



React js

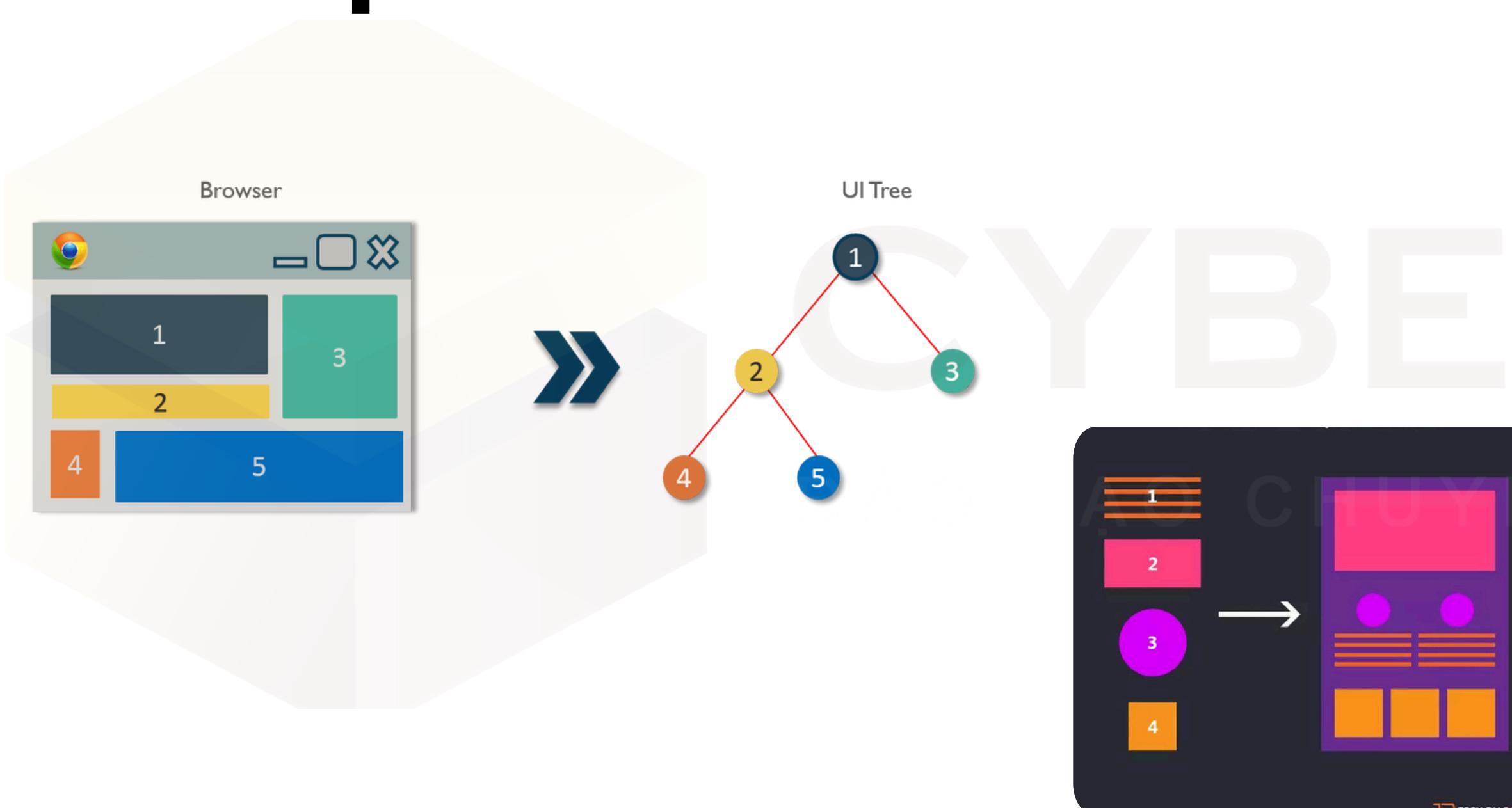
React fundamentals



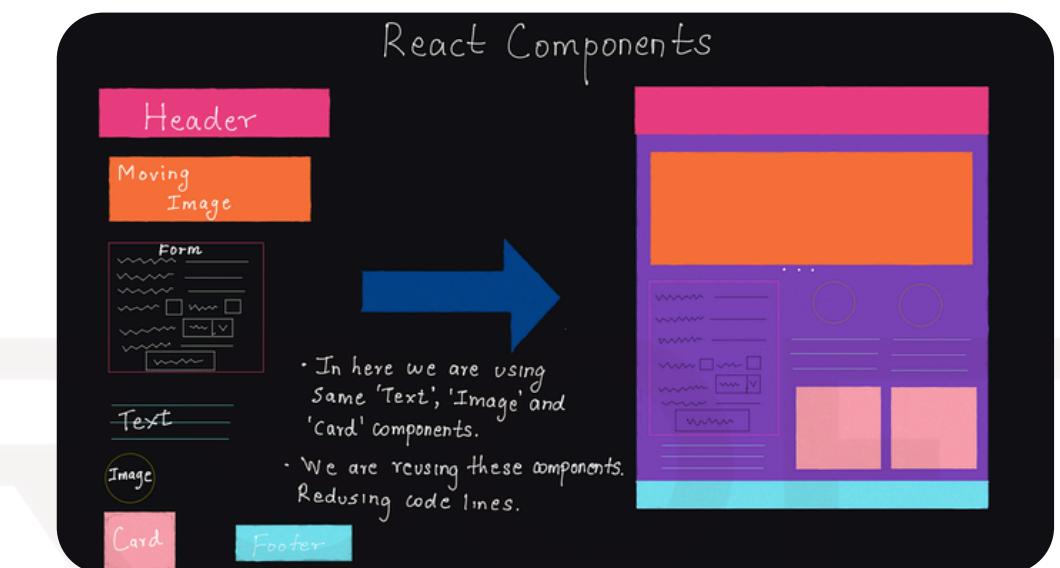
Play

React Component

1. REACT XÂY DỰNG GIAO DIỆN ỨNG DỤNG THEO KIẾN TRÚC COMPONENT.
2. REACT HỖ TRỢ XÂY DỰNG SPA (SINGLE PAGE APPLICATION).



Chia giao diện và ứng dụng
thành nhiều thành phần



Bảo trì mà mở rộng dễ dàng hơn



REACT VITE

React Vite là một công cụ phát triển front-end mới nổi, được xây dựng trên cơ sở Vite, một công cụ phát triển nhanh chóng của **Evan You, người tạo ra Vue.js**. Điểm mạnh của React Vite bao gồm:

TỐC ĐỘ PHÁT TRIỂN

React Vite sử dụng cơ sở của Vite, giúp tạo ra môi trường phát triển nhanh hơn so với các công cụ khác như Create React App (CRA) hoặc webpack. Vite sử dụng ESBuild để tăng tốc quá trình biên dịch các tệp JavaScript và JSX. ESBuild là công cụ biên dịch nhanh bởi vì nó được viết bằng ngôn ngữ Go và có khả năng thực hiện các tác vụ biên dịch song song.

HOT MODULE REPLACEMENT (HMR)

HMR cho phép bạn thấy các thay đổi ngay lập tức khi bạn chỉnh sửa mã nguồn của mình, mà không cần phải làm mới trình duyệt. Điều này giúp tăng tốc độ phát triển và tăng hiệu suất làm việc.

CẤU HÌNH ĐƠN GIẢN

TypeScript là một lựa chọn phổ biến trong cộng đồng React, và React Vite hỗ trợ TypeScript nền tảng một cách mạnh mẽ, giúp việc phát triển ứng dụng React với TypeScript trở nên dễ dàng hơn.

KHẢ NĂNG TÍCH HỢP VỚI ECOSYSTEM REACT

React Vite có thể tích hợp dễ dàng với các thư viện và công cụ khác trong hệ sinh thái React như React Router, Redux, và nhiều thứ khác.



CÀI ĐẶT REACT VITE

1. CÀI ĐẶT NODEJS & NPM

```
https://nodejs.org/en/download
```

2. CÀI ĐẶT VITE BẰNG NPM

```
npm create vite@latest
```

3. CÀI ĐẶT YARN VÀ SỬ DỤNG YARN CÀI VITE

```
npm install yarn -g
```

```
yarn create vite react-cybersoft --template react
```

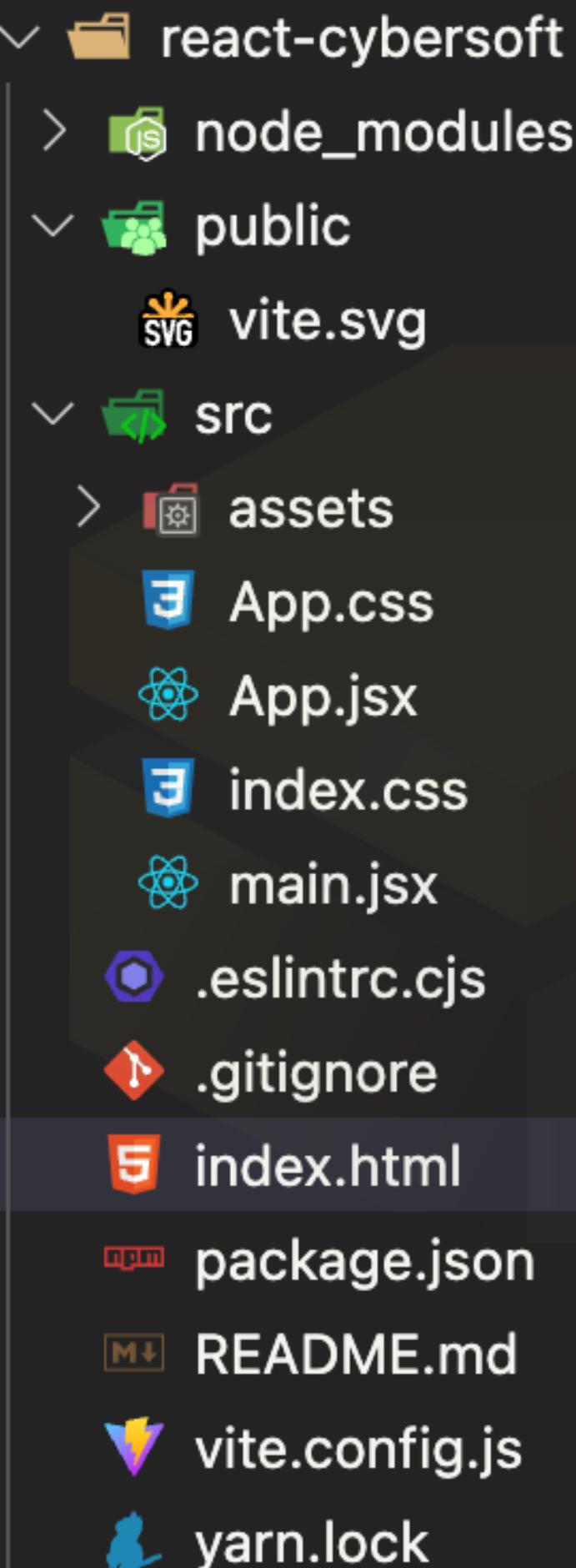
4. RUN PROJECT

```
cd react-cybersoft
```

```
yarn install hoặc npm install
```

```
yarn run dev hoặc npm run dev
```

DEMO_VITE_CYBERSOFT



- **node_modules**: Thư mục này chứa tất cả các thư viện và module mà dự án của bạn cài đặt vào bằng yarn hoặc npm.
- **public**: Thư mục này chứa các tệp tĩnh như favicon, biểu tượng, và tệp manifest.
- **src**: Thư mục chính chứa mã nguồn của dự án. Thường bao gồm các tệp JavaScript hoặc JSX, cũng như CSS hoặc các tệp tài nguyên khác liên quan trực tiếp đến mã nguồn.
- **assets**: Thư mục con trong src, thường được sử dụng để lưu trữ các tài nguyên như hình ảnh, video hoặc các tệp CSS, font...
- **app.jsx**: Một tệp JavaScript hoặc JSX chứa thành phần gốc (root component) của ứng dụng React.
- **main.jsx**: Tệp JavaScript hoặc JSX khởi động ứng dụng React bằng cách render thành phần gốc vào DOM trên trang index.html.
- **.eslintrc.cjs**: Tệp cấu hình cho ESLint, một công cụ giúp phát hiện và báo cáo lỗi vấn đề trong mã JavaScript, giúp code chuẩn và dễ đọc hơn.
- **.gitignore**: Loại bỏ các đường dẫn folder không muốn commit của git
- **index.html**: Tệp HTML chính của dự án, nơi thành phần gốc của React được mount. Thường chứa một thẻ <div> mà React sẽ gắn kết vào.
- **package.json**: Tệp cấu hình quan trọng cho dự án, chứa thông tin về dự án và liệt kê các thư viện mà dự án sử dụng. Nó cũng định nghĩa các lệnh script chạy trong dự án.
- **README.md**: Tệp Markdown chứa thông tin về dự án, hướng dẫn sử dụng, và các thông tin khác mà người dùng cần biết khi làm việc với dự án.
- **vite.config.js**: Tệp cấu hình cho Vite, công cụ build và development server mà bạn sử dụng trong dự án. Nó cho phép bạn tùy chỉnh cách Vite xử lý các tài nguyên, tối ưu hóa và biên dịch.
- **yarn.lock**: Tệp khóa được tạo ra và quản lý bởi Yarn để đảm bảo rằng các phụ thuộc được cài đặt một cách nhất quán trong các môi trường phát triển khác nhau, bằng cách lưu trữ chính xác thông tin về phiên bản của mỗi gói phụ thuộc đã cài đặt.

Các khái niệm cơ bản React functional component



Component (functional Component)

1

2

3

4

5

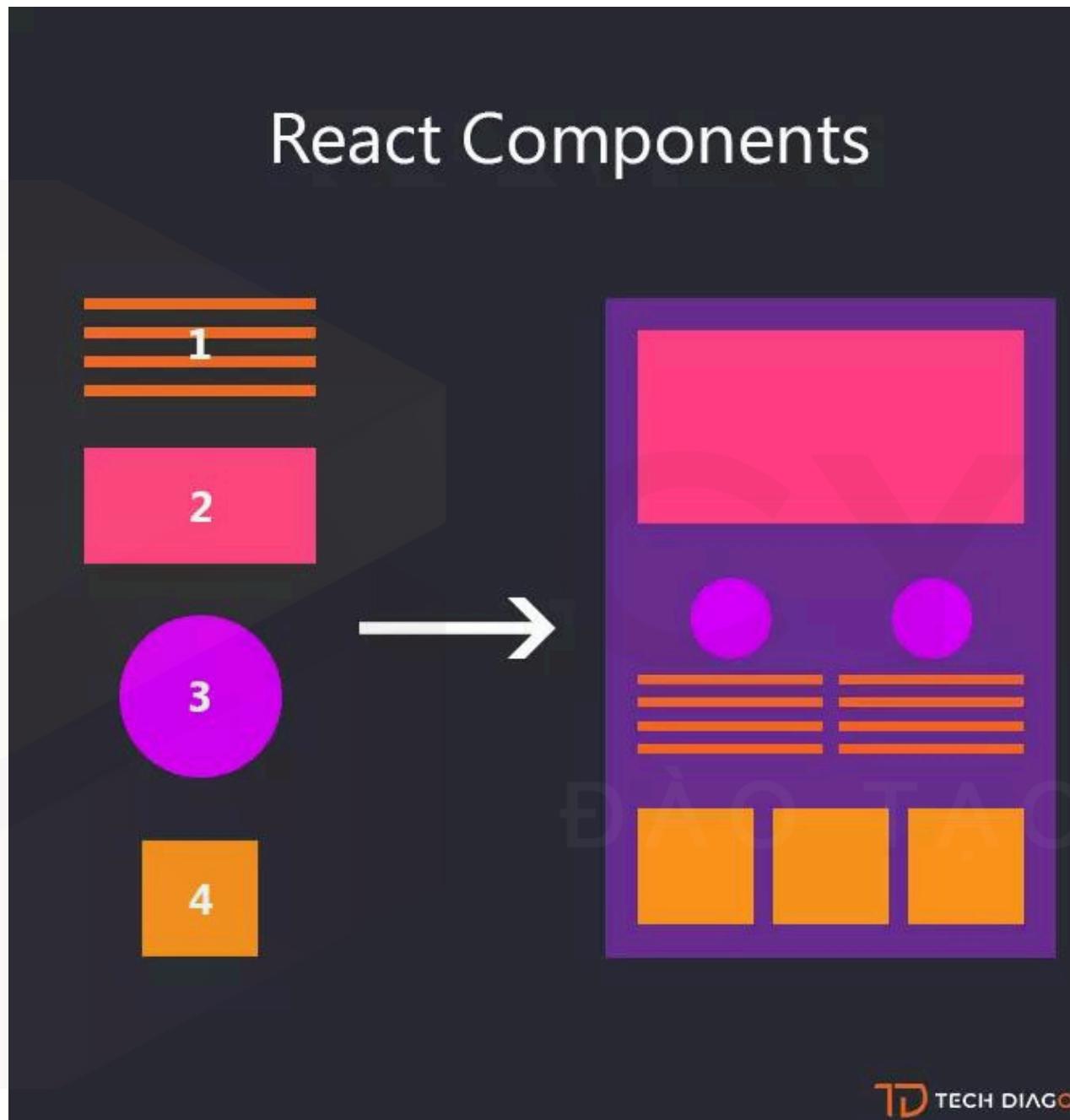
6

7

8

9

10



- React component có 2 dạng: **Class component** và **functional component**
- Tuy nhiên trong khoá học này chúng ta sẽ chỉ học về functional component (class các bạn có thể xem thêm video trên hệ thống) vì:
 - Tại thời điểm hiện tại trong lập trình React, xu hướng chung là sử dụng các **function components** kết hợp với **Hooks** thay vì **class components**. Điều này dựa trên nhiều lý do và sự khuyến khích từ cộng đồng React và nhóm phát triển của React.
 - **Function components** thường đơn giản hơn và ngắn gọn hơn class components. Chúng không yêu cầu định nghĩa các phương thức như **constructor**, **render**, hay xử lý **this**.
 - Dễ hiểu và Dễ dùng
 - **Function components** cùng với **Hooks** giúp mã nguồn dễ hiểu hơn.
 - **Hooks** hỗ trợ nhiều tiện ích hơn với cú pháp ngắn hơn cũng cho phép các nhà phát triển bên thứ 3 phát triển dễ dàng với nhiều thư viện.

Các khái niệm cơ bản React functional component



Component (functional Component)

Hướng dẫn cài đặt Cài đặt extension: [React snippet](#) và [extension html to jsx](#)

1

2

3

4

5

6

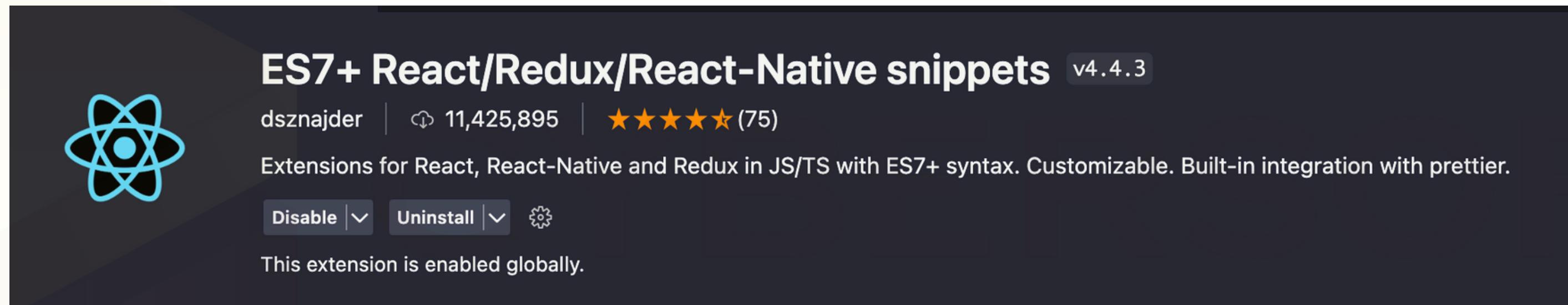
7

8

9

10

<https://marketplace.visualstudio.com/items?itemName=dsznajder.es7-react-js-snippets>



Tạo component:

rafce

Tên component viết
hoa ký tự đầu tiên

Nội dung component viết trong lệnh
return và được bao bởi 1 thẻ (thường
là thẻ div hoặc <></>)

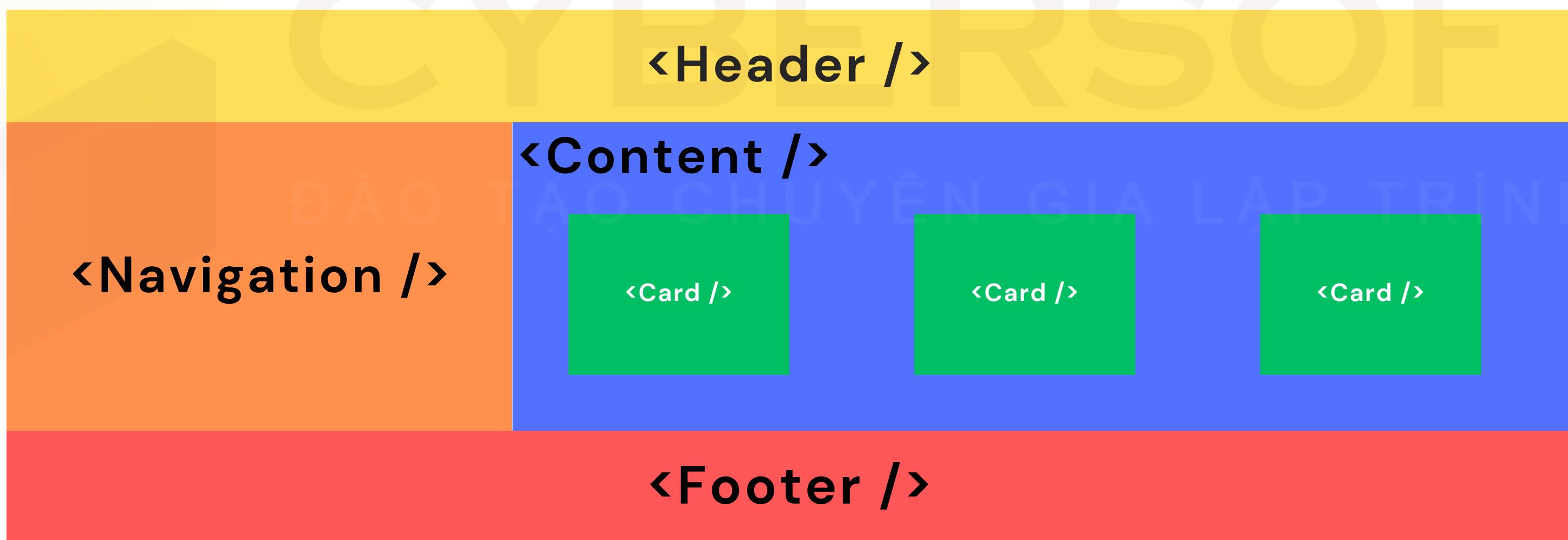
```
const HeaderCybersoft = () => {  
  return (  
    <div>HeaderCybersoft</div>  
  )  
}  
  
export default HeaderCybersoft
```

Các khái niệm cơ bản React functional component



Component

- Bài tập thực hành Sử dụng cdn bootstrap
 - Thực hành 1:
 - Tạo 1 component Header với react class component chứa tiêu đề là cyberlearn.vn
 - Tạo 1 component Product với react functional component nội dung chứa card bootstrap
 - Thực hành 2:
 - Sử dụng bootstrap tạo các component như hình bên dưới



1

2

3

4

5

6

7

8

9

10

Các khái niệm cơ bản React functional component



Data binding (interpolation)

1

Jsx cho phép ta lồng javascript vào HTML thông qua dấu {}

2

```
<div>{data}</div>
```

3

```
<div>{method()}</div>
```

4

Lưu ý: dữ liệu binding bao gồm

- string
- number
- jsx (các thẻ html hoặc thẻ component)

5

6

7

8

9

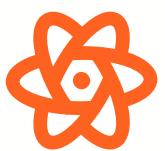
10

```
4 const Databinding = () => {
5   //primitive value
6   const fullName = 'Khải cybersoft';
7   //reference value
8   const prod = ...
9 }
10 //function
11 const renderCard = () => {
12   //Nội dung return của hàm có thể là :string, number, boolean hoặc jsx (các thẻ html của react hoặc component của react)
13   return <div className='card w-25'>
14     <img src={prod.img} alt='...' />
15     <div className='card-body'>
16       <h3>{prod.name}</h3>
17       <p>{prod.price}</p>
18       <button className='btn btn-dark'>Detail</button>
19     </div>
20   </div>
21 }
22 const divBadge = <div className='badge bg-danger'>{prod.name}</div>
23 return (
24   <div className='container'>
25     <span id="full-name" className='badge bg-dark'>
26       {fullName}
27     </span>
28     {renderCard()}
29     {divBadge}
30   </div>
31 )
32
33
34
35
36
37
38 export default Databinding
```

binding hàm

binding giá trị

Các khái niệm cơ bản React functional component



Event handler(Xử lý sự kiện)

1

Các sự kiện `onClick`, `onChange`, `onSubmit` ... trong javascript đều có thể sử dụng trong react.

2

Tuy nhiên sẽ có khác biệt về cú pháp => Tham khảo ví dụ bên dưới:

3

Truyền dạng callback function (khi xảy ra sự kiện hàm mới thực thi)

```
<Button onClick={callback_function}> Click me ! </button>
```

4

Truyền dạng anonymous function

```
<jsx onEvent={(event) => {
    method1()
    method2()
}}> Click me ! </jsx>
```

5

6

7

8

9

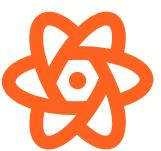
10

```
1 //rafce
2 import React from 'react' 6.9k (gzipped: 2.7k)
3 const HandleEvent = () => {
4   const showMessage = (e) => {
5     console.log('hello hello', e)
6   }
7   const showMessageName = (name) => {
8     console.log('hello hello', name)
9   }
10  return () => {
11    const khai = 'Khai, 4 weeks ago • dp'
12    return (
13      <div className='container'>
14        <h3>Handle event</h3>
15        <button className='btn btn-dark' onClick={(e) => {
16          console.log('clicked !')
17        }}> click me</button>
18        <br /> <br />
19        <button className='btn btn-danger' onClick={showMessage}>show message</button>
20        <button className='btn btn-danger' onClick={showMessage}>show mess</button>
21        <button className='btn btn-danger' onClick={(e) => {
22          showMessageName('Khải')
23        }}>show mess name</button>
24        <input className='w-25 form-control' onChange={(e) => {
25          const value = e.target.value;
26          document.querySelector('#txt').innerHTML = value;
27          console.log('value', value);
28        }} />
29        <span id="txt" className='d-block my-2'></span>
30      </div>
31    )
32  }
33}
34 export default HandleEvent
```

anonymous function

callback function

Các khái niệm cơ bản React functional component



Event handler(Xử lý sự kiện)

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

- Passing Arguments to Event Handlers

Để truyền tham số qua sự kiện ta có thể sử dụng 2 cách bên dưới

```
<Button onClick={callback_function.bind(this,param)}> Click me ! </button>
```

Truyền dạng anonymous function

```
<jsx onEvent={(event) => {
    method1(param)
    method2(param)
}}> Click me ! </jsx>
```

```
1 //rafce
2 import React from 'react'  6.9k (gzipped: 2.7k)
3 const HandleEvent = () => {
4   const showMessage = (e) => {
5     console.log('hello hello', e)
6   }
7   const showMessageName = (name) => {
8     console.log('hello hello', name)
9   }
10  return () => {
11    khai, 4 weeks ago • dp
12    <div className='container'>
13      <h3>Handle event</h3>
14      <button className='btn btn-dark' onClick={(e) => {
15        console.log('clicked !')
16      }}> click me</button>
17      <br /> <br />
18      <button className='btn btn-danger' onClick={showMessage}>show message</button>
19      <button className='btn btn-danger' onClick={showMessage}>show mess</button>
20      <button className='btn btn-danger' onClick={(e)=> {
21        showMessageName ('Khải')
22      }}>show mess name</button>
23
24
25      <input className='w-25 form-control' onChange={(e)=>{
26        const value = e.target.value;
27        document.querySelector('#txt').innerHTML = value;
28        console.log('value',value);
29      }} />
30      <span id="txt" className='d-block my-2'></span>
31    </div>
32  }
33
34 export default HandleEvent
```

anonymous function

callback_function

Các khái niệm cơ bản React functional component



Conditional Rendering

1

Conditional Rendering trong React cho phép hiển thị các thành phần hoặc các phần khác nhau của giao diện dựa trên điều kiện nhất định.

