



Internship FE

Week 3 - Reactjs-Report

Contents

I. React CLI (Command – Line – Interface).....	3
1. Install React.....	3
2. Folder & structure	4
II. Component	5
1. Component	5
2. Make component.....	7
3. Import and use component	7
III. Component communication	10
IV. Binding.....	12
1. Data binding.....	12
2. Class and style binding.....	13
V. Event handling.....	13
1. Click Event	13
2. Input Event.....	13
3. Change Event	14
VI. Rendering.....	15
1. Condition rendering	15
2. Rendering Lists	16
VII. State & Lifecycle Hook	18
1. State.....	18
2. Lifecycle	20
IX. Forms.....	23
1. Value binding.....	23
2. onChange Event Handler.....	25
3. Controlled Components	25
4. Uncontrolled Components	25
X. Routing.....	25
1. BrowserRouter	25
2. Route	25

3. Link	26
4. NavLink	26
5. Switch	26
XI. API calls.....	27
1. Rest API	27
2. Request, Response	27
3. Methods	27
4. Axios	27
5. Using axios to consume APIs	28
XII. Redux / Redux toolkit.....	28
1. Redux	28
2. Redux Toolkit	29
XIII. Internationalization	29

I. React CLI (Command – Line – Interface)

1. Install React

- Muốn cài đặt được Reactjs thì trước tiên phải cài đặt Node.js
- Có rất nhiều cách cài đặt và tạo mới 1 project về React

Cách 1: cài đặt, khởi tạo theo cách thông thường

NPX

```
npx create-react-app my-app
```

NPM

```
npm init react-app my-app
```

Yarn

```
yarn create react-app my-app
```

Cách 2: cài đặt, khởi tạo theo Production-grade React Frameworks

Nextjs (full-stack React framework)

```
npx create-next-app
```

Remix (full-stack React framework với các routing lồng nhau)

```
npx create-remix
```

Gasby (React framework cho fast CMS-backend websites)

```
npx create-gatsby
```

Exp (cho native apps, cũng là React frameword)

```
npx create-expo-app
```

Cách 3: cài đặt và khởi tạo thông qua Vite (một công cụ xây dựng nhằm cung cấp trải nghiệm phát triển nhanh hơn và gọn gàng hơn)

NPM : → `npm create vite@latest`

Yarn: → `yarn create vite`

PNPM: → `pnpm create vite`

- Sau khi đã cài đặt thông qua 1 trong 3 cách trên thì màn hình sẽ hiển thị:

```
C:\Users\Neema\Desktop\ViteTut>npm init vite
npx: installed 6 in 20.942s
✓ Project name: ... ViteTutorial
✓ Package name: ... vitetutorial
? Select a framework: » - Use arrow-keys. Return to submit.
  vanilla
  vue
> react
  preact
  lit-element
  svelte
```

→ Từ đó chọn react để tiến hành tiếp tục cài đặt

- Ngoài ra ta còn có thể sử dụng template khi cài đặt:

`npx create-react-app my-app --template [template-name]`

Ví dụ: `npx create-react-app my-app --template typescript`

- Sau khi đã xong hết tất cả các bước trên thì start thôi:

`cd my-app` (chuyển tới folder my-app mới tạo)

`npm start` (start dự án, nếu xài yarn thì đổi là `yarn start`)

2. Folder & structure

- Sau khi tạo xong, project của ta sẽ trông như thế này:

my-app/

 README.md

 node_modules/

 package.json

 public/

index.html
favicon.ico
src/
App.css
App.js
App.test.js
index.css
index.js
logo.svg

- Để xây dựng dự án, các tệp này phải tồn tại với tên tệp chính xác:
 - + public/index.html: là 1 page template (chứa nội dung mà ứng dụng web app)
 - + src/index.js : là 1 js entypoint (điểm đầu vào của toàn ứng dụng)
- node_modules : thư mục này chứa các file có liên quan tới dự án xây dựng webapp bằng reactjs. Bao gồm thư viện, các file cần thiết từ bên thứ 3
- src : chứa các file code làm việc
- package.json : chứa các script và thông tin các module cần sử dụng trong dự án.
- gitignore : chứa các thành phần ta sẽ bỏ qua khi commit code lên git

II. Component

1. Component

- Component trong react có nghĩa là các thành phần để cấu thành nên 1 trang web
- Component gồm 2 loại : Function component và Class component

Class component

- Kết hợp sử dụng class của ES6 để định nghĩa 1 component, được kế thừa từ React.Component
- Sử dụng các lifecycle methods như componentDidMount, componentDidUpdate, componentWillUnmount,... để quản lý vòng đời của thành phần.

