What is React Component?

Components are rightly defined as the essential building blocks of any application created with React, and a single app most often consists of many components. A component is in essence, a piece of the user interface - splitting the user interface into reusable and independent parts, each of which can be processed separately.

Fig: Facebook Components

Important aspects of components include:

- Re-usability: We can reuse a component used in one area of the application in another area. This speeds up development and helps avoid cluttering of code.
- Nested components: A component can contain within itself, several more components. This helps in creating more complex design and interaction elements.
- Render method: In its minimal form, a component must define a render method specifying how it renders to the DOM.
- Passing Props (Properties): A component can also receive props. These are properties that its parent passes to specify particular values.

Let's move on and look into the types of components.

Type of React Components

There are two types of components in React:

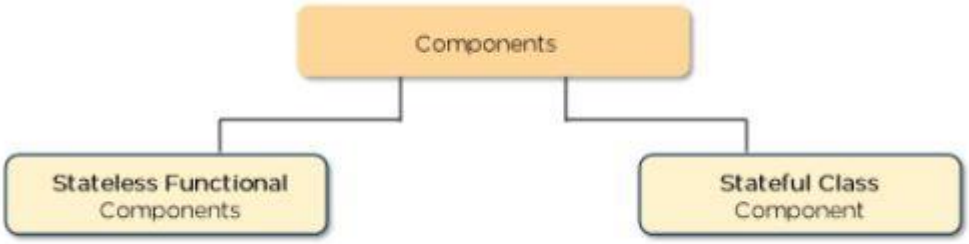


Fig: React components

- Functional components: These types of components do not have a state of their own and only possess a render method. They are also referred to as stateless components. They may by way of props (properties), derive data from other components.

```
const App = () => {

  return (

    <div>

      <h1>React Functional Component</h1>

    </div>

  );

};
```

The code depicted above creates a functional component App that returns an h1 heading on a web page. Since it is a functional component, it does not hold or manage any state of its own.

- Class components: These types of components hold and manage their unique state and have a separate render method to return JSX on the screen. They are referred to as stateful components too, as they can possess a state.

```
class App extends React.Component {

  constructor(props) {

    super(props);
```

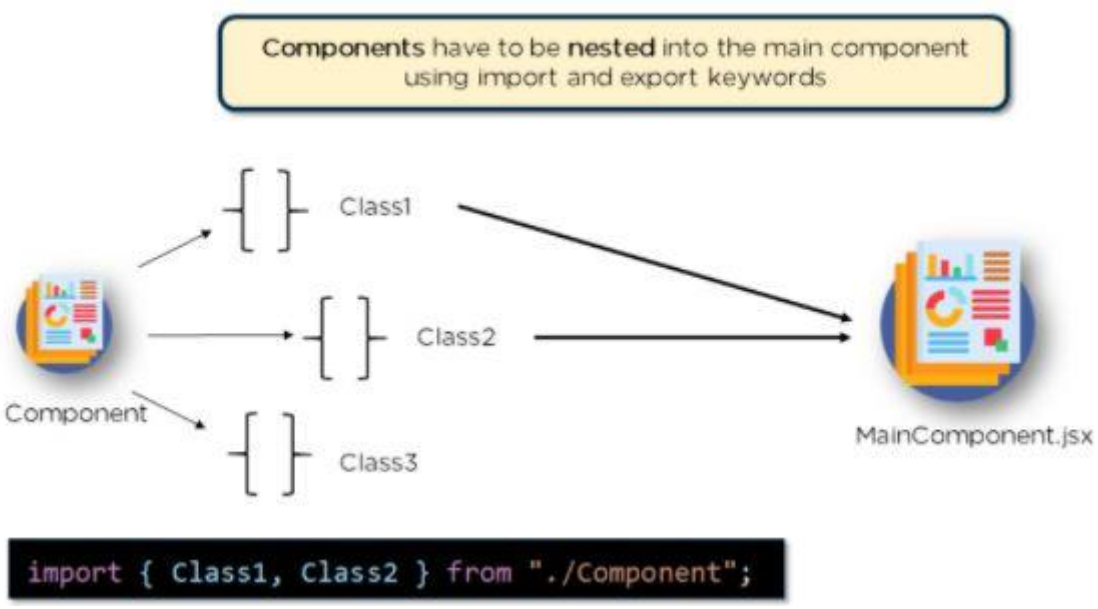
```
    this.state = {
      value: '',
    };
  }
  onChange = event => {
    this.setState({ value: event.target.value });
  };
  render() {
    return (
      <div>
        <h1>React Class Component</h1>
        <input
          value={this.state.value}
          type="text"
          onChange={this.onChange}
        />
        <p>{this.state.value}</p>
      </div>
    );
  }
}
```

