```
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayOut />,
    errorElement: <Error />,
    children: [
      { path: "/", element: <Body /> },
      {
        path: "about",
        element: <About />,
        children: [
          {       You, 1 minute ago • Uncommitted changes
            path: "profile",
            element: <Profile />,
          },
        ],
      },
      { path: "/contact", element: <Contact /> },
      { path: "/restaurant/:id", element: <RestrauntMenu /> },
    ],
  }, //
]);
const root = ReactDOM.createRoot(document.getElementById("root"));
```

If we write path:"profile" it means profile relative to its parent.

If we write "/profile" it means it is relative to localhost1234 i.e the root

Class based Component

Class Profile extends React.Component{
render(){
<h1>hello this is my h1 </h1>
}

}
This render methods returns jsx

pter 08 - Let's get Classy > ProfileClass.jsx > ...

```jsx
import React from "react";
class ProfileClass extends React.Component {
  render() {
    return <h1>This is my h1 tag {this.props.name} </h1>;
  }
}
export default ProfileClass;
```

And if you want to keep state , keep it like this

```jsx
import React from "react";
class ProfileClass extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 90,
    };
  }
  render() {
    return (
      <h1>
        This is my h1 tag {this.props.name}
        {this.state.count}{" "}
      </h1>
    );
  }
}
export default ProfileClass;
```

```jsx
import React from "react";
class ProfileClass extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 90,
    };
    console.log("Constructor called");
  }

  render() {
    console.log("Component is rendered");
    return (
      <h1>
        This is my h1 tag {this.props.name}
        {this.state.count}{" "}
        <button
          onClick={() => {
            this.setState({
              count: 99,
            });
          }}
        >
          Click here
        </button>
      </h1>
    );
  }
}
export default ProfileClass;
```

constructor(props)

super(props)

This.state for state and this.setstate for setting it

First constructor is called and then component is rendered and then CompountdidMount is invoked.

So the best place to make Api call is CompountdidMount

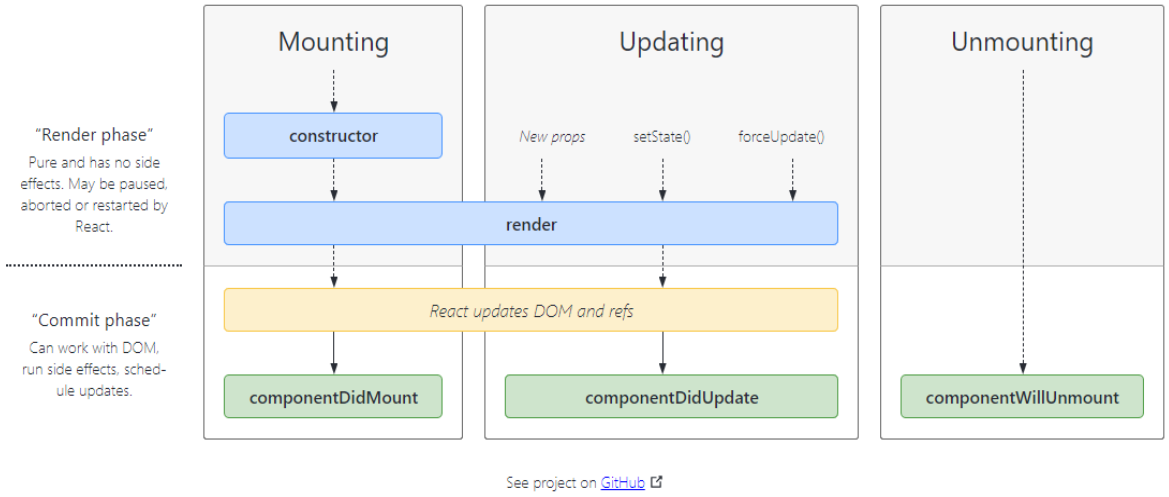
Why Component Did Mount is the best place to make an api call?

Because it is called after the first render.
Suppose we have a parent component whose has constructor , render and componentDidMount and child also has same constructor, render and ComponentDidMount.
Which one will be called in which sequence?

