**REPORT 9**

ReactJS (Tiếp)

Học viên: Hà Thị Phương Loan – FE104

1. **Khởi tạo dự án React với Create React App**

**1.1 Cài đặt NodeJS và npm**

- Để cài đặt môi trường chạy React trước tiên cần phải cài đặt NodeJS và npm. Đây là một nền tảng bắt buộc.

- Để kiểm tra NodeJS và npm đã được cài đặt chưa, chúng ta có thể mở terminal và gõ dòng lệnh:

**node –v**

**npm –v**

Khi cài đặt thành công sẽ hiển thị ra phiên bản NodeJS và npm hiện đã cài đặt.

**1.2 Khởi tạo ReactJS App**

**Bước 1**: Vào thư mục chứa dự án và mở terminal gõ dòng lệnh:

npx create-react-app my-app

Trong đó my-app là tên thư mục chứa dự án của bạn. Giả sử muốn khởi tạo dự án có tên là first-project, chúng ta sẽ có:

npx create-react-app first-project

**Bước 2**: Sau khi quá trình khởi tạo hoàn tất, vào thư mục vừa được khởi tạo, chúng ta sẽ thấy có các thư mục được khởi tạo tự động:

* src: chứa những đoạn mã mà chúng ta viết sau này: component, page, router,…
* public: chứa các file ảnh,… và chứa file index.html
* File App.js là component chính của app
* node\_module: các module cài tự động khi tạo react app
* package.json và package-lock.json: chứa thông tin của các module cần thiết

Ngoài ra chúng ta có thể tạo thêm những thư mục cần thiết theo mục đích sử dụng. Ví dụ:

* Assets: Nơi chứa hình ảnh, font chữ, css, scss,…
* Components: nơi chứa từng các thành phần nhỏ trong trang web: navbar, banner, listItem,…
* Pages: Nơi chứa các trang trong trang web như HomePage, LoginPage, … tập hợp từ 1 hoặc nhiều components lại
* fetchApi: Nơi chứa các hàm để call api từ phía server
* Routers.js: Nơi khai báo các đường dẫn của trang web ứng với Page nào
* Db.json: File database cho Json-server

….

**Bước 3**: Khởi chạy dự án

Để khởi chạy dự án chúng ta phải truy cập vào thư mục vừa được khởi tạo và mở terminal gõ dòng lệnh:

npm start

Lúc này ReactJS sẽ khởi chạy dự án mặc định

**Bước 4**: Xây dựng ứng dụng ReactJS của bạn

Chú ý:

* Trong file index.js có dòng lệnh document.getElementById(‘root’). Đây là vị trí tất cả các component sẽ render ra. Chúng ta có thể tìm id root trong file public/index.html
* Khi khởi chạy dự án thành công, chúng ta sửa đổi các file thì ReactJS sẽ tự động reload lại sự thay đổi đó. Chúng ta không cần khởi động lại server bằng cách thủ công nữa.

1. **SCSS**

* Khái niệm: SCSS là một chương trình tiền xử lý CSS (CSS preprocessor) giúp chúng ta viết ra những đoạn mã CSS nhanh hơn và có cấu trúc rõ ràng hơn.
* SCSS sử dụng cú pháp giống với Ruby có phần mở rộng là .scss
* Các tính năng cơ bản của SCSS:
  1. **Xếp chồng (Nested Rules)**

VD:

* Với code css thường:

.navbar ul.menu {

list-style: none;

}

.navbar ul.menu li {

padding: 3px;

}

.navbar ul.menu li a {

text-decoration: none;

}

* Viết lại với SCSS sử dụng tính năng xếp chồng:

.navbar {

ul.menu {

list-style: none;

li {

padding: 3px;

a {

text-decoration: none;

}

}

}

}

* 1. **Biến (variable)**

Để sử dụng biến với SCSS, bạn chỉ cần đặt tên cho biến – bắt đầu bằng $. Biến chứa đựng các giá trị mà chúng ta dùng nhiều lần ví dụ như mã màu, font hay kiểu chữ.

VD:

$RedColor = #fff;

.navbar {

ul.menu {

list-style: none;

li {

padding: 3px;

a {

text-decoration: none;

color: $Redcolor

}

}

}

}

* 1. **Quy tắc Mixin**

Mixin giúp tạo các hàm được sử dụng trong SCSS, bạn hoàn toàn có thể truyền các tham số vào bên trong nó để sử dụng. Công dụng của nó là mang nhiều thuộc tính mà bạn đã quy ước trong một mix nào đó rồi @include vào một thành phần bất kỳ mà không cần phải viết lại các thuộc tính đó.

VD:

@mixin colorVsStyle {

color: #f06;

font-style: italic;

}

.navbar {

ul.menu {

list-style: none;

li {

padding: 5px;

a {

text-decoration: none;

@include colorVsStyle;

}

}

}

}

* 1. **Kế thừa (Extends)**

Đây là một tính năng quan trọng được sử dụng rất nhiều, cho phép kế thừa tất cả các thuộc tính của một class hoặc một cái gì mã chúng ta đã khai báo sẵn.

VD:

.error {

color: red;

}

.critical-error {

@extend .error;

bottom: 1px solid red;

font-weight: bold;

}

* 1. **Import**

Đối với SCSS, chúng ta có thể import các file .scss vào nhau để sử dụng.

