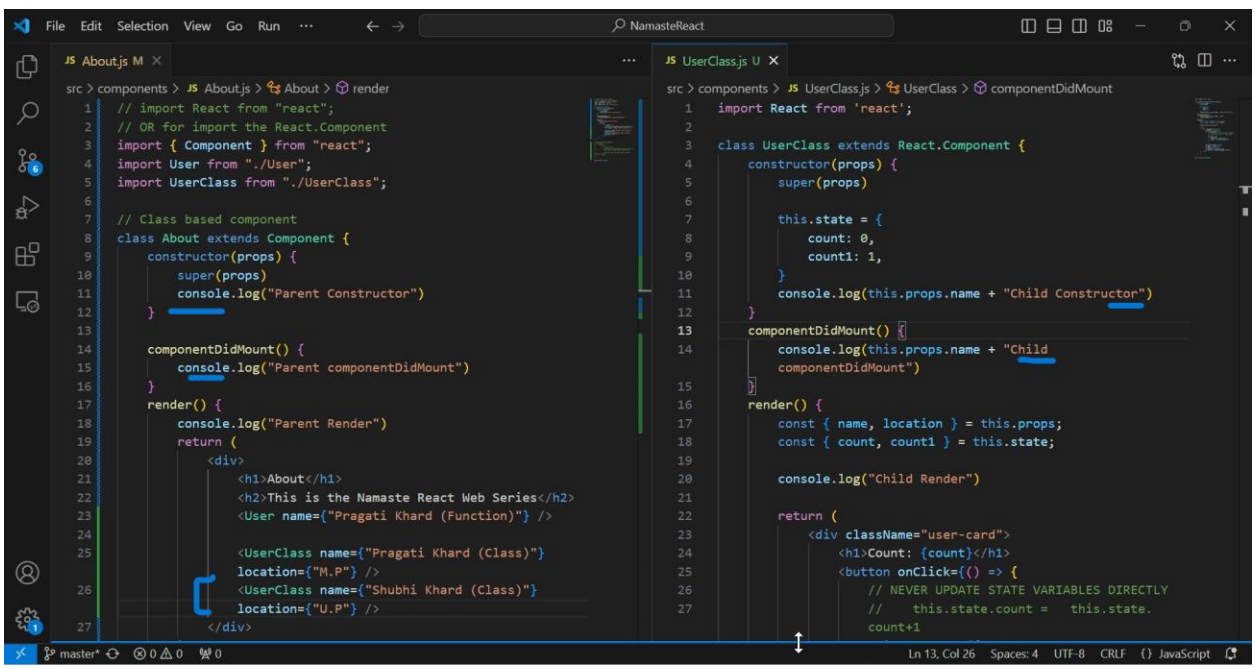
Find out the console.log order-
First the parent constructor will be called then parent render will be called it see there is the child also so it will trigger the lifecycle method of child first so it will call the Pragati Khard constructor then it will call the Pragati Khard render then it will call Pragati Khard componentDidMount() will called once the Pragati Khard lifecycle is finished then it will trigger the lifecycle method of Shubhi khard then the Shubhi khard constructor will be called the Shubhi khard render will be called and the Shubhi khard componentDidMount called once both the child lifecycle mounted then Parent componentDidMount will be called but this is wrong

Parent Constructor
Parent Render
Pragati Khard (Class)Child Constructor
Pragati Khard (Class)Child Render
Pragati Khard (Class)Child componentDidMount

Shubhi Khard (Class)Child Constructor
Shubhi Khard (Class)Child Render
Shubhi Khard (Class)Child componentDidMount
Parent componentDidMount

