

Theory 07

React Hooks, React Router

WEEK 02

Nội dung chính

- ❑ TỔNG QUAN VỀ REACT HOOK
- ❑ CÁC HOOK CĂN BẢN
- ❑ CÁC HOOK BỔ SUNG
- ❑ CÁC HOOK TÙY CHỈNH
- ❑ CƠ CHẾ ROUTING
- ❑ CẤU HÌNH REACT ROUTER

03

06

13

23

26

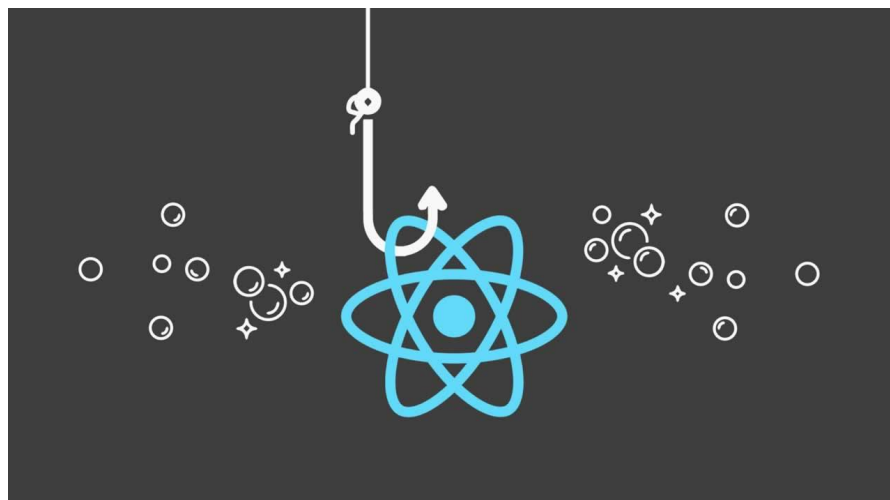
29

AGENDA



Tổng quan về React Hook

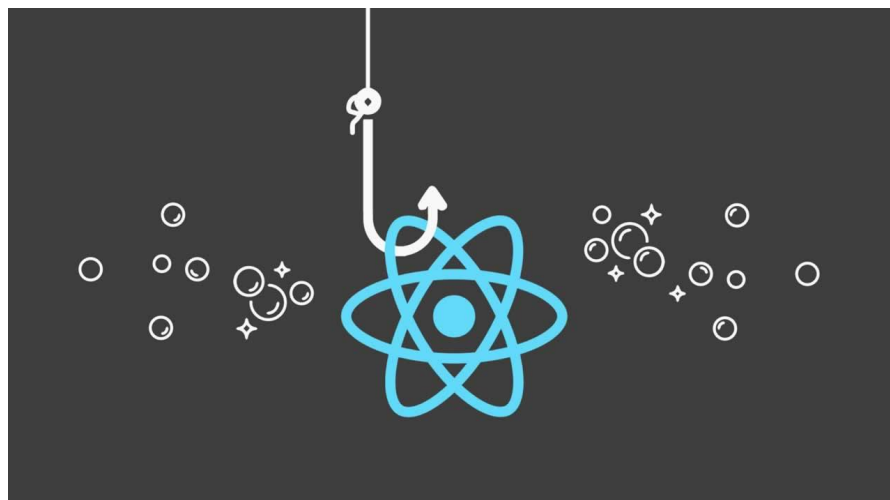
- ❑ Sau một thời gian làm việc với **Class Component** trong React, có lẽ chúng ta sẽ bắt gặp một trong số các vấn đề sau:
 - Các **Component** quá lớn, sẽ có thể bị “**Wrapper hell**” - các component được lồng (nested) vào nhau nhiều tạo ra một **DOM** tree phức tạp.
 - Sự rắc rối của **life-cycle** trong **Class**



Tổng quan về React Hook

□ Hook là gì?

