

**TRƯỜNG ĐẠI HỌC GIA ĐỊNH**

**KHOA CÔNG NGHỆ THÔNG TIN**

**-------🙠🕮🙢-------**



**TIỂU LUẬN**

**ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM**

Họ tên : Lê Hoàng Chí Nhân

Lớp : K13DCPM01

MSSV : 1911040018

GVHD : Ts.Trần Hồng Quang

TPHCM,03-2022

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

….………………………………………………………………………….

Giảng viên hướng dẫn

**Mục Lục**

[I. Lời mở đầu 1](#_Toc15166)

[II. Nội dung 2](#_Toc29124)

[1 . JSX 2](#_Toc28285)

[2 . Component 3](#_Toc17574)

[3 . Props 4](#_Toc5537)

[4 . State 5](#_Toc16124)

[5 . Component life cycle 6](#_Toc27122)

[6 . Handle Event 9](#_Toc16433)

[7 . Virtual Dom 10](#_Toc4995)

[8 . Hook 13](#_Toc17677)

[8.1 Usestate 13](#_Toc25822)

[8.2 UseEffect 15](#_Toc851)

[8.3 UseContext 16](#_Toc7337)

[III. TEST APP REACT 19](#_Toc26422)

# Lời mở đầu

ReactJS là thư viện JavaScript được sử dụng để xây dựng các thành phần UI có thể tái sử dụng. Theo tài liệu chính thức của React, thì ReacJS được định nghĩa như sau :

 React là một thư viện để xây dựng giao diện người dùng. Nó khuyến khích việc tạo ra các thành phần UI có thể tái sử dụng, trình bày dữ liệu thay đổi theo thời gian. Rất nhiều người sử dụng React như V trong MVC. React trừu tượng hóa DOM , cung cấp một mô hình lập trình đơn giản hơn và hiệu suất tốt hơn. React cũng có thể kết xuất trên máy chủ bằng NodeJS và nó có thể cung cấp nguồn tài nguyên cho các ứng dụng gốc bằng React Native. React thực hiện luồng dữ liệu dữ liệu một chiều, làm giảm khối lượng truyền tải và dễ dàng hơn so với ràng buộc dữ liệu truyền thống.

Khi nhắc đến Reactjs thì người ta thường nói đến Single Page Application. Trong khi những framework khác cố gắng hướng đến mô hình MVC hoàn thiện thì React nổi bật với sự đơn giản và dễ dàng phối hợp với những thư viện Javascript khác. Nếu như Angular là 1 framework cho phép nhúng code javascript trong code html thông qua các attribute như ng-model, ng-repeat … thì react là 1 library cho phép nhúng code html trong code javascript nhờ vào JSX, bạn có thể dễ dàng lồng các đoạn HTML vào trong Javascript. Tích hợp giữa javascript và HTML vào trong JSX làm cho các component dễ hiểu hơn.

# Nội dung

## JSX

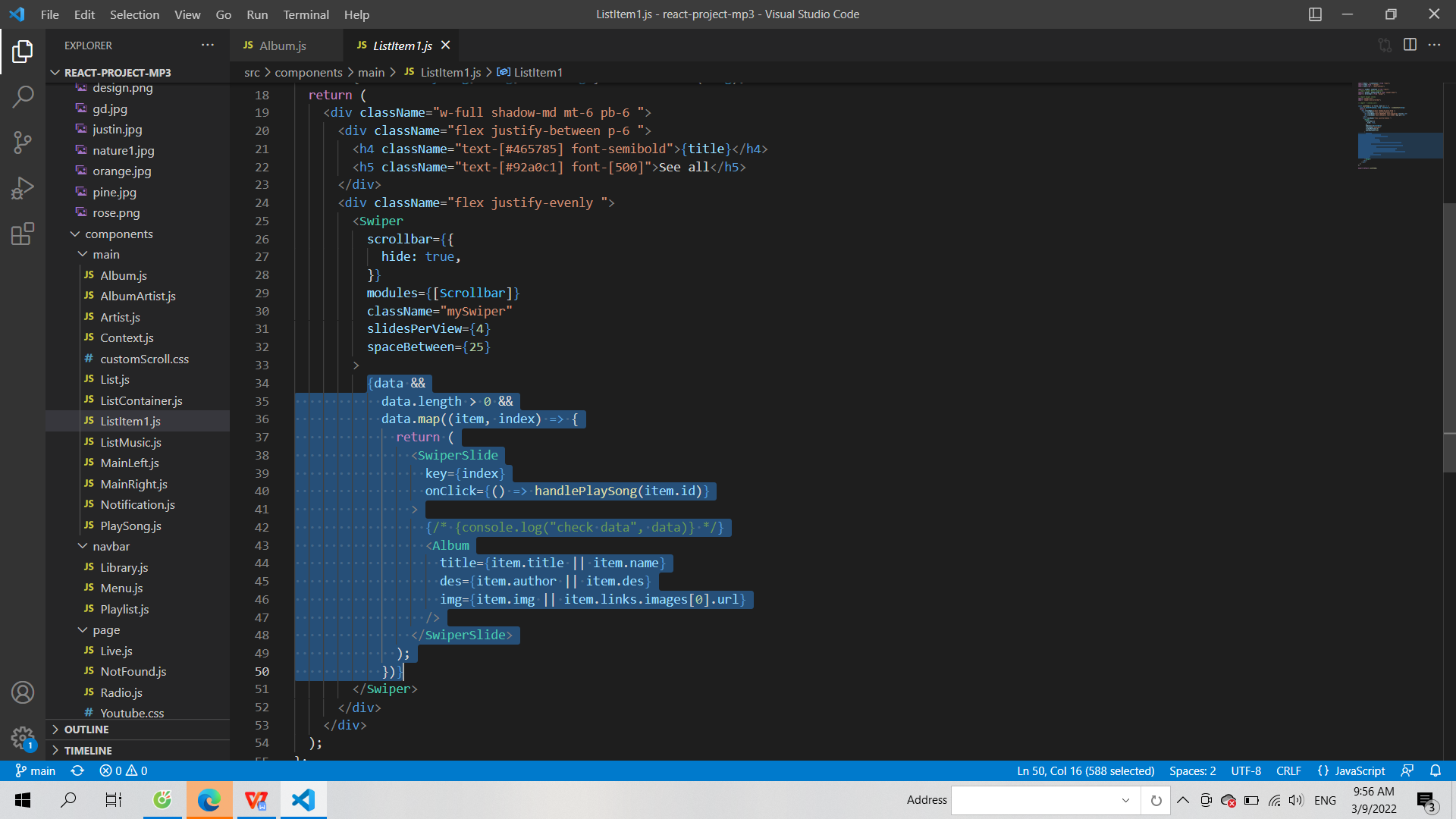
**JSX** là viết tắt là Javascript XML, một template languges nhưng nó lại mang hầu hết tính năng của Javascript. Nó cho phép bạn viết các đoạn mã HTML trong React một cách dẽ dàng và có cấu trúc hơn. React sử dụng JSX cho việc xây dựng bố cục thay vì javascript thông thường. JSX giúp tạo ra các React 'elements'. Việc sử dụng nó trong ReactJS rất hữu ích bởi:

JSX giúp cho việc xây dựng các ứng dụng React một cách nhanh hơn, dễ tối ưu trong việc complie code sang javascript.

JSX rất dễ xem các lỗi trong quá trình triển khai bởi hầu hết các lỗi sẽ được hiển thị trong quá trình compile, không như các đoạn mã HTML có thể thừa thiếu các thể div khiến giao diện bị hiển thị sai. JSX lại hoàn toàn ngược lại, khi bạn quên đóng div chẳng hạn thì nó lập tực sẽ hiển thị lỗi.

Quy ước đặt tên của JSX gần giống với HTML, React DOM sử dụng thuộc tính camelCase cho tên của thuộc tính cho phép chuyển đổi dễ hơn giữa HTML và JSX. Ví dụ trong HTML có thuộc tính class, JSX sẽ chuyển thành className, tabindex -> tabIndex.

Viết code js trong jsx bằng việc thêm {write javascript….}



Cú pháp khá giống với HTML nên dễ dàng cho việc viết chuyển đổi.

