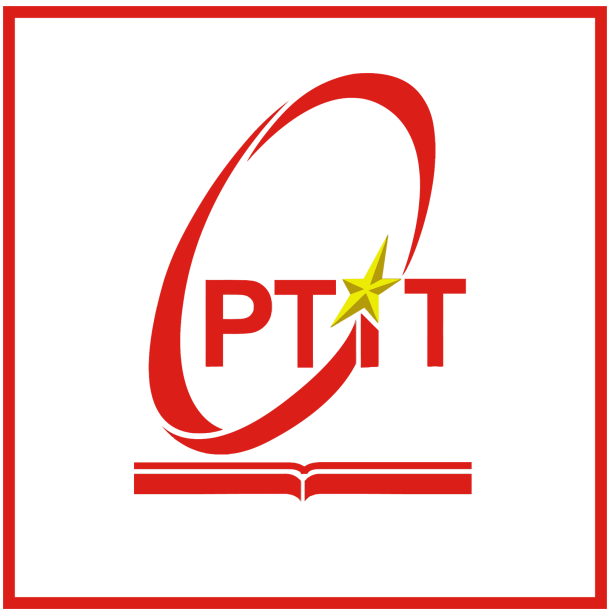
BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÁO CÁO THỰC TẬP CƠ SỞ TUẦN 5

TÌM HIỂU VỀ REACTJS

Giảng viên hướng dẫn: TS. Kim Ngọc Bách

Sinh viên thực hiện:

- Nguyễn Quang Trung - B22DCDT321

MỤC LỤC

Giới Thiệu

React.js là một thư viện Javascript đang nổi lên trong những năm gần đây với xu hướng Single Page Application. Trong khi những framework khác cố gắng hướng đến một mô hình MVC hoàn thiện thì React nổi bật với sự đơn giản và dễ dàng phối hợp với những thư viện Javascript khác. Hôm nay chúng ta sẽ tìm hiểu những khái niệm cơ bản trong React .

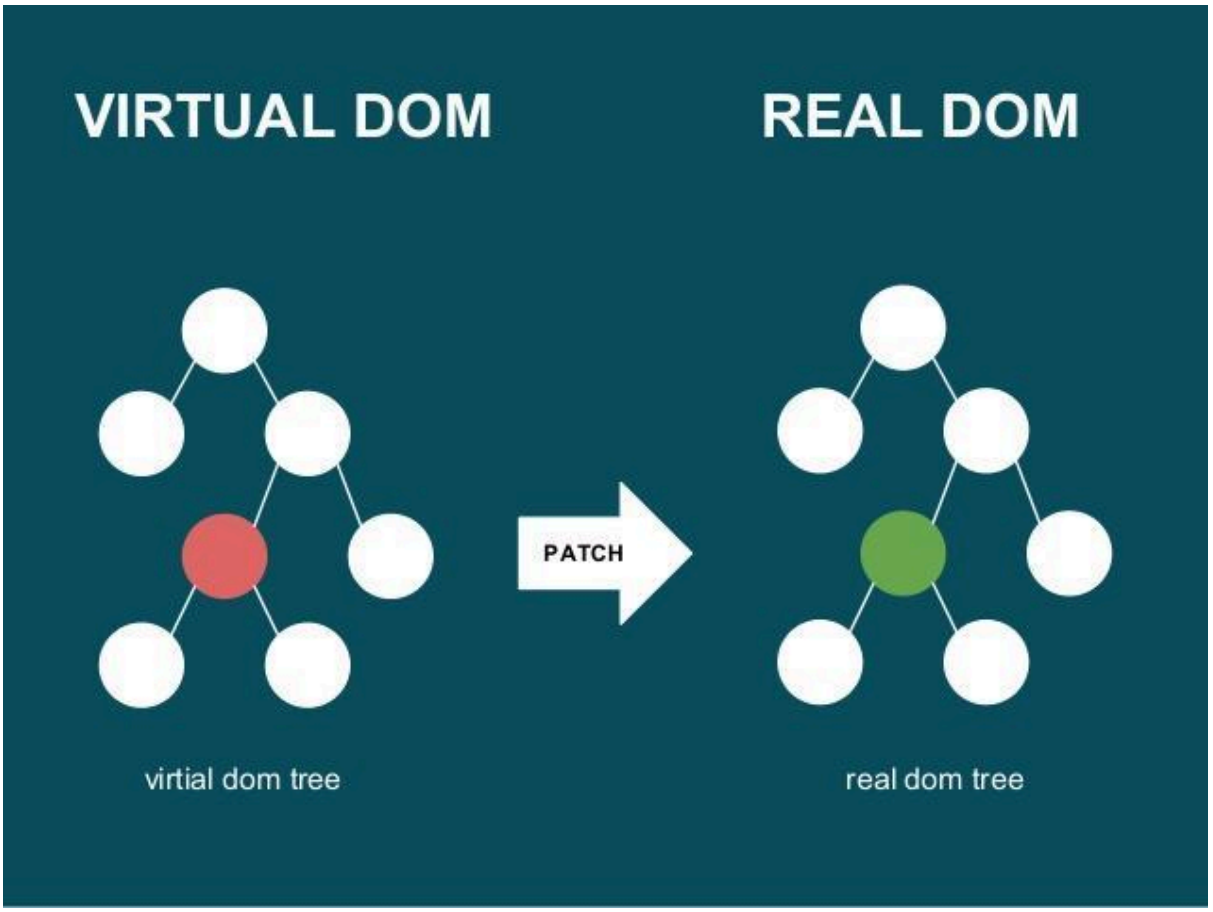
1. React là gì ?

