# **Day 3**

# **Xây dựng cấu trúc project với react-native**

Vì React chỉ là một thư viện nên nó sẽ không có các ràng buộc trong việc sắp xếp và cấu trúc thư mục dự án của bạn. Điều hay của việc này là nó sẽ cho chúng ta được tự do tham khảo các phương pháp khác nhau và áp dụng chúng ngay lập tức khi thấy phù hợp. Tuy vậy, từ một góc nhìn khác thì nó cũng có thể gây ra sự "bối rối" cho những lập trình viên mới bắt đầu bước vào thế giới của React.

Trong bài viết này, tôi sẽ cho các bạn thấy những phương pháp mà tôi đã sử dụng trong một thời gian và có khả năng mở rộng khá là tốt. Những phương pháp này không phải là những thứ hoàn toàn mới mà chúng được tổng hợp và tinh chỉnh lại dựa trên những phương pháp đã được mọi người sử dụng.

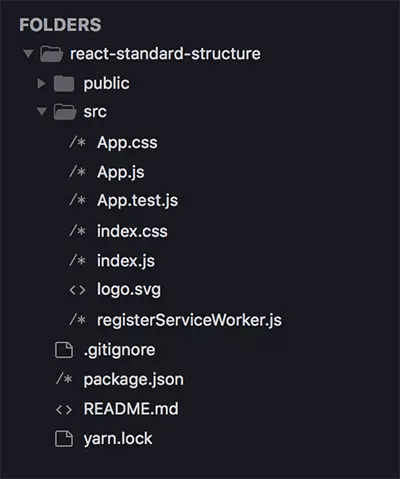
Note: Không có phương pháp nào trong bài viết này là hoàn hảo cả. Bạn chỉ cần áp dụng những phương pháp mà bạn nghĩ rằng nó hợp lý và áp dụng nó một cách phù hợp với mình.

## **Cấu trúc thư mục**

Một trong các thắc mắc mà tôi thường thấy mọi người gặp phải là về việc tổ chức cấu trúc files và thư mục như thế nào. Trong bài viết này, tôi sẽ coi như bạn có một cấu trúc tối giản nhất được tạo bằng create-react-app.

create-react-app sẽ tạo cho chúng ta một project cơ bản. Chứa trong thư mục gốc của nó là các file: .gitignore, package.json, [README.md](http://readme.md/" \t "https://viblo.asia/p/_blank), yarn.lock

Nó cũng tạo ra các thư mục: public và src. Trong đó, thư mục src chính là nơi chứa source code của chúng ta.

Bạn có thể xem hình mô tả cấu trúc được tạo ra ở dưới đây:

Trong bài viết này, tôi sẽ chỉ tập trung vào thư mục src. Tất cả những thứ bên ngoài nó sẽ được để nguyên.

## **Containers và Components**

Có lẽ các bạn đã nhìn thấy việc tách ra các Containers và các Presentational Component trong thư mục gốc của một số project. Tức là, trong src, bạn sẽ thấy có một folder tên là components và một thư mục khác tên là containers:

src

├─ components

└─ containers

Tuy nhiên, với phương pháp tổ chức như thế này bạn có thể sẽ gặp phải một vài vấn đề như sau:

* ****Các rule sẽ mang tính chủ quan**** - Bạn không có các quy định rõ ràng về việc component như thế nào thì xếp vào loại Container hay xếp vào loại Presentational Component. Điểm khác biệt để bạn phân biệt giữa hai loại có thể mang tính chủ quan và khi bạn ở trong một team, sẽ rất khó để mà xảy ra việc tất cả các developer đều có sự nhận định vầ phân chia giống nhau trong việc này.
* ****Bỏ sót tính động của các component**** - Dù cho bạn có thể thống nhất được trong việc phân loại component thì các component cũng rất dễ bị thay đổi theo vòng đời của dự án. Việc chuyển đổi loại của một component sẽ khiến bạn phải di chuyển chúng từ thư mục components sang containers hoặc ngược lại.
* ****Xuất hiện các component trùng tên**** - Tên của các component nên có tính biểu thị cũng như phải là độc nhất trong một ứng dụng để tránh việc lẫn lộn về vai trò của chúng. Với phương pháp tổ chức thư mục như thế này sẽ tạo ra một kẽ hở cho việc hai component với tên giống hệt nhau có thể xảy ra.
* ****Giảm hiệu suất làm việc của dev**** - Bạn sẽ phải liên tục di chuyển giữa folder containers và folder components dù cho là đang code chỉ một feature bởi vì thông thường một feature sẽ sử dụng component ở cả hai loại.

Cách phân chia folder theo hai loại component này còn có một biến thể khác đó là phân chia thêm theo module nữa.

Ví dụ, trong app của bạn, bạn có module User. Bên trong nó, bạn có hai folder dành riêng cho hai loại component:

src

└─ User

├─ components

└─ containers

Với phương pháp này, việc di chuyển giữa hai folder được giảm đi. Tuy nhiên, nếu như app của bạn có nhiều module, thì kết quả là bạn sẽ có hàng đống folder containers và components.