But the above output is wrong

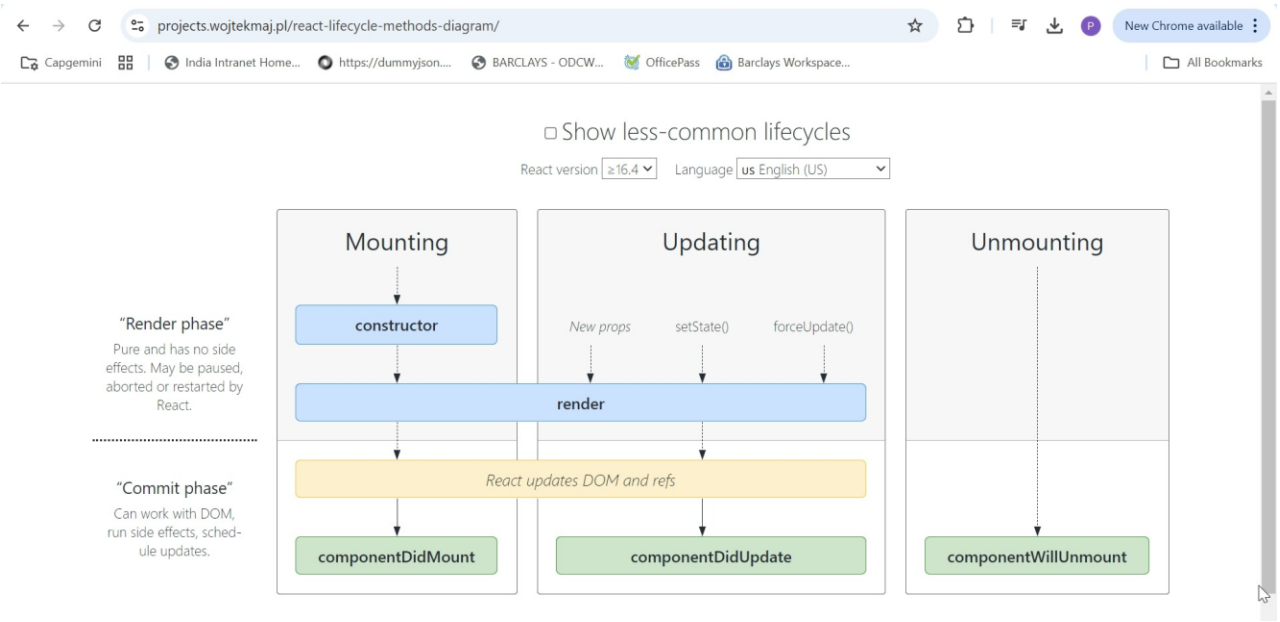
Below screenshot is the order




```
Console was cleared
Pragati Khard (Class)Child Constructor        UserClass.js:11
Pragati Khard (Class)Child Render             UserClass.js:20
Shubhi Khard (Class)Child Constructor         UserClass.js:11
Shubhi Khard (Class)Child Render              UserClass.js:20
Pragati Khard (Class)Child componentDidMount  UserClass.js:14
Shubhi Khard (Class)Child componentDidMount   UserClass.js:14
>
```

Reason – Refer react lifecycle method diagram

https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/



Only focusing about Mounting box

When the component mounting it is mounting in the 2 phases -

Render phase and Commit phase

when the component is mounting constructor is called then render is called then the react updates the DOM and the componentDidMount is called so that is why this is the best place to make Api call and this lifecycle is for every child and every parent in react.

First the parent constructor will be called then parent render will be called it see there is the child also so it will trigger the lifecycle method of child first so it will call the Pragati Khard constructor then it will call the Pragati Khard render but there is <u>2 childern in the parent component react optimises it</u> <u>react will not call the Pragati Khard componentDidMount()</u>
It will basically batch the render phases of 2 child. So what will happen these 2 childes render phases will happen then will commit phase happen together. This is the optimization of react.

So, first Pragati Khard Constructor will call them Pragati Khard render will call the the Shubhi khard constructor will be called the Shubhi khard render will be called and the commit phases will be batch together so the render phase will be batch and the commit

phase will be batch and the Pragati Khard componentDidMount and Shubhi khard componentDidMount called . So that is why output will in this order.

