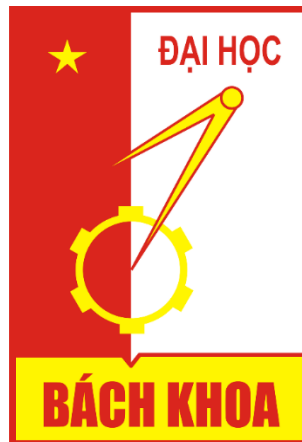


TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
Viện công nghệ thông tin và truyền thông



BÁO CÁO PROJECT 3

**Đề tài : Tìm hiểu NodesJS và ReactJS – Áp dụng xây dựng web đánh giá
kết quả chuẩn hóa văn bản (Text Normalization)**

Giảng viên hướng dẫn: TS. Nguyễn Thị Thu Trang

Trợ giảng hướng dẫn: Phạm Quang Minh

Sinh viên thực hiện: Nguyễn Duy Hoài Lâm

MSSV: 20173225

Hà Nội, Tháng 01-2021

Mục lục

Chương 1. Reactjs.....	4
1. Định nghĩa.....	4
2. Lợi ích khi sử dụng Reactjs.....	4
3. Virtual DOM	5
4. Components trong React	5
4.1: Khái niệm Component.....	5
4.2: Danh sách các hàm/phương thức của Component	5
4.3: Vòng đời của 1 Component.....	6
5. State trong React	7
5.1. Khởi tạo một state.....	7
5.2. Cập nhật một state	8
6. Props trong React	10
6.1. Truyền props trong các components.....	10
6.2. Nhận props trong components.....	11
6.3. Ví dụ thực tế	11
7. Xử lý các sự kiện trong React	13
8. Truyền tham số vào Hàm Bất Sự kiện	16
9. Các hook trong React	16
9.1. Giới thiệu về React Hooks.....	16
9.2. Basic Hooks.....	16
9.3. Additional Hooks.....	19
10. Redux	22
10.1. Redux là gì?.....	22
10.2. Tại sao lại nên sử dụng Redux.....	22

Project 3

10.3. Nguyên lý của Redux	24
10.4. Cách sử dụng Redux cơ bản	25
11. Redux Hook.....	28
11.1. Sử dụng Hooks trong React Redux().....	28
11.2. useSelector()	28
11.3. Cơ chế So sánh và cập nhật	29
11.4. Sử dụng memoizing selectors.....	30
11.5. useDispatch()	33
11.6. useStore().....	34
Chương 2. NodeJS	35
1. Định nghĩa.....	35
2. Ứng dụng của NodeJS.....	35
3. Nhược điểm NodeJS	35
4. Ưu điểm NodeJS	35
5. Với những ưu - nhược trên, khi nào chúng ta sẽ dùng Node.js?.....	36
6. Ví dụ Hello World trong Node.js	37
Chương 3. Trang web ứng dụng (Text Normalization)	38
1. Client	40
2. Server	55
2.1. Thực hiện api Expand.....	57
2.2. Thực hiện chức năng SendMail.....	58
Chương 4. Phụ lục	59
1. Hướng dẫn cài đặt	59
2. Tài liệu tham khảo.....	60

Chương 1. Reactjs

1. Định nghĩa

- **React** là thư viện **JavaScript** phổ biến nhất để xây dựng giao diện người dùng (UI). Nó cho tốc độ phản hồi tuyệt vời khi user nhập liệu bằng cách sử dụng phương pháp mới để render trang web.
- React là một thư viện **GUI** nguồn mở JavaScript tập trung vào một điều cụ thể; hoàn thành nhiệm vụ UI hiệu quả. Nó được phân loại thành kiểu “V” trong mô hình **MVC** (Model-View-Controller).
- React là một thư viện UI phát triển tại Facebook để hỗ trợ việc xây dựng những thành phần (components) UI có tính tương tác cao, có trạng thái và có thể sử dụng lại được. React được sử dụng tại Facebook trong production, và www.instagram.com được viết hoàn toàn trên React.
- Một trong những điểm hấp dẫn của React là thư viện này không chỉ hoạt động trên phía client, mà còn được render trên server và có thể kết nối với nhau. React so sánh sự thay đổi giữa các giá trị của lần render này với lần render trước và cập nhật ít thay đổi nhất trên DOM.

2. Lợi ích khi sử dụng Reactjs

- React cho phép sử dụng lại components đã được phát triển thành các ứng dụng khác có cùng chức năng. Tính năng tái sử dụng component là một lợi thế khác biệt cho các lập trình viên.
- React component dễ viết hơn vì nó sử dụng **JSX**, mở rộng cú pháp tùy chọn cho JavaScript cho phép kết hợp HTML với JavaScript.
- JSX là một sự pha trộn tuyệt vời của JavaScript và **HTML**. Nó làm rõ toàn bộ quá trình viết cấu trúc trang web. Ngoài ra, phần mở rộng cũng giúp render nhiều lựa chọn dễ dàng hơn.
- JSX có thể không là phần mở rộng cú pháp phổ biến nhất, nhưng nó được chứng minh là hiệu quả trong việc phát triển components đặc biệt hoặc các ứng dụng có khối lượng lớn.
- React cho phép tạo giao diện người dùng có thể được truy cập trên các công cụ tìm kiếm khác nhau. Tính năng này là một lợi thế rất lớn vì không phải tất cả các khung JavaScript đều thân thiện với **SEO**.
- Ngoài ra, vì React có thể tăng tốc quá trình của ứng dụng nên có thể cải thiện kết quả SEO. Cuối cùng **tốc độ web đóng một vai trò quan trọng trong tối ưu hóa SEO**.

3. Virtual DOM

4. Components trong React

4.1: Khái niệm Component

Trong React, để xây dựng trang web sử dụng những thành phần (component) nhỏ. Chúng ta có thể tái sử dụng một component ở nhiều nơi, với các trạng thái hoặc các thuộc tính khác nhau, trong một component lại có thể chứa thành phần khác. Mỗi component trong React có một trạng thái riêng, có thể thay đổi, và React sẽ thực hiện cập nhật component dựa trên những thay đổi của trạng thái.

Khái niệm component trong React là một trong những thành phần quan trọng nhất của React. Do vậy việc tìm hiểu rõ các method, cách thức hoạt động cũng như vai trò của các method đều rất cần thiết.

4.2: Danh sách các hàm/phương thức của Component

- `constructor(props)` :
 - Hàm này Thực hiện việc thiết lập state cho component.
 - Việc sử dụng `super(props)` là để có thể sử dụng `this.props` trong phạm vi hàm `constructor` này

```
constructor(props){  
  super(props)  
}
```

- `componentWillMount()` :
 - Thực hiện một số tác vụ, hàm này được gọi một lần trên cả client và server trước khi render diễn ra.
- `componentDidMount()` :
 - Thực hiện một số tác vụ, hàm này được gọi một lần chỉ trên client, sau khi render.
 - Hàm này rất hữu dụng khi làm việc thêm với Map, bởi vì map chỉ render được khi có node (id) trong DOM
- `componentWillUnmount()` :
 - Thực hiện một lần duy nhất, khi component sẽ unmount, được gọi trước khi tách component.
 - Hàm này hữu dụng khi cần xoá các timer không còn sử dụng

Project 3

- `componentWillReceiveProps(nextProps) :`
 - Hàm này thực hiện liên tục mỗi khi props thay đổi
- `shouldComponentUpdate(nextProps, nextState) :`
 - Thực hiện khi state và props thay đổi
 - Hàm này sẽ trả về kết quả true/false, sẽ cần sử dụng đến hàm này để xử lý xem có cần update component không.
- `componentWillUpdate(nextProps, nextState) :`
 - Hàm này thực hiện dựa vào kết quả của hàm trên (`shouldComponentUpdate`)
 - Nếu hàm trên trả về false, thì React sẽ không gọi hàm này
- `componentDidUpdate(prevProps, prevState) :`
 - Hàm này thực hiện sau khi component được render lại, từ kết quả của `componentWillUpdate`
- `render() :`

```
render() {  
  return (  
    <div>  
      // thực hiện việc render  
    </div>  
  );  
}
```

- Khai báo kiểu biến cho props

```
User.propTypes = {};
```

- Khai báo giá trị mặc định cho props

```
User.defaultProps = {}
```

4.3: Vòng đời của 1 Component

- Khởi tạo Component
 - Khởi tạo Class (đã thừa kế từ class Component của React)
 - Khởi tạo giá trị mặc định cho Props (`defaultProps`)
 - Khởi tạo giá trị mặc định cho State (trong hàm constructor)
 - Gọi hàm `componentWillMount()`
 - Gọi hàm `render()`
 - Gọi hàm `componentDidMount()`

Project 3

- Khi State thay đổi
 - Cập nhật giá trị cho state
 - Gọi hàm `shouldComponentUpdate()`
 - Gọi hàm `componentWillUpdate()` – với điều kiện hàm trên return true
 - Gọi hàm `render()`
 - Gọi hàm `componentDidUpdate()`
- Khi Props thay đổi
 - Cập nhật giá trị cho props
 - Gọi hàm `componentWillReceiveProps()`
 - Gọi hàm `shouldComponentUpdate()`
 - Gọi hàm `componentWillUpdate()` – với điều kiện hàm trên return true
 - Gọi hàm `render()`
 - Gọi hàm `componentDidUpdate()`
- Khi Unmount component:
Gọi hàm `componentWillUnmount()`

5. State trong React

State là một object có thể được sử dụng để chứa dữ liệu hoặc thông tin về components. State có thể được thay đổi bất cứ khi nào mong muốn. Khác với props có thể truyền props sang các components khác nhau thì state chỉ tồn tại trong phạm vi của components chứa nó, mỗi khi state thay đổi thì components đó sẽ được render lại. Trong các dự án React, state được dùng để phản hồi các yêu cầu từ người dùng, hay lưu trữ một dữ liệu nào đó trong components.

Thao tác với state trong ReactJS

Chúng ta có thể thao tác với **state** trong một component rất dễ dàng bằng cách sử dụng class components. Bên dưới mình sẽ chỉ ra các thao tác với state trong một component.

5.1. Khởi tạo một state

Chúng ta có thể khởi tạo một state bằng cách gán giá trị cho biến `this.state`:

```
this.state = { name : 'LÂM' }
```

lấy giá trị của state bằng cách gọi `this.state`:

```
console.log(this.state.name) // LÂM
```

Ví dụ bên dưới có một class components và sẽ tiến hành khởi tạo state bên trong phương thức constructor :

Project 3

```
import React from "react";

class App extends React.Component {
  constructor(props) {
    super(props);
    //Chỉ định một state
    this.state = { name: "Nguyễn Duy Hoài Lâm " };
  }
  render() {
    return (
      <div>
        <h1>Học ReactJS căn bản tại {this.state.name} </h1>
      </div>
    );
  }
}
export default App;
```

Trong hầu hết các trường hợp nên khởi tạo state bên trong hàm constructor() để tránh gặp các lỗi không mong muốn. Vì đây sẽ là hàm khởi chạy đầu tiên khi một components được gọi.

5.2. Cập nhật một state

Trong các components cần phải thao tác với state rất nhiều , ngoài thêm và lấy giá trị của state còn phải cập nhật các states để ReactJS có thể tự động re-render lại components. Điều này khá quan trọng, giả sử đang cho người dùng nhập vào một form nào đó và khi click *Lưu* thì nội dung được điền trong form lúc này sẽ phải hiển thị ra màn hình. Đây là lúc cần dùng đến *cách thay đổi giá trị của một state*.

Để **cập nhật một state** sử dụng phương thức:

```
this.setState({
  name: 'newValue'
})
```

Ngoài ra cũng có thể lấy giá trị của state trước khi cập nhật:

```
this.setState((state) => {
  return newValue;
});
```

Bên dưới là một ví dụ về cập nhật state index khi nhấn vào click vào button tương ứng :

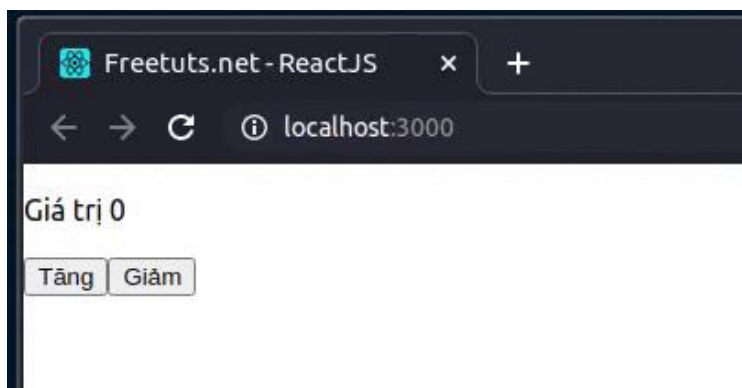
Project 3

```
import React from "react";

class App extends React.Component {
  constructor(props) {
    super(props);
    //Chỉ định một state
    this.state = { index: 1 };
  }
  render() {
    return (
      <div>
        <p>Giá trị {this.state.index}</p>
        <button onClick={() => {
          this.setState({
            index: this.state.index + 1
          })
        }}>Tăng</button>
        <button
          onClick={() => {
            this.setState({
              index: this.state.index - 1
            })
          }}
        >Giảm
      </div>
    );
  }
}

export default App;
```

Chúng ta sẽ thấy giá trị của state sẽ được thay đổi mỗi khi click vào button Tăng hoặc Giảm :



Project 3

State là một khái niệm khá đơn giản nhưng cũng hết sức quan trọng trong component, mặc dù props và state hay đi đôi với nhau nhưng nó hoàn toàn khác biệt với nhau.

6. Props trong React

Props là một object được truyền vào trong một components, mỗi components sẽ nhận vào props và trả về react element.

Props cho phép chúng ta giao tiếp giữa các components với nhau bằng cách truyền tham số qua lại giữa các components.

Khi một components cha truyền cho component con một props thì components con chỉ có thể đọc và không có quyền chỉnh sửa nó bên phía components cha.

Cách truyền một props cũng giống như cách thêm một attributes cho một element HTML. Ví dụ:

```
const App = () => <Welcome name="Lâm"></Welcome>
```

Trong ví dụ bên trên, component có tên Welcome sẽ nhận được giá trị của props có tên name vừa mới được truyền vào.