Vì những lý do đó, khi đề cập tới vấn để sắp xếp folder và file thì không cần quan tâm đến việc phận chia các component của chúng ta theo các khái niệm presentational và container. Chúng ta sẽ đặt toàn bộ các component vào thư mục components, ngoại trừ các component dùng làm screens.

Dù cho việc phân tách containers và presentational component thành các folder riêng là không cần thiết thì việc nắm được sự khác nhau giữa hai khái niệm này vẫn là cần thiết.

## **Phân tách và gộp nhóm code**

Bên trong folder components, chúng ta sẽ nhóm các file theo module/feature.

Trong việc CRUD của user, chúng ta sẽ chỉ cần có module User. Vì vậy, cấu trúc của chúng ta sẽ như sau:

src

└─ components

└─ User

├─ Form.jsx

└─ List.jsx

Khi một component được tạo thành bởi nhiều hơn một file, chúng ta sẽ đặt component này và các file khác của nó vào một folder có cùng tên. VD: Giả sử bạn có file Form.css chứa style cho Form.jsx. Trong trường hợp này, cấu trúc của bạn sẽ như thế này:

src

└─ components

└─ User

├─ Form

│ ├─ Form.jsx

│ └─ Form.css

└─ List.jsx

File test sẽ ở cùng một chỗ với file được test, Trong trường hợp ở trên, test cho ****Form.jsx**** sẽ ở trong cùng folder với nó và được đặt tên là ****Form.spec.jsx****

### **UI Components**

Ngoài việc phần chia components theo modules, chúng ta cũng sẽ thêm một folder UI bên trong src/components để chứa tất cả các component chung ở trong đó.

UI Components là các component chung không thuộc về riêng một module nào cả. Chúng là các component mà bạn có thể đưa vào một thự viện open source vì chúng chẳng chứa các logic dành riêng cho một ứng dụng nào cả. Ví dụ về các component này đó là: Buttons, Inputs, Checkboxes, Selects, Modals, Data display elements, v.v…

## **Đặt tên cho components**

Ở trên chúng ta đã thấy cách cấu trúc thư mục và phân chia component. Tuy nhiên vẫn còn một câu hỏi: Đặt tên chúng như thế nào?

Đặt tên ở đây là về việc đặt tên cho ****class**** hoặc cho các ****const**** dùng để định nghĩa component:

class ****MyComponent**** extends Component { }

const ****MyComponent**** () => {};

Như đã đề cập ở phần trước, tên của component cần phải rõ ràng và là duy nhất ở trong một app để có thể khiến chúng dễ tìm hơn và tránh lẫn lộn.

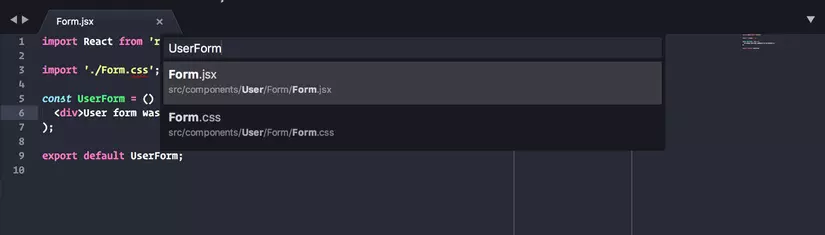
Tên của một component rất hữu ích khi chúng ta cần phải debug bằng tool (ví dụ như là React Dev Tools), vì khi có lỗi xảy ra thì nó luôn được hiển thị cùng với tên component gặp lỗi.

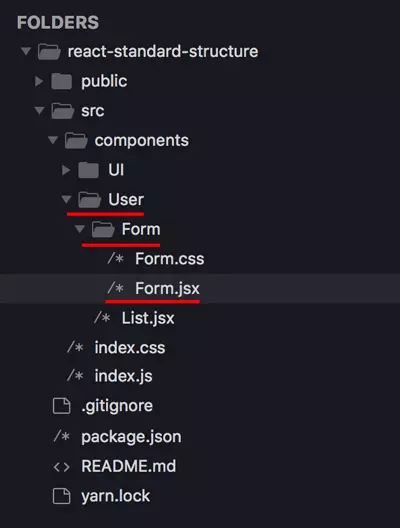
Để đặt tên cho component, chúng ta sẽ dựa theo path, tức là đặt tên component dựa theo đường dẫn tương đối tới folder components (hoặc src trong trường hợp file nằm ở ngoài folder components). Cơ bản thì, một component được đặt ở: components/User/List.jsx sẽ được đặt tên là UserList.

Khi file ở trong một folder có cùng tên với nó, chúng ta không cần lặp lại nó ở trong tên của component. Ví dụ, component ở: components/User/Form/Form.jsx, sẽ được đặt tên là UserForm chứ không phải là UserFormForm.