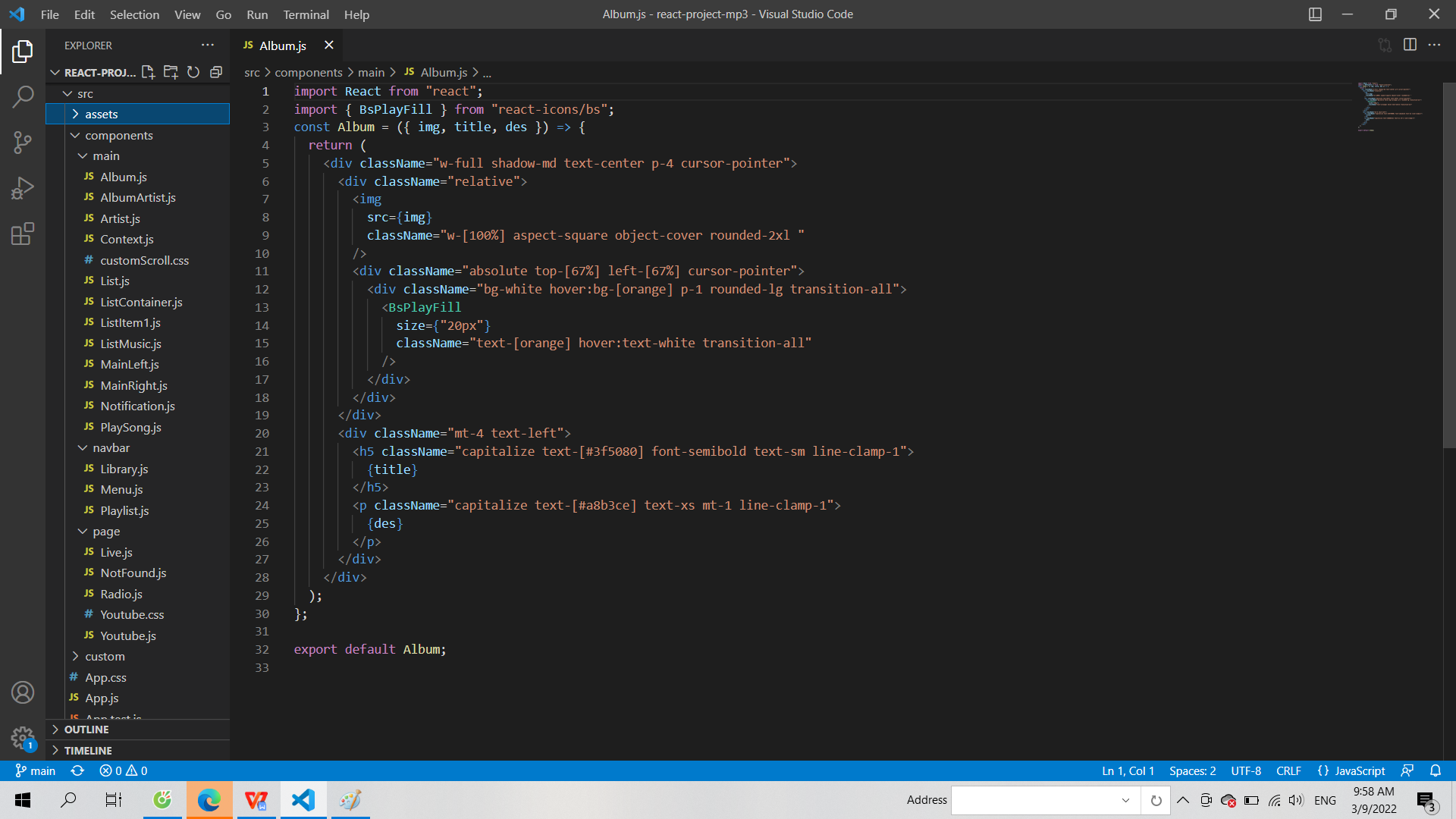
## Component

**Components** giúp phân chia các UI (giao diện người dùng) thành các phân nhỏ để dễ dàng quản lý và tái sử dụng. Giả sử mình có một website gồm nhiều phần bố cục khác nhau và mình muốn chia nhỏ các phần ra để dễ quản lý.

Trong mỗi React App đều có thể chứa rất nhiều components, mỗi **components** trong đó thường nhận về các props và trả về React elements từ đó hiển thị ra cho UI. **Components trong React** thường được viết theo 2 loại chính đó là functional component và class components. Bên dưới là một functional components:

const App = () =>  <h1>Hello Freetuts.net</h1>;

