**InterView React Native**

1. **Component trong React Native là gì?**

**Component** là thành phần cơ bản để xây dựng giao diện người dùng (UI) trong React Native. Mỗi component là một phần độc lập, tự đóng gói và có thể tái sử dụng được.

**Có hai loại component chính trong React Native:**

* **Native components:** Là các component được xây dựng dựa trên các thành phần UI nguyên bản của từng nền tảng (Android và iOS). Ví dụ: View, Text, Image, Button, etc.
* **Custom components:** Là các component do bạn tự xây dựng, có thể kết hợp các native components và các component khác để tạo ra các thành phần UI phức tạp hơn.

**Mỗi component có một số đặc điểm quan trọng:**

* **Props:** Dữ liệu được truyền vào component từ component cha.
* **State:** Trạng thái nội bộ của component, có thể thay đổi theo thời gian.
* **Render:** Hàm trả về giao diện UI của component.

**Lợi ích của việc sử dụng component:**

* **Tái sử dụng:** Có thể sử dụng lại component nhiều lần trong ứng dụng.
* **Bảo trì:** Giúp code dễ bảo trì và quản lý hơn.
* **Tính mô-đun:** Giúp chia nhỏ giao diện UI thành các phần nhỏ, dễ dàng phát triển và sửa lỗi.

**Virtual DOM là gì?**Virtual DOM (DOM ảo) là một cấu trúc dữ liệu nhẹ mô phỏng cấu trúc DOM thực tế của ứng dụng React Native. Nó được sử dụng để tối ưu hóa hiệu suất hiển thị bằng cách chỉ cập nhật các phần của giao diện người dùng bị thay đổi.  
**Cách thức hoạt động:**

1. Khi bạn tạo hoặc cập nhật một component trong React Native, React sẽ tạo ra một bản sao của DOM thực tế, gọi là Virtual DOM.
2. React so sánh Virtual DOM mới với Virtual DOM cũ để xác định các phần nào của giao diện người dùng bị thay đổi.
3. React chỉ cập nhật các phần bị thay đổi trong DOM thực tế, giúp cải thiện hiệu suất và giảm thiểu việc sử dụng tài nguyên.

**Single Page Application (SPA) trong React Native là gì?**

1. **Sự khác nhau giữa class component và functional component trong React Native ?**

**Class Components:**

* Là các thành phần được định nghĩa bằng cách sử dụng class.
* Yêu cầu phải kế thừa từ lớp cơ sở React.Component.
* Sử dụng phương thức render() để trả về phần tử JSX.
* Có thể sử dụng các phương thức vòng đời như componentDidMount(), componentDidUpdate(), và componentWillUnmount() để thực hiện các tác vụ như quản lý trạng thái và giao tiếp với API.
* Có thể sử dụng trạng thái (state) và các phương thức để quản lý trạng thái.

**Class Components:**

* Mạnh: Có thể sử dụng trạng thái và các phương thức vòng đời.
* Yếu: Cú pháp phức tạp hơn, code có thể dài và khó hiểu hơn.

**Functional Components:**

Mạnh: Cú pháp ngắn gọn hơn, dễ đọc và dễ hiểu hơn. Sử dụng Hooks giúp quản lý trạng thái và các phương thức vòng đời.

Yếu: Trước React 16.8, không thể sử dụng trạng thái và các phương thức vòng đời, nhưng với React Hooks, điều này đã được khắc phục.

1. **Hãy trình bày những giai đoạn có trong life cycle của một component trong React Native**

*Vòng đời (lifecycle) của một component trong React Native bao gồm 3 giai đoạn chính:*

**1. Giai đoạn khởi tạo (Mounting):**

* constructor(props):
  + Được gọi khi component được tạo.
  + Dùng để khởi tạo state và các biến cục bộ.
  + Lưu ý: Không nên gọi setState trong constructor.

**static getDerivedStateFromProps(nextProps, prevState):**

* + Được gọi trước khi render mỗi lần.
  + Dùng để cập nhật state dựa trên props mới.
  + Trả về null nếu không cần cập nhật state.

**render():**

* + Trả về React element mô tả giao diện người dùng của component.
  + Được gọi nhiều lần trong suốt vòng đời của component.

**componentDidMount():**

* + Được gọi sau khi render lần đầu tiên.
  + Dùng để thực hiện các tác vụ sau khi component được gắn vào DOM.
  + Ví dụ: lấy dữ liệu từ API, thiết lập lắng nghe sự kiện.