6.1. Truyền props trong các components

Có thể truyền dữ liệu từ một component với nhau bằng cách truyền như một attributes trong HTML element như sau:

```
const App = () => <Welcome tenProps1="giatri1" tenProps2="giatri2">Giá trị của props.children</Welcome>
```

Giả sử muốn truyền cho components có tên Welcome các giá trị như:

```
const App = () => <Welcome name="Nguyễn Trí" age="18" gender = "1">Xin chào Nguyễn Duy Hoài Lâm</Welcome>
```

Vậy trong components Welcome giá trị của props sẽ là một object bao gồm các giá trị truyền vào :

```
{
  name: "Nguyễn Trí",
  age: 18,
  gender : 1,
  children: "Xin chào Nguyễn Duy Hoài Lâm"
}
```

Khi truyền một giá trị bên trong một tags thì nó sẽ là giá trị của thuộc tính children trong object props như bên.

Project 3

6.2. Nhận props trong components

Chúng ta có thể nhận giá trị của một **props** bằng cách nhận vào tham số trong functional components và `this.props` trong một class components. Ví dụ :

```
//Nhận giá trị của props trong class components bằng this.props
import React, { Component } from "react";
class Welcome extends Component {
  render() {
    console.log(this.props) //Giá trị của props
    return (
      <div>
        <h1>Xin chào {this.props.name} !</h1>
      </div>
    );
  }
}
export default Welcome;
//Nhận props trong functional components bằng cách
//chỉ định tham số trong function.
import React from "react";
const Welcome = (props) => {
  console.log(props) //Giá trị của props
  return (
    <div>
      <h1>Xin chào {props.name} !</h1>
    </div>
  );
};
export default Welcome;
```

6.3. Ví dụ thực tế

Giả sử muốn truyền các props có tên `name`, `type`, `color`, `size`,... vào trong components có tên `Clothes`. Thực hiện các bước lần lượt như sau :

File `Clothes.js`:

Project 3

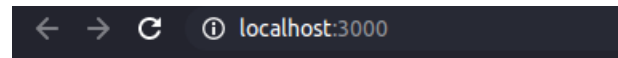
```
import React from "react";
const Clothes = (props) => {
  console.log(props) //Giá trị của props
  return (
    <div>
      <h1>{props.children}</h1>
      <ul>
        <li><b>Tên:</b> {props.name}</li>
        <li><b>Loại:</b> {props.type}</li>
        <li><b>Màu:</b> {props.color}</li>
        <li><b>Kích cỡ:</b> {props.size}</li>
      </ul>
      <hr></hr>
    </div>
  );
};
export default Clothes;
```

Component này sẽ hiển thị các props được truyền vào bao gồm: *name, type, age, size,....*

Tiếp theo ở file `App.js`, import component `Clothes` và truyền vào đó các props.

```
import React from "react";
import Clothes from "./Clothes"; //Import component vào
function App(props) {
  return (
    <div>
      <Clothes name="Quần jean" type="Skinny" color ="Đen" size = "L">Clothes 1</Clothes>
      <Clothes name="Váy" type="váy công chúa" color ="Trắng" size = "M">Clothes 2</Clothes>
    </div>
  );
}
export default App;
```

Bên trên truyền vào các props cần thiết, và gọi component `Clothes` 2 lần với các props khác nhau.



Clothes 1

- **Tên:** Quần jean
- **Loại:** Skinny
- **Màu:** Đen
- **Kích cỡ:** L

Clothes 2

- **Tên:** Váy
- **Loại:** váy công chúa
- **Màu:** Trắng
- **Kích cỡ:** M

7. Xử lý các sự kiện trong React

Việc bắt sự kiện của những element React rất giống với những element DOM. Có một số khác biệt về cú pháp như:

- Những sự kiện của React được đặt tên theo dạng camelCase, thay vì lowercase.
- Với JSX, có thể sử dụng “hàm” (function) để bắt sự kiện thay vì phải truyền vào một chuỗi.

Ví dụ, đoạn HTML sau:

```
<button onclick="activateLasers()">Activate Lasers</button>
```

sẽ có đôi chút khác biệt trong React:

```
<button onClick={activateLasers}> Activate Lasers</button>
```

Một điểm khác biệt nữa trong React là không thể trả về false để chặn những hành vi mặc định mà phải gọi preventDefault trực tiếp. Lấy ví dụ với đoạn HTML sau, để chặn hành vi mặc định của đường dẫn là mở trang mới, có thể viết:

```
<a href="#" onclick="console.log('The link was clicked.');" return false">Click me</a>
```

Project 3

Còn trong React, có thể làm như thế này:

```
function ActionLink() {  
  function handleClick(e) { e.preventDefault(); console.log('The link was clicked.')}  
  return (  
    <a href="#" onClick={handleClick}> Click me  
    </a>  
  );  
}
```

Ở đây, e là một sự kiện ảo (synthetic event). React định nghĩa những sự kiện ảo này dựa trên chuẩn W3C, nên không cần lo lắng về sự tương thích giữa những browser. React events không hoạt động chính xác giống như những event nguyên bản (native event). Hãy tham khảo tài liệu về SyntheticEvent để tìm hiểu thêm.

Khi làm việc với React, thường không cần phải gọi addEventListener để gắn listener cho element DOM sau khi nó được khởi tạo. Thay vào đó, chỉ cần cung cấp một listener ngay lần đầu element được render.

Khi định nghĩa component bằng class ES6, một mẫu thiết kế phổ biến là sử dụng phương thức của class để bắt sự kiện. Ví dụ, component Toggle dưới đây render một chiếc nút để người dùng thay đổi giữa state “ON” và “OFF”:

```
class Toggle extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {isToggleOn: true};  
  }  
  handleClick() { this.setState(state => ({isToggleOn: !state.isToggleOn})); }  
  render() {  
    return (  
      <button onClick={this.handleClick}> {this.state.isToggleOn ? 'ON' : 'OFF'}  
      </button>  
    );  
  }  
}  
  
ReactDOM.render(  
  <Toggle />,  
  document.getElementById('root')  
)
```

phải cẩn thận về ý nghĩa của this trong những callback JSX. Trong JavaScript, những phương thức của class mặc định không bị ràng buộc. Nếu quên ràng buộc this.handleClick và truyền nó vào onClick, this sẽ có giá trị là undefined khi phương thức này được thực thi.

Project 3

Đây không phải là tính chất của React mà là một phần trong cách những hàm JavaScript hoạt động. Thông thường, nếu trở tới phương thức mà không có () theo sau như `onClick={this.handleClick}`, nên ràng buộc phương thức đó.

Nếu thấy việc gọi bind phiền phức thì có hai giải pháp. Trong trường hợp đang sử dụng cú pháp thuộc tính class public thử nghiệm, có thể dùng những thuộc tính class để ràng buộc callback một cách chính xác:

```
class LoggingButton extends React.Component {
  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
    );
  }
}
```

Cú pháp này được bật theo mặc định trong Create React App.

Nếu không sử dụng cú pháp thuộc tính class, có thể dùng “hàm rút gọn” arrow function trong callback:

```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('this is:', this);
  }

  render() {
    // Cú pháp này đảm bảo `this` được ràng buộc trong handleClick
    <button/>
  }
}
```

Vấn đề với cú pháp này nằm ở chỗ một callback khác sẽ được khởi tạo mỗi khi `LoggingButton` render. Điều này ổn với đa số trường hợp. Tuy nhiên, nếu callback này được truyền dưới dạng prop xuống những component con, những component này sẽ bị render lại. Nói chung, chúng tôi khuyến khích việc ràng buộc ở constructor hoặc sử dụng cú pháp thuộc tính class để ngăn chặn vấn đề về hiệu suất này.

8. Truyền tham số vào Hàm Bắt Sự kiện

Bên trong một vòng lặp, người ta thường muốn truyền thêm một parameter cho một event handler. Ví dụ như, nếu id là ID của dòng (row), thì 2 dòng code bên dưới sẽ work:

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>  
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

Hai dòng code trên là tương đương, và lần lượt sử dụng hàm rút gọn và Function.prototype.bind.

9. Các hook trong React

9.1. Giới thiệu về React Hooks

Hooks chính thức được giới thiệu trong phiên bản React 16.8. Nó cho phép chúng ta sử dụng state và các tính năng khác của React mà không phải dùng đến Class

Điều này có nghĩa là từ phiên bản 16.8 trở đi, chúng ta đã có thể sử dụng **state** trong **stateless (functional) component**, việc mà từ trước tới nay ta bắt buộc phải khai báo **Class**. Có thể thấy, các nhà phát triển React họ đang muốn hướng đến 1 tương lai **Functional Programming** thay vì sử dụng những **Class** mà chỉ nghe cái tên thôi là ta đã nghĩ ngay đến **OOP**. Cộng với việc không sử dụng Class kế thừa từ React Component nữa nên giờ đây kích thước bundle sẽ được giảm đáng kể bởi code sử dụng Hooks.

Để có thể thay thế được **Class** thì React Hooks cung cấp cho chúng ta một bộ các built-in Hooks, giúp chúng ta sử dụng được các thành phần tạo nên React, có 2 loại built-in đó là: **Basic**

Hooks và **Additional Hooks**

9.2. Basic Hooks

9.2.1. useState

Cái tên nói lên tất cả hàm này nhận đầu vào là giá trị khởi tạo của 1 state và trả ra 1 mảng gồm có 2 phần tử, phần tử đầu tiên là state hiện tại, phần tử thứ 2 là 1 function dùng để update state (giống như hàm setState cũ vậy). Ví dụ:

Ngày trước dùng Class thì viết như sau

Project 3

```
constructor(props) {  
  super(props);  
  this.state = { isLoading: false }  
}  
  
onClick() {  
  this.setState({  
    isLoading: true,  
  })  
}
```

Còn bây giờ thì chỉ cần viết ngắn gọn:

```
const [isLoading, setLoading] = useState(false);  
  
onClick() {  
  setLoading(true)  
}
```

Khi muốn update state cho `isLoading` là `true` thì chỉ cần gọi đến hàm `setLoading(true)` là Ok, rất đơn giản và gọn nhẹ phải không nào. Nếu như đang làm việc với React-Redux để quản lý State thì mình khuyên chỉ nên sử dụng `useState` để quản lý các UI State (là những state có giá trị boolean nhằm mục đích render ra UI) để tránh việc conflict với cả Redux State và maintain sau này.

9.2.2. useEffect

Như đã giới thiệu trong phần mở đầu về sự phức tạp trong các hàm Lifecycle thì để thay thế nó chúng ta sẽ có hàm **useEffect**. Nó giúp chúng ta xử lý các side effects, `useEffect` sẽ tương đương với các hàm **componentDidMount**, **componentDidUpdate** và **componentWillUnmount** trong Lifecycle. Ví dụ:

Project 3

```
import { callApi } from './actions'

const App = ({ callApi, data }) => {
  useEffect(() => {
    callApi('some_payload_')
  }, [])
  return (
    <div>
      {data.map(item => {// do something
        })}
    </div>
  )
}

const mapDispatchToProps = dispatch => ({
  callApi: (keyword) => dispatch(callApi)
})

export default connect({}, mapDispatchToProps)(App)
```

Có thể thấy trong `useEffect` ta cũng có thể thực hiện công việc call API giống như hàm `ComponentDidMount` ngày trước. Để tránh việc hàm `useEffect` luôn chạy vào mỗi khi có thay đổi State thì ta có thể truyền vào tham số thứ 2 trong `useEffect` đó là 1 array, trong array này ta có thể truyền vào đó những giá trị mà `useEffect` sẽ subscribe nó, tức là chỉ khi nào những giá trị đó thay đổi thì hàm `useEffect` mới được thực thi. Hoặc cũng có thể truyền vào 1 array rỗng thì khi đó nó sẽ chỉ chạy 1 lần đầu tiên sau khi render giống với hàm **ComponentDidMount** Ví dụ:

```
useEffect(
  () => {
    const subscription = props.source.subscribe();
    return () => {
      subscription.unsubscribe();
    };
  },
  [props.source], // giá trị được subscribe
);
```

Project 3

Còn 1 vấn đề nữa đó là trong hàm `useEffect` ta có thể return về 1 function (chú ý là bắt buộc phải return về function) thì khi làm điều này nó sẽ tương đương với việc ta sử dụng hàm

LifeCycle **`componentWillUnmount`**

Tổng kết lại thì đây là những gì ta cần nhớ trong hàm `useEffect`:

```
useEffect(() => {  
  // almost same as componentDidMount  
  console.log('mounted!');  
  return () => {  
    // almost same as componentWillUnmount  
    console.log('unmount!');  
  };  
}, []);
```

9.3. Additional Hooks

9.3.1. `useReducer`

Thực tế khi sử dụng `useState` thì nó sẽ trả về 1 phiên bản đơn giản của `useReducer`, vậy nên chúng ta có thể coi `useReducer` như một phiên bản nâng cao hơn dùng để thay thế cho việc sử dụng `useState`. Nếu đã làm việc với `React-Redux` thì chắc hẳn sẽ dễ dàng nhận ra flow quen thuộc này phải không nào. Giống như reducer trong `Redux` thì `useReducer` cũng nhận vào một reducer dạng `(state, action)` và trả ra một `newState`. Khi sử dụng chúng ta sẽ nhận được một cặp bao gồm current state và dispatch function. Ví dụ:

Project 3

```
const initialState = { count: 0 }

function reducer(state, action) {
  const [count, setCount] = useState(0);
  switch (action.type) {
    case 'INCREMENT':
      return setCount(count + 1);
    case 'DECREMENT':
      return setCount(count - 1);
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState)
  return (
    <>
      <StyledLogo src={logo} count={count} />
      <Count count={count} />
      <div style={{ display: 'flex' }}>
        <button onClick={() => dispatch({ type: 'DECREMENT' })}> - </button>
        <button onClick={() => dispatch({ type: 'INCREMENT' })}> + </button>
      </div>
    </>
  )
}
```

9.3.2. useMemo

useMemo giúp ta kiểm soát việc được render dư thừa của các component con, nó khá giống với hàm **shouldComponentUpdate** trong Lifecycle. Bằng cách truyền vào 1 tham số thứ 2 thì chỉ khi tham số này thay đổi thì thằng useMemo mới được thực thi. Ví dụ:

- **Không** sử dụng useMemo:

```
const NotUsingMemo = ({ products }) => {
  const soldoutProducts = products.filter(x => x.isSoldout === true);
};
```

- **Có** sử dụng useMemo:

Project 3

```
const UsingMemo = ({ products }) => {
  const soldoutProducts = useMemo(
    () => products.filter(x => x.isSoldout === true),
    [products] // watch products
  );
};
```

9.3.3. useCallback

useCallback có nhiệm vụ tương tự như useMemo nhưng khác ở chỗ function truyền vào useMemo bắt buộc phải ở trong quá trình render trong khi đối với useCallback đó lại là function callback của 1 event nào đó như là onClick chẳng hạn. Ví dụ:

```
const App = () => {
  const [text, setText] = React.useState('');

  return (
    <>
      <input type="text" value={text} onChange={e => setText(e.target.value)} />
      <Wrap />
    </>
  );
};

const Wrap = () => {
  const [isChecked, setIsChecked] = React.useState(false);
  const toggleChecked = useCallback(() => setIsChecked(!isChecked), [
    isChecked
  ]);

  return <Checkbox value={isChecked} onClick={toggleChecked} />;
};

const Checkbox = React.memo(({ value, onClick }) => {
  console.log('Checkbox is renderd!');
  return (
    <div style={{ cursor: 'pointer' }} onClick={onClick}>
      {value ? '☑' : '☐'}
    </div>
  );
});
```

Trong ví dụ trên ta sử dụng useCallback cho sự kiện onClick, điều này có nghĩa là việc thay đổi giá trị text trong ô Input sẽ không làm component Checkbox bị re-render.