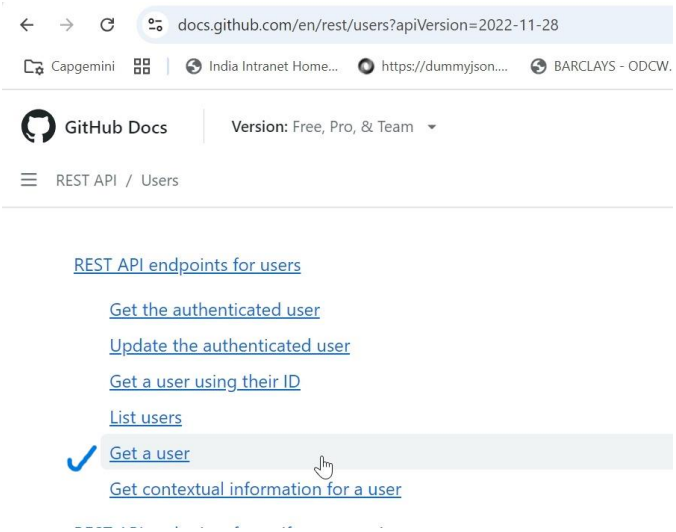
```
-Parent Constructor
-Parent Render

This is the render phase for both the cildrens

   -Pragati Khard (Class)Child Constructor
   -Pragati Khard (Class)Child Render
   -
   -Shubhi Khard (Class)Child Constructor
   -Shubhi Khard (Class)Child Render

<DOM UPDATED - IN SINGLE BATCH>

This is the commit phase for both the cildrens

-Pragati Khard (Class)Child componentDidMount
-Shubhi Khard (Class)Child componentDidMount

-Parent componentDidMount
```

Make the API calls in class-based component -

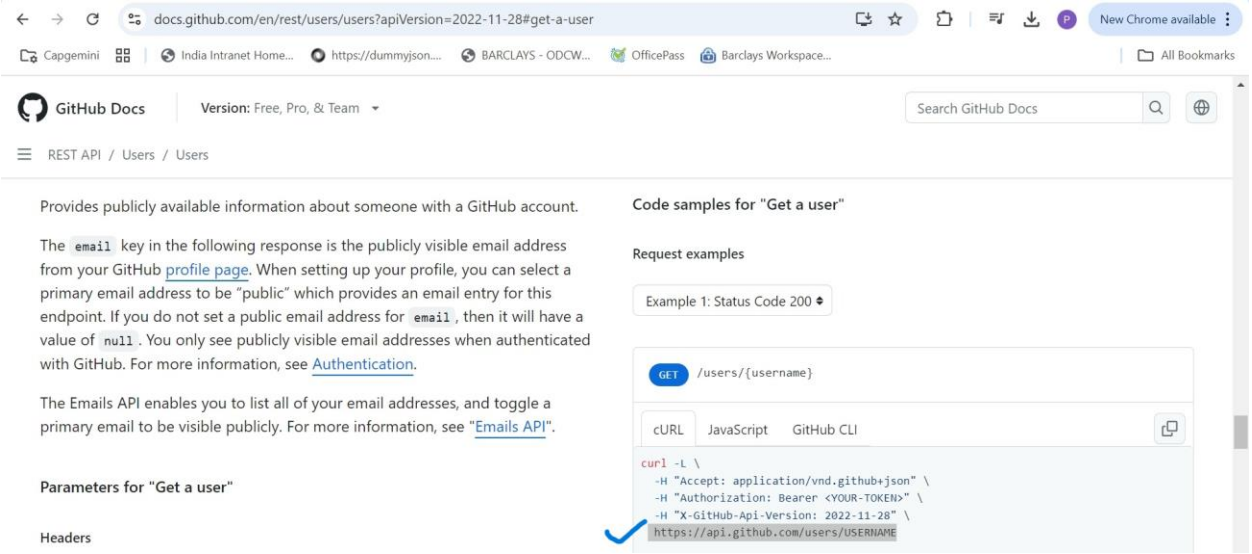
We are using the github user API-

1. Search git hub user API

2. Choose



3. Choose https://api.github.com/users/USERNAME



In please of USERNAME use your github username like-
https://api.github.com/users/pragkhard

```
JS About.js M    JS UserClass.js U ×    ⓘ README.md M
src > components > JS UserClass.js > UserClass > componentDidMount
1   import React from 'react';
2
3   class UserClass extends React.Component {
4       constructor(props) {
5           super(props)
6
7           this.state = {
8               count: 0,
9               count1: 1,
10          }
11          // console.log(this.props.name + "Child Constructor")
12      }
13      async componentDidMount() {
14          // console.log(this.props.name + "Child componentDidMount")
15
16          const data = await fetch("https://api.github.com/users/pragkhard")
17          const json = await data.json();
18
19          console.log(json)
20      }
21      render() {
22          const { name, location } = this.props;
23          const { count, count1 } = this.state;
24
25          // console.log(this.props.name + "Child Render")
26
27          return (
28              <div className="user-card">
29                  <h1>Count: {count}</h1>
```