**2. Giai đoạn cập nhật (Updating):**

* **shouldComponentUpdate(nextProps, nextState):**
  + Được gọi trước khi render mỗi lần.
  + Cho phép component kiểm soát xem có cần cập nhật hay không.
  + Trả về true nếu component cần cập nhật, false nếu không.
* **static getDerivedStateFromProps(nextProps, prevState):**
  + Được gọi trước khi render mỗi lần.
  + Dùng để cập nhật state dựa trên props mới.
  + Trả về null nếu không cần cập nhật state.
* **render():**
  + Trả về React element mô tả giao diện người dùng của component.
  + Được gọi nhiều lần trong suốt vòng đời của component.
* **getSnapshotBeforeUpdate(prevProps, prevState):**
  + Được gọi ngay trước khi render trong quá trình cập nhật.
  + Dùng để lưu lại giá trị cũ của props và state trước khi cập nhật.
* **componentDidUpdate(prevProps, prevState, snapshot):**
  + Được gọi sau khi render trong quá trình cập nhật.
  + Dùng để thực hiện các tác vụ sau khi component được cập nhật.
  + Ví dụ: cập nhật DOM dựa trên state mới.

**3. Giai đoạn hủy bỏ (Unmounting):**

* **componentWillUnmount():**
  + Được gọi trước khi component bị hủy bỏ.
  + Dùng để thực hiện các tác vụ dọn dẹp trước khi component bị gỡ khỏi DOM.
  + Ví dụ: hủy đăng ký lắng nghe sự kiện, giải phóng bộ nhớ.

**Lưu ý:**

Vòng đời của component được gọi là "lifecycle" vì nó mô tả quá trình "sinh ra", "sống" và "chết" của một component.

Mỗi giai đoạn trong vòng đời có mục đích riêng và được gọi vào các thời điểm khác nhau.

Việc hiểu rõ vòng đời của component rất quan trọng để viết code React Native hiệu quả và dễ bảo trì.

1. **những điều kiện nào làm cho một component rerender trong React Native**

**Thay đổi props:**

* Khi props của component thay đổi, component sẽ render lại.
* Ví dụ: khi bạn truyền một giá trị mới cho prop của component, component sẽ render lại với giá trị mới.

**Thay đổi state:**

* Khi state của component thay đổi, component sẽ render lại.
* Ví dụ: khi bạn sử dụng setState để cập nhật state, component sẽ render lại với state mới.

**Gọi forceUpdate:**

* Gọi forceUpdate sẽ buộc component render lại ngay cả khi props và state không thay đổi.
* Sử dụng forceUpdate một cách cẩn thận vì nó có thể ảnh hưởng đến hiệu suất.

**Thay đổi context:**

* Khi context mà component sử dụng thay đổi, component sẽ render lại.
* Ví dụ: nếu bạn sử dụng useContext để truy cập context và context thay đổi, component sẽ render lại.

**Thay đổi cấu trúc DOM:**

* Nếu cấu trúc DOM của component thay đổi bên ngoài React, component sẽ render lại.
* Ví dụ: nếu bạn thay đổi DOM trực tiếp bằng JavaScript, component sẽ render lại.

**Sử dụng useEffect:**

* Khi callback của useEffect thay đổi, component sẽ render lại.
* Sử dụng useEffect một cách cẩn thận để tránh render lại quá nhiều.

**Sử dụng useMemo:**

* Khi giá trị trả về của useMemo thay đổi, component sẽ render lại.
* Sử dụng useMemo một cách cẩn thận để tránh render lại quá nhiều.

**4. Stateless Component và Stateful Component trong React Native**

**Stateless Component:**

Cấu trúc: Là hàm JavaScript đơn giản, nhận props và trả về JSX.

Trạng thái: Không có trạng thái nội bộ.

Vòng đời: Không có các phương thức vòng đời.

Hiệu suất: Nhanh hơn vì không cần cập nhật trạng thái.

**Ví dụ:**

const MyComponent = ({ name }) => {

return <h1>Hello, {name}!</h1>;

};

**Stateful Component:**

Cấu trúc: Là class kế thừa từ React.Component.

Trạng thái: Có thể quản lý trạng thái nội bộ bằng this.state.

Vòng đời: Có các phương thức vòng đời như componentDidMount, componentDidUpdate.

Hiệu suất: Chậm hơn vì cần cập nhật trạng thái.

Ví dụ:

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

render() {

return (

<div>

<h1>Count: {this.state.count}</h1>

<button onClick={() => this.setState({ count: this.state.count + 1 })}>

Increase

</button>

</div>

);

}

}

**Lựa chọn:**

* **Stateless Component:** Nên sử dụng cho các component đơn giản, không cần quản lý trạng thái.
* **Stateful Component:** Nên sử dụng cho các component phức tạp, cần quản lý trạng thái, tương tác với người dùng.