- Trước khi Hooks được giới thiệu, Class component là cách thức chính thức để quản lý trạng thái và tác vụ trong React.
- Có thể hỗ trợ một số tính năng phức tạp hơn, nhưng cũng dễ gây hiểu nhầm và có cấu trúc phức tạp hơn so với Function component.
- Có thể nhận đầu vào là 1 đối tượng props (properties) bất kì

Function Component

- Được viết dưới dạng 1 hàm js thông thường hoặc ES6
- Sử dụng hooks như useState, useEffect, useContext,... để quản lý trạng thái và các tác vụ khác mà trước đây chỉ có thể thực hiện trong Class component.
- Ngắn gọn hơn Class component
- Không có khái niệm “this” → giúp tránh nhầm lẫn và loại bỏ việc phải ràng buộc (bind) các phương thức.
- Không hỗ trợ một số lifecycle methods, thay vào đó sử dụng useEffect thay thế để thực hiện các tác vụ liên quan
- Không thể xử lý những công việc phức tạp như quản lý state hoặc sử dụng các life cycle trong component này. Vì thế, component này còn có một tên khác là stateless component (component không có state).
- Từ phiên bản React 16.8 React tung ra React hooks cho phép chúng ta sử dụng state và những tính năng phức tạp khác trong function components.

Lưu ý:

- Khi định nghĩa một component dưới dạng function hoặc class, nó không được phép thay đổi props của chính nó. **Props chỉ dùng để đọc.**
- Mọi React Components đều phải như một pure function đối với props của chúng.

2. Make component

```
1 export default function Profile() {  
2   return (  
3       
7   )  
8 }  
9
```



Ví dụ về cách tạo component và component này tên là Profile

Có thể thấy rằng component đã ở trên, trước tiên phải có cú pháp export default để sau này có thể gọi cái component này ở một file js khác. Sau đó để định nghĩa component thì định nghĩa 1 hàm js function Profile() {}

3. Import and use component

Để sử dụng component Profile ta mới tạo thì có 2 cách

Cách 1 : viết chung 1 file js thì không cần import gì cả, ta gọi thẳng tới


Cách 2 : Viết mỗi component ở mỗi file js khác nhau, ví dụ ta có Profile.js và Galery.js thì khi Galery.js muốn gọi component Profile thì phải import Profile.js vào, đồng thời ở Profile.js phải export ra thì Galery.js mới gọi được tới Profile

Ví dụ dưới là cách 1 :

App.jsDownloadResetFork

```
1 function Profile() {
2   return (
3     
7   );
8 }
9
10 export default function Gallery() {
11   return (
12     <section>
13       <h1>Amazing scientists</h1>
14       <Profile />
15       <Profile />
16       <Profile />
17     </section>
18   );
19 }
```

Amazing scientists



Ví dụ cách 2 :

```
1 import Gallery from './Gallery.js';
2
3 export default function App() {
4   return (
5     <Gallery />
6   );
7 }
8
```

→ Ở component App, ta muốn gọi tới component Gallery thì phải import vào

- Còn 1 loại import nữa, đó là import function thôi (ở trên là import default function, khác so với loại import này), tức là khi file js đó có rất nhiều hàm, ta muốn gọi 1 function thôi, thì ta sử dụng loại import này, cú pháp thì thêm 1 dấu {} thôi, ví dụ sau :

```
App.js  Gallery.js  Profile.js

1  export function Profile() {
2    return (
3      
7    );
8  }
9
```

Gallery muốn gọi tới function Profile đó thì đây là cách import

```
App.js  Gallery.js  Profile.js

1  import { Profile } from './Profile.js';
2
3  export default function Gallery() {
4    return (
5      <section>
6        <h1>Amazing scientists</h1>
7        <Profile />
8        <Profile />
9        <Profile />
10     </section>
11   );
12 }
13
```

III. Component communication

- Có 3 cách giao tiếp giữa các components:

1. Truyền dữ liệu từ component cha sang component con: sử dụng props
2. Truyền dữ liệu từ component con sang component cha: sử dụng callbacks
3. Truyền dữ liệu giữa các component con với nhau: props / callbacks hoặc sử dụng Redux, hoặc sử dụng Context API

Ví dụ về cha sang con:

Component cha

```
export default function Profile() {  
  return (  
    <Avatar  
      person={{ name: 'Lin Lanying', imageId: '1bX5QH6' }}  
      size={100}  
    />  
  );  
}
```

Component con

```
3 function Avatar({ person, size }) {  
4   return (  
5     <img  
6       className="avatar"  
7       src={getImageUrl(person)}  
8       alt={person.name}  
9       width={size}  
10      height={size}  
11    />  
12  );  
13 }
```