9.3.4. Kết luận hook

Ngoài những hook cơ bản hay được sử dụng mà mình đã giới thiệu ở trên thì vẫn còn 1 số hook khác như là `useContext`, `useRef`, `useLayoutEffect`, `useDebugValue`, `useImperativeHandle` các có thể vào trang chủ của react hooks để tìm hiểu thêm nhé.

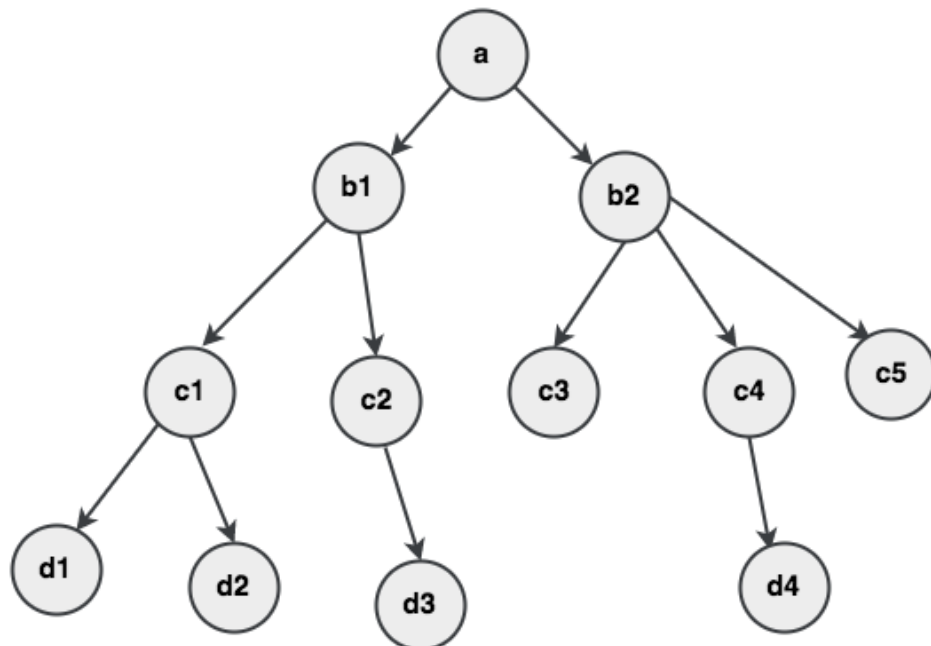
10.Redux

10.1. Redux là gì?

Redux js là một thư viện Javascript giúp tạo ra thành một lớp quản lý trạng thái của ứng dụng. Được dựa trên nền tảng tư tưởng của ngôn ngữ **Elm** kiến trúc **Flux** do Facebook giới thiệu, do vậy Redux thường là bộ đôi kết hợp hoàn hảo với React (`ReactJs` và `React Native`).

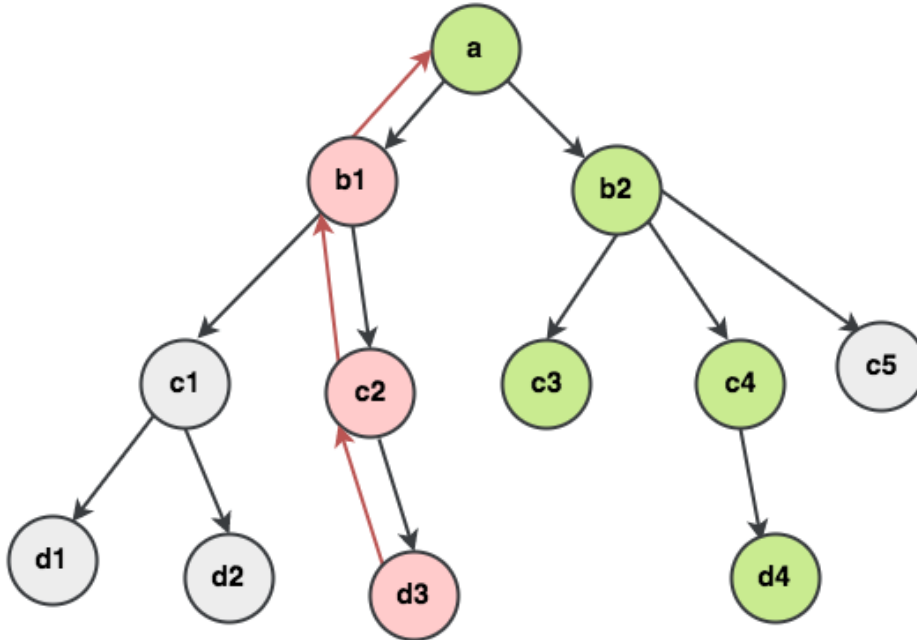
10.2. Tại sao lại nên sử dụng Redux

- Giả sử chúng ta có 1 ứng dụng các node như trong hình là tượng trưng cho một single page application



Project 3

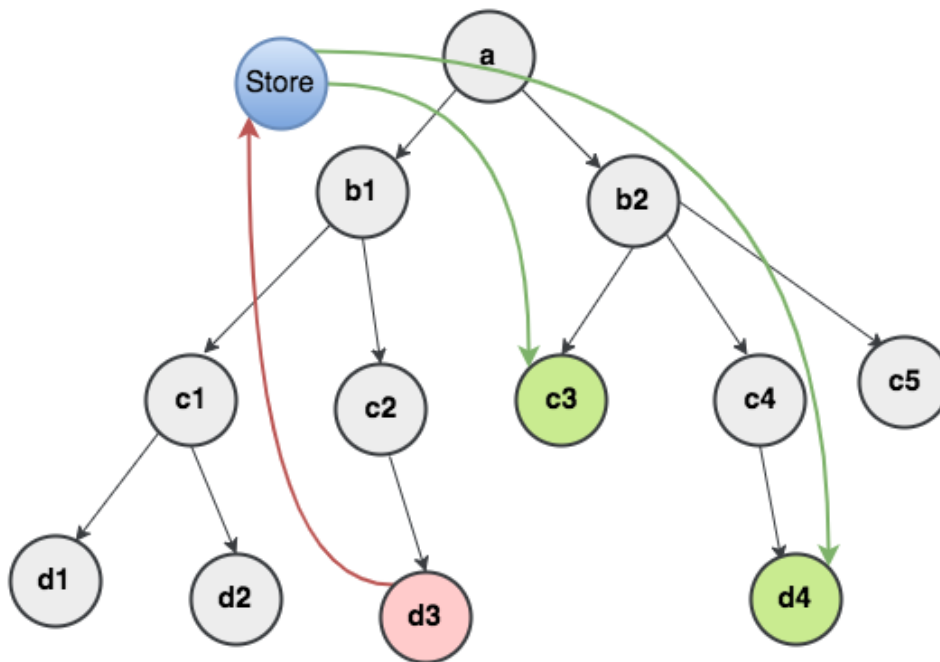
- Giả sử có một hành động nào đó được kích hoạt ở node d3 và ta muốn thay đổi trạng thái (state) ở d4 và c3 thì luồng dữ liệu sẽ được truyền từ node d3 trở về node a rồi từ node a mới truyền được đến các node d4 và c3



Cập nhật trạng thái (state) cho node d4: d3-c2-b1-a-b2-c4-d4

Cập nhật trạng thái (state) cho node c3: d3-c2-b1-a-b2-c3

Với những bài toán nhỏ thì chúng ta hoàn toàn có thể dùng `ReactJs` để cập nhật các trạng thái (state) một cách dễ dàng mà không cần dùng đến `Redux`. Nhưng nếu là một bài toán lớn thì sao lúc này nếu chỉ sử dụng `ReactJs` để cập nhật các trạng thái (state) thì thật sự là một khó khăn rất là lớn. Từ những nhược điểm trên thì `Redux` ra đời nhằm khắc phục nhược điểm đó.



Từ hình vẽ ta thấy để giải quyết bài toán trên ta chỉ cần `dispatch` một action từ node `d3` về store rồi `d4` và `c3` chỉ cần connect tới store cả cập nhật data thay đổi thế là bài toán được giải quyết một cách dễ dàng.

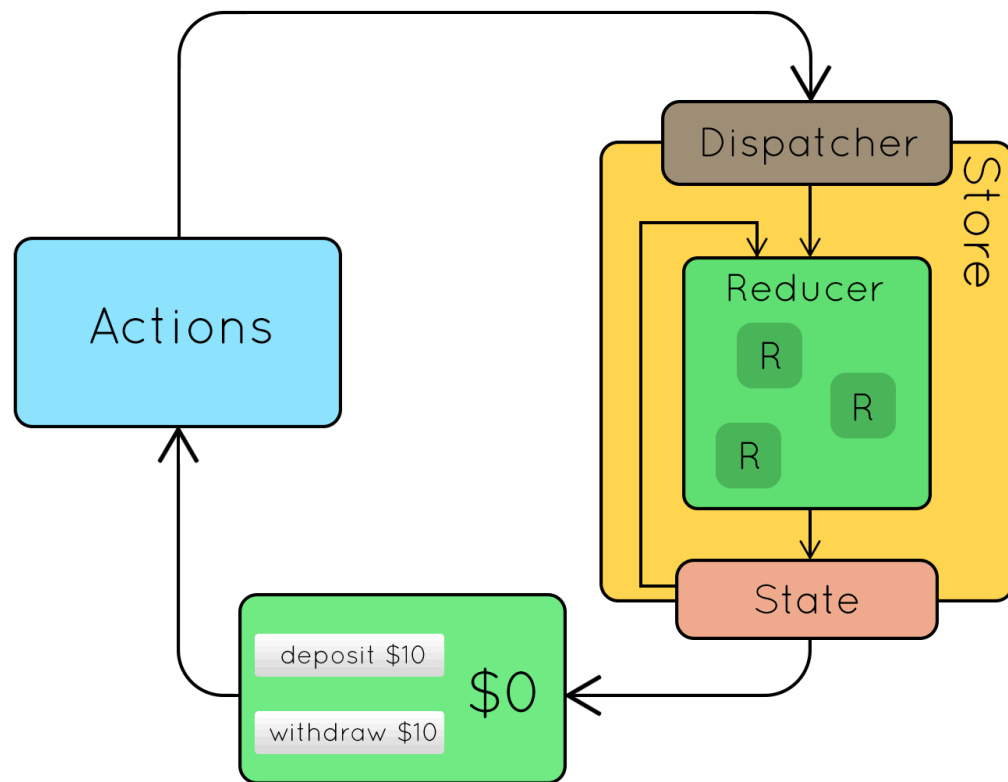
- Để cài đặt `Redux` cho ứng dụng của thì chạy lệnh sau:

```
npm install --save redux
```

10.3. Nguyên lý của Redux

`Redux` được xây dựng dựa trên 3 nguyên lý như sau:

- Trạng thái (`state`) của toàn bộ ứng dụng được lưu trong trong 1 store duy nhất là 1 Object mô hình tree
- Chỉ có 1 cách duy nhất để thay đổi trạng thái (`state`) đó là tạo ra một action (là 1 object mô tả những gì xảy ra)
- Để chỉ rõ trạng thái (`state`) tree được thay đổi bởi 1 action phải viết pure reducers



Hình ảnh minh họa nguyên lý hoạt động của Redux.