The code mentioned above creates a class component App that extends Component class from React library. The class has a constructor that contains the state of the web application. onChange function updates the value of the state with the value that the user enters in the input field. render() method outputs the input field to the screen. The state gets updated with the value that the user enters, and the value output is presented to the screen inside the <p> tags.

React Nesting Components

In React, we can nest components inside within one another. This helps in creating more complex User Interfaces. The components that are nested inside parent components are called child components. Import and Export keywords facilitate nesting of the components.

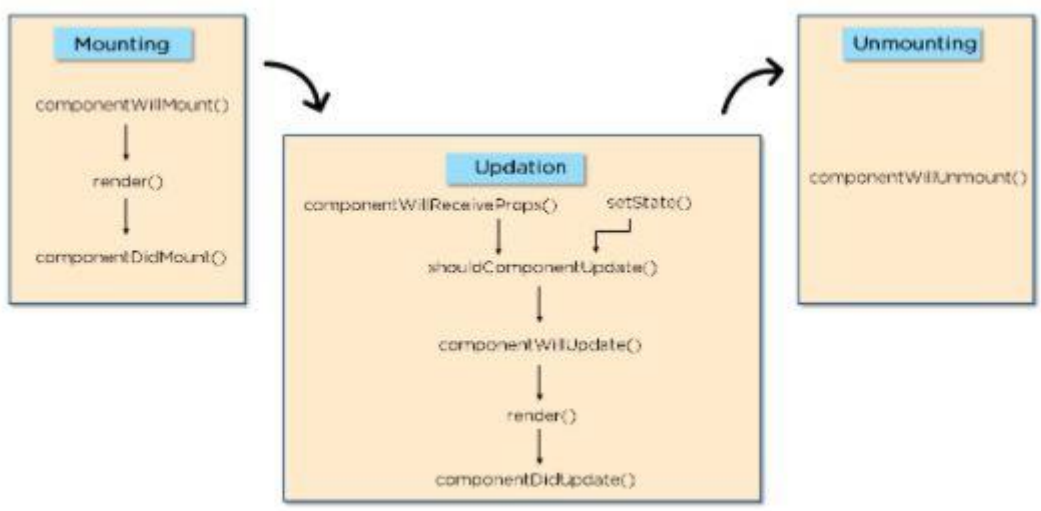
- Export - This keyword is used to export a particular module or file and use it in another module.
- Import - This keyword is used to import a particular module or file and use it in the existing module.



Importing named values allows the user to import multiple objects from a file.

React Components Lifecycle

We have learned so far that the React web applications are actually a collection of independent components that run according to the interactions made with them. Each component in React has a life-cycle that you can monitor and change during its three main phases. There are three phases: Mounting, Updating, and Unmounting. You can use life-cycle methods to run a part of code at particular times in the process or application. You can refer to the below diagram to get a visual idea of the life-cycle methods and their interactions in each phase.



Mounting

Mounting is the phase when the component is mounted on the DOM and then rendered on the webpage. We call these methods in the following order when an element is being created and inserted into the DOM:

- componentWillMount()
- render()
- componentDidMount()

componentWillMount()

This function is called right before we mount the component on the DOM. So after this method executes, the component gets mounted on the DOM. It executes this method before the first render of the React application. So all the tasks that have to be done before the component mounts are defined in this method.

render()

This function mounts the component on the browser.

componentDidMount()

We invoke this function right after the component is mounted on the DOM i.e., this function gets called once after the render() function is executed for the first time. Generally, in React applications, we do the API calls within this method.