→ Có thể thấy rằng ví dụ ở trên thì Profile là component cha, Avatar là component con, Khi component cha gọi tới Avatar thì có truyền data vào trong Avatar là person với size, khi Avatar muốn sử dụng thì phải khai báo ở tham số của hàm, sau đó gọi vào và sử dụng bình thường thôi

Ví dụ từ con sang cha sử dụng callbacks

```
import React, {useState} from 'react';

function Parent() {
  const [message, setMessage] = useState('')

  callbackFunction = (childData) => {
    setMessage(childData)
  },

  return (
    <div>
      <Child parentCallback={callbackFunction}/>
      <p> {message} </p>
    </div>
  );
}
```

→ Lúc đầu định nghĩa 1 callback func ở component cha, sau đó truyền cái callback này vào component con qua props (giống kiểu truyền từ cha sang con)

```
function Child(props) {
  sendData = () => {
    props.parentCallback("Message from Child");
  },

  return (
    // Gọi function sendData bất cứ khi nào bạn muốn truyền dữ liệu lên Parent component (khi có sự
  )
);
```

→ Sau đó qua component con, truyền ngược dữ liệu lại sang component cha bằng cách gọi props.parentCallback()

Về truyền data giữa các component con (siblings)

Cách 1: sử dụng giống như 2 ví dụ trên (cách này làm rối, lộn nhiều cấp, phức tạp, các nhiều các tầng dần càng rối như mớ hỗn loạn)

Cách 2: sử dụng redux (như kiểu một cái store để quản lý state, dễ dàng lấy ra và dễ phân biệt)

Cách 3: sử dụng Context API (tạo ra một "context" (bối cảnh) chứa dữ liệu và giá trị mà muốn chia sẻ trong toàn bộ ứng dụng React. Sau đó, các thành phần có thể truy cập và sử dụng dữ liệu này mà không cần phụ thuộc vào việc truyền dữ liệu qua props)

IV. Binding

1. Data binding

- Là quá trình liên kết dữ liệu giữa các thành phần (components) và trạng thái (state) của ứng dụng. Khi dữ liệu trong trạng thái thay đổi, các thành phần liên quan sẽ được tự động cập nhật và hiển thị dữ liệu mới

- Data binding giúp đồng bộ hóa trạng thái với giao diện người dùng một cách tự động, không cần phải làm thêm bước tương tác với DOM trực tiếp.

- Có 2 loại data binding:

1. **One-way data binding** : là một cơ chế cho phép dữ liệu được truyền từ thành phần cha xuống thành phần con thông qua props. Điều này có nghĩa là khi dữ liệu thay đổi ở thành phần cha, nó sẽ làm cho các thành phần con được cập nhật lại với dữ liệu mới. Tuy nhiên, khi dữ liệu thay đổi ở thành phần con, nó không ảnh hưởng đến thành phần cha.
2. **Two-way data binding**: React không hỗ trợ data binding hai chiều theo cách thông thường (như Angular). Trong React, để thực hiện data binding hai chiều, ta cần sử dụng các thư viện hoặc mô-đun bên ngoài để xử lý việc đồng bộ hóa dữ liệu giữa trạng thái và giao diện người dùng.

→ Cơ chế mà React sử dụng để đạt được data binding một chiều là thông qua việc sử dụng trạng thái (state) của các component. Khi trạng thái thay đổi, React sẽ tự động re-render các component liên quan với trạng thái đó để hiển thị dữ liệu mới.

2. Class and style binding

- là cách liên kết lớp class và style của các components với các điều kiện hoặc trạng thái của ứng dụng

Class binding: cho phép thêm hoặc loại bỏ các lớp (class) CSS cho một thành phần dựa trên một điều kiện (giống như kiểu là mình xài toán tử 3 ngôi ở cái className, quyết định xem nó active hay disabled thì ta sử dụng điều kiện để thay đổi tên của cái className sao cho phù hợp vss CSS của mình)

Style binding: giống với class binding nhưng cái này là thay đổi style, sử dụng css bên trong code luôn, nên cái này thay đổi style cũng dựa trên giống toán tử 3 ngôi

V. Event handling

- Có nghĩa là xử lý sự kiện, giống như là khi người dùng click chuột, khi người dùng di chuột vào, ...

- Giống như handling trong DOM nhưng có một số sự khác biệt như là: viết khác syntax như là viết theo kiểu camel case, truyền jsx vào 1 func để handle event chứ không phải là truyền 1 string

1. Click Event

- Sự kiện Click xảy ra khi người dùng nhấp chuột lên một component nào đó. Để xử lý sự kiện Click → gắn một hàm xử lý vào thuộc tính onClick của thành phần.