2

Cú pháp:

3

Ternary operator (Toán tử 3 ngôi)

4

```
{isLoggedIn ? <h1>Welcome back!</h1> : <h1>Please sign up.</h1>}
```

5

Toán tử && (And logic)

6

```
{isLoggedIn && <h1>Welcome back!</h1>}
```

7

Hoặc đơn giản nhất dùng method

8

```
function Welcome(props) {
  if (props.isLoggedIn) {
    return <h1>Welcome back!</h1>;
  } else {
    return <h1>Please sign up.</h1>;
  }
}
```

9

10

```
1 import React, { useState } from 'react';  6.9k (gzipped: 2.7k)
2
3 function Welcome() {
4   const [isLoggedIn, setIsLoggedIn] = useState(false);
5
6   // Sử dụng câu lệnh if
7   if (isLoggedIn) {
8     return <h1>Welcome back! (using if)</h1>;
9   }
10
11 // Sử dụng toán tử ba ngôi trong JSX
12 return (
13   <div>
14     {isLoggedIn ? (
15       <h1>Welcome back! (using ternary operator)</h1>
16     ) : (
17       <h1>Please sign up. (using ternary operator)</h1>
18     )}
19
20   /* Sử dụng toán tử && */
21   {isLoggedIn && <p>You are logged in. (using &&)</p>}
22   {!isLoggedIn && <p>You are not logged in. Please sign up or log in. (using &&)</p>}
23
24   /* Button to toggle login state */
25   <button onClick={() => setIsLoggedIn(!isLoggedIn)}>
26     {isLoggedIn ? 'Log Out' : 'Log In'}
27   </button>
28   </div>
29 )
30
31 export default Welcome;
```

Các khái niệm cơ bản React functional component



State

1

2

3

4

5

6

7

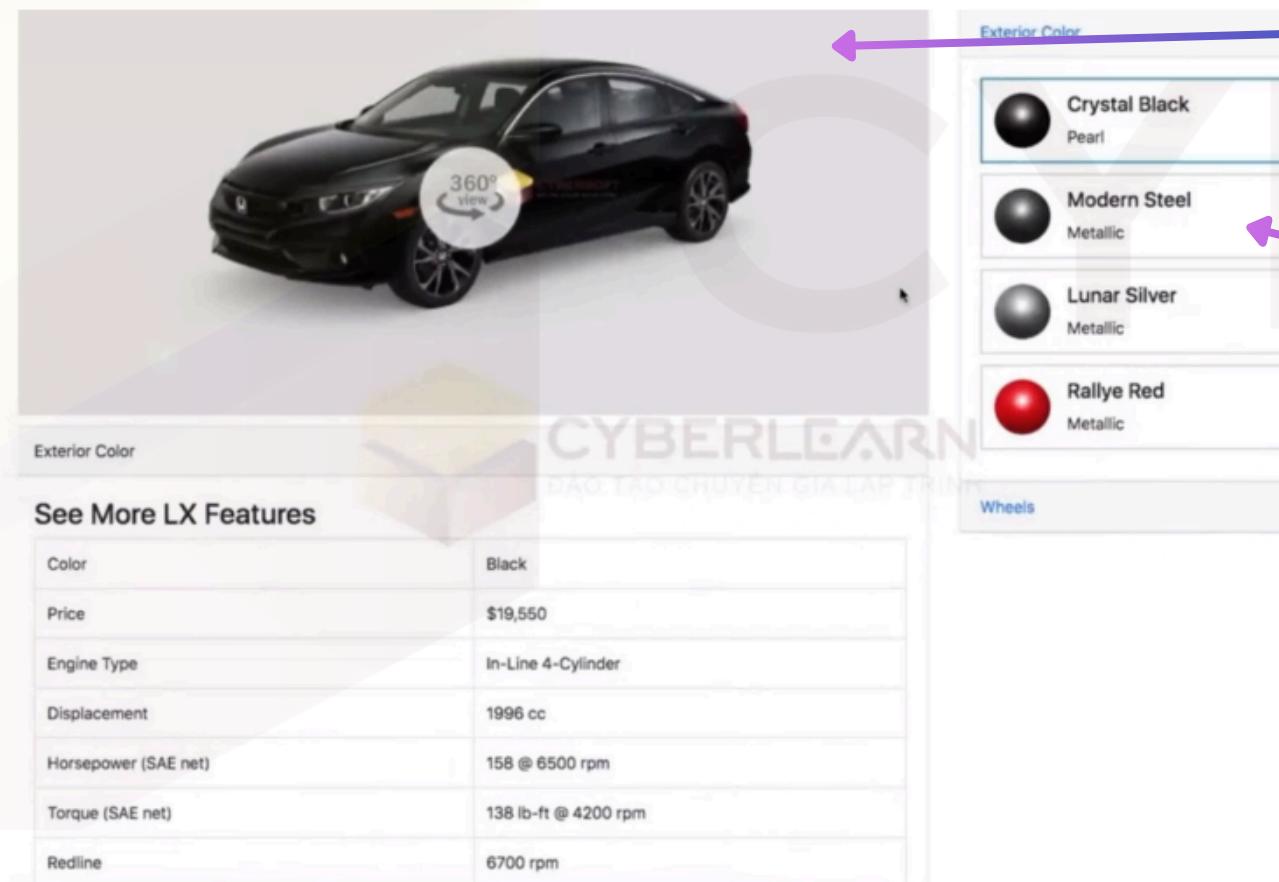
8

9

10

- state là gì ?

- State đại diện cho giá trị thay đổi của component khi event xảy ra. Ví dụ như click button thì hình thay đổi hay zoom font chữ, ...
 - state có thể là giá trị: string, number, boolean, <jsx /> hoặc callback function



state hiện tại là string (thuộc tính của src)

khi click vào button thì hình xe
thay đổi tương ứng

Các khái niệm cơ bản React functional component



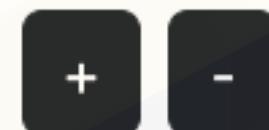
State

Để setup state trong react functional component ta sử dụng hook useState. Thực hành ví dụ bên dưới:

```
import { useState } from 'react'
const [state, setState] = useState(default_value)
```

State

Count: 10



state

binding state

```
1 //state: state là giá trị thay đổi bởi sự kiện trong component
2 import { useState } from 'react' 4.1k (gzipped: 1.8k)
3 const DemoState = () => {
4   const [state, setState] = useState(10) →
5   return (
6     <div className='container'>
7       <h3>State</h3>
8       <h3>Count: <span className='text-danger'>{state}</span></h3> →
9     >
10      <button className='btn btn-dark' onClick={()=>{...}}>+</button>
11      <button className='btn btn-dark ms-2'>-</button> →
12    </div>
13  )
14}
15
16 export default DemoState
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Các khái niệm cơ bản React functional component



State

- Bài tập thực hành

Lorem ipsum dolor sit amet consectetur adipisicing elit. Ab, alias.

zoom in **zoom out**

- Xây dựng chức năng tăng giảm số lượng cho 2 button + và -
- Yêu cầu:
 - Xây dựng layout
 - Xác định state là gì ? (string, boolean, number, object, array)
 - Binding state lên lên phần giao diện cần thiết
 - Xử lý thay đổi fontsize khi người dùng bấm vào các nút

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Các khái niệm cơ bản React functional component



State

- Các bước giải quyết 1 bài toán hay 1 chức năng trong ReactJS

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Các bước giải quyết 1 task hoặc 1 tính năng trong react

Xây dựng giao diện hoàn chỉnh 100%

Xác định state là gì?

- number ?
- boolean ?
- string ?
- object?
- array?

Binding State lên giao diện

Xử lý sự kiện thay đổi state => setState

Bài tập:



Yêu cầu:

- 1 Xây dựng layout
- 2 Xác định state là gì ? (string, boolean, number, object, array)
- 3 Binding state lên lên phần giao diện cần thiết
- 4 Xây dựng chức năng để khi người dùng click vào các button thì xe sẽ đổi màu tương ứng
- 5 Resource:

https://drive.google.com/drive/folders/1xMdYF3L_1OQUO6I8Dm1AsXYhlJnr3iO2?usp=sharing

Change color car



Red color Silver color Black color



Bài tập:

Yêu cầu:

1

Xây dựng layout (Có thể sử dụng bootstrap hoặc thư viện)

2

Xác định state là gì ? (string, boolean, number, object, array)

3

Binding state lên lên phần giao diện cần thiết

4

Xây dựng chức năng để khi người dùng click vào button like hoặc dislike thì ảnh sẽ thay đổi

5

Resource:

Hình ảnh ngẫu nhiên được lấy: [https://i.pravatar.cc?u=\[tham_so\]](https://i.pravatar.cc?u=[tham_so])

6

Xử lý setState thay đổi tham số lấy từ số ngẫu nhiên có thể dùng hàm có sẵn của js: `Math.floor(Math.random() * 70);`

7

8

9

10

Bài tập:



Yêu cầu:

- Xây dựng ứng dụng tăng giảm phóng chữ cho 2 nút button
- Xây dựng giao diện
- Xác định state là gì ?
- Binding state lên thuộc tính tương ứng
- Xây dựng xử lý setState

LOREM IPSUM DOLOR SIT AMET CONSECTETUR ADIPISCING ELIT. AB, ALIAS.

+ zoom in - zoom out

1

2

3

4

5

6

7

8

9

10

Các khái niệm cơ bản React functional component



Props

1

2

3

4

5

6

7

8

9

10

Props (viết tắt của properties) là một object được truyền vào một `<Component />`. Tại `<Component />` nhận props thì ta có thể sử dụng `props.[props_name]` tương ứng để nhận dữ liệu và binding lên giao diện.

Các giá trị mà props nhận có thể là : `string, boolean, number, null, undefined, object, name, callback`

```
const ParentComponent = () => {
  return <ChildComponent name="Freetuts" />;
};

const ChildComponent = (props) => {
  return <h1>Xin chào, {props.name}</h1>;
};
```



Một số ví dụ tham khảo



1

1. String:

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}  
  
const element = <Welcome name="Sara" />;
```

2

2. Boolean:

```
function Alert(props) {  
  return props.isError ? <div>Error!</div> : <div>All good!</div>;  
}  
  
const element = <Alert isError={true} />;
```

3

3. Number:

```
function Age(props) {  
  return <div>Your age is {props.age}</div>;  
}  
  
const element = <Age age={30} />;
```

4

4. Null và Undefined (thường được sử dụng để không hiển thị một component):

```
function Greeting(props) {  
  return props.message ? <div>{props.message}</div> : null;  
}  
  
const element = <Greeting message={null} />;
```

5

5. Object:

```
function Profile(props) {  
  return (  
    <div>  
      <h1>{props.user.name}</h1>  
      <p>Age: {props.user.age}</p>  
    </div>  
  );  
}  
  
const user = { name: 'John Doe', age: 28 };  
const element = <Profile user={user} />;
```

6

6. Array:

```
function ShoppingList(props) {  
  return (  
    <ul>  
      {props.items.map((item, index) => (  
        <li key={index}>{item}</li>  
      ))}  
    </ul>  
  );  
}  
  
const items = ['Apple', 'Orange', 'Banana'];  
const element = <ShoppingList items={items} />;
```

7

7. Callback (hàm):

```
function Button(props) {  
  return <button onClick={props.onClick}>Click me!</button>;  
}  
  
function handleClick() {  
  alert('Button clicked!');  
}  
  
const element = <Button onClick={handleClick} />;
```

8

9

10

8. JSX (truyền một component như là một prop):

```
function Page(props) {  
  return <div>{props.header}</div>;  
}  
  
const headerComponent = <h1>Header</h1>;  
const element = <Page header={headerComponent} />;
```

Bài tập props



Cho mảng dataJson.

1

Link: https://drive.google.com/file/d/1ccQ1TptUOYhS2FcRAfU9_ LEi6oczW2ZX/view?usp=sharing.

- Tạo cấu trúc component như sau:

<ProductList>

```
<ProductItem item={data_item} />  
<ProductItem item={data_item} />  
<ProductItem item={data_item} />
```

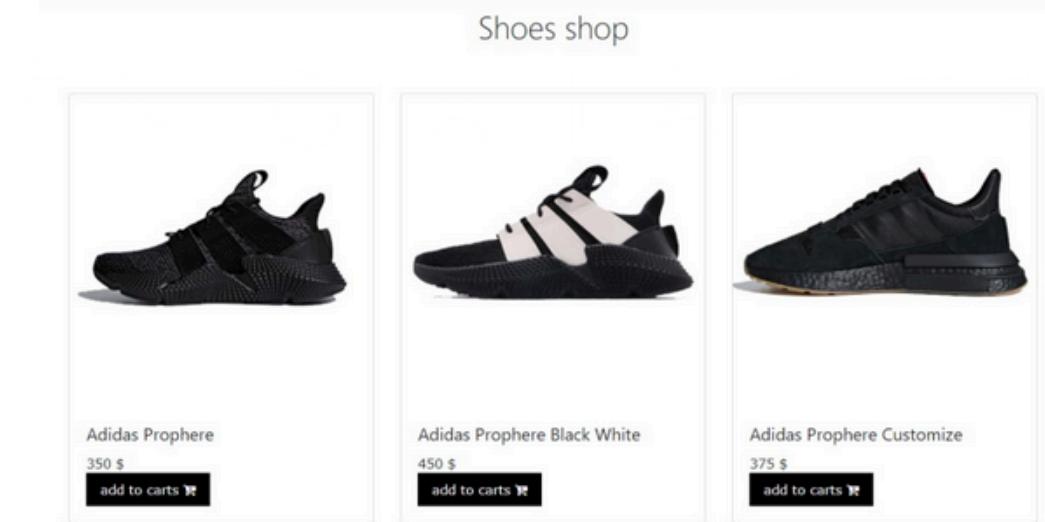
</ProductList>

- Yêu cầu dùng props để tạo giao diện như sau. Có thể dùng bootstrap, w3css, material... để hỗ trợ

6

Gợi ý:

```
import React from 'react';  6.9k (gzipped: 2.7k)  
  
// Dữ liệu sản phẩm  
const arrProduct = [  
  // ...dữ liệu sản phẩm của bạn  
];  
  
const ProductList = () => {  
  
  return <>  
    /* Nội dung component */  
  </>  
}  
  
export default ProductList
```



Lifting state up



1

2

3

4

5

6

7

8

9

10

Trong react khi phân chia component đôi lúc state ta đặt 1 nơi nhưng các event xử lý lại nằm ở nhiều nơi. Lifting state up là cách để chúng ta xác định state được đặt tại component nào.

Lấy ví dụ về chức năng xem chi tiết đơn hàng với cấu trúc component như sau

Ta có 2 component là DanhSachSanPham, SanPham

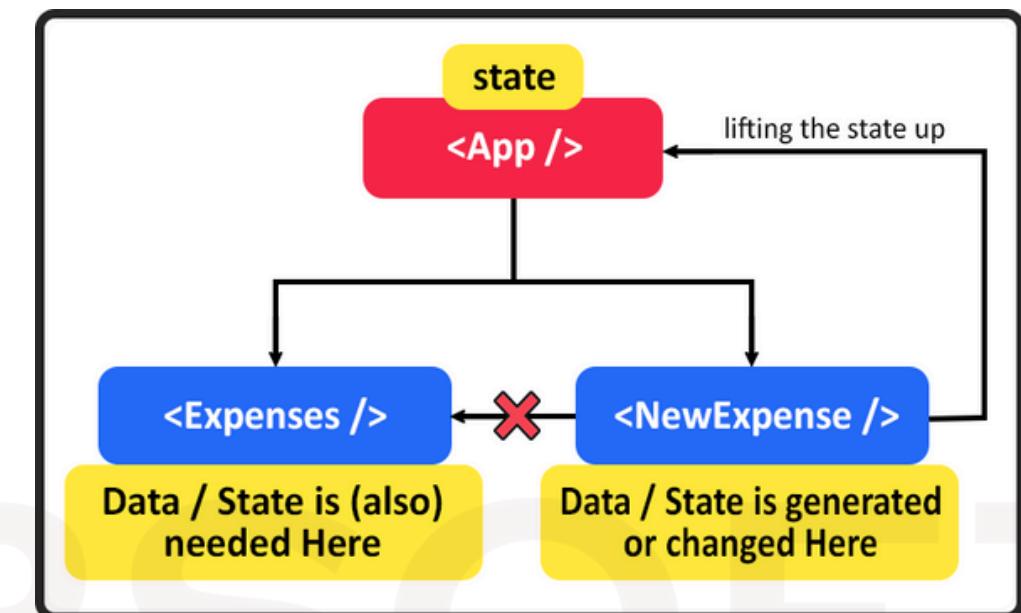
+ DanhSachSanPham chứa: 2 thuộc tính là

- mangSanPham
- state.sanPhamChiTiet

Yêu cầu:

- Khi click vào nút xem chi tiết thì thông tin của sản phẩm được click sẽ được cập nhật vào this.state.sanPhamChiTiet làm cho nội dung bên dưới thay đổi. Điều này thật đơn giản khi ta binding tất cả html chỉ trên 1 component duy nhất DanhSachSanPham.
- Tuy nhiên ta đã tách các sanPham ra thành riêng 1 component nên bên trong component sanPham không chứa state.sanPhamChiTiet nữa nên ta không thể cập nhật được. Do vậy tại component cha bắt buộc ta phải viết 1 hàm để truyền vào sự kiện click của component con để sau khi click thì nó sẽ lấy được dữ liệu tại hàm ta xây dựng ở component cha từ đó cập nhật lại state.sanPhamChiTiet => làm cho giao diện thay đổi
- Link resource:

<https://drive.google.com/drive/folders/1g15gVNroRB6e2vOJBkuMwgCg9byVcbzt>



Danh sách sản phẩm

VinSmart Live	Xem chi tiết	onclick
Meizu 16Xs	Xem chi tiết	onclick
Iphone XS Max	Xem chi tiết	onclick

VinSmart Live	Thông số kỹ thuật
Màn hình	AMOLED, 6.2", Full HD+
Hệ điều hành	Android 9.0 (Pie)
Camera trước	20 MP
Camera sau	Chinh 48 MP & Phụ 8 MP, 5 MP
RAM	4 GB
ROM	64 GB

Lifting state up



Bài tập:

1

Yêu cầu: Tạo cấu trúc component như bên dưới, xây dựng xử lý khi người dùng click vào nút xem chi tiết tại component ProductItem thì state của modal sẽ thay đổi.

2

Như hình ảnh bên dưới

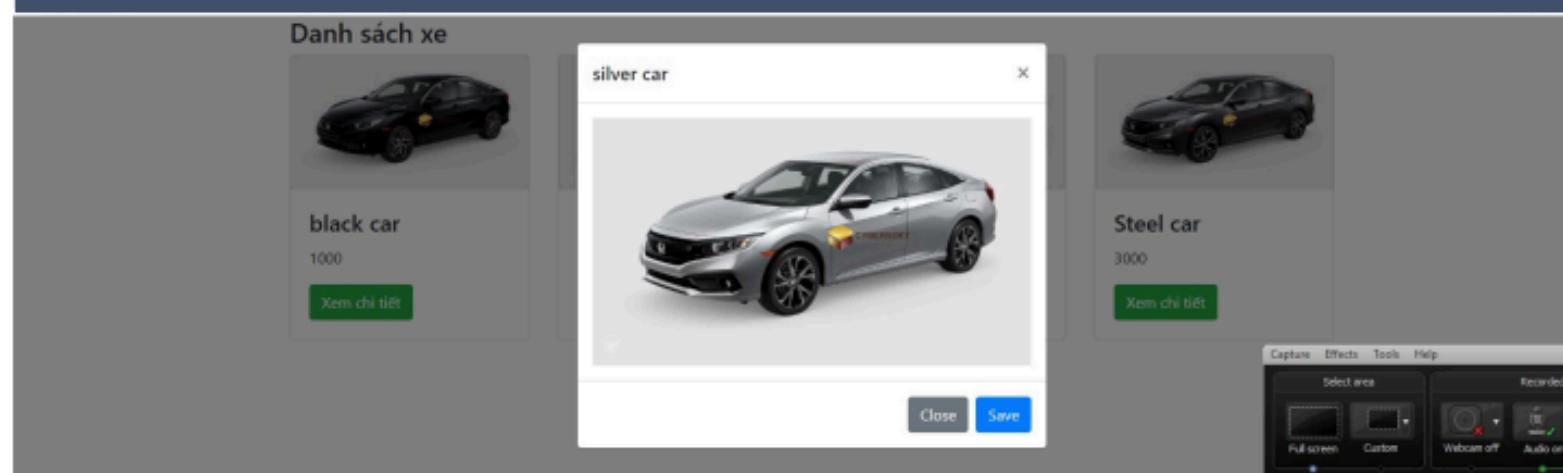
<ExerciseCarStore>

```
<Modal content={this.state.productDetail} />
<ProductList productsData={products} setStateModal = {this.setStateModal}>
  <ProductItem item={product} setStateModal={this.props.setStateModal}> </ProductItem>
  <ProductItem item={product} setStateModal={this.props.setStateModal}> </ProductItem>
  <ProductItem item={product} setStateModal={this.props.setStateModal}> </ProductItem>
</ ProductList >
</ExerciseCarStore >
```

Sử dụng link để lấy hình ảnh: <https://drive.google.com/open?id=1kaGEHc7IQJYCuXO-GHRO-KUFY1Xj9LHD>

Sử dụng data sau (Nhớ config lại so với thư mục hình ảnh của source hiện tại):

```
products = [
  { id: 1, name: 'black car', img: './images/products/black-car.jpg', price: 1000 },
  { id: 2, name: 'red car', img: './images/products/red-car.jpg', price: 2000 },
  { id: 3, name: 'silver car', img: './images/products/silver-car.jpg', price: 3000 },
  { id: 3, name: 'Steel car', img: './images/products/steel-car.jpg', price: 4000 }
];
```



3

4

5

6

7

8

9

10

Lifting state up



Bài tập:

1

- Tạo cấu trúc component như bên dưới, xây dựng xử lý khi người dùng click vào nút xem chi tiết tại component ProductItem thì state của modal sẽ thay đổi. Như hình ảnh bên dưới
 - <ExerciseCart>
 - <Cart cartData={cartData} />
 - <ProductList productsData={products} addToCart = {this.addToCart}>
 - <ProductItem item={product} addToCart={this.props.addToCart}></ProductItem>
 - <ProductItem item={product} addToCart = {this.props.addToCart}></ProductItem>
 - <ProductItem item={product} addToCart = {this.props.addToCart}></ProductItem>
 - </ ProductList >
 - </ ExerciseCart >
 - Sử dụng link để lấy hình ảnh: <https://drive.google.com/open?id=1g15gVNroRB6e2vOJBkuMwgCg9byVcbzt>
 - Sử dụng data sau (Nhớ config lại so với thư mục hình ảnh của source hiện tại):

2

3

4

5

6

7

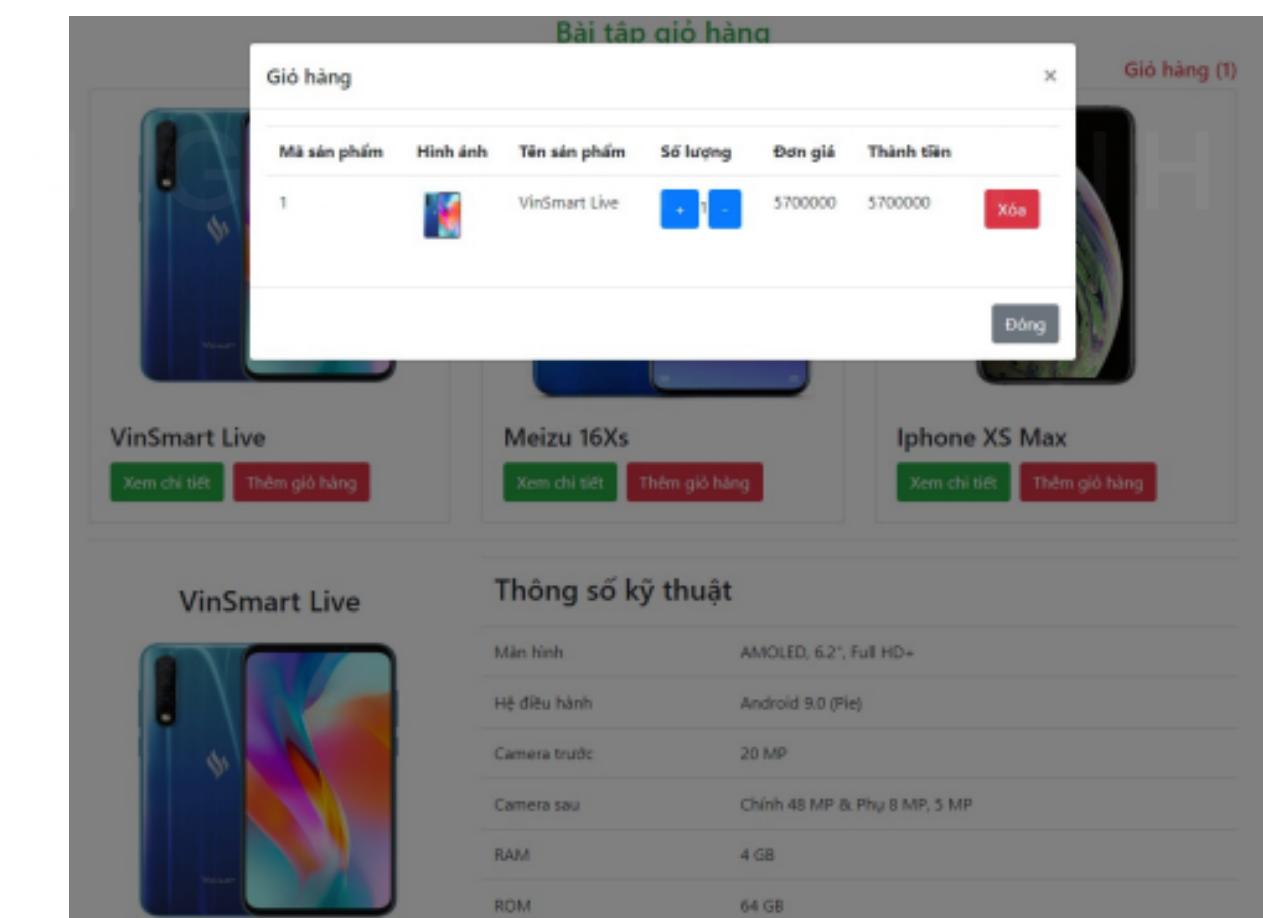
8

9

10

data.json

```
const mangSanPham = [  
  { maSP: 1, tenSP: "VinSmart Live", manHinh: "AMOLED, 6.2, Full HD+", heDieuHanh: "Android 9.0 (Pie)",  
   cameraTruoc: "20 MP", cameraSau: "Chính 48 MP & Phụ 8 MP, 5 MP", ram: "4 GB", rom: "64 GB", giaBan:  
   5700000, hinhAnh: "./img/vsphone.jpg" },  
  { maSP: 2, tenSP: "Meizu 16Xs", manHinh: "AMOLED, FHD+ 2232 x 1080 pixels", heDieuHanh: "Android 9.0  
   (Pie); Flyme", cameraTruoc: "20 MP", cameraSau: "Chính 48 MP & Phụ 8 MP, 5 MP", ram: "4 GB", rom: "64 GB",  
   giaBan: 7600000, hinhAnh: "./img/meizuphone.jpg" },  
  { maSP: 3, tenSP: "Iphone XS Max", manHinh: "OLED, 6.5, 1242 x 2688 Pixels", heDieuHanh: "iOS 12",  
   cameraTruoc: "7 MP", cameraSau: "Chính 12 MP & Phụ 12 MP", ram: "4 GB", rom: "64 GB", giaBan: 27000000,  
   hinhAnh: "./img/applephone.jpg" }];
```





- Để thực thi api trong react ta cần xác định thời điểm call api khi nào ? Ví dụ api thực thi trong 1 sự kiện (event click, event drag drop, ...) hay thực thi sau khi component load xong hoặc component thay đổi.

1

- Để xác định được các trường hợp đó ta sẽ cần đến 2 hook quan trọng (**useState**, **useEffect**). Vì tất cả dữ liệu từ api đều là **state** (Dữ liệu sẽ thay đổi giá trị khi gọi api).