10.4. Cách sử dụng Redux cơ bản

Nếu muốn sử dụng Redux, chúng ta cần nhớ 4 thành phần chính của Redux : Store, Views , Actions, và Reducers. Chi tiết 4 thành phần được mô tả như sau:

Store: Là nơi quản lý trạng thái (state), có thể truy cập để lấy trạng thái (state) ra, update state hiện có, và listener để nhận biết xem có thay đổi gì không, và cập nhật qua views.

```
import { createStore } from 'redux';|
const store = createStore(myReducer);
console.log('Default:', store);
```

Khi mở trình duyệt và bật F12 lên thì ta nhận được kết quả như sau:

Project 3

```
Default: demo.js:39
▼ {dispatch: f, subscribe: f, getState: f, replaceReducer: f, Symbol(observable): f} ⓘ
  ▶ dispatch: f dispatch(action)
  ▶ getState: f getState()
  ▶ replaceReducer: f replaceReducer(nextReducer)
  ▶ subscribe: f subscribe(listener)
  ▶ Symbol(observable): f observable()
  ▶ __proto__: Object
```

trong đó, có 3 phương thức quan trọng cần chú ý đó là:

getState(): Giúp lấy ra state hiện tại

dispatch(action): Thực hiện gọi 1 action

subscribe(listener): Luôn lắng nghe xem có thay đổi gì không rồi ngay lập tức cập nhật ra View

Actions : nó là 1 pure object định nghĩa 2 thuộc tính lần lượt là type: kiểu của action, ví dụ như 'TOGGLE_STATUS', payload: giá trị tham số mà action creator truyền lên.

```
var action = {
  type : 'TOGGLE_STATUS',
  payload : "// tham số"
};
```

Reducers: Khác với actions có chức năng là mô tả những thứ gì đã xảy ra, nó không chỉ rõ state nào của response thay đổi, mà việc này là do reducers đảm nhiệm, nó là nơi xác định state sẽ thay đổi như thế nào, sau đó trả ra một state mới.

```
var myReducer = (state = initialState, action) => {
  if (action.type === 'TOGGLE_STATUS') {
    let newState = {...state}
    newState.status = !state.status;
    return newState; // mục đích của reducer là trả ra cái state mới
  }
  return state;
}
```

View: Hiển thị dữ liệu được cung cấp bởi Store

Để hiểu rõ hơn chúng ta cùng đi vào ví dụ sau:

Project 3

```
import { createStore } from 'redux';

var initialState = {
  status : false
}

var myReducer = (state = initialState, action) => {
  if (action.type === 'TOGGLE_STATUS') {
    let newState = {...state}
    newState.status = !state.status;
    return newState; // mục đích của reducer là trả ra cái state mới
  }

  return state;
}

const store = createStore(myReducer); // Khởi tạo store

console.log('Default:', store.getState());
var action = { type : 'TOGGLE_STATUS' };

store.dispatch(action); // lúc này action ở trên myReducer chính là action này

console.log('TOGGLE_STATUS', store.getState());
```

Chạy lên chúng ta nhận được kết quả như sau:

Default: ▼ {status: false} ⓘ	demo.js:41
status: false	
▶ __proto__: Object	
TOGGLE_STATUS ▼ {status: true} ⓘ	demo.js:47
status: true	
▶ __proto__: Object	

Phân tích ví dụ trên một chút ta có:

Bước 1: Khởi tạo store cho project và tham số nhận vào là một reducer myReducer

```
const store = createStore(myReducer);
```

Bước 2: Sau đó ta muốn thay đổi trạng thái của status thì ta gọi một action với type là "TOGGLE_STATUS" dùng hàm dispatch() của store:

```
store.dispatch(action);
```

Bước 3: Lúc này là nhiệm vụ của reducer, sẽ phân tích action được gửi lên là gì và sau đó xử lý và cuối cùng trả ra một state mới

Project 3

```
var myReducer = (state = initialState, action) => {
  if (action.type === 'TOGGLE_STATUS') {
    let newState = {...state}
    newState.status = !state.status;
    return newState; // mục đích của reducer là trả ra cái state mới
  }

  return state;
}
```

Bước 4: Hàm subscribe() trong store sẽ làm nhiệm vụ cập nhật tình hình thay đổi ra View.

Trên đây là nguyên lý hoạt động cơ bản của Redux.

action -> reducer -> store -> view

11.Redux Hook

11.1. Sử dụng Hooks trong React Redux()

Nếu trước đây sử dụng connect(), và phải wrap App component của bằng 1 component là <Provider> như ví dụ này:

```
const store = createStore(rootReducer)

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
)
```

với update mới, có thể import và sử dụng bất kỳ thứ gì trong React redux hook api vào trực tiếp function component.

11.2. useSelector()

```
const result : any = useSelector[(selector : Function, equalityFn? : Function)]
```

Project 3

Thay thế cho việc dùng `mapStateToProps` để lấy state của Redux store ra sử dụng trong component. selector sẽ được gọi dựa theo function truyền vào trong tham số đầu tiên của `useSelector`, với tham số đầu tiên của function chính là store state.

chú ý:

- Function là tham số của `useSelector` nên là 1 pure function.
-
- Nó sẽ chạy mỗi khi action được dispatch hoàn tất.
-
- Một số điểm khác biệt việc dùng `useSelector` và `mapState`:
 - - Selector có thể trả về bất kỳ giá trị nào không chỉ là 1 object. Kết quả trả về của selector sẽ được sử dụng giống như kết quả trả về của `useSelector()` hook.
 - - Khi action được dispatch, `useSelector()` sẽ so sánh (shallow compare) giá trị mới và cũ của selector, nếu chúng khác nhau thì component sẽ render lại. còn không thì việc render lại sẽ không diễn ra.
 - - hàm get selector không nhận `ownProps` làm tham số. tuy nhiên props có thể được sử dụng thông qua closure (xem tại đây).
 - - Phải cẩn trọng khi sử dụng memoizing selectors.
-

có thể gọi nhiều lần `useSelector()` trong 1 function component, mỗi lần gọi nó sẽ tạo 1 subscription tới redux store riêng.

11.3. Cơ chế So sánh và cập nhật

Khi function component render, selector function sẽ được gọi và kết quả của nó sẽ được trả về từ `useSelector()` hook (giá trị trả về sẽ được cache để trả về cho lần gọi tới nếu selector không bị thay đổi).

Tuy nhiên, nếu 1 action được dispatch trên redux store. `useSelector()` chỉ render lại khi selector được subscribe trả về mới phải khác với giá trị cũ của nó. tại bản v7.1.0-alpha.5, phép so sánh giữ 2 phiên bản cũ và mới của selector là `===`. đây chính là sự khác biệt so với `connect()` mà chúng ta hay sử dụng ngày trước. Chúng chỉ được so sánh `==` trong giá trị trả về của `mapState` để xác định component có cần render lại hay không. Việc dùng `useSelector` sẽ giúp quản lý redux state chặt chẽ hơn.

Project 3

Với `mapState`, tất cả các selector được trả về trong một object. tạo ra object mới mỗi lần cũng không là vấn đề, `connect()` sẽ so sánh từng selector trong nó.

Với việc dùng `useSelector`, đương nhiên rằng việc trả về 1 object mới thì nó sẽ re-render lại rồi. vậy đừng nên trả về trong `useSelector` một object. thay vào đó hãy thử:

- dùng nhiều `useSelector` cho riêng các selector muốn sử dụng.
- dùng `reselect` hoặc tự viết 1 lib ra để quản lý state dạng này.
- dùng `shallowEqual` như một function truyền vào tham số thứ 2 của `useSelector` => mục đích cuối cùng cũng như việc tương tự sử dụng `_.isEqual()` hoặc `Immutable.js` để so sánh

Ví dụ: cách dùng đơn giản

```
import React from 'react'
import { useSelector } from 'react-redux'

export const CounterComponent = () => {
  const counter = useSelector(state => state.counter)
  return <div>{counter}</div>
}
```

Dùng props để xác định cái gì được lấy ra:

```
import React from 'react'
import { useSelector } from 'react-redux'

export const TodoListItem = props => {
  const todo = useSelector(state => state.todos[props.id])
  return <div>{todo.text}</div>
}
```

11.4. Sử dụng memoizing selectors

Khi muốn sử dụng `useSelector()` chỉ trong component, xem ví dụ bên dưới. một instance mới của selector được tạo ngay sau khi component được render. Nó hoạt động nếu không giữ một state nào. Nó sẽ dùng để tạo ra một giá trị selector mới.

Project 3

Tuy nhiên việc tạo memoizing selector từ việc sử dụng createSelector từ reselect có state bên trong, và do đó phải cẩn thận khi sử dụng chúng. Dưới đây có thể tìm thấy các kịch bản sử dụng điển hình của memoizing selector.

Khi selector chỉ phụ thuộc vào state, chỉ cần đảm bảo rằng nó được khai báo bên ngoài component để sử dụng mỗi lần render.

```
import React from 'react'
import { useSelector } from 'react-redux'
import { createSelector } from 'reselect'

const selectNumOfDoneTodos = createSelector(
  state => state.todos,
  todos => todos.filter(todo => todo.isDone).length
)

export const DoneTodosCounter = () => {
  const NumOfDoneTodos = useSelector(selectNumOfDoneTodos)
  return <div>{NumOfDoneTodos}</div>
}

export const App = () => {
  return (
    <>
      <span>Number of done todos:</span>
      <DoneTodosCounter />
    </>
  )
}
```

Điều tương tự cũng đúng nếu selector phụ thuộc vào props của component, nhưng sẽ chỉ được sử dụng trong một trường hợp duy nhất component:

Project 3

```
import React from 'react'
import { useSelector } from 'react-redux'
import { createSelector } from 'reselect'

const selectNumOfTodosWithIsDoneValue = createSelector(
  state => state.todos,
  (_, isDone) => isDone,
  (todos, isDone) => todos.filter(todo => todo.isDone === isDone).length
)

export const TodoCounterForIsDoneValue = ({ isDone }) => {
  const NumOfTodosWithIsDoneValue = useSelector(state =>
    selectNumOfTodosWithIsDoneValue(state, isDone)
  )

  return <div>{NumOfTodosWithIsDoneValue}</div>
}

export const App = () => {
  return (
    <>
      <span>Number of done todos:</span>
      <TodoCounterForIsDoneValue isDone={true} />
    </>
  )
}
```

Tuy nhiên, khi selector được sử dụng trong nhiều instance của component và phụ thuộc vào props của component, cần đảm bảo rằng mỗi instance của component có phiên bản selector riêng của nó

```
import React, { useMemo } from 'react'
import { useSelector } from 'react-redux'
import { createSelector } from 'reselect'

const makeNumOfTodosWithIsDoneSelector = () =>
  createSelector(
    state => state.todos,
    (_, isDone) => isDone,
    (todos, isDone) => todos.filter(todo => todo.isDone === isDone).length
  )

export const TodoCounterForIsDoneValue = ({ isDone }) => {
  const selectNumOfTodosWithIsDone = useMemo(
    makeNumOfTodosWithIsDoneSelector,
    []
  )

  const numOfTodosWithIsDoneValue = useSelector(state =>
    selectNumOfTodosWithIsDoneValue(state, isDone)
  )

  return <div>{numOfTodosWithIsDoneValue}</div>
}

export const App = () => {
  return (
    <>
      <span>Number of done todos:</span>
      <TodoCounterForIsDoneValue isDone={true} />
      <span>Number of unfinished todos:</span>
      <TodoCounterForIsDoneValue isDone={false} />
    </>
  )
}
```


11.5. useDispatch()

```
const dispatch = useDispatch()
```

Hook này trả về một tham chiếu tới dispatch function trong redux store. dùng nó để dispatch một action. ví dụ:

```
import React from 'react'
import { useDispatch } from 'react-redux'

export const CounterComponent = ({ value }) => {
  const dispatch = useDispatch()

  return (
    <div>
      <span>{value}</span>
      <button onClick={() => dispatch({ type: 'increment-counter' })}>
        Increment counter
      </button>
    </div>
  )
}
```

nếu sử dụng 1 function có dùng dispatch trong đó mà truyền tới component con dưới dạng props. thì nên dùng useCallback để wrap lại nó. mục đích là để lưu trữ nó lại cho lần gọi sử dụng sau. ví nếu không các component con có thể bị render lại nếu dispatch thay đổi.

Project 3

```
import React, { useCallbakck } from 'react'
import { useDispatch } from 'react-redux'

export const CounterComponent = ({ value }) => {
  const dispatch = useDispatch()
  const incrementCounter = useCallbakck(
    () => dispatch({ type: 'increment-counter' }),
    [dispatch]
  )
  return (
    <div>
      <span>{value}</span>
      <MyIncrementButton onIncrement={incrementCounter} />
    </div>
  )
}

export const MyIncrementButton = React.memo(({ onIncrement }) => (
  <button onClick={onIncrement}>Increment counter</button>
))
```

11.6. useStore()

```
const store = useStore()
```

cái tên nói lên tất cả, giống với việc lấy Redux store và truyền store vào `<Provider>`. không nên dùng nó, hãy dùng `useSelector` vì nó đã giải quyết được vấn đề lấy redux state của rồi.

Chương 2. NodeJS

1. Định nghĩa

- Node.js là một mã nguồn mở, một môi trường cho các máy chủ và ứng dụng mạng.
- Node.js sử dụng Google V8 JavaScript engine để thực thi mã, và một tỷ lệ lớn các mô-đun cơ bản được viết bằng JavaScript. Các ứng dụng node.js thì được viết bằng JavaScript.
- Node.js chứa một thư viện built-in cho phép các ứng dụng hoạt động như một Webserver mà không cần phần mềm như Nginx, Apache HTTP Server hoặc IIS.
- Node.js cung cấp kiến trúc hướng sự kiện (event-driven) và non-blocking I/O API, tối ưu hóa thông lượng của ứng dụng và có khả năng mở rộng cao
- Mọi hàm trong Node.js là không đồng bộ (asynchronous). Do đó, các tác vụ đều được xử lý và thực thi ở chế độ nền (background processing)

2. Ứng dụng của NodeJS

- Xây dựng websocket server (Chat server)
- Hệ thống Notification (Giống như facebook hay Twitter)
- Ứng dụng upload file trên client
- Các máy chủ quảng cáo
- Các ứng dụng dữ liệu thời gian thực khác.

3. Nhược điểm NodeJS

- Ứng dụng nặng tốn tài nguyên Nếu cần xử lý các ứng dụng tốn tài nguyên CPU như encoding video, convert file, decoding encryption... hoặc các ứng dụng tương tự như vậy thì không nên dùng NodeJS (Lý do: NodeJS được viết bằng C++ & Javascript, nên phải thông qua thêm 1 trình biên dịch của NodeJS sẽ lâu hơn 1 chút). Trường hợp này hãy viết 1 Addon C++ để tích hợp với NodeJS để tăng hiệu suất tối đa !
- NodeJS và ngôn ngữ khác NodeJS, PHP, Ruby, Python .NET ... thì việc cuối cùng là phát triển các App Web. NodeJS mới sơ khai như các ngôn ngữ lập trình khác. Vậy nên đừng hi vọng NodeJS sẽ không hơn PHP, Ruby, Python... ở thời điểm này. Nhưng với NodeJS có thể có 1 ứng dụng như mong đợi, điều đó là chắc chắn !