Ví dụ:

```
<button onClick={handleClick}>Click me</button>
```

→ Khi ta dùng nhấp chuột lên nút "Click me", hàm handleClick sẽ được gọi và xuất ra thông báo "Button clicked!" trong console.

2. Input Event

- Sự kiện Input xảy ra khi giá trị của một thành phần có thể nhập liệu (ví dụ: input, textarea) thay đổi. Để xử lý sự kiện Input → gắn một hàm xử lý vào thuộc tính onChange của thành phần.

Ví dụ:

```
function MyInput() {
  const [text, setText] = useState("");

  const handleChange = (event) => {
    setText(event.target.value);
  };

  return (
    <input type="text" value={text} onChange={handleChange} />
  );
}
```

→ khi người dùng nhập liệu vào input, hàm handleChange sẽ được gọi và cập nhật trạng thái text với giá trị mới của input.

3. Change Event

- Sự kiện Change xảy ra khi giá trị của một thành phần thay đổi.
- Sự kiện này có thể áp dụng cho nhiều loại thành phần, không chỉ các thành phần có thể nhập liệu.

Ví dụ:

```
function MySelect() {
  const [selectedOption, setSelectedOption] = useState('option1');

  const handleSelectChange = (event) => {
    setSelectedOption(event.target.value);
  };

  return (
    <select value={selectedOption} onChange={handleSelectChange}>
```

```

    <option value="option1">Option 1</option>
    <option value="option2">Option 2</option>
    <option value="option3">Option 3</option>
  </select>
);
}

```

→ Khi người dùng chọn một tùy chọn khác trong thẻ <select>, hàm handleChange sẽ được gọi và cập nhật trạng thái selectedOption với giá trị tùy chọn mới.

VI. Rendering

1. Condition rendering

- Kiểu như điều kiện để render thứ gì đó ra màn hình
- Phổ biến nhất là toán tử 3 ngôi ternary operator (sử dụng if else thông thường vẫn được)

Ví dụ:

```

if (isPacked) {
  return <li className="item">{name} ✓</li>;
}
return <li className="item">{name}</li>;

```

→ Sử dụng if thông thường (rất dài code)

```

return (
  <li className="item">
    {isPacked ? name + ' ✓' : name}
  </li>
);

```

→ Sử dụng ternary operator (code ngắn hơn rất nhiều)

- Ngoài ra còn có toán tử logical là AND operator (&&)

Ví dụ:


```
return (
  <li className="item">
    {name} {isPacked && '✓'}
  </li>
);
```

→ Ở đây có nghĩa isPacked mà true thì sẽ hiện thêm dấu tick ở đằng sau (nói chung là cho code ngắn gọn hơn thôi)

- Ta cũng có thể gán câu điều kiện cho 1 biến nào đó, ví dụ:

```
1 function Item({ name, isPacked }) {
2   let itemContent = name;
3   if (isPacked) {
4     itemContent = name + " ✓";
5   }
6   return (
7     <li className="item">
8       {itemContent}
9     </li>
10  );
```

2. Rendering Lists

- Là quá trình hiển thị một danh sách các thành phần (components) từ một mảng dữ liệu

- Khi có một mảng chứa các dữ liệu và muốn hiển thị chúng dưới dạng danh sách → sử dụng các phương pháp trong React để tái-render mỗi thành phần dựa trên dữ liệu trong mảng đó.

→ Sử dụng filter() và map() để filter và transform

Ví dụ:

```
const people = [{
  id: 0,
  name: 'Creola Katherine Johnson',
  profession: 'mathematician',
}, {
  id: 1,
  name: 'Mario José Molina-Pasquel Henríquez',
  profession: 'chemist',
}, {
  id: 2,
  name: 'Mohammad Abdus Salam',
  profession: 'physicist',
}, {
  name: 'Percy Lavon Julian',
  profession: 'chemist',
}, {
  name: 'Subrahmanyam Chandrasekhar',
  profession: 'astrophysicist',
}
];
```

→ Đây là 1 mảng people, trong đó chứa id, name và profession theo dạng key-value

```
const chemists = people.filter(person =>
  person.profession === 'chemist'
);
```

→ Sử dụng filter để lọc ra những người 'chemist'

```
const listItems = chemists.map(person =>
  <li>
    <img
      src={getImageUrl(person)}
      alt={person.name}
    />
    <p>
      <b>{person.name}</b>
      { ' ' + person.profession + ' ' }
      known for {person.accomplishment}
    </p>
  </li>
);
```

→ Sử dụng map để có thể lặp ra các thuộc tính của mảng person, từ đó lọc tới cuối mảng

Bước cuối cùng là return về listItems như đã định nghĩa

```
return <ul>{listItems}</ul>;
```

* Lưu ý: Phải luôn truyền key vào mỗi khi muốn redering lists, key phải là unique (giống như ID)

VII. State & Lifecycle Hook

1. State

- Là một đối tượng JavaScript dùng để lưu trữ và quản lý dữ liệu có thể thay đổi trong một component, cho phép component lưu trữ thông tin và cập nhật dữ liệu mà không cần phải tương tác trực tiếp với DOM.
- Khi dữ liệu trong state thay đổi → tự động re-render component để hiển thị dữ liệu mới.