Hình ảnh setup foder components

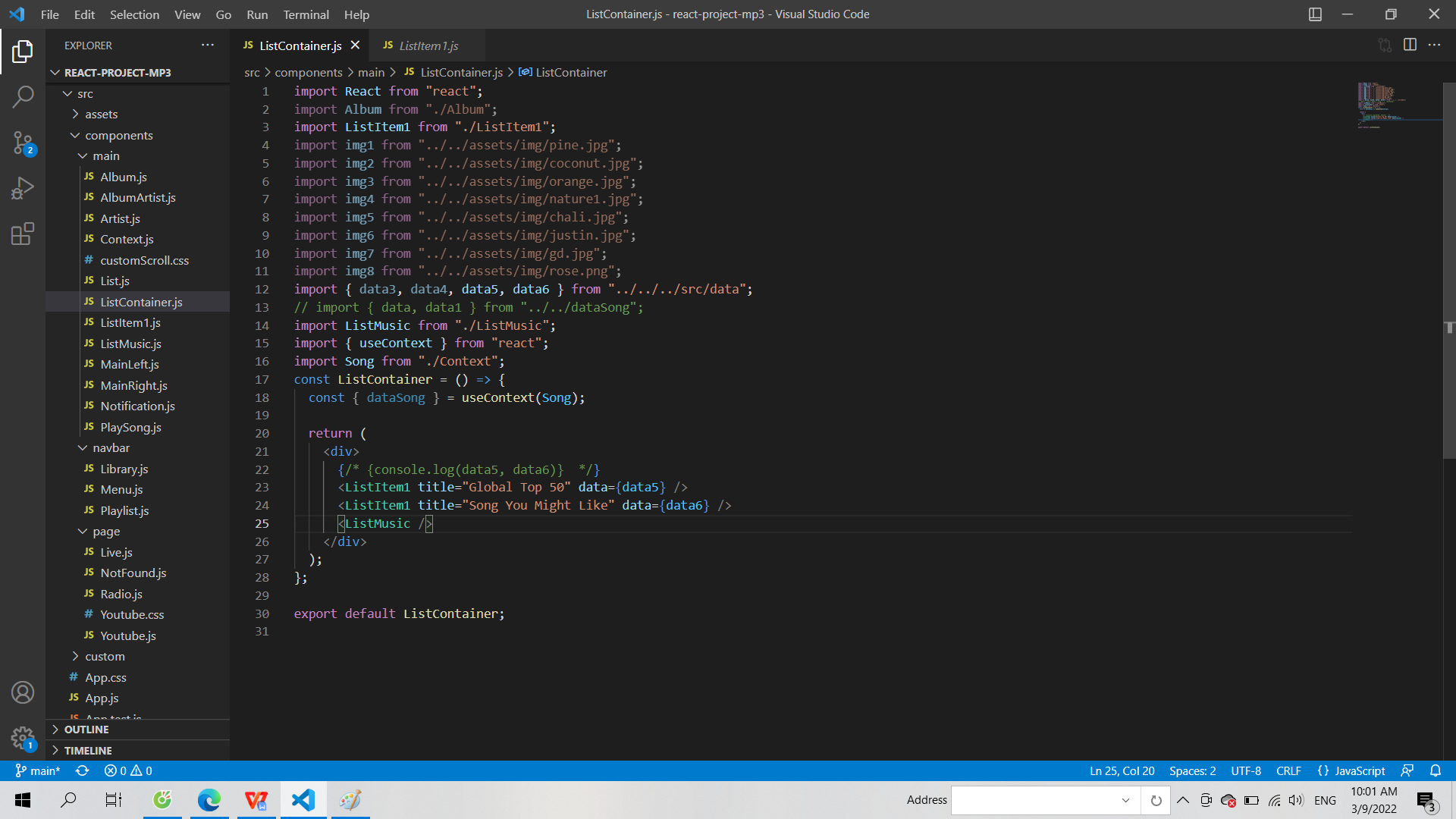


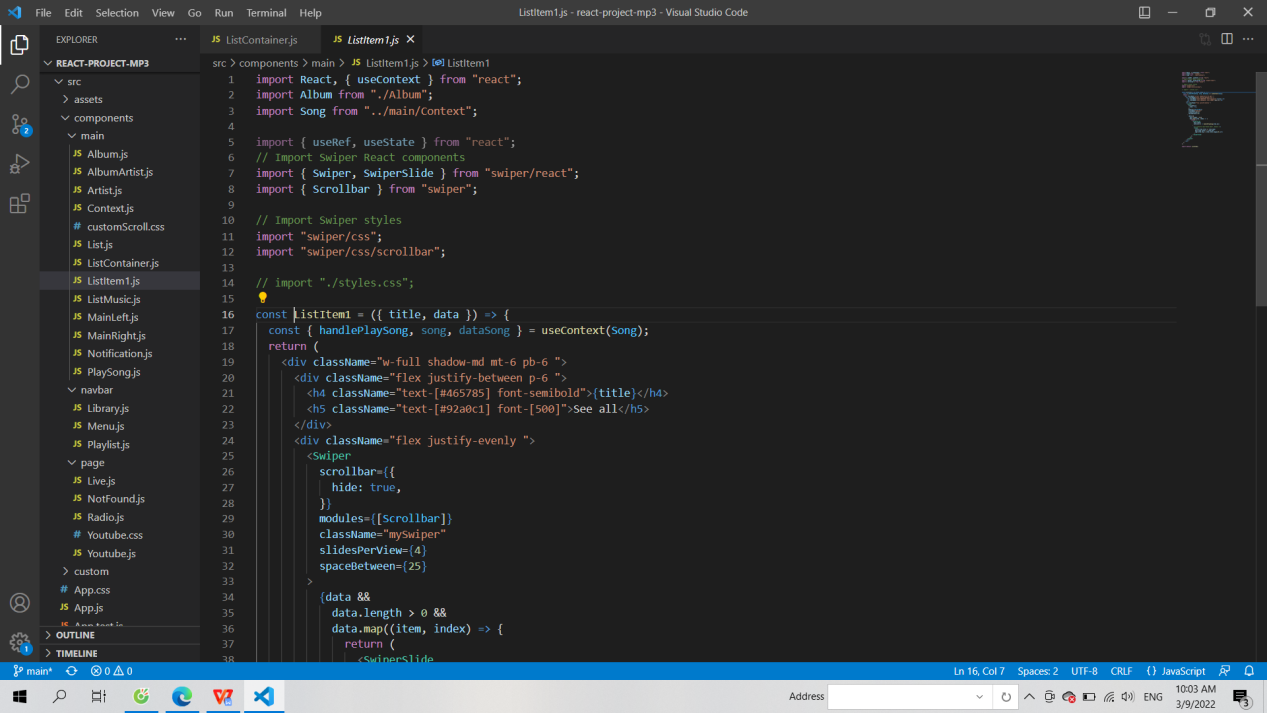
## 3.Props

**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.

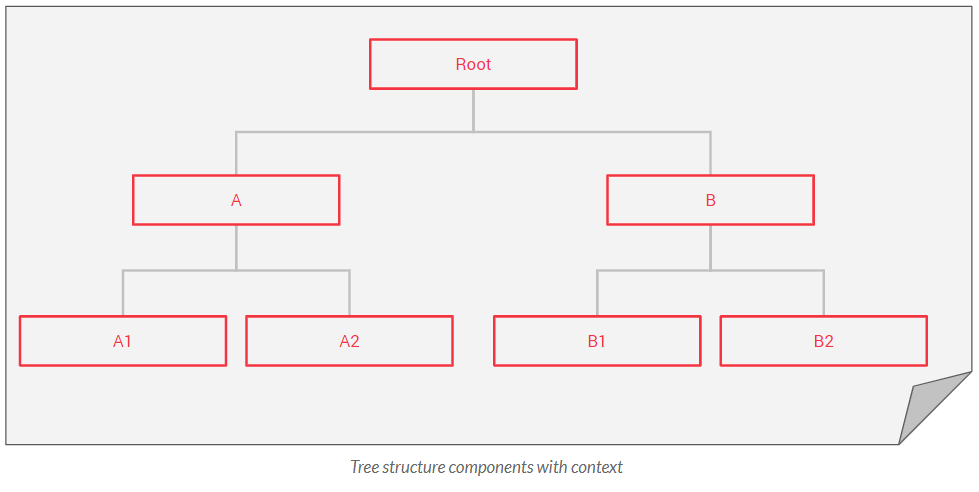
Khi một components cha truyền cho component con một props thì components con chỉ có thể đọc và không có quyền chỉnh sửa nó bên phía components cha.

Vì dụ minh họa





**-** Khi nhắc tới props phải nhắc đến props drilling :

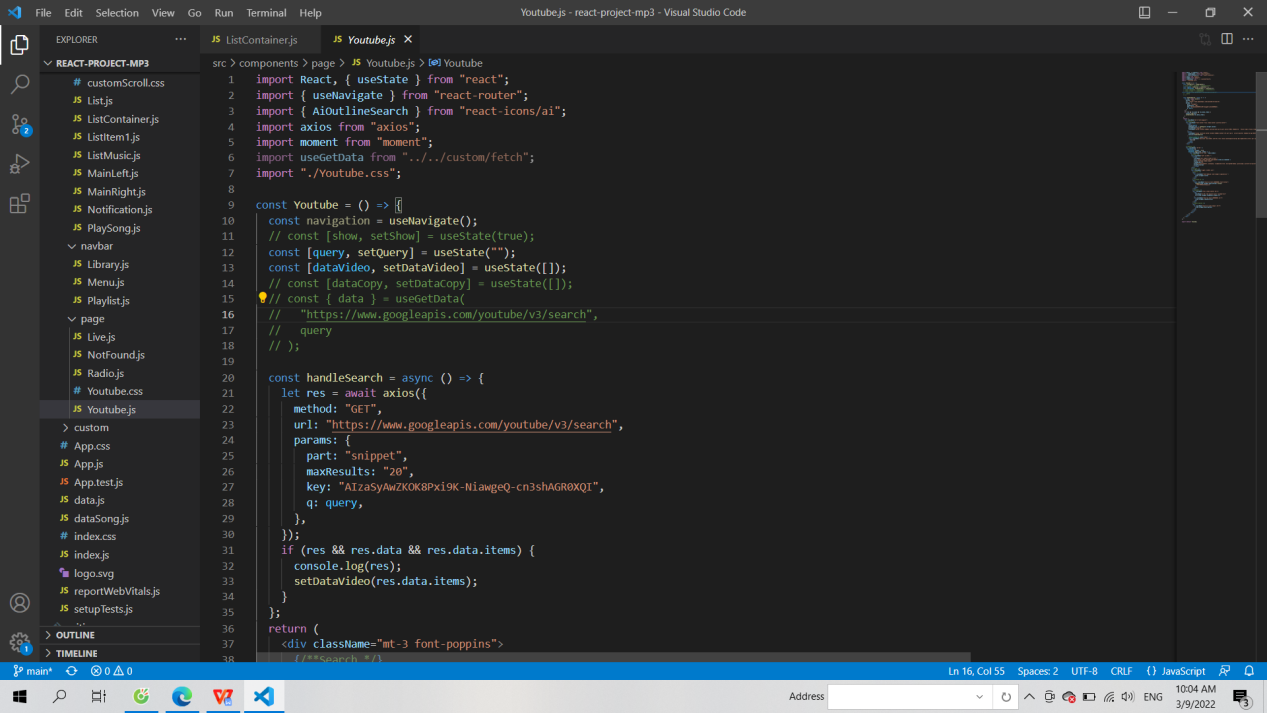


Theo chúng ta thấy component A là component cha truyền props qua B và trong component B lại có con là component C . Để component C nhận được dữ liệu phải từ A -> B -> C . Đó là props drilling . Giải pháp là chúng ta dùng Context Api hoặc Redux

## 4 . State

**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 bạn 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.

Vì dụ minh họa



Sự khác nhau giữa state và props

**State -**Dữ liệu chỉ nằm trong phạm vi của một component. Nó được sở hữu bởi một components cụ thể mà chỉ là của component đó thôi. Ví dụ, như người yêu bạn chỉ là của bạn vậy =))). Và mỗi khi state thay đổi thì component cũng phải thay đổi theo.