2

- Hướng dẫn cài đặt axios & fetch**

3

Ví dụ:

4

- Sử dụng link api sau để load dữ liệu lên component products:
- Link: <https://apistore.cybersoft.edu.vn/api/Product>
- Khi click vào button api -> gọi api -> set lại state products

5

```
const [products, setProducts] = useState([])
const getApi = async () => {
  const res = await fetch('https://apistore.cybersoft.edu.vn/api/Product');
  const data = await res.json();
  setProducts(data.content);
  console.log(data);
}
```

6

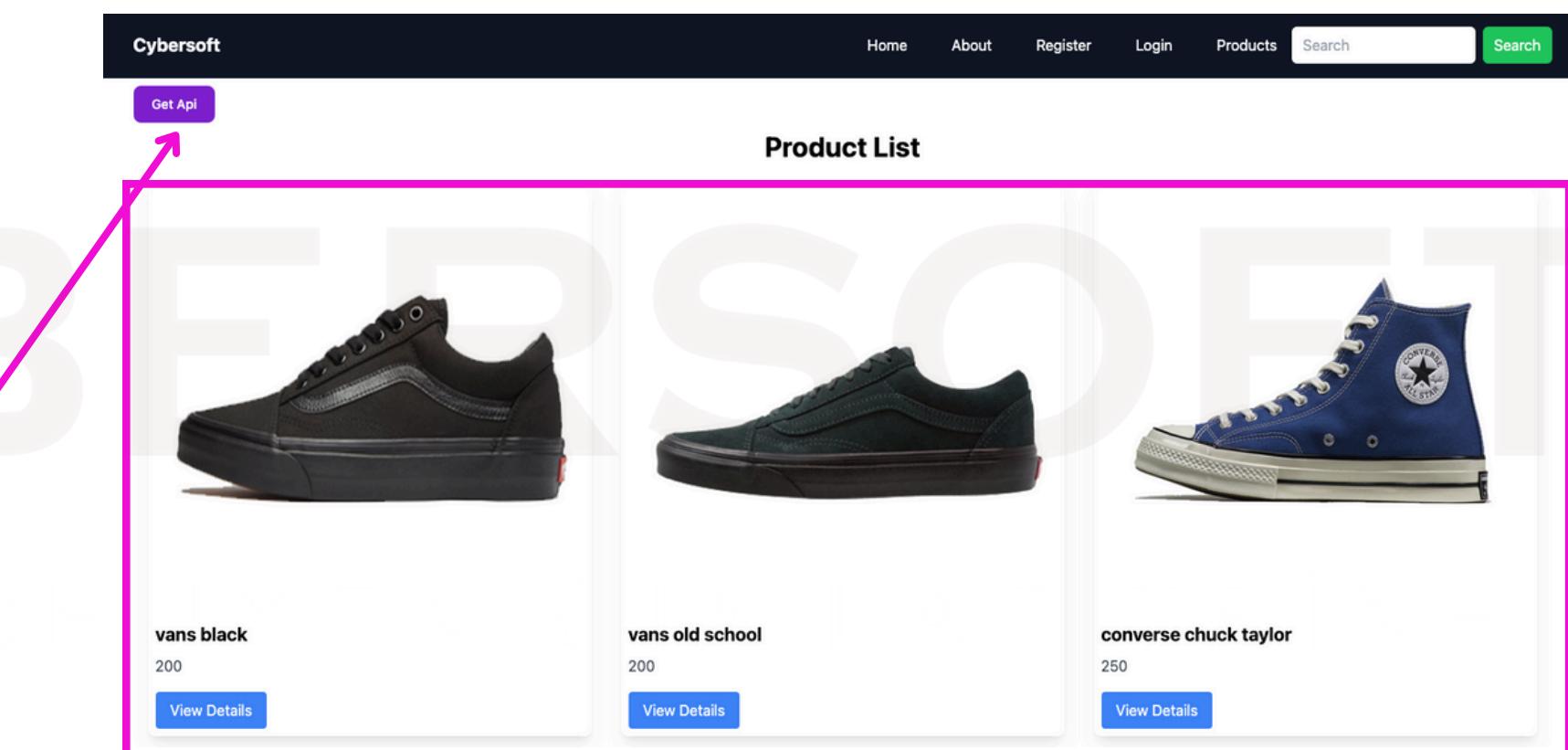
7

```
<button type="button" onClick={(e)=>{
  getApi()
}}>Get Api</button>
```

8

9

10



[state]productList = [{}, {}, {}]



• Bài tập:

1

Dựa vào link api sau xây dựng component Todolist: <https://svcy.myclass.vn/swagger/ui/index#/ToDoList> Link api
Với các tính năng:

- add task
- delete task
- complete task
- reject task

2

3

4

5

6

7

8

9

10

ToDoList	
GET	/api/ToDoList/GetAllTask
GET	/api/ToDoList/GetTask
POST	/api/ToDoList/AddTask
DELETE	/api/ToDoList/deleteTask
PUT	/api/ToDoList/doneTask
PUT	/api/ToDoList/rejectTask

Link api

The screenshot shows a "ToDo List" application interface. At the top, there is a navigation bar with links for Home, About, Register, Login, Products, a search input field, and a search button. Below the navigation bar is a section titled "ToDo List" containing a text input field labeled "Enter new todo" and a blue "Add" button. A list of five tasks is displayed, each with a green "Complete" button and a red "Delete" button. The tasks are:

- delectus aut autem
- quis ut nam facilis et officia qui
- laboriosam mollitia et enim quasi adipisci quia provident illum
- fugiat veniam minus
- et porro tempora

Ui tham khảo

HOOK USEEFFECT

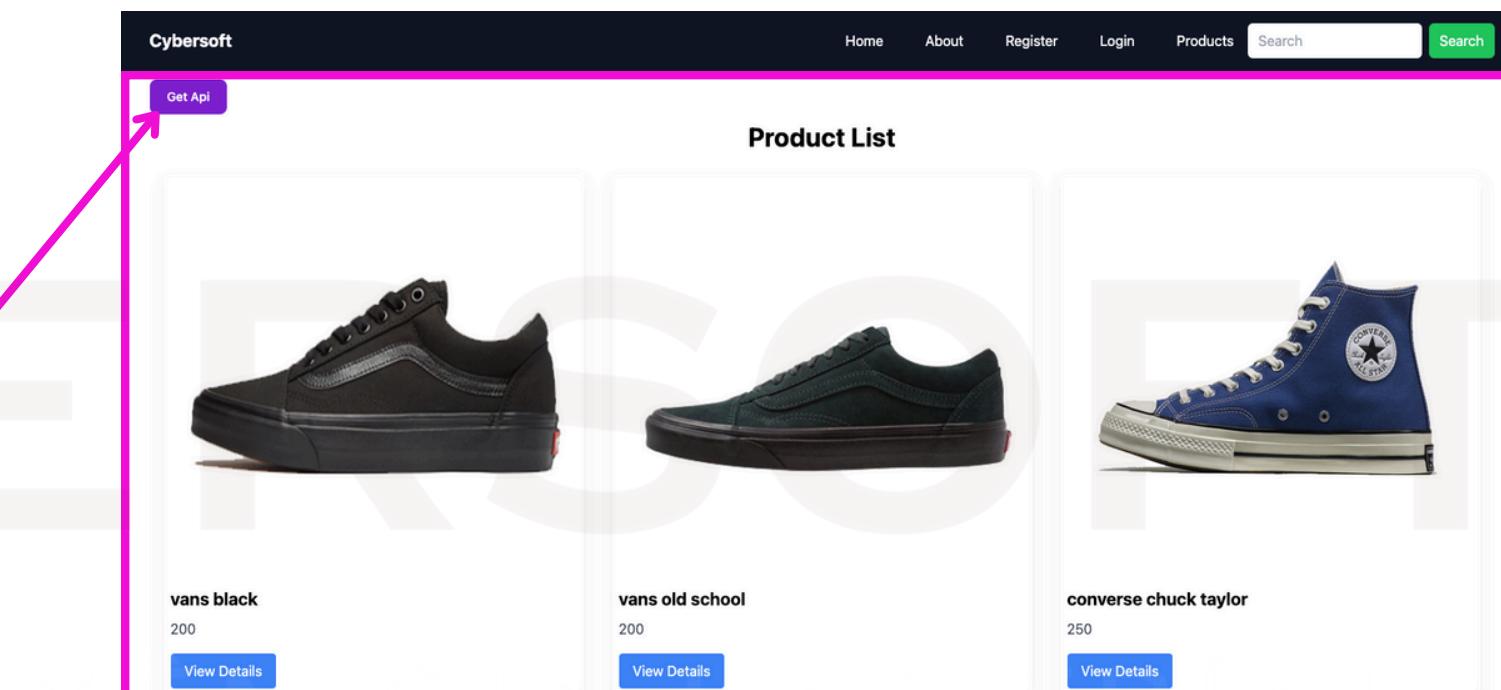
• UseEffect

Tương tự dom window.onload đõi với react việc thiết lập các xử lý **sau khi** <jsxTag /> được render hoặc mõi khi state thay đổi (<jsxTag /> render lại). Ta se~sử dụng useEffect để can thiệp vào quá trình này.

Ví dụ như callAPI tự động sau lân đâù tiên được load (Mounting component):

```
const [products, setProducts] = useState([])
const getApi = async () => {
    const res = await fetch('https://apistore.cybersoft.edu.vn/api/Product');
    const data = await res.json();
    setProducts(data.content);
    console.log(data);
}

useEffect(() => {
    //Call api ngay sau khi component load xong (Mounting component)
    getApi()
}, [])
```



useEffect có 2 giá trị:

useEffect(EffectCallback,DependencyList)

EffectCallback: là hàm thực thi sau khi render hoặc sau khi tham sô 'DependencyList' thay đổi

DependencyList ([dep1,dep2,dep3]): là khi trong trong các giá trị dependency thay đổi thì EffectCallback se~ được thực thi lại.

[state]productList = [{} , {} , {}]

HOOK USEEFFECT

- Các trường hợp của useEffect:

Các trường hợp của useEffect:

- 1 - **DependencyList** rỗng ([]):

- **useEffect** chỉ chạy một lần sau lần render đầu tiên và không bao giờ chạy lại.
- Thường được sử dụng để gọi API hoặc thiết lập các thao tác chỉ cần thực hiện một lần khi component mount.

```
useEffect(() => {  
  // Chạy một lần khi component mount  
  console.log('Component mounted');  
}, []); // Dependency list rỗng
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

HOOK USEEFFECT

- Các trường hợp của useEffect:

- 2 - **DependencyList** ([giá_trị_1, giá_trị_2]):

- **useEffect** sẽ chạy 1 lần đầu tiên sau khi component render và sẽ chạy lại khi một trong các giá trị trong mảng phụ thuộc thay đổi.
 - Thường được sử dụng trong việc setup tự động khi 1 state thay đổi sẽ thực thi lại 1 logic với state đó.
 - Ví dụ: Các page detail hoặc search với các param và queryParams (phân router sẽ demo chi tiết)

```
useEffect(() => {  
  // Chạy một lần khi component mount  
  console.log('Component mounted');  
}, [dependency1, dependency2,...]); // Dependency list rỗng
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

HOOK USEEFFECT

- Các trường hợp của useEffect:

3 - useEffect với cleanup function

- Sử dụng cleanup function để dọn dẹp tài nguyên khi component unmount hoặc trước khi chạy effect tiếp theo.

```
import React, { useState, useEffect } from 'react'; 6.9k (gzipped: 2.7k)

const TimerEffect = () => {
  const [count, setCount] = useState(0);
  useEffect(() => {
    // Thiết lập một interval để tăng count mỗi giây – sau này sẽ ứng với 1 số api chạy ngầm
    const timer = setInterval(() => {
      setCount((prevCount) => prevCount + 1);
    }, 1000);
    // Cleanup function để dọn dẹp interval khi component unmount
    return () => {
      clearInterval(timer);
      console.log('Timer cleared'); //clean up các sự kiện chạy ngầm tại đây
    };
  }, []); // Chạy một lần khi component mount

  return (
    <div>
      <p>Count: {count}</p>
    </div>
  );
}
export default TimerEffect;
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

HOOK USEEFFECT

- Các trường hợp của useEffect:

4 - Default không có dependency

- useEffect sẽ chạy sau mỗi lần render (ít dùng).

```
import React, { useState, useEffect } from 'react'; 6.9k (gzipped: 2.7k)

const TimerEffect = () => {
  const [count, setCount] = useState(0);
  useEffect(() => {
    // Thiết lập một interval để tăng count mỗi giây – sau này sẽ ứng với 1 số api chạy ngầm
    const timer = setInterval(() => {
      setCount((prevCount) => prevCount + 1);
    }, 1000);
    // Cleanup function để dọn dẹp interval khi component unmount
    return () => {
      clearInterval(timer);
      console.log('Timer cleared'); //clean up các sự kiện chạy ngầm tại đây
    };
  }, []); // Chạy một lần khi component mount

  return (
    <div>
      <p>Count: {count}</p>
    </div>
  );
}
export default TimerEffect;
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10



Form - validation



Để lấy dữ liệu từ form trong reactjs ta có thể sử dụng hook useState thông qua sự kiện onChange hoặc onInput thao tác với các trường id hoặc name của thẻ.

1

2

3

4

5

6

7

8

9

10

```
const FormDemo = () => {
  const [userLogin, setUserLogin] = useState({
    email: '',
    password: ''
  })

  const handleChange = (e) => {
    const {value, name} = e.target;
    setUserLogin({
      ...userLogin,
      [name]: value
    })
  }

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log(userLogin)
  }
}
```

The screenshot shows a login interface with two input fields and a button. The first input field is labeled 'Email' and contains the value 'cybersoft@gmail.com'. The second input field is labeled 'Password' and contains the value '.....'. Below the inputs is a blue button with the text 'Đăng Nhập'.

```
<form onSubmit={handleSubmit} className="space-y-6">
  <div>
    <label htmlFor="email" className="block text-sm font-medium text-gray-700">Email</label>
    <input name="email" id="email" className="mt-1 block w-full px-3 py-2 bg-white border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm" onChange={handleChange}>
  </div>
  <div>
    <label htmlFor="password" className="block text-sm font-medium text-gray-700">Password</label>
    <input type="password" name="password" id="password" required className="mt-1 block w-full px-3 py-2 bg-white border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm" onChange={handleChange}>
  </div>
  <div>
    <button type="submit" className="w-full bg-indigo-600 text-white py-2 px-4 rounded-md shadow-sm hover:bg-indigo-700 focus:outline-none focus:ring-2 focus:ring-offset-2 focus:ring-indigo-500">Đăng Nhập</button>
  </div>
</form>
```



Form - validation



Để xử lý validation trong react js ta có thể chia thành 2 dạng:

- + kiểm tra bắt buộc nhập (required): Kiểm tra value với giá trị rỗng.

- + kiểm tra định dạng (regex): Kiểm tra value với biểu thức regex dựa vào data-attribute của thẻ.

1

2

3

4

5

6

7

8

9

10

```

const [userLogin, setUserLogin] = useState({
  values: {
    email: '',
    password: ''
  },
  errors: {
    email: '',
    password: ''
  }
})

const handleChange = (e) => {
  let {name,value} = e.target;
  let values = {...userLogin.values}; //Clone giá trị hiện ra 1 biến mới
  let errors = {...userLogin.errors}; // Clone lỗi hiện tại ra 1 biến mới
  let dataType = e.target.getAttribute('data-type');
  values[name] = value; //Gán mới thuộc tính value
  let messError = '';
  if(value[name] === '') {
    messError = `${name} is required!`;
  }else {
    //Check các format regex
    if(dataType){
      switch(dataType){
        case 'email' :
          let regexEmail = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$/;
          if(!regexEmail.test(values[name])) {
            messError = 'email is invalid!';
          };break;
      }
    }
  }
  //Gán lỗi trên trường name sau khi đã xử lý
  errors[name] = messError;
  //Sau khi đã xử lý values và errors gán vào state cả values và errors mới
  setUserLogin({
    values:values,
    errors:errors
  })
}

```

Clone các giá trị từ state ra để xử lý theo name và value

Lấy giá trị data-type (để xác định loại lôï cần xử lý)

Nếu lôï khác rỗng xem data-type của thẻ đó có hay không nếu có sẽ xem loại nào để dùng regex tương ứng kiểm tra

```

<div>
  <label htmlFor="email" className="block text-sm font-medium text-gray-700">Email</label>
  <input data-type="email" name="email" id="email" className="mt-1 block w-full px-2 bg-white border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm" onChange={handleChange} />
  {userLogin.errors.email && <p className='text-red-500'>{userLogin.errors.email}</p>}
</div>

```

Sau khi xử lý lôï và giá trị mới ta sẽ gán lại vào state



Form - validation



Sử dụng với thư viện formik - yup

Ngoài ra ta có thể dùng các thư viện như react form hay formik yup để rút ngắn phần code và xử lý.
Ngoài ra còn có các thư viện như antd hay material vừa hỗ trợ ui vừa hỗ trợ form.

1

2

3

4

5

6

7

8

9

10

```
yarn add formik yup
```

```
npm i formik yup
```

```
import React from 'react'; 6.9k (gzipped: 2.7k)
import { useFormik } from 'formik'; 38k (gzipped: 11.8k)
import * as Yup from 'yup'; 43k (gzipped: 13.3k)

const FormikYupDemo = () => {
  const formik = useFormik({
    initialValues: {
      email: '',
      password: ''
    },
    validationSchema: Yup.object({
      email: Yup.string()
        .email('Email is invalid')
        .required('Email is required'),
      password: Yup.string()
        .required('Password is required')
    }),
    onSubmit: values => {
      console.log(values);
    }
  });
}
```

Sử dụng formik để lấy giá trị từ form
cùng như kết hợp yup để xử lý lõi

Câu lệnh hiển thị lõi

```
<div>
  <label htmlFor="email" className="block text-sm font-medium text-gray-700">Email</label>
  <input
    type="email"
    name="email"
    id="email"
    className="mt-1 block w-full px-3 py-2 bg-white border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm"
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}
    value={formik.values.email}
  />
  {formik.touched.email && formik.errors.email ? (
    <p className="text-red-500">{formik.errors.email}</p>
  ) : null}
</div>
```

Login



Form - validation



Sử dụng với thư viện formik - yup

Ngoài ra ta có thể dùng các thư viện như react form hay formik yup để rút ngắn phần code và xử lý.
Ngoài ra còn có các thư viện như antd hay material vừa hỗ trợ ui vừa hỗ trợ form.

1

2

3

4

5

6

7

8

9

10

```
yarn add formik yup
```

```
npm i formik yup
```

```
import React from 'react'; 6.9k (gzipped: 2.7k)
import { useFormik } from 'formik'; 38k (gzipped: 11.8k)
import * as Yup from 'yup'; 43k (gzipped: 13.3k)

const FormikYupDemo = () => {
  const formik = useFormik({
    initialValues: {
      email: '',
      password: ''
    },
    validationSchema: Yup.object({
      email: Yup.string()
        .email('Email is invalid')
        .required('Email is required'),
      password: Yup.string()
        .required('Password is required')
    }),
    onSubmit: values => {
      console.log(values);
    }
  });
}
```

Sử dụng formik để lấy giá trị từ form
cùng như kết hợp yup để xử lý lỗi

Cáu hình hiển thị các sự kiện onChange
và onBlur cũng như hiển thị lỗi

```
<div>
  <label htmlFor="email" className="block text-sm font-medium text-gray-700">Email</label>
  <input
    type="email"
    name="email"
    id="email"
    className="mt-1 block w-full px-3 py-2 bg-white border border-gray-300 rounded-md shadow-sm placeholder-gray-400 focus:outline-none focus:ring-indigo-500 focus:border-indigo-500 sm:text-sm"
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}
    value={formik.values.email}
  />
  {formik.touched.email && formik.errors.email ? (
    <p className="text-red-500">{formik.errors.email}</p>
  ) : null}
</div>
```

Login



Form - validation



Bài tập

Xây dựng Component chứa form đăng ký (Register Form) như sau:

- **Nội dung form bao gồm:**
 - userName
 - email
 - Gender (true | false) Sử dụng input radio đặt name giống nhau value true(name) false (nữ)
 - Country (USA, Canada, UK, Australia) => khi select lấy value
 - password
- **Yêu cầu**
- **Xây dựng giao diện dùng tailwindcss**
 - userName: không được bỏ trống
 - email: Không được bỏ trống và phải đúng định dạng email
 - gender: check 1 trong 2 mặc định
 - country: quốc gia default giá trị đầu tiên(USA, Canada, UK, Australia).
 - password:Không được để trống. Ít nhất 8 ký tự.

1

2

3

4

5

6

7

8

9

10

User Registration

Username

Email

Gender

Male Female

Country

Australia

Password

Register

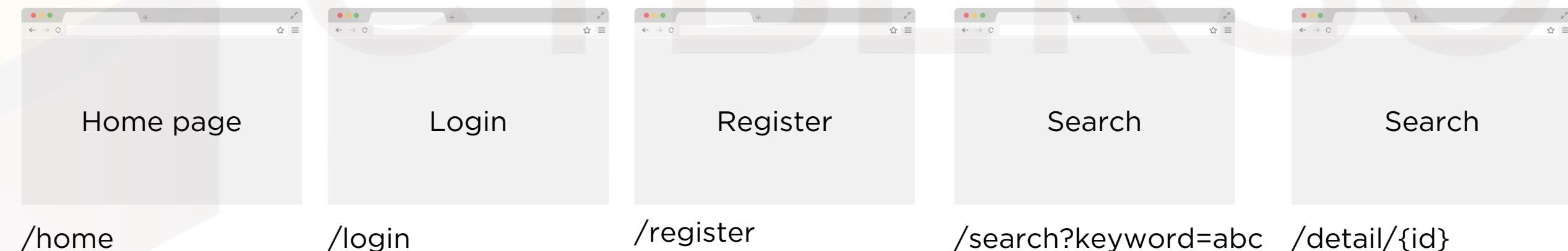
Các khái niệm cơ bản React functional component



Router base

Routing là gì ?

- Routing là cơ chế trong single page giúp ta chuyển đổi qua lại giữa các component
- Để sử dụng được routing với reactjs, ta cần package hỗ trợ đó là React-router-dom
- Tiến hành cài đặt: **npm install --save react-router-dom**
- Sử dụng: tại app.js, ta đang có 2 component không thể hiện cùng lúc, đó là danh sách khóa học, và danh sách sinh viên . Do đó ta sẽ dùng routing để quản lý, dựa theo đường dẫn url để hiển thị component tương ứng.



1

2

3

4

5

6

7

8

9

10

Các khái niệm cơ bản React functional component



Router base

- Cấu hình cở bản:

- Sử dụng Component BrowserRouter bao phân nội dung muốn hiển thị khi path trùng với url.
- Thẻ Route cấu hình path trùng với url thì sẽ hiển thị component đó.
- Nếu component không được đặt trong thẻ Route thì luôn được hiển thị.

```
import React from 'react';
import Footer from './components/Footer/Footer';
import Header from './components/Header/Header';
import About from './pages/About/About';
import Contact from './pages/Contact/Contact';
import Home from './pages/Home/Home';

import { BrowserRouter, Route } from 'react-router-dom'

function App() {
  return (
    <BrowserRouter>
      <div className="mb-5">
        <Header />
      </div>
      <div className="p-5">
        <Route exact path='/home' component={Home} />
        <Route exact path='/contact' component={Contact} />
        <Route exact path='/about' component={About} />
      </div>
      <Footer />
    </BrowserRouter>
  );
}

export default App;
```

1

2

3

4

5

6

7

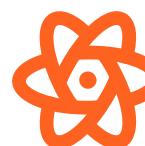
8

9

10



Các khái niệm cơ bản React functional component



Router base

- Nested routing

React-router-dom v.6+ cho phép ta định tuyến các route lồng nhau nhằm quản lý url cuộn như các giao diện 1 cách dễ dàng hơn nhìn vào hình bên dưới trong đó:

Routes: chứa một tập hợp các `<Route>` và đảm bảo rằng chỉ có một trong các `<Route>` con của nó được render tại một thời điểm, dựa trên đường dẫn URL hiện tại.

path="*": định nghĩa cho các đường dẫn không phù hợp với bất kì path nào trong Routes.

Route: đại diện cho 1 đường dẫn tương ứng `<Route />`, các đường dẫn con bên trong sẽ được load trên giao diện của component cha tại phần `<Outlet />`

Index: xác định route mặc định load ra trên `<Outlet />` của layout

Navigate component: Chuyển hướng về `path name` tương ứng

```
const HomeLayout = () => {
  return (
    <>
      <Header />
      <div className='content' style={{minHeight:800}}>
        <Outlet />
      </div>
    </>
  )
}

export default HomeLayout
```

```
function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path='/' element={<HomeLayout />}>
          <Route index element={<Home />}></Route>
          <Route path="about" element={<About />}></Route>
          <Route path="register" element={<Register />}></Route>
          <Route path="login" element={<Login />}></Route>
          <Route path="products" element={<Products />}></Route>
          <Route path="search" element={<Search />}></Route>
          <Route path="detail" >
            <Route path=':id' element={<ProductDetail />}></Route>
          </Route>
          <Route path='*' element={<Navigate to={'/'} />}></Route>
        </Route>
      </Routes>
    </BrowserRouter>
  )
}

export default App
```

1

2

3

4

5

6

7

8

9

10

Các khái niệm cơ bản React functional component



Router base

- Nested routing

Bài tập:

Xây dựng layout component và tổ chức nesting route cho các trang như hình:

- product-management (Component layout chứa phân outlet)
 - index (ListProduct) - component sẽ được load mặc định với link **/product-management**
 - create-product(CreateProductComponent) - component sẽ được load với link **/product-management/create-product**
 - update-product(UpdateProductComponent) - component sẽ được load với link **/product-management/update-product**

Route: **/product-management**

ProductLayoutComponent

The screenshot shows a dashboard with a sidebar containing a 'Products' button. The main content area is a table listing three products: Product 3, Product 2, and Product 1. Each row includes columns for NAME, IMG, PRICE, and TYPE, along with Edit, Delete, and View detail buttons. At the bottom, it shows 'Showing 1 to 3 of 3 results' and a 'Per page: 10' dropdown.

	NAME	IMG	PRICE	TYPE	
<input type="checkbox"/>	Product 3		4999	A	Edit Delete View detail
<input type="checkbox"/>	Product 2		3999	B	Edit Delete View detail
<input type="checkbox"/>	Product 1		2999	A	Edit Delete View detail



Các khái niệm cơ bản React functional component



Router base

1

2

3

4

5

6

7

8

9

10

Link Component: <NavLink> (Hoặc <Link>)

- Thư viện react router dom hỗ trợ ta thẻ **NavLink** thay thế cho thẻ **<a>** với **href** đổi thành **to**, giúp cho ta có thể chuyển đổi qua lại giữa các trang mà không cần load lại toàn bộ html của trang đó.
- Ngoài ra **<NavLink>** còn cho phép ta custom link css, khi người dùng gõ đúng đường dẫn thì thẻ **NavLink** sẽ được active class hoặc style thông qua thuộc tính tương ứng **activeClass**, **activeStyle**.

```
<NavLink  
      exact  
      activeStyle={{color:red}}  
      activeClassName="my-link">  
    Cyberlearn.vn  
</NavLink>
```

Các khái niệm cơ bản React functional component



Router base

1

2

3

4

5

6

7

8

9

10

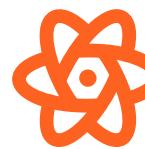
Hook của React Router trong Functional Components

Các hook useHistory, useRouteMatch, và useLocation được sử dụng để điều hướng, lấy thông tin route hiện tại và quản lý URL trong các functional component.

- useNavigate
- useParams
- useSearchParams
- useMatch
- useLocation



Các khái niệm cơ bản React functional component



Router base

- **useNavigate**

useNavigate cho phép ta chuyển hướng trang trong 1 xử lý. Tương tự thẻ link tuy nhiên useNavigate dùng để sử dụng trong các event (bao gồm các kiểm tra if else ... mới điệu hướng thay vì như thẻ link khi ta click sẽ chuyển đến đường dẫn đích liền).

```
import React from 'react';
import { useNavigate } from 'react-router-dom';

const NavigateExample = () => {
  const navigate = useNavigate();

  const goToHome = () => {
    navigate('/home');
  };

  return (
    <div>
      <h1>Navigate Example</h1>
      <button onClick={goToHome}>Go to Home</button>
    </div>
  );
};

export default NavigateExample;
```

```
<NavLink to="/product/1" state={{fromHome:true}}> New product </NavLink>
const goToProduct = () => {
  navigate('/product/1', { state: { fromHome: true } });
};
```

Có thể truyền thêm State thông qua navigate hoặc link. State không hiển thị trên url

1

2

3

4

5

6

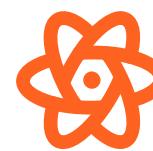
7

8

9

10

Các khái niệm cơ bản React functional component



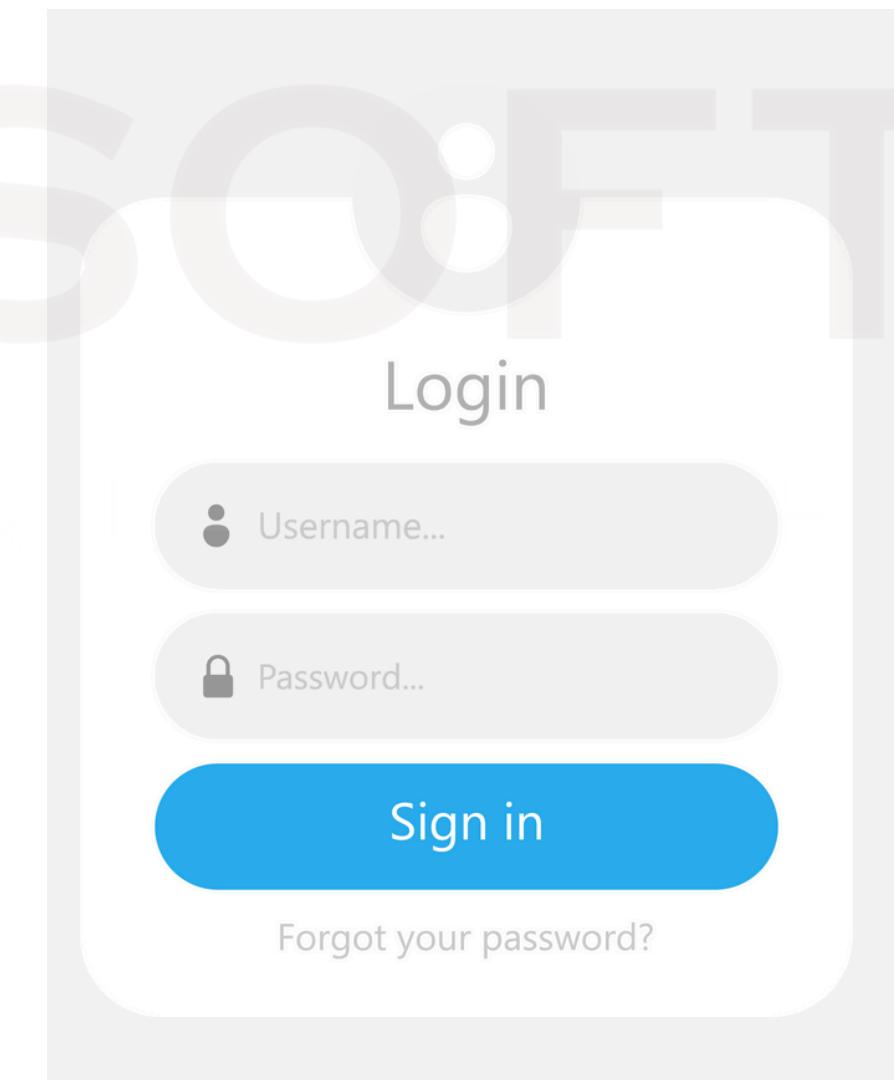
Router base

- **useNavigate**

Bài tập: Xây dựng trang login, profile, forgot password hãy xây dựng chức năng khi đăng nhập đúng thì chuyển hướng qua profile và khi đăng nhập sai thì chuyển qua forgot password.
(Lấy ví dụ: username: cybersoft, password: cybersoft)

```
<Routes>
  <Route path="/login" element={<Login />} />
  <Route path="/profile" element={<Profile />} />
  <Route path="/forgotpass" element={<ForgotPassword />} />
</Routes>
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10





Các khái niệm cơ bản React functional component



Router base

- **useParams**

useParams là một hook trong thư viện react-router-dom, được sử dụng để truy cập các tham số của URL trong ứng dụng React. Thường được sử dụng trong các trang chi tiết sản phẩm, chi tiết bài viết, chi tiết bài blog... dựa vào id trên thanh url

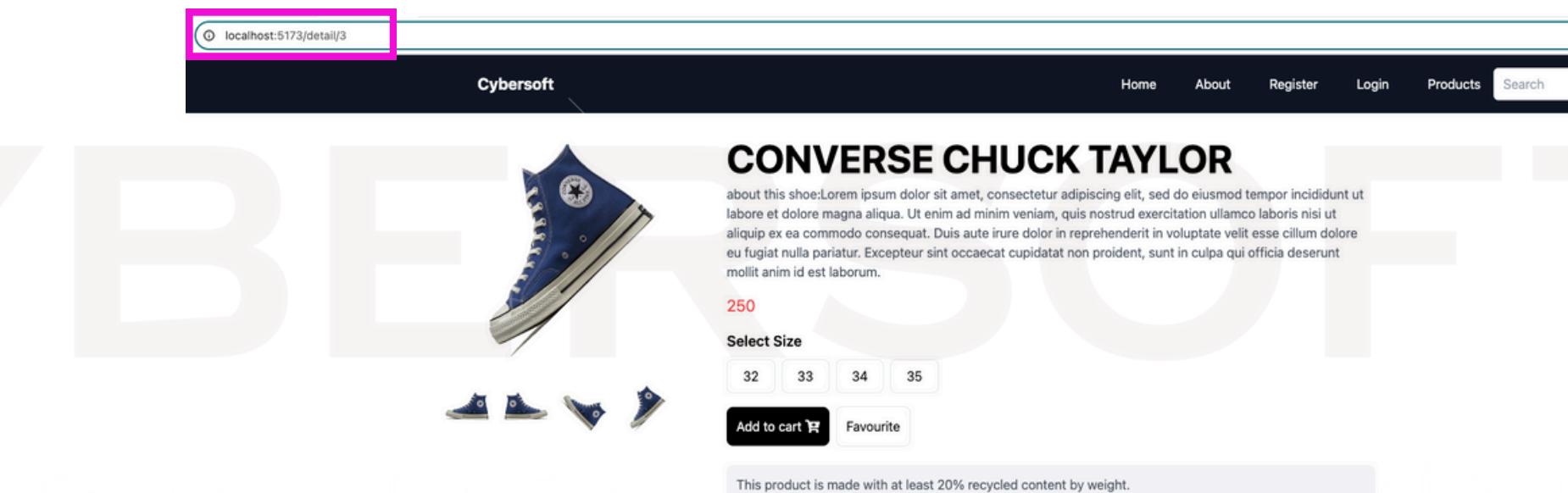
```
function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path=''' element={<HomeLayout />}>
          <Route index element={<Home />}></Route>
          <Route path="about" element={<About />}></Route>
          <Route path="register" element={<Register />}></Route>
          <Route path="login" element={<Login />}></Route>
          <Route path="products" element={<Products />}></Route>
          <Route path="search" element={<Search />}></Route>