4. Ưu điểm NodeJS

Đặc điểm nổi bật của Node.js là nó nhận và xử lý nhiều kết nối chỉ với một single-thread. Điều này giúp hệ thống tốn ít RAM nhất và chạy nhanh nhất khi không phải tạo thread mới cho mỗi

Project 3

truy vấn giống PHP. Ngoài ra, tận dụng ưu điểm non-blocking I/O của Javascript mà Node.js tận dụng tối đa tài nguyên của server mà không tạo ra độ trễ như PHP

- JSON APIs Với cơ chế event-driven, non-blocking I/O(Input/Output) và mô hình kết hợp với Javascript là sự lựa chọn tuyệt vời cho các dịch vụ Webs làm bằng JSON.
- Ứng dụng trên 1 trang(Single page Application) Nếu định viết 1 ứng dụng thể hiện trên 1 trang (Gmail?) NodeJS rất phù hợp để làm. Với khả năng xử lý nhiều Request/s đồng thời thời gian phản hồi nhanh. Các ứng dụng định viết không muốn nó tải lại trang, gồm rất nhiều request từ người dùng cần sự hoạt động nhanh để thể hiện sự chuyên nghiệp thì NodeJS sẽ là sự lựa chọn của .
- Shelling tools unix NodeJS sẽ tận dụng tối đa Unix để hoạt động. Tức là NodeJS có thể xử lý hàng nghìn Process và trả ra 1 luồng khiến cho hiệu suất hoạt động đạt mức tối đa nhất và tuyệt vời nhất.
- Streaming Data (Luồng dữ liệu) Các web thông thường gửi HTTP request và nhận phản hồi lại (Luồng dữ liệu). Giả sử sẽ cần xử lý 1 luồng dữ liệu cực lớn, NodeJS sẽ xây dựng các Proxy phân vùng các luồng dữ liệu để đảm bảo tối đa hoạt động cho các luồng dữ liệu khác.
- Ứng dụng Web thời gian thực Với sự ra đời của các ứng dụng di động & HTML 5 nên Node.js rất hiệu quả khi xây dựng những ứng dụng thời gian thực (real-time applications) như ứng dụng chat, các dịch vụ mạng xã hội như Facebook, Twitter,...

5. Với những ưu - nhược trên, khi nào chúng ta sẽ dùng Node.js?

Node.js rất hấp dẫn. Nhưng khi quyết định bắt tay xây dựng một dự án bằng Node.js, hãy đặt câu hỏi: “Tôi có nên dùng Node.js hay không?”. Và dưới đây là một trong số những câu trả lời cho điều đó.

KHÔNG nên sử dụng Node.js khi:

- Xây dựng các ứng dụng hao tốn tài nguyên: đừng mơ mộng đến Node.js khi đang muốn viết một chương trình convert video. Node.js hay bị rơi vào trường hợp thất cổ chai khi làm việc với những file dung lượng lớn.
- Một ứng dụng chỉ toàn CRUD: Node.js không nhanh hơn PHP khi làm các tác vụ mang nặng tính I/O như vậy. Ngoài ra, với sự ổn định lâu dài của các webserver script khác, các tác vụ CRUD của nó đã được tối ưu hóa. Còn Node.js? Nó sẽ lòi ra những API cực kỳ kỳ quặc.

Project 3

- Khi cần sự ổn định trong ứng dụng của : Chỉ với 4 năm phát triển của mình (2009-2013), version của Node.js đã là 0.10.15 (hiện tại tới thời điểm này là v0.10.35). Mọi API đều có thể thay đổi – một cách không tương thích ngược – hãy thật cẩn thận với những API mà đang dùng, và luôn đặt câu hỏi: “Khi nó thay đổi, nó sẽ ảnh hưởng gì đến dự án của tôi?”
- Và quan trọng nhất: chưa hiểu hết về Node.js Node.js cực kỳ nguy hiểm trong trường hợp này, sẽ rơi vào một thế giới đầy rẫy cạm bẫy, khó khăn. Với phần lớn các API hoạt động theo phương thức non-blocking/async việc không hiểu rõ vấn đề sẽ làm cho việc xuất hiện những error mà thậm chí không biết nó xuất phát từ đâu? Và một mối lo hơn nữa: Khi cộng đồng Node.js chưa đủ lớn mạnh, và sẽ ít có sự support từ cộng đồng. Khi mà phần lớn cộng đồng cũng không khá hơn là bao.

Nên dùng Node.js khi nào?

- Node.js thực sự tỏa sáng trong việc xây dựng RESTful API (json). Gần như không có ngôn ngữ nào xử lý JSON dễ dàng hơn Javascript, chưa kể các API server thường không phải thực hiện những xử lý nặng nề nhưng lượng concurrent request thì rất cao. Mà Node.js thì xử lý non-blocking. Chẳng còn gì thích hợp hơn Node.js trong trường hợp này!
- Những ứng dụng đòi hỏi các giao thức kết nối khác chứ không phải chỉ có http. Với việc hỗ trợ giao thức tcp, từ nó có thể xây dựng bất kỳ một giao thức custom nào đó một cách dễ dàng.
- Những ứng dụng thời gian thực: Khỏi phải nói vì Node.js dường như sinh ra để làm việc này!
- Những website stateful. Node.js xử lý mọi request trên cùng một process giúp cho việc xây dựng các bộ nhớ đệm chưa bao giờ đơn giản đến thế: Hãy lưu nó vào một biến global, và thế là mọi request đều có thể truy cập đến bộ nhớ đệm đó. Caching sẽ không còn quá đầu đầu như trước đây, và có thể lưu cũng như chia sẻ trạng thái của một client với các client khác ngay trong ngôn ngữ, chứ không cần thông qua các bộ nhớ ngoài!
- Quan trọng nhất: yêu thích và muốn sử dụng nó.

6. Ví dụ Hello World trong Node.js

Dưới đây là đoạn mã xuất ra màn hình “Hello World” của Node.js :

```
var http = require('http');
http.createServer(function (request, response) {
  response.writeHead(200, { 'Content-Type': 'text/plain' });
  response.end('Hello World\n');
}).listen(8080, "127.0.0.1");
console.log('Server running at http://127.0.0.1:8080');
```

Chương 3. Trang web ứng dụng (Text Normalization)

Text Normalization là trang web sử dụng reactjs và nodejs và mô hình client-server kiểm tra độ chính xác của cá API chuẩn hóa văn bản bao gồm API chuẩn hóa từ viết tắt và API chuẩn hóa bằng luật đưa từ viết sai thành đúng.


Chức năng của trang web:

- Thêm thông tin cần kiểm tra: Thông tin thêm bao gồm từ cần chuẩn hóa (input) và kết quả mong muốn (expected) đúng. Có 3 cách thêm
 - + Thêm trực tiếp: chọn nút thêm mới và nhập từ vào các ô tương ứng
 - + Upload file: chọn nút upload để tải lên một file excel chứa dữ liệu tương ứng.
 - + Thêm thông qua link google sheet: nhập link google sheet chứa dữ liệu và chọn nút bên cạnh.
- Kiểm tra kết quả: Thông tin sau thêm sẽ hiển thị bảng bên dưới. Người dùng ấn RUN để kiểm tra. Kết quả chuẩn hóa từ API sẽ được hiển thị trong cột output. Kết quả đánh giá đúng sai (pass/fail) sau khi so sánh output và expected được hiển thị trong cột result. Tỷ lệ đúng sai hiển thị phía cuối góc trái bảng.
- Gửi Email kết quả kiểm tra: Sau toàn bộ dữ liệu kiểm tra thành công toàn bộ bảng kết quả sẽ được gửi về email người dùng.

Project 3


THÊM MỚI +1

UPLOAD

Type URL 

INPUT	EXPECTED	OUTPUT	RESULT
~đh#	đại học		
~ đh # bách khoa	đại học bách khoa		
đại học bach khoa ha noi	đại học bách khoa hà nội		
~ đh #	đại học		
đại học ~ bk # hà nội	đại học bách khoa hà nội		
dai hoc ~ bk # hà nội	đại học bách khoa hà nội		
~ đh # bách khoa ~ hn #	đại học bách khoa hà nội		

Rows per page: 10 ▾ 1-8 of 8 < >

PASS	0/8 (0.0 %)	CLEAR 
FAIL	0/8 (0.0 %)	

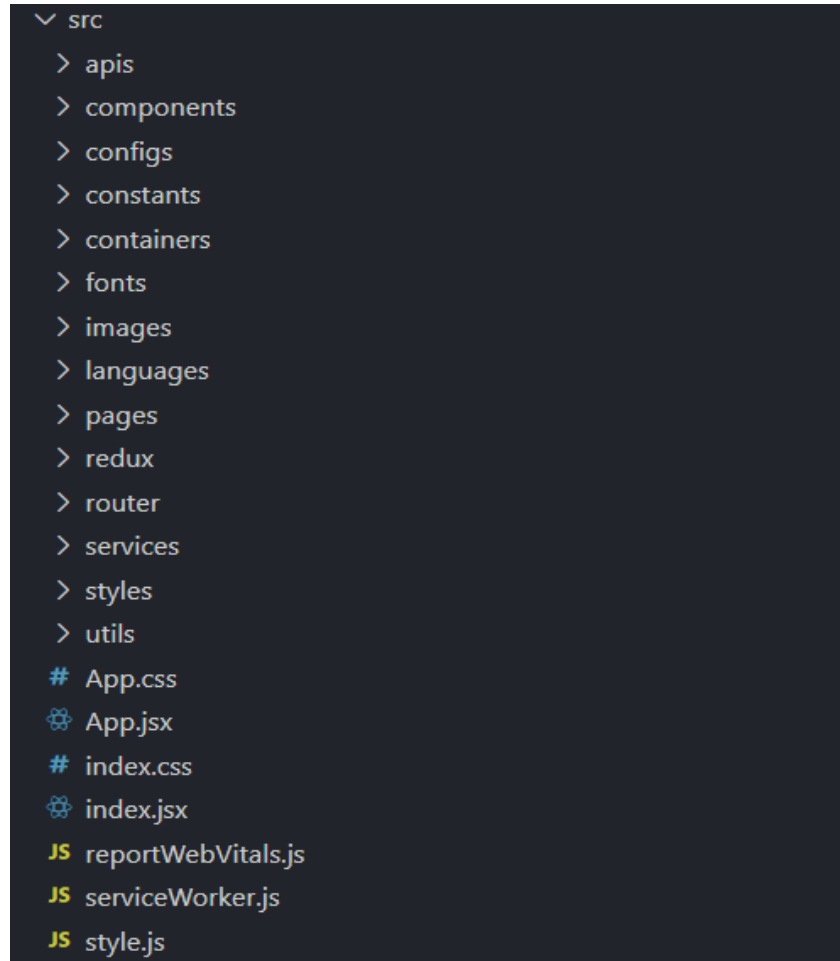
RUN ▶

Giao diện hệ thống

Project 3

1. Client

Sử dụng Reactjs để xây dựng client cho Project

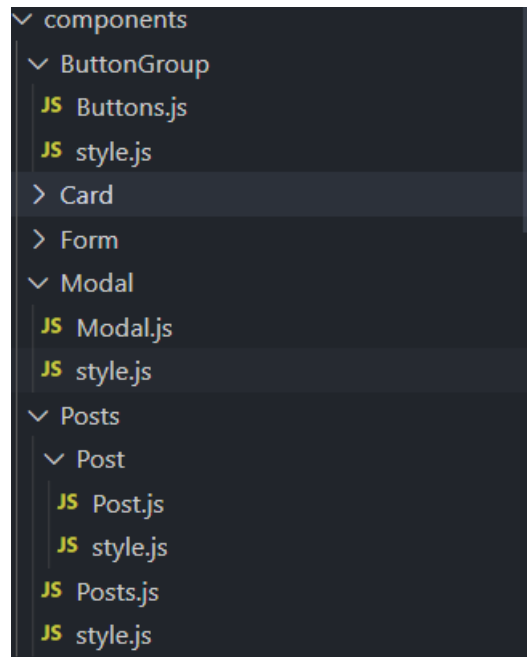


Cấu trúc quản lý code của client

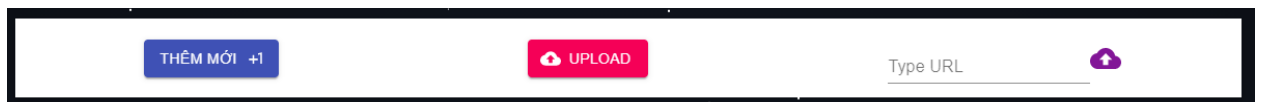
Như đã giới thiệu ở trên thì việc React sẽ chia ra thành các component để quản lý và render ra giao diện thì em đã chia các folder trong thư mục src theo đúng mục đích.

Folder components :

Project 3



- ButtonsGroup/Buttons.js : Component chứa các nút thực hiện các việc như đóng mở Modal thêm 1 dữ liệu, Chọn file excel từ máy tính hay import dữ liệu từ url online.



- Modal/Modal.js : Component chứa 1 Modal thực hiện việc thêm mới 1 dữ liệu vào trong bảng. Được hiển thị dựa vào việc thay đổi trạng thái sau khi Button THÊM MỚI +1 click

Project 3

Thêm mới

Văn bản cần chỉnh sửa

Kết quả mong muốn

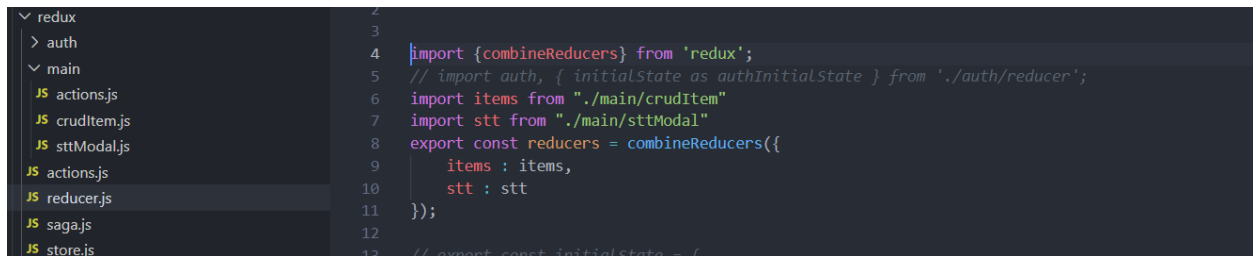
SUBMIT

- Posts/Posts.js : Component chứa bảng bao gồm thông tin của dữ liệu gốc, dữ liệu mong muốn, thống kê đúng sau, nút để gọi api chạy toàn bộ bảng, nút xóa trắng bảng dữ liệu.