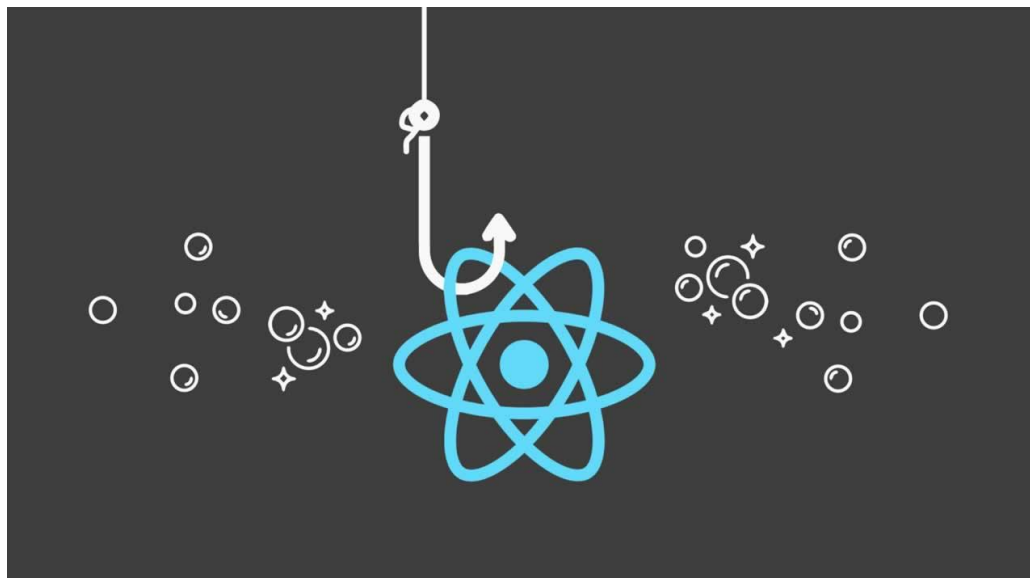
- **Hook** là một tính năng bổ sung mới trong React 16.8.
- **Hook** là những hàm cho phép kết nối React **state** và **life-cycle** vào các **Component** sử dụng dạng **Function** (Functional Component)
- Bạn có thể sử dụng **state** và **life-cycles** mà không cần dùng **ES6 class**



Tổng quan về React Hook

❑ Lợi ích của Hook:

- Khiến các component trở nên gọn nhẹ hơn.
- Giảm đáng kể số lượng code, dễ tiếp cận hơn
- Cho phép sử dụng state ngay trong Functional Component



Các Hook căn bản

❑ useState Hook:

- Là một hook căn bản của ReactJS phiên bản > 16.8.
- Giúp chúng ta có thể sử dụng state trong Functional Component
- **Input:** **initialState** (**value** hoặc **function**)
- **Output:** [**state**, **setState**] (giá trị của **state** và hàm **cập nhật state**)



React **Hooks**
useState

Các Hook căn bản

□ useState Hook:

- Cú pháp: `const [username, setUsername] = useState("default value");`
- Trong đó:
 - **username**: **state** muốn khởi tạo
 - **default username value**: giá trị khởi tạo ban đầu cho **state**
 - **setUsername**: hàm để cập nhật giá trị của **state username**



React Hooks
useState

Các Hook căn bản

□ useState Hook:

○ Ví dụ:

```
JS App.js x
1 import React, { useState } from "react";
2
3 export default function Counter() {
4   const [count, setCount] = useState(0);
5   return (
6     <>
7     Count: {count}
8     <button onClick={() => setCount(1)}>Reset</button>
9     <button onClick={() => setCount((prevCount) => prevCount - 1)}>-</button>
10    <button onClick={() => setCount((prevCount) => prevCount + 1)}>+</button>
11    </>
12  );
13 }
```


Các Hook căn bản

□ **useEffect** Hook:

- Là hook giúp bạn có thể làm việc với **life-cycle** trong **Functional Component**.
- **useEffect** hook là **sự kết hợp giữa 3 phương thức** **componentDidMount**, **componentDidUpdate** và **componentWillUnmount** lại với nhau
- **useEffect** cho phép xử lý các logic trong các vòng đời của component và được gọi mỗi khi có bất cứ sự thay đổi nào trong một component



Các Hook căn bản

❑ **useEffect** Hook:

- Cú pháp: **useEffect**(effectFunction, arrayDependencies)
- Trong đó:
 - **effectFunction**: được gọi lại mỗi khi có giá trị mảng **arrayDependencies** thay đổi hoặc **arrayDependencies** rỗng thì khi bất kỳ **state** nào thay đổi.
 - **arrayDependencies**: mảng các **state** chúng ta muốn quan sát sự thay đổi.