**Props -**Dữ liệu đường truyền từ component cha cho componet con, components con khi nhận được sẽ chỉ được đọc mà không thể thay đổi dữ liệu đó.

Sự khác nhau chính của 2 khái niệm này là component sở hữu dữ liệu. State là chỉ riêng nó có thể sử dụng. Props là dữ liệu mà component con được nhận về từ một component cha.

## 5 . Component life cycle

Chúng ta có thể thấy được mọi thứ trong thế giới đều hoạt động theo một chu kì (ví dụ như con người và cây cối). Cây được mọc lên, sẽ phát triển rồi đến một khoảng thời gian nào đó là sẽ chết đi. Trong React Component cũng vậy, một chu kì cũng xuất hiện, components được khởi tạo (hiển thị ra DOM), update, và kết thúc (unmount),..đó được gọi là một component life cycle.

React cho phép chúng ta tham gia vào các giai đoạn của mỗi component bằng cách sử dụng các phương thức được xây dựng sẵn trong mỗi giai đoạn đó. Khi một components được khởi chạy nó sẽ phải trải qua 4 giai đoạn chính:

+ initialization

+ mounting

+ updating

+ unmounting

**Initialization**

Đây là giai đoạn mà thành phần sẽ bắt đầu hành trình của mình bằng cách khởi tạo state và props

**Mounting**

Giai đoạn này được thực hiện sau khi quá trình initialization(khởi tạo) được hoàn thành. Nó thực hiện nhiệm vụ chuyển **virtual DOM (DOM ảo)** trong React thành **DOM** và hiển thị trên trình duyệt. Component sẽ được render lần đầu tiên, ở đây chúng ta có 2 phương thức để tham gia vào giai đoạn này.

#### **componentWillMount()**

Được khởi chạy khi một component chuẩn bị được mount (tức là trước khi thực hiện render), sau khi thực hiện xong componentWillMount() thì component mới có thể được mount.

**Lưu ý:** Chúng ta **không** nên thực hiện bất cứ thay đổi nào liên quan đến state, props hay call API ở trong hàm này, bởi thời gian chuẩn bị mount -> mount rất ngắn nên các tác vụ đó không thể hoàn thành kịp. Khiến cho component render ra kết quả không như bạn mong muốn.

#### **componentDidMount()**

Được gọi khi component đã được mount (render thành công ), quá trình này xảy ra sau khi componentWillMount() thực hiện xong. Trong phương thức này bạn có thể gọi API, thay đổi state, props.

**Updating**

Đây là giai đoạn thứ ba mà các component phải thực hiện, sau giai đoạn initialization (khởi tạo ) , mount (render lần đầu),... . Trong giai đoạn này, dữ liệu của các phần (props & state) sẽ được cập nhật để đáp ứng với các sự kiện của người dùng như click, gõ, v.v. Điều này dẫn đến việc re-render lại component, ở trong giai đoạn này chúng ta sẽ có 4 phương thức chính:

#### **shouldComponentUpdate()**

Phương thức này xác định xem component có nên được render lại hay không ? Theo mặc định, nó trả về true. Nhưng bạn có thể thay đổi giá trị trả về của nó theo từng trường hợp.

Nó sẽ nhận về 2 tham số truyền vào là nextState và nextProps.

#### **componentWillUpdate()**

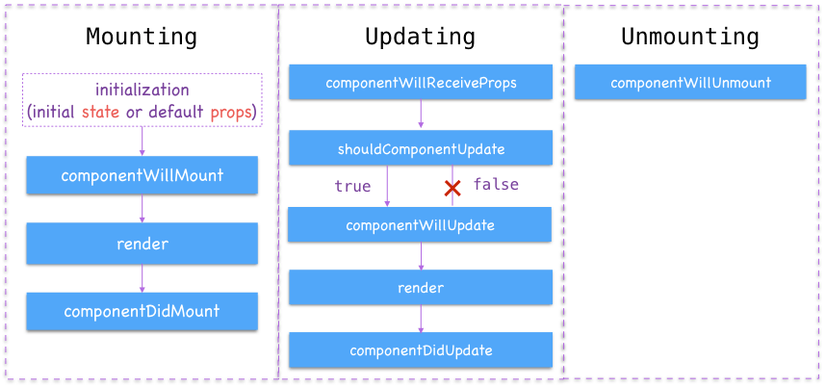
Phương thức này được gọi trước khi tiến hành re-render, bạn có thể thực hiện các hành động như update state, props,...trong phương thức này trước khi tiến hành re-render. Giống như shouldComponentUpdate(), componentWillUpdate() sẽ nhận vào 2 tham số đó là nextState và nextProps

#### **ComponentDidUpdate()**

Phương thức này được gọi khi component đã re-render xong. Chúng ta có ví dụ về cả 3 phương thức về đề cập ở trên.

**Unmounting**

Đây là bước cuối cùng trong mỗi component, khi tất cả các tác vụ hoàn thành và bạn tiến hành unmount DOM. Quá trình này chỉ có duy nhất 1 phương thức đó là componentWillUnmount() :



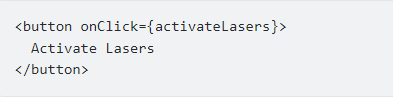
## Handle Event

Xử lý các sự kiện trong React rất giống với xử lý các sự kiện trên các phần tử DOM. Có một số khác biệt về cú pháp:

Các sự kiện React được đặt tên bằng camelCase, thay vì chữ thường. Ví dụ: onclick -> onClick, onchange -> onChange

Với JSX, bạn truyền một hàm để bắt sự kiện, thay vì một chuỗi như HTML thông thường.

Ví dụ minh họa



## Virtual Dom

Trước khi tìm hiểu về virtual dom là gì chúng ta sẽ tìm hiểu dom là gì với hình ảnh minh họa phía dưới:



**DOM** là tên gọi tắt của **Document Object Model** (Mô hình Đối tượng Tài liệu), là một chuẩn được định nghĩa bởi W3C dùng để truy xuất và thao tác trên code HTML hay XML bằng các ngôn ngữ lập trình thông dịch (scripting language) như Javascript.

**DOM** giúp thao tác với dữ liệu theo mô hình hướng đối tượng do các phần tử trong **DOM** có cấu trúc được định nghĩa thành các đối tượng, phương thức, thuộc tính để có thể truy xuất dễ dàng. Chúng được coi như các node và được biểu diễn dưới dạng **DOM Tree**.

**HTML DOM** cung cấp API để duyệt và chỉnh sửa các node. Nó chứa các phương thức như getElement, removeChild hay appenChild

var content = document.getElementById("myContent");

content.parentNode.removeChild(item);

Như hình ở trên, **HTML DOM** được cấu trúc dạng cây. Thực sự rất ngon vì có thể duyệt cây rất dễ dàng. Thật là đen, ở đây không phải cứ dễ là tốc độ nhanh.

Điều chậm ở đây là layout mà các trình duyệt phải làm bất cứ khi nào DOM thay đổi. Mỗi khi DOM thay đổi, trình duyệt cần phải tính toán lại CSS, thực hiện dựng lại trang web. Đây là việc cần có thời gian.

Và hiện nay **DOM Tree** thực sự rất lớn. Việc những trang web SPA ngày càng được phát triển mạnh và đa dạng, vì vậy việc sửa đổi **DOM Tree** là liên tục không ngừng và sửa đổi rất nhiều.

