**REACTJS Questions**

Table of Contents

**[1.](#_Toc201094528)****[What is Reactjs?](#_Toc201094528)** [2](#_Toc201094528)

[**2.** **Library for React Components** 2](#_Toc201094529)

[**3.** **What is Virtual DOM** 2](#_Toc201094530)

[**4.** **Props / Properties vs State** 2](#_Toc201094531)

[**5.** **Lifting State Up in React** 3](#_Toc201094532)

[**6.** **different phases of component lifecycle** 3](#_Toc201094533)

[**7.** **lifecycle methods of React** 3](#_Toc201094534)

[**8.** **context API** 4](#_Toc201094535)

[**9.** **using super constructor with props argument** 4](#_Toc201094536)

[**10.** **portals in React** 4](#_Toc201094537)

[**11.** **error boundaries in React v16** 4](#_Toc201094538)

[**12.** **What will happen if you use props in initial state** 5](#_Toc201094539)

[**13.** **Stale Closure** 5](#_Toc201094540)

[**14.** **onScroll của ReactJS** 6](#_Toc201094541)

[**15.** **Design Pattern in React** 7](#_Toc201094542)

[**16.** **Context API vs Redux** 7](#_Toc201094543)

[**17.** **Tại sao gọi reactjs là one-way-binding** 7](#_Toc201094544)

[**18.** **Key concepts trong Reactjs** 8](#_Toc201094545)

[**19.** **Tại sao gọi Reactjs là Library mà ko phải Framework** 8](#_Toc201094546)

[**20.** **Redux** 8](#_Toc201094547)

1. **What is Reactjs?**

* Reactjs is a library for building User Interface. It helps us to build Single Page Application (SPA) It created by Facebook (now is Meta). Reactjs is a component-base Architecture, so we can build reusable components

1. **Library for React Components**

* Selectable items for React
* React Target Time
* React Selections
* React Available Times

1. **What is Virtual DOM**

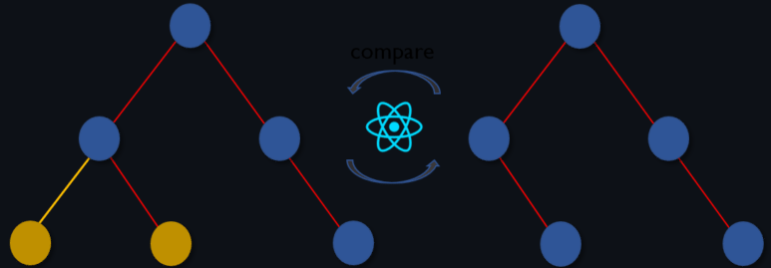
An in-memory representation of Real DOM. Giữ UI trong phần memory and dc đồng bộ với real DOM.

The Virtual DOM works in 3 steps.

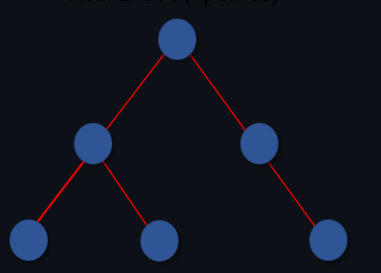
1. Whenever any underlying **data changes**, the **entire UI is re-rendered in Virtual DOM** representation.



1. Then compare the differences of nodes



1. Once the calculations are done, the real DOM will be updated with only the things that have actually changed.



1. **Props / Properties vs State**

|  |  |
| --- | --- |
| Props | State |
| Read-only | Can be changed |
| immutable | mutable |
| Chuyền data từ component này sang component khác | Chỉ tồn tại trong component đó |
| Được controlled bởi component bên ngoài | Chỉ dc controlled trong component đó |

1. **Lifting State Up in React**

Sharing state by moving state up to the closest parent component, if 2 child-components are using a single state, moving that state up to the parent of those.

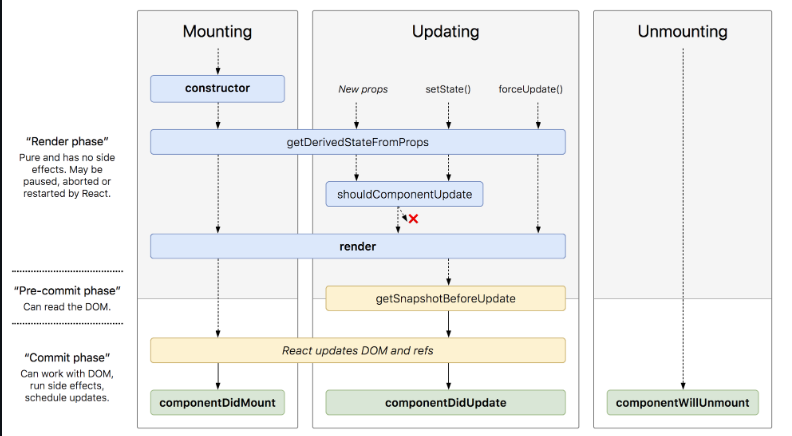
1. **different phases of component lifecycle**