React là một thư viện UI phát triển tại Facebook để hỗ trợ việc xây dựng những thành phần (components) UI có tính tương tác cao, có trạng thái và có thể sử dụng lại được. React được sử dụng tại Facebook trong production, và www.instagram.com được viết hoàn toàn trên React.

Một trong những điểm hấp dẫn của React là thư viện này không chỉ hoạt động trên phía client, mà còn được render trên server và có thể kết nối với nhau.

React sử dụng khái niệm DOM ảo (Virtual DOM) để chọn lựa và render những phần tử của node dựa trên sự thay đổi trạng thái khiến cho ta chỉ cần thay đổi ít thành phần nhất có thể để giữ DOM update.

2. Virtual DOM hoạt động như thế nào ?



Tưởng tượng bạn có một vật thể được thiết kế dựa trên một người. Nó có mọi bộ phận một người có thể có, và phản ánh lại trạng thái hiện tại của người đó. Đây là điều cơ bản React làm với DOM.

Bây giờ thử nghĩ rằng nếu bạn lấy vật thể đó và thay đổi một vài bộ phận. Thêm râu, bắp tay và đôi mắt xanh. Trong React, khi chúng ta tạo ra thay đổi, sẽ diễn ra 2 việc. Đầu tiên, React chạy một thuật toán so sánh sự khác biệt để phát hiện ra thay đổi. Bước thứ 2 là điều hòa bằng cách cập nhật DOM với kết quả của thuật toán ở bước 1.

Cách React hoạt động là thay vì lấy một người thật và tái tạo lại từ đầu, React chỉ thay đổi khuôn mặt và tay. Điều này có nghĩa là khi bạn nhập vào 1 đoạn text và render xong thì đoạn text này sẽ không bị ảnh hưởng cho tới khi parent node được sắp xếp lại.

Vì React sử dụng một DOM giả nên cũng có khá nhiều ý tưởng thú vị xung quanh thư viện này. Ví dụ như ta có thể render DOM giả này trên server.

3. Component

React Component

Component

Lưu ý quan trọng

Khuyến nghị định nghĩa các component bằng hàm (function) thay vì lớp (class). Hãy xem cách di chuyển (migrate) từ class sang function.

Component là lớp cơ sở cho các React component được định nghĩa bằng class trong JavaScript. Mặc dù React vẫn hỗ trợ component dạng class, nhưng chúng tôi không khuyến khích sử dụng class trong mã mới.