Now how to update the json data on to the webpage-

Now we will create the state variable and update the state which will just get the data

Create the state variable with dummy data-

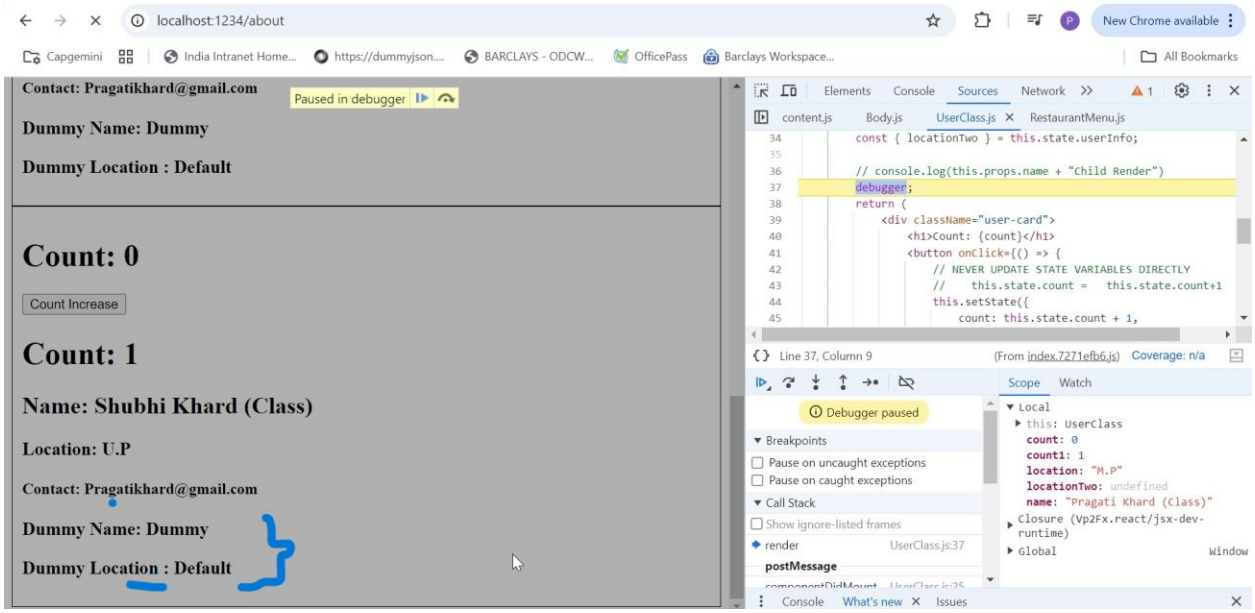
```
this.state = {
        count: 0,
        count1: 1,
        userInfo: {
            name: "Dummy",
            location: "Defect",
        }}
```

How I will update state userInfo , will update using setState and the setState will have the object userInfo so I will put json inside userInfo like-

```
this.setState({
        userInfo: json,
    })
```

```
JS About.js M    JS UserClass.js U ×    ⓘ README.md M
src > components > JS UserClass.js > UserClass > render
1   import React from 'react';
2
3   class UserClass extends React.Component {
4       constructor(props) {
5           super(props)
6
7           this.state = {
8               count: 0,
9               count1: 1,
10
11              userInfo: {
12                  name: "Dummy",
13                  location: "Defect",
14              }
15          }
16          // console.log(this.props.name + "Child Constructor")
17      }
18      async componentDidMount() {
19          // console.log(this.props.name + "Child componentDidMount")
20
21          const data = await fetch("https://api.github.com/users/pragkhard")
22          const json = await data.json();
23
24          this.setState({
25              userInfo: json,
26          })
27
28          console.log(json)
29      }
```

