Kiểu viết scss khác css:

Scss là kiểu viết nested

Cách viết scss

Viết scss gọn hơn, tối ưu hơn css

**npm install body-parser@1.19.0 dotenv@8.2.0 ejs@3.1.5** [**express@4.17.1**](mailto:express@4.17.1)

Body-parser: lấy các tham số mà client gửi lên server

Dotenv: dùng câu lệnh process.env. .Lấy tham số khi báo trong môi trường

Ejs: view engine

Express: giúp chạy project

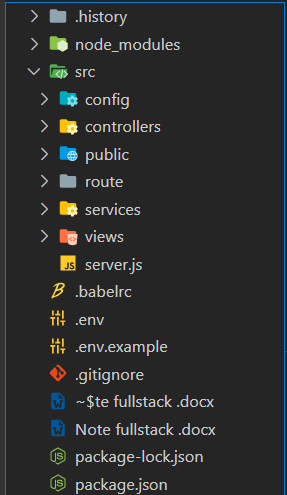
**npm install --save-dev @babel/core@7.12.10 @babel/preset-env@7.12.10 @babel/node@7.12.10** [**nodemon@2.0.7**](mailto:nodemon@2.0.7)

babel là compile để dịch bất kể phiên bản nào của javascript(trong 1 dự án)

nodemon để có sự thay đổi file thì server tự restart

Ejs giống như file jsp(java) và blade(PHP)

# Mô hình MVC



## Controllers

let getHomePage = (req, res) => {

    // vì ở view engine đã cài đặt là nằm ở src/views nên không cần đường dẫn

    // render ra view

    return res.render('homepage.ejs');

}

let getAboutPage = (req, res) => {

    return res.render('test/about.ejs')

}

module.exports = {

    getHomePage: getHomePage,

    getAboutPage:getAboutPage

}

## Route

import express from 'express';

import homeController from '../controllers/homeController';

let router = express.Router();

let initWebRoutes = (app) => {

    router.get('/', homeController.getHomePage);

    router.get('/about', homeController.getAboutPage);

    return app.use("/", router);

}

module.exports = initWebRoutes;

## Views

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Huy dz</h1>

</body>

</html>

## Config

import express from 'express';

let configViewEngine = (app) => {

    // static để server biết chỉ được lấy data từ đâu

    app.use(express.static("./src/public"));

    app.set("view engine", "ejs");

    //đường link lấy view engine

    app.set("views","./src/views")

}

// export ra

module.exports = configViewEngine;

## Server

import express from "express";

// lấy tham số từ client trả về server. VD/user?id=7

import bodyParser from "body-parser";

import viewEngine from "./config/viewEngine";

import initWebRoutes from "./route/web";

// gọi tới config của thư viện dotenv để có thể process.env.

require('dotenv').config();

// tạo 1 instance của express

let app = express();

// config app

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({extended: true}))

viewEngine(app);

initWebRoutes(app);

// lấy ra từ file env

let port = process.env.PORT || 6969;

app.listen(port, () => {

    console.log("Backend Nodejs is running on port " + port)

})

# Cài đặt sequelize CLI

Command line interface

* Cung cấp câu lệch để thao tác với cơ sở dữ liệu

Npm install –save-dev [sequelize-cli@6.2.0](mailto:sequelize-cli@6.2.0)

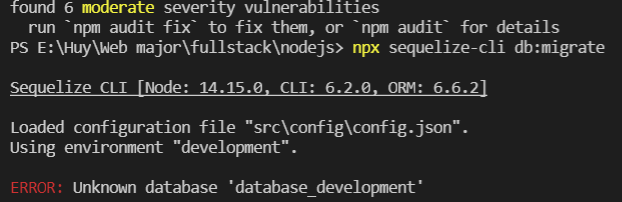
Link tham khảo: https://dev.to/ceceliacreates/sequelize-basics-for-beginners-part-one-2lc6