        <Route path="detail" >
          <Route path=':id' element={<ProductDetail />}></Route>
        </Route>

        <Route path='*' element={<Navigate to={'/'}></Route>}></Route>
      </Routes>
    </BrowserRouter>
  )
}

export default App
```

AppComponent



Page Detail

```
const ProductDetail = (props) => {
  const { id } = useParams();
```

DetailComponent

1

2

3

4

5

6

7

8

9

10



Các khái niệm cơ bản React functional component



Router base

- **useParams**

useParam thường được kết hợp với useEffect trong 1 số trường hợp ví dụ như trong trang Detail chứa các liên kết dẫn đến chính trang Detail thì ta cần UseEffect chạy lại với id mới để cập nhật state.

1

2

3

4

5

6

7

8

9

10

id sẽ được lấy lại từ url sau mỗi lần render (hay còn gọi là global state)

```
<div className='w-2/3 mx-auto'>
  <h2 className='text-2xl font-bold mb-4'>Reated Product</h2>
  <div className='grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3 gap-4'>
    {productDetail.relatedProducts?.map((product, index) => (
      <div key={index} className='bg-white p-4 rounded-lg shadow-lg'>
        <img src={product.image} alt={product.name} className='w-full rounded-lg mb-4' />
        <h3 className='text-xl font-bold'>{product.name}</h3>
        <p className='text-gray-600'>{product.shortDescription}</p>
        <p className='text-2xl text-red-500 mt-2 font-bold'>{product.price} $</p>

        {/* Khi click vào link này thì chính component này sẽ render lại (vì vậy nếu để dependency []
         [] thì sẽ chỉ render 1 lần) => vì vậy phải cài đặt id param làm dependency để khi click
         link useEffect tự chạy lại với id mới */}
        <NavLink to={`/detail/${product.id}`} alt={product.title} className='bg-blue-500
          text-white py-2 px-4 rounded hover:bg-blue-600 inline-block mt-2'>View detail</NavLink>
      </div>
    )))
  </div>
</div>
```

Nội dung page detail sẽ link lại chính page detail với param mới thông qua thẻ NavLink

```
const ProductDetail = (props) => {
  const { id } = useParams();

  const [selectedSize, setSelectedSize] = useState(42);
  const [productDetail, setProductDetail] = useState({}); // Set state cho productDetail sau khi lấy dữ liệu từ api về
  const [selectedImage, setSelectedImage] = useState(); // Set state cho image khi ảnh chi tiết được chọn

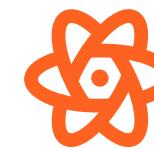
  const getProductDetailApi = async () => {
    const res = await fetch(`https://apistore.cybersoft.edu.vn/api/Product/getbyid?id=${id}`);
    const data = await res.json();
    setProductDetail(data.content);
    setSelectedImage(data.content.image);
  }

  useEffect(() => {
    getProductDetailApi()
  }, [id]) //Thêm dependency id để khi id(param url) thay đổi thì hàm getProductDetail sẽ tự chạy lại với id param mới
}
```

Cấu hình useEffect với dependency là id để khi id thay đổi useEffect sẽ thực thi lại với id mới



Các khái niệm cơ bản React functional component



Router base

- **useSearchParams**

useSearchParams là một hook trong thư viện react-router-dom phiên bản 6 trở lên, được sử dụng để truy cập và sửa đổi các tham số truy vấn (query parameters) trong URL. UseSearchParams thường được áp dụng trên các trang tìm kiếm mà không cần phải tải loại toàn bộ trang.

url: http://localhost:5173/product-management?keyword=iphone

```
const [searchParam, setSearchParam] = useSearchParams();
searchParam.get('keyword'); // lấy giá trị (iphone) thông qua param (keyword)
setSearchParams({keyword:'new_value'}); //gán giá trị lên param của url
```

1

2

3

4

5

6

7

8

9

10

Các khái niệm cơ bản React functional component



Router base

- **useSearchParams**

Ví dụ xây dựng tính năng tìm kiếm sản phẩm cho page Product(Sử dụng link /api/ProductApi/getAll)

1

The screenshot shows the ProductApi section in the Swagger UI. It lists several operations:

- GET /api/ProductApi/getall
- GET /api/ProductApi/get/{id}
- PUT /api/ProductApi/update/{id}
- POST /api/ProductApi/create
- DELETE /api/ProductApi/delete/{id}

Each operation is color-coded: GET in blue, PUT in orange, POST in green, and DELETE in red.

Link api: <https://apitraining.cybersoft.edu.vn/swagger/ui/index#/ProductApi>

Link dùng thực hành cho các ví dụ

2

3

4

5

6

7

8

9

10

The screenshot shows a dashboard titled "Dashboard" with a sub-section "Products". It displays a table of products with columns: NAME, IMG, PRICE, and TYPE. Three products are listed:

NAME	IMG	PRICE	TYPE
Product 3		4999	A
Product 2		3999	B
Product 1		2999	A

Each row has "Edit | Delete | View detail" links at the bottom. The footer indicates "Showing 1 to 3 of 3 results" and "Per page: 10".

Tái sử dụng lại layout từ bài tập nesting route

Hoặc có thể build giao diện khác tương tự

Các khái niệm cơ bản React functional component



Router base

- useSearchParams

```
const ProductList = () => {
  const [searchParam, setSearchParam] = useSearchParams(); //Sử dụng useSearchParams để lấy giá trị từ url hoặc
  //set giá trị ?param cho url (ví dụ:domain.com?param=abc)
  const keyword = searchParam.get('keyword');
```

```
const getProductByKeyword = async () => {
  if (keyword) {
    let res = await fetch(`https://apitraining.cybersoft.edu.vn/api/ProductApi/getall?keyword=${keyword}`);
    const data = await res.json();
    console.log(data);
    setProducts(data);
  }
}
useEffect(() => {
  //call api lần đầu tiên sau khi trang load và sẽ call lại mỗi khi url thay đổi giá trị param
  getProductByKeyword()
}, [keyword]);
```

1

2

3

4

5

6

7

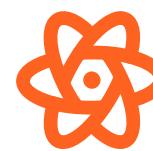
8

9

10

Gọi api lấy từ tham số và useEffect mỗi khi tham số url thay đổi thì call lại api.
Tương tự useParams.

Các khái niệm cơ bản React functional component



Router base

- **useMatch**

- useMatch là một hook cho phép bạn kiểm tra xem URL hiện tại có khớp với một pattern (mẫu) cụ thể hay không. Hook này hữu ích khi bạn cần xác định xem một route cụ thể có đang được truy cập hay không và lấy thông tin về `các tham số` trong URL.

- Lấy ví dụ ta có trường hợp

- Cân xây dựng 1 crud cho 1 nghiệp vụ (Ví dụ: Products). Ta cần xây dựng các page (route) productList, Create Product, EditProduct. Thay vì phải xây dựng 3 route như /products , /create, /edit ta chỉ cần xây dựng 2 route link (/products và 1 route link dùng cho cả edit và create)

	Name	Price	Action
<input type="checkbox"/>	Product 3	4999	
<input type="checkbox"/>	Product 2	3999	
<input type="checkbox"/>	Product 1	2999	

1

2

3

4

5

6

7

8

9

10



Các khái niệm cơ bản React functional component



Router base

- **useMatch**

Sử dụng useMatch cho trường hợp 1 component load cho cả from create và edit của ứng dụng.

```
import React from 'react'; 6.9k (gzipped: 2.7k)
import { useFormik } from 'formik'; 38k (gzipped: 11.8k)
import * as Yup from 'yup'; 43k (gzipped: 13.3k)
import { useMatch, useNavigate } from 'react-router-dom'; 6.3k (gzipped: 2.7k)

const ProductForm = ({ initialData = {} }) => {
  const match = useMatch('/product-management/product/:id');
  const navigate = useNavigate();
  const isEdit = !!match;

  const id = match?.params.id || null;
  const formik = useFormik({
    initialValues: {
      ...
    },
    validationSchema: Yup.object({
      ...
    }),
    onSubmit: (values) => {
      if (isEdit) {
        // Update product
        console.log('Updating product:', values);
      } else {
        // Create product
        console.log('Creating product:', values);
      }
      navigate('/products'); // Redirect after submit
    },
  });
}
```

Sử dụng useMatch để kiểm tra url
render ra các phần nội dung tương ứng
(Edit hoặc Create)

```
<Routes>
  <Route path='product-management' element={<ProductLayout />}>
    <Route index element={<ProductList />}></Route>
    <Route path='product' element={<ProductForm />}></Route>
    <Route path='product/:id' element={<ProductForm />}></Route>
  </Route>
</Routes>
```

Cả 2 route đều link đến
1 component

<http://localhost:5175/product-management/product/1>

Edit Product

ID

Name

Price

Image URL

Description

Type

Deleted

Update

1

2

3

4

5

6

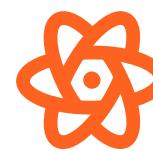
7

8

9

10

Các khái niệm cơ bản React functional component



Router base

- **useSearchParams**
- useMatch là một hook cho phép bạn kiểm tra xem URL hiện tại có khớp với một pattern (mẫu) cụ thể hay không. Hook này hữu ích khi bạn cần xác định xem một route cụ thể có đang được truy cập hay không và lấy thông tin về `các tham số` trong URL.
- Lấy ví dụ ta có trường hợp
 - Cân xây dựng 1 crud cho 1 nghiệp vụ (Ví dụ: Products). Ta cân xây dựng các page (route) productList, Create Product, EditProduct. Thay vì phải xây dựng 3 route như /products , /create, /edit ta chỉ cần xây dựng 2 route link (/products và 1 route link dùng cho cả edit và create)

	Name	Price	Action
<input type="checkbox"/>	Product 3	4999	<input checked="" type="button"/> Edit
<input type="checkbox"/>	Product 2	3999	<input checked="" type="button"/> Edit
<input type="checkbox"/>	Product 1	2999	<input checked="" type="button"/> Edit

Showing 1 to 3 of 3 results Per page 10 ▾

1

2

3

4

5

6

7

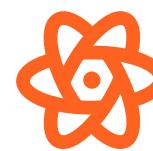
8

9

10



Các khái niệm cơ bản React functional component



Router base

- **useLocation**

useLocation cung cấp cho ta tất cả tham số trên url, link, state,...

```
▼ {pathname: '/product-management/product', search: '?key=abc', hash: '', state: null, key: 'default'} ⓘ
  hash: ""
  key: "default"
  pathname: "/product-management/product"
  search: "?key=abc"
  state: null
  ► [[Prototype]]: Object
```

Loaded new content

1

2

3

4

5

6

7

8

9

10

Các khái niệm cơ bản React functional component



Router base

Bài tập tổng hợp

QuanLyNhanVienApi

Show/Hide | List Operations | Expand Operations

GET /api/QuanLyNhanVienApi/LayDanhSachNhanVien

GET /api/QuanLyNhanVienApi/LayThongTinNhanVien

PUT /api/QuanLyNhanVienApi/CapNhatThongTinNhanVien

POST /api/QuanLyNhanVienApi/ThemNhanVien

DELETE /api/QuanLyNhanVienApi/XoaNhanVien

Cho link api sau: <https://apitraining.cybersoft.edu.vn/swagger/ui/index#/QuanLyNhanVienApi>

Tạo route và xây dựng giao diện cho api trên để quản lý thông tin nhân viên.

1

2

3

4

5

6

7

8

9

10

Redux

11

12

13

14

15

16

17

18

19

20

- Bản chất làm việc với React là việc truyền dữ liệu giữa các component và thay đổi state để re-render lại giao diện component
- Redux là thư viện cung cấp cho ta một store trung tâm, lưu trữ tất cả các state, từ component muôn thay đổi state chỉ cần truy cập tới store thay đổi.
- Qua ví dụ bài tập giờ hàng trên cho thấy để lưu trữ nguồn dữ liệu thay đổi (render lại giao diện mỗi khi setState được gọi).
- Ta phải xác định component nào là nơi chứa dữ liệu thay đổi binding và phải xem xét thêm điều kiện nó có chứa nút xử lý (hoặc bắt kể event nào) để gọi hàm setState không? Ta phải chọn 1 component nào chứa cả 2 yếu tố trên, vì vậy dẫn đến việc truyền dữ liệu từ props cha sang con nhiều cấp. Redux ra đời để khắc phục tình trạng đó, nó chỉ xét store (nơi chứa tất cả các state của component này được lưu tập trung) với component mà không phụ thuộc props từ component cha

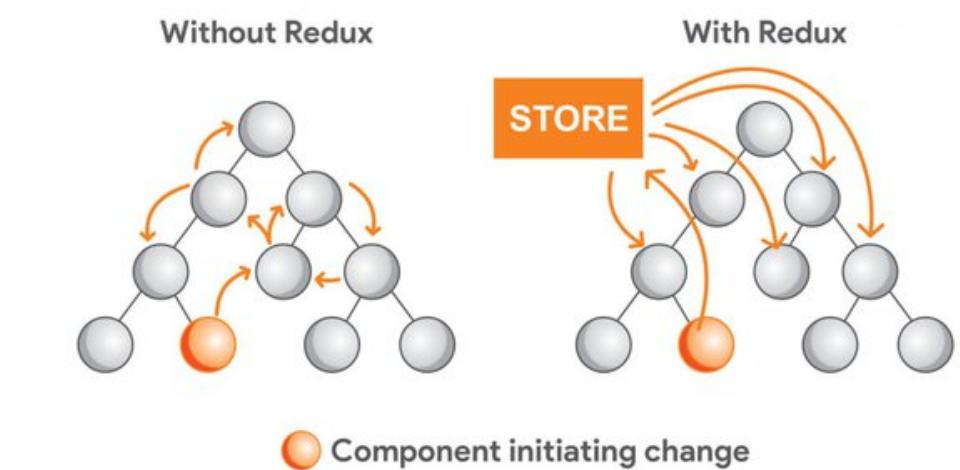
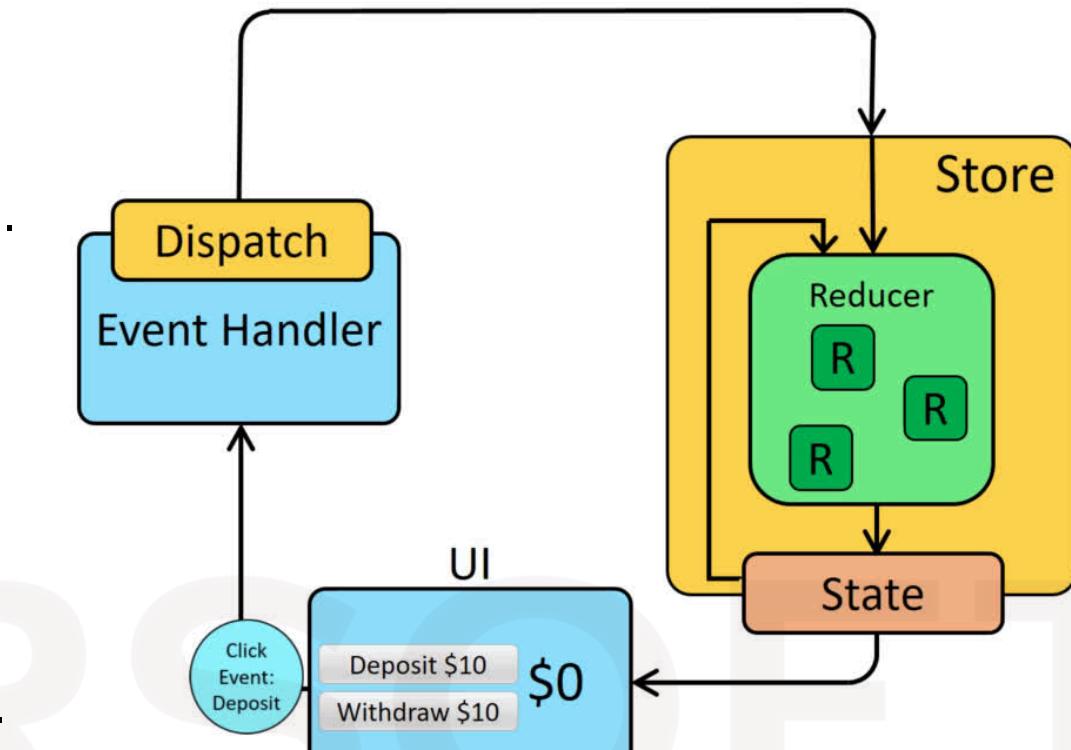
Cài đặt

1. CÀI ĐẶT @REDUXJS/TOOLKIT

```
npm i @reduxjs/toolkit
```

2. CÀI ĐẶT REACT-REDUX

```
npm i react-redux
```



Redux

11

12

13

14

15

16

17

18

19

20

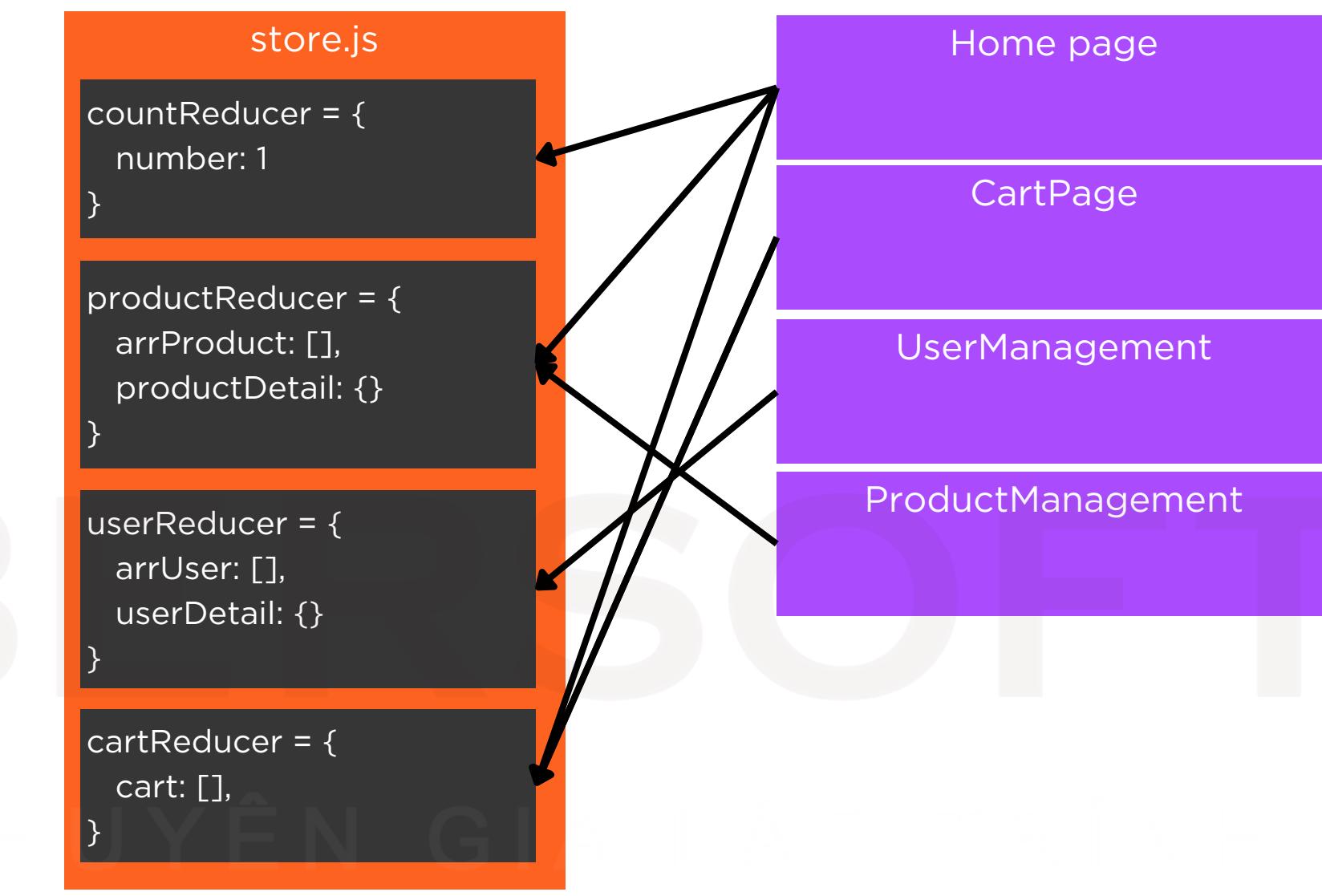
- Store là nơi chia sẻ các state cho toàn ứng dụng bất kì component nào kết nối với store cũng đều có thể thao tác với state mà không cần quan tâm đến thứ tự như Lifting state up.
- Cài đặt:

```
yarn add @reduxjs/toolkit react-redux
```

hoặc

```
npm i @reduxjs/toolkit react-redux
```

Cài đặt redux developer tool: <https://chromewebstore.google.com/detail/redux-devtools/lmhkpmbekcpmknklooeibfkpmffibljd>





Setup

11

12

13

14

15

16

17

18

19

20

```
import { configureStore } from '@reduxjs/toolkit'; 16.9k (gzipped: 6.2k)
import countReducer from './reducers/countReducer'
export const store = configureStore({
  reducer: {
    // Nơi liệt kê các state của ứng dụng
    countReducer: countReducer
  }
})
```

store.js

```
//setup redux
import { store } from './store/store'
import { Provider } from 'react-redux' 4.6k (gzipped: 1.9k)

function App() {
  return (
    <BrowserRouter>
      <Provider store={store}>
        <Routes> ...
        </Routes>
        <Routes> ...
        </Routes>
        <Provider>
          <BrowserRouter>
            ...
          </BrowserRouter>
        
```

app.js

Các hook chính sử dụng trong redux

- useSelector
- useDispatch

• useSelector:

- Dùng để lấy dữ liệu từ store về dựa trên tên thuộc tính reducer trên store.js

• useDispatch

- Dùng để đưa dữ liệu lên store thông qua action payload
- action_payload = {
 type:'action_name',
 payload: giá_trị
}

Ví dụ: Tăng giảm số lượng

11

12

13

14

15

16

17

18

19

20

```
import { configureStore } from '@reduxjs/toolkit'; 16.9k (gzipped: 6.2k)
import countReducer from './reducers/countReducer'
export const store = configureStore({
  reducer: {
    //Nơi liệt kê các state của ứng dụng
    countReducer: countReducer
  }
})
```

store.js

```
//rxslice: Để tạo redux slice
import { createSlice } from '@reduxjs/toolkit' 16.8k (gzipped: 6.2k)

const initialState = {
  count: 1 //state default
}
const countReducer = createSlice({
  name: 'countReducer', //reducer name
  initialState,
  reducers: {
    setCountAction: (state, action) => { //1
      state.count = state.count + action.payload;
    }
  }
);
export const [setCountAction] = countReducer.actions
/*
(1) hàm setCountAction tạo ra
actionPayload = {
  type: 'countReducer/setCountAction',
  payload: 1 hoặc -1
}
*/
export default countReducer.reducer
```

countReducer.js

Cybersoft

[Home](#) [About](#) [Register](#) [Login](#) [Products](#)

Demo Increase decrease quantity

+ 1 -

UI

```
const DemoStateNumber = () => {
  const { count } = useSelector(state => state.countReducer); //Lấy dữ liệu từ store về
  const dispatch = useDispatch();
  const handleChangeQuantity = (quantity) => {
    //Tạo action thủ công
    // const action = {
    //   type: 'countReducer/setCountAction',
    //   payload: quantity
    // }
    // hoặc tạo action payload thông qua rxslice action 2 cách tương đương nhau
    const action = setCountAction(quantity)
    dispatch(action);
  }
}
```