Các Hook căn bản

❑ **useEffect** Hook:

- **useEffect** hook sẽ được gọi 2 lần
 - 1 lần khi **render** component.
 - 1 lần khi state trong mảng **arrayDependencies** thay đổi.



React **Hooks**
useState

Các Hook căn bản

□ useEffect Hook:

○ Ví dụ:

```
JS App.js x
1 import React, { useState, useEffect } from "react";
2
3 export default function App() {
4   const [count, setCount] = useState(0);
5   const [something, setSomething] = useState(false);
6   const handleClick = () => setCount(count + 1);
7
8   useEffect(() => {
9     console.log("Watch Here");
10    document.title = "Count is: " + count;
11  });
12
13  return (
14    <div>
15      <p>Title page will be change when we click the button</p>
16      <button onClick={handleClick}>Increment Count</button>{" "}
17      <button onClick={() => setSomething(!something)}>
18        Change something{" "}
19      </button>
20    </div>
21  );
22 }
```

Các Hook bổ sung

❑ useContext Hook:

- Là **hook** giúp tạo các biến toàn cục có thể truyền **props** qua các **Component**.
- **useContext** là phương pháp thay thế cho “prop drilling” (truyền props từ ông nội sang cha và sang con)
- **useContext** đơn giản, nhẹ nhàng hơn thay vì sử dụng **Redux** quản lý **state**.



React **Hooks**
useState

Các Hook bổ sung

□ useContext Hook:

- Cú pháp: **const value = useContext(ExampleContext);**
- Trong đó:
 - **ExampleContext**: **Context** được tạo ra từ hàm **createContext**
 - **useContext**: **hook** giúp truy xuất các **value** đã được định nghĩa trong **Context**
 - **value**: **value** được lấy ra từ **Context**

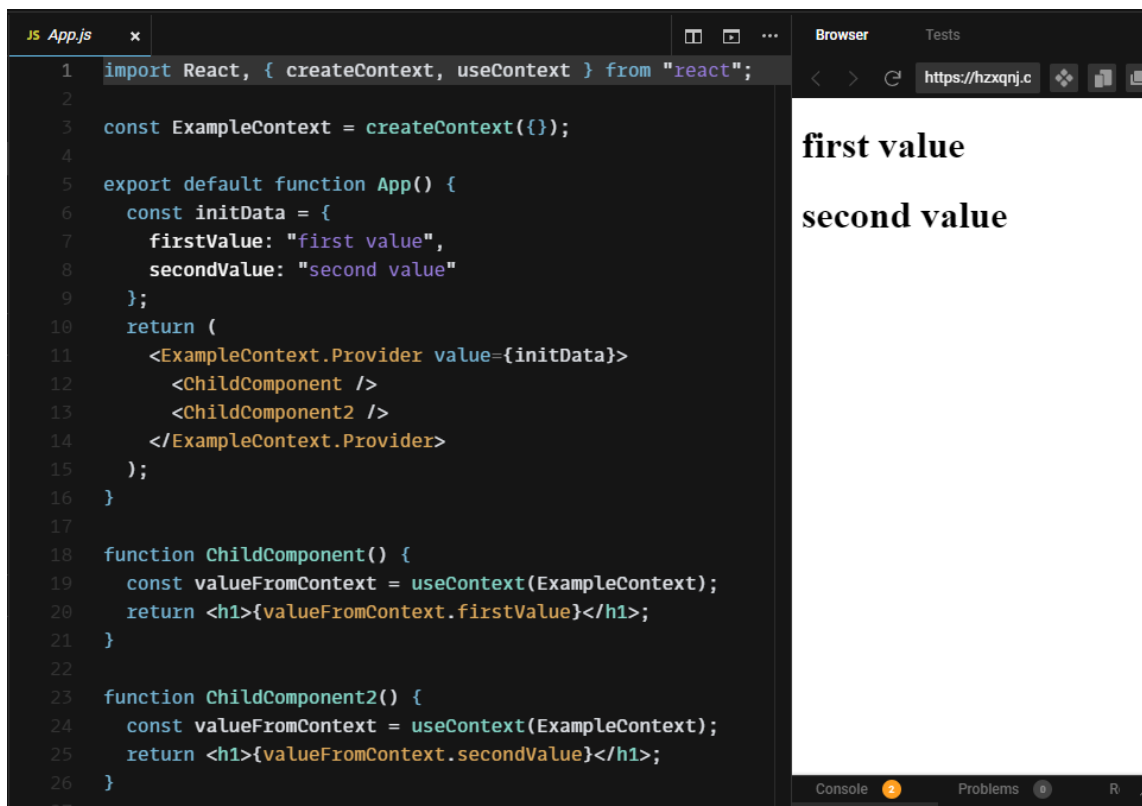


React **Hooks**
useState

Các Hook bổ sung

□ useContext Hook:

○ Ví dụ:



```
1 import React, { createContext, useContext } from "react";
2
3 const ExampleContext = createContext({});
4
5 export default function App() {
6   const initData = {
7     firstValue: "first value",
8     secondValue: "second value"
9   };
10  return (
11    <ExampleContext.Provider value={initData}>
12      <ChildComponent />
13      <ChildComponent2 />
14    </ExampleContext.Provider>
15  );
16 }
17
18 function ChildComponent() {
19   const valueFromContext = useContext(ExampleContext);
20   return <h1>{valueFromContext.firstValue}</h1>;
21 }
22
23 function ChildComponent2() {
24   const valueFromContext = useContext(ExampleContext);
25   return <h1>{valueFromContext.secondValue}</h1>;
26 }
```

first value
second value

Các Hook bổ sung

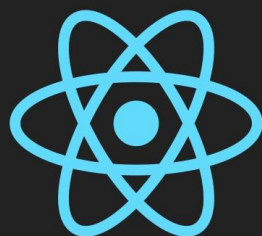
□ useContext Hook:

- Ví dụ (giải thích)
- Tạo **Provider** với giá trị là **Initial Value** bằng **createContext**("Init Value")
 - **Provider**: thành phần cung cấp các giá trị (**props**)
 - **createContext**: tạo **Context** cho **Component**
- Gọi **Context** (**ExampleContext**): **<ExampleContext.Provider value={initData}>**
 - **ExampleContext.Provider**: **Provider** được tạo ra từ **Context** đã tạo
 - **ProviderContext**: **Provider** luôn cần tồn tại như một trình bao bọc xung quanh phần tử cha, bất kể bên trong có các giá trị như nào
- Sử dụng **useContext** để truy xuất **props** đã tạo ở **Context** với cú pháp như sau:
const valueFromContext = useContext(ExampleContext);
- **value={initData}**: **props** tự động được define ở các **Component** con

Các Hook bổ sung

□ useMemo Hook:

- Là **hook** cho phép bạn cache lại kết quả tính toán giữa các lần **render** của **component** bằng cách “ghi nhớ” lại giá trị của lần render trước
- **useMemo** giúp ta kiểm soát việc được render dư thừa của các component con
- Bằng cách truyền vào 1 tham số thứ 2 vào **Component** thì chỉ khi tham số này thay đổi thì **useMemo** mới được thực thi



useMemo?

Các Hook bổ sung

□ **useMemo Hook:**

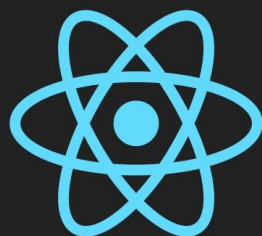
○ Cú pháp:

const MemoizedComp = **useMemo**() => <ChildComp **text**=**{text}**/>, [**text**]);

○ Trong đó:

➤ **useMemo**: ghi nhớ giá trị lần render trước của **Component**.

➤ **text**: giá trị của lần render trước cần ghi nhớ



useMemo?

Các Hook bổ sung

□ useMemo Hook:

○ Ví dụ:

```
JS UseMemo.js x
1 import React, { useState, useMemo } from "react";
2
3 const UseMemoComponent = () => {
4   const [text, setText] = useState("Hello!");
5
6   const ChildComponent = ({ text }) => {
7     console.log("rendered again!");
8     return <div>{text}</div>;
9   };
10
11   const MemoizedComponent = useMemo(() => <ChildComponent text={text} />, [
12     text
13   ]);
14
15   return (
16     <div>
17       <button onClick={() => setText("Hello!")}>Hello! </button>
18       <button onClick={() => setText("Hola!")}>Hola!</button>
19       {MemoizedComponent}
20     </div>
21   );
22 };
23
24 export default UseMemoComponent;
```