Updating

We call the methods in this phase whenever State or Props in a component get updated. This allows the React application to “react” to user inputs, such as clicks and pressing keys on the keyboard, and ultimately create a responsive web user interface. These methods are called in the following order to carry on the update of the component:

- componentWillReceiveProps()
- setState()
- shouldComponentUpdate()
- componentWillUpdate()
- render()
- componentDidUpdate()

componentWillRecieveProps()

We call This function before a component gets its props reassigned.

setState() Function

This function can be called explicitly at any moment. This function is used to update the state of a component.

shouldComponentUpdate()

This function is called before rendering a component and tells React whether it should update the component on receiving new props or state. It takes the new Props and new State as the arguments and returns whether or not to re-render the component by returning either True or False.

componentWillUpdate()

This function is called once before the render() function is executed after the update of State or Props.

render()

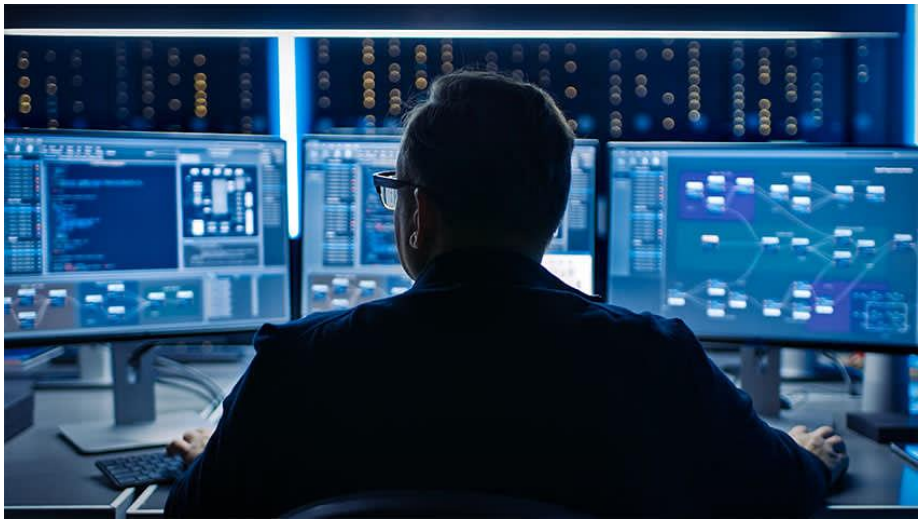
The component gets rendered when this function is called.

componentDidUpdate() Function

This function is called once after the render() function is executed after the update of State or Props.

ReactJS Components: Type, Nesting, and Lifecycle

Lesson 4 of 12By Taha Sufiyan
Last updated on Feb 15, 202380620



PreviousNext

Table of Contents

React is perhaps the most popular front-end JavaScript library in today's times. From startups to big enterprises and corporations, React is finding phenomenal applications in many of these organizations. Leading and popular companies like Airbnb, Netflix, The New York Times are using this highly-useful technology on their websites, and their mobile applications. React's popularity grew mainly because of the increased performance it delivered to web applications when compared to Angular. React introduced several concepts that overcame the drawbacks of previous frontend frameworks.

This article lets you explore and learn one of the most important React concepts, Components. One of the reasons React has become so popular is the implementation of reusable User Interface elements. Components allow the User Interface to be simpler, modular, and hence, reusable.

You will find this article covering the following topics:

- What is React Component?
- Type of React Components
- Nesting React Components
- React Components Lifecycle

What is React Component?

Components are rightly defined as the essential building blocks of any application created with React, and a single app most often consists of many components. A component is in essence, a piece of the user interface - splitting the user interface into reusable and independent parts, each of which can be processed separately.