```
return (
  <div className='container'>
    <h3 className='display-4'>Demo Increase decrease quantity</h3>
    <div className="flex items-center space-x-4">
      <button className='btn btn-danger' onClick={()=>{
        handleChangeQuantity(1);
      }}>+</button>
      <h1 className='fs-1'>{count}</h1>
      <button className='btn btn-primary' onClick={()=>{
        handleChangeQuantity(-1);
      }}>-</button>
    </div>
  </div>
)
```

Component

Biên soạn: Trương Tân Khải - CYBERSOFT.EDU.VN

Redux



- **Bài tập thực hành làm với redux**

Lorem ipsum dolor sit amet consectetur adipisicing elit. Ab, alias.

zoom in **zoom out**

- Xây dựng chức năng tăng giảm số lượng cho 2 button + và -
- Yêu cầu:
 - Xây dựng layout
 - Sử dụng redux để xử lý chức năng tăng giảm fontsize (fontSizeReducer)

11

12

13

14

15

16

17

18

19

20

Redux



- **Bài tập thực hành làm với redux**

11

12

13

14

15

16

17

18

19

20

Yêu cầu:

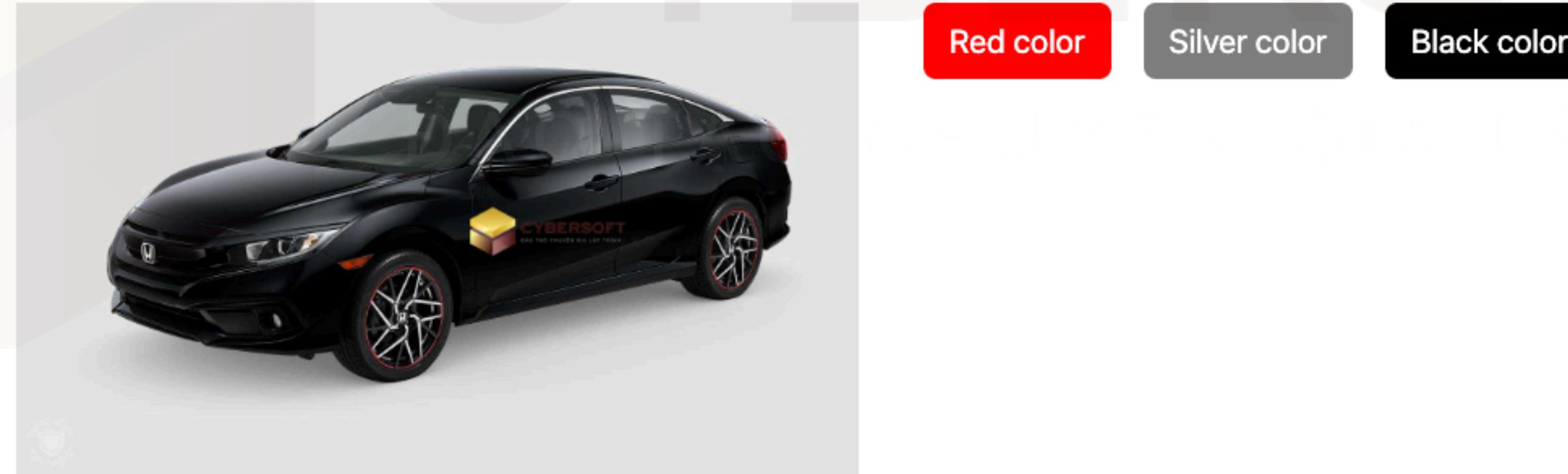
Xây dựng layout

Sử dụng redux để quản lý state hình ảnh (carReducer)

Resource:

https://drive.google.com/drive/folders/1xMdYF3L_1OQUO6I8Dm1AsXYhIJnr3iO2?usp=sharing

Change color car

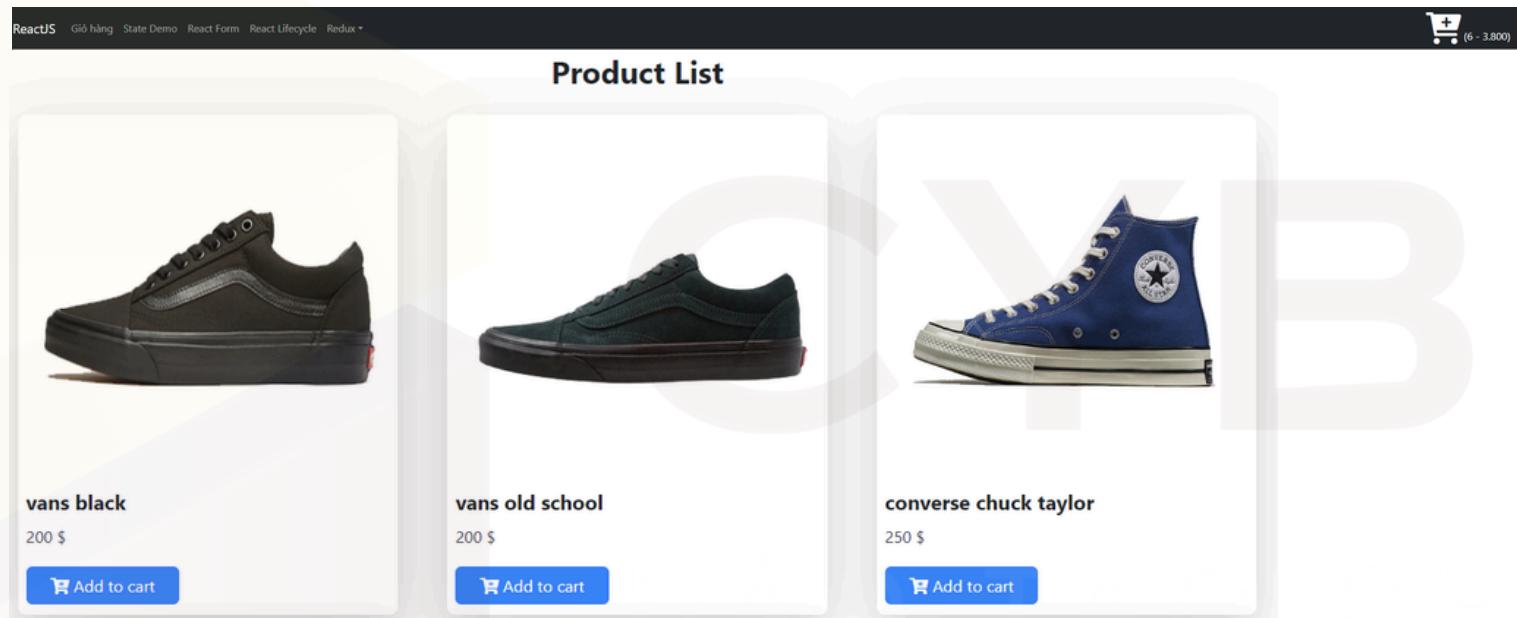


Redux



• BÀI TẬP TỔNG HỢP:

- Xây dựng /home page tạo route tương ứng (index) (call api lấy dữ liệu từ link sau): <https://apistore.cybersoft.edu.vn/api/Product>
- Xây dựng /cart page tổ chức reducer lưu trữ thông tin giỏ hàng
- Tại phần header kết nối với store để lấy số lượng sản phẩm trong giỏ hàng (hiển thị dạng icon)
- Tại phần /route cart page dùng useSelector để lấy dữ liệu từ store về



/home page hoặc tạo route tên khác

Giỏ hàng

Mã SP	Tên sản phẩm	Hình ảnh	Giá bán	Số lượng	Thành tiền	
1	VinSmart Live		570	<button>+</button> 4 <button>-</button>	2280	
2	Meizu 16Xs		760	<button>+</button> 2 <button>-</button>	1520	

/cart

11

12

13

14

15

16

17

18

19

20

Middleware redux

11

12

13

14

15

16

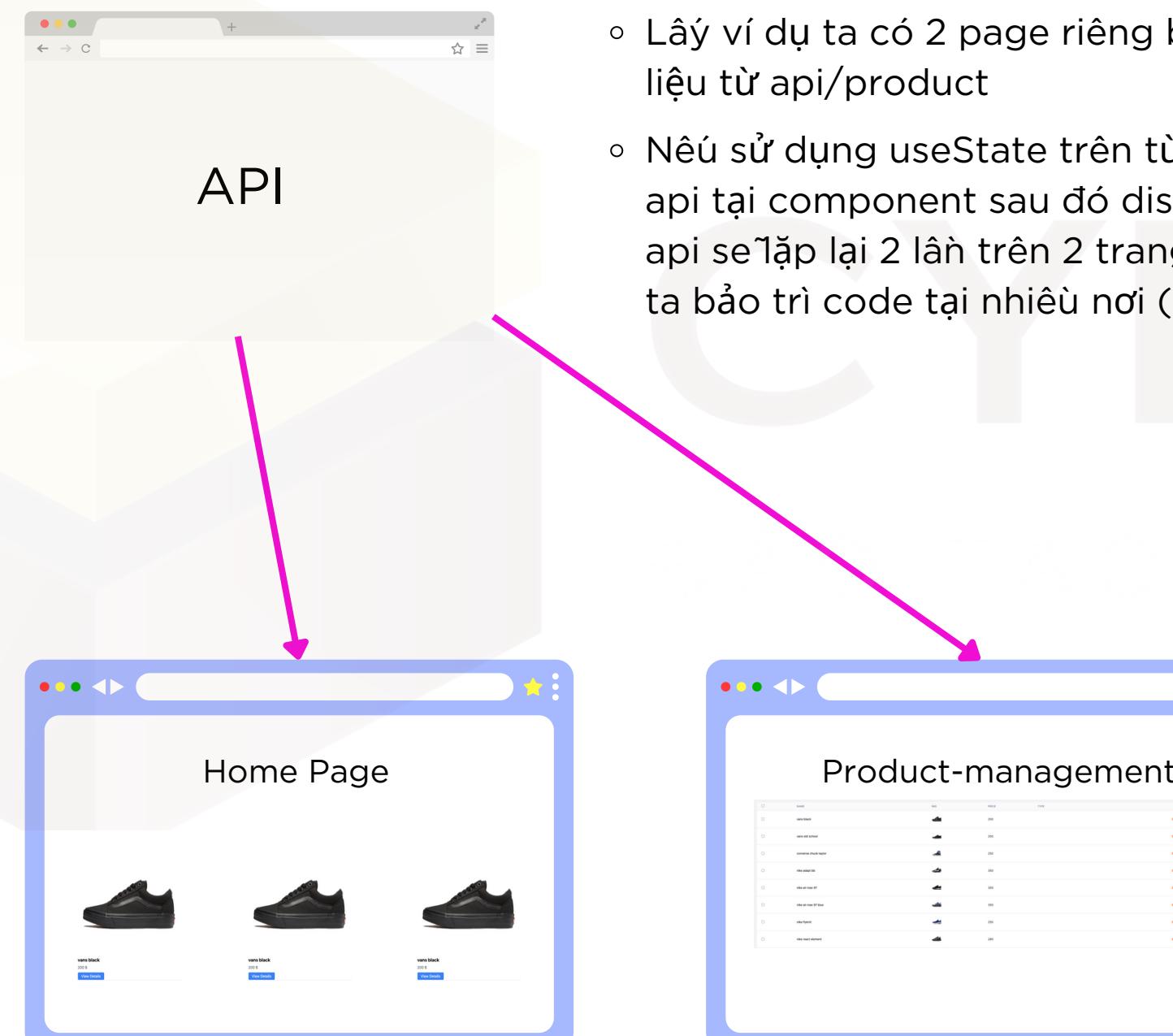
17

18

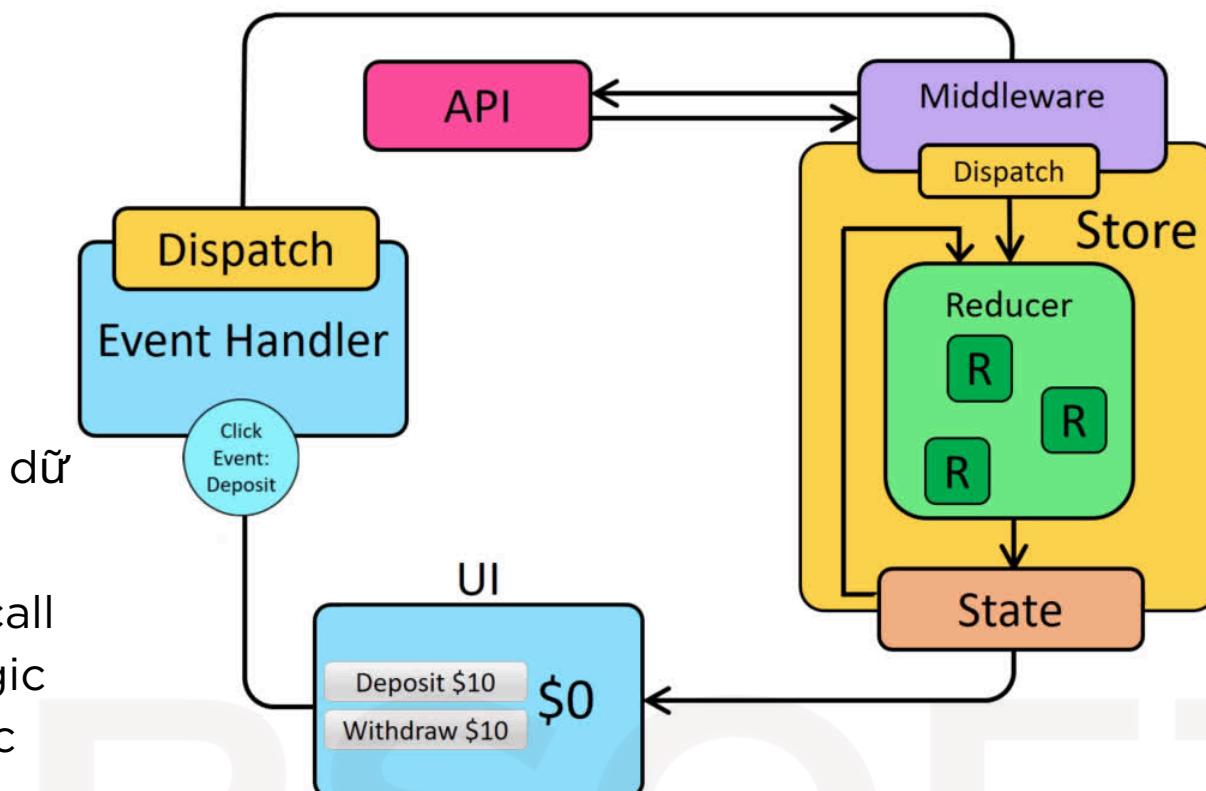
19

20

- Khi ta setup các dữ liệu là state được lấy từ api hoặc qua nhiều bước xử lý mới có được giá trị để đưa vào payload action đưa lên redux. Thì các nghiệp vụ này nếu thực hiện đi thực hiện lại trên nhiều component khác nhau thì sẽ dẫn đến việc lặp code. Vì vậy redux cung cấp cho ta thêm 1 middle ware là action async để xử lý cho trường hợp này.



- Lấy ví dụ ta có 2 page riêng biệt cần call api chung dữ liệu từ api/product
- Nếu sử dụng useState trên từng component hoặc call api tại component sau đó dispatch lên redux thì logic api sẽ lặp lại 2 lần trên 2 trang. Điều sẽ dẫn đến việc ta bảo trì code tại nhiều nơi (Đề bị thiêu sot)



Redux cung cấp ta actionAsync để tách các phần logic ra hàm riêng để tái sử dụng mà không phụ thuộc vào component.

Action Payload:

```
const action = {  
  type: 'action_name',  
  payload: data  
}
```

Action Async:

```
const actionAsync = (dispatch, getState) => {  
  // (B1) thực thi logic lấy được data  
  // (B2) lấy data tạo ra action payload  
  // (B3) Dùng hàm dispatch đưa dữ liệu lên reducer  
}
```

Middleware redux

- **setup:**

11

12

13

14

15

16

17

18

19

20

```
import { createSlice } from '@reduxjs/toolkit'; 16.8k (gzipped: 6.2k)
const initialState = {
  arrProduct: []
}
const productReducer = createSlice({
  name: 'productReducer',
  initialState,
  reducers: {
    setArrayProductAction : (state,action) => {
      state.arrProduct = action.payload
    }
  }
);
export const {setArrayProductAction} = productReducer.actions
export default productReducer.reducer
//----- Action async -----
export const getAllProductAsync = async (dispatch,getState) => {
  //Bước 1: Call api
  const res = await fetch(`https://apistore.cybersoft.edu.vn/api/Product`);
  const data = await res.json();
  //Bước 2: Sau khi lấy dữ liệu api tạo action payload
  const action = setArrayProductAction(data.content);
  //Bước 3: dispatch action payload
  dispatch(action);
}
```

productReducer

actionPayload

Tạo action async thực thi logic api sau đó có được dữ liệu đưa lên reducer

```
//Dùng useSelector để lấy arrProduct từ productReducer về
const {arrProduct} = useSelector(state => state.productReducer);
const dispatch = useDispatch();
const getAllProduct = async () => {
  //Import action async từ file ngoài vào component (logic api đã được tách ra file khác)
  const actionAsync = getAllProductAsync;
  //Dispatch để thực thi action async
  dispatch(actionAsync);
}
useEffect(()=>{
  //Gọi api
  getAllProduct();
},[])
```

Tạo action async thực thi logic api sau đó có được dữ liệu đưa lên reducer

getState là hàm của actionAsync cung cấp để tương tác với các state khác trên redux

Dispatch thực thi logic action async

Middleware redux - Closure Function

11

12

13

14

15

16

17

18

19

20

- Trường hợp action async có tham số (getById hay getByKeyword) thì ta truyền tham số như thế nào ??? Trước khi bàn về vấn đề đó ta sẽ tìm hiểu 1 khái niệm về **closure function**.
- Đối với function bình thường chỉ có thể truy cập các biến cục bộ của chính nó và các biến toàn cục. Sau khi hàm hoàn tất thực thi, các biến cục bộ của nó không còn tồn tại.
- Closure function là một hàm trong JavaScript có thể ghi nhớ và truy cập các biến từ phạm vi (scope) của hàm cha ngay cả sau khi hàm cha đã hoàn tất thực thi. Điều này có nghĩa là hàm con có thể "nhớ" được môi trường nơi nó được tạo ra.

Ví dụ về Function Bình Thường:

```
javascript
function normalFunction() {
  let message = "Hello, World!";
  console.log(message);
}

normalFunction(); // Hello, World!
```

Trong ví dụ này, biến `message` chỉ tồn tại trong phạm vi của `normalFunction`. Sau khi hàm hoàn tất thực thi, biến `message` không còn tồn tại.

Ví dụ về Closure:

```
javascript
function createGreeting(name) {
  return function() {
    console.log("Hello, " + name);
  };
}

const greetJohn = createGreeting("John");
greetJohn(); // Hello, John
```

Trong ví dụ này, hàm ẩn danh được trả về từ `createGreeting` vẫn có thể truy cập biến `name` từ phạm vi của `createGreeting`, ngay cả sau khi `createGreeting` đã hoàn tất thực thi.

Middleware redux - Closure Function

11

12

13

14

15

16

17

18

19

20

```
export const getProductDetailByIdAsync = (id) => {  
  
  return async (dispatch, getState) => {  
    //Bước 1: Call api  
    const res = await fetch(`https://apistore.cybersoft.edu.vn/api/Product/getbyid?id=${id}`);  
    const data = await res.json();  
    //Bước 2: Sau khi lấy dữ liệu api tạo action payload  
    const action = setProductDetailAction(data.content);  
    //Bước 3: dispatch action payload  
    dispatch(action);  
  }  
}
```

productReducer

Dùng closure để tạo ra action
async có tham số và dispatch

closure function

action async

Component

```
//Lấy tham số từ url  
const { id } = useParams();  
//Sử dụng useSelector để lấy productDetail từ redux về  
const {productDetail} = useSelector(state => state.productReducer);  
const dispatch = useDispatch();  
const getProductDetailApi = async () => {  
  //Dispatch thực thi action async  
  const actionAsync = getProductDetailByIdAsync(id);  
  dispatch(actionAsync);  
  
  //Set state cho image khi ảnh chi tiết được chọn  
  setSelectedImage(productDetail.image);  
}  
useEffect(() => {  
  getProductDetailApi()  
}, [id]) //Thêm dependency id để khi id(param url) thay đổi thì hàm  
getProductDetail sẽ tự chạy lại với id param mới
```

khai, 5 days ago • d

TANSTACK - REACT QUERY

11

12

13

14

15

16

17

18

19

20



Tanstack

Tanstack query





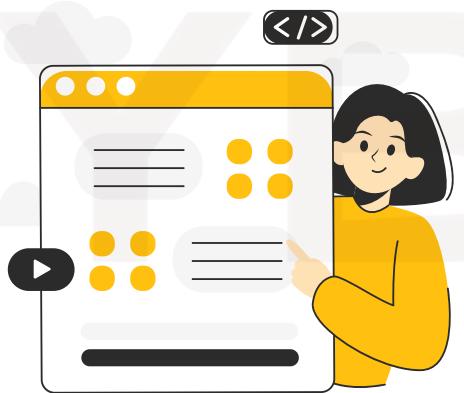
Tan stack query

1. TanStack Query là một thư viện JavaScript hiệu quả để quản lý trạng thái và đồng bộ hóa dữ liệu từ xa trong các ứng dụng React.
2. TanStack query cung cấp cho ta các hook để cache dữ liệu 1 cách dễ dàng và dễ sử dụng nhằm tối ưu giảm thiểu việc gọi api nhiều lần.

Quản lý dữ liệu server state hiệu quả:



Tối ưu hóa hiệu suất và UX:



Dễ dàng tích hợp và mở rộng:



- TanStack Query, hay còn gọi là React Query, cung cấp giải pháp mạnh mẽ và dễ sử dụng để quản lý dữ liệu từ server trong các ứng dụng React.
- Nó giúp giảm bớt sự phức tạp của việc xử lý trạng thái không đồng bộ, cải thiện hiệu suất và trải nghiệm người dùng bằng cách tự động hóa các tác vụ như fetching, caching, synchronizing và updating dữ liệu.

- Với cơ chế caching thông minh và khả năng background refetching, TanStack Query đảm bảo rằng người dùng luôn có được dữ liệu mới nhất mà không cần phải chờ đợi.
- Công cụ này còn cung cấp nhiều tính năng hữu ích như pagination, infinite scrolling và optimistic updates, giúp cải thiện trải nghiệm người dùng một cách đáng kể.

- TanStack Query tích hợp dễ dàng với các thư viện và framework khác như Axios, GraphQL, và Next.js, giúp nhà phát triển có thể triển khai nhanh chóng vào dự án hiện có.
- Với API linh hoạt và dễ sử dụng, cùng với cộng đồng hỗ trợ mạnh mẽ, TanStack Query là công cụ lý tưởng cho cả những dự án nhỏ lẫn các ứng dụng quy mô lớn.



Cài đặt tanstack query

1. CÀI ĐẶT THƯ VIỆN REACT QUERY

```
npm install @tanstack/react-query
```

2. ĐỂ DỄ DÀNG THEO DÕI VÀ DEBUG, BẠN CÓ THỂ CÀI ĐẶT REACT QUERY DEVTOOLS: (TƯƠNG TỰ REDUX DEVTOOL HAY REACT DEV TOOL)

```
npm install @tanstack/react-query-devtools
```

Import các thư viện react query từ tanstack

Import thư viện react query devtool

Sử dụng component QueryClientProvider bao bọc ứng dụng (Tương tự Provider redux)

Gắn vào component hiển thị icon devtool của react query

```
1 import ReactDOM from 'react-dom/client'  513 (gzipped: 321)
2 import App from './App'
3
4 //Cài đặt tanstack (react query)
5 import { QueryClient } from '@tanstack/react-query';  21.9k (gzipped: 6.5k)
6 import { QueryClientProvider } from '@tanstack/react-query';  1.3k (gzipped: 722)
7
8 //Cài đặt query devtool
9 import { ReactQueryDevtools } from '@tanstack/react-query-devtools';  291 (gzipped: 213)
10 const queryClient = new QueryClient();
11
12
13 ReactDOM.createRoot(document.getElementById('root')).render(
14   <QueryClientProvider client={queryClient}>
15     <App />
16     <ReactQueryDevtools initialIsOpen={false} />
17   </QueryClientProvider>
18 ,
19 )
```



Hooks và các thành phần

useQuery
useMutation
useQueryClient



useQuery

Hook này giúp fetch dữ liệu từ server và quản lý các trạng thái liên quan như loading, error và success. Nó tự động caching dữ liệu và hỗ trợ background refetching để đảm bảo dữ liệu luôn mới nhất.

Lấy ví dụ:

- Bởi cảnh ở đây ta có trang home page mỗi lần truy cập vào route này, ta lại fetch data từ api lại (bất kể dùng redux hay useState) api đều được gọi lại sau mỗi khi mount component. Tuy nhiên dữ liệu của trang này ít khi thay đổi (chỉ thay đổi khi phía admin thêm sản phẩm và điều này diễn ra không thường xuyên).
- Vậy có cách nào tối ưu việc load data này không ? Ta có thể cache lại data của danh sách sản phẩm này ?

useQuery sẽ giúp ta làm việc này

Cybersoft Shoes

Welcome to the Shoe Store Cybersoft

Find the best shoes for every occasion

Shop Now

Our Products

Vans Black

Vans Old School

Converse Chuck Taylor

Buy Now

Buy Now

Buy Now

Setup

Bên phải component sử dụng useQuery
để load thông tin sản phẩm từ api

Hàm gọi api để lấy data

hook useQuery

trả về các giá trị sau:

- **data**: dữ liệu return từ hàm api
- **error**: lỗi nếu fetch api bị lỗi
- **isLoading**: trạng thái pending của api (true, false)

• **Giá trị nhận vào của useQuery là object gồm:**

- **queryKey**: tên khoá của tập dữ liệu cache
- **queryFunction**: Hàm này sẽ được React Query gọi để lấy dữ liệu khi cần thiết, ví dụ như khi query lần đầu tiên được tạo hoặc khi cần refetch dữ liệu. khá giống với hàm của useEffect.
- **staleTime**: Thời gian mà dữ liệu được coi là mới và không cần refetch.
- **cacheTime**: Hết hạn, dữ liệu sẽ bị loại bỏ khỏi cache.

```
1  import React, { useState } from 'react'  6.9k (gzipped: 2.7k)
2  import axios from 'axios';  57.5k (gzipped: 21.1k)
3  import { useQuery } from '@tanstack/react-query';  11.9k (gzipped: 4.2k)
4
5  const fetchProducts = async () => {
6    const { data } = await axios.get('https://apistore.cybersoft.edu.vn/api/Product');
7    return data.content;
8  };
9
10 const DemoTanStack = () => {
11   const { data, error, isLoading } = useQuery({
12     queryKey: ['products'],
13     queryFn: fetchProducts,
14     staleTime: 1000 * 60 * 1, // Thời gian qui định dữ liệu cũ sau 1 phút
15     cacheTime: 1000 * 60 * 5, // Cache dữ liệu trong vòng 5 phút
16   });
17
18   if (isLoading) return <div>Loading...</div>;
19   if (error) return <div>Error: </div>;
```



useMutation - useQueryClient

Trong React Query được sử dụng để thực hiện các thao tác thay đổi dữ liệu (như POST, PUT, DELETE) và quản lý các trạng thái liên quan như loading, error, và success. Nó rất hữu ích khi bạn cần thực hiện các hành động thay đổi dữ liệu trên server và phản hồi ngay lập tức trong giao diện người dùng.

Lấy ví dụ:

- Ví dụ ta có 1 trang quản lý thông tin tất cả các user và các trang như create, update, delete user.
- Khi ta thêm 1 user hoặc thay đổi user ở các page còn lại ta muốn page quản lý (get All) thực hiện những cập nhật mới này.

useMutation và useQueryClient sẽ giúp ta làm việc này

User List				
ID	First Name	Last Name	Email	
1	George	Bluth	george.bluth@reqres.in	
2	Janet	Weaver	janet.weaver@reqres.in	
3	Emma	Wong	emma.wong@reqres.in	
4	Eve	Holt	eve.holt@reqres.in	
5	Charles	Morris	charles.morris@reqres.in	
6	Tracey	Ramos	tracey.ramos@reqres.in	

user-list

Sign Up

Email: khai@cybersoft.edu.vn
Password:
Name: khai truong
Gender: Male Female
Phone: 0909090909

User signed up successfully!

create-user

Setup

- Ta có 2 page với nhau:

- /users
- /create-user

- Tại page userList ta cấu hình call api
lấy dữ liệu về và sử dụng useQuery
để quản lý

Xây dựng hàm call api trả về userList

```
1 import React, { useState } from 'react'; Calculating...
2 import { useQuery } from '@tanstack/react-query'; 11.9k (gzipped: 4.2k)
3 import axios from 'axios'; 57.5k (gzipped: 21.1k)
4
5 const fetchUsers = async (page) => {
6   const { data } = await axios.get(`https://reqres.in/api/users?page=${page}`);
7   return data;
8 };
```

setup useQuery để quản lý data
của userList lấy từ api

```
10 const userList = () => {
11   const [page, setPage] = useState(1);
12   const { data, error, isLoading } = useQuery({
13     queryKey: ['users', page],
14     queryFn: () => fetchUsers(page),
15     keepPreviousData: true,
16   });
17
18   if (isLoading) return <div>Loading...</div>;
19   if (error) return <div>Error: {error.message}</div>;
20
21   const totalPages = data.total_pages;
22
23   return (
24     <div className="container mt-5">
25       <h2 className="text-center mb-4">User List</h2>
26       <table className="table table-striped">
27         <thead>
28           <tr>
29             <th>ID</th>
30             <th>First Name</th>
31             <th>Last Name</th>
32             <th>Email</th>
33           </tr>
34         </thead>
35         <tbody>
36           {data.data.map((user) => (
37             <tr key={user.id}>
38               <td>{user.id}</td>
39               <td>{user.first_name}</td>
40               <td>{user.last_name}</td>
41               <td>{user.email}</td>
42             </tr>
43           ))}
44         </tbody>
45       </table>
46     </div>
47   );
48 }
```

User List

ID	First Name	Last Name	Email
1	George	Bluth	george.bluth@reqres.in
2	Janet	Weaver	janet.weaver@reqres.in
3	Emma	Wong	emma.wong@reqres.in
4	Eve	Holt	eve.holt@reqres.in
5	Charles	Morris	charles.morris@reqres.in
6	Tracey	Ramos	tracey.ramos@reqres.in

Previous 1 2 Next

Nội dung component route /users chứa userList

Setup

- Ta có 2 page với như sau:

- /users
- /create-user

- Tại page createUser ta cần làm các bước sau:

- Thiết lập form với các thuộc tính name tương ứng dữ liệu api
- Sử dụng các thư viện form như react form hay formik hoặc tự code với es6 để lấy thông tin nhập liệu từ người dùng
- Trong sự kiện xử lý submit của form ta thiết lập gọi phương thức `mutation.mutate` hoặc `await mutation.mutateAsync`. truyền vào dữ liệu form. Mutation sẽ nhận giá trị từ form và xử lý thông qua `mutationFn`.
- Sau khi api thực thi xong thì hàm `onSuccess` của mutation sẽ chạy, tại đây ta có thể dùng thư viện hook `useQueryClient` để **nạp lại dữ liệu** cho `useQuery` thông qua `queryKey` bằng phương thức `queryClient.invalidateQueries`.

User List

ID	First Name	Last Name	Email
1	George	Bluth	george.bluth@reqres.in
2	Janet	Weaver	janet.weaver@reqres.in
3	Emma	Wong	emma.wong@reqres.in
4	Eve	Holt	eve.holt@reqres.in
5	Charles	Morris	charles.morris@reqres.in
6	Tracey	Ramos	tracey.ramos@reqres.in

Previous 1 2 Next

Xây dựng hàm api tại route CreateUser

```
1 import React, { useState } from 'react'; Calculating...
2 import { useMutation, useQueryClient } from '@tanstack/react-query'; 4.5k (gzipped: 2k)
3 import axios from 'axios'; 57.5k (gzipped: 21.1k)
4
5 const signupUser = async (newUser) => {
6   const { data } = await axios.post('https://apistore.cybersoft.edu.vn/api/Users/signup', newUser);
7   return data;
8 }
```

Trong nội dung component sử dụng hook `useMutation` và `useQueryClient` để thực hiện api và trigger render data từ `useQuery`

```
const queryClient = useQueryClient();
const mutation = useMutation({
  mutationFn: signupUser,
  onSuccess: () => {
    // Bạn có thể thêm hành động sau khi đăng ký thành công, ví dụ như chuyển hướng
    alert('User signed up successfully!');
    queryClient.invalidateQueries('users');
  },
});

const handleChange = (e) => {
  const { name, value, type, checked } = e.target;
  setFormData((prevFormData) => ({
    ...prevFormData,
    [name]: type === 'checkbox' ? checked : value,
  }));
};

const handleSubmit = (e) => {
  e.preventDefault();
  mutation.mutate(formData);
};
```

Nội dung component route /create-user chứa thông tin form nhập liệu với các trường



Configuration options

Một trong những điểm mạnh của React Query chính là khả năng tùy biến linh hoạt thông qua các tùy chọn cấu hình. Các tùy chọn này cho phép chúng ta kiểm soát chi tiết hành vi của truy vấn dữ liệu, từ việc tải lại dữ liệu khi cửa sổ lấy lại tiêu điểm đến việc xác định thời gian dữ liệu được giữ trong bộ nhớ cache.

- **Giá trị nhận vào của useQuery là object gồm:**
- **queryKey['fetchData', dependency]**: Khi **dependency** thay đổi, React Query sẽ tự động gọi lại hàm **fetchUserData** với **userId** mới và cập nhật dữ liệu.
- **queryFunction**: Hàm này sẽ được React Query gọi để lấy dữ liệu khi cần thiết, ví dụ như khi query lần đầu tiên được tạo hoặc khi cần refetch dữ liệu. khá giống với hàm của **useEffect**.
- **staleTime**: Thời gian mà dữ liệu được coi là mới và không cần refetch.
- **cacheTime**: Hết hạn, dữ liệu sẽ bị loại bỏ khỏi cache.
- **refetchOnReconnect**: Dữ liệu sẽ được tải lại khi mạng internet được khôi phục
- **refetchOnWindowFocus**: Khi click vào tab cửa sổ thì dữ liệu sẽ tải lại.
- **refetchInterval**: Tự động tải lại dữ liệu sau 1 khoảng time qui định
- **enabled**: kích hoạt truy vấn ngay lập tức
- **keepPreviousData**: Giữ lại dữ liệu cũ khi đang tải dữ liệu mới