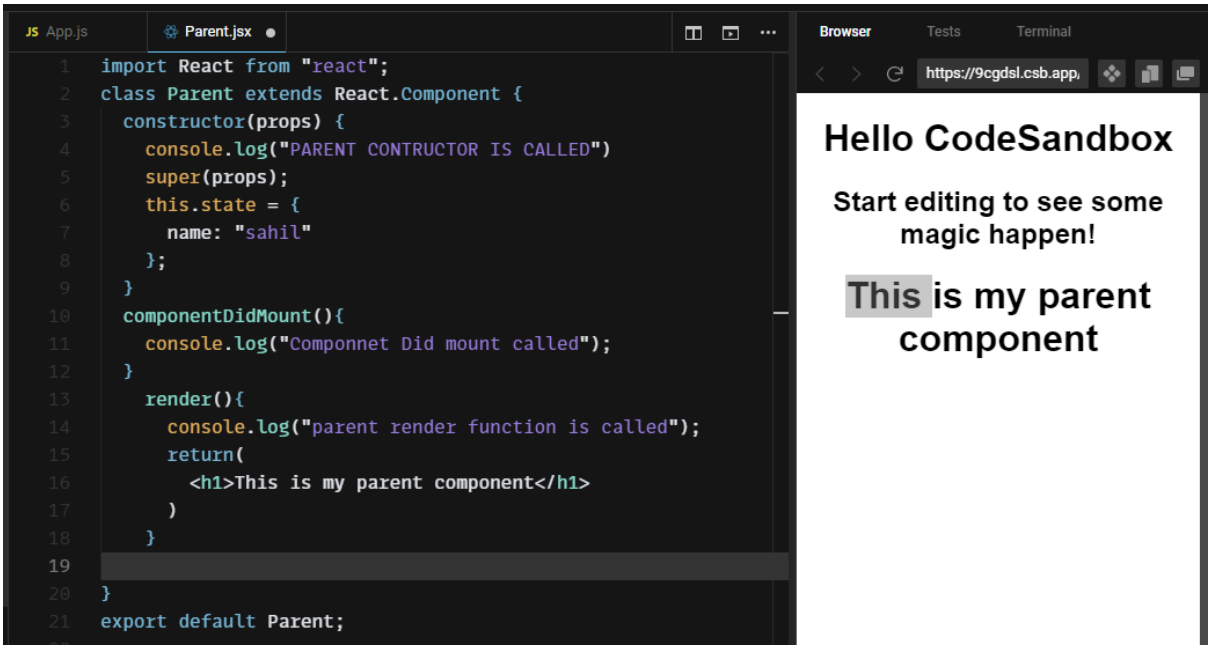
First is parent constructor.
Then parent render function is called.
Then it will go to children.
In child component, first the constructor of child is called, then it calls the child Render.
Then next thing is child component ComponnetDidMount
And the last thing is parent component componentDidMount

So in a component, constructor is called, then render method is called and then componentDidMount is called.

[React lifecycle methods diagram]



Normally componentDidMount is called for parent first and then for child.
But suppose there is an api call in child component, so it will not change that much and call componentDidMount for parent first and then ComponentDidMount for child later



In the above code snippet, Firstly the constructor method will be called.
Because all the states with their initial values will be initialised in constructor only.

Next phase is render. Where the jsx is put inside the dom.
Then we have componentDidMount which is invoked.
Normally ComponentDidMount is called for api calls.
Because in first render, it will fill the dom with same empty data and when we call api in ComponentDidMount, it will fill it with new data.

Lets do one thing. Lets do one thing call an api in ComponentDidMount and see what happens?

Now we called an api in componentDidMount and it

changes the state . So now, the diffing algorithm will work.
Component Will again render.
And now this time, componentDidUpdate will be called.
No componentDidMount gets called.
Note down, componentDid update will not be called for the first render.
componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount()

Parent component

```
import Child from "./Child";
class Parent extends React.Component {
  constructor(props) {
    console.log("PARENT CONTRUCTOR IS CALLED");
    super(props);
    this.state = {
      name: "sahil"
    };
  }
  async componentDidMount() {
    console.log("Componnet Did mount called for parent");
    const data = await fetch("https://api.github.com/users/akshaymarch7");
    const json = await data.json();
    console.log(json);

    this.setState({
      name: json.company
    });
  }
  componentDidUpdate() {
    console.log("Component Did Update is called for parent component");
  }
  render() {
    console.log("parent render function is called");
    return (
      <div>
        <Child />
      </div>
    );
  }
```

Child component

```
import React from "react";
class Child extends React.Component {
  constructor(props) {
    console.log("constructor is called for child component");
    super(props);
    this.state = {
      child1: "c1"
    };
  }
  render() {
    console.log("child component is rendered");
    return <>this is my child Component</>;
  }
  componentDidMount() {
    console.log("componentDidMount is called for child component");
  }
  componentDidUpdate() {
    console.log("component did update for child ");
  }
}

export default Child;
```

```
Console was cleared                                   index.js:27
PARENT CONTRUCTOR IS CALLED                           index.js:27
PARENT CONTRUCTOR IS CALLED                           index.js:27
parent render function is called                      index.js:27
parent render function is called                      index.js:27
constructor is called for child component             index.js:27
constructor is called for child component             index.js:27
child component is rendered                           index.js:27
child component is rendered                           index.js:27
componentDidMount is called for child component       index.js:27
Componnet Did mount called for parent                 index.js:27
componentDidMount is called for child component       index.js:27
Componnet Did mount called for parent                 index.js:27
▸{login: 'akshaymarch7', id: 12824231, node_id: 'MDQ6VXNlcjEyODI0MjMx', avatar_url: 'https://avatars.githubusercontent.com/u/12824231?v=4', gravatar_id: '', …}   index.js:27
▸{login: 'akshaymarch7', id: 12824231, node_id: 'MDQ6VXNlcjEyODI0MjMx', avatar_url: 'https://avatars.githubusercontent.com/u/12824231?v=4', gravatar_id: '', …}   index.js:27
parent render function is called                      index.js:27
parent render function is called                      index.js:27
child component is rendered                           index.js:27
child component is rendered                           index.js:27
component did update for child                        index.js:27
Component Did Update is called for parent component   index.js:27
```

In the above example,
First of all, the parent component constructor will be invoked. Then the render function of the parent will be called.
Then inside the render of parent, it will see the child component.
For the child component, it will call the constructor of child, then it will call the render function of child.
Then it will call the component did mount of child.
Though the console statement of component did mount the child component appears twice as we are in strict mode.
Then componentDidMount of parent will be called.
We are doing an api call , there which is changing one state of parent component, so it will update the virtual dom , update real dom and then render function of the parent will be called. Then it will render child component as the child is inside the parent.
For child, it will call componentDid update
And Then componentDid update will be called for parent.