Ví dụ:

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

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

1. context

Trong class component, context có thể truy cập qua this.context. Để sử dụng, bạn cần khai báo static contextType.

Một class component chỉ đọc được một context tại một thời điểm.

```
class Button extends Component {
  static contextType = ThemeContext;

render() {
        const theme = this.context;
        const className = 'button-' + theme;
        return <button
className={className}>{this.props.children}</button>;
```

```
  }
}
```

2. props

Props được truyền vào class component thông qua this.props.

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

3. state

State trong class component được truy cập qua this.state, và phải là một đối tượng. Không được thay đổi state trực tiếp – hãy dùng this.setState.

```
class Counter extends Component {
  state = { age: 42 };

handleAgeChange = () => {
      this.setState({ age: this.state.age + 1 });
  };

render() {
      return (
      <>
      <button onClick={this.handleAgeChange}>Tăng
tuổi</button>
      <p>Bạn {this.state.age} tuổi.</p>
      </>
      );
  }
}
```

4. constructor(props)

Dùng để khởi tạo state và ràng buộc (bind) các phương thức:

```
class Counter extends Component {
  constructor(props) {
      super(props);
```

```
        this.state = { counter: 0 };
        this.handleClick = this.handleClick.bind(this);
    }

    handleClick() {
        // ...
    }
}
```

5. componentDidMount()

Được gọi ngay sau khi component được gắn (mount). Thường dùng để fetch dữ liệu, thiết lập đăng ký, hoặc thao tác DOM.

6. componentDidUpdate(prevProps, prevState, snapshot?)

Chạy sau mỗi lần cập nhật, trừ lần render đầu tiên. Dùng để fetch lại dữ liệu khi props/state thay đổi.

7. componentWillUnmount()

Chạy trước khi component bị gỡ (unmount). Dùng để hủy fetch, hủy đăng ký hoặc dọn dẹp DOM.

8. componentDidCatch(error, info)

Dùng để bắt lỗi khi component con bị lỗi trong quá trình render.

9. Các phương thức vòng đời khác (Deprecated)

Các phương thức sau đã bị loại bỏ:
- componentWillMount → UNSAFE_componentWillMount
- componentWillReceiveProps → UNSAFE_componentWillReceiveProps
- componentWillUpdate → UNSAFE_componentWillUpdate

10. forceUpdate(callback?)

Dùng để ép component render lại, bỏ qua shouldComponentUpdate.

11. getSnapshotBeforeUpdate(prevProps, prevState)

Dùng để lấy dữ liệu từ DOM trước khi cập nhật.

12. React Class Component: render, setState, shouldComponentUpdate

render()

Phương thức `render` là phương thức duy nhất bắt buộc trong một class

component.

Phương thức `render` nên chỉ định những gì bạn muốn hiển thị trên màn hình, ví dụ:

```jsx
import { Component } from 'react';

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

React có thể gọi `render` bất kỳ lúc nào, vì vậy bạn không nên giả định rằng nó sẽ chạy vào một thời điểm cụ thể. Thông thường, `render` nên trả về một đoạn JSX, nhưng một vài kiểu trả về khác (như chuỗi) cũng được hỗ trợ. Để tính toán JSX trả về, phương thức này có thể đọc `this.props`, `this.state`, và `this.context`.

Bạn nên viết phương thức `render` như một hàm thuần (pure function), nghĩa là nó nên trả về cùng một kết quả nếu `props`, `state`, và `context` giống nhau. Nó cũng không nên chứa hiệu ứng phụ (side effects) như thiết lập đăng ký hay tương tác với API trình duyệt. Các hiệu ứng phụ nên xảy ra trong event handler hoặc trong các phương thức như `componentDidMount`.

Tham số: `render` không nhận tham số.

Giá trị trả về: Bất kỳ node React hợp lệ nào như `<div />`, chuỗi, số, portal, node rỗng (null, undefined, true, false) hoặc mảng node.

Lưu ý:
- `render` nên được viết như một hàm thuần.
- `render` sẽ không được gọi nếu `shouldComponentUpdate` được định nghĩa và trả về `false`.
- Khi chế độ Strict Mode bật, React sẽ gọi `render` hai lần ở môi trường phát

triển và bỏ qua một kết quả để phát hiện side effect.
- Không có sự tương ứng 1-1 giữa gọi `render` và `componentDidMount` hoặc `componentDidUpdate`.

---

setState(nextState, callback?)
Gọi `setState` để cập nhật trạng thái của component React.