```
1 import { useQuery, useQueryClient } from '@tanstack/react-query'; 11.9k (gzipped: 4.2k)
2 async function fetchDataFunction() {
3   const response = await fetch('https://api.example.com/data');
4   if (!response.ok) {
5     throw new Error('Network response was not ok');
6   }
7   return response.json();
8 }
9 function ExampleComponent({ userId }) {
10   const queryClient = useQueryClient();
11   const { data, error, isLoading, isPreviousData } = useQuery({
12     queryKey: ['fetchData', userId], // Khóa truy vấn bao gồm một phụ thuộc là userId
13     queryFn: fetchDataFunction, // Hàm truy vấn dùng để lấy dữ liệu từ API
14     refetchOnWindowFocus: true, // Dữ liệu sẽ được tải lại khi cửa sổ lấy lại tiêu điểm
15     staleTime: 5000, // Dữ liệu sẽ không được coi là cũ trong vòng 5 giây
16     cacheTime: 10000, // Dữ liệu sẽ được giữ trong bộ nhớ cache trong vòng 10 giây sau khi không còn sử dụng
17     refetchOnReconnect: true, // Dữ liệu sẽ được tải lại khi kết nối mạng được khôi phục
18     refetchInterval: 60000, // Tự động tải lại dữ liệu sau mỗi 60 giây
19     enabled: true, // Kích hoạt truy vấn ngay lập tức
20     keepPreviousData: true, // Giữ lại dữ liệu cũ khi đang tải dữ liệu mới
21   });
22   const handleRefresh = () => {
23     queryClient.invalidateQueries(['fetchData', userId]); // Vô hiệu hóa truy vấn 'fetchData' với userId
24   };
25   if (isLoading) return 'Loading...';
26   if (error) return 'An error has occurred: ' + error.message;
27   return (
28     <div>
29       {isPreviousData && <p>Loading new data...</p>}
30       {/* Render dữ liệu */}
31       {JSON.stringify(data)}
32       <button onClick={handleRefresh}>Refresh Data</button>
33     </div>
34   );
35 }
36
37 export default ExampleComponent;
```

Custom hook

11

12

13

14

15

16

17

18

19

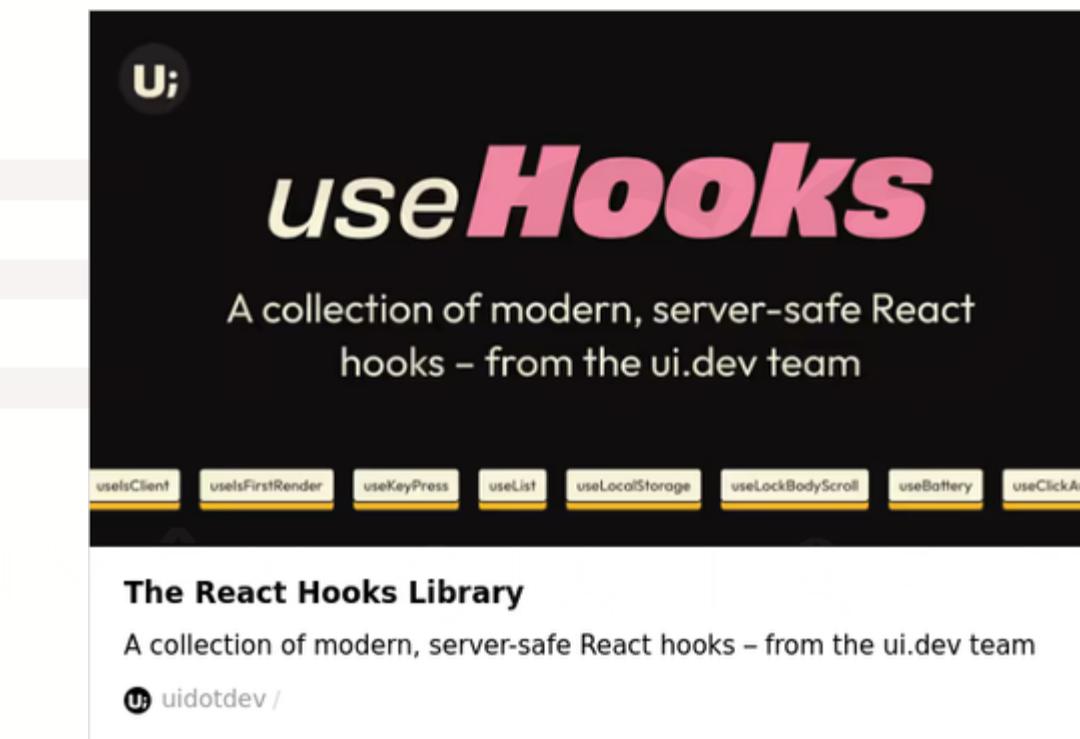
20

Custom Hook là một cách để bạn tạo ra các hàm có thể tái sử dụng giúp quản lý logic và trạng thái trong các component React. Nói 1 cách đơn giản custom hook sử dụng được tất cả các khái niệm của 1 function component (như state, props param, các hook khác ...) tuy nhiên giá trị trả về không phải là JSX.Element.

Tại sao phải sử dụng custom hook?

- Vì trong quá trình thiết kế 1 component ta có những xử lý logic lặp đi lặp lại như callapi, lấy pram từ url, hay lấy giá trị từ redux và dispatch, ... ta có thể đóng gói các logic này lại thành custom hook để thứ nhất việc viết code trở nên gọn gàng hơn, thứ 2 dễ bảo trì hơn khi có sự thay đổi.

```
1 import axios from 'axios'  56.8k (gzipped: 20.8k)
2 import React, { useEffect, useState } from 'react'  6.9k (gzipped: 2.7k)
3
4 const useGetApi = (url) => {
5     const [data, setData] = useState({})
6     const getApi = async () => {
7         const res = await axios({
8             url: url,
9             method: 'GET'
10        })
11        setData(res.data)
12    }
13    useEffect(()=>{
14        getApi();
15    },[])
16    return data
17}
18 export default useGetApi
```



Link custom hook tham khảo: <https://usehooks.com>

Ví dụ về 1 custom hook để call api đơn giản

Biên soạn: Trương Tân Khải - CYBERSOFT.EDU.VN

AUTHORIZATION - INTERCEPTOR

11

12

13

14

15

16

17

18

19

20

- Interceptor trong Axios là một chức năng cho phép bạn thực hiện các tác vụ trước khi một request được gửi đi hoặc sau khi một response được nhận về. Điều này rất hữu ích cho việc xử lý chung như thêm thông tin xác thực, log lỗi, hay thực hiện xử lý trước khi hiển thị dữ liệu.
- Lấy ví dụ 1 số api chúng ta cần phải chèn token vào thì mới có thể gọi được, tuy nhiên rất nhiều api sử dụng token này thì ta phải lặp đi lặp lại việc chèn token(dẫn đến khi token thay đổi thì phải sử dụng rất nhiều chỗ).
- Vì vậy axios interceptor hỗ trợ ta cấu hình request và response để xử lý các thông tin gửi về server cũng như xử lý các mã lỗi nhận về client 1 cách dễ dàng hơn

Thay vì gọi api dùng axios.get | post | put ... thì sau khi cấu hình interceptor ta có thể sử dụng instance axios tên api để thay thế

```
import axios from 'axios';

// Giả sử bạn có các thông tin userLogin và accessToken
const userLogin = 'yourUsername';
const accessToken = 'yourAccessToken';

// Tạo một instance của axios
const api = axios.create({
  baseURL: 'https://api.example.com', // URL cơ bản cho các request của bạn
});

// Thêm request interceptor
api.interceptors.request.use(
  function(config) {
    // Thêm userLogin và accessToken vào headers của request
    config.headers['userLogin'] = userLogin;
    config.headers['Authorization'] = `Bearer ${accessToken}`;
    return config;
  },
  function(error) {
    // Xử lý lỗi trước khi request được gửi đi
    return Promise.reject(error);
  }
);

// Thêm response interceptor nếu cần
api.interceptors.response.use(
  function(response) {
    // Xử lý dữ liệu response
    return response;
  },
  function(error) {
    // Xử lý lỗi từ response
    return Promise.reject(error);
  }
);

export default api;
```

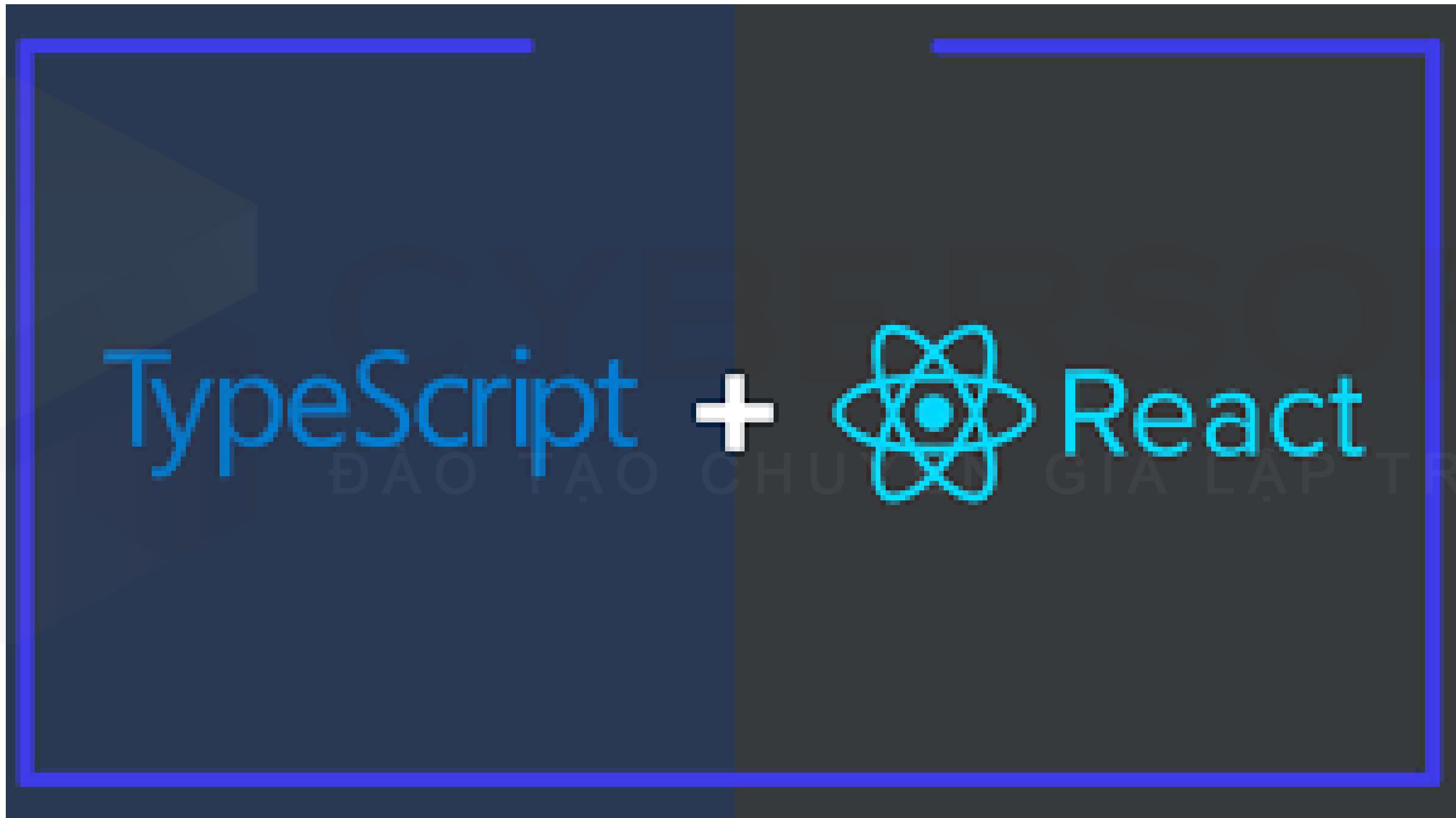
Các phần mở rộng

- **Antd**
- **Responsive**
- **Styledcomponent**
- **HOC - CONTAINER COMPONENT**
- ...

11
12
13
14
15
16
17
18
19
20

TypeScript - react (Xem trên pdf)

- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20



NEXTJS và các thành phần mở rộng

11

12

13

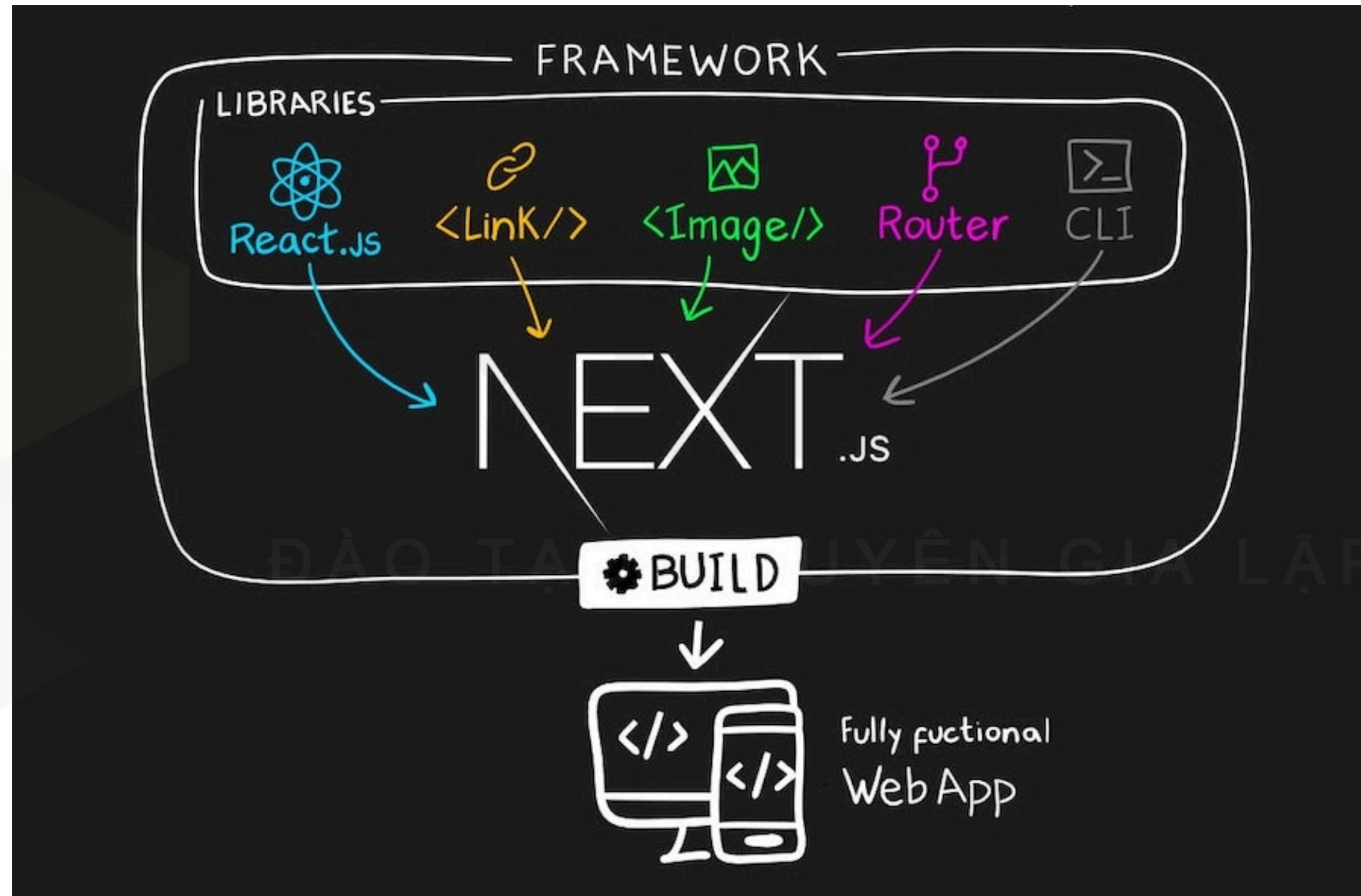
14

15

16

17

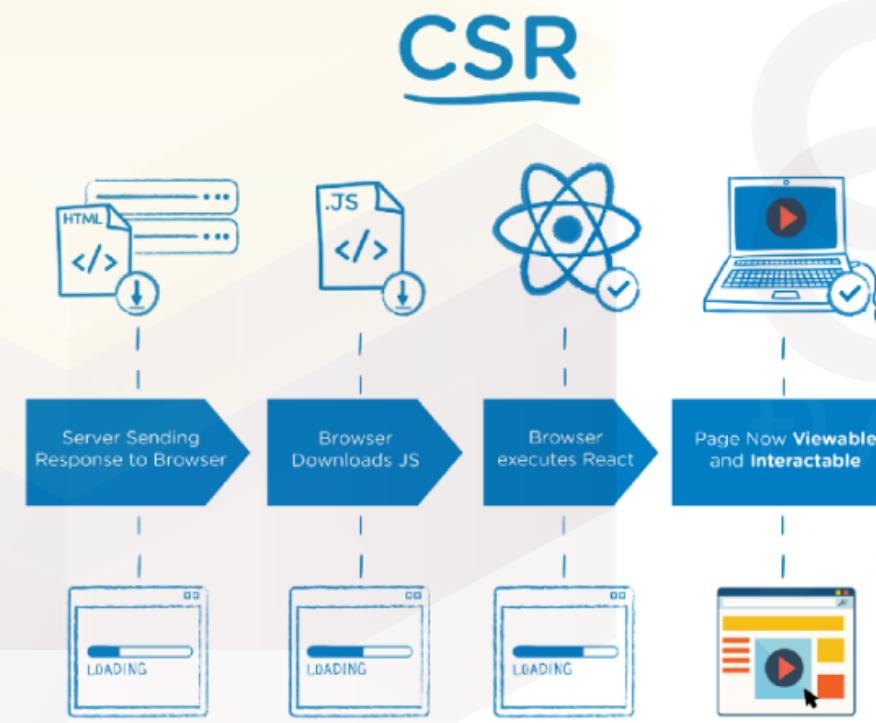
18+



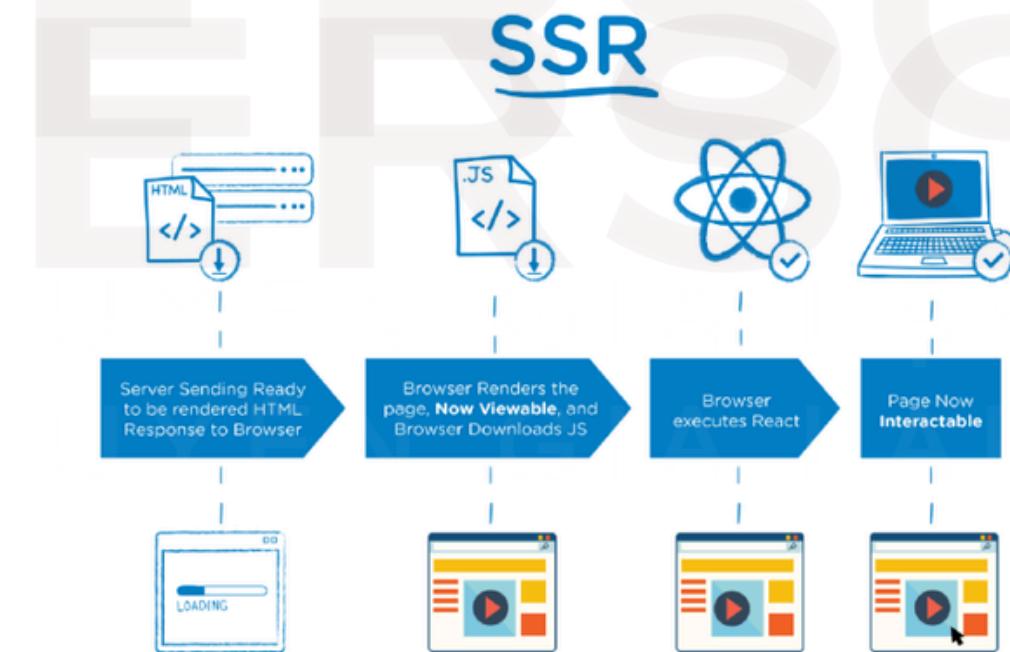


CSR & SSR

Bạn đã từng nghe về Server-Side Rendering (SSR) và Client-Side Rendering (CSR)? Đây là hai công nghệ phổ biến trong phát triển web, mỗi công nghệ có cách tiếp cận khác nhau để render trang web, mang lại những lợi ích và hạn chế riêng. Ngoài ra còn có 1 số công nghệ khác để render web, tuy nhiên phần lớn là sự kết hợp giữa 2 công nghệ này như : ISR (Static Site Generation), SSG (Static Site Generation), Progressive Hydration ...



client side rendering

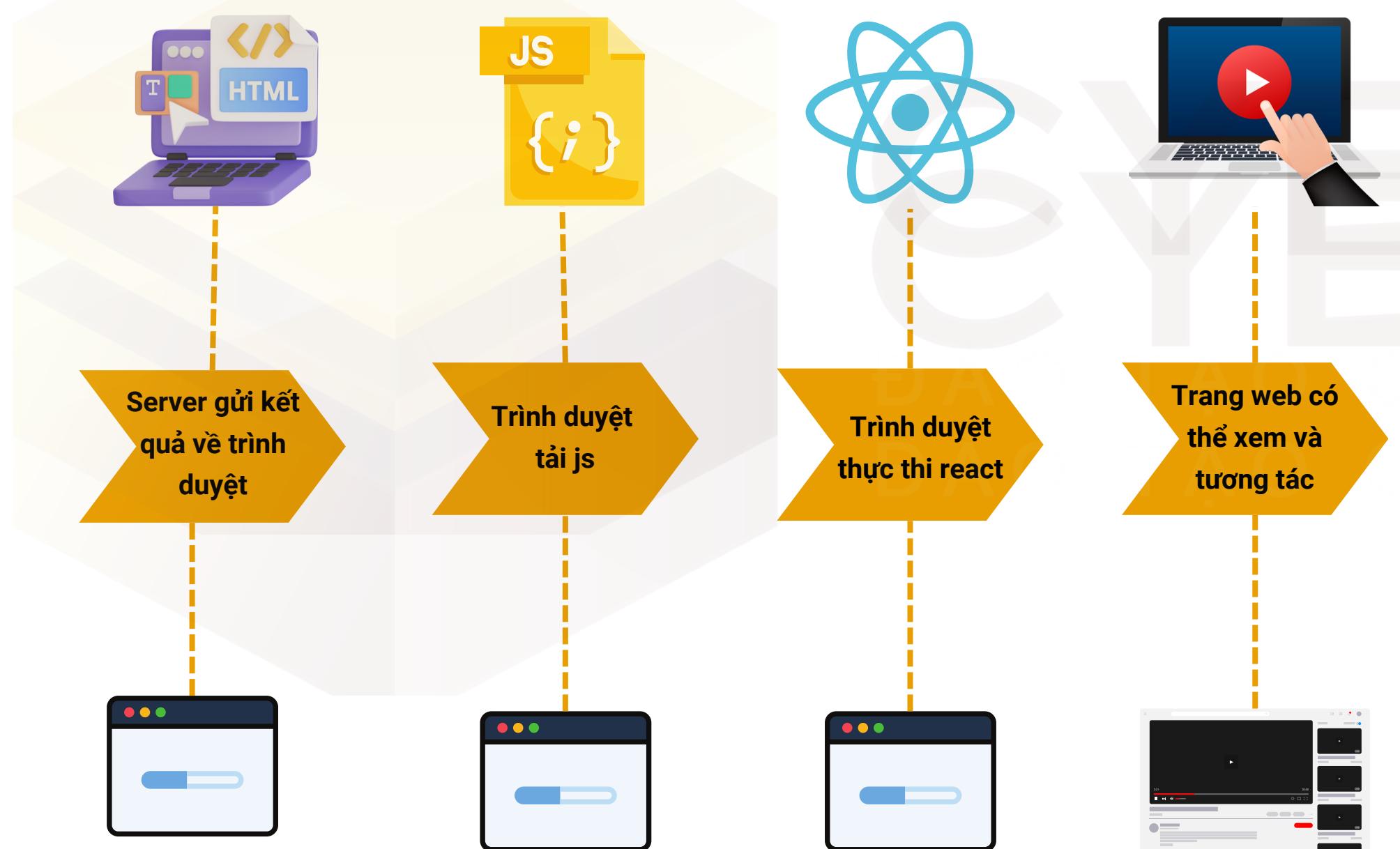


server side rendering



CSR (Client side rendering)

- CSR là kỹ thuật render trang web trên phía client (trình duyệt).
- Thay vì server render và gửi HTML hoàn chỉnh, server chỉ gửi một trang HTML cơ bản và các tập tin JavaScript.
- JavaScript sẽ tải dữ liệu và render nội dung động trên trình duyệt.



****Ưu điểm:****

- Trải nghiệm người dùng mượt mà và tương tác nhanh chóng.
- Giảm tải cho server sau khi trang đầu tiên tải xong.
- Tối ưu cho các ứng dụng có nhiều tương tác phức tạp.

****Nhược điểm:****

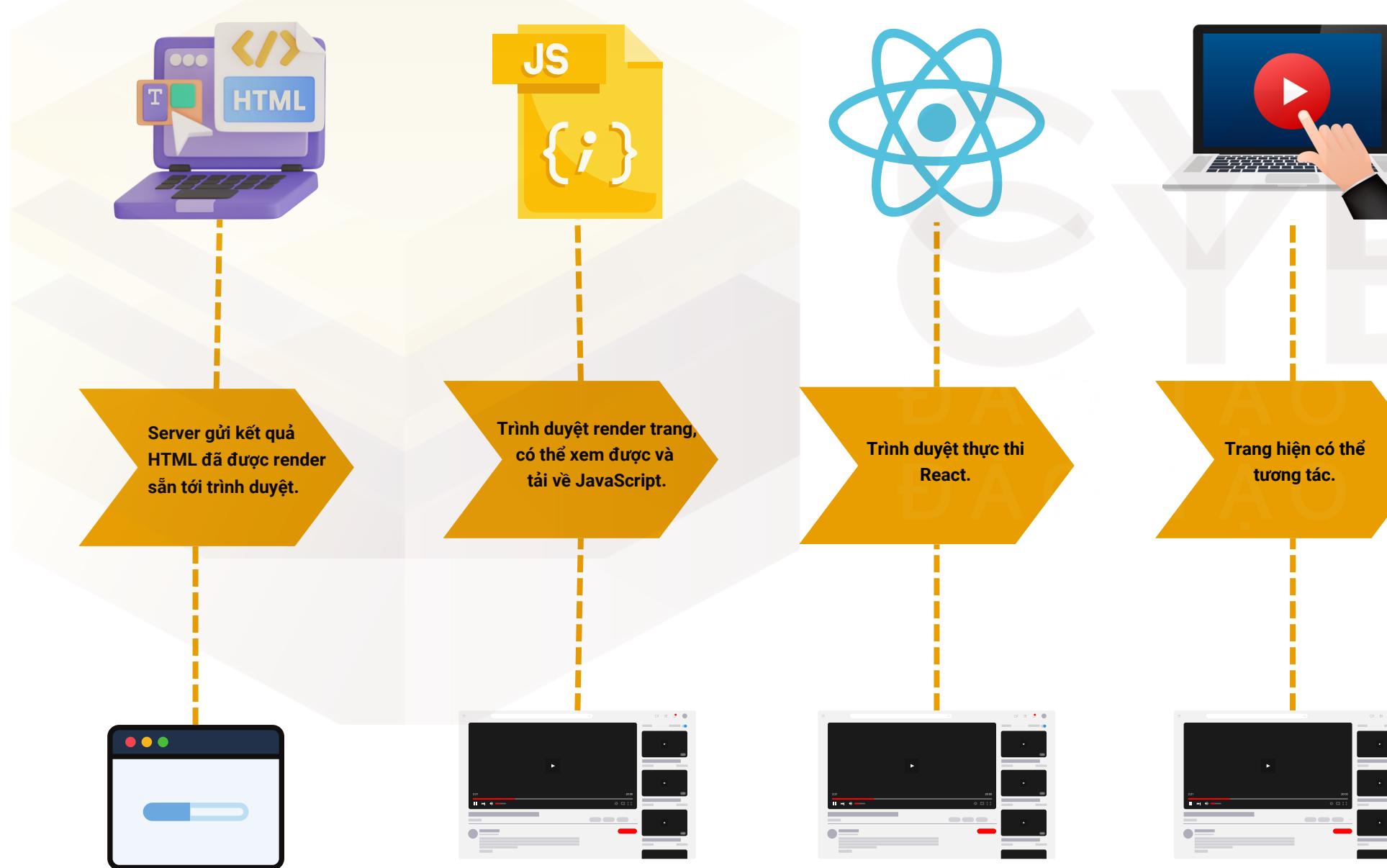
- Thời gian tải trang đầu tiên có thể chậm hơn.
- SEO có thể kém hơn nếu không có biện pháp tối ưu.

Như trên hình, ta có thể nhận thấy trình duyệt sẽ load tất cả nội dung về máy, một khi đã load thành công, nó sẽ thực thi code, lúc này nội dung trang web mới được hiển thị ra.



SSR (SSR side rendering)

- Server Side Rendering (SSR) là kỹ thuật render trang web trên server trước khi gửi về client.
- Khi người dùng gửi request đến server, server sẽ xử lý và render toàn bộ trang HTML.
- Trang HTML hoàn chỉnh sau đó được gửi về client, nơi trình duyệt sẽ hiển thị nội dung ngay lập tức.



****Ưu điểm:****

- Cải thiện SEO: Nội dung trang được render sẵn, giúp các công cụ tìm kiếm dễ dàng thu thập thông tin.
- Thời gian tải trang đầu tiên nhanh hơn: Người dùng thấy nội dung ngay lập tức mà không cần đợi JavaScript tải và thực thi.
- Phù hợp với các trang có nội dung tĩnh nhiều: Như blog, trang tin tức, bán hàng.

****Nhược điểm:****

- Tăng tải cho server: Server phải xử lý và render mỗi request từ người dùng.
- Thời gian phản hồi có thể lâu hơn: Do server phải render HTML trước khi gửi về client.

Như trên hình, ta có thể nhận thấy trình duyệt sẽ load tất cả nội dung về máy, một khi đã load thành công, nó sẽ thực thi code, lúc này nội dung trang web mới được hiển thị ra.



Next.js là gì?

Next.js là một framework phát triển web được xây dựng dựa trên React, được phát triển bởi công ty Vercel. Nó cho phép các nhà phát triển xây dựng các ứng dụng React với các tính năng mạnh mẽ như kết xuất phía server (server-side rendering), tạo trang tĩnh (static site generation), và tạo trang động (dynamic routing) dễ dàng. Next.js là một trong những công cụ phổ biến nhất để phát triển các ứng dụng React hiện đại với hiệu suất cao và trải nghiệm người dùng tốt.

Cài đặt nextjs

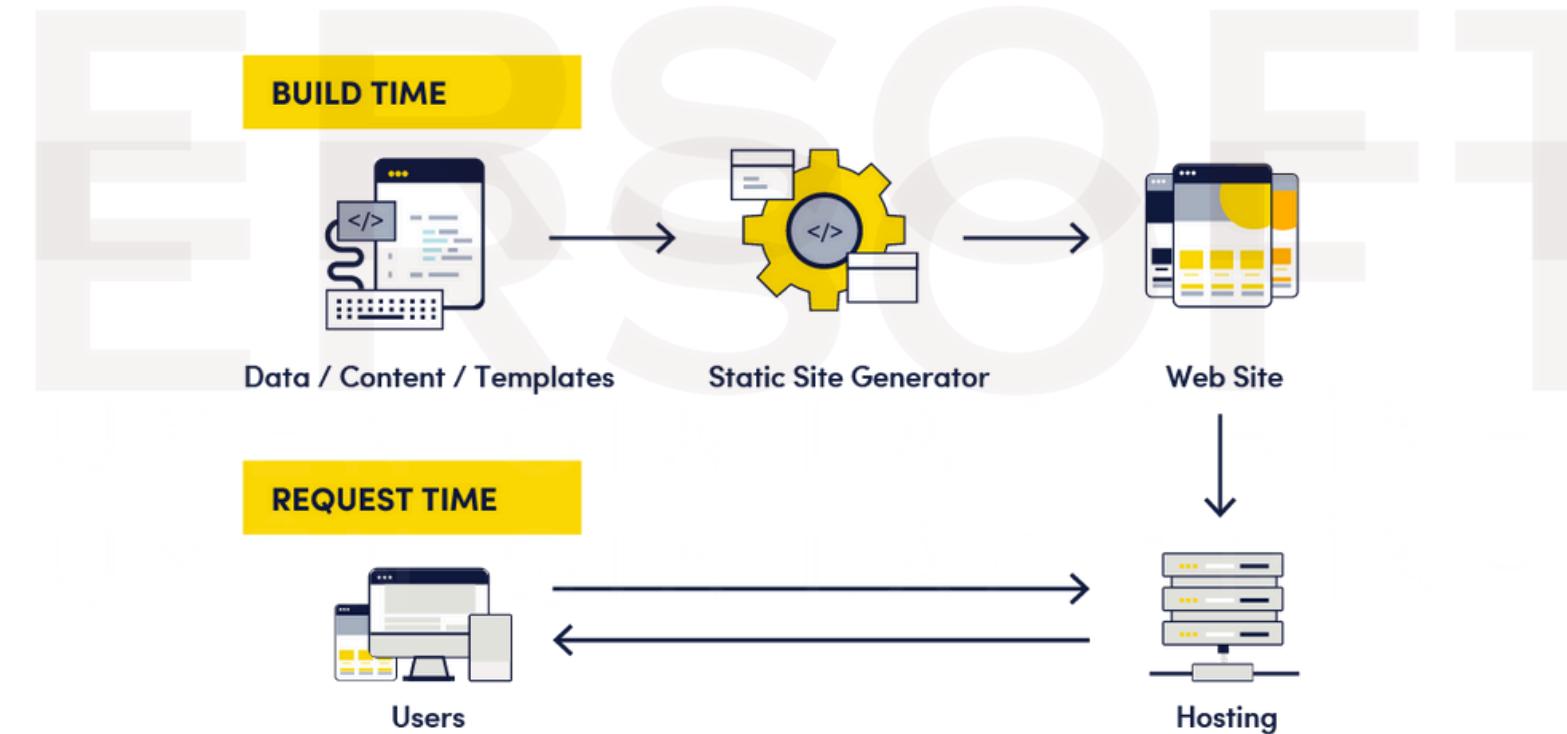
```
npx create-next-app@latest
```

Trở vào thư mục dự án start dự án

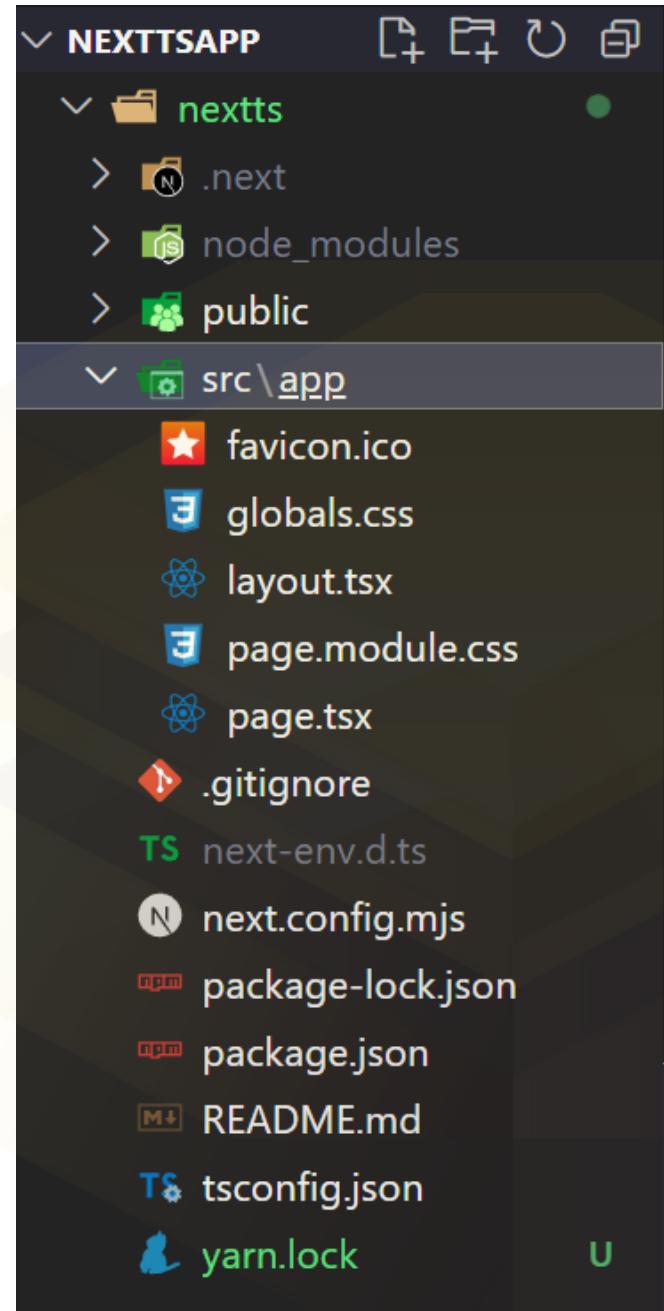
```
yarn install  
yarn run dev
```

build

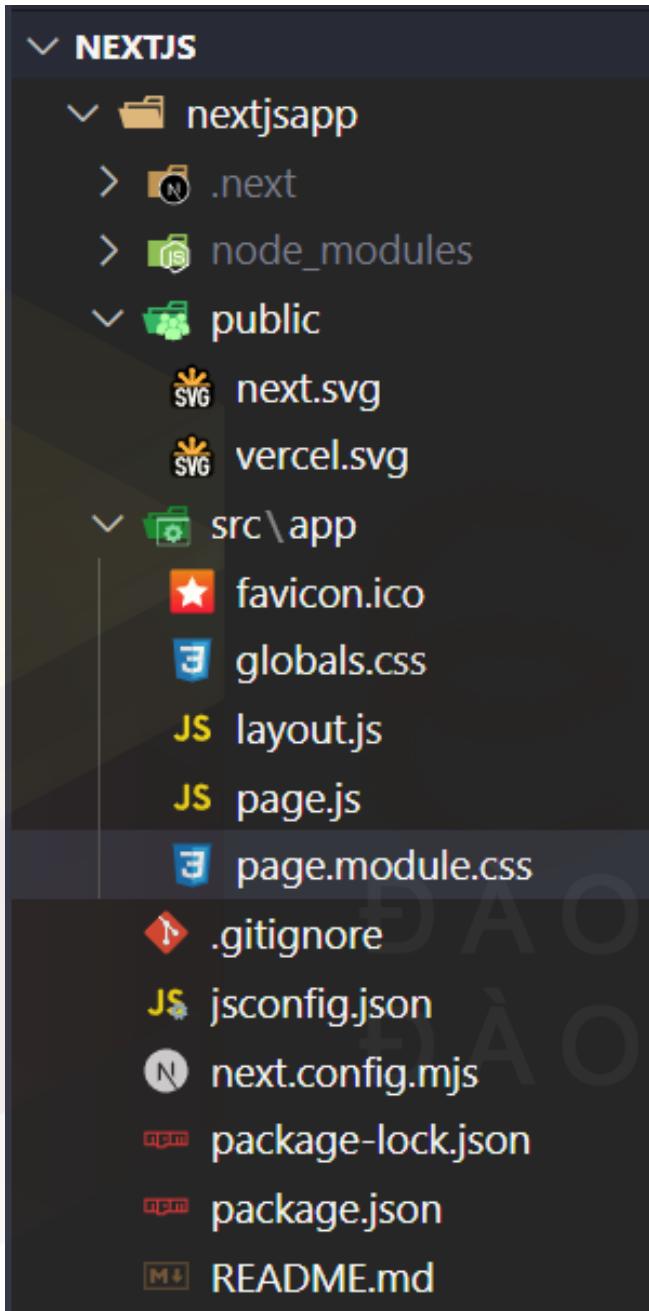
```
yarn run build
```



Cấu trúc dự án



nextjs typescript



nextjs

- **nextjs có 2 phiên bản là page router và app router (>14.0).**
Ở đây ta sẽ học theo approuter.
- **public:** chứa các tài nguyên resource như hình ảnh, icon v...v... tương tự react.
- **app:** là thư mục chứa source code toàn ứng dụng trong app ta sẽ tổ chức các thư mục (folder) tương ứng với các page.
- **layout.jsx hay layout.tsx** là file dạng template tương tự container component bên react để xây dựng các bộ cục dùng dung cho các trang
- **page.jsx hay page.tsx** tương ứng với trang chủ giống index.jsx ta học bên reactjs
- **next.config.mjs:** Nó cho phép bạn tùy chỉnh và mở rộng các chức năng mặc định của Next.js. Dưới đây là một số tính năng phổ biến mà bạn có thể cấu hình trong next.config.mjs
 - ví dụ như cấu hình webpack
 - cài đặt biến môi trường (Environment Variables)
 - Thiết lập chuyển hướng trang (Redirects)
 - customize hình ảnh (Image Optimization)
 - cấu hình Internationalized Routing và còn nhiều config khác
- Các tập tin còn lại tương tự reactjs

Routing NEXT.js

Trước đây nextjs cấu hình các pages (các trang web như trang chủ, liên hệ, đăng ký, đăng nhập ...) thông qua việc đặt các file js trong thư mục pages cách cấu hình này được gọi là Page router. Tuy nhiên từ phiên bản next 14. trở đi nextjs có thêm 1 cách cấu hình khác gọi là AppRouter. Trong khóa học này mình sẽ chọn phần app để cấu hình.

Cấu trúc thư mục