Fix lỗi khi cài sequelize

<https://dev.to/tradecoder/how-to-fix-error-nodemon-ps1-cannot-be-loaded-because-running-scripts-is-disabled-on-this-system-34fe>

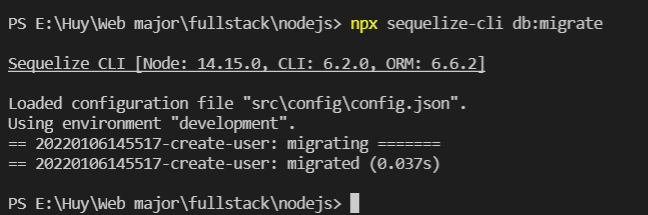
npx sequelize-cli model:generate --name User --attributes firstName:string,lastName:string,email:string

fix lỗi



Sequelize sẽ chạy vào file “.env” để lấy 1 biến NODE\_ENV để cho phép chạy vào development

Kết quả



# Sequelize

Sequelize là một package hỗ trợ sử dụng ORM với môi trường Node.JS . Trong video này, chúng ta sẽ cùng nhau tìm hiểu cách kết nối tới database với Sequelize, đồng thời học cách lấy dữ liệu từ database để render ra view.

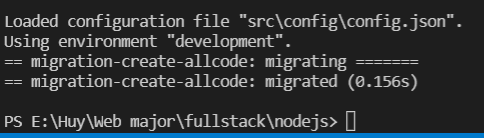
<https://sequelize.org/master/manual/getting-started.html>

## Cơ chế vận hành sequelize quản lí database

Có file migrations gốc để build database schema cho version gốc. Các versions sau phải tạo 1 file mới => Tránh làm hỏng dữ liệu gốc.

Running Migrations

npx sequelize-cli db:migrate



* Sau mỗi lần update migration bảng mới

## Tạo seeder

npx sequelize-cli seed:generate --name demo-user

## Chỉnh sửa seeder

return queryInterface.bulkInsert('Users', [{

firstName: 'John',

lastName: 'Doe',

email: 'example@example.com',

createdAt: new Date(),

updatedAt: new Date()

}]);

## Running seeder

npx sequelize-cli db:seed:all

## Tạo file connectDB.js ở folder config

const { Sequelize } = require('sequelize');

// Option 3: Passing parameters separately (other dialects)

const sequelize = new Sequelize('booking\_healthcare', 'root', null , {

  host: 'localhost',

  dialect: 'mysql'

});

let connectDB = async() => {

    try {

        await sequelize.authenticate();

        console.log('Connection has been established successfully.');

      } catch (error) {

        console.error('Unable to connect to the database:', error);

      }

}

module.exports = connectDB;

* **Biến sequelize để connect vs database**
* **Biến connect DB để check connect**

## Trả biến **connect DB** tới server.js

### 

## API

Application Programming Interface

## Cách sử dụng nhiều phiên bản Node.JS song song

1. Unistall node.js
2. Install nvm (1.1.7)

<https://github.com/coreybutler/nvm-windows/releases>

1. Đọc readme
2. Nvm list (kiểm tra các nodejs)
3. Install node.js( các versions của nodejs muốn cài đặt) bằng nvm install 11.15.7
4. Nvm use 11.15.0 để chuyển version

## Cách cài project REACT.JS

1. ở file .gitignore

để không leak password thì điền phần #misc là .env để loại không upload file .env

1. terminal/ npm install

Lưu ý: tạo biến cho react-app thì cần để tên là 

1. Redux giúp lưu thông tin xuyên suốt project(isLoggedin:false)

Căn giữa bằng position=absolute:

        width: 400px;

        border-radius: 10px;

        background-color: #fff;

        position: absolute;

        margin: auto;

        top: 0;

        bottom: 0;

        right: 0;

        left: 0;

        height: 450px;

props là 1 biến lưu trữ giá trị xuyên suốt chương trình (lắng nghe thay đồi)

bắt đầu 1 hàm thì dùng {()}.

