ỦY BAN NHÂN DÂN TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC SÀI GÒN

Mã sinh viên	**:**	**3121410376**
Sinh viên	**:**	**Lý Thanh Phát**
Nhóm/lớp	**:**	**1**
Giảng viên hướng dẫn	**:**	**Phạm Thi Vương**
Năm học	**:**	**2024-2025**

ĐỀ CƯƠNG TIỂU LUẬN HỌC PHẦN CÁC CÔNG NGHỆ LẬP TRÌNH HIỆN ĐẠI

TÌM HIỂU REACTJS

NGÀNH: CÔNG NGHỆ THÔNG TIN

TRÌNH ĐỘ ĐÀO TẠO: ĐẠI HỌC

TP. HỒ CHÍ MINH, THÁNG 11 NĂM 2024

Mục lục

A. Mở Đầu

I. Lý do chọn đề tài

Việc chọn đề tài ReactJS cho tiểu luận không chỉ xuất phát từ sự phổ biến của thư viện này trong cộng đồng phát triển ứng dụng web, mà còn từ những lợi ích vượt trội mà nó mang lại cho quy trình phát triển phần mềm. Trong bối cảnh công nghệ ngày càng phát triển, yêu cầu về việc tạo ra các ứng dụng web mượt mà, tương tác và có khả năng mở rộng là rất cần thiết. ReactJS đáp ứng đầy đủ những yêu cầu này nhờ vào khả năng quản lý giao diện người dùng thông qua các thành phần độc lập, cho phép tái sử dụng mã nguồn một cách hiệu quả.

Hơn nữa, ReactJS hỗ trợ tốt cho việc tối ưu hóa hiệu suất thông qua cơ chế Virtual DOM, giúp giảm thiểu thời gian phản hồi của ứng dụng và nâng cao trải nghiệm người dùng. Ngoài ra, với sự phát triển mạnh mẽ của cộng đồng, các nhà phát triển có thể dễ dàng tìm thấy tài liệu, hướng dẫn và giải pháp cho các vấn đề phát sinh trong quá trình phát triển.

Đề tài này không chỉ giúp tôi hiểu rõ hơn về cách thức hoạt động và lợi ích của ReactJS, mà còn cho phép tôi khám phá sâu hơn về cách thư viện này được áp dụng trong các dự án thực tế. Qua đó, tôi hy vọng sẽ trang bị cho mình những kiến thức cần thiết để áp dụng ReactJS vào các dự án phát triển ứng dụng web trong tương lai.

II. Mục tiêu đề tài

Mục tiêu của việc chọn đề tài ReactJS không chỉ đơn thuần là tìm hiểu về một thư viện JavaScript nổi tiếng, mà còn nhằm đạt được những mục tiêu cụ thể sau:

- **Nâng cao kiến thức chuyên môn:** Đề tài này giúp tôi hiểu rõ về cấu trúc, nguyên lý hoạt động và các tính năng nổi bật của ReactJS. Qua việc nghiên cứu, tôi mong muốn có cái nhìn sâu sắc hơn về cách thức xây dựng giao diện người dùng hiện đại.
- **Khám phá ứng dụng thực tiễn:** Tôi muốn tìm hiểu cách mà ReactJS được áp dụng trong các dự án thực tế, từ việc phát triển ứng dụng web đơn giản đến các ứng dụng phức tạp như mạng xã hội hay thương mại điện tử. Điều này sẽ giúp tôi thấy được sự linh hoạt và hiệu quả của ReactJS trong nhiều lĩnh vực khác nhau.
- **Phát triển kỹ năng lập trình:** Qua quá trình nghiên cứu và thực hành với ReactJS, tôi muốn cải thiện kỹ năng lập trình của mình, từ việc viết mã sạch, tối ưu hóa hiệu suất ứng dụng cho đến việc áp dụng các phương pháp phát triển hiện đại như lập trình hướng đối tượng và lập trình hàm.
- **Đánh giá và so sánh với các công nghệ khác:** Mục tiêu là so sánh ReactJS với các thư viện và framework khác như Angular hay Vue.js, từ đó hiểu được ưu và nhược điểm của từng công nghệ, giúp đưa ra quyết định hợp lý khi lựa chọn công cụ cho dự án phát triển ứng dụng.