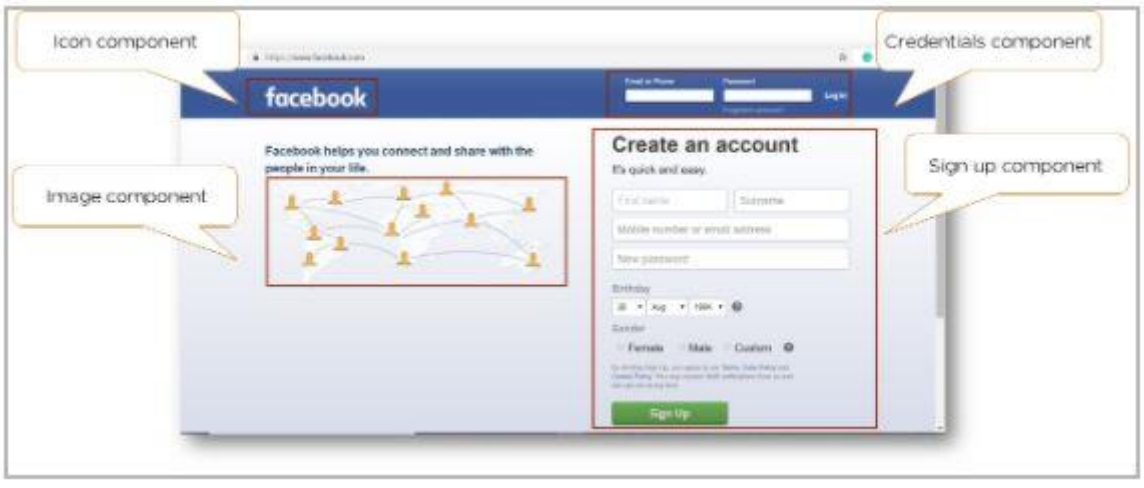


Fig: Facebook Components

Important aspects of components include:

- Re-usability: We can reuse a component used in one area of the application in another area. This speeds up development and helps avoid cluttering of code.
- Nested components: A component can contain within itself, several more components. This helps in creating more complex design and interaction elements.
- Render method: In its minimal form, a component must define a render method specifying how it renders to the DOM.
- Passing Props (Properties): A component can also receive props. These are properties that its parent passes to specify particular values.

Let's move on and look into the types of components.

Type of React Components

There are two types of components in React:

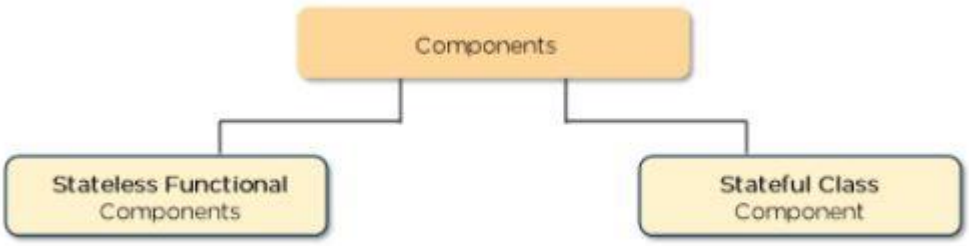


Fig: React components

- Functional components: These types of components do not have a state of their own and only possess a render method. They are also referred to as stateless components. They may by way of props (properties), derive data from other components.

```
const App = () => {

  return (

    <div>

      <h1>React Functional Component</h1>

    </div>

  );
```

```
};
```

The code depicted above creates a functional component App that returns an h1 heading on a web page. Since it is a functional component, it does not hold or manage any state of its own.

- Class components: These types of components hold and manage their unique state and have a separate render method to return JSX on the screen. They are referred to as stateful components too, as they can possess a state.

```
class App extends React.Component {

  constructor(props) {

    super(props);

    this.state = {

      value: '',

    };

  }

  onChange = event => {

    this.setState({ value: event.target.value });

  };

  render() {

    return (

      <div>

        <h1>React Class Component</h1>

        <input

          value={this.state.value}

          type="text"
```