Xem xét một **DOM** được tạo bởi hàng nghìn div, Có quá nhiều phương thức để xử lý các event click, submit, ... Điển hình của việc xử lý event trong jQuery sẽ là:

* Tìm tất cả các node liên quan đến event
* Cập nhật nó nếu thấy cần thiết

Chúng ta gặp 2 vấn đề:

* Thực sự rất khó để quản lý. Tưởng tượng xem nếu phải chỉnh sửa một đoạn xử lý event mà không nắm được context thì bạn sẽ phải bơi thật sâu trong code để có thể xem nó đang làm gì. Tốn thời gian và rủi ro cao.
* Nó không hiệu quả. Có nhất thiết cứ phải đi tìm tất cả những gì liên quan không? Hay có thể thông minh hơn bằng cách chỉ tìm node nào cần cập nhật.

ReactJs đến và cho chúng ta giải pháp, thay vì xử lý **DOM Tree** thủ công, chúng ta định nghĩa các component trông giống giống thế còn ReactJs sẽ thực hiện công việc ở tầng thấp hơn. Thực chất, công việc xử lý sẽ được **HTML DOM API** đảm nhiệm ở các tầng đó. Đây chính xác là cách mà **Virtual DOM** hoạt động.

****Virtual DOM**** không được tạo ra bởi React tuy nhiên nó được React sử dụng và cung cấp miễn phí.

Một cách tổng quát thì nó là một định dạng dữ liệu JavaScript nhẹ được dùng để thể hiện nội dung của DOM tại một thời điểm nhất định nào đó. Nó có tất cả các thuộc tính giống như **DOM** nhưng không có khả năng tương tác lên màn hình như **DOM**.

Bạn có thể tưởng tượng, ở **DOM** có thẻ div và các thẻ p ở trong, ReactJs sử dụng **Virtual DOM** bằng cách tạo ra các object React.div và React.p và khi tương tác, ta sẽ tương tác qua các object đó một cách nhanh chóng mà không phải đụng tới **DOM** hay **DOM API** của nó.

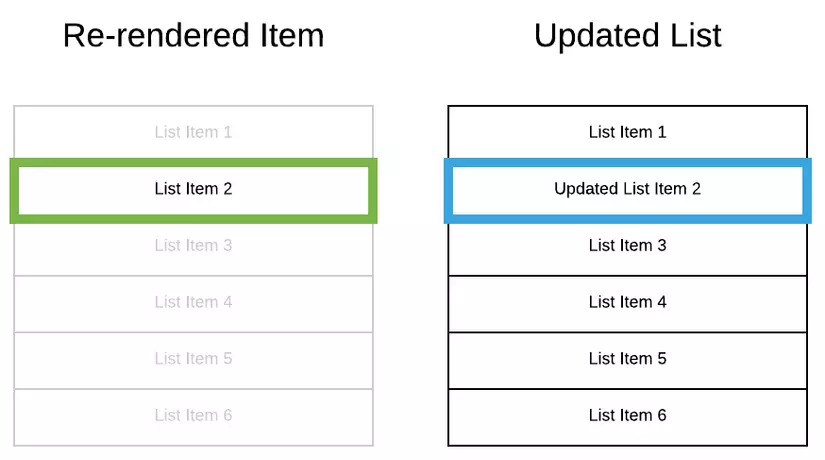
Đây là lí do tại sao JSX của code ReactJs có thể trông như code HTML thuần tuý

****Virtual DOM**** sử dụng key, ref mà ở **DOM** không có  và **Virtual DOM** được tạo mới sau mỗi lần render lại.

Tuy nhiên, sự đặc biệt của **Virtual DOM** nằm ở Snapshots & Diffing Như giải thích ở trước đó, cách hoạt đông của **Virtual DOM** trong React đó là:

React lấy một snapshot của **Virtual DOM** (có thể hiểu là bản ghi trạng thái ngay lúc đó) ngay trước khi áp dụng bất kỳ bản cập nhật nào. Sau đó, nó sử dụng snapshot này để so sánh với một **Virtual DOM** được cập nhật trước khi thực hiện các thay đổi.

Khi cập nhật được cấp cho **Virtual DOM**, quá trình tiếp theo React sử dụng thuật toán Diffing để so sánh và đối chiếu để biết được sự cập nhật được diễn ra ở đâu sau đó cập nhật nó mà bỏ qua những elements không liên quan.



Một lợi ích gắn liền khi sử dụng React là chúng ta có thể kiểm soát việc re-render của các component bằng cách sử dụng phương thức setState và shouldComponentUpdate

## 8 . Hook

### 8.1 Usestate

Trong bài viết này chúng ta sẽ cùng nhau đi tìm hiểu về **React Hook useState**. Từ phiên bản 16.8 trở đi React cho ra mắt một tính năng mới đó là React Hooks, nó cho phép chúng ta làm việc với state, life cycle, và hàng loạt tính năng khác trong một functional component.

**useState** là một hook cho phép chúng ta quản lý các state trong một functional component, Bằng cách gọi React.useState bên trong một functional compoenent, bạn đã có thể làm việc với state một cách nhanh chóng.

Để sử dụng useState, trước tiên chúng ta cần import nó vào component.

import React, { useState } from "react";

Tiếp theo, bạn chỉ cần sử dụng useState bằng cú pháp:

const [nameState, funcUpdate] = useState(defaultState)

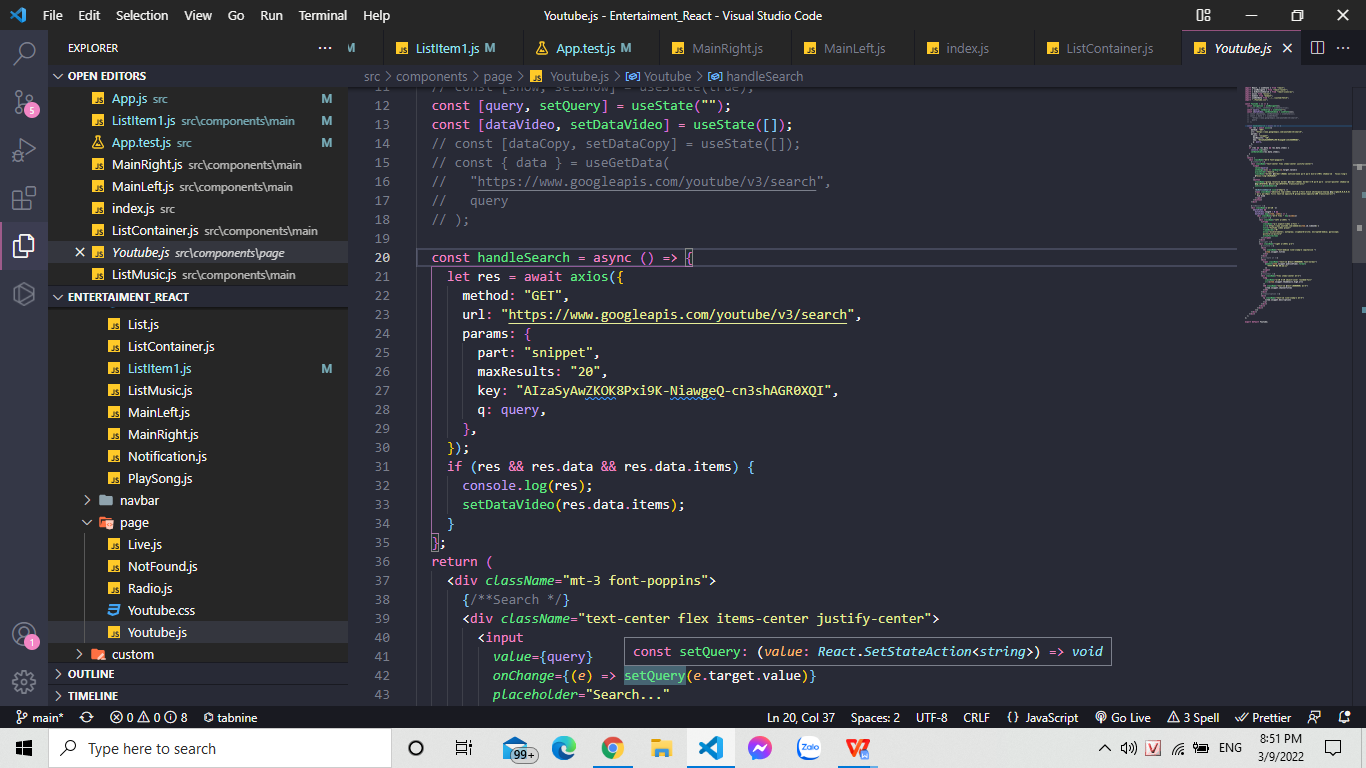
Ở đây chúng ta có 3 đối số cần chú ý đến:

+ nameState: đây là gía trị mặc định của state.