Mounting:

* mount component to browser DOM. **constructor(), getDerivedStateFromProps(), render(),** and **componentDidMount()** lifecycle methods.
* Nếu data thay đổi mà component vẫn đang hiện thì sẽ ko trigger những hàm trên

Updating:

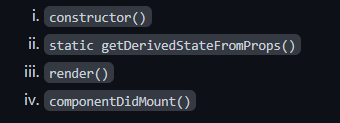
* take new props or updating state (setState). **getDerivedStateFromProps(), shouldComponentUpdate(), render(), getSnapshotBeforeUpdate()** and **componentDidUpdate()** lifecycle methods.
* ở giai đoạn change DOM sẽ có 3 bước:
  + Render: no side effect. (nghĩa là sẽ chưa apply vào DOM). Chỉ chạy hàm **render**()
  + Pre-commit: trước khi apply vào DOM, đây là lúc React đọc DOM thông qua hàm **getSnapshotBeforeUpdate()**
  + Commit: apply vào DOM, và chạy lifecycle methods **componentDidMount()** for mounting, **componentDidUpdate()** for updating, and **componentWillUnmount()**



Unmounting:

* deleted or removed from browser DOM. **componentWillUnmount()** lifecycle method

Lifecycle methods order.



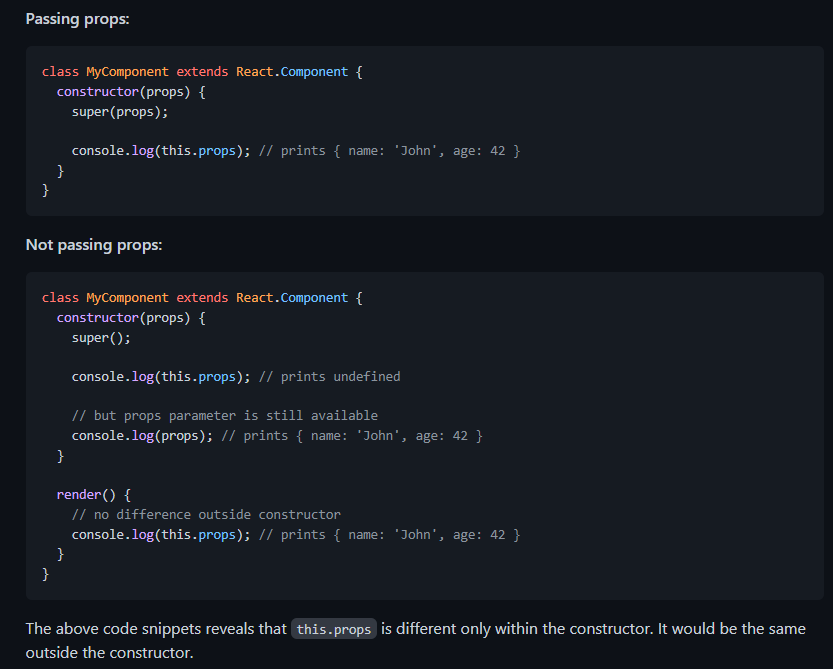
1. **lifecycle methods of React**

* **componentWillMount()**: chạy trước khi rendering. Hàm này **chỉ nên dùng** ở tầng App level để thêm config thôi. (hoặc ko dùng)
* **componentDidMount()**: chạy sau khi render lần đầu tiên của component. Dùng để call API, AJAX, addEventLisners
* **componentWillReceiveProps(nextProps)**: trigger khi props thay đổi. Dùng trong trường hợp set props vào state (state transitions)
* **shouldComponentUpdate(nextProps, nextState)**: **true**. Xét xem có re-render ko
* **componentDidUpdate(prevProps, prevState, snapshot)**: trigger ngay khi updating xảy ra.
* **componentWillUnmount()**: khi delete/remove component khỏi DOM. (nếu component bị **display:none** hay bị hide đi nhưng vẫn nằm trên DOM thì ko trigger hàm này )

1. **context API**

* context provide a way to pass data without passing props down through many levels of components

1. **using super constructor with props argument**

child class constructor cannot use this until super() has been called.(same as ES6 sub-class). 