INPUT	EXPECTED	OUTPUT	RESULT
~đh#bách khoa	đại học bách khoa		
Rows per page: 10 1-1 of 1 < >			
PASS	0/1 (0.0 %)		CLEAR
FAIL	0/1 (0.0 %)		RUN ▶

Project 3

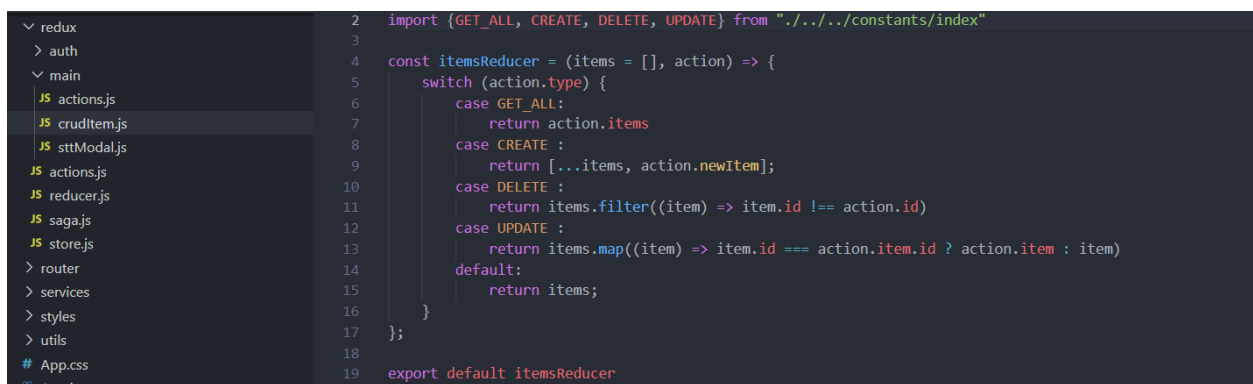
Folder redux :



```
1
2
3
4 import {combineReducers} from 'redux';
5 // import auth, { initialState as authInitialState } from './auth/reducer';
6 import items from "../main/crudItem"
7 import stt from "../main/sttModal"
8 export const reducers = combineReducers({
9   items : items,
10  stt : stt
11 });
12
13 // export const initialState = {
```

File reducer.js

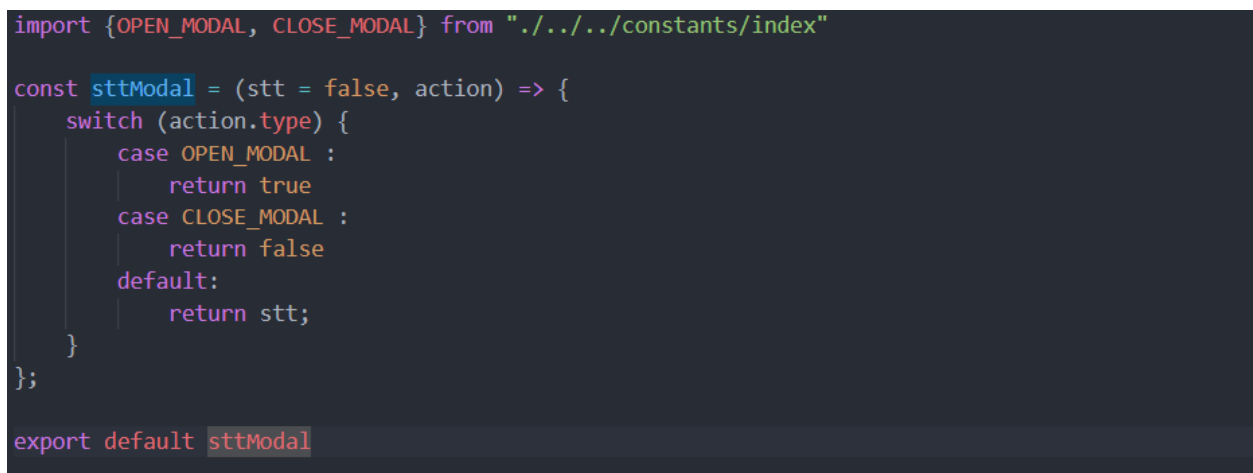
Các reducer chính là nơi nhận những thay đổi và trả về giá trị mới thông qua actions được yêu cầu từ các components.



```
1
2 import {GET_ALL, CREATE, DELETE, UPDATE} from "../constants/index"
3
4 const itemsReducer = (items = [], action) => {
5   switch (action.type) {
6     case GET_ALL:
7       return action.items
8     case CREATE :
9       return [...items, action.newItem];
10    case DELETE :
11      return items.filter((item) => item.id !== action.id)
12    case UPDATE :
13      return items.map((item) => item.id === action.item.id ? action.item : item)
14    default:
15      return items;
16   }
17 }
18
19 export default itemsReducer
```

File crudItems.js

itemReducer là kết quả trả về sau khi thực hiện các components yêu cầu reducer thực hiện action thêm, sửa xóa dữ liệu trong bảng và được lưu vào store để hiển thị dữ liệu ra bảng.



```
1 import {OPEN_MODAL, CLOSE_MODAL} from "../constants/index"
2
3 const sttModal = (stt = false, action) => {
4   switch (action.type) {
5     case OPEN_MODAL :
6       return true
7     case CLOSE_MODAL :
8       return false
9     default:
10      return stt;
11   }
12 }
13
14 export default sttModal
```

File sttModal.js

Project 3

sttModal là kết quả trả về sau khi components (Button Thêm Mới +1) yêu cầu reducer thực hiện action đổi trạng thái hiển thị của Modal thêm mới. Trạng thái này sẽ lưu vào store và khi component Modal nhận được là True thì sẽ bật lên. Còn không sẽ ẩn đi.

```
export const openModal = () => async(dispatch) => {
  try {
    dispatch({type : OPEN_MODAL})
  } catch (error) {
    console.log(error.message)
  }
}

export const closeModal = () => async(dispatch) => {
  try {
    dispatch({type : CLOSE_MODAL})
  } catch (error) {
    console.log(error.message)
  }
}
```

actions openModal và closeModal trong file actions.js

Các functions này thực hiện các actions với type tương ứng dùng để cập nhập trạng thái của Components Modal thêm mới vào trong sttReducer trong store.

```
import {GET_ALL, DELETE, CREATE, UPDATE, OPEN_MODAL, CLOSE_MODAL} from '../constants/index'
import * as api from '../apis/callApi'

export const getAllItems = () => async(dispatch) => {
  try {
    dispatch({type : GET_ALL, items : []})
  } catch (error) {
    console.log(error.message)
  }
}

export const createNewItem = (newItem) => async(dispatch) => {
  try {
    dispatch({type : CREATE, newItem : newItem})
  } catch (error) {
    console.log(error.message)
  }
}

export const updateItem = (id, item) => async(dispatch) => {
  try {
    dispatch({type : UPDATE, currentItem : {item : item, id : id}})
  } catch (error) {
    console.log(error.message)
  }
}
```

actions getAllItems, createNewItem, updateItem

Các functions này thực hiện các actions với type tương ứng dùng để load items từ store ra

Project 3

bảng, Thêm mới dữ liệu vào trong items trong store và cập nhập item vào store.

Foder src/apis:



```
src
├── apis
│   ├── api.js
│   ├── auth.js
│   ├── callApi.js
│   └── index.js
└── ...

4 import axios from 'axios'
5
6 const url = 'http://localhost:5000'
7
8 export const expandWordApi = (info) => axios.post(`${url}/expand`, info)
9 export const postDataExcel = (data) => axios.post(`${url}/data2excel`, data)
```

File src/apis/callApi.js

Chứa các file để gọi api của server (ở đây là Nodejs) thông qua package “axios” thuận tiện cho việc request và nhận response dữ liệu dạng json. Ở đây em viết 2 api chính.

API expand : để expand các từ cần được làm rõ nghĩa dựa theo ngữ cảnh. Dữ liệu info ở đây là từng cặp viết tắt và từ ngữ cảnh làm rõ (VD : ~đh#bách khoa)

API send mail : để gửi file kết quả với định dạng excel về số liệu và bảng kết quả đã chạy expands. Data gắn ở request sẽ là list dữ liệu lấy từ bảng kết quả và thông số về độ chính xác.

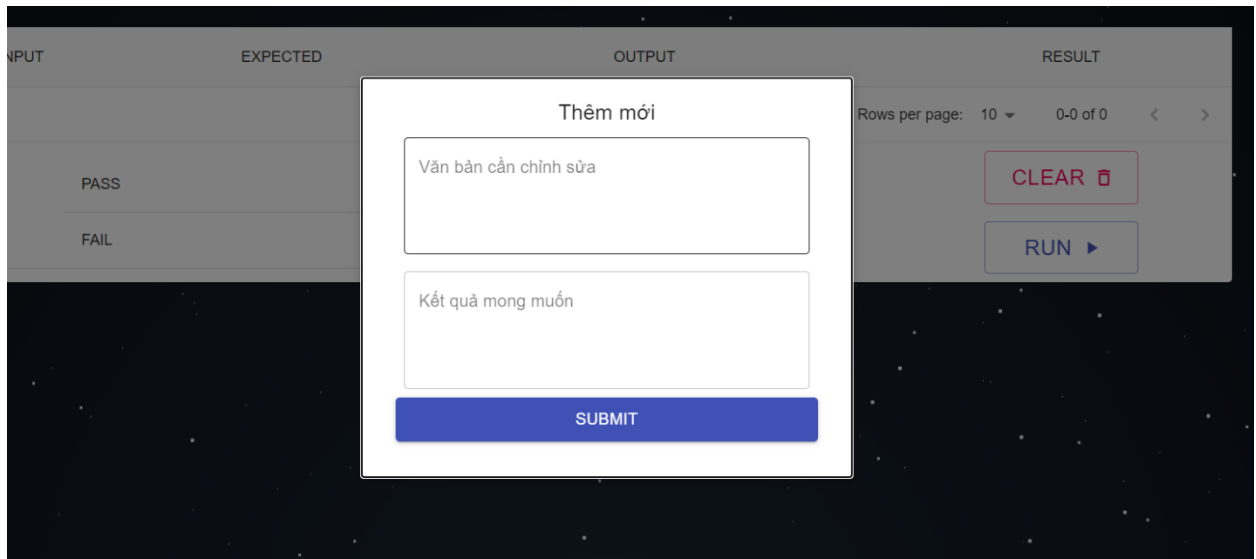
Các chức năng chính phía Client :

- Chức năng thêm mới 1 dữ liệu:

Mỗi khi Button Thêm mới +1 click thì sẽ dispatch 1 action có type : OPEN_MODAL. .

Và reducer sẽ thực hiện action và lưu sttModal thành true. Modal thêm mới thông qua hàm useSelector sẽ lấy sttModal = true từ store để bật lên. Và sẽ set giá trị mặc định ở các ô input thông qua hàm useState của React hook thành trống.

Project 3



Modal hiển thị khi chọn Thêm Mới +1

Sau khi click vào Submit. Component sẽ lấy thông tin của text trong 2 ô input trên đồng thời tạo ra 1 uuid (sẽ giải thích vì sao cần) và

thực thi `createNewItem` ở trong file `redux/main/actions.js` và dispatch 1 action có type :

`CREATE` để thêm mới 1 dữ liệu cần làm rõ, đồng thời sẽ dispatch 1 action `closeModal` có type : `CLOSE_MODAL` để đóng Modal lại.

```
const classes = useStyles();
const [postData, setPostData] = useState({input : '', expected : '', output : '', result : '', id : ''})
const stt = useSelector(state => state.stt)
const dispatch = useDispatch()
const handleSubmit = async(e) => {
  e.preventDefault()
  postData.id = uuidv4()
  dispatch(createNewItem(postData))
  dispatch(closeModal())
  setPostData({input : '', expected : '', output : '', result : '', id : ''})
}
```

Code thực hiện submit dữ liệu thêm mới

- Chức năng thêm mới nhiều dữ liệu từ file .xls:

Sau khi chọn file .xls trong máy và dùng package “xlsx” trong react để load dữ liệu thì ta sẽ nhận được 1 list các items, mỗi item bao gồm dữ liệu muốn expand và dữ liệu mong muốn, sẽ được mở rộng thêm các trường như id, result, output. Mỗi 1 item sẽ thực thi `createNewItem` ở trong file `redux/main/actions.js` và dispatch 1 action có type : `CREATE` để thêm mới 1 dữ liệu cần làm rõ.