+ funcUpdate: function dùng để cập nhật state. Giả sử mình muốn cập nhật giá trị của state mình chỉ cần gọi funcUpdate('giá trị mới của state').

+ defaultState: giá trị mặc định của state khi được khởi tạo lần đầu.

Ví dụ minh họa

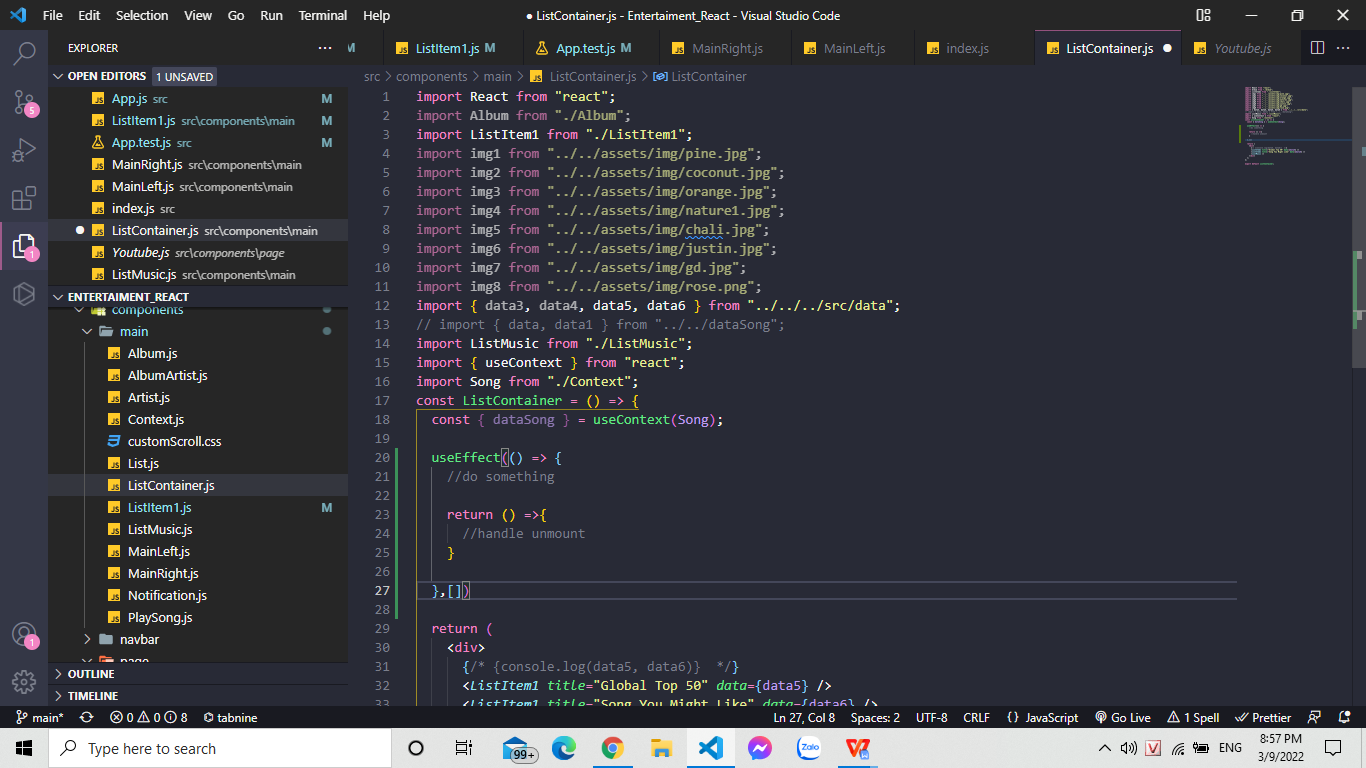


### 8.2 UseEffect

useEffect() là function nắm bắt tất cả các sự thay đổi của code. Trong một function component, việc sử dụng life cycle không React hỗ trợ, bởi vậy rất khó để debug, cũng như nắm bắt được quá trình khởi chạy của component.

useEffect() sinh ra để làm điều này, nó được khởi chạy khi giá trị của một biến nào đó thay đổi, hay component đã được render ra,...useEffect() có thể thay thế hòan toàn các life cycle trong class component. Chúng ta có thể sử dụng nó bằng cú pháp :

useEffect là một hook trong React Hooks cho phép chúng ta làm việc với các [life cycle](https://freetuts.net/component-life-cycle-trong-reactjs-2387.html" \o "component life cycle trong reactjs 2387 html" \t "https://freetuts.net/_blank)ở functional component. Có thể hiểu đơn giản rằng useEffect Hook là của 3 phương thức componentDidMount, componentDidUpdate và componentWillUnmount kết hợp lại với nhau.



Lifecyle là một phần rất quan trọng trong một component. Trong một vài trường hợp chúng ta cần phải fetch data từ API khi component đã được render, hay thực hiện hành động nào đó khi một component được update. Bởi vậy có thể thấy rằng phương thức quan trọng và hay được sử dụng nhất trong life cycle đó là componentDidMount, componentDidUpdate.

Nhưng trong một functional component không thể làm việc với các life cycle này bằng cách thông thường, bởi vậy useEffect Hooks sinh ra để làm điều này.

useEffect(effectFunction, arrayDependencies)

arrayDependencies: [] => chạy useEffect 1 lần duy nhất khi UI được mount

arrayDependencies: không có => luôn chạy useEffect sau mỗi lần component đươc re-render

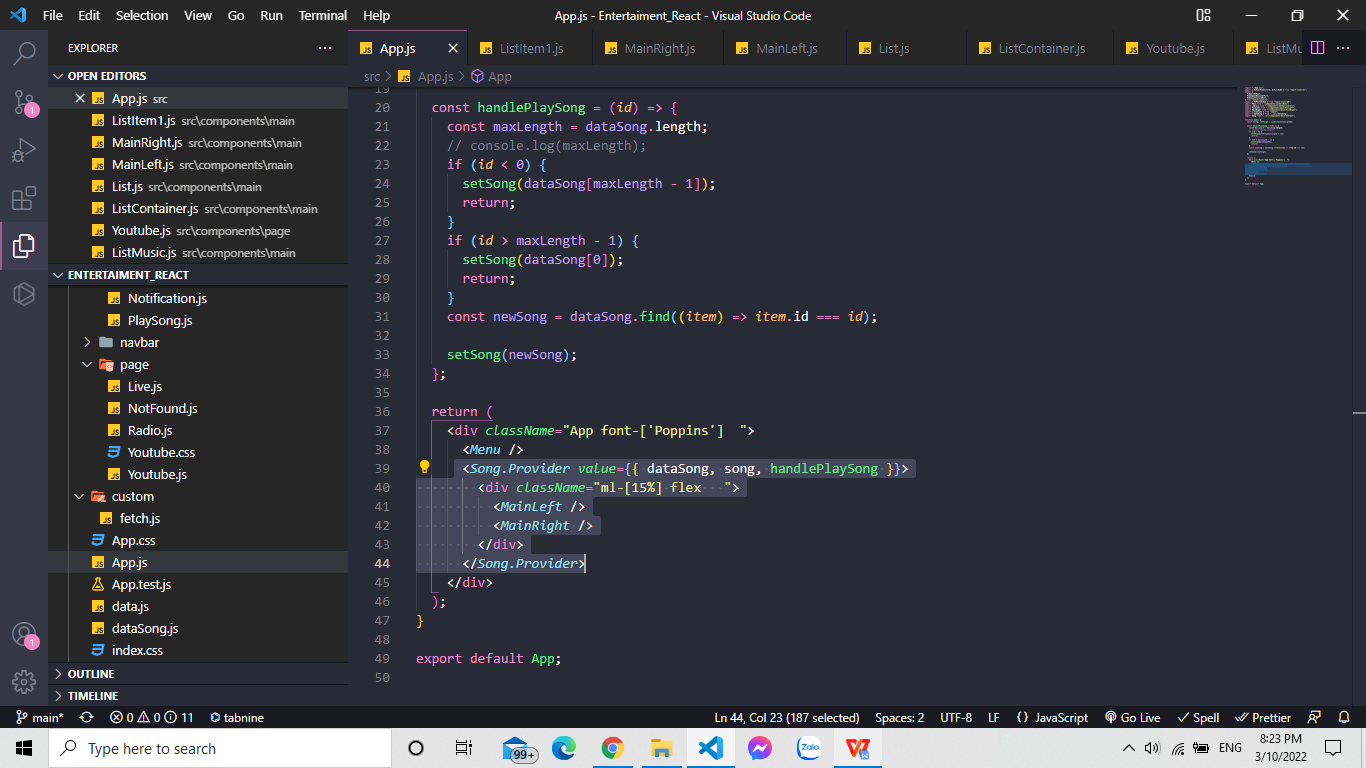
arrayDependencies: [state] => useEffect chạy khi điều kiện ràng buộc thay đổi