1. **portals in React**

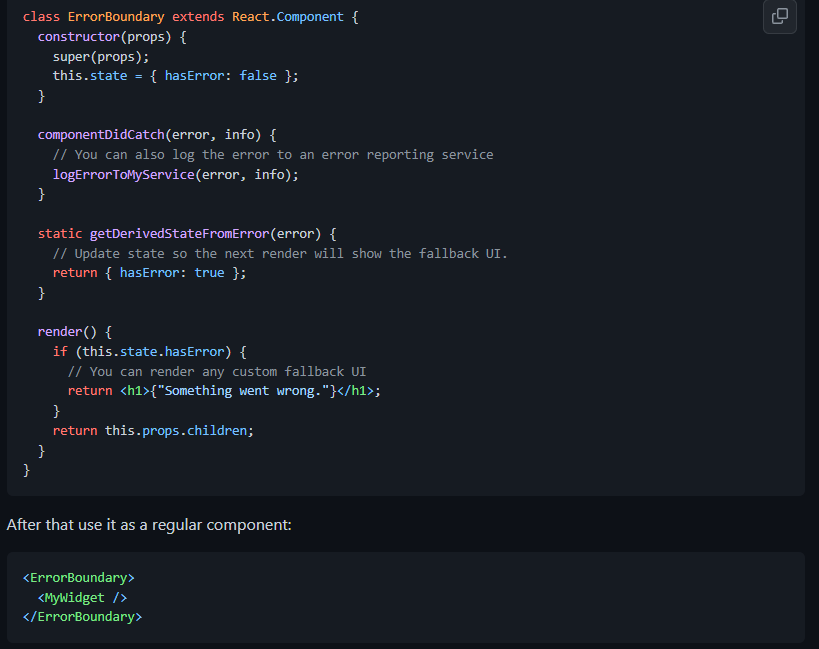
render childrent into DOM node that is outside of parent component

advantages and disadvantages of ReactJS

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| Increase performance with Visual DOM | Complex library, need to know many concepts |
| JSX apply html and javascript | Since it is a library, Not a framework. There are many styles, many structures |

1. **error boundaries in React v16**

* là component để hứng lỗi nếu component gãy, và show fallback UI thay cho việc trắng trang
* **componentDidCatch(error, info)** or **static getDerivedStateFromError()**



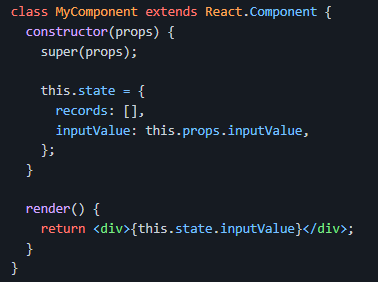
* nên để ở tầng component ngoài cùng để bắt lỗi

ko dùng call API, add listener trong componentWillMount()

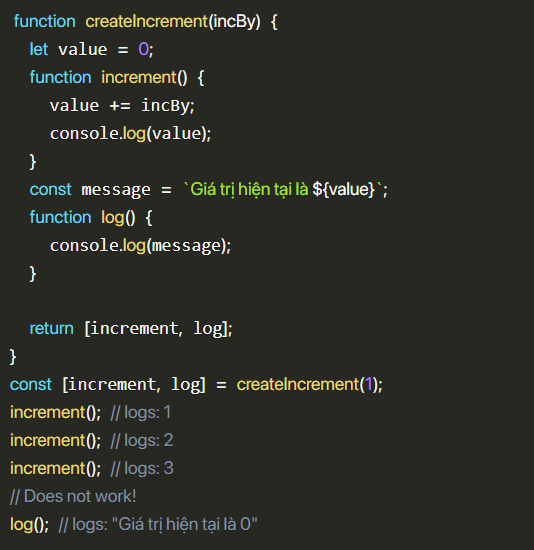
* do componentWillMount() chạy trước render()
* ko re-render
* ko thấy dc cây DOM

1. **What will happen if you use props in initial state**

Props are chaged and component not refresh, new props will not apply.



1. **Stale Closure**

 Ở đây khi gọi hàm **log(),** mong muốn của mình sẽ log ra giá trị là 3, nhưng giá trị lại là 0. Đúng bằng **value = 0** ban đầu.

Lỗi xảy ra do message được tạo ra khi gọi hàm **createIncrement** và không được cập nhật mỗi lần **increment**

**Stale closure** là một biến trong closure bị **outdate**

How to fix



Di chuyển code khởi tạo **message** vào trong hàm **log**(), mỗi khi gọi hàm **log** thì **message** sẽ được tạo lại và đảm bảo không bị outdate

1. **onScroll của ReactJS**

trong các thẻ div có hàm onScroll của ReactJS. Nên ko cần dùng addEventListener cho ref nữa.

