REACT

-JSX

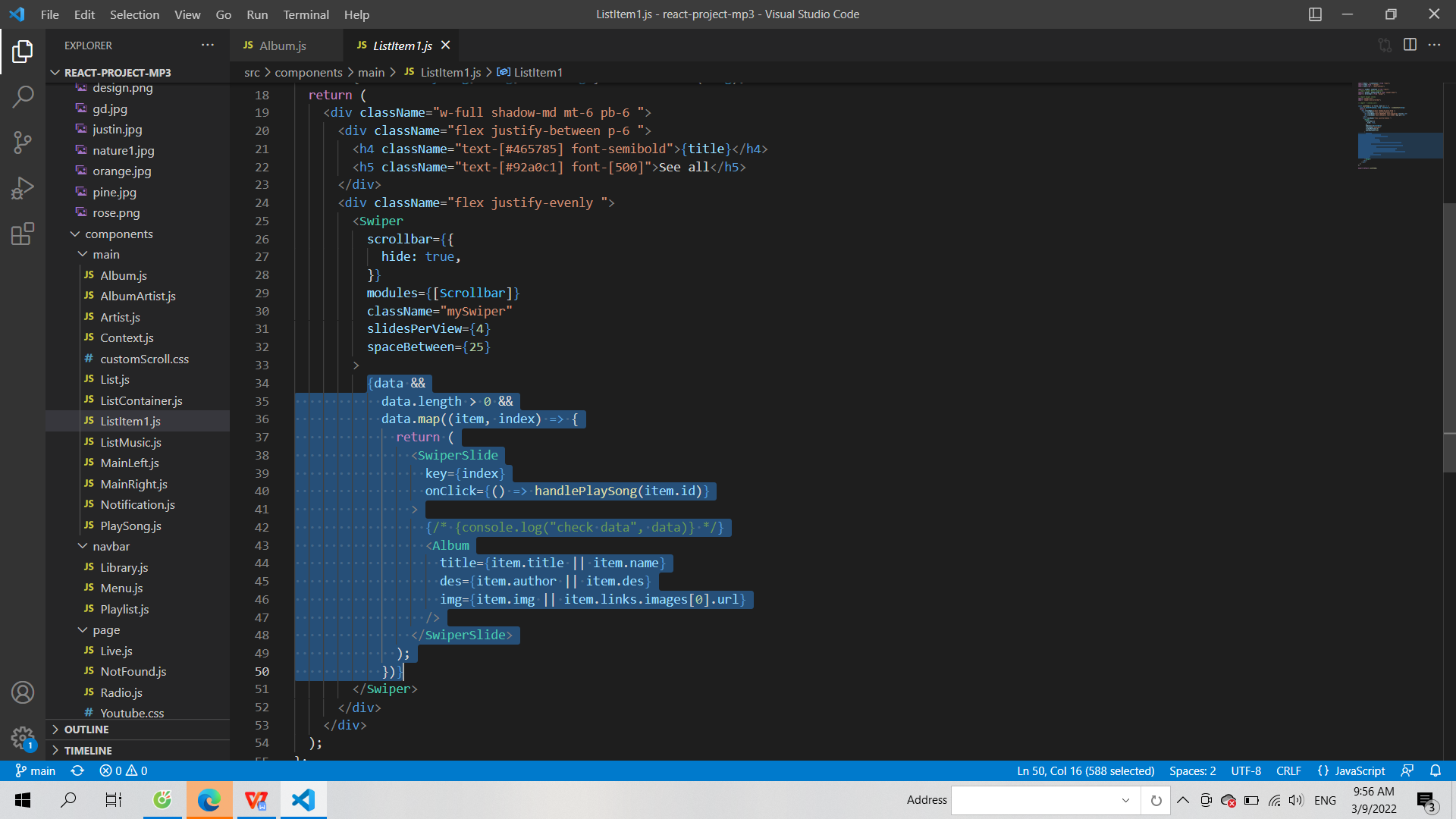
**JSX** là viết tắt là Javascript XML, một template languges nhưng nó lại mang hầu hết tính năng của Javascript. Nó cho phép bạn viết các đoạn mã HTML trong React một cách dẽ dàng và có cấu trúc hơn. React sử dụng JSX cho việc xây dựng bố cục thay vì javascript thông thường. JSX giúp tạo ra các React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