VD:

* Ta có file \_header.scss để CSS riêng cho header.
* \_body.scss để CSS riêng cho body.
* \_footer.scss để CSS riêng cho footer.
* Tiếp theo tại file style.css ta chỉ cần @import 3 file trên để sử dụng. Cú pháp:

@import 'header';

@import 'body';

@import 'footer';

* 1. **Vòng lặp**

VD:

@for $i from 1 through 7 {

.col-#{$i} {

width: $i \* 2em;

}

}

$speackers: bob-banker, partty-plume, sandra-smith;

@each $speacker in $speakers {

.#{$speaker}-profile {

background-image: url('/img/#{$speaker}.png');

}

}

* 1. **Mệnh đề điều kiện IF**

VD:

$contrast: high;

body {

font-family: $text-font;

font-size: em(18px);

@if $contrast == high {

color: #000;

} @else if $contrast == low {

color: #999;

} @else {

color: red;

}

}

1. **React-router**

* Khái niệm: React Router là một thư viện cho việc điều hướng URL tiêu chuẩn trong React, cho phép chúng ta có thể đồng bộ UI với URL. React Router được thiết kế với API đơn giản cho phép giải quyết các vấn đề về URL một cách nhanh chóng.
* Để sử dụng React Router cần phải cài đặt thư viện này vào trong dự án React bằng cách sử dụng NPM:

**npm install react-router-dom**

* Sau khi cài đặt thành công, trong trường hợp cần dùng đến React Router bạn chỉ cần import nó component đó.

VD:

import {

BrowserRouter,

Routes,

Route

} from 'react-router-dom';

* Các thành phần trong react-router:
  1. **BrowserRouter vs HashRouter**
* React-Router cung cấp 2 thành phần hay sử dụng đó là BrowserRouter & HashRouter. Hai thành phần này khác nhau ở kiểu URL mà chúng sẽ tạo ra và đồng bộ.
* BrowserRouter: Được sử dụng phổ biến hơn, nó sử dụng History API có trong HTML5 để theo dõi lịch sử bộ định tuyến của bạn.
* HashRouter: Sử dụng hash của URL (window.location.hash) để ghi nhớ mọi thứ.
  1. **Route**
* Route: Định nghĩa một ánh xạ giữa một URL và một Component. Khi người dùng truy cập theo một URL trên trình duyệt, một Component tương ứng sẽ được render trên giao diện.

VD:

<Routes>

<Route path="/" element={<Home />} />

<Route path="profile/\*" element={<Profile />} />

</Routes>

Trong đó:

* path: Là đường dẫn trên URL.
* element: Là component sẽ đươc load ra tương ứng với Route đó.
  1. **Link**
* Cho phép chúng ta chuyển qua lại giữa các component thông qua URL, tương tự như thẻ a trong html.

VD: <Link to="/about">About</Link>

* 1. **Redirect**
* Chức năng dùng để chuyển trang.
* Có thể truy xuất thông tin trang trước đó thông qua đối tượng location

1. **Recoil**

* Khái niệm: Recoil chính là 1 thư viện dùng để quản lý state của React
* Ưu điểm:
* Tạo ra một công cụ quản lý state đơn giản, với cú pháp trực quan giống như state đơn thuần của react.
* Sử dụng các state riêng lẻ (gọi là các atom, phân biệt bằng key).
* Hỗ trợ luôn quản lý state bất đồng bộ (không cần middleware rườm rà như Redux)
* Cung cấp khả năng lưu trữ state lâu dài
* Các API chính của Recoil
  1. **Recoil Root**
* Là component bọc bên ngoài để có thể sử dụng Recoil bên trong. Nó như Provider trong redux, và không cần truyền vào store nào cả.
* Có thể có nhiều root cùng tồn tại, và khi nested thì root trong cùng sẽ có tác dụng.
  1. **Atom**
* Recoil lưu trữ dữ liệu theo các đơn vị gọi là atom, React component có thể truy cập tới dữ liệu chứa trong atom và thay đổi giá trị của dữ liệu
* atom yêu cầu 2 tham số là mã định danh duy nhất (unique key) và giá trị mặc định của dữ liệu.
  1. **Selectors**
* Selectors trả về dữ liệu đã được tính toán
* Selectors nhận vào hai tham số, mã định danh duy nhất và hàm get trả về dữ liệu được tính toán.
  1. **Hooks**
* Component sẽ tương tác với dữ liệu thông qua các hooks được cung cấp bởi recoil
* useRecoilState: giống useState của React, nhận vào recoil state, trả về giá trị của state cùng với hàm để thay đổi giá trị state.

VD: const [todoList, setTodoList] = useRecoilState(listTodoState);

* useRecoilValue: cũng nhận vào recoil state, nhưng khác là chỉ trả về giá trị của state - dùng để truy xuất giá trị state.

VD: const todoList = useRecoilValue(listTodoState);

* useSetRecoilState: giống useRecoilState, nhưng chỉ trả về hàm để thay đổi giá trị state.

VD: const setTodoList = useRecoilValue(listTodoState);

* useResetRecoilState: trả về hàm để reset state về giá trị default.

VD: const resetTodoList = useResetRecoilState(newListState);