**5. React hook là gì? hãy kể tên những hook trong React Native**

React Hook là một tính năng mới trong React cho phép bạn sử dụng trạng thái và các tính năng React khác mà không cần sử dụng các lớp (class components). Hook giúp bạn tái sử dụng logic stateful giữa các component một cách dễ dàng hơn và làm cho mã của bạn trở nên dễ hiểu hơn.

**Dưới đây là một số hook phổ biến trong React Native:**

* **useState:** Hook này cho phép bạn sử dụng trạng thái trong các functional components. Nó cung cấp một cách để khai báo biến trạng thái và cập nhật giá trị của nó.
* **useEffect:** Hook này cho phép bạn thực hiện các hành động sau mỗi lần render hoặc sau khi trạng thái thay đổi. Điều này có thể là việc gọi API, cập nhật trạng thái, hoặc thực hiện các tác vụ hiệu suất.
* **useContext:** Hook này cho phép bạn truy cập vào Context trong React. Nó giúp bạn truy cập vào dữ liệu toàn cục một cách dễ dàng từ bất kỳ component nào trong ứng dụng của bạn.
* **useReducer:** Hook này cho phép bạn quản lý trạng thái của một component bằng cách sử dụng một hàm reducer, tương tự như trong Redux.
* **useCallback:** Hook này giúp bạn tránh việc tạo lại hàm callback mới sau mỗi lần render. Điều này hữu ích khi bạn truyền các hàm callback vào các component con mà không muốn chúng render lại mỗi khi component cha render lại.
* **useMemo**: Hook này giúp bạn tối ưu hóa hiệu suất bằng cách memoize các giá trị tính toán. Nó sẽ tính toán giá trị chỉ khi các dependency đã thay đổi.
* **useRef:** Hook này cho phép bạn lưu trữ các tham chiếu đến các phần tử DOM hoặc các giá trị khác trong component mà không gây ra việc render lại khi giá trị thay đổi.
* **useImperativeHandle:** Hook này cho phép bạn tùy chỉnh giao diện cho các thực thể được tạo ra bởi useRef. Nó thường được sử dụng trong thư viện UI.
* Những hook này cung cấp các cơ chế khác nhau để quản lý trạng thái, hiệu suất và logic trong ứng dụng React Native của bạn một cách linh hoạt và mạnh mẽ.

**6. Làm sao để thay đổi giá trị của state mà không là component rerender trong React Native.**

Có một số cách để thay đổi giá trị của state mà không làm component React Native render lại:

* **Sử dụng** setState **với callback:**

JavaScript

const MyComponent = () => {

const [count, setCount] = useState(0);

const handleClick = () => {

setCount(prevCount => prevCount + 1);

};

return (

<div>

<h1>Count: {count}</h1>

<button onClick={handleClick}>Increase</button>

</div>

);

};

* **Sử dụng** useMemo**:**

JavaScript

const MyComponent = () => {

const [count, setCount] = useState(0);

const memoizedCount = useMemo(() => count \* 2, [count]);

return (

<div>

<h1>Count: {memoizedCount}</h1>

<button onClick={() => setCount(count + 1)}>Increase</button>

</div>

);

};

* **Sử dụng** useEffect**:**

JavaScript

const MyComponent = () => {

const [count, setCount] = useState(0);

useEffect(() => {

// Do something with count that doesn't cause a re-render

}, [count]);

return (

<div>

<h1>Count: {count}</h1>

<button onClick={() => setCount(count + 1)}>Increase</button>

</div>

);

};

* **Sử dụng** shouldComponentUpdate**:**

JavaScript

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

shouldComponentUpdate(nextProps, nextState) {

return this.state.count !== nextState.count;

}

render() {

return (

<div>

<h1>Count: {this.state.count}</h1>

<button onClick={() => this.setState({ count: this.state.count + 1 })}>

Increase

</button>

</div>

);

}

}

**7. Khi setState, giá trị của state có được cập nhật ngay lập tức hay không**

Khi sử dụng setState trong React Native, giá trị của state không được cập nhật ngay lập tức. Thay vào đó, nó được lên lịch cập nhật trong lần render tiếp theo của component.

**Lý do:**

React sử dụng một cơ chế gọi là "batching" để cập nhật state.

Batching giúp cải thiện hiệu suất bằng cách giảm thiểu số lần render component.

Khi bạn gọi setState, React sẽ thêm thay đổi state vào một hàng đợi.