Các Hook bổ sung

❑ **useCallback Hook:**

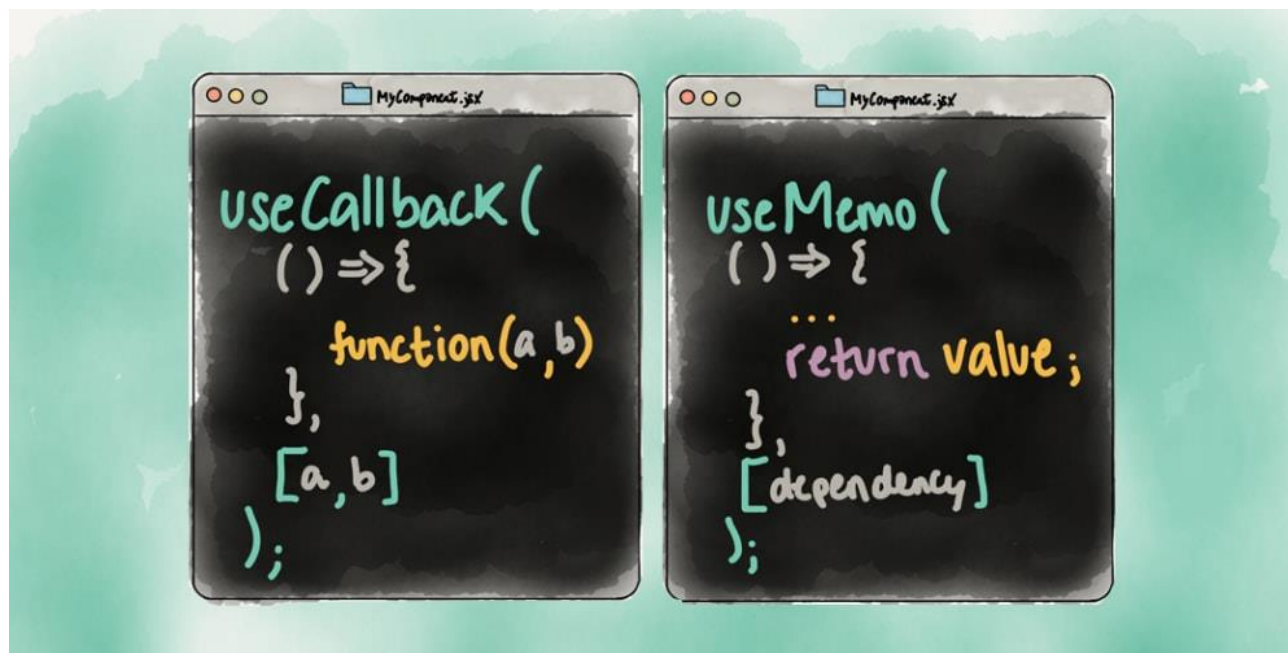
- Là **hook** trả về một function và một array chứa các dependencies (những biến số được truyền vào từ bên ngoài mà function này phụ thuộc khi chạy)
- **useCallback** sử dụng cơ chế memorization – ghi nhớ kết quả của một function vào trong memory và sẽ trả về function được ghi nhớ trong trường hợp các dependencies này không thay đổi.



Các Hook bổ sung

❑ useCallback Hook:

○ **useCallback** có nhiệm vụ tương tự như **useMemo** nhưng khác ở chỗ function truyền vào **useMemo** **bắt buộc phải ở trong quá trình render**



Các Hook bổ sung

□ useCallback Hook:

○ Ví dụ:

```
JS UseCallback.js x
1 import React, { useState, useEffect, useCallback } from "react";
2
3 const ChildComponent = ({ loggingStatus }) => {
4   useEffect(() => {
5     loggingStatus();
6   }, [loggingStatus]);
7   return <div />;
8 };
9
10 const UseMemoComponent = () => {
11   const [count, setCount] = useState(0);
12   const loggingStatus = useCallback(() => {
13     console.log("Run from ChildComponent");
14   }, []);
15   const addMore = () => {
16     setCount((prev) => prev + 1);
17   };
18
19   return (
20     <div>
21       <p>Current: {count}</p>
22       <ChildComponent loggingStatus={loggingStatus} />
23       <button onClick={addMore}>Click</button>
24     </div>
25   );
26 };
27
28 export default UseMemoComponent;
```