How the lifecycle will work – As soon as userClass loaded the constructor was load and the contractor was called the state variable created with some default value. After constructor render happen and the state variable have some default value so the render happen with the default value name:”dummy” location:”default” that means react will update the DOM with dummy data



After that constructor was called render happen and react updated the DOM with some dummy data now the componentDidMount was called with the API call was made, when the API call was made it called setState when the setState was called the mounting cycle happen now will see the updating cycle. When we call the setState updating cycle begins.
Mounting cycle finished when the component rendered once, component render once with some dummy data very quickly we didn't wait for API call to finish , component render with some dummy data so the user see something that is the reason we are using simmer ui.
When we do the setState updating phase will start and setState updates the state variable. When the state variable updates reacts trigger renders once again but this time the state variable changes with the updated value/ new value so react will render it so now in the updating cycle react will updates the DOM with the new value and after that comonentDidUpdate will call .

componentDidUpdate – compoenntDidUpdate called later on at the end.

```
* --- MOUNTING ----

 Constructor (dummy)
 Render (dummy)
    <HTML Dummy >
 Component Did MOunt
    <API Call>
    <this.setState> -> State variable is updated

 ---- UPDATE-------------

    render(APi data)
    <HTML (new API data>)
    componentDidUpdate
```

First the constructor is called then render is called and this time constructor and render is update with dummy data when the render happen now the webpage is loaded and now the HTML has dummy data on the webpage for few millisecond then componentDidMount is called and when the componentDidMount is called we make it API calls inside componentDidMount when the API call is called after that we do the this.setState() now this finish the mounting cycle. Once the mounting cycle is finished now setState was called when there is setState it triggers the reconciliation process and update the cycle. Now the update cycle will call when the setState is called render method will called once again and now the update cycle is begin and in the update cycle render method will called but when the setState called the render method will happen with API data because now with this setState state variable updated so the render will call with the updated data , render is happen now the react updates DOM , now the webpages is loaded . HTML is loaded with new API data at this point user will see API data on webpage. After that it will called componentDidUpdate. This is how the whole cycle method works.

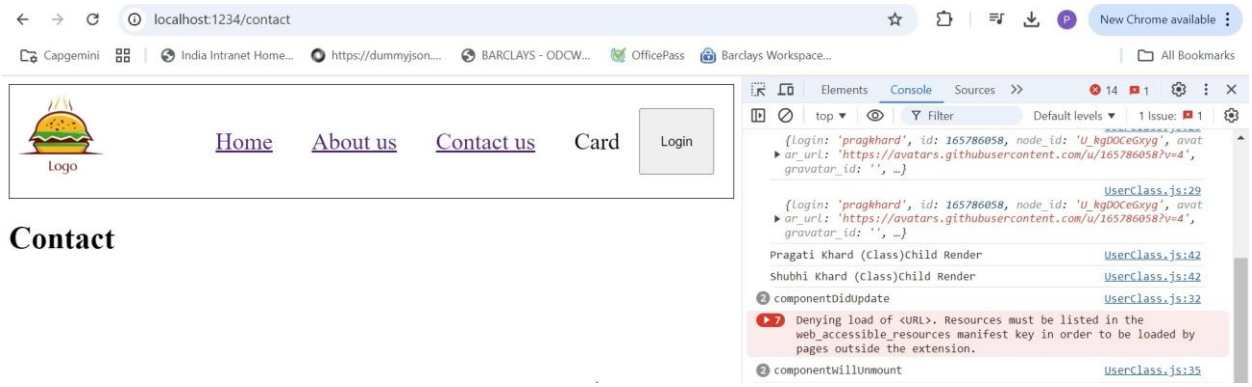
```
                                                    index.7271efb6.js:71
Download the React DevTools for a better development experience:
https://reactjs.org/link/react-devtools
Pragati Khard (Class)Child Constructor                  UserClass.js:17
Pragati Khard (Class)Child Render                       UserClass.js:39
Shubhi Khard (Class)Child Constructor                   UserClass.js:17
Shubhi Khard (Class)Child Render                        UserClass.js:39
Pragati Khard (Class)Child componentDidMount            UserClass.js:20
Shubhi Khard (Class)Child componentDidMount             UserClass.js:20
                                                        UserClass.js:29
  {login: 'pragkhard', id: 165786058, node_id: 'U_kgDOCeGxyg', avat
▶ ar_url: 'https://avatars.githubusercontent.com/u/165786058?v=4',
  gravatar_id: '', …}
                                                        UserClass.js:29
  {login: 'pragkhard', id: 165786058, node_id: 'U_kgDOCeGxyg', avat
▶ ar_url: 'https://avatars.githubusercontent.com/u/165786058?v=4',
  gravatar_id: '', …}
Pragati Khard (Class)Child Render                       UserClass.js:39
Shubhi Khard (Class)Child Render                        UserClass.js:39
(2) componentDidUpdate                                  UserClass.js:32
```