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

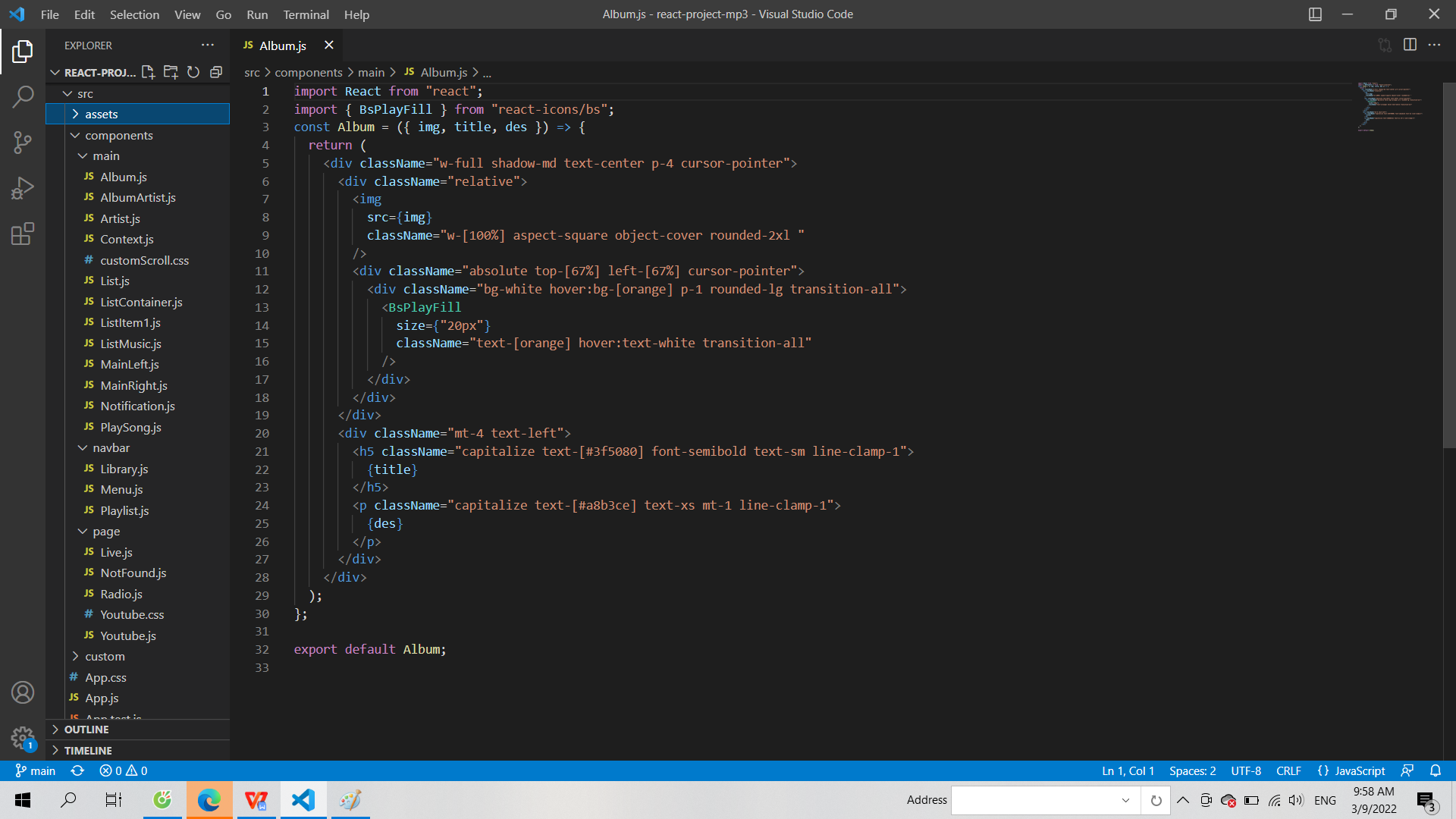
-COMPONENT

**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

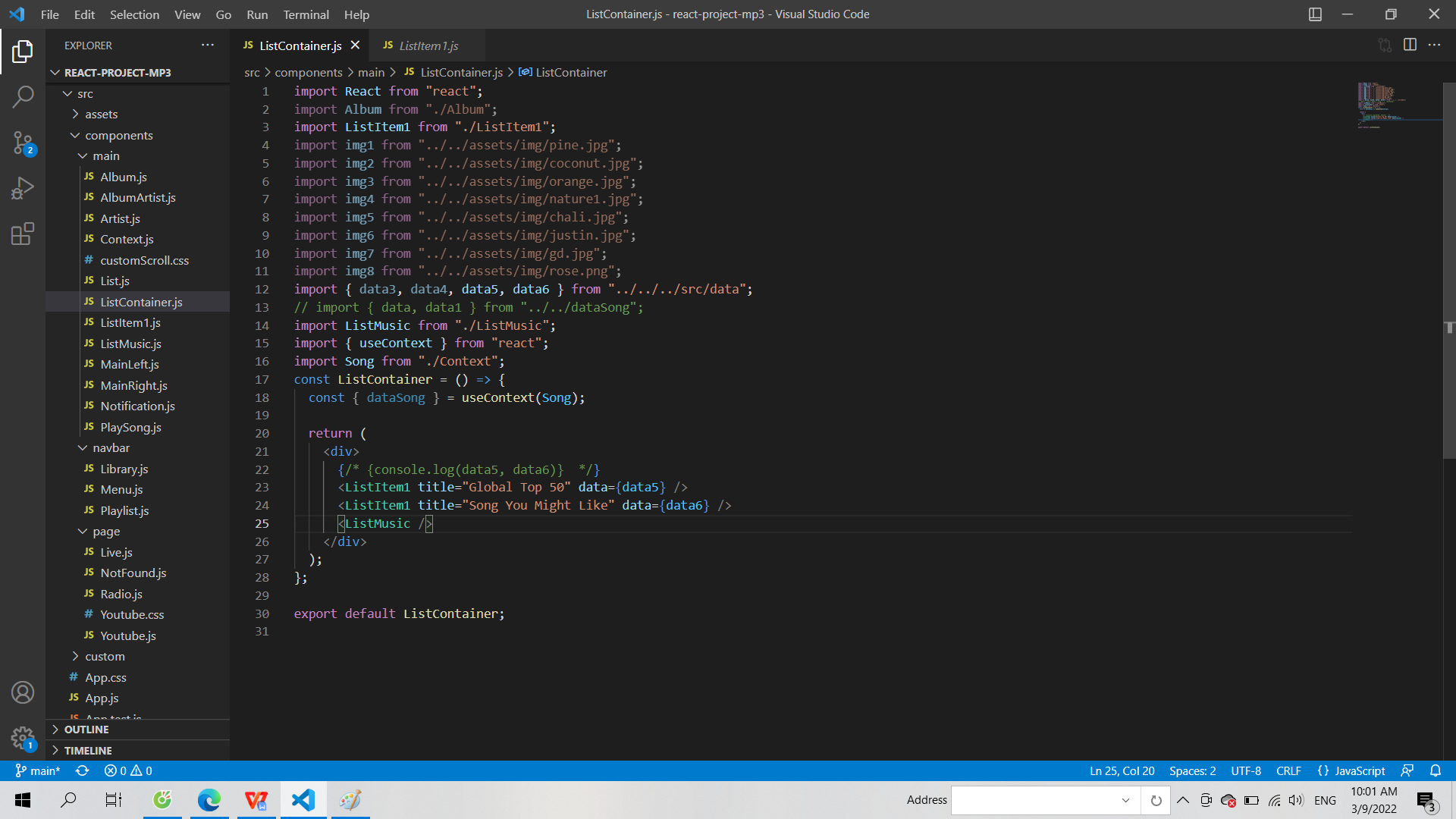


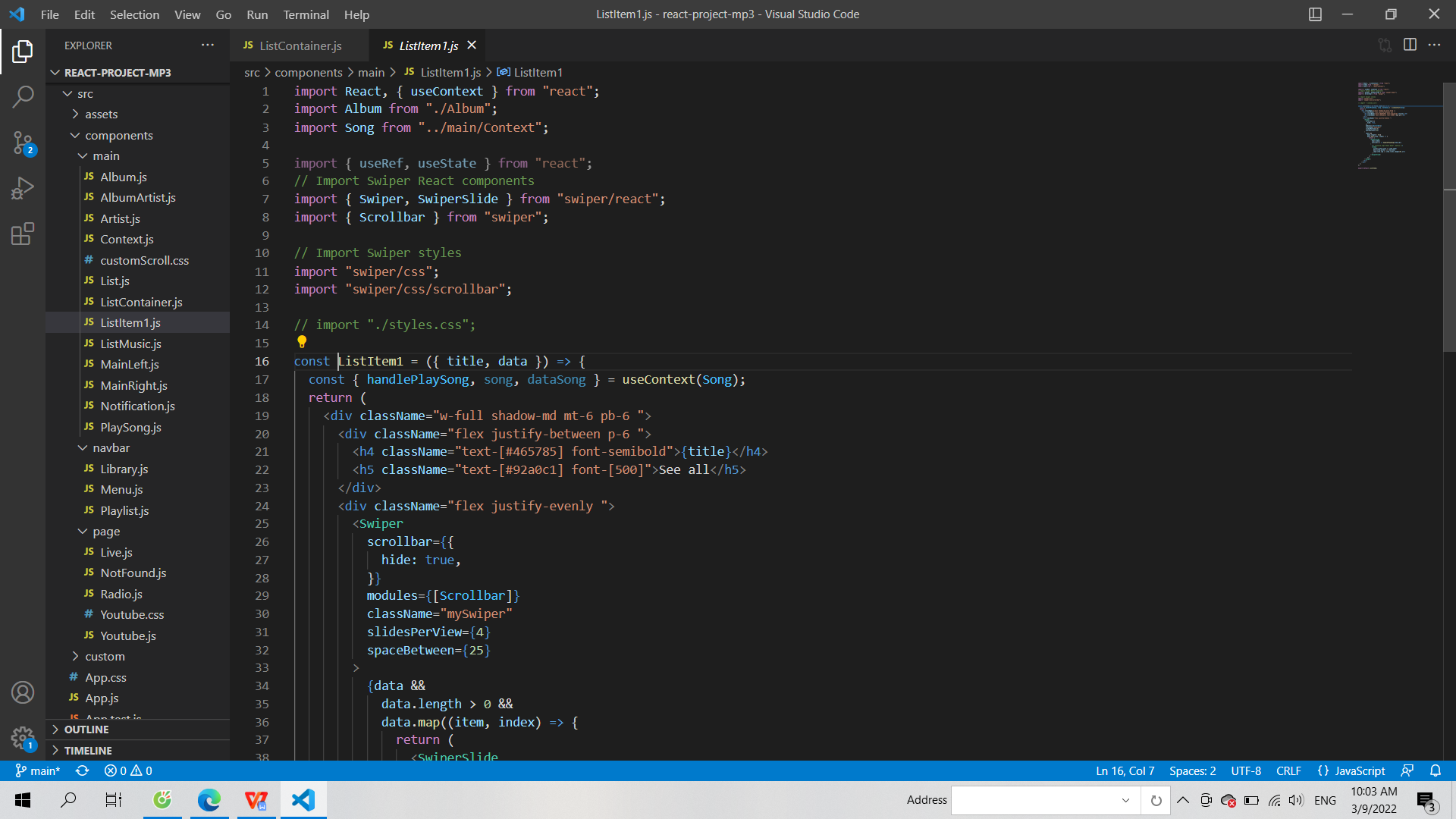
-PROPS

**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

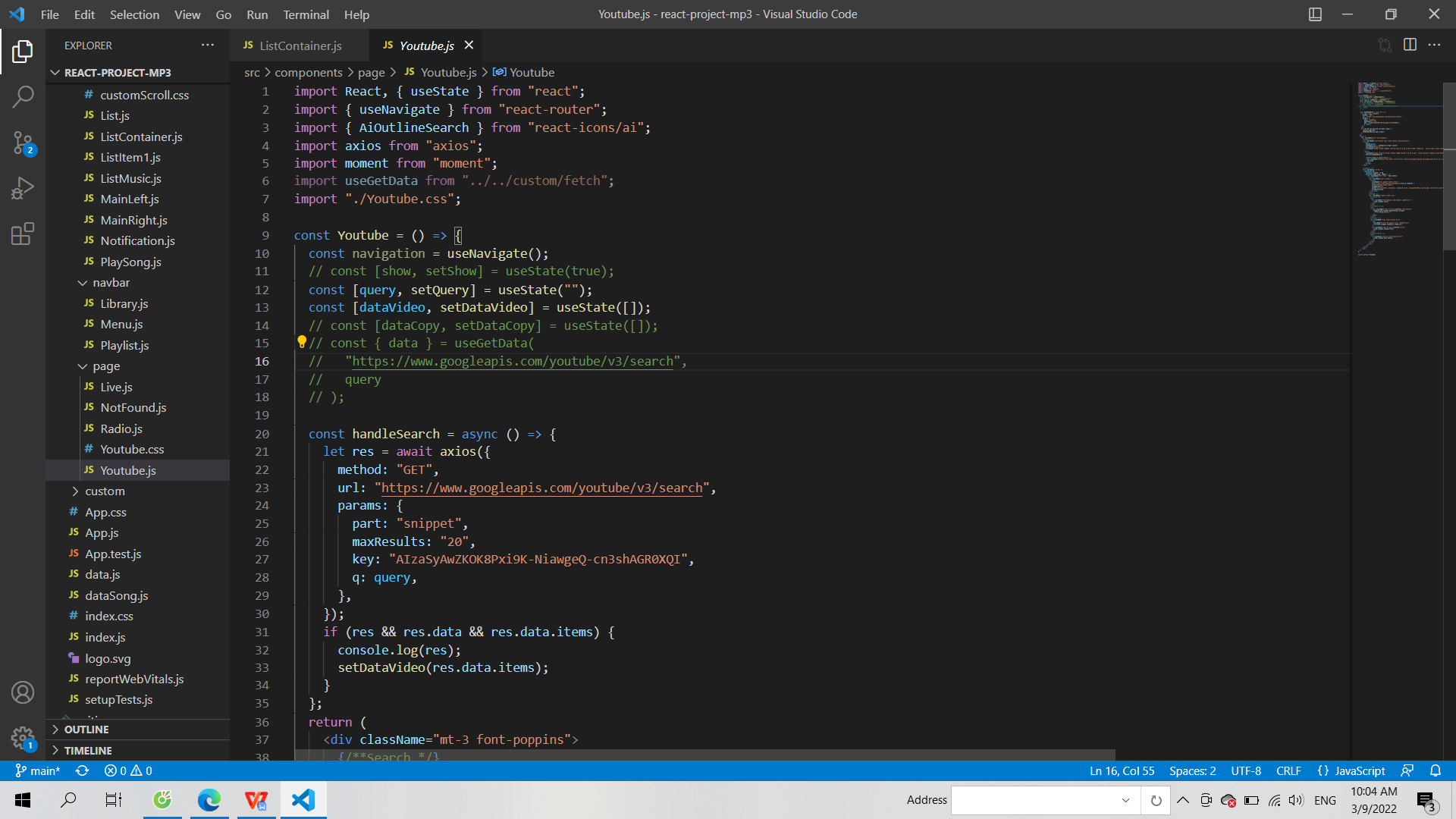