```
plaintext
Sao chép mã

/pages
|-- _app.js      # Custom App component for common layout
|-- index.js     # Home page
|-- about.js     # About page
|-- blog
|   |-- index.js  # Blog listing page
|   |-- [id].js    # Blog post page (dynamic route)
```

Ví dụ về `pages/_app.js`:

```
javascript
Sao chép mã

// pages/_app.js
import Layout from '../components/Layout';

function MyApp({ Component, pageProps }) {
  return (
    <Layout>
      <Component {...pageProps} />
    </Layout>
  );
}

export default MyApp;
```

NextJS < 14

Page router sử dụng tên file làm tên đường dẫn ví dụ:

- /pages/index.js -> /
- /pages/about.js -> /about
- /pages/contact.js -> /contact

App Router với Layout

Trong App Router, layout được tổ chức và sử dụng rõ ràng hơn với các thư mục `layout`.

Cấu trúc thư mục

```
plaintext
Sao chép mã

/app
|-- layout.js      # Root layout
|-- page.js        # Home page
|-- about
|   |-- layout.js  # About layout
|   |-- page.js    # About page
|-- blog
|   |-- layout.js  # Blog layout
|   |-- page.js    # Blog listing page
|   |-- [id]         # Dynamic route directory
|       |-- page.js # Blog post page (dynamic route)
```

Ví dụ về `app/layout.js`:

```
javascript
Sao chép mã

// app/layout.js
export default function RootLayout({ children }) {
  return (
    <html>
      <head>
        <title>My App</title>
      </head>
      <body>
        <header>Header content</header>
        <main>{children}</main>
        <footer>Footer content</footer>
      </body>
    </html>
  );
}
```

NextJS >= 14 sử dụng folder để làm đường dẫn ví dụ:

- /page.js -> /
- /register/page.js -> /register
- /about/page.js -> /about
- /contact/page.js -> /contact

Routing NEXT.js

Trước đây nextjs cấu hình các pages (các trang web như trang chủ, liên hệ, đăng ký, đăng nhập ...) thông qua việc đặt các file js trong thư mục pages cách cấu hình này được gọi là Page router. Tuy nhiên từ phiên bản next 14. trở đi nextjs có thêm 1 cách cấu hình khác gọi là AppRouter. Trong khóa học này mình sẽ chọn phần app để cấu hình.

Cấu trúc thư mục

```
plaintext
Sao chép mã

/pages
|-- _app.js      # Custom App component for common layout
|-- index.js     # Home page
|-- about.js     # About page
|-- blog
|   |-- index.js  # Blog listing page
|   |-- [id].js    # Blog post page (dynamic route)
```

Ví dụ về `pages/_app.js`:

```
javascript
Sao chép mã

// pages/_app.js
import Layout from '../components/Layout';

function MyApp({ Component, pageProps }) {
  return (
    <Layout>
      <Component {...pageProps} />
    </Layout>
  );
}

export default MyApp;
```

NextJS < 14

Page router sử dụng tên file làm tên đường dẫn ví dụ:

- /pages/index.js -> /
- /pages/about.js -> /about
- /pages/contact.js -> /contact

App Router với Layout

Trong App Router, layout được tổ chức và sử dụng rõ ràng hơn với các thư mục `layout`.

Cấu trúc thư mục

```
plaintext
Sao chép mã

/app
|-- layout.js      # Root layout
|-- page.js        # Home page
|-- about
|   |-- layout.js  # About layout
|   |-- page.js    # About page
|-- blog
|   |-- layout.js  # Blog layout
|   |-- page.js    # Blog listing page
|   |-- [id]         # Dynamic route directory
|       |-- page.js # Blog post page (dynamic route)
```

Ví dụ về `app/layout.js`:

```
javascript
Sao chép mã

// app/layout.js
export default function RootLayout({ children }) {
  return (
    <html>
      <head>
        <title>My App</title>
      </head>
      <body>
        <header>Header content</header>
        <main>{children}</main>
        <footer>Footer content</footer>
      </body>
    </html>
  );
}
```

NextJS >= 14 sử dụng folder để làm đường dẫn ví dụ:

- /page.js -> /
- /register/page.js -> /register
- /about/page.js -> /about
- /contact/page.js -> /contact

Routing NEXT.js

Một số lưu ý khi làm việc với nextjs. Next.js 14 sử dụng các cơ chế render sau:

- **Server-Side Rendering (SSR):**

- Xử lý javascript: Trên máy chủ (server).
- Flow: Client yêu cầu -> Máy chủ render HTML, CSS, JavaScript -> Gửi về trình duyệt -> Trình duyệt hiển thị và thực thi JavaScript.

- **Static Site Generation (SSG):**

- Xử lý javascript: Trên máy chủ (server) trong quá trình build.
- Flow: Build -> Tạo file HTML, CSS, JavaScript tĩnh -> Lưu trữ trên máy chủ -> Client yêu cầu -> Máy chủ gửi file tĩnh -> Trình duyệt hiển thị.

- **Incremental Static Regeneration (ISR):**

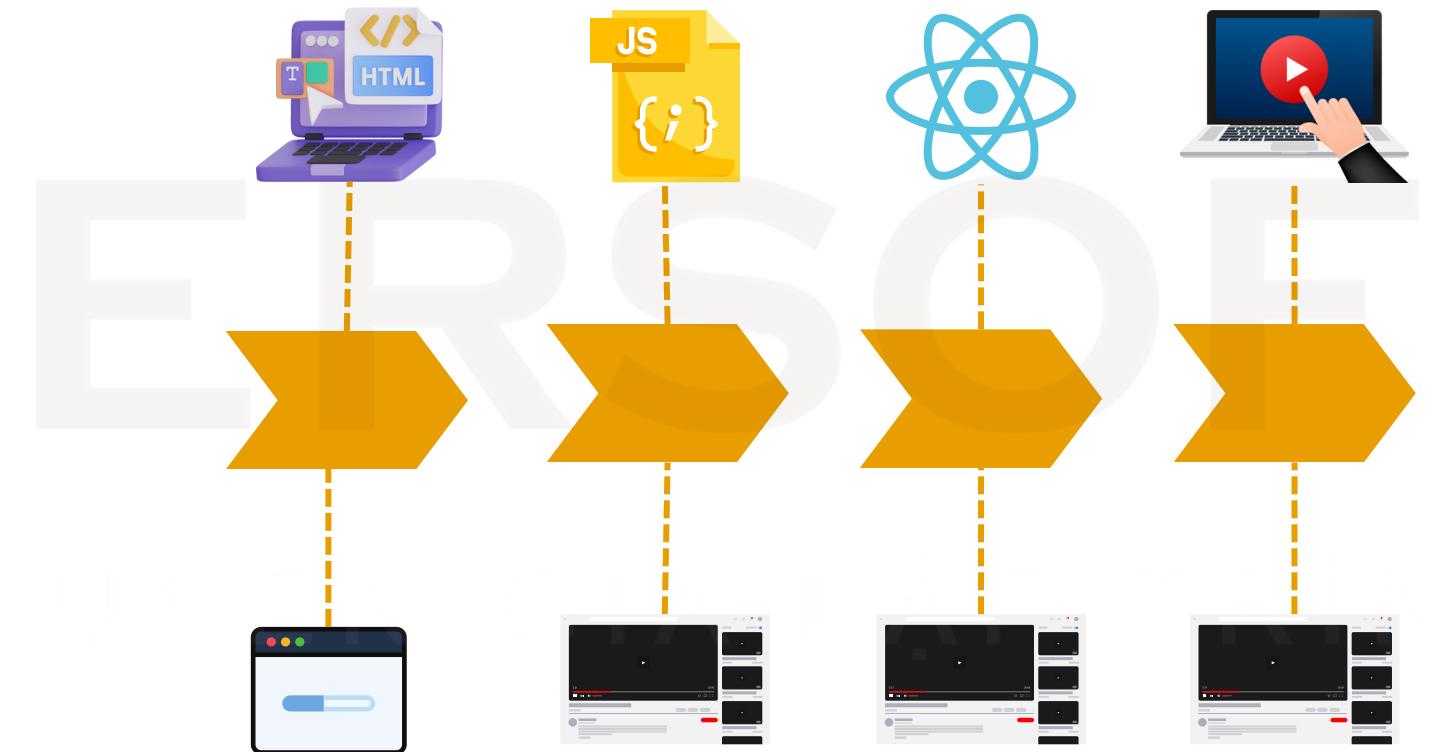
- Xử lý javascript: Trên máy chủ (server) trong quá trình build ban đầu và khi revalidate.
- Flow: Tương tự SSG, nhưng có thêm bước revalidate để cập nhật nội dung định kỳ hoặc theo yêu cầu.

- **Client-Side Rendering (CSR):**

- Xử lý javascript: Trên trình duyệt (client).
- Flow: Yêu cầu -> Máy chủ gửi khung HTML cơ bản -> Trình duyệt tải và thực thi JavaScript -> JavaScript fetch dữ liệu, render nội dung và cập nhật DOM.

- **Tóm lại:**

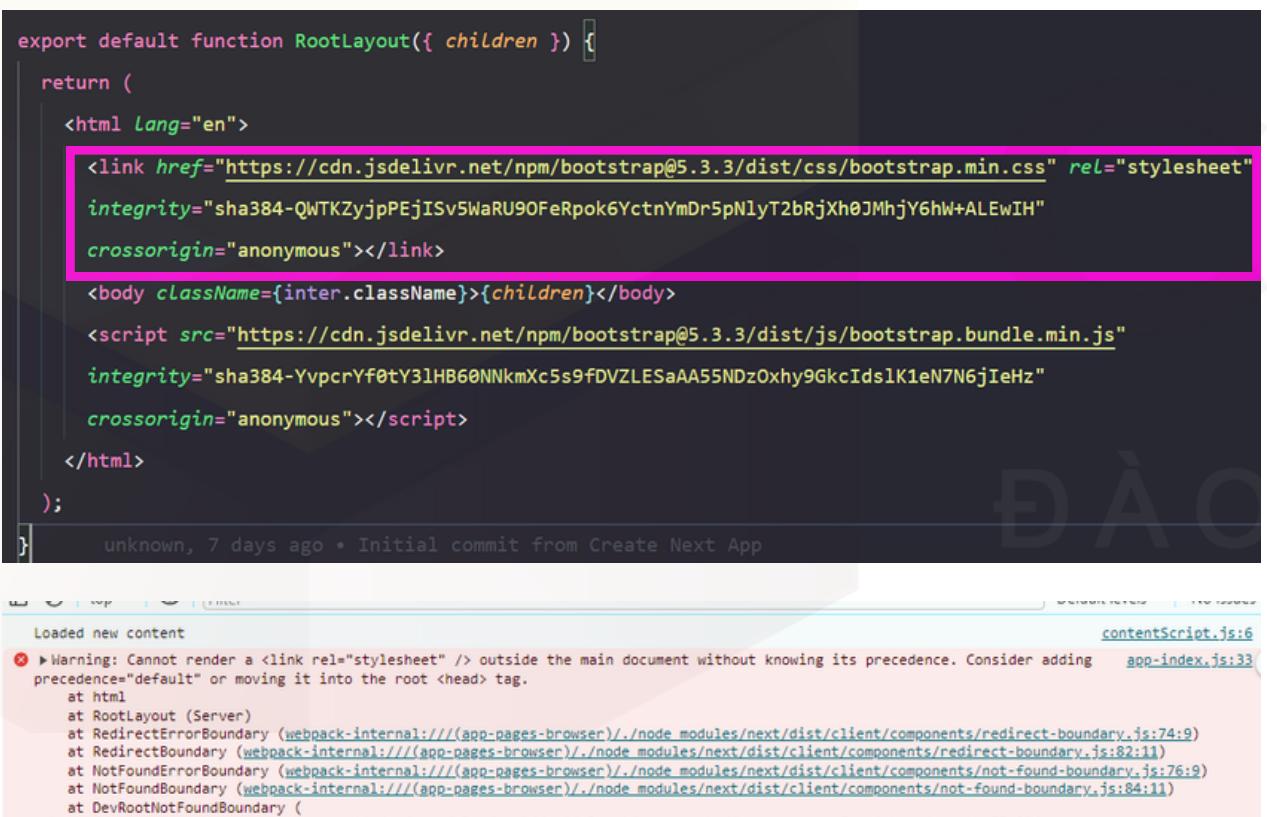
- Next.js 14 ưu tiên SSR và SSG để tối ưu SEO và hiệu suất.
- ISR được sử dụng khi cần cập nhật nội dung thường xuyên nhưng không real-time.
- CSR được sử dụng cho các ứng dụng web động và tương tác cao.
- Việc xử lý script diễn ra ở máy chủ (SSR, SSG, ISR) hoặc trình duyệt (CSR).



Một số lưu ý khi sử dụng nextjs

1. Sử dụng cdn cho thẻ <link> và <script>

Vì hầu hết nextjs sử dụng cơ chế render html từ phía server (máy chủ - khác với react xử lý script trên máy người dùng) nên cũng cần lưu ý: đối với việc sử dụng cdn là đưa 1 đoạn mã (script, link ...) từ web khác vào dự án chúng ta, mà mọi xử lý hiện tại của nextjs đa phần đều nằm ở máy chủ xử lý nên khá nhạy cảm (dễ bị DDoS, XXS, ...) nên trình duyệt sẽ cảnh báo. Vì vậy mặc định nextjs sẽ cảnh báo.



```
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
            integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
            crossorigin="anonymous"></link>
      <body className={inter.className}>{children}</body>
      <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
             integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIds1K1eN7N6jIeHz"
             crossorigin="anonymous"></script>
    </html>
  );
}

unknown, 7 days ago • Initial commit from Create Next App
```

Loaded new content contentScript.js:6

Warning: Cannot render a <link rel="stylesheet" /> outside the main document without knowing its precedence. Consider adding precedence="default" or moving it into the root <head> tag.
at html
at RootLayout (Server)
at RedirectErrorBoundary (webpack-internal:///app-pages-browser/../node_modules/next/dist/client/components/redirect-boundary.js:74:9)
at RedirectBoundary (webpack-internal:///app-pages-browser/../node_modules/next/dist/client/components/redirect-boundary.js:82:11)
at NotFoundErrorBoundary (webpack-internal:///app-pages-browser/../node_modules/next/dist/client/components/not-found-boundary.js:76:9)
at NotFoundBoundary (webpack-internal:///app-pages-browser/../node_modules/next/dist/client/components/not-found-boundary.js:84:11)
at DevRootNotFoundBoundary (

warning khi chèn link vào RootLayout



```
next.config.mjs M X
You, 6 seconds ago | 2 authors (You and others)

1  /** @type {import('next').NextConfig} */
2  const nextConfig = {
3    assetPrefix: 'https://cdn.jsdelivr.net', // Đường dẫn cơ sở cho các tài nguyên tĩnh
4    images: {
5      domains: ['https://cdn.jsdelivr.net'], // Cho phép tải hình ảnh từ domain này
6    },
7  };
8
9  export default nextConfig;
```

config cho cdn tại file next.config.mjs

Một số lưu ý khi sử dụng nextjs

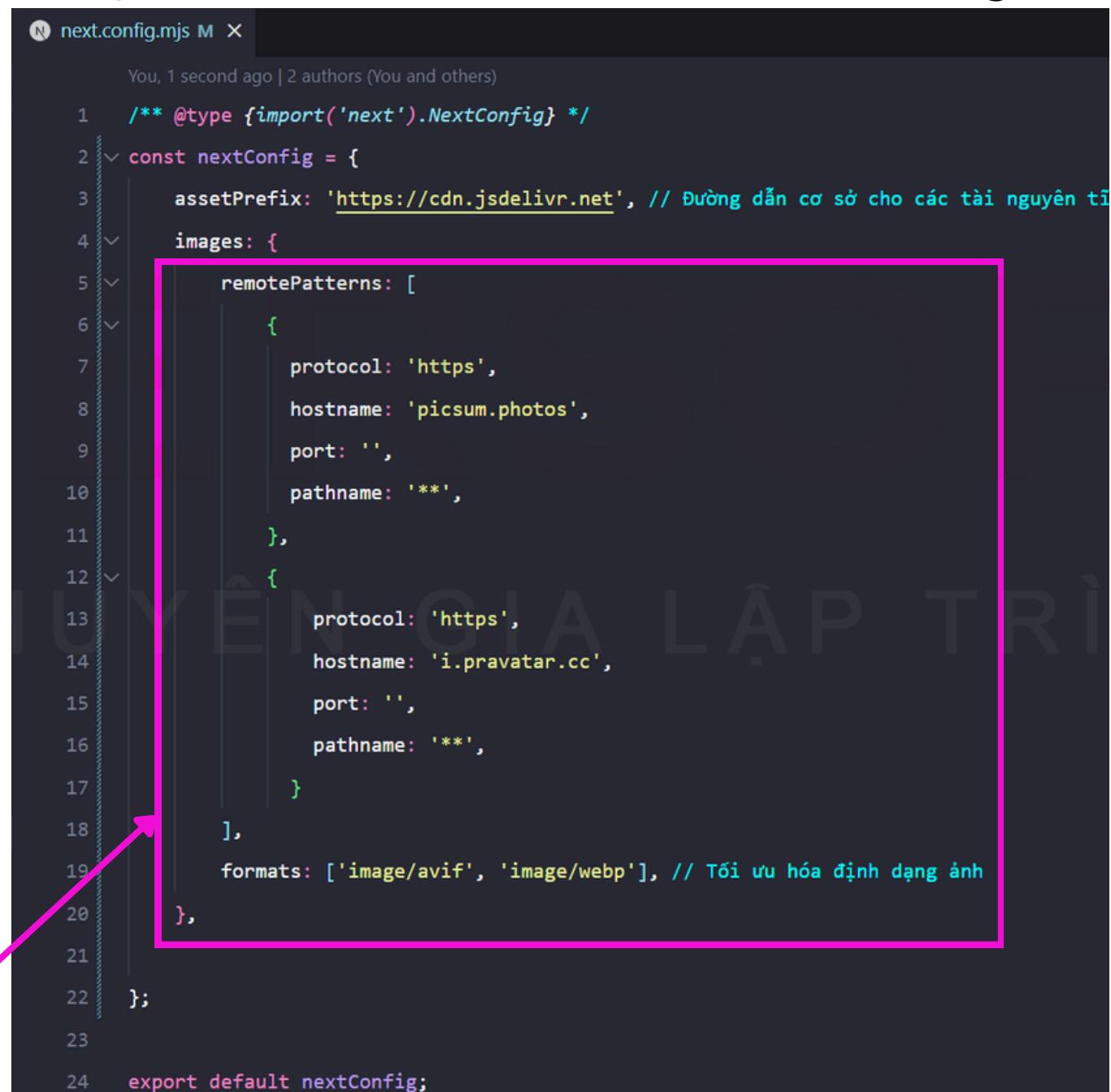
2. Sử dụng hình ảnh từ link ngoài dự án

Tương tự cdn đối với hình ảnh khi tham chiếu từ nguồn khác cũng sẽ được cảnh báo, để cấu hình cho các hình ảnh từ link web ngoài thì ta cấu hình trong file `next.config.mjs`

```
1 import Image from 'next/image' 16.5k (gzipped: 6.2k)
2 import React from 'react' 6.9k (gzipped: 2.7k)
3
4 const Demo = () => {
5   return (
6     <div>
7       <Image src='https://i.pravatar.cc' alt='...' width={200} height={200} crossOrigin="anonymous" quality={100}>
8     </div>
9   )
10 }
11
12 export default Demo
```

```
Warning: Invalid DOM property `crossorigin`. Did you mean `crossOrigin`?
  at link
  at html
  at RedirectErrorBoundary (webpack-internal:///((ssr)/./node_modules/next/dist/client/components/redirect-boundary.js:73:9)
  at RedirectBoundary (webpack-internal:///((ssr)/./node_modules/next/dist/client/components/redirect-boundary.js:81:11)
  at ReactDevOverlay (webpack-internal:///((ssr)/./node_modules/next/dist/client/components/react-dev-overlay/app/ReactDevOverlay.js:87:9)
  at HotReload (webpack-internal:///((ssr)/./node_modules/next/dist/client/components/react-dev-overlay/app/hot-reloader-client.js:322:11)
  at Router (webpack-internal:///((ssr)/./node_modules/next/dist/client/components/app-router.js:202:11)
  at ErrorBoundaryHandler (webpack-internal:///((ssr)/./node_modules/next/dist/client/components/error-boundary.js:112:9)
  at ErrorBoundary (webpack-internal:///((ssr)/./node_modules/next/dist/client/components/error-boundary.js:158:11)
  at AppRouter (webpack-internal:///((ssr)/./node_modules/next/dist/client/components/app-router.js:556:13)
  at Lazy
```

Tương tự việc load 1 hình ảnh từ domain khác lên ứng dụng web nextjs cũng sẽ được warning
Nextjs cung cấp cho ta thẻ `<Image />` thay vì thẻ img giúp hình ảnh được optimize hơn



```
/* @type {import('next').NextConfig} */
const nextConfig = {
  assetPrefix: 'https://cdn.jsdelivr.net', // Đường dẫn cơ sở cho các tài nguyên tĩnh
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'picsum.photos',
        port: '',
        pathname: '**',
      },
      {
        protocol: 'https',
        hostname: 'i.pravatar.cc',
        port: '',
        pathname: '**',
      }
    ],
    formats: ['image/avif', 'image/webp'], // Tối ưu hóa định dạng ảnh
  },
};

export default nextConfig;
```

Cấu hình cho các tham số từ domain liên kết

Một số lưu ý khi sử dụng nextjs

3. Sử dụng font

Sử dụng từ đường dẫn từ font.google (tương tự cdn bootstrap, thư viện, ...) hoặc dùng thư viện next/font/google được cài đặt sẵn trong ứng dụng

```
1 import "./globals.css";
2 import Script from "next/script"; 10.4k (gzipped: 3.6k)
3 import { Inter, Roboto } from "next/font/google"; 659 (gzipped: 385) ← Cách này thường hay được sử dụng vì nó
4
5 const inter = Inter({ subsets: ["latin"] });
6 const roboto = Roboto({ subsets: ['latin'], weight: ['400', '700'] }); // Chọn các weights bạn cần
7
```

```
14 export default function RootLayout({ children }) {
15   return (
16     <html lang="en">
17       <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
18           integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pN1yT2bRjXh0JMhjY6hW+ALEwIH"
19           crossorigin="anonymous"></link>
20       <h3 className={roboto.className}>Học nextjs tại cybersoft academy</h3>
21       <body className={inter.className}>{children}</body>
22       <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
23           integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIds1K1eN7N6jIeHz"
24           crossorigin="anonymous"></script>
25     </html>
26   );
27 }
```

Các thành phần router trong nextjs

1. Link Component

Sử dụng `<Link>` từ `next/link` để điều hướng giữa các trang một cách hiệu quả mà không cần tải lại toàn bộ trang.