Project 3

```
const handleUpload = (e) => {
  e.preventDefault();

  var files = e.target.files, f = files[0];
  var reader = new FileReader();
  reader.onload = function (e) {
    excel2datas = []
    var data = e.target.result;
    let readedData = XLSX.read(data, {type: 'binary'});
    const wsname = readedData.SheetNames[0];
    const ws = readedData.Sheets[wsname];

    /* Convert array to json*/
    let dataParse = XLSX.utils.sheet_to_json(ws, {header:1});
    dataParse.splice(0, 1)
    excel2datas = dataParse.map(data => ({input : data[0], expected : data[1], output : '', result : '', id : uuidv4()}))
  };
  reader.readAsBinaryString(f)
}

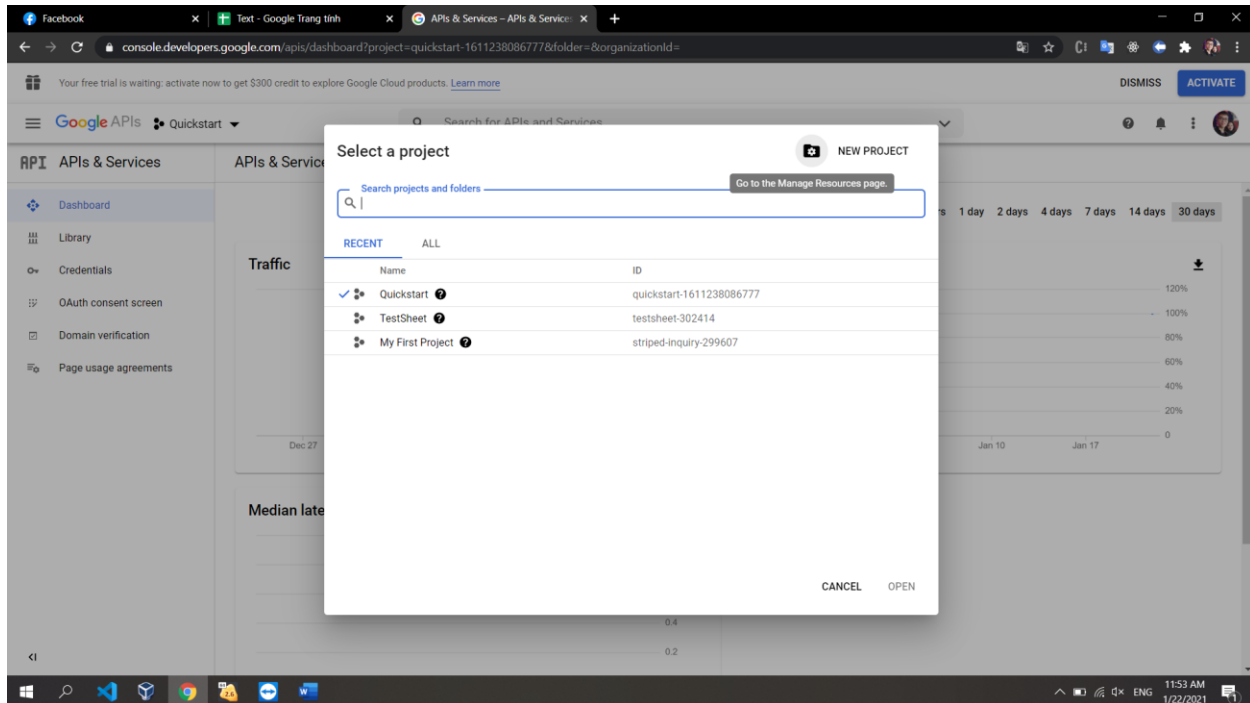
const handlePostDataFromExcel = () => {
  excel2datas.forEach((newItem) => dispatch(createNewItem(newItem)))
  excel2datas = []
  setdatainput('')
}
```

Code thực hiện thêm nhiều dữ liệu từ excel

- Chức năng thêm dữ liệu từ url

Các bước thực hiện :

Tạo project api của google : <https://console.developers.google.com/apis>



Project 3

The first screenshot shows the 'New Project' page in the Google Cloud Platform console. The browser tabs include Facebook, Text - Google Trang tính, and New Project - Google API Console. The URL is console.developers.google.com/projectcreate?previousPage=%2Fapis%2Fdashboard%3Fproject%3Dtestsheet-302414&folder=&organizationId=0. A message at the top states: 'Your free trial is waiting: activate now to get \$300 credit to explore Google Cloud products. [Learn more](#)'. There are 'DISMISS' and 'ACTIVATE' buttons. The 'Project name' field contains 'SheetProject'. Below it, the 'Project ID' is 'sheet-project-302504'. The 'Location' is set to 'No organization'. There are 'CREATE' and 'CANCEL' buttons at the bottom.

The second screenshot shows the 'APIs & Services' dashboard for the 'Sheet Project'. The browser tabs include Facebook, Text - Google Trang tính, and APIs & Services - APIs & Services. The URL is console.developers.google.com/apis/dashboard?project=sheet-project-302504&folder=. A message at the top states: 'You don't have any APIs available to use yet. To get started, click "Enable APIs and services" or go to the [API library](#)'. The left sidebar shows a navigation menu with 'Dashboard', 'Library', 'Credentials', 'OAuth consent screen', 'Domain verification', and 'Page usage agreements'. The main content area has a '+ ENABLE APIS AND SERVICES' button. A toast notification at the bottom says: 'Now viewing project "Sheet Project" in organization "No organization"'. The Windows taskbar at the bottom shows the time as 11:55 AM on 1/22/2021.

Project 3

The screenshot shows the Google Cloud Platform console interface. At the top, there's a navigation bar with the Google APIs logo and a dropdown menu for 'Sheet Project'. Below this, the 'API Library' is displayed, showing a list of APIs categorized by Machine learning, Google Workspace, and YouTube. The Google Drive API is highlighted, and its details are shown in a modal or expanded view. The details include the API name, description, and a button to 'ENABLE' it. Below the details, there's a section for 'Overview' and 'Additional details'.

Machine learning

- Dialogflow API**
Google
Builds conversational interfaces
- Cloud Vision API**
Google
Image Content Analysis
- Cloud Natural Language API**
Google
Provides natural language understanding technologies, such as sentiment analysis, entity...
- Cloud Speech-to-Text API**
Google
Speech recognition

Google Workspace

- Google Drive API**
Google
The Google Drive API allows clients to access resources from Google Drive
- Google Calendar API**
Google
Integrate with Google Calendar using the Calendar API
- Gmail API**
Google
Flexible, RESTful access to the user's inbox
- Google Sheets API**
Google
The Sheets API gives you full control over the content and appearance of your spreadsheet

YouTube

Google Drive API
Google

The Google Drive API allows clients to access resources from Google Drive

ENABLE **TRY THIS API**

Click to enable this API

OVERVIEW **DOCUMENTATION**

Overview

The Google Drive API allows clients to access resources from Google Drive.

About Google

Google's mission is to organize the world's information and make it universally accessible and useful. Through products and platforms like Search, Maps, Gmail, Android, Google Play, Chrome and YouTube, Google plays a meaningful role in the daily lives of billions of people.

Additional details

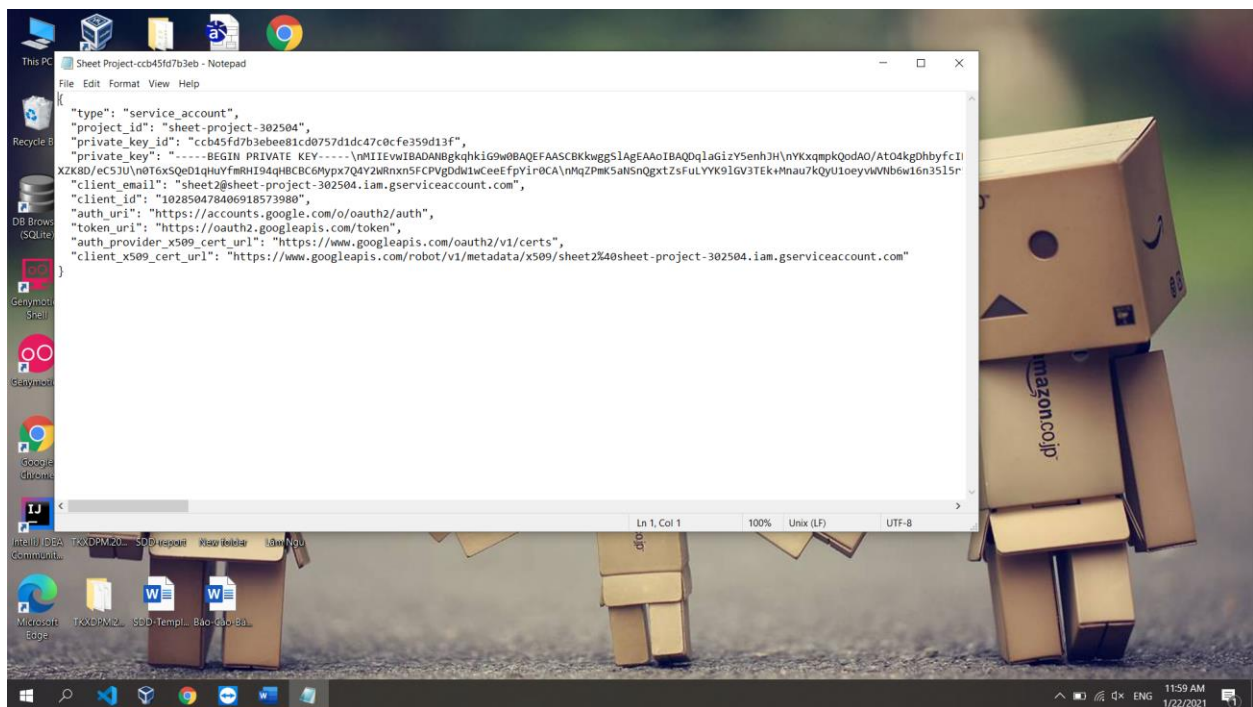
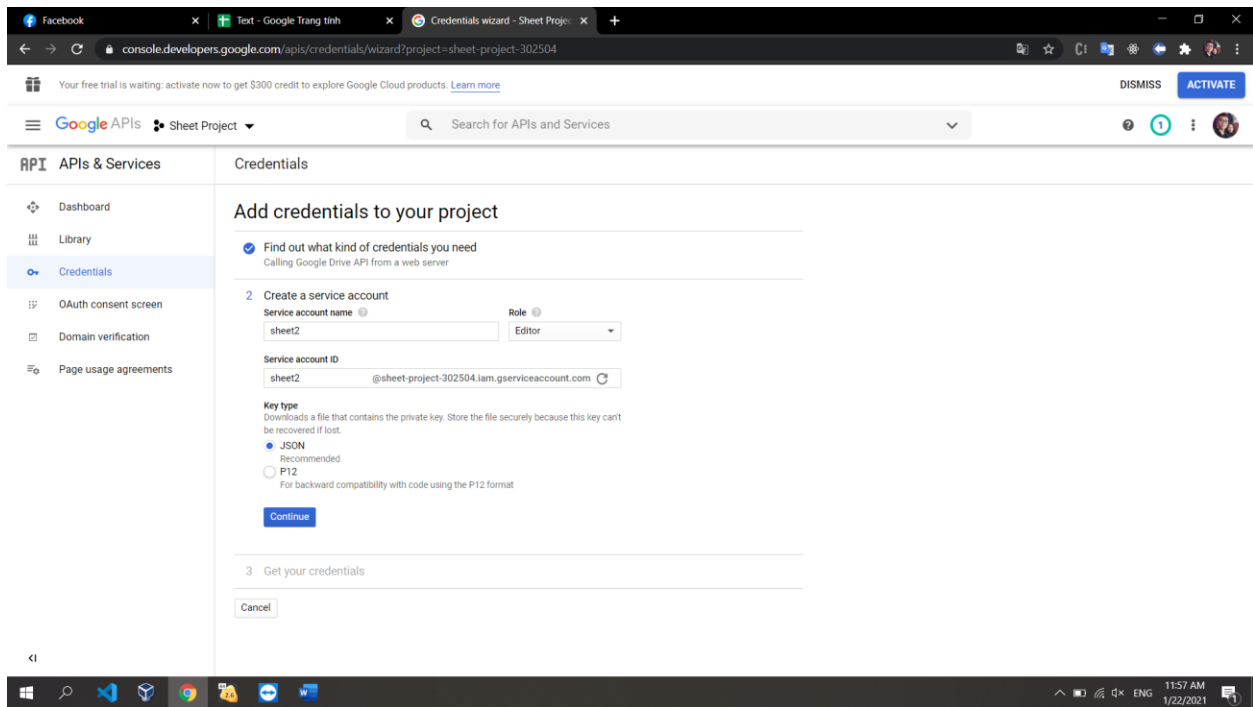
Type: [APIs & services](#)
Last updated: 12/10/19
Category: [Storage](#), [Google Workspace](#)
Service name: drive.googleapis.com

Project 3

The first screenshot shows the Google Cloud Platform console for the Google Drive API. The left sidebar contains a navigation menu with 'Overview', 'Metrics', 'Quotas', 'Credentials', and 'Drive UI Integration'. The main content area is titled 'Overview' and includes a 'DISABLE API' button. A message states: 'To use this API you may need credentials. Click 'Create credentials' to get started.' Below this, there is a 'Details' section with the following information: Name: Google Drive API, By: Google, Service name: drive.googleapis.com, Overview: The Google Drive API allows clients to access resources from Google Drive, and Activation status: Enabled. To the right of the details is a 'Traffic by response code' chart showing 'Request/sec (2 hr average)' with a y-axis from 0 to 1.0% and an x-axis from Dec 27 to Jan 17. A message indicates 'No data is available for the selected time frame.' Below the details is a 'Tutorials and documentation' section with links for 'Learn more' and 'Try in API Explorer'. A 'CREATE CREDENTIALS' button is located in the top right corner of the overview section.

The second screenshot shows the 'Credentials wizard' for the Google Drive API. The left sidebar contains a navigation menu with 'Dashboard', 'Library', 'Credentials', 'OAuth consent screen', 'Domain verification', and 'Page usage agreements'. The main content area is titled 'Credentials' and includes a '1 Find out what kind of credentials you need' section. This section contains several questions and options: 'Which API are you using?' with 'Google Drive API' selected; 'Where will you be calling the API from?' with 'Web server (e.g. node.js, Tomcat)' selected; 'What data will you be accessing?' with 'Application data' selected; and 'Are you planning to use this API with App Engine or Compute Engine?' with 'No, I'm not using them' selected. A 'What credentials do I need?' button is located at the bottom of the wizard.

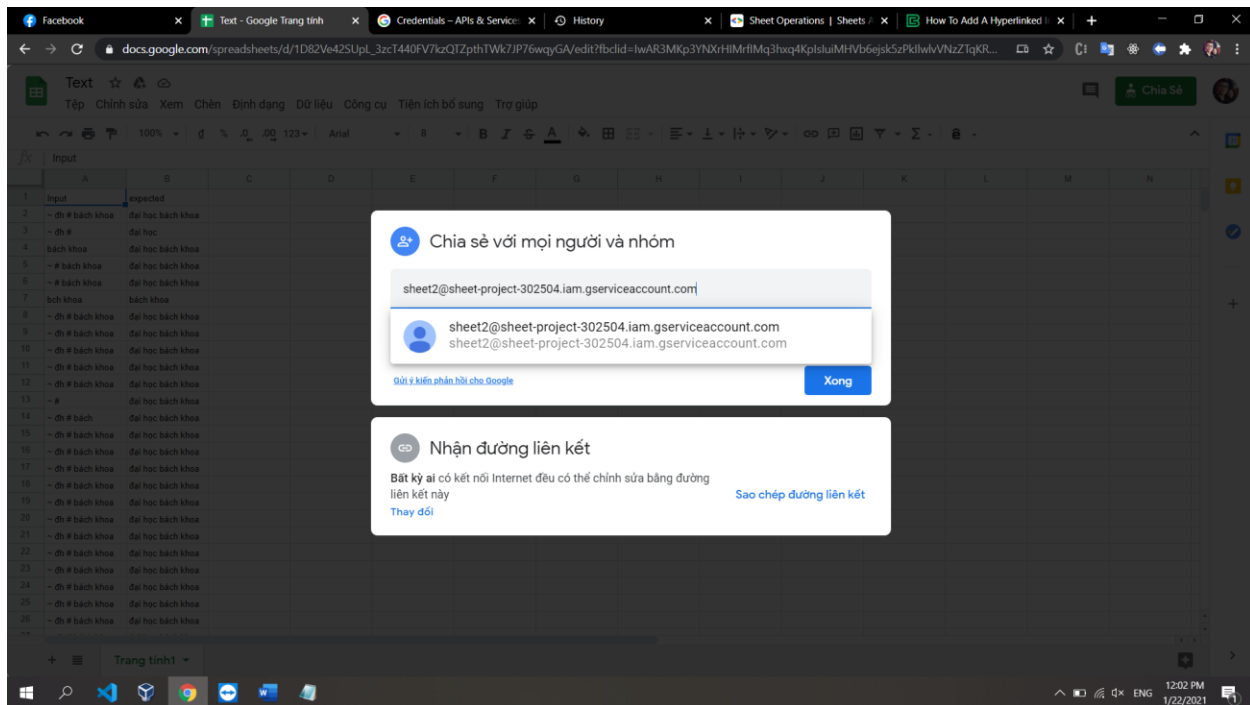
Project 3



Định dạng file json thu được

Sao chép client_email và quay trở lại trang spreadsheets của google để chia sẻ tài liệu này cho client_email

Project 3



URL của google sheet đã được chia sẻ và có thể sử dụng vào Ứng dụng. Phần config để server đọc được url em sẽ giải thích ở phần sau.