onClick={() => {this.handleLogin()}}

Absolute không làm ảnh hưởng tới elements khác

            .login-input{

                margin: 7px 0;

                .custom-input-password{

                    position: relative;

                    i{

                        position: absolute;

                        right: 5px;

                        top: 10px;

                    }

                }

            }

# Sequelize findAll exclude field

# Life cycle

 Run component:

    1. Run constructor -> init state

    2. Did mount (gán giá trị cho biến state). Vì để lấy dữ liệu từ API để set state cho component

    3. Render

# Sử dụng query param với GET

Cách truyền id vào để lấy dữ liệu user từ backend

* Giải pháp: cần truyền lên param

Backend

    let id = req.query.id; //All, id

Frontend

    return axios.get(`/api/get-all-user?id=${inputId}`)

# SetState-cập nhật State- hàm bất đồng bộ

        let response = await getAllUsers('ALL');

        if(response && response.errCode ===0) {

            this.setState({

                arrUser: response.users

            },() => {

            console.log('check state user',this.state.arrUser);// hàm callback để lấy ra giá trị

            })// cập nhật state. Hàm bất đồng bộ

            console.log('check state user 1',this.state.arrUser);//

        }

        console.log('get user from node.js: ', response)

# Map khác gì for

arrUsers && arrUsers.map ((item,index) => {

                                console.log('checkmap',item,index)

                                return (

                                    <tr>

                                        <td>{item.email}</td>

                                        <td>{item.firstName}</td>

                                        <td>{item.lastName}</td>

                                        <td>{item.address}</td>

                                        <td>

                                            <button className= "btn-edit"><i className="fas fa-pencil-alt"></i></button>

                                            <button className= "btn-delete"><i className="fas fa-trash"></i></button>

                                        </td>

                                    </tr>

                                    //<>: fragments. React cần render ra khối

                                )

                            })

* Map sử dụng item và index để lặp vòng

# Reset Button CSS

width: 50px;

      border: none;

      outline: none;

      background: transparent;

      color: orange;

# Sử dụng arrow function cho handle event

 handleAddNewUser = () => {

        alert('click me')

    }

<button className="btn btn-primary px-3"

                    onClick= {()=> this.handleAddNewUser()}>

                    <i className="fas fa-plus"></i>

                    Add new users</button>

# Hackerthemes

Nên review

# Tạo modal bằng thư viện bằng reactstrap

Tìm trong file “package.json” tìm ra reactstrap

return (

            <Modal isOpen={true} toggle={()=>{this.toggle()}} className={'abcClassName'}>

            <ModalHeader toggle={()=>{this.toggle()}} >Create a user</ModalHeader>

            <ModalBody>

                Lorem ipsum dolor sit amet, consectetur adipiscing elit

            </ModalBody>

            <ModalFooter>

                <Button color="primary" onClick={()=>{this.toggle()}}>Do st</Button> {''}

                <Button color="primary" onClick={()=>{this.toggle()}}></Button>

            </ModalFooter>

            </Modal>

        )

# Properties

Component cha và con => nested component

Cha muốn truyền dữ liệu state tới con thì cần dùng properties để nhận

Props của con => chính là state của cha .VD: {this.state.isOpenModalUser}

This.props.isOpen để truy cập vào 1 biến cụ thể

# Life cycle của React

    Run component:

    1. Run constructor -> init state

    2. Did mount (gán giá trị cho biến state). Vì để lấy dữ liệu từ API để set state cho component

    3. Render

# Cha toggle truyền tới con bằng function

toggleUserModal = () => {

        this.setState({

            isOpenModalUser: !this.state.isOpenModalUser,

        })

    }

<ModalUser

          isOpen = {this.state.isOpenModalUser}

         toggleFromParent = {toggleUserModal}

         test={'abc'}

 />

Truyền tới con bằng attribute.

toggle = () => {

        this.props.toggleFromParent();