Các Hook tùy chỉnh

- ❑ **Custom Hooks** là những hooks mà do lập trình viên tự định nghĩa với mục đích thực hiện chức năng nào đó, được sử dụng để chia sẻ logic giữa các **components**
- ❑ React đã định nghĩa sẵn các hooks như **useState, useEffect, useContext,...**
- ❑ Khi đặt tên một **custom hooks** phải có từ khóa **use** ở đầu, ví dụ như: `useClick()`, `useClock()`, `useQuery()`



Các Hook tùy chỉnh

- ❑ Ví dụ (không dùng Custom Hook)
- ❑ Và bây giờ nếu bạn muốn dùng **window width** ở component khác thì phải lặp lại phần code trên. Đây là lúc custom hooks phát huy tác dụng

```
import { useState, useEffect } from 'react'
import Sidebar from 'components/Sidebar'

const App = () => {
  const [width, setWidth] = useState<number>(window.innerWidth)

  useEffect(() => {
    const handler = () => {
      setWidth(window.innerWidth)
    }

    window.addEventListener('resize', handler)

    return () => {
      window.removeEventListener('resize', handler)
    }
  }, [])

  return (
    <>
      {width >= 1024 && <Sidebar />}
    </>
  )
}
```


Các Hook tùy chỉnh

- ❑ Ví dụ (có dùng Custom Hook)
- ❑ Tạo ra hook **useWindowSize** để giải quyết vấn đề lặp code ở các **component** bên trên

```
import { useWindowSize } from 'hooks'

const App = () => {
  const { width, height } = useWindowSize()

  return (
    <>
      {width >= 1024 && <Sidebar />}
    </>
  )
}
```

```
import { useState, useEffect } from 'react'

export const useWindowSize = () => {
  const [windowSize, setWindowSize] = useState({
    width: window.innerWidth,
    height: window.innerHeight,
  })

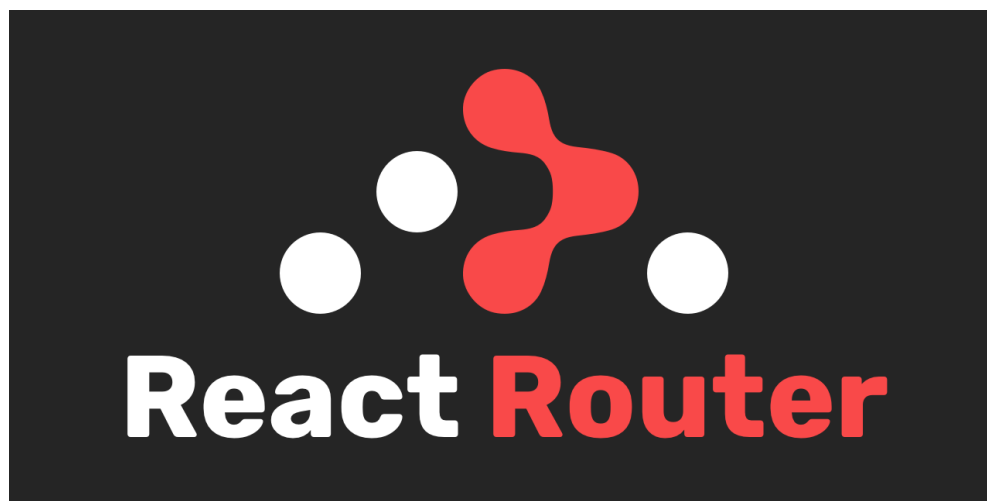
  useEffect(() => {
    const handler = () => {
      setWindowSize({
        width: window.innerWidth,
        height: window.innerHeight,
      })
    }
    window.addEventListener('resize', handler)

    return () => {
      window.removeEventListener('resize', handler)
    }
  }, [])

  return windowSize
}
```