```
import Link from 'next/link';
```

```
<Link href="/path"> Link name </Link>
```

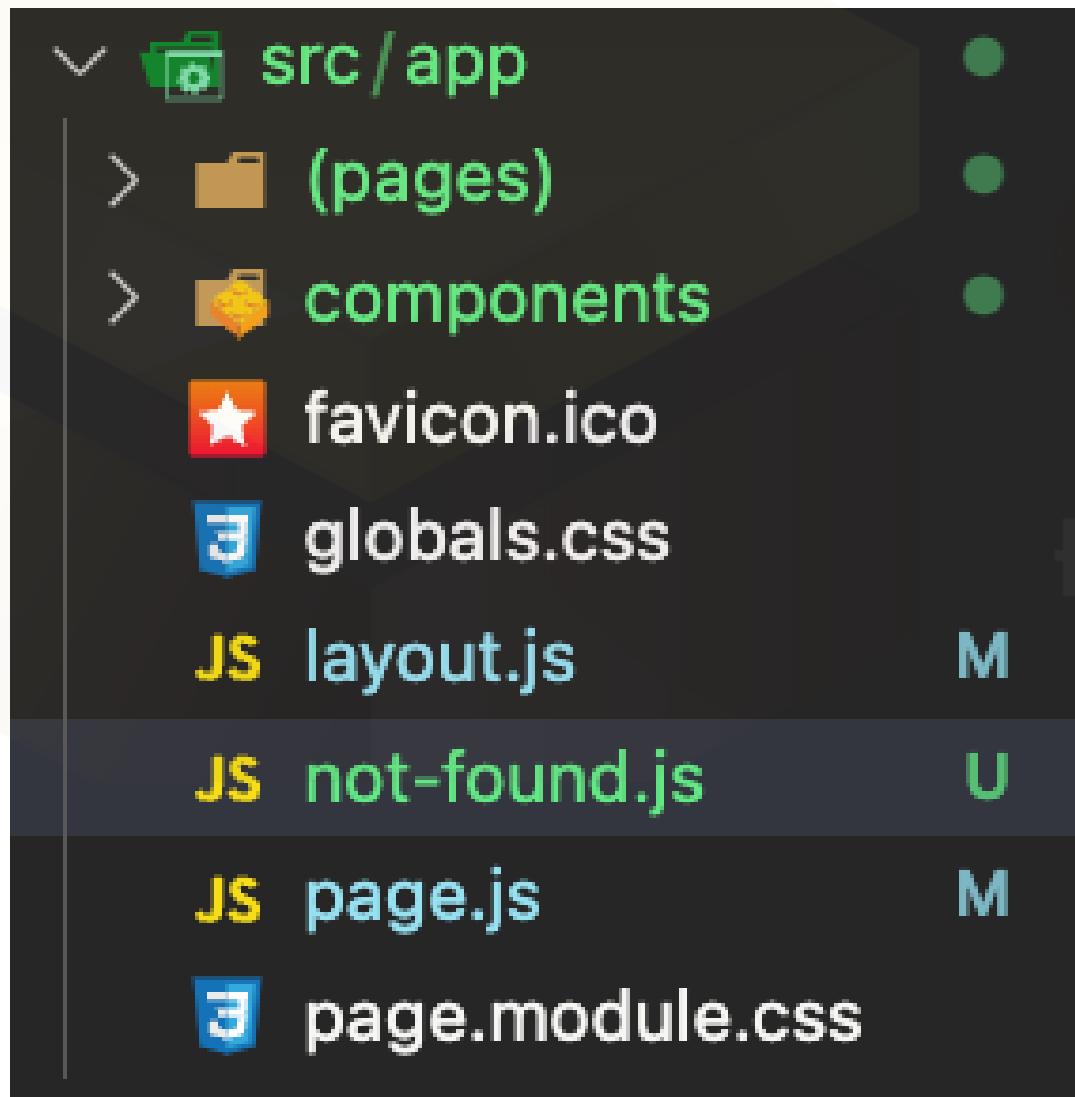
2. Chuyển hướng trang sau 1 xử lý

```
'use client'  
import { useRouter } from 'next/navigation'  
import React from 'react'  
const Component = () => {  
  const router = useRouter();  
  return <>  
    <button onClick={() => {  
      router.push('path')  
    }}> Handle click </button>  
  </>  
}
```

Các thành phần router trong nextjs

3. Chuyển hướng trang 404 khi đường dẫn không có

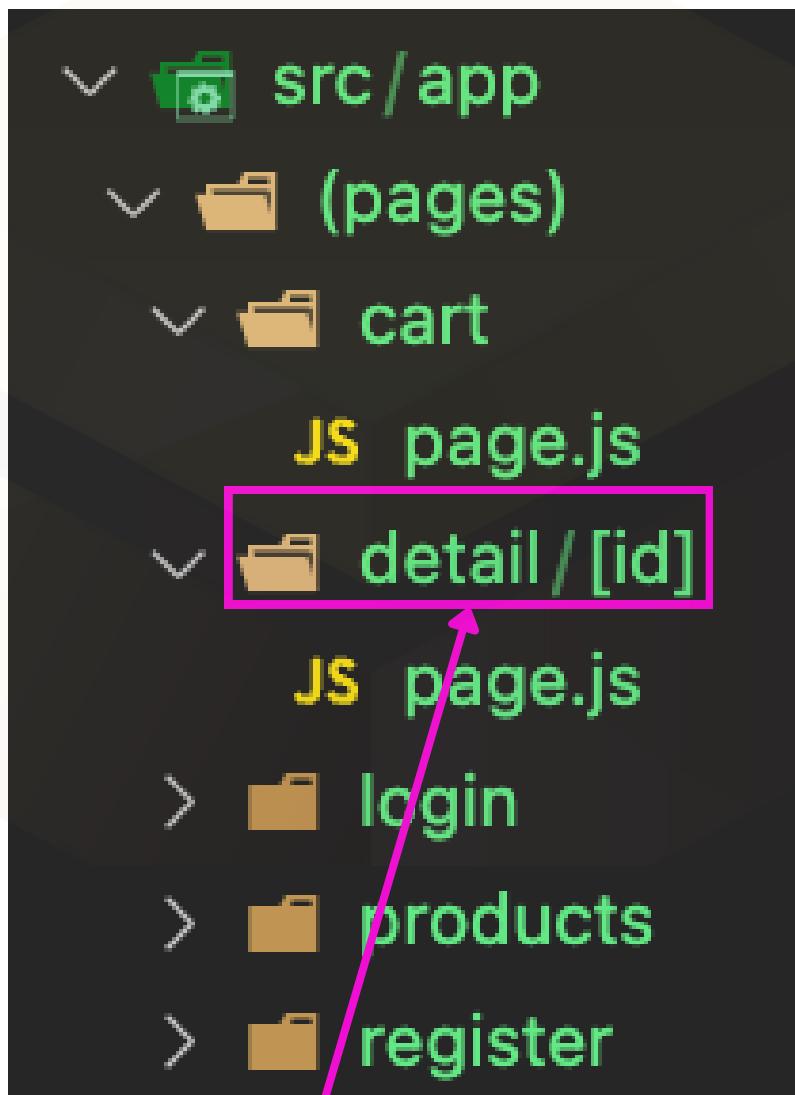
Trong thư mục app tạo file not-found.js . Khi gõ đường dẫn không có trong thư mục app (trang không tồn tại) thì nextjs sẽ tự động chuyển hướng về trang này. (not-found.js)



Các thành phần router trong nextjs

4. Dynamic routing (get by id page)

Dynamic Routing trong Next.js cho phép ta tạo các route động bằng cách sử dụng các tên file và thư mục có chứa các dấu ngoặc vuông ([]) ví dụ như [id]. Ứng dụng tính năng này để ta xây dựng các trang như detail, chi tiết sản phẩm, bài viết blog, hồ sơ người dùng...



Xây dựng 1 trang page detail với [ten_tham_so]

xây dựng hàm call api từ tham số và nhận kết quả trả về từ hàm này thông qua return

The top snippet is a server component (getProductById.js) that uses axios to fetch data from an API:

```
1 import axios from 'axios' 61.7k (gzipped: 22.7k)
2 const getProductById = async (id) => {
3
4   try {
5     const res = await axios.get(`https://apistore.cybersoft.edu.vn/api/Product/getbyid?id=${id}`);
6     return res.data.content;
7   } catch (error) {
8     console.error('Error fetching data:', error);
9     return { id, title: 'Error', content: 'Could not fetch content.' };
10  }
11 }
```

The bottom snippet is a client component (page.js) that receives the fetched data via props.params.id and logs it:

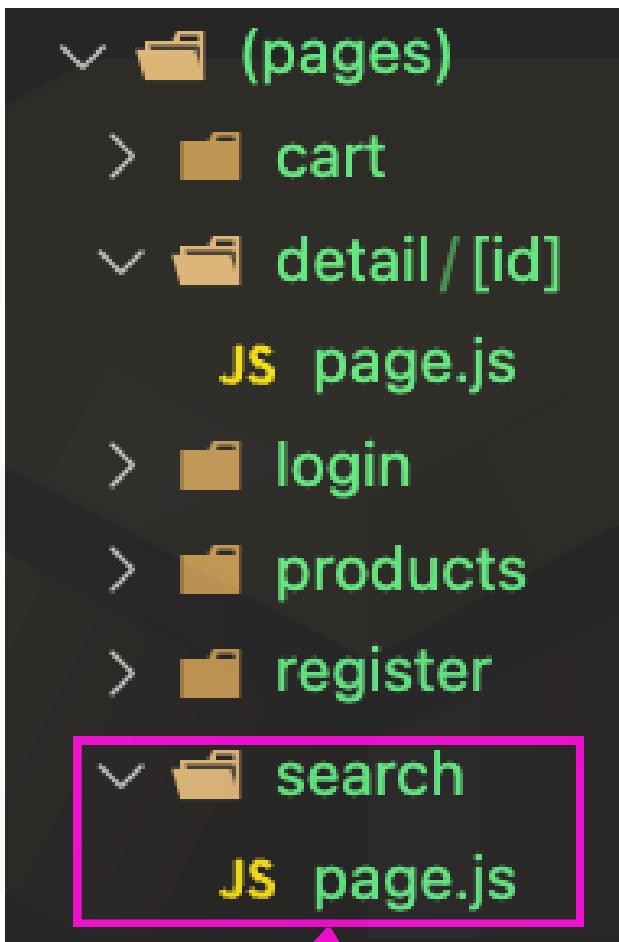
```
12 const page = async (props) => {
13   const { id } = props.params;
14   const res = await getProductById(id);
15   console.log(res)
16
17   return (
18     <div className='container'>
19       <div id={id}>
20         <div className='card w-25'>
21           <div className='card-header'>
22             <h3> product name: {res.name}</h3>
23           </div>
24
25           <div className='card-body'>
26             <img src={res.image} alt='...' className='w-100' />
27             <h3>{res.name}</h3>
28             <p>{res.price}</p>
29             <button className='btn btn-success'>Add to card</button>
30           </div>
31         </div>
32       </div>
33     </div>
34   )
35 }
```

Xây dựng server component sử dụng props.param. [ten_tham_so] để lấy giá trị từ url tại phía server

Các thành phần router trong nextjs

4. Dynamic routing (search page)

Để xây dựng các page như search param như react khi sử dụng nextjs ta sẽ xây dựng như sau:



Tạo thư mục cho
search page

Tương tự lấy id tại **search page**
chúng ta dùng props tại
component server để lấy
param từ url thông qua thuộc
tính `props.searchParams`

```
<form className="d-flex my-2 my-lg-0" onSubmit={(e) => {
  e.preventDefault();
  router.push(`/search?keyword=${keyword}`);
}}>
  <input className="form-control me-sm-2" type="text" placeholder="Search" onChange={(e) => {
    setKeyword(e.target.value);
}} />
  <Link href={`/search?keyword=${keyword}`} className="btn btn-outline-success my-2 my-sm-0" type="submit">
    Search
  </Link>
</form>
```

Link đến trang search bằng thẻ Link hoặc `useRouter` của `next/navigation`

```
async function fetchProducts(keyword) {
  const response = await axios(`http://localhost:3000/api/products?keyword=${keyword}`);
  console.log(response.data)
  return response.data
}

export default async function ProductsPage(props) {
  console.log(props)
  let keyword = props.searchParams.keyword || ''
  const data = await fetchProducts(keyword)
  return (
    <div>
      <h1>Product List</h1>
      <ul>
        {data?.content?.map((product) => (
          <li key={product.id}>{product.name} - <Link className="btn btn-success" href={`/detail/${product.id}`}>View detail</Link></li>
        ))}
      </ul>
    </div>
  );
}
```

Các thành phần router trong nextjs

5. API Routes:

API Route trong Next.js là một cách để tạo các endpoint API trong ứng dụng Next.js. API Routes cung cấp một cách dễ dàng và hiệu quả để tạo API mà không cần phải thiết lập một server backend riêng biệt hoặc ta có thể setup resful api tương tự cách viết class service kết nối api của backend đồng thời có thể tùy biến lại status code từ backend.

Ngoài ra Api route thường được kết hợp với 1 số công nghệ phổ biến như:

- **Database (Cơ sở dữ liệu)**

- API Routes thường được sử dụng để truy xuất và thao tác dữ liệu từ cơ sở dữ liệu. Các thư viện phổ biến bao gồm:

- Prisma: ORM mạnh mẽ và linh hoạt cho Node.js và TypeScript.
- TypeORM: ORM cho TypeScript và JavaScript (ES7).
- Mongoose: Thư viện ODM cho MongoDB.

- **Authentication (Xác thực)**

- API Routes thường kết hợp với các giải pháp xác thực để bảo vệ các endpoint API. Các thư viện phổ biến bao gồm:

- NextAuth.js: Giải pháp xác thực hoàn chỉnh cho Next.js.
- JWT (JSON Web Tokens): Thư viện xác thực dựa trên token.

- **File Upload (Tải lên tệp)**

- API Routes có thể xử lý việc tải lên tệp bằng các thư viện như:
 - Formidable: Thư viện xử lý form và file upload.
 - Multer: Middleware xử lý multipart/form-data, thường được sử dụng với Express.

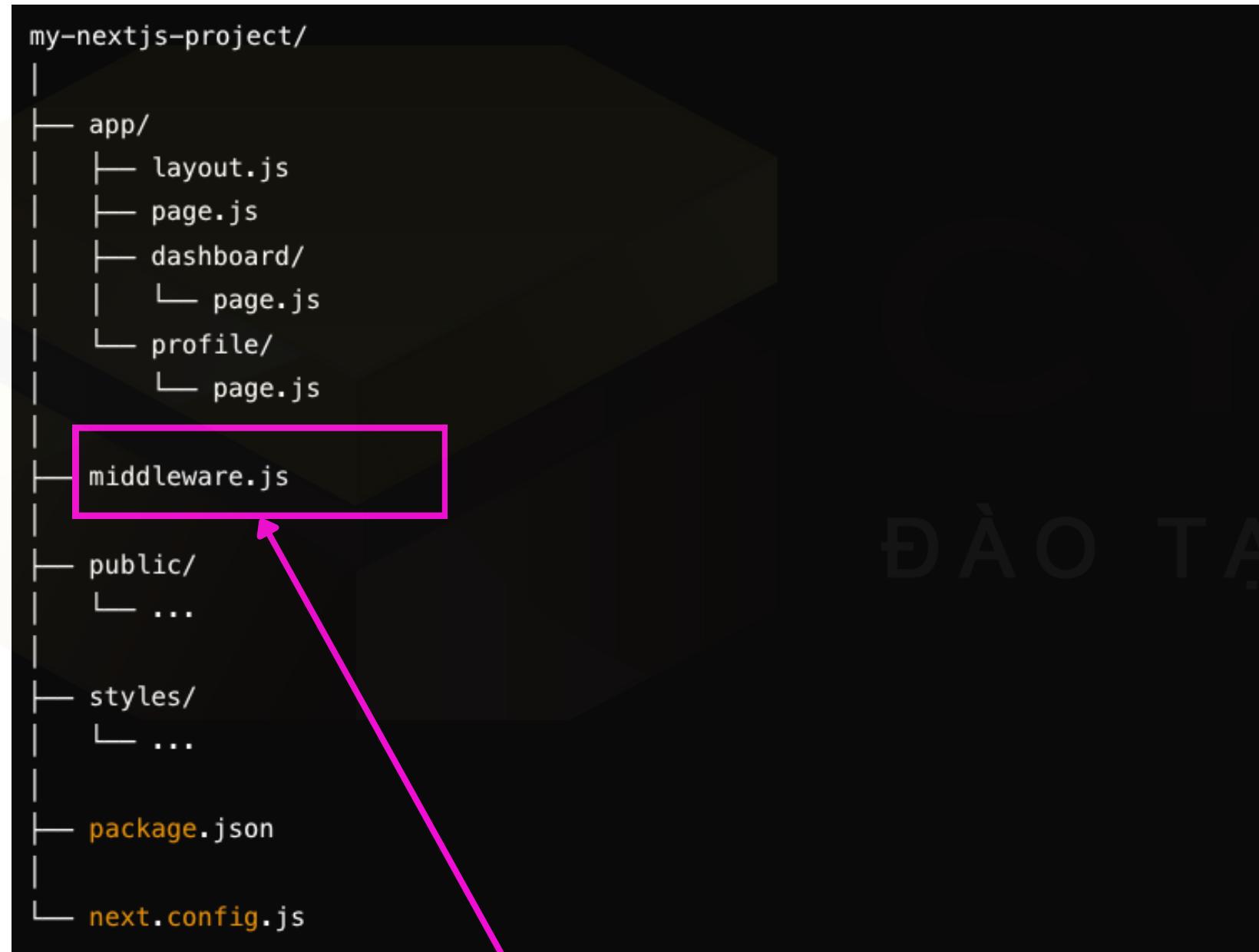
```
1 import { NextResponse } from 'next/server'; 38.5k (gzipped: 13.9k)
2
3 export async function GET() {
4   try {
5     const response = await fetch('https://apistore.cybersoft.edu.vn/api/Product', {
6       headers: {
7         'Content-Type': 'application/json',
8         // Thêm headers khác nếu cần, ví dụ Authorization
9         // 'Authorization': `Bearer YOUR_ACCESS_TOKEN`
10      },
11    });
12    if (!response.ok) {
13      throw new Error('Failed to fetch data');
14    }
15    const data = await response.json();
16    return NextResponse.json(data, { status: 200 });
17  } catch (error) {
18    return NextResponse.json({ message: 'Internal Server Error', error: error.message }, { status: 500 });
19  }
20}
```

Tạo các API route trong thư mục api

Các thành phần router trong nextjs

6. Middleware

Middleware cho phép ta thêm logic tùy chỉnh vào quá trình xử lý yêu cầu (request) và phản hồi (response) trong ứng dụng Next.js.



middleware được tạo

```
import { NextResponse } from 'next/server'; 38.5k (gzipped: 13.9k)

// Middleware function
export function middleware(request) {
    // Kiểm tra nếu người dùng không đăng nhập
    if (!request.cookies.get('user-token')) {
        // Chuyển hướng người dùng đến trang đăng nhập
        return NextResponse.redirect(new URL('/login', request.url));
    }

    // Tiếp tục xử lý request nếu người dùng đã đăng nhập
    return NextResponse.next();
}

// Định nghĩa đường dẫn mà middleware sẽ áp dụng
export const config = {
    matcher: ['/dashboard/:path*', '/profile/:path*'],
};
```

middleware giúp điều hướng trang khi chưa đăng nhập

Lưu ý: middleware thực thi trên server vì vậy các thứ như localstorage hay session storage không thể truy cập được

TÍNH NĂNG TRONG NEXTJS

1. Server action

Tương tự API Routes, Server Actions đều cung cấp các cách để xử lý logic server-side, nhưng chúng có các đặc điểm và mục đích sử dụng khác nhau. Dưới đây là sự so sánh giữa API Routes và Server Actions:

API Routes:

- Tạo trong thư mục api và file tên route
- Độc lập và có thể được gọi từ bất kỳ đâu.
- Hỗ trợ đầy đủ các phương thức HTTP.
- Phù hợp cho các endpoint API RESTful.
- Có thể gây ra overhead do HTTP request.



Server Actions:

- Cho phép gọi trực tiếp các hàm server-side từ client-side.
- Giảm overhead do không tạo ra HTTP request.
- Dễ bảo trì do ít sự phân tách giữa server-side và client-side logic.
- Giới hạn phạm vi và linh hoạt so với API Routes.

```
// Hàm thêm sản phẩm vào cơ sở dữ liệu
const addProductToDatabase = async (product) => {
  // Logic để thêm sản phẩm vào cơ sở dữ liệu
  console.log('Product added to database:', product);
  return { success: true, message: 'Product added successfully' };
};
```

Có thể tạo ra bất cứ đâu và được xử lý tại server

TÍNH NĂNG TRONG NEXTJS

2. Static Site Generation (SSG)

Với SSG, server không cần phải xử lý và render trang cho mỗi yêu cầu (hiệu suất cao). Điều này giúp giảm tải cho server, làm cho nó có thể phục vụ nhiều người dùng hơn mà không cần tài nguyên máy chủ lớn. Các trang tĩnh có thể được crawl dễ dàng bởi các công cụ tìm kiếm vì nội dung đã có sẵn khi trang được tải. Điều này giúp cải thiện xếp hạng SEO của trang web, làm cho nó dễ dàng được tìm thấy trên các công cụ tìm kiếm như Google.



Trong page blog tạo các đường
dẫn qua trang post [id]

<https://jsonplaceholder.typicode.com/posts>

```
1  async function fetchPost(id) {
2
3    const res = await fetch(`https://jsonplaceholder.typicode.com/posts/${id}`);
4    next: { revalidate: 60 * 60 * 24 }, // Tự động làm mới cache sau 60 giây * 60 phút * 24 giờ => cache 1 ngày
5  );
6
7  const post = await res.json();
8
9  return post;
10}
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

TÍNH NĂNG TRONG NEXTJS

2. Static Site Generation (SSG)

Nếu muốn setup với axios các bạn cài thêm thư viện:

```
yarn add axios-cache-adapter  
npm i axios-cache-adapter
```

```
import axios from 'axios';  
import { setupCache } from 'axios-cache-adapter';  
  
// Tạo một instance của axios với cache  
const cache = setupCache({  
  maxAge: 15 * 60 * 1000 // Cache trong 15 phút  
});  
  
const api = axios.create({  
  adapter: cache.adapter  
});
```

Tạo axios instance và setup cài đặt cache

TÍNH NĂNG TRONG NEXTJS

3. Incremental Static Regeneration

ISR mở rộng khả năng của SSG bằng cách cho phép bạn tái tạo các trang tĩnh một cách tự động mà không cần phải build lại toàn bộ ứng dụng. ISR cho phép bạn cập nhật nội dung của các trang tĩnh sau khi đã được build, giúp giữ cho nội dung của bạn luôn mới mà vẫn duy trì hiệu suất cao của các trang tĩnh.

```
export async function GET(request) {
  const products = [
    { id: 1, name: 'post 1', view: 100 },
    { id: 2, name: 'post 2', view: 200 },
    { id: 3, name: 'post 3', view: 300 },
  ];

  return new Response(JSON.stringify(products), {
    headers: { 'Content-Type': 'application/json' },
  });
}
```

Api route

trình khai ISR

```
async function fetchProducts() {
  const res = await fetch('http://localhost:3000/api/post', {
    next: { revalidate: 10 },
  });

  const products = await res.json();
  return products;
}

export default async function page() {
  const products = await fetchProducts();

  return (
    <div>
      {products.map((item)=>{
        return <div>
          { item.name }
        </div>
      ))}
    </div>
  );
}
```

Page: Post

So sánh SSG và ISR

SSG: Khi dữ liệu không thay đổi thường xuyên hoặc yêu cầu cập nhật nhanh chóng. Dùng cho các trang có nội dung cố định.

ISR: Khi dữ liệu thay đổi thường xuyên nhưng không cần cập nhật ngay lập tức. Dùng cho các trang cần cập nhật định kỳ mà không cần rebuild toàn bộ ứng dụng.

Tổng hợp kiến thức

1. Enhanced App Router:

Next.js 14 cải thiện mạnh mẽ router mới (app router) với nhiều tính năng linh hoạt hơn so với pages router.

2. Improved Static and Dynamic Rendering:

Next.js 14 cung cấp các tùy chọn tốt hơn cho cả rendering tĩnh và động, cho phép kết hợp chúng dễ dàng hơn trong cùng một ứng dụng.

3. Optimized Performance:

Các cải tiến về hiệu suất bao gồm tải trước thông minh, giảm thời gian tải và tối ưu hóa hình ảnh tự động, đảm bảo các trang tải nhanh hơn.

4. Incremental Static Regeneration (ISR):

ISR tiếp tục được cải tiến, cho phép các trang tĩnh được tái tạo lại định kỳ mà không cần phải build lại toàn bộ ứng dụng. Điều này giúp cải thiện hiệu suất và khả năng mở rộng.

5. Server Actions:

Tính năng mới cho phép bạn dễ dàng thực hiện các hành động server-side, như lấy dữ liệu hoặc thực hiện logic phức tạp, trực tiếp trong các component.

6. App Directory Layouts:

App router hỗ trợ layouts mạnh mẽ, cho phép chia sẻ layout giữa các routes và component dễ dàng hơn.

7. Enhanced API Routes:

API routes trong app directory được cải tiến với khả năng cấu hình tốt hơn và dễ dàng sử dụng trong các ứng dụng phức tạp.

8. Improved Error Handling:

Next.js 14 cung cấp cách xử lý lỗi tốt hơn, giúp phát hiện và sửa lỗi nhanh chóng trong quá trình phát triển.

9. Better Development Experience:

Cải tiến trải nghiệm phát triển với hot-reloading, error overlay được cải tiến và khả năng debug tốt hơn.

10. React Server Components:

Hỗ trợ tốt hơn cho React Server Components, cho phép rendering phía server hiệu quả hơn và tối ưu hóa thời gian tải trang.

11. SEO Improvements:

Cải thiện SEO với khả năng cấu hình meta tags tốt hơn và tối ưu hóa schema.org tự động.

12. Enhanced Middleware:

Middleware được cải thiện với khả năng kiểm soát logic xử lý yêu cầu tốt hơn trước khi chuyển tiếp đến các routes cụ thể.