    }

# CSS class ngang hàng bằng &.

    .input-container {

      display: flex;

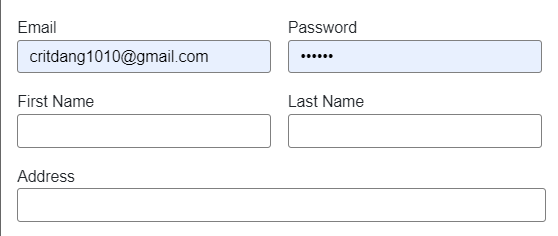
      flex-direction: column;

      width: 48%;

      &.max-width-input {

        width: 100%;

      }



Address 100% và các input khác không có class “max-width-input” thì chỉ 50%.

State để quản lí giá trị biến trong vòng đời của 1 component.

# Onchange

Truyền event vào để gửi tới hàm handleOnChangeInput

<input type="email" onChange={(event) => {this.handleOnChangeInput(event)}}/>

Code để trả về console

    handleOnChangeInput = (event) => {

        console.log(event.target.value)

    }

**WARNING: 5 trường cần onchange thì viết hàm onchange => Không cần thiết**

    handleOnChangeInput = (event,id) => {

        console.log(event.target.value,id)

    }

* **Truyền vào id để biết đang onChange hàm nào**

# SetState: Sửa State trực tiếp

Lỗi: dữ liệu render bất thường

**Sử dụng để lưu trữ giá trị vào từng trường**

**Code tệ**

handleOnChangeInput = (event,id) => {

        // bad code. modify state

        this.state[id] = event.target.value;

        this.setState({

            ...this.state

        },() => {

            console.log('check bad',this.state)

        })

        console.log(event.target.value,id)

    }

        this.state = {

            email: '',

            password: '',

        }

        this.state.email === this.state['email'] bản chất object là mảng

**Code hay**

        // good code

        let copyState = {...this.state};//{} tượng trưng 1 object

        copyState[id] = event.target.value;

        this.setState({

            ...copyState

        })

        console.log('copy state',copyState);

        console.log(event.target.value,id)

**Không nên modify trực tiếp vào vì this.state ở trường hợp trên sẽ bị gọi lại**

# Check valid tài khoản – giải pháp thay thế nhiều vòng lặp để check

Tạo nút gọi tới handleAddNewUser

 onClick={()=>{this.handleAddNewUser()}}>Add new</Button> {''}

Hàm xử lí:

    checkValideInput = () => {

        let isValid = true;

        let arrInput = ['email', 'password','firstName','lastName','address'];

        for(let i = 0; i < arrInput.length; i++) {

            if(!this.state[arrInput[i]]){

                isValid = false;

                alert('Missing parameter: ' +arrInput[i]);

                break;

            }

        }

        return isValid;

    }

    handleAddNewUser = () => {

        let isValid = this.checkValideInput();

        if(isValid === true) {

            //call API create modal

            // truyền data của new user sang cho cha -UserManage

            this.props.createNewUser(this.state);

        }

    }

Hàm createNewUser được gọi thông qua props và được truyền state – từ hàm con .

Hàm createNewUser ở hàm cha

    createNewUser = async (data) => {

        try{

             let response = await createNewUserService(data);

             console.log(response);

             if(response && response.errCode !== 0) {

                alert(response.errMessage)

             }else {

                 //gọi lại API

                await this.getAllUsersFromReact();

                //tắt modal sau khi reload bằng gọi API bên trên

                this.setState({

                    isOpenModalUser: false

                })

             }

        }catch(e){

            console.log(e)

        }

    }

Truyền hàm createNewUser tới con bằng props từ cha

<ModalUser

                    isOpen = {this.state.isOpenModalUser}

                    toggleFromParent = {this.toggleUserModal}

                    createNewUser = {this.createNewUser}

                    //this.createNewUser không để "()" vì khi truyền function thì hàm cha không biết dữ liệu con gửi tới là gì