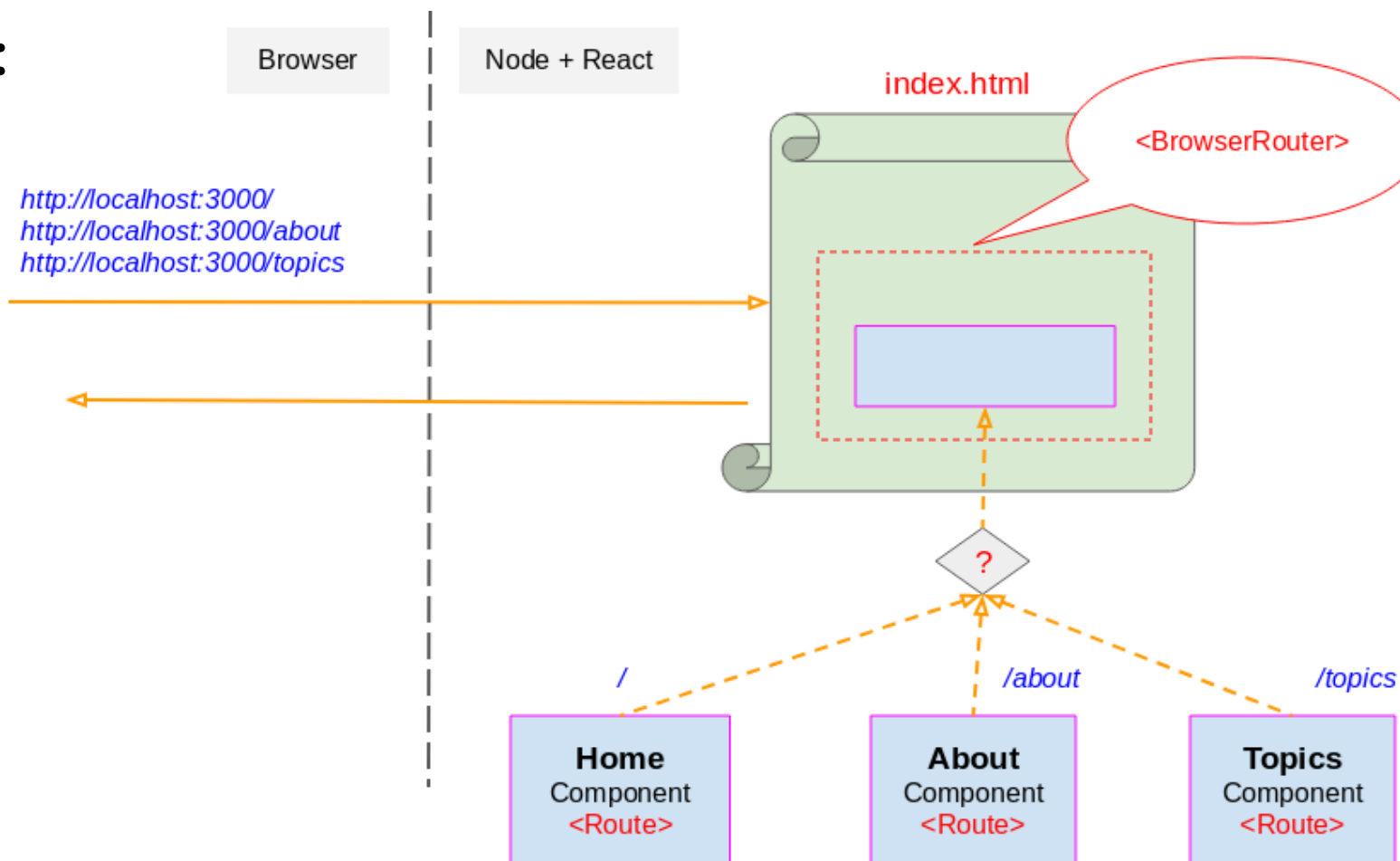
Cơ chế Routing

- ❑ Với **HTML**, khi điều hướng từ trang này sang trang khác, ta sẽ sử dụng thẻ **<a>**
- ❑ Trong **React JS** thường xây dựng những **Single Page Application** (SPA) khi chúng ta làm như vậy thì sẽ **reloading** lại toàn bộ page từ server
- ❑ **SPA** chỉ có 1 trang **HTML**, nhưng bao gồm nhiều page views (components), và sẽ load ra page view tương ứng dựa trên route chứa component đó