- **Chuẩn bị cho tương lai nghề nghiệp:** Cuối cùng, việc nghiên cứu ReactJS sẽ giúp tôi trang bị kiến thức và kỹ năng cần thiết để đáp ứng nhu cầu thị trường lao động, nơi ReactJS đang trở thành một trong những kỹ năng rất được ưa chuộng.

III. Tổng quan về ReactJS

1. Lịch sử phát triển

ReactJS được phát triển bởi Facebook vào năm 2011 nhưng chính thức được công bố vào năm 2013. Sự ra đời của React xuất phát từ nhu cầu của Facebook trong việc cải thiện hiệu suất và khả năng mở rộng của giao diện người dùng cho ứng dụng mạng xã hội của mình. Trước khi có React, Facebook gặp khó khăn trong việc xử lý các thay đổi giao diện phức tạp do lượng người dùng ngày càng tăng và yêu cầu về tính năng ngày càng đa dạng.

- Giai đoạn khởi đầu (2011 - 2013)

Vào khoảng năm 2011, Facebook đã bắt đầu phát triển React như một giải pháp cho các vấn đề về hiệu suất trong việc xây dựng giao diện. React được thiết kế với ý tưởng "Component-based architecture" (kiến trúc dựa trên thành phần), cho phép các nhà phát triển chia nhỏ giao diện thành các thành phần độc lập có thể tái sử dụng. Sự ra đời của Virtual DOM là một trong những điểm đột phá, giúp giảm thiểu việc cập nhật trực tiếp lên DOM thực, từ đó tối ưu hóa hiệu suất của ứng dụng.

- Phát hành và phổ biến (2013 - 2015)

Vào tháng 5 năm 2013, React được công bố như một dự án mã nguồn mở, thu hút sự chú ý của cộng đồng lập trình viên. Ngay sau khi ra mắt, React nhanh chóng được áp dụng trong nhiều dự án lớn, không chỉ tại Facebook mà còn ở các công ty khác như Instagram và Airbnb. Năm 2015, React đã chính thức phát hành phiên bản 0.14, với nhiều cải tiến và tính năng mới, như hỗ trợ cho việc sử dụng React trên cả server (Node.js) và client (trình duyệt).

- Sự trưởng thành và phát triển bền vững (2016 - nay)

Từ năm 2016 trở đi, React tiếp tục phát triển mạnh mẽ với việc ra mắt React 16, đi kèm với nhiều cải tiến đáng kể, bao gồm hỗ trợ cho "Error Boundaries" và "Fragments", giúp cải thiện khả năng quản lý lỗi và tổ chức cấu trúc của ứng dụng. Năm 2018, React 16.8 được giới thiệu với tính năng Hooks, cho phép các nhà phát triển sử dụng state và các tính năng khác của React mà không cần phải tạo một thành phần lớp (class component). Sự phát triển không ngừng của React được hỗ trợ bởi một cộng đồng lớn mạnh và nhiều thư viện phụ trợ như Redux (quản lý trạng thái), React Router (điều hướng) và Next.js (SSR - Server Side Rendering). Những công cụ này đã giúp React trở thành một trong những thư viện phổ biến nhất trong phát triển ứng dụng web hiện đại.

2. Khái niệm và nguyên lý hoạt động

ReactJS là một thư viện JavaScript dùng để xây dựng giao diện người dùng, đặc biệt là giao diện người dùng theo mô hình component. Với mô hình **Virtual DOM** (DOM ảo), React giúp tối ưu hóa việc cập nhật giao diện, giúp ứng dụng hoạt động nhanh chóng và mượt mà hơn. ReactJS thường được sử dụng để phát triển các ứng dụng **Single Page**

Application (SPA), nơi chỉ một trang web duy nhất được tải, và mọi thay đổi trong giao diện đều được xử lý một cách hiệu quả mà không cần tải lại toàn bộ trang.

B.Thân Bài:

Các công việc đã làm trong đồ án

1. Xây dựng cấu trúc bài báo cáo

- **Mở đầu**: Giới thiệu về React.js, lý do chọn React cho dự án, và tầm quan trọng của việc hiểu các thành phần cơ bản của React.
- **Thân bài**: Chia bài viết thành các phần nhỏ theo các mục bạn đã đề ra, bao gồm thiết lập môi trường, các hook cơ bản, quản lý sự kiện, xử lý bất đồng bộ, và tối ưu hóa. Mỗi mục có thể được chia nhỏ thành các phần chi tiết để đi sâu vào từng khía cạnh.
- **Kết luận**: Tổng kết những gì đã tìm hiểu, nêu ra các ưu điểm và hạn chế của React.js dựa trên kiến thức đã trình bày, và kết luận về việc ứng dụng React trong dự án của bạn.