State trong Class component

- Trước phiên bản React 16.8 (khi đó chưa có hook useState), state thường được sử dụng trong class component bằng cách sử dụng thuộc tính state của component và phương thức setState() để cập nhật state.

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

```
class MyClassComponent extends Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }  
}
```

```

increment = () => {
  this.setState({ count: this.state.count + 1 });
};

render() {
  return (
    <div>
      <p>Count: {this.state.count}</p>
      <button onClick={this.increment}>Increment</button>
    </div>
  );
}
}

```

→ state ở trên được khai báo trong constructor với state là count=0 (định nghĩa count trong state)

→ Khi người dùng nhấp vào nút "Increment", phương thức increment sẽ được gọi, và setState được sử dụng để cập nhật giá trị của state count. Sau đó, React sẽ tái-render component và hiển thị giá trị mới.

State trong Functional Component

- Trước khi phiên bản React 16.8, Functional Components không hỗ trợ sử dụng state. Tuy nhiên, từ phiên bản React 16.8 trở đi, React giới thiệu "Hooks" và cung cấp hook useState để cho phép Functional Component sử dụng state.

Ví dụ:

- So với state component thì khai báo state đơn giản và ngắn gọn hơn

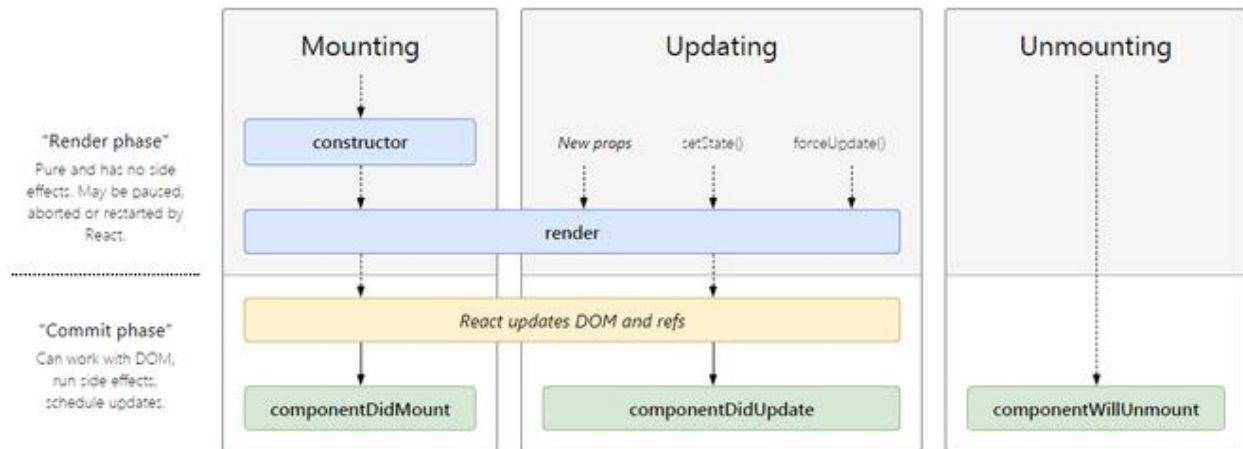
```
const [count, setCount] = useState(0);
```

Còn với hàm increment thì xài cú pháp khác:

```
const increment = () => {
  setCount(count + 1);
};
```

- Còn lại thì giống với Class component, chỉ khác những cái gì mới đề cập ở trên thôi

2. Lifecycle



- Lifecycle của component trong reactjs là quá trình từ khi tạo ra, thay đổi và hủy bỏ component. Gồm 3 giai đoạn:

1. Mounting: Giai đoạn này xảy ra khi một component được tạo và chèn vào cây DOM để hiển thị lần đầu tiên, khi component mounting thì sẽ gọi ra các hàm (`constructor`, `render`, `componentDidMount`)

→ hàm `constructor` để khai báo các state, các props của component (

→ hàm `componentDidMount` để định nghĩa 1 component được tạo ra sau khi đi qua hàm `constructor` và `render` lần đầu tiên, thường được dùng để gọi API lấy data, `setState` để cập nhật state

→ hàm `didmount` này chỉ chạy duy nhất 1 lần khi component được tạo ra

2. Updating: Giai đoạn này xảy ra khi component đã được tạo và nhận được cập nhật về state hoặc props.

→ có thể không gọi hoặc gọi nhiều lần nếu có sự update đến từ component (khi props thay đổi, khi state thay đổi, `forceUpdate`)

3. Unmounting: Giai đoạn này xảy ra khi component bị gỡ bỏ khỏi cây DOM, tức là component không còn tồn tại trên giao diện người dùng.