### 8.3. UseContext

React context API là một cách cơ bản để tạo các biến toàn cục có thể được truyền qua trong ứng dụng React. đây là phương pháp thay thế cho “prop drilling”, hoặc truyền props từ ông nội sang cha và sang con, v..v.. Context thường được coi là đơn giản hơn, nhẹ nhàng hơn thay vì sử dụng Redux cho quản lý state.

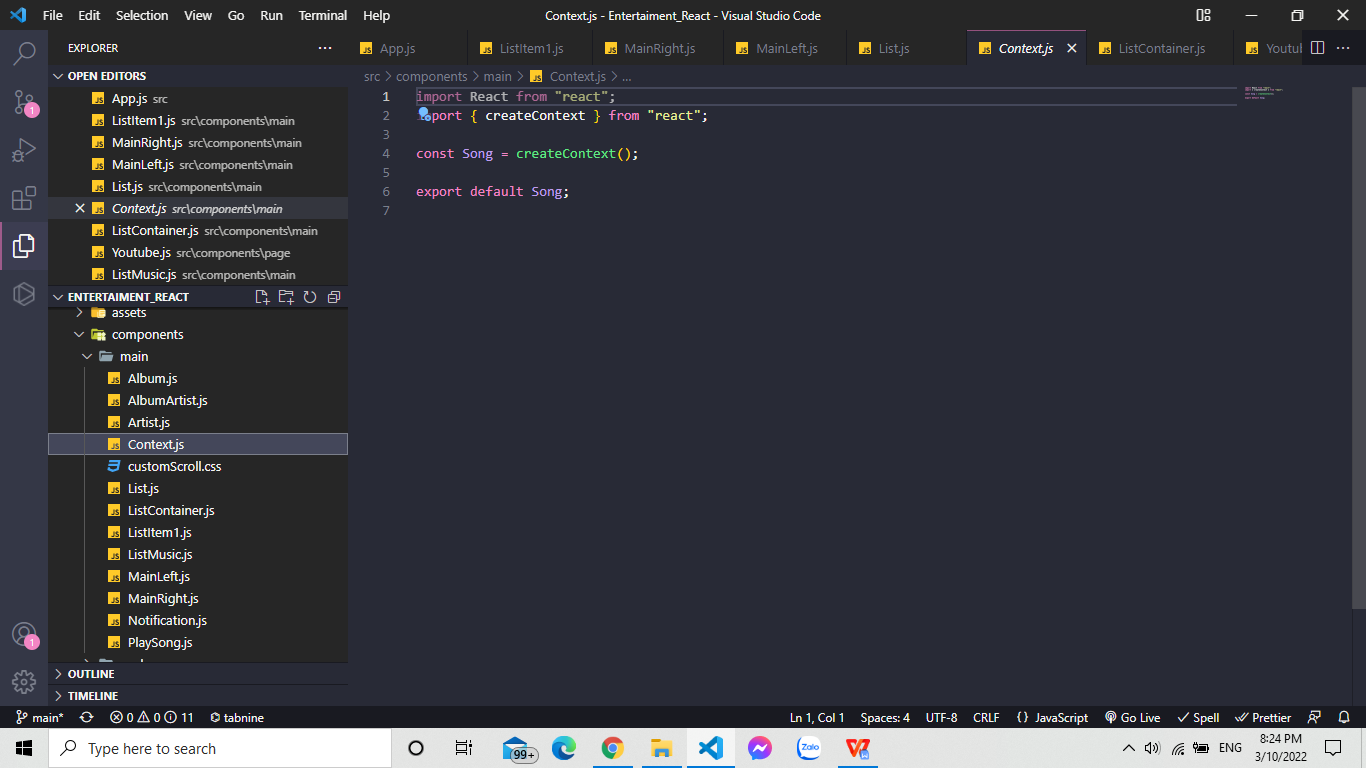
Tạo Context, và gọi nó là Song. Điều này sẽ cung cấp Song.Provider. Những gì thành phần này làm đơn giản là :

Provider : thành phần cung cấp giá trị

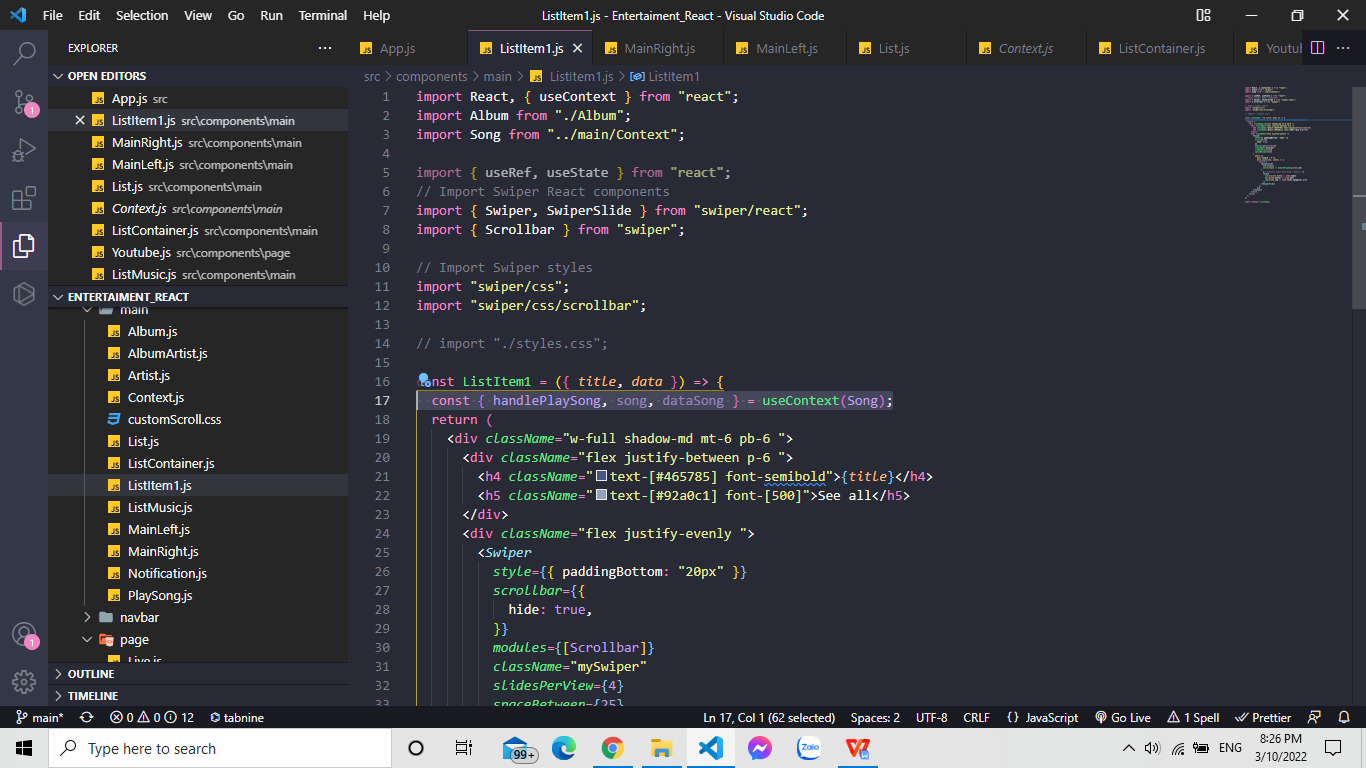


Provider luôn cần tồn tại như một trình bao bọc xung quanh phần tử cha, bất kể bên trong có các giá trị như nào.