2. Tìm hiểu về thiết lập môi trường cho một dự án React.js

Để cài đặt môi trường React và sử dụng Create React App, bạn có thể làm theo các bước sau:

2.1. Cài đặt Node.js

Đầu tiên, bạn cần cài đặt **Node.js**. Create React App yêu cầu Node.js để chạy.

- Truy cập trang web Node.js và tải xuống phiên bản LTS (Long Term Support) phù hợp với hệ điều hành của bạn.
- Cài đặt Node.js bằng cách làm theo hướng dẫn trên màn hình.

2.2. Kiểm tra cài đặt

Sau khi cài đặt, mở terminal (hoặc Command Prompt trên Windows) và kiểm tra phiên bản Node.js và npm (Node Package Manager):

Terminal
node -v npm -v

Nếu bạn thấy phiên bản hiển thị, nghĩa là cài đặt thành công.

2.3. Khởi tạo 1 ứng dụng React với Vite

Chạy lệnh terminal sau để tạo project

Terminal
npm create vite

Sau đó vite sẽ hiện lên cho bạn nhập vào tên của dự án

Terminal
> npx

```
> create-vite

? Project name: › vite-project
//<vite-project> tên dự án của bạn
```

Sau đó bạn sẽ chọn loại dự án bạn mong muốn ở đây là reactjs,

Terminal

```
? Select a framework: › - Use arrow-keys. Return to submit.
   Vanilla
   Vue
❯  React
   Preact
   Lit
   Svelte
   Solid
   Qwik
   Others
```

Chọn tiếp loại ngôn ngữ

Terminal

```
? Select a variant: › - Use arrow-keys. Return to submit.
   TypeScript
   TypeScript + SWC
❯   JavaScript
   JavaScript + SWC
   Remix ↗
```

Bước cuối cùng

Terminal

```
Done. Now run:

  cd <vite-project>
//<vite-project> tên dự án của bạn đã tạo
  npm install
  npm run dev
```

2.4. Chạy ứng dụng React

Di chuyển vào thư mục dự án bạn vừa tạo:

Terminal
cd <vite-project> //<vite-project> tên project của bạn

Nhập lệnh sau để chạy:

Terminal
npm run dev

- Lệnh này sẽ mở một cửa sổ trình duyệt và chạy ứng dụng React trên địa chỉ http://localhost:????

Lưu ý: **????** sẽ là port mà react lúc khởi chạy sẽ thông báo.

3. Tìm hiểu về các hook - useState, useEffect, useContext

3.1. Hooks là gì ?

Hook trong ReactJS là một tính năng được giới thiệu từ phiên bản **React 16.8**, mang lại khả năng sử dụng các tính năng quan trọng của React như **state** và các **phương thức vòng đời** (lifecycle methods) mà không cần phải sử dụng các **class component** truyền thống. Trước khi có Hooks, để quản lý trạng thái hoặc thực hiện các tác vụ như cập nhật DOM hoặc gọi API khi component được render, lập trình viên phải sử dụng các component dạng class với các phương thức như **componentDidMount**, **componentDidUpdate**, và **componentWillUnmount**.

Tuy nhiên, việc sử dụng class components đôi khi có thể phức tạp và khó quản lý, đặc biệt là khi các logic xử lý trạng thái và vòng đời trở nên phức tạp hoặc bị phân tán giữa nhiều phương thức khác nhau. Điều này dẫn đến sự khó khăn trong việc tái sử dụng logic, tối ưu hóa mã, và bảo trì ứng dụng.

a. Hook mang lại lợi ích như sau:

- **Giảm bớt sự phức tạp của class components:** Với Hook, bạn không cần phải chuyển đổi giữa các functional components và class components khi cần quản lý trạng thái hoặc xử lý các tác vụ vòng đời. Hook cho phép bạn giữ logic quản lý trạng thái và vòng đời trong cùng một component chức năng (function component), giúp mã trở nên gọn gàng và dễ hiểu hơn.
- **Tái sử dụng logic dễ dàng hơn**: Trước đây, việc tái sử dụng logic trong class components yêu cầu phải sử dụng **Higher-Order Components (HOCs)** hoặc **Render Props**, cả hai cách tiếp cận này đều có thể dẫn đến mã phức tạp và khó hiểu. Hook cung cấp một cách tiếp cận tự nhiên hơn để tái sử dụng logic thông qua các **custom hooks**.
- **State và Side Effects**: Hooks như **useState** và **useEffect** là hai trong số các Hook phổ biến nhất.
    - **useState** cho phép bạn thêm **state** vào một **functional component**, giúp quản lý trạng thái dễ dàng hơn mà không cần phải chuyển sang **class component**.
    - **useEffect** thay thế các phương thức vòng đời truyền thống như **componentDidMount, componentDidUpdate**, và **componentWillUnmount**.

Nó cho phép bạn thực hiện các **side effects** (tác vụ phụ) như gọi API, đăng ký sự kiện, và dọn dẹp tài nguyên sau khi component được render hoặc cập nhật.

- **Hỗ trợ tất cả các tính năng của React**: Hook cho phép bạn sử dụng tất cả các tính năng mạnh mẽ của React như state management, context API, memoization, refs, và nhiều hơn nữa chỉ với các functional components. Điều này giúp cho việc phát triển ứng dụng trở nên nhất quán và đơn giản hơn khi bạn không phải lựa chọn giữa việc sử dụng class component hay functional component.
- **Dễ dàng học và sử dụng**: Hook đã đơn giản hóa rất nhiều việc học React. Những người mới bắt đầu có thể dễ dàng tiếp cận React mà không cần phải hiểu sự khác biệt giữa class components và functional components. Điều này giúp cho việc học tập và sử dụng React trở nên trực quan và ít rào cản hơn.

b. Các Hook phổ biến trong React bao gồm:

- **useState**: Quản lý state trong functional components.
- **useEffect**: Xử lý các tác vụ vòng đời và side effects như gọi API hoặc tương tác với DOM.
- **useContext**: Sử dụng Context API để chia sẻ dữ liệu giữa các components mà không cần phải truyền props qua nhiều cấp độ.
- **useReducer**: Thay thế cho useState khi cần quản lý trạng thái phức tạp hoặc cần xử lý các actions, tương tự như Redux.
- **useMemo** và **useCallback**: Tối ưu hóa hiệu suất bằng cách ghi nhớ giá trị hoặc hàm để tránh tính toán lại không cần thiết.
- **useRef**: Truy cập trực tiếp đến DOM hoặc lưu trữ giá trị mà không gây render lại component.

c. Kết luận

Hook là một cải tiến quan trọng trong React, giúp lập trình viên có thể viết mã dễ dàng hơn, tái sử dụng logic tốt hơn và giảm thiểu sự phức tạp của component. Chúng không chỉ làm cho việc quản lý trạng thái trở nên linh hoạt hơn mà còn hỗ trợ đầy đủ các tính năng mạnh mẽ của React mà không cần phải sử dụng class component. Sự ra đời của Hook đã thay đổi cách tiếp cận phát triển ứng dụng React, giúp mã nguồn trở nên đơn giản và dễ bảo trì hơn, đặc biệt là đối với các dự án lớn và phức tạp.

3.2. useState

useState là một trong những **React Hook** cơ bản nhất và được sử dụng để quản lý **state** (trạng thái) trong **functional components**. Trước khi có **useState**, trạng thái chỉ có thể được quản lý trong **class components**. **useState** đã giúp các **functional components** trở nên mạnh mẽ hơn, cho phép chúng quản lý và theo dõi trạng thái mà không cần chuyển sang class component.

i. Cách sử dụng useState

useState là một **hook** được gọi bên trong functional component để tạo và quản lý một trạng thái cụ thể.

```
const [<state>, <setState>] = useState(<initialState>);
```