                />

Gọi lại API để update thông tin mới bằng getAllUsersFromReact()

    getAllUsersFromReact = async () => {

        let response = await getAllUsers('ALL');

        if(response && response.errCode === 0) {

            this.setState({

                arrUsers: response.users

            })

        }

    }

# Xóa dữ liệu người dùng

Tạo nút delete

<button className= "btn-delete" onClick={() => this.handleDeleteUser(item)} ><i className="fas fa-trash"></i></button>

//truyền item vào vì item là từng object của arrUsers

Hàm xử lí handleDeleteUser

    handleDeleteUser = async (user) => {

        console.log('click delete',user)

        try {

            let res = await deleteUserService(user.id)

            if(res && res.errCode === 0) {

                await this.getAllUsersFromReact();

            }else{

                alert(res.errMessage)

            }

        }catch(e){

            console.log(e)

        }

    }

Hàm deleteUserService từ userService

const deleteUserService = (userId) => {

    return axios.delete('/api/delete-user', {

        data:{

        id: userId

    }

    });

}

Nếu res không lỗi thì update lại giao diện tức thời

    getAllUsersFromReact = async () => {

        let response = await getAllUsers('ALL');

        if(response && response.errCode === 0) {

            this.setState({

                arrUsers: response.users

            })

        }

    }

# Delete và xóa dữ liệu sau khi nhập modal

Fire event:

Child -> parent(props)

Parent -> child(ref)

* Giải pháp khác: emitter. Giải quyết được cả 2

**Truyền data bằng emitter từ cha tới con**

Hàm cha

emitter.emit('EVENT\_CLEAR\_MODAL\_DATA', {'id': 'your id'})

Hàm con nhận dữ liệu

    listenToEmitter() {

        emitter.on('EVENT\_CLEAR\_MODAL\_DATA', data => {

            console.log('listen emitter from parent:' ,data)

        })

    }

**Áp dụng vào việc xóa dữ liệu modal sau khi nhập xong**

1. **Khai báo emitter**

import EventEmitter from 'events';

const \_emitter = new EventEmitter();

\_emitter.setMaxListeners(0) //unlimit listener

export const emitter = \_emitter;

1. **Import emitter vào cha**

import { emitter } from "../../utils/emitter";

1. **Fire sự kiện ở cha**

emitter.emit('EVENT\_CLEAR\_MODAL\_DATA')

1. **Lắng nghe sự kiện ở con**

constructor(props) {

        super(props);

        this.state = {

            email: '',

            password: '',

            firstName: '',

            lastName: '',

            address: ''

        }

        this.listenToEmitter()

    }

    listenToEmitter() {

        emitter.on('EVENT\_CLEAR\_MODAL\_DATA', () => {

            //reset state

            this.setState({

                email: '',

                password: '',

                firstName: '',

                lastName: '',

                address: ''

            })

        })

    }

# Toggle modal

Gán vào hàm kích hoạt

 <ModalEditUser

                    isOpen = {this.state.isOpenModalEditUser}

                    toggleUserEditModal = {this.toggleUserEditModal}

                    // createNewUser = {this.createNewUser}

                />

Hàm xử lí

    toggleUserEditModal = () => {

        this.setState({

            isOpenModalUser: !this.state.isOpenModalUser,

        })

    }

# Component đã mount nhưng dữ liệu chưa cập nhật

                <ModalEditUser

                    isOpen = {this.state.isOpenModalEditUser}

                    toggleFromParent = {this.toggleUserEditModal}

                    currentUser = {this.state.userEdit}

                    //truyền qua cho con

                    // createNewUser = {this.createNewUser}

                />

                }

            //Khi ấn vào component thì mới được mount chứ không phải từ lúc HTML

# Update User

Muốn biến thay đổi thì gán vào state

                    isOpen = {this.state.isOpenModalEditUser}

# Fix bug CORS

Client -> call api(route) -> controller

Client -> call api(route) -> middleware -> controller

import cors from 'cors';

app.use(cors({ origin: true}));