```jsx
class Form extends Component {
  state = { name: 'Taylor' };

  handleNameChange = (e) => {
        const newName = e.target.value;
        this.setState({ name: newName });
  }

  render() {
        return (
        <>
        <input value={this.state.name} onChange={this.handleNameChange} />
        <p>Hello, {this.state.name}.</p>
        </>
        );
  }
}
```

`setState` đưa thay đổi vào hàng đợi, báo cho React rằng component và các component con của nó cần re-render với state mới.

Cảnh báo: Gọi `setState` không thay đổi `this.state` ngay lập tức trong đoạn code hiện tại:

```js
function handleClick() {
```

```
  console.log(this.state.name); // "Taylor"
  this.setState({ name: 'Robin' });
  console.log(this.state.name); // Vẫn là "Taylor"!
}
```

Dùng hàm để cập nhật theo state trước đó:

```js
handleIncreaseAge = () => {
  this.setState(prevState => {
        return { age: prevState.age + 1 };
  });
}
```

Tham số:
- `nextState`: đối tượng hoặc hàm.
- `callback` (tùy chọn): gọi sau khi update được thực hiện.

Giá trị trả về: `setState` không trả về gì.

Lưu ý:
- React có thể gom nhiều cập nhật vào một lần render.
- Sử dụng `componentDidUpdate` hoặc callback của `setState` nếu bạn cần thực hiện hành động sau khi state thay đổi.

---

shouldComponentUpdate(nextProps, nextState, nextContext)
Nếu được định nghĩa, React sẽ gọi để quyết định có cần re-render hay không.

Bạn có thể so sánh `this.props` với `nextProps` và `this.state` với `nextState` và trả về `false` để bỏ qua render.

```jsx
class Rectangle extends Component {
```

```
  state = { isHovered: false };