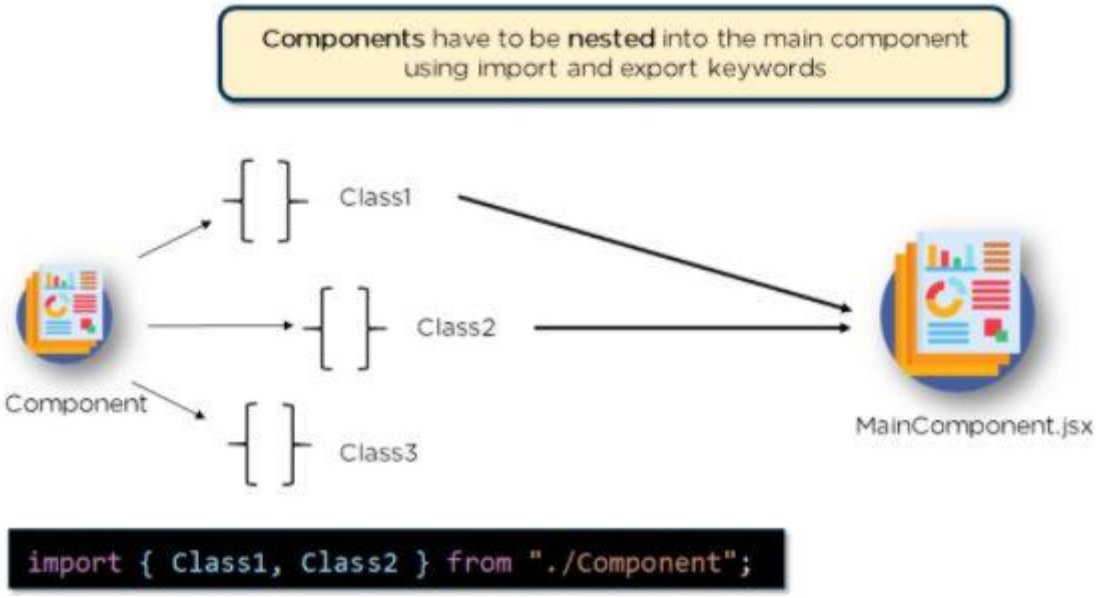
```
          onChange={this.onChange}
        />
        <p>{this.state.value}</p>
      </div>
    );
  }
}
```

The code mentioned above creates a class component App that extends Component class from React library. The class has a constructor that contains the state of the web application. onChange function updates the value of the state with the value that the user enters in the input field. render() method outputs the input field to the screen. The state gets updated with the value that the user enters, and the value output is presented to the screen inside the <p> tags.

React Nesting Components

In React, we can nest components inside within one another. This helps in creating more complex User Interfaces. The components that are nested inside parent components are called child components. Import and Export keywords facilitate nesting of the components.

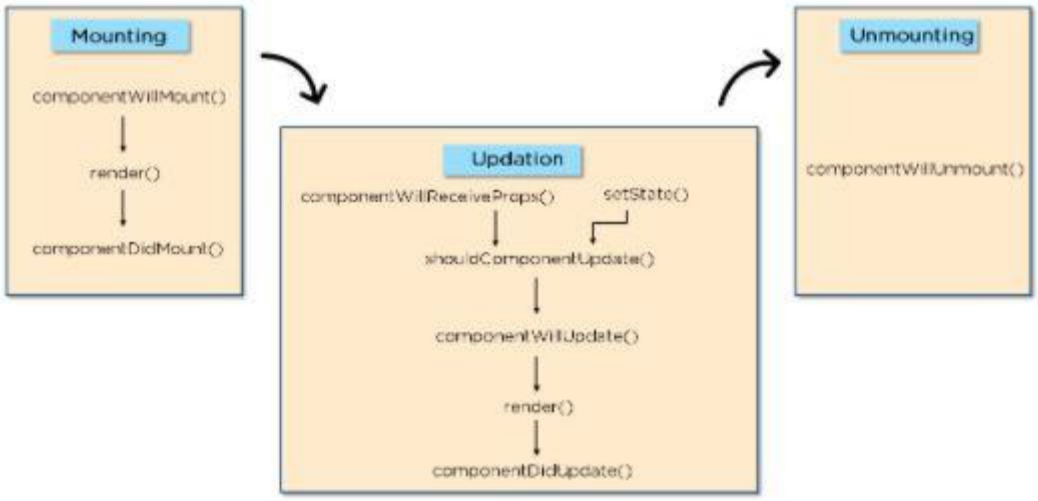
- Export - This keyword is used to export a particular module or file and use it in another module.
- Import - This keyword is used to import a particular module or file and use it in the existing module.