Fix bằng cách thêm middlewares

// Add headers before the routes are defined

app.use(function (req, res, next) {

    // Website you wish to allow to connect

    res.setHeader('Access-Control-Allow-Origin', 'http://localhost:3001');

    // Request methods you wish to allow

    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH, DELETE');

    // Request headers you wish to allow

    res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');

    // Set to true if you need the website to include cookies in the requests sent

    // to the API (e.g. in case you use sessions)

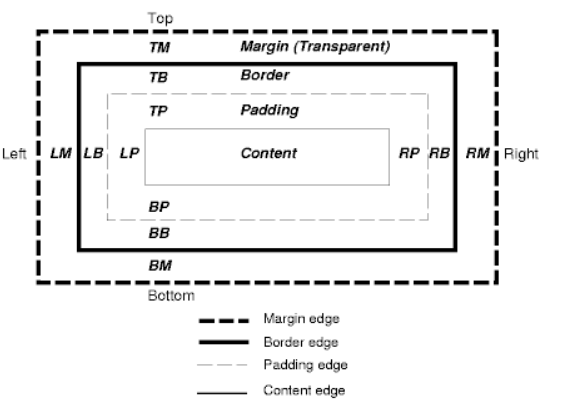
    res.setHeader('Access-Control-Allow-Credentials', true);

    // Pass to next layer of middleware

    next();

});

# Difference between padding and margin



Margin is applied to the **outside** of your element hence affecting how far your element is away from other elements.  
Padding is applied to the **inside** of your element hence affecting how far your element's content is away from the border.

# Width: fit-content

The **fit-content** behaves as fit-content(stretch). In practice this means that the box will use the available space, but never more than [max-content](https://developer.mozilla.org/en-US/docs/Web/CSS/max-content).

## [Examples](https://developer.mozilla.org/en-US/docs/Web/CSS/fit-content#examples)

### [Using fit-content for box sizing](https://developer.mozilla.org/en-US/docs/Web/CSS/fit-content#using_fit-content_for_box_sizing)

#### HTML

<div class="container">

<div class="item">Item</div>

<div class="item">Item with more text in it.</div>

<div class="item">Item with more text in it, hopefully we have added enough text so the text will start to wrap.</div>

</div>

Copy to Clipboard

#### CSS

.container {

border: 2px solid #ccc;

padding: 10px;

width: 20em;

}

.item {

width: -moz-fit-content;

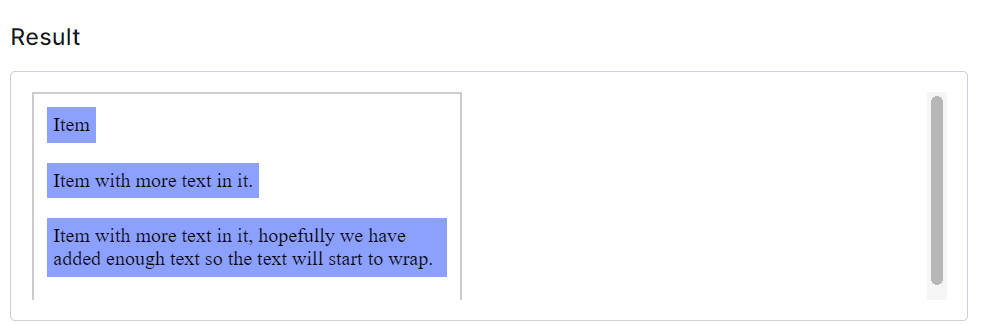
width: fit-content;

background-color: #8ca0ff;

padding: 5px;

margin-bottom: 1em;

}



* Class “Item” không vượt quá “container” ( max-content)

# Chỉnh mờ ảnh

    background-image: url('../../assets/header-background.jpg'), linear-gradient(rgba(0,0,0,0.25),rgba(255,255,255,0.1));