  shouldComponentUpdate(nextProps, nextState) {
        if (
        nextProps.position.x === this.props.position.x &&
        nextProps.position.y === this.props.position.y &&
        nextProps.size.width === this.props.size.width &&
        nextProps.size.height === this.props.size.height &&
        nextState.isHovered === this.state.isHovered
        ) {
        return false;
        }
        return true;
  }
}
```

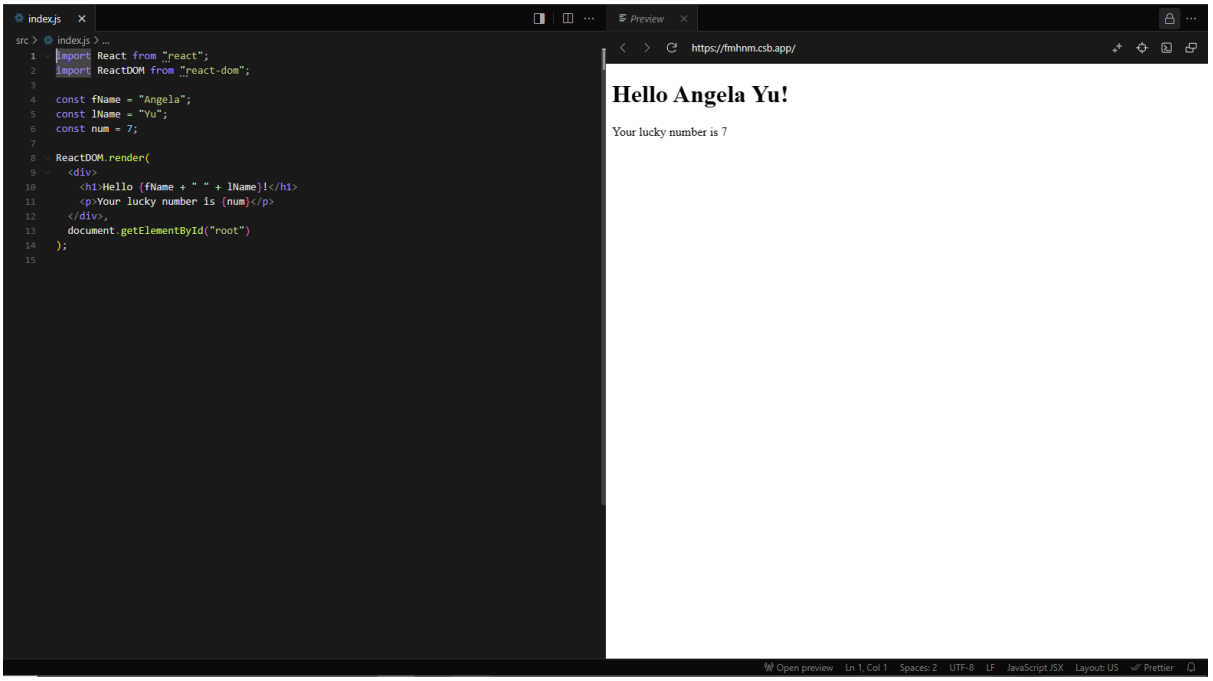
Tham số:
- `nextProps`: props mới.
- `nextState`: state mới.
- `nextContext`: context mới (nếu có).

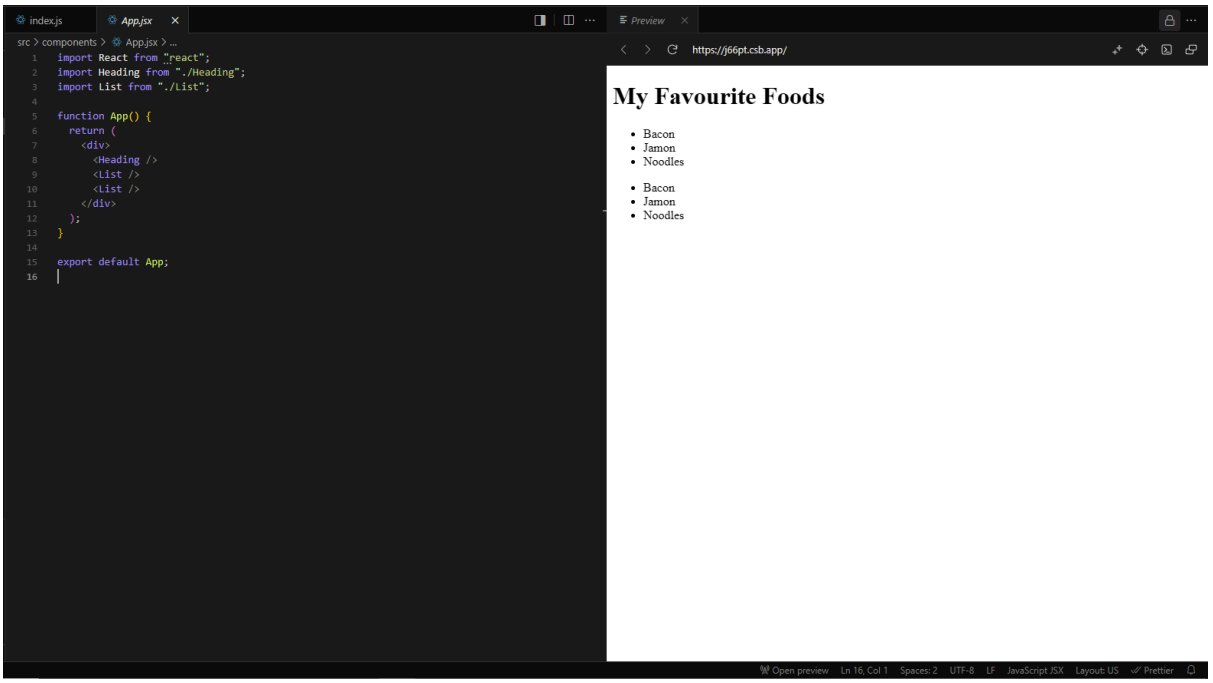
Giá trị trả về: `true` để cho phép render, `false` để bỏ qua.

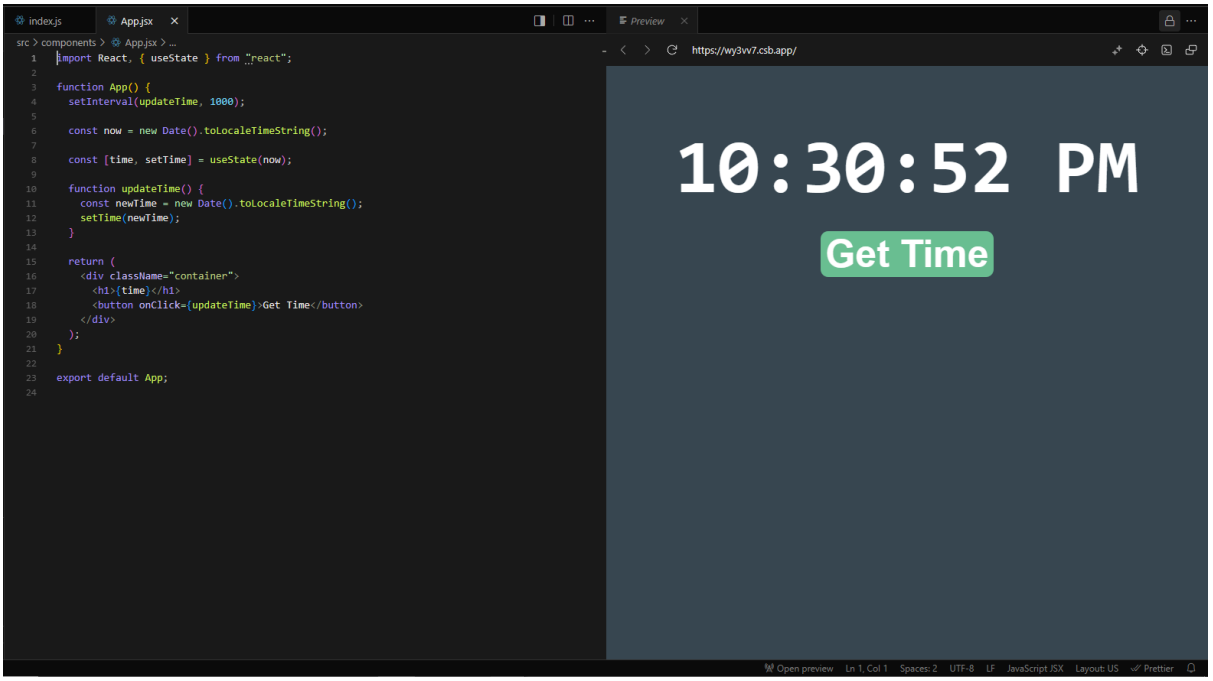
Lưu ý:
- Chỉ nên dùng cho tối ưu hiệu năng.
- Ưu tiên dùng `PureComponent` thay vì tự viết.
- Không nên so sánh sâu (deep comparison) hoặc dùng `JSON.stringify`.
- Trả về `false` không ngăn component con re-render.
- React vẫn có thể re-render nếu cần thiết.

Ghi chú:
Tối ưu class component bằng `shouldComponentUpdate` tương tự như dùng `memo` trong function component.

Ví dụ thực hành:






index.js
App.jsx
src > components > App.jsx > ...
import React, { useState } from "react";
function App() {
setInterval(updateTime, 1000);
const now = new Date().toLocaleTimeString();
const [time, setTime] = useState(now);
function updateTime() {
const newTime = new Date().toLocaleTimeString();
setTime(newTime);
}
return (

<h1>{time}</h1>
<button onClick={updateTime}>Get Time</button>

);
}
export default App;
Preview
https://wy3vv7.csb.app/
10:30:52 PM
Get Time
Open preview
Ln 1, Col 1
Spaces: 2
UTF-8
LF
JavaScript JSX
Layout: US
Prettier