Sau khi tất cả các thay đổi state được thêm vào hàng đợi, React sẽ thực hiện một lần render duy nhất để cập nhật tất cả các thay đổi.

**Lưu ý:**

* Việc cập nhật state không đồng bộ có thể gây ra một số vấn đề.
* Ví dụ, nếu bạn sử dụng giá trị state trong một hiệu ứng (useEffect), giá trị state có thể chưa được cập nhật khi hiệu ứng được thực thi.
* Để giải quyết vấn đề này, bạn có thể sử dụng callback của setState.

**Callback của** setState**:**

* Callback của setState được gọi sau khi state được cập nhật.

## 6. Ref là gì ? Sự khác nhau giữ ref và state trong React Native

## 7. Có bao nhiêu kiểu Ref trong React Native

## 8. Làm sao để component cha có thể gọi hàm của component con?

## 9. useEffect là gì.

## 10. Có những cách dùng UseEffect nào?

## 11. Cần lưu ý gì sử dụng dependencies ?

## 12 . Cơ chế so sánh các dependencies của useEffect là gì?

## 13. Side Effect là gì ? Kể tên một số side effect bạn thường xử lý?

## 14. Clean up function của useEffect có chức năng gì ? Khi nào cần sử dụng ? Ví dụ.

## 15. useCallback và useMemo là gì ? Giống và khác nhau thế nào ?

## 16. Custom hooks là gì ? Hãy kể tên những custom hooks bạn thường dùng ?

## 17. React – router – dom là gì ? Tại sao phải sử dụng react – router – dom?

## 18. Nested routing là gì ? Hãy mô tả cách triển khai.

## 19. Redux là gì ? Mô hình hoạt động của Redux như thế nào?

## 20. Sự khác nhau giữa Action và Reducer là gì?

## 21. Làm sao để dispatch một Action.

## 22. MiddleWare là gì ? Khi nào cần sử dụng middleware ?

## 23. Làm sao để xuất dữ liệu từ Store.

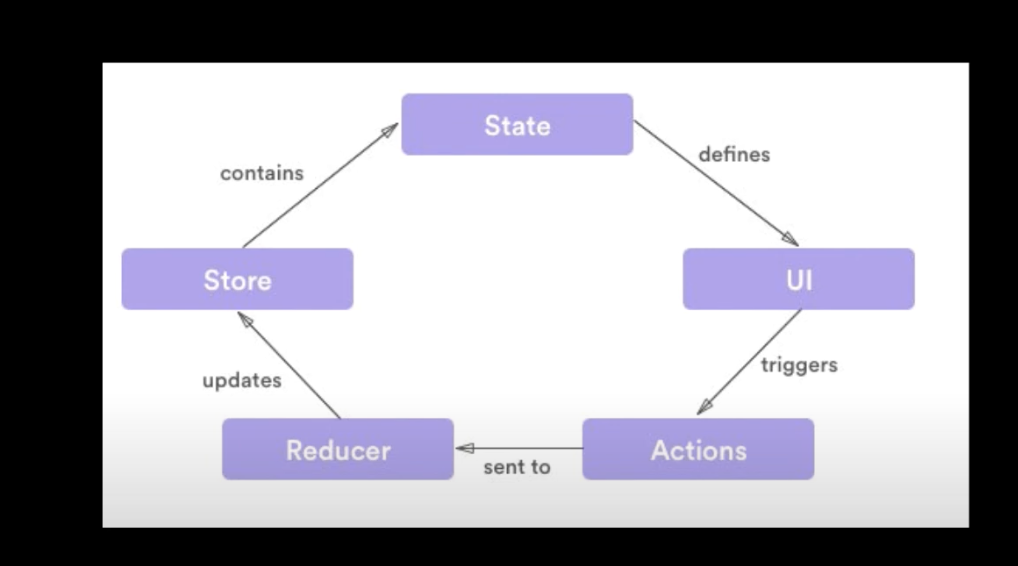
## 24. Context API là gì ?

## 25. Cách thức hoạt động của Context API như thế nào?

## 26. Context API có thể thay thế redux được không ?

## 27. So sánh context api và redux.

**Khái niệm Redux:**

****

**Các bước chạy của Redux**

**UI:** giao diện người dùng **Trigger:** Event của User thực hiện sent action đến Reducer.

**Reducer:** thực hiện xử lý action (nhận biết vai trò của action được truyền tới, state trả về là cái gì) sau đó Update lên Store

**Store:** một trong những thành phần chính của Redux, lưu giữ State và trả về cho UI

State difines UI: State đổi thì UI đổi theo.