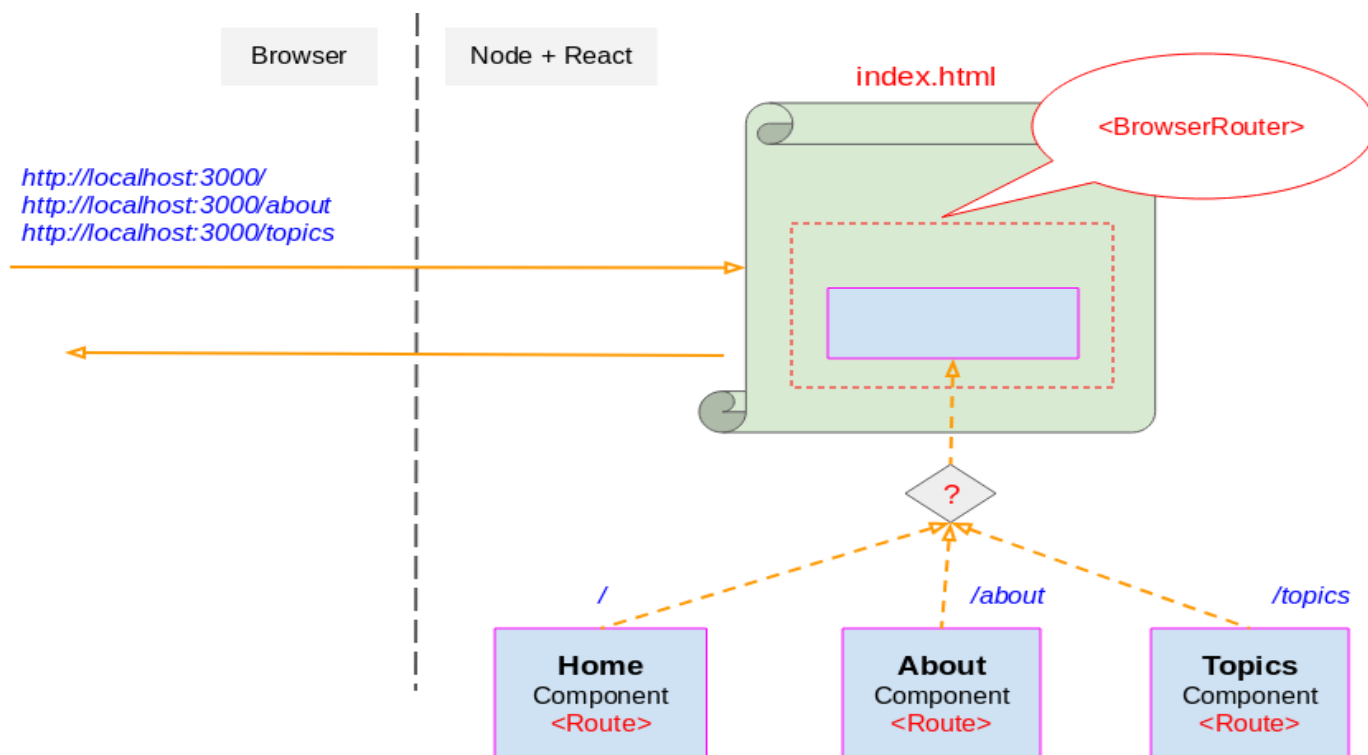
Cơ chế Routing

□ Minh họa:



Cơ chế Routing

❑ **React Router** là một thư viện **routing** sử dụng để điều hướng từ page này sang page khác mà không **refresh browser**



Cấu hình React Router

❑ Ví dụ:

<Router>

```
<div className="App">  
  <Route path="/" exact component={Home} />  
  <Route path="/about" component={About} />  
  <Route path="/contact" component={Contact} />  
  <Route component={NotFound}/>  
</div>
```

</Router>

❑ Trong đó:

- **path**: đường dẫn trên **URL**
- **exact**: giúp **route** này chỉ hoạt động nếu trình duyệt phù hợp với giá trị tuyệt đối của thuộc tính **path** của nó.
- **component**: là component sẽ được load ra tương ứng với **Route** đó.

Tóm tắt bài học

- ☐ Tổng quan về React Hook
- ☐ Các Hook căn bản
- ☐ Các Hook bổ sung
- ☐ Các Hook tùy chỉnh
- ☐ Cơ chế Routing
- ☐ Cấu hình React Router