3.3. useEffect

i. Cách sử dụng

useEffect là một trong những **React Hook** quan trọng được sử dụng để xử lý **side effects** trong các functional components. Nó được giới thiệu từ React phiên bản 16.8 và cho phép bạn thực hiện các hành động như gọi API, cập nhật DOM, hoặc thiết lập các subscription mà trước đây chỉ có thể thực hiện trong **class components** với các phương

thức vòng đời như **componentDidMount**, **componentDidUpdate**, và **componentWillUnmount**.
Có 3 loại useEffect:

- useEffect(function) *(1)*
- useEffect(function,[]) *(2)*
- useEffect(function,[dependencies]) *(3)*

Ở đây:

- <function> là một hàm có giá trị trả về
- <dependencies> là các biến mà bạn đã tạo

	(1)	(2)	(3)
Khác nhau	Callback khi component được render	Callback 1 lần duy nhất khi mounted	chỉ callback khi mà giá trị dependencies bị thay đổi
Giống nhau	Callback luôn được gọi khi component được mounted và thực hiện sau return		

- **<state>:** tên biến lưu trữ giá trị.
- **<setState>**: hàm gọi thay đổi giá trị lưu trữ.
- **<initialState>:** gía trị khởi tạo ban đầu cho biến, và thể hiện kiểu dữ liệu muốn khởi tạo.
- **Lưu ý:** phải đặt tên [state, setState] theo quy tắc ví dụ: [count, setCount], [img,setImg] để dễ dàng kiểm soát.

3.4. useContext

i. Cách sử dụng

useContext là một hook trong React, được sử dụng để truy cập vào context API, cho phép chia sẻ dữ liệu giữa các component mà không cần phải truyền props qua nhiều cấp (prop drilling). Context API thường được sử dụng khi bạn có một số dữ liệu hoặc trạng thái muốn chia sẻ trên toàn bộ hoặc một phần lớn của ứng dụng, như là dữ liệu về người dùng đã đăng nhập, ngôn ngữ hiển thị, hoặc chủ đề (theme).

- **Tạo một context**: Sử dụng React.createContext để tạo ra một context.

```
export const <nameContext> = createContext()
//<nameContext> là tên biến context mà bạn muốn đặt
```

- **Cung cấp context (Provider)**: Dùng component Provider của context đó để cung cấp dữ liệu xuống các component con.

```
// <nameContext> là tên context bạn đã tạo
// <{value}> là danh sách các biến bạn sẽ truyền vào context
< <nameContext>.Provider value=<{value}>>

//Các component con có thể nhận được context của bạn
      {children}
</nameContext.Provider>
```

- **Sử dụng context (useContext)**: Tại bất kỳ component con nào, bạn có thể dùng hook useContext để truy xuất giá trị từ context.

```
import {useContext} from 'react'
import {nameContext} from './App.js'

//Đây là cách gọi ra context bạn đã tạo
const name = useContext(nameContext)
```

4. Tìm hiểu về Event Handling

6.1. Sự kiện phạm vi bên trong component

a. Khái niệm

Sự kiện phạm vi bên trong component trong ReactJS là các sự kiện được liên kết trực tiếp với các phần tử giao diện người dùng (UI elements) như nút bấm, trường nhập liệu, hộp kiểm, hoặc bất kỳ thành phần nào khác mà người dùng có thể tương tác. Các sự kiện này bao gồm những sự kiện phổ biến như **onClick, onChange, onSubmit, onMouseOver**, và nhiều loại sự kiện khác. Những sự kiện này được gọi là **event handlers** (trình xử lý sự kiện), và chúng giúp ứng dụng của bạn phản hồi lại những thao tác của người dùng một cách trực tiếp và tương tác.

Các sự kiện này thực sự được thực hiện khi người dùng **click**, **nhập thông tin**, hoặc thực hiện các hành động khác trên giao diện. Sự kiện sẽ kích hoạt một loạt các hành động cụ thể như cập nhật trạng thái, gửi dữ liệu, hiển thị thông báo, hoặc thay đổi giao diện. Chúng ta sẽ đi sâu hơn vào từng khía cạnh của sự kiện và cách chúng hoạt động bên trong một component React.

6.2. Sự kiện phạm vi toàn cục

a. Khái niệm

Event handling phạm vi toàn cục (global event handling) đề cập đến việc xử lý các sự kiện không chỉ giới hạn trong một component hoặc phần tử cụ thể, mà có thể xảy ra ở bất kỳ đâu trong ứng dụng hoặc trên trang web. Các sự kiện này thường được lắng nghe và xử lý ở cấp độ toàn cục, chẳng hạn như sự kiện xảy ra trên **cửa sổ trình duyệt**, **toàn bộ tài liệu HTML**, hoặc **toàn bộ trang web**, thay vì chỉ trong một thành phần UI cụ thể.