# Dấu “&” trong CSS

            .language-vi{

                cursor: pointer;

                &.actice{

                    color:orange;

                }

            }

.active và language-vi ngang hàng

# Redux là gì

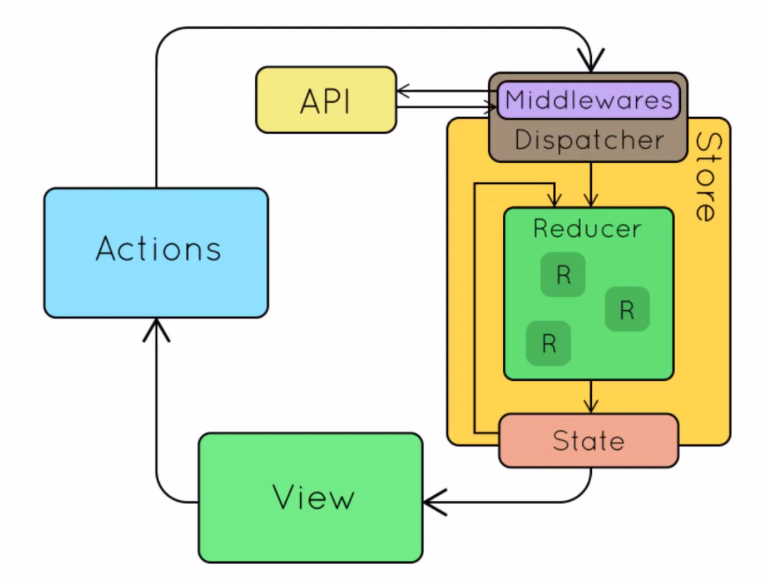
Giống như local storage để lưu biến state của react mọi nơi để tránh gọi lại API.

Với Redux, state của ứng dụng được giữ trong một nơi gọi là store và mỗi componentđều có thể access bất kỳ state nào mà chúng muốn từ chúng store này.

Redux chạy song song react

**Tại sao không lưu ở local storage**

Không an toàn và hiệu năng redux tốt hơn



*Bố cục redux*

**Các hoạt động khi chương trình khởi động:**

Index.js

import { Provider } from 'react-redux';

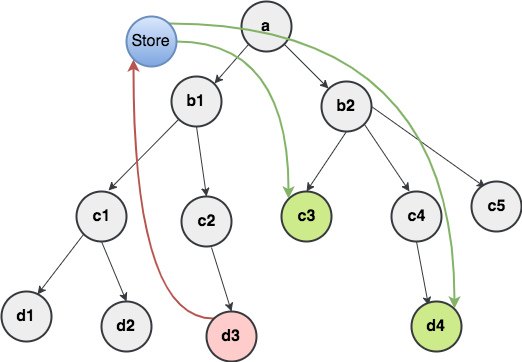
import reduxStore, { persistor } from './redux';

persistor: giúp lưu biến như ở localStorage

Sau đó chạy tới redux và có file store/actions(fire event)

File reducer lấy data và map vào state rồi state truyền sang view

## **Hiểu cách Redux làm việc**



*Cái cách mà Redux hoạt động là khá đơn giản. Nó có 1 store lưu trữ toàn bộ state của app. Mỗi component có thể access trực tiếp đến state được lưu trữ thay vì phải send drop down props từ component này đến component khác.*

Có 3 thành phần của Redux: Actions, Store, Reducers.

### ***1. Actions***

Actions đơn giản là các events. Chúng là cách mà chúng ta send data từ app đến Redux store. Những data này có thể là từ sự tương tác của user vs app, API calls hoặc cũng có thể là từ form submission.