Server sau khi giải mã url sẽ trả về 1 list các items, mỗi item bao gồm dữ liệu muốn expand và dữ liệu mong muốn, sẽ được mở rộng thêm các trường như id, result, output. Mỗi 1 item sẽ thực thi createNewItem ở trong file redux/main/actions.js và dispatch 1 action có type : CREATE để thêm mới 1 dữ liệu cần làm rõ.

```
const uploadDataFromUrl = async () => {  
  let datas = await addDataFromUrl({url : dataInput.url_input})  
  console.log(datas)  
  datas.forEach((data) => {  
    dispatch(createNewItem({input : data.input, expected : data.expected, output : '', result : '', id : uuidv4()}))  
  })  
  setDataInput({...dataInput, url_input : ''})  
}
```

Đoạn code thực hiện thêm dữ liệu từ url

- Chức năng chạy toàn bộ dữ liệu trong bảng.

Component Posts/Posts.js sẽ sử dụng useSelector của redux để load các items trong store và đổ dữ liệu ra bảng sau khi xử lý dữ liệu (tính toán độ đúng sai, pass/fail).

Sau khi click vào button RUN trong components Posts/Post.js. Các item trong items chưa

Project 3

được gọi api (fail hoặc pass) sẽ lần lượt được gọi thông qua hàm expandCallAPI trong redux/main/actions.js để gọi api lên server Nodejs xử lý và nhận về dữ liệu. Dữ liệu nhận về có thể fail hoặc pass sẽ được dispatch 1 action có type : UPDATE để update fail hoặc pass dựa theo id mà đã tạo ra trong package “uuidv4” ở quá trình createNew.

```
const dispatch = useDispatch()
const items = useSelector(state => state.items)
const total = items.length
let pass = 0
let fail = 0
items.forEach((i) => {
  if (i.result === 'pass') pass += 1
  else if (i.result === 'fail') fail += 1
})
const text1 = `${pass}/${total} (${total !== 0 ? (pass*100/total).toFixed(1) : 0} %)`
const text2 = `${fail}/${total} (${total !== 0 ? (fail*100 /total).toFixed(1): 0} %)`
const handleChangeRowsPerPage = (event) => {
  setRowsPerPage(+event.target.value);
  setPage(0);
};

const handleCallApi = async (e) => {
  e.preventDefault()
  let reqs = []
  items.forEach((item) => {if(item.result === '') reqs.push(item)})
  if(reqs.length) {
    await dispatch(expandCallAPI(reqs))
    handleCallSendEmail()
  }
}
```

Code thực hiện quá trình call api expand từ client lên nodejs

Các items sau khi được gọi api hết sẽ thực hiện hàm handleCallSendEmail() để gửi dữ liệu sẽ được Nodejs mã hóa về dạng .xls và gửi file cho email.

Project 3

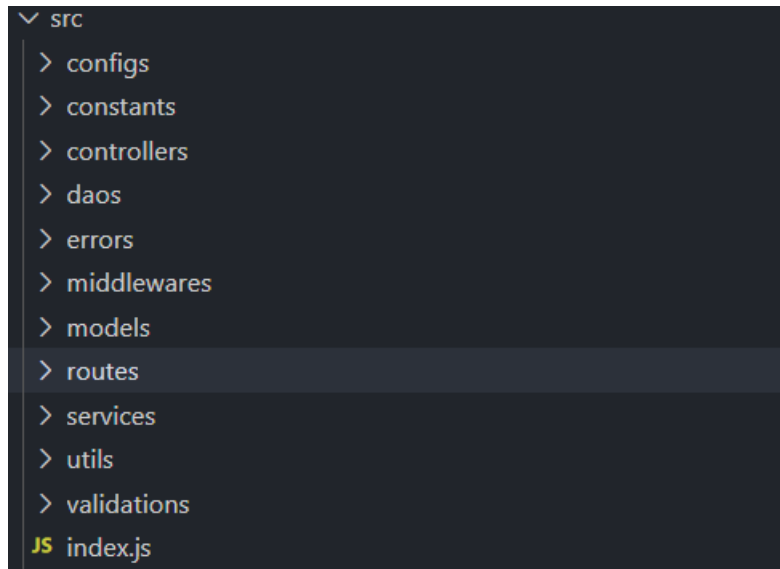
```
const handleCallSendEmail = () => {  
  let pass = 0  
  let fail = 0  
  
  items.forEach((i) => {  
    if (i.result === 'pass') pass += 1  
    else if(i.result === 'fail') fail += 1  
  })  
  let text1 = `${pass}/${total} (${total !== 0 ? (pass*100/total).toFixed(1) : 0} %)`  
  let text2 = `${fail}/${total} (${total !== 0 ? (fail*100 /total).toFixed(1): 0} %)`  
  
  let data2excel = {  
    items : items,  
    pass : text1,  
    fail : text2  
  }  
  sendMailExcel(data2excel)  
}
```

Code thực hiện việc gửi dữ liệu sẽ thành .xls

Project 3

2. Server

Sử dụng Nodejs để làm server cho Project



Cấu trúc quản lý code của server

Project được xây dựng dựa trên Mô hình MVC(Model-View-Controller) với Reactjs là View. Phần Model và Controller nằm ở phía Server Nodejs.

Folder src/configs :

```
const {
  PORT,

  MONGO_HOST,
  MONGO_PORT,
  MONGO_DATABASE,
  MONGO_USERNAME,
  MONGO_PASSWORD,

  JWT_SECRET_KEY,
  JWT_EXPIRES_TIME,
} = process.env;

const { A_WEEK } = require('../constants');

module.exports = {
  PORT: PORT || 3000,
  MONGO_URI: `mongodb://${MONGO_HOST}:${MONGO_PORT}/${MONGO_DATABASE}`,
  JWT_SECRET_KEY,
  JWT_EXPIRES_TIME: parseInt(JWT_EXPIRES_TIME, 10) || A_WEEK,
};
```

File index.js

Project 3

File chứa các config mặc định cho ứng dụng như port, host, uri connect với cơ sở dữ liệu (MongoDB).

Folder src/Controllers :

```
const authService = require('../services/auth');

const register = async (req, res) => {
  const { email, name, password } = req.body;
  const user = await authService.register({ email, name, password });
  return res.send({ status: 1, result: user });
};

const login = async (req, res) => {
  const { email, password } = req.body;
  const accessToken = await authService.login(email, password);
  return res.send({ status: 1, result: { accessToken } });
};

const verifyAccessToken = async (req, res) => {
  const { accessToken } = req;
  const { user } = await authService.verifyAccessToken(accessToken);
  res.send({ status: 1, result: { user } });
};

module.exports = { register, login, verifyAccessToken };
```

File auth.js

authController có vai trò thực hiện đăng ký, đăng nhập, xác thực token, kết nối trực tiếp với client thông qua các Router trong folder router và sử dụng Services trong folder Services để xử lý dữ liệu request và trả về dữ liệu cho client.

Folder services :

Chứa các file thực thi các hàm mà Controller sẽ cần, kết nối trực tiếp với Model thông qua xử lý bất đồng bộ

Folder router :

Chứa các file điều hướng dữ liệu cho controller mỗi khi client get, post, put, ... lên server.

Project 3

2.1. Thực hiện api Expand

```
router.post('/expand', async(req, res) => {
  let data2req = req.body
  let adr1 = data2req.sentenceWithAbbrev.indexOf("~")
  let adr2 = data2req.sentenceWithAbbrev.indexOf("#")

  if(adr1 === -1 || adr2 === -1){
    try {
      let res2normalize = await axios.post('http://43.239.223.87:5000/text_normalize', {sentence : data2req.sentenceWithAbbrev})
      return res.status(200).send({expand : res2normalize.data.content})
    } catch (error) {
      return res.status(500).send("Lỗi rồi :))")
    }
  }else{
    let normalize1 = data2req.sentenceWithAbbrev.slice(0, adr1).trim()
    let normalize2 = data2req.sentenceWithAbbrev.slice(adr1 + 1, adr2).trim()
    let expand = data2req.sentenceWithAbbrev.slice(adr2 + 1).trim()
    try {
      let res2normalize1 = await axios.post('http://43.239.223.87:5000/text_normalize', {sentence : normalize1}) : ''
      let res2normalize2 = await axios.post('http://43.239.223.87:5000/text_normalize', {sentence : normalize2}) : ''
      data2req = {sentenceWithAbbrev : (`${res2normalize1} ~ ${expand} # ${res2normalize2}`).trim() }
      let {data} = await axios.post('http://43.239.223.87:5050/expand', data2req)
      data.expand = (normalize1 + " " + (data.expand !== 'null' ? data.expand : expand) + " " + normalize2).trim()
      return res.status(200).send(data)
    } catch (error) {
      return res.status(500).send("Lỗi rồi :))")
    }
  }
})
```

Sau khi client post data tới server thông qua phương thức post("/expand"), dữ liệu sẽ được giải mã dạng json thông qua req.body (sử dụng package "bodyParser" của Nodejs). Dữ liệu sẽ được xử lý từng phần viết tắt và từ ngữ cảnh để gọi api nomalize (bach khoa) thông qua "axios" tới server khác (trả về bách khoa). Sau khi nhận được kết quả thì sẽ gọi api expand (~đh#bách khoa) để lấy được đầy đủ thông tin của từ đh (đại học) và trả về cho Client.

2.2. Thực hiện chức năng SendMail

```
router.post("/data2excel", async(req, res) => {
  try {
    const buffer = await bufferExcel(req.body.items, req.body.pass, req.body.fail)
    const filename = 'lamthon.xlsx';
    const transporter = createTransport({
      host: process.env.MAIL_HOST,
      port: process.env.MAIL_PORT,
      secure: false,
      auth: {
        user: process.env.MAIL_PUBLIC_ME,
        pass: process.env.PASS_PUBLIC_ME,
      },
    });
    const mailOptions = {
      from: process.env.MAIL_PUBLIC_ME,
      to: [process.env.MAIL_SEND],
      subject: 'Lâm thôn App: Kết quả chạy ứng dụng :))',
      html: `<h2> Bạn nhận được Email vì đã sử dụng ứng dụng của Lâm thôn</h2>
        <h3>Đọc file để xem kết quả :D</h3>`,
      attachments: [
        {
          filename,
          content: buffer,
          contentType:
            'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
        },
      ],
    };
    await transporter.sendMail(mailOptions);
    return res.status(200).send("OKKKKK rồi nha :)")
  } catch (error) {
    return res.status(500).send("Lỗi rồi :(")
  }
})
```

Sau khi Client gửi yêu cầu thông qua phương thức post(/data2excel) thì dữ liệu được đọc thông qua req.body và xử lý trong hàm buffExcel với mục đích binary dữ liệu dạng .xls để gửi email. Thực hiện cấu hình transporter với host, port, auth được config trong file env. Và sử dụng package “nodemailer” của Nodejs thực hiện việc gửi mail file .xls chứa kết quả và thống kê giữa 2 email với nhau. Gửi thành công sẽ trả về cho Client còn không sẽ báo lỗi cho Client.

3. Thực thi việc giải mã url và trả về data

Cài đặt : Đặt file json đã tải được sau khi cài đặt project api của google vào folder src/router. Và dùng package “google-spreadsheet” để đọc id của url và trả về kết quả cho client.

Project 3

```
router.post("/get-data-from-url", async(req, res) => {
  let url = req.body.url
  let start_id = url.indexOf("/d/") + 3
  let end_id = url.indexOf("/edit?")
  console.log(url)
  try {
    let id = url.slice(start_id, end_id)
    const doc = new GoogleSpreadsheet(id)
    await promisify(doc.useServiceAccountAuth)(creds)
    const info = await promisify(doc.getInfo)()
    const sheet = info.worksheets[0]
    const rows = await promisify(sheet.getRows)({
      offset : 1
    })
    let data2res = []
    rows.forEach(row => {
      if(row.input && row.expected) data2res.push({input : row.input, expected : row.expected })
    })
    return res.status(200).send(data2res)
  } catch (error) {
    return res.status(500).send("Lỗi doc url :))")
  }
})
```

Đoạn code thực hiện đọc dữ liệu từ google sheet thông qua nodejs

Chương 4. Phụ lục

1. Hướng dẫn cài đặt

- B1: mở folder text normalization (có thể mở bằng các IDE hỗ trợ code nodejs và reactjs)
- B2: Mở command line với 2 file client và server.
- B3: Chạy lệnh npm install trên cả 2 màn hình command line
- B4: vào file /server/.env thiết lập email dùng để gửi và nhận



- B5: Chạy lệnh npm start trên cả 2 màn hình command line

2. Tài liệu tham khảo

- Các trang học lập trình nodejs, reactjs:
 - + https://www.tutorialspoint.com/nodejs/nodejs_first_application.htm
 - + <https://www.w3schools.com>
- Google Api:
<https://developers.google.com/sheets/api/quickstart/nodejs>