**-STATE**

**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.

# -**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()**

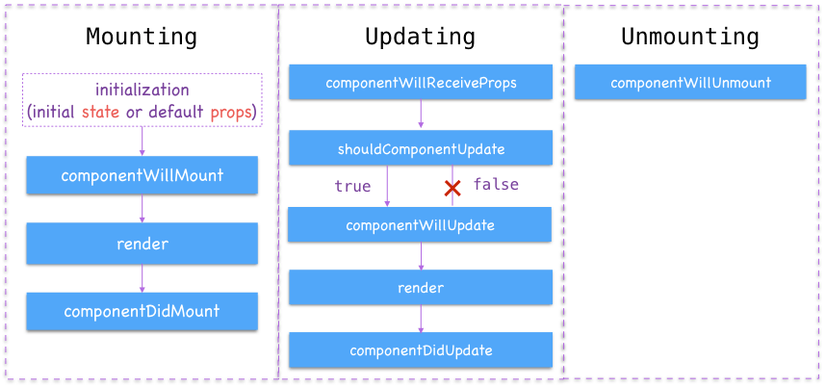
Phương thức này được gọi trước khi tiến hành re-render, bạn có thể thực hiện các hành động như update state, props,...trong phương thức này trước khi tiến hành re-render. Giống như shouldComponentUpdate(), componentWillUpdate() sẽ nhận vào 2 tham số đó là nextState và nextProps

#### **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

<button onClick={changeName}>

  Change Name

</button>

**-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] = React.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

-**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**,**vàcomponentWillUnmount 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 lifce 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) mfemf