Importing named values allows the user to import multiple objects from a file.

React Components Lifecycle

We have learned so far that the React web applications are actually a collection of independent components that run according to the interactions made with them. Each component in React has a life-cycle that you can monitor and change during its three main phases. There are three phases: Mounting, Updating, and Unmounting. You can use life-cycle methods to run a part of code at particular times in the process or application. You can refer to the below diagram to get a visual idea of the life-cycle methods and their interactions in each phase.



Mounting

Mounting is the phase when the component is mounted on the DOM and then rendered on the webpage. We call these methods in the following order when an element is being created and inserted into the DOM:

- componentWillMount()
- render()
- componentDidMount()

componentWillMount()

This function is called right before we mount the component on the DOM. So after this method executes, the component gets mounted on the DOM. It executes this method before the first render of the React application. So all the tasks that have to be done before the component mounts are defined in this method.

render()

This function mounts the component on the browser.

componentDidMount()

We invoke this function right after the component is mounted on the DOM i.e., this function gets called once after the render() function is executed for the first time. Generally, in React applications, we do the API calls within this method.

Updating

We call the methods in this phase whenever State or Props in a component get updated. This allows the React application to “react” to user inputs, such as clicks and pressing keys on the keyboard, and ultimately create a responsive web user interface. These methods are called in the following order to carry on the update of the component:

- componentWillReceiveProps()
- setState()
- shouldComponentUpdate()
- componentWillUpdate()
- render()
- componentDidUpdate()

componentWillRecieveProps()

We call This function before a component gets its props reassigned.

setState() Function

This function can be called explicitly at any moment. This function is used to update the state of a component.

shouldComponentUpdate()

This function is called before rendering a component and tells React whether it should update the component on receiving new props or state. It takes the new Props and new State as the arguments and returns whether or not to re-render the component by returning either True or False.

componentWillUpdate()

This function is called once before the render() function is executed after the update of State or Props.

render()

The component gets rendered when this function is called.

componentDidUpdate() Function

This function is called once after the render() function is executed after the update of State or Props.

Unmounting

This is the last phase of the life-cycle of the component where the component is unmounted from the DOM. This phase has only one method:

componentWillUnmount()

This function is called once before we remove the component from the page and this marks the end of the life-cycle.

Now that we are familiar with all the life-cycle methods of React components, let us now look at an example to see how each function is implemented in the application code:

```
import React from "react";

import ReactDOM from "react-dom";
```

```
class Demo extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: "Hello!" };
  }
  componentWillMount() {
    console.log("componentWillMount()");
  }
  componentDidMount() {
    console.log("componentDidMount()");
  }
  switchState() {
    this.setState({ value: "Bye!" });
  }
  render() {
    return (
      <div>
        <h1>Greetings from Simplilearn, {this.state.value}</h1>
        <h2>
          <a onClick={this.changeState.bind(this)}>Click Here!</a>
        </h2>
```

```
      </div>
    );
  }
  shouldComponentUpdate(nextProps, nextState) {
    console.log("shouldComponentUpdate()");
    return true;
  }
  componentWillUpdate() {
    console.log("componentWillUpdate()");
  }
  componentDidUpdate() {
    console.log("componentDidUpdate()");
  }
}
ReactDOM.render(<Demo />, document.getElementById("root"));
```

Get Ahead of the Curve and Master React Today

Now that you've understood one of the most fundamental React's concepts, you could look to learn further and obtain the skills necessary to leverage React's rising popularity. Simplilearn's comprehensive Post Graduate Program in Full Stack Web Development could be an ideal next step for you to achieve the goal and become career-ready upon completion. To learn more, do take a look at our YouTube playlist which covers the key concepts of React in more detail. If you're a web and mobile developer, React training will greatly benefit you by broadening your skills and your career horizons.