First the mounting cycle is happens then the updating cycle is happens.

componentWillUnmount- This function will call just before the component is unmount.

Unmount means when this component will go from this HTML.

When it is gone when I will move to the new page.



When I move to the contact us page then componentWillUnmount will call.

Live batch

Deeper inside of it-

Don't compare the functional based component to the class based component in respect of coding.

How we can write this code in class bases component –

Functional component -

```
useEffect(() => {
      fetchData();
   }, [count, count2]);
```

Class component -

```
componentDidMount(prevProps, prevState) {
      if (
         this.state.count == !prevState.count ||
         this.state.count2 == !prevState.count2
      )
       console.log("Parent componentDidMount")
   }
```

If we are using the 2 useEffect in functional component, then how we can write in the class based component –

Functional component –

```
useEffect(() => {
      fetchData();
   }, [count]);

useEffect(() => {
      fetchData();
   }, [count2]);
```

Class component -

```
componentDidMount(prevProps, prevState) {
      if (this.state.count == !prevState.count  )
     if (this.state.count2 == !prevState.count2)
       console.log ("Parent componentDidMount")
   }
```

React is single page application when you are changing your pages it will not reloading your page. It will just be changing your page. But it is the bad thing of single page application.

```
 componentDidMount() {
      setInterval(() => {
         console.log ("NAMASTE REACT OP")
      }, 1000)
      console.log ("Parent componentDidMount")
   }
```

Suppose we do the setInterval , it will print the NAMASTE REACT OP after the sec, the issue with the single page application is when if I move to the new page still it will call and print NAMASTE REACT OP this is the proble of single page application. Because when your are changing or switching your page it will not reloading it it is just changing the components, react will reconciling . After a while it will hanging on the browser.

How we can resolve this issue- we can resolve this issue by using clear.Interval in componetWillUnmount

Issue –

```
componentDidMount() {
      this.timer = setInterval(() => {
         console.log ("NAMASTE REACT OP")
      }, 1000)
      console.log ("Parent componentDidMount")
   }
```

Resolved –

```
componentWillUnmount() {
    clearInterval(this.timer)
    console.log(componentWillUnmount)
  }
```

So now we are switching from one page to another page it will not call NAMASTE REACT OP

What happen when we are using the setInterval inside the useEffect-

```
useEffect(() => {
    setInterval(() => {
      console.log("NAMASTE REACT OP Functional Comp.")
    }, 1000);
  }, [])
```

If we are switching from one page to another page still it will print "NAMASTE REACT OP Functional Comp."

We can resolved by this way-

```
useEffect(() => {
    const timer = setInterval(() => {
      console.log("NAMASTE REACT OP Functional Comp.")
    }, 1000);

    return () => {
      clearInterval(timer);
    };
  }, [])
```

Return the function inside useEffect, this function will called when you are unmounting the component

Like this we can get the componentWillUnmount() method in functional component.

Output question-

```
useEffect(() => {
      console.log("useEffect")

      return () => {
         console.log("useEffect Return")
      };
   }, [])
   console.log("Render")
```

o/p –

useEffect

Render

useEffect return // when we are switching from one comp to another