Global event handling cho phép ứng dụng của bạn phản hồi các thao tác của người dùng trên phạm vi lớn, bao gồm các sự kiện như thay đổi kích thước cửa sổ, cuộn trang, nhấn phím, hoặc thao tác kéo-thả. Điều này rất hữu ích khi bạn muốn thực hiện những thay đổi hoặc hành động mà không chỉ phụ thuộc vào một thành phần cụ thể.

Lưu ý: khi dùng xong component đó người dùng phải cleanup để tránh bị rò rỉ bộ nhớ cho lần mount component tiếp theo

5. Async/Await và Fetch API trong React.js

Fetch API là một giao diện cung cấp phương thức để thực hiện các yêu cầu HTTP (HyperText Transfer Protocol) trong môi trường JavaScript hiện đại, chủ yếu được sử dụng để lấy tài nguyên qua mạng. Nó được giới thiệu để thay thế XMLHttpRequest, một phương pháp cũ kỹ và cồng kềnh trong việc thực hiện các yêu cầu HTTP. Fetch API là một phần của tiêu chuẩn ECMAScript và đã trở thành một phương thức chuẩn để thực hiện các tác vụ lấy dữ liệu bất đồng bộ, chẳng hạn như gọi API từ các máy chủ.

5.1. Khái niệm cơ bản về Fetch API

Fetch là một phương pháp toàn cục (global method) có sẵn trong hầu hết các trình duyệt hiện đại. Nó trả về một đối tượng **Promise**, đại diện cho yêu cầu HTTP đang được thực hiện. Sau khi yêu cầu hoàn thành, promise này sẽ được **resolve** hoặc **reject** tùy thuộc vào kết quả của yêu cầu.

```
fetch(url, options)
.then(response => {
// Xử lý phản hồi
})
.catch(error => {
 // Xử lý lỗi
});
```

Trong đó:

- **url** là đường dẫn đến tài nguyên mà bạn muốn lấy.
- **options** là một đối tượng tùy chọn (optional object), cho phép bạn cấu hình các tham số cho yêu cầu như phương thức HTTP (GET, POST), headers, body, v.v.

5.2. Ưu điểm của Fetch API

Fetch API có nhiều cải tiến so với các kỹ thuật cũ như XMLHttpRequest, giúp nó trở thành lựa chọn ưu tiên cho các lập trình viên hiện đại:

- **Cú pháp đơn giản và dễ hiểu**: Một trong những lợi thế lớn nhất của Fetch API là cú pháp rõ ràng và dễ sử dụng hơn nhiều so với XMLHttpRequest. Việc gọi một API hay thực hiện một yêu cầu HTTP với Fetch trở nên ngắn gọn hơn, giảm thiểu sự phức tạp trong mã nguồn.
  Ví dụ, với XMLHttpRequest, bạn phải xử lý nhiều trạng thái khác nhau, còn Fetch API đơn giản hóa điều này bằng cách sử dụng **Promise**, giúp mã dễ đọc hơn.
- **Hoạt động bất đồng bộ theo kiểu hiện đại**: Fetch API hoạt động trên nguyên tắc bất đồng bộ, nghĩa là khi bạn gửi một yêu cầu, nó sẽ tiếp tục thực hiện các tác vụ khác mà không phải chờ cho đến khi nhận được phản hồi. Điều này cải thiện hiệu suất ứng dụng và trải nghiệm người dùng, vì ứng dụng không bị "đơ" trong khi chờ kết quả.
- **Hỗ trợ đầy đủ các chuẩn HTTP hiện đại**: Fetch API hỗ trợ tất cả các phương thức HTTP như GET, POST, PUT, DELETE, PATCH, giúp bạn dễ dàng thực hiện các tác vụ CRUD (Create, Read, Update, Delete) trên máy chủ. Ngoài ra, nó còn cung cấp khả năng quản lý header, body của yêu cầu, hỗ trợ JSON, form data và nhiều định dạng dữ liệu khác.
- **Promise-based**: Fetch API hoạt động dựa trên cơ chế Promise, giúp xử lý các tác vụ bất đồng bộ một cách rõ ràng và mạch lạc. Điều này thay thế cho cách tiếp cận dựa trên callback của XMLHttpRequest, vốn thường gây ra hiện tượng "callback hell" và làm cho mã khó bảo trì.
- **Cung cấp các phương pháp cho yêu cầu và phản hồi**: Fetch API không chỉ giúp gửi yêu cầu mà còn cung cấp các phương pháp để xử lý phản hồi. Ví dụ,