→ Khi ta không render component hoặc chuyển trang thì component sẽ bị hủy bỏ để render nội dung mới lên.

→ Dùng để hủy timeout, clearInterval. (nếu không hủy bỏ thì sẽ bị chạy hoài liên tục), reset dữ liệu nếu cần thiết.

→ hàm WillUnmount() chỉ chạy 1 lần duy nhất khi component trong vòng đời của component. Tương tự Mount() chỉ chạy 1 lần duy nhất.

Đối với Functional Component thì **useEffect** được sử dụng để thay thế cho các lifecycle method như: componentDidMount, componentDidUpdate, componentWillReceiveProps, componentWillUnmount.

a) **ComponentDidMount()**

Tương ứng với lifecycle này là cách viết sau

```
useEffect(() => {  
  
    // Bạn viết code xử lý logic tại đây  
  
}, []);
```

b) **ComponentDidUpdate()**

Tương ứng với lifecycle này là cách viết sau:

```
useEffect(() => {  
  
    // Bạn viết code xử lý logic tại đây  
  
});
```

c) **ComponentWillUnmount()**

Tương ứng là:

```
useEffect(() => {

    // hàm được trả về sẽ được gọi khi component unmount

    return () => {

        // Bạn viết code xử lý logic tại đây khi component unmount.

    }

}, [])
```

VIII. Hooks

- Theo định nghĩa của React: Hooks là các hàm mà cho phép bạn “hook into (móc vào)” trạng thái của React và các tính năng vòng đời từ các hàm components. Hooks không hoạt động bên trong class
- Giống như kiểu giúp ta sử dụng các tính năng trong functional component mà trước đây chỉ có class component mới dùng được
- Hooks có rất nhiều như là useState, useEffect, useContext, useReducer,...

1. useEffect

- useEffect là một React Hook cho phép đồng bộ hóa một thành phần với một hệ thống bên ngoài (synchronize)
- Thực hiện các tác vụ không thuộc về render như: gọi API, tương tác DOM, sự kiện, ...
- Giúp thực hiện các tác vụ không đồng bộ và cập nhật trạng thái của component dựa trên sự thay đổi của state và props
- Cú pháp: `useEffect(setup, dependencies?)`
 - setup: Là một hàm callback sẽ được gọi sau mỗi lần component render. Giống như code với phương thức `componentDidMount`, `componentDidUpdate`, và `componentWillUnmount` trong class component
 - dependencies: Là một mảng chứa các giá trị tham chiếu. Nếu có một hoặc nhiều giá trị trong mảng thay đổi sau mỗi lần render, setup sẽ được gọi lại. Nếu dependencyArray không được cung cấp, setup sẽ được gọi sau mỗi lần render.
 - useEffect sẽ chạy setup sau mỗi lần render của component. Điều này bao gồm cả lần render đầu tiên (component được mount) và sau mỗi lần render kế tiếp (component được update). Tuy nhiên, nếu muốn setup chỉ chạy một lần khi component mount và không chạy lại sau các lần render tiếp theo → có thể truyền vào một mảng rỗng [] làm dependencyArray.

2. useState

- Sử dụng trong functional components để tạo và quản lý trạng thái (state) của component.

- Cú pháp: `const [state, setState] = useState(initialState);`

→ state: giá trị hiện tại của state, dc khởi tạo bằng initialState. Khi component render lần đầu tiên, state sẽ chứa initialState, và sau đó có thể thay đổi khi bạn gọi setState để cập nhật state.

→ setState: Là hàm được cung cấp bởi useState, dùng để cập nhật giá trị của state. Khi gọi setState, React sẽ thực hiện re-render component với giá trị state mới, và trạng thái giao diện người dùng sẽ được cập nhật tương ứng

→ useState chỉ chạy một lần duy nhất trong quá trình render, do đó giá trị khởi tạo của state chỉ được đánh giá một lần khi component mount. Tuy nhiên, nếu gọi useState trong một if hoặc một vòng lặp, React sẽ hiểu mỗi lần render là một useState mới, dẫn đến việc mất state sau mỗi lần render. Điều này có thể giải quyết bằng cách sử dụng hooks ở cấp cao hơn hoặc sử dụng useReducer.