Note:

Khi chuyển tab khác thì nếu thanh scroll đang ở giữa div thì cũng sẽ trigger hàm onScroll này vì scrollHeight === offsetHeight khi re-render lại div . Fix: scrollTop > 0

const onScroll = (*e*: React.UIEvent<HTMLDivElement>) => {

        if (isLoadingMore || !isAllowLoadMore) return

        let scrollTop = *e*.currentTarget.scrollTop,

            scrollHeight = *e*.currentTarget.scrollHeight,

            offsetHeight = *e*.currentTarget.offsetHeight;

        if (scrollTop > 0 && (scrollTop + offsetHeight >= scrollHeight - 50)) {

            onLoadMore();

        }

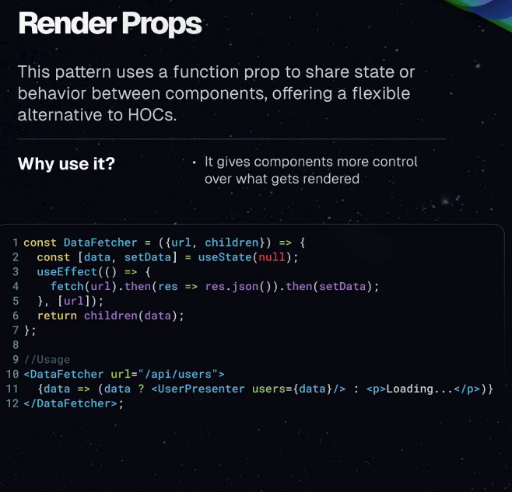
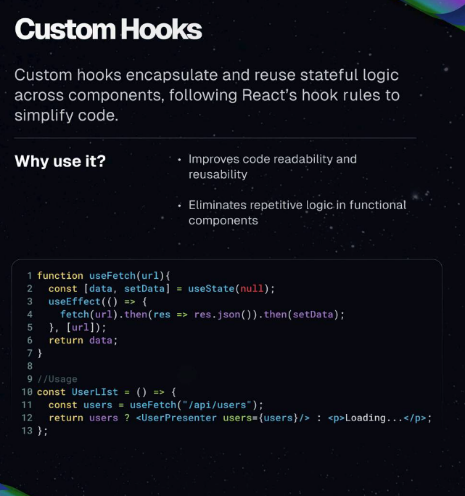
    }

return <div className={`exticket-feedback-list ticket-tabs\_\_body`} onScroll={onScroll}>

.....

    </div>

1. **Design Pattern in React**



1. **Context API vs Redux**

|  |  |
| --- | --- |
| **Context API** | **Redux** |
| - Built in feature in React  - to share data globally to other components  - avoid props drilling from parent components to many nested components | - a library for state management  - to store and share data to whole application  - more complex than context API, but it has debugger tool for Redux,  - Redux has middleware ecosystem handle **asynchronous tasks** or side effects function (API call) with redux-thunk  - data persistency  - Redux is suitable for large-scale or complex data, because  - 1 way flow so easy to debug. |

1. **Tại sao gọi reactjs là one-way-binding**

Vì trong phần UI (phần HTML ) của react ko trực tiếp làm thay đổi state. Thay vào đó, React có thể dùng prop để show data và gắn 1 cái event handler dưới dạng props để thay đổi data

1. **Key concepts trong Reactjs**

|  |  |
| --- | --- |
| Key concepts | Công dụng |
| **component-base library** | * Xây dựng các components để dễ dàng tái sử dụng * Là component-base library. Các components này sẽ có state và cách render components sẽ hoàn toàn độc lập |
| **Virtual DOM** | * Để kiểm tra nếu có node nào khác nhau thì sẽ update node đó |
| **State management** | * Có context API để quản lý state toàn cục |
| **JSX (JavaScript XML)** | * Cho phép viết code JS trong HTML để làm code ngắn gọn và dễ đọc |

1. **Tại sao gọi Reactjs là Library mà ko phải Framework**

* Reactjs **ko có architecture cụ thể**, nên có tính linh hoạt cao. Ko yêu cầu phải theo chuẩn kiến trúc nào,
* **Ko cần phải xây dựng theo cây thư mục** nào cả
* Ko có **handling routing**. Phải dùng thêm **react-router**
* **Ko có hệ thống state management** rõ ràng. **Context API** chỉ hỗ trợ share data cho các nested component. Ko có reducers. Ko có debugger tool like Redux, thiếu middleware để xử lý data.

1. **Redux**

Thành phần của Redux

Store: save/ keep the state of redux

Reducer: Reducer là 1 hàm nhận state ban đầu và 1 action

Action: 1 object luôn có thuộc tính type để xác định 1 hàm động cụ thể

Dispatch: gửi 1 action(object) đến reducer. Thường store.dispatch action

Có redux devtool, 1 way binding vì luôn đi theo flow

Flow: UI => dispatch action => reducer => return new state => state dc lưu vào store => render UI