bạn có thể dễ dàng lấy dữ liệu từ phản hồi dưới dạng JSON bằng phương thức response.json(). Ngoài ra, bạn có thể lấy phản hồi dưới dạng văn bản (response.text()), Blob (response.blob()), hoặc đối tượng dạng nhị phân (response.arrayBuffer()), tùy thuộc vào loại dữ liệu mà API trả về.

- **Hỗ trợ CORS (Cross-Origin Resource Sharing)**: Fetch API hỗ trợ đầy đủ việc xử lý các yêu cầu HTTP qua các miền khác nhau (cross-origin). Điều này rất quan trọng khi bạn cần gọi API từ một miền khác với miền mà trang web của bạn đang chạy, và cần tuân thủ các quy tắc bảo mật liên quan đến chính sách CORS.

5.3. Cách thức hoạt động của Fetch API

Khi bạn gọi phương thức fetch, trình duyệt sẽ tạo một yêu cầu HTTP đến máy chủ mục tiêu và trả về một đối tượng Promise. Promise này đại diện cho tiến trình của yêu cầu. Trong Promise này, bạn có thể sử dụng phương thức then() để xử lý phản hồi khi yêu cầu thành công, hoặc catch() để xử lý lỗi khi yêu cầu không thành công (ví dụ, do lỗi mạng hoặc server trả về mã lỗi HTTP).

6. Error Handling, Debugging, và Optimization trong React.js

6.1. Xử Lý Lỗi (Error Handling)

Xử lý lỗi là một phần quan trọng trong phát triển ứng dụng React để đảm bảo rằng người dùng có thể trải nghiệm ứng dụng một cách liền mạch và không gặp phải các sự cố không mong muốn. Các lỗi có thể xuất hiện từ nhiều nguồn khác nhau như:

- **Lỗi mạng:** Khi ứng dụng cố gắng truy cập API nhưng không thể kết nối do sự cố mạng.
- **Lỗi dữ liệu:** Khi dữ liệu mà ứng dụng nhận được không đúng định dạng hoặc không thể xử lý.
- **Lỗi logic:** Các lỗi trong mã nguồn, chẳng hạn như truy cập vào một thuộc tính không tồn tại.

Để xử lý các lỗi này, bạn có thể sử dụng các phương pháp như khối **try-catch**, và các **Error Boundaries**. Error Boundaries cho phép bạn bắt lỗi ở cấp độ component và hiển thị giao diện thay thế thay vì làm sập toàn bộ ứng dụng. Việc này giúp cải thiện trải nghiệm người dùng, vì họ vẫn có thể sử dụng phần còn lại của ứng dụng mà không gặp phải sự cố nghiêm trọng.

6.2. Gỡ Lỗi (Debugging)

Gỡ lỗi là một quá trình thiết yếu trong phát triển phần mềm, đặc biệt là trong các ứng dụng React, nơi mà các component có thể được cấu trúc phức tạp. Có nhiều công cụ và phương pháp gỡ lỗi mà bạn có thể sử dụng:

- **Công cụ Phát triển React:** Sử dụng tiện ích mở rộng React Developer Tools giúp bạn dễ dàng kiểm tra các component, props, state và theo dõi cách chúng thay đổi theo thời gian.
- **Ghi log Console:** Ghi lại thông tin trong console bằng console.log là một cách đơn giản để theo dõi giá trị của các biến và xác định xem luồng thực thi có đúng như mong đợi hay không.

- **Gỡ lỗi trong IDE:** Nhiều IDE hiện nay hỗ trợ gỡ lỗi với các tính năng như breakpoint, cho phép bạn dừng ứng dụng tại một điểm cụ thể và kiểm tra giá trị của các biến và state.