IX. Forms

1. Value binding

- khi làm việc với các form (biểu mẫu) để thu thập thông tin từ người dùng, giá trị của các input (ví dụ: text, checkbox, radio, select, ...) cần phải được liên kết với state của component để có thể theo dõi và xử lý các thay đổi. Quá trình này được gọi là "value binding" (liên kết giá trị).

- Đảm bảo rằng giá trị hiển thị của các input phụ thuộc vào trạng thái của component, và mọi thay đổi trong input sẽ được cập nhật vào state của component.

Ví dụ:

```
function MyForm() {  
  // Khởi tạo state "name" với giá trị ban đầu là rỗng  
  const [name, setName] = useState("");  
  
  const handleChange = (event) => {  
    // Cập nhật giá trị "name" khi người dùng nhập liệu vào input
```



```

    setName(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    // Xử lý dữ liệu sau khi người dùng submit form
    console.log('Name submitted:', name);
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Name:
        <input
          type="text"
          value={name} // Liên kết giá trị của input với state "name"
          onChange={handleChange} // Xử lý sự kiện thay đổi giá trị của input
        />
      </label>
      <button type="submit">Submit</button>
    </form>
  );

```

→ Trong ví dụ trên, chúng ta sử dụng useState để tạo state "name" và liên kết giá trị của input text với state "name" thông qua thuộc tính value={name} của input.

→ Khi người dùng nhập liệu vào input, hàm handleChange được gọi và cập nhật giá trị "name" trong state, dẫn đến việc hiển thị giá trị mới của input. Khi người dùng nhấn nút "Submit", hàm handleSubmit được gọi và chúng ta có thể sử dụng giá trị "name" trong state để xử lý dữ liệu của form.

2. onChange Event Handler

- Sử dụng onChange event handler để lắng nghe sự kiện thay đổi giá trị của input và cập nhật state của component dựa trên giá trị mới của input.

3. Controlled Components

- Controlled components là mô hình mà giá trị của các input phụ thuộc hoàn toàn vào state của component. Bằng cách liên kết giá trị của input với state và lắng nghe sự kiện onChange, bạn đảm bảo rằng giá trị của input luôn được cập nhật và điều khiển bởi state của component.

4. Uncontrolled Components

- Uncontrolled components là mô hình mà giá trị của input không phụ thuộc vào state. Ta không cần liên kết giá trị của input với state và sử dụng ref để truy cập giá trị của input khi cần thiết.

*Ngoài ra còn có 1 số lib hỗ trợ về form như là React Hook Form, Formik

X. Routing

- React không hỗ trợ Router

→ Sử dụng thư viện react-router

- React-router là một thư viện routing, được sử dụng để quản lý việc điều hướng và xác định cách các components sẽ được hiển thị dựa trên URL hiện tại của ứng dụng.

→ Nói chung là giống điều hướng trang, giống thanh menu vậy đó

Các components chính trong React-router:

1. BrowserRouter

- Là một thành phần bao bọc toàn bộ ứng dụng và sử dụng HTML5 history API để đồng bộ hóa URL hiện tại với trạng thái của ứng dụng → cho phép React Router theo dõi URL và điều hướng trang một cách rất mượt mà không cần tải lại toàn bộ trang.

2. Route

- Định nghĩa một ánh xạ (mapping) giữa một URL và một Component. Điều đó có nghĩa là khi người dùng truy cập theo một URL trên trình duyệt, một Component tương ứng sẽ được render trên giao diện.

Ví dụ:

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

Trong đó:

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

3. Link

- Dùng để tạo các liên kết trong ứng dụng. Thay vì sử dụng thẻ `<a>` thông thường → sử dụng Link để điều hướng giữa các trang của ứng dụng mà không tải lại toàn bộ trang.

```
<Link to="/about">About</Link>
```

Link to đó giống với a href

4. NavLink

- NavLink thì rất giống với Link về cách sử dụng, nhưng NavLink tốt hơn vì nó hỗ trợ thêm một số thuộc tính như là **activeClassName** và **activeStyle** 2 thuộc tính này giúp cho khi mà nó trùng khớp thì nó sẽ được active lên và chúng ta có thể style cho nó.

5. Switch

- Thành phần Switch được sử dụng để bao bọc các thành phần Route. Nó giúp chỉ hiển thị thành phần đầu tiên mà khớp với URL hiện tại trong Switch → có ích khi muốn chỉ hiển thị một route duy nhất cho mỗi URL.

Các tính năng

- Nested Routes: Routes lồng lẫn nhau thôi
- Dynamic Routes: Routes động, như kiểu truyền vào userID
- URL parameters : tham số URL

XI. API calls

1. Rest API

- Giống như kiểu là nguyên tắc để thiết kế API , được biểu diễn dưới dạng JSON,
- Được truy cập thông qua endpoint với các phương thức HTTP: GET , POST, PUT, DELETE,...