*Actions được gửi bằng cách sử dụng store.dispatch() method, chúng phải có một type property để biểu lộ loại action để thực hiện. Chúng cũng phải có một payload chứa thông tin. Actions được tạo thông qua một action creator.*

const setLoginStatus = (name, password) => {

return {

type: "LOGIN",

payload: {

username: "foo",

password: "bar"

}

}

}

### ***2. Reducers***

Reducers là các function nguyên thủy chúng lấy state hiện tại của app, thực hiện một action và trả về một state mới. Những states này được lưu như những objects và chúng định rõ cách state của một ứng dụng thay đổi trong việc phản hồi một action được gửi đến store.

Đây là một ví dụ về cách mà Reducers hoạt động trong Redux:

const LoginComponent = (state = initialState, action) => {

switch (action.type) {

case "LOGIN":

return state.map(user => {

if (user.username !== action.username) {

return user;

}

if (user.password == action.password) {

return {

...user,

login\_status: "LOGGED IN"

}

}

});

default:

return state;

}

};

### ***3. Store***

Store lưu trạng thái ứng dụng và nó là duy nhất trong bất kỳ một ứng dụng Redux nào. Bạn có thể access các state được lưu, update state, và đăng ký or hủy đăng ký các listeners thông qua helper methods.

Tạo một store cho một login app:

const store = createStore(LoginComponent);

Các actions thực hiện trên một state luôn luôn trả về một state mới. Vì vậy, state này là đơn giản và dễ đoán.

Bây giờ, chúng ta đã biết hơn một chúng về Redux, hãy trở lại với ví dụ Login component và xem cách cách mà Redux có thể giúp chúng ta được gì.

class App extends React.Component {

render() {

return (

<div>

<Status user={this.props.user.name}/>

<Login login={this.props.setLoginStatus}/>

</div>

)

}

}

**Sử dụng vào dự án**

Khi setup đúng thì sẽ có folder “store” chứa 2 folder “actions” và “reducers”

ở folder actions

actionType.js – các loại action được khai báo.

    //user

    ADD\_USER\_SUCCESS: 'ADD\_USER\_SUCCESS',

userActions.js – export dữ liệu của action ra

export const addUserSuccess = () => ({

    type: actionTypes.ADD\_USER\_SUCCESS

})

ở folder reducers:

userReducer.js – xử lý action tới props

    switch (action.type) {

        case actionTypes.USER\_LOGIN\_SUCCESS:

            return {

                ...state,

                isLoggedIn: true,

                userInfo: action.userInfo

            }

* **userReducer giúp fire actions**

# Tích hợp thư viện React-Slider-Carousel

Thư viện react-slick:

npm install --save react-slick@0.28.1

<https://www.npmjs.com/package/react-slick>

Thư viện slick-carousel để lấy css :

npm install --save [slick-carousel@1.8.1](mailto:slick-carousel@1.8.1)

<https://www.npmjs.com/package/slick-carousel>

1. Thêm component Specialty vào Page

        return (

            <div>

                <HomeHeader />

                <Specialty />

            </div>

        );

1. Cài đặt vào file “Specialty”

import Slider from 'react-slick';

// Import css files

import "slick-carousel/slick/slick.css";

import "slick-carousel/slick/slick-theme.css";

# Customize tùy chỉnh React-Slider-Carousel

**Xem lại video**

# [Bug] Thuộc tính background

Background trong css là viết tắt của background-position, background-repeat, background-size,background-color vì vậy lưu ý nếu không sẽ bị ghi đè

# Kích thước ảnh

Sửa kích thước con chứ không nên sửa kích thước cha bị bóp lại

# Tạo Sticky Header

Lỗi: background ăn vào nền của Header Menu

Vì background mặc định của sticky là transparent

.home-header-container {

    position: sticky;

    top: 0;

    background-color: white;

    z-index: 10000;

# [Bug] sử dụng sql trong xampp

Khi dùng insert và có keys thì cần có bảng để nhận diện được nơi muốn thêm data

INSERT INTO ALLCODES (ALLCODES.type,ALLCODES.key, ALLCODES.valueEn, ALLCODES.valueVi) values ('TIME','T5','1:00 PM - 2:00 PM', '13:00 - 14:00' );