Việc gỡ lỗi một cách hiệu quả không chỉ giúp bạn sửa lỗi mà còn cải thiện chất lượng mã và đảm bảo rằng ứng dụng hoạt động như mong đợi.

6.3. Tối Ưu Hóa Hiệu Suất (Performance Optimization)

Tối ưu hóa hiệu suất là rất quan trọng để đảm bảo rằng ứng dụng React của bạn hoạt động mượt mà và nhanh chóng. Một số kỹ thuật tối ưu hóa bao gồm:

- **Ghi nhớ (Memoization):** Sử dụng các kỹ thuật ghi nhớ giúp lưu trữ kết quả của các phép toán tốn kém để tránh tính toán lại khi các đầu vào không thay đổi.
- **Phân tách mã (Code Splitting):** Chia mã thành các phần nhỏ hơn giúp tải nhanh hơn và giảm thời gian tải ban đầu của ứng dụng.

Tải lười (Lazy Loading): Chỉ tải các component khi người dùng cần, thay vì tải tất cả ngay từ đầu, giúp giảm kích thước tải trang.

C. Kết Luận

I. Tổng Kết Về ReactJS

ReactJS đã chứng tỏ được sức mạnh và tính linh hoạt của nó trong việc phát triển ứng dụng web hiện đại. Là một thư viện mã nguồn mở được phát triển bởi Facebook, ReactJS giúp các lập trình viên xây dựng giao diện người dùng một cách hiệu quả thông qua việc sử dụng các component có thể tái sử dụng. Điều này không chỉ giảm thiểu lượng mã cần viết mà còn tăng cường khả năng bảo trì và mở rộng ứng dụng.

Một trong những điểm mạnh nổi bật của ReactJS là khả năng quản lý trạng thái (state management) một cách hiệu quả. Với việc sử dụng các hook như useState, useEffect, và nhiều thư viện bên ngoài như Redux, lập trình viên có thể dễ dàng quản lý và theo dõi các thay đổi trong trạng thái ứng dụng. Điều này góp phần tạo ra trải nghiệm người dùng mượt mà và thân thiện hơn, ngay cả trong những ứng dụng phức tạp với nhiều tương tác.

Bên cạnh đó, hệ sinh thái phong phú xung quanh ReactJS với nhiều công cụ và thư viện hỗ trợ, như React Router cho điều hướng, Material-UI cho thiết kế giao diện, và Axios cho tương tác với API, giúp các lập trình viên có thể dễ dàng tích hợp và mở rộng chức năng cho ứng dụng của họ. Tất cả những yếu tố này làm cho ReactJS trở thành lựa chọn tối ưu cho các dự án web hiện đại, từ các ứng dụng nhỏ cho đến các hệ thống phức tạp.

II. Tương Lai Của ReactJS

Nhìn về phía trước, tương lai của ReactJS hứa hẹn sẽ rất sáng sủa. Với sự phát triển không ngừng của cộng đồng và các cải tiến liên tục từ đội ngũ phát triển tại Facebook, ReactJS dự kiến sẽ tiếp tục giữ vững vị thế của mình trong lĩnh vực phát triển ứng dụng web. Các tính năng mới đang được giới thiệu, như **Concurrent Mode** và **Server Components**, không chỉ nâng cao hiệu suất mà còn cải thiện trải nghiệm phát triển cho các lập trình viên.

Concurrent Mode cho phép React xử lý nhiều tác vụ đồng thời, giúp cải thiện hiệu suất và giảm thiểu thời gian tải trang. Điều này cực kỳ quan trọng trong bối cảnh người dùng ngày càng yêu cầu trải nghiệm mượt mà và nhanh chóng. **Server Components** cung cấp khả năng xử lý các component trên máy chủ, giảm thiểu lượng mã JavaScript cần tải xuống phía client, từ đó giúp tăng tốc độ tải trang.
Ngoài ra, với sự gia tăng trong việc sử dụng các công nghệ như Progressive Web Apps (PWA) và Static Site Generation (SSG), ReactJS cũng có khả năng thích ứng và phát triển để đáp ứng những yêu cầu mới trong phát triển ứng dụng. Sự linh hoạt và khả năng mở rộng của ReactJS sẽ tiếp tục thu hút các nhà phát triển và doanh nghiệp, từ đó khẳng định vị trí của nó trong tương lai phát triển web.