Trong ví dụ trên, em khai báo :

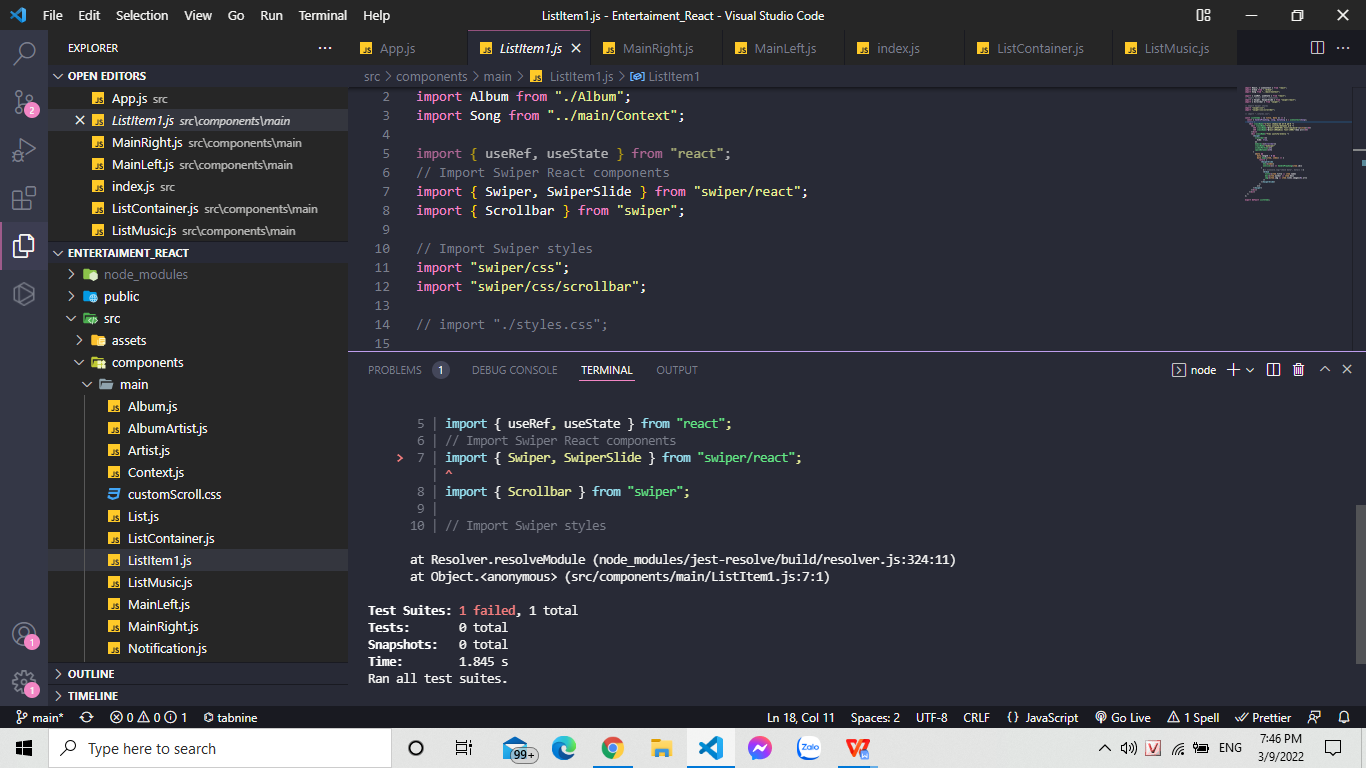
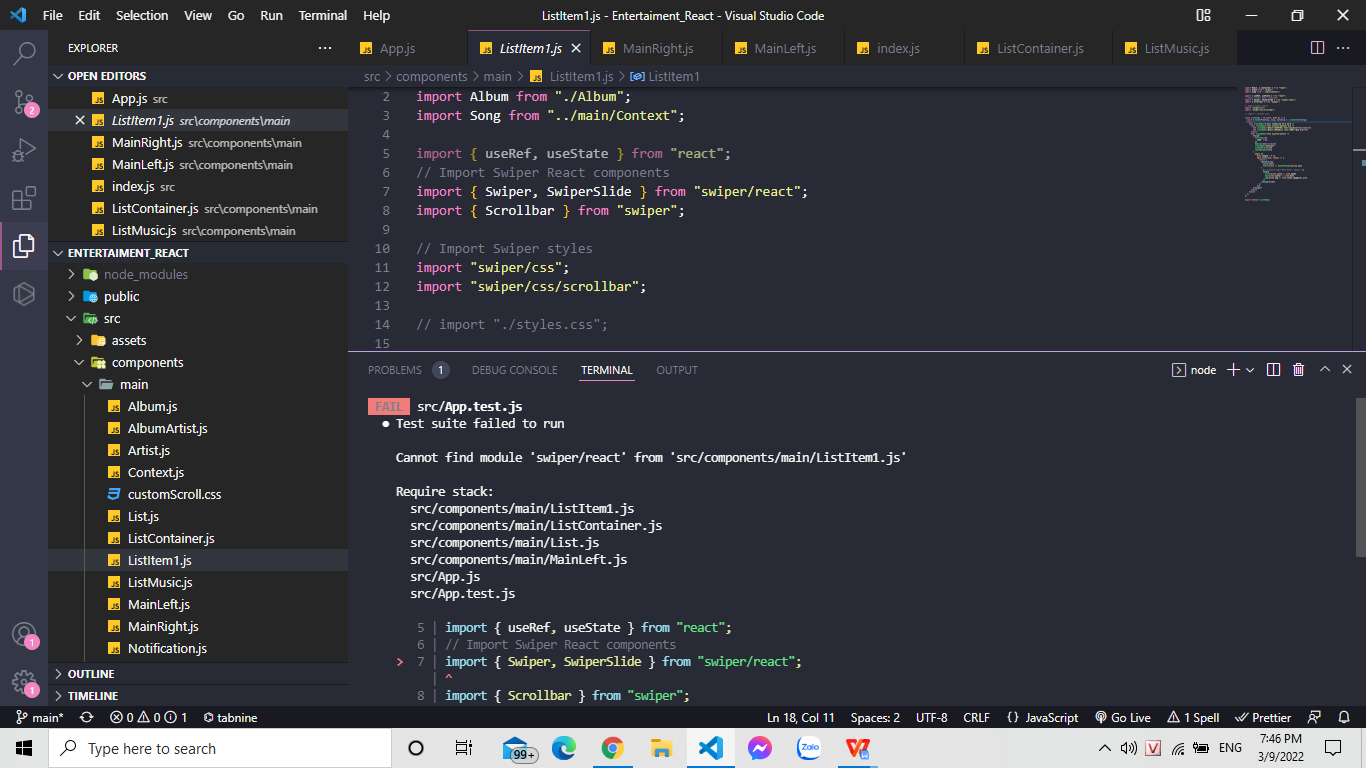


Sau đó e sử dụng Song.Provider có prop value = {dataSong, song, handlePlaySong} bao lại các component được subscribe  
Sau khi gọi các component con bên trong provider này. Để component đã subscribe để sử dụng data từ Context Api chúng ta sử dụng UseContext



# TEST APP REACT

Mở terminal trong vs code -> nhập npm test -> phần mềm sẽ tiến hành test và về kết quả (trả về lỗi cần fix)



Kết quả trả về phần mềm test báo không thể tìm thấy module ‘swiper/react’ nhưng khi em npm start để khởi động app lên để check xem component đã nhận được module ‘swiper/react’ không . Thì kết quả vẫn chạy bình thường.