2. Request, Response

- Khi gọi API, mình sẽ gửi request tới server từ client đến server
- Request bao gồm các thông tin như method, endpoint, data gửi đi (POST, PUT),
→ sau khi nhận request, server xử lý và trả về response cho client (res cũng bao gồm status code, data trả về và header)

3. Methods

- Với Restful API thì các methods này mới nêu ở trên bao gồm GET, POST, PUT, DELETE là sử dụng nhiều nhất
- GET: lấy data
- PUT: cập nhật data đã tồn tại sẵn
- POST: tạo mới 1 cái gì đó
- DELETE: xóa data

4. Axios

- Thư viện HTTP client để gọi API
- Hỗ trợ xử lý các yêu cầu HTTP với các methods get post put delete,...

5. Using axios to consume APIs

- Cài đặt → npm install axios / yarn install axios

- Các phương thức chính liên quan :

- axios.get(url, config): Gửi yêu cầu HTTP GET tới URL được chỉ định và trả về một Promise với phản hồi từ server.
- axios.post(url, data, config): Gửi yêu cầu HTTP POST tới URL được chỉ định với dữ liệu (payload) được gửi kèm và trả về một Promise với phản hồi từ server.
- axios.put(url, data, config): Gửi yêu cầu HTTP PUT tới URL được chỉ định với dữ liệu (payload) được gửi kèm và trả về một Promise với phản hồi từ server.
- axios.delete(url, config): Gửi yêu cầu HTTP DELETE tới URL được chỉ định và trả về một Promise với phản hồi từ server.

- Axios có thể config được, ví dụ như Interceptors của axios, giúp xử lý các req và res trước khi đc handle bởi catch hoặc then

XII. Redux / Redux toolkit

1. Redux

- Là 1 PATTERN

- Redux là 1 lib js để quản lý và cập nhật state

- Giống như 1 cái kho để chứa state

- Quản lý Global state, như kiểu là các component tại mọi nơi có thể truy xuất và cập nhật, giải quyết vấn đề truyền dữ liệu giữa các component con với nhau

→ Sử dụng redux khi số lượng state quá nhiều và state được sử dụng ở rất nhiều nơi trong dự án

→ Sử dụng redux khi state thường xuyên phải cập nhật, logic code cập nhật state phức tạp, dự án code lớn

Kiến trúc của Redux:

Reducers: là 1 func, có 2 tham số, tham số đầu là state hiện tại, tham số thứ 2 là action. Func này được sử dụng để cập nhật lại giá trị state ở kho chung dựa trên cái action mình truyền vào là gì. Khi đã xác định được action thì ta sẽ cập nhật lại dựa theo kiểu copy (immutable)

- Nhưng mà cái reducer này lại có nguyên tắc của nó: State mới này luôn luôn được cập nhật dựa trên giá trị state trước đó và không bao giờ được thay đổi trực tiếp giá trị ban đầu của state vì nó immutability, phải tạo ra 1 phiên bản copy và không được code theo kiểu bất đồng bộ trong reducer này

Action : chỉ đơn giản là 1 object, gửi dữ liệu đến store Redux

- Thường có 2 keys là type và payload

Dispatch : về cơ bản thì nó cũng chỉ là 1 func

- cách duy nhất để cập nhật state trong store là phải sử dụng dispatch func này

- tham số truyền vào dispatch này là 1 action

- việc dispatch này ta coi như kiểu là bắn đi 1 cái action từ phía UI

2. Redux Toolkit

- Giống như redux, sinh ra để giúp redux config ngắn gọn hơn

- Giảm bớt sự phức tạp khi làm việc với redux

XIII. Internationalization

- Được gọi là i18next (đời kế tiếp của i18n), là 1 framework về Internationalization, được viết bằng và dành cho JS

- hỗ trợ nhiều ngôn ngữ và văn hóa

- phù hợp , tối ưu cho server-side rendering

Cài đặt:

- Theo npm: # npm

\$ npm install react-i18next i18next --save

- Theo CDN:

<https://unpkg.com/react-i18next@13.0.2/react-i18next.min.js>

<https://unpkg.com/react-i18next@13.0.2/react-i18next.js>

- Để i18next cấu hình sẵn trong tất cả các component, phải bọc component App bằng component I18nextProvider

- Các bước thực hiện: cài đặt, cấu hình i18n → Chuẩn bị các bản dịch → Sử dụng api do thư viện cung cấp → Xử lý ngôn ngữ ưu tiên → Kiểm tra & bảo đảm tính consistantly của bản dịch