Cách đặt tên như trên sẽ đem đến một vài lợi ích như sau:

### **Tìm kiếm file dễ hơn**

Khi dùng một số editor vs như VSCode, chỉ cần search bằng tên của component UserName bạn có thể tìm ra đúng file cần mở:

Nếu bạn muốn tìm file trên cây thư mục, bạn cũng có thể dễ dàng tìm thấy nó bằng cách di chuyển theo tên component:

### **Tránh được việc lặp lại tên khi import**

Đặt tên theo cách này, bạn sẽ đặt tên ****file**** theo context của nó. Ví dụ, xem xét ****form**** phía trên, nó là ****user form****, nhưng vì chúng ta đã ở trong folder ****User****, chúng ta không cần lặp lại từ đó trong tên của ****file**** component nữa. Vì vậy, chúng ta chỉ cần đặt nó là ****Form.jsx****

Khi mới bắt đầu làm việc với React, tôi thường để đầy đủ tên file. Tuy nhiên điều này khiến bạn lăp lại tên rất nhiều lần và import path trở nên quá dài. Hãy thử nhìn sự khác biệt giữa hai phương pháp:

import ScreensUserForm from './screens/User/UserForm';// vsimport ScreensUserForm from './screens/User/Form';

Ở ví dụ trên, có thể bạn không thấy được ưu điểm của cách này so với cách kia. Nhưng nếu ứng dụng của bạn lớn hơn một chút nữa, bạn có thể thấy được sự khác biệt rõ ràng hơn. Hãy thử xem ví dụ dưới đây:

import MediaPlanViewChannel from './MediaPlan/MediaPlanView/MediaPlanViewChannel';// vsimport MediaPlanViewChannel from './MediaPlan/View/Channel';

Hãy thử tưởng tượng sự lặp này tăng lên gấp 5 hoặc 10 lần cho một file thì sẽ như thế nào.

Với những lý do như vậy, chúng ta nên luôn luôn đặt tên file dựa theo context của nó và đặt tên component dựa theo đường dẫn tương đối của file chứa nó tới folder components hoặc src.

## **Screens**

Screens, như tên của nó đã tự mô tả, chính là các màn hình có trong ứng dụng của chúng ta.

Trong việc CRUD của user, chúng ta sẽ cần có một màn hình cho danh sách các user, một màn hình cho việc tạo mới user và một màn hình cho việc edit các user đã tồn tại.

Một screen là nơi mà bạn sử dụng các component để tạo thành một trang của ứng dụng. Lý tưởng nhất thì screen sẽ không chứa bất kì logic nào cả và sẽ chỉ đơn giản là một functional component (nôm na tức là không có state riêng của nó).

Chúng ta sẽ đặt các screen trong một thư mục riêng trong src, bởi vì chúng cần được gộp dựa theo việc định nghĩa các route chứ không phải theo module:

src

├─ components

└─ screens

└─ User

├─ Form.jsx

└─ List.jsx

Giả sử chúng ta điều hướng bằng react-router, chúng ta sẽ đặt file Root.jsx bên trong folder screen, và chúng ta sẽ định nghĩa ở trong nó các routes của ứng dụng.

Code của Root.jsx sẽ trông tương tự như sau:

import React, { Component } from 'react';import { Router } from 'react-router';import { Redirect, Route, Switch } from 'react-router-dom';

import ScreensUserForm from './User/Form';import ScreensUserList from './User/List';

const ScreensRoot = () => (

<Router>

<Switch>

<Route path="/users" component={ScreenUserList} />

<Route path="/users/new" component={ScreenUserForm} />

<Route path="/users/:id" component={ScreenUserForm} />

</Switch>

</Router>);

export default ScreensRoot;

Hãy chú ý tới việc chúng ta đặt tất cả screen bên trong một folder có cùng tên với route, /users -> User. Hãy tạo mỗi folder cho mỗi route và nhóm các route phụ ở trong đó. Trong trường hợp này, chúng ta đã tạo folder User và chúng ta đặt screen List và screen Form ở trong đó. Phương pháp này sẽ giúp bạn dễ dàng tìm được screen nào được render ra trong từng route mà chỉ cần nhìn vào url.

Một screen có thể được dùng để render ở hai route khác nhau như trong ví dụ trên với các route tạo mới và edit user.

Bạn có thể để ý thấy rằng các screen component đều có tiền tố Screen trong tên của chúng. Khi component được đặt bên ngoài folder components, chúng ta nên đặt tên nó dựa theo đường dẫn tương đối tới folder src. Một component ở src/screens/User/List,jsx sẽ có tên là ScreenUserList.

Cấu trúc file dự án của chúng ta cuối cùng sẽ như thế này:

src

├─ components

│ ├─ User

│ │ ├─ Form

│ │ │ ├─ Form.jsx

│ │ │ └─ Form.css

│ │ └─ List.jsx

│ └─ UI

│

└─ screens

├─ User

│ ├─ Form.jsx

│ └─ List.jsx

└─ Root.jsx

## **Tóm lại**

* ****Presentational**** và ****Container**** component đều được đặt trong src/components
* Nhóm components theo ****module/feature****.
* Đặt các component chung trong src/components/UI
* Giữ các ****screen**** ở mức đơn giản nhất, với cấu trúc và code tối thiểu.
* Nhóm các screen theo các route được định nghĩa. Với rote /users chúng ta sẽ có screen ở file /src/screens/User/List.jsx.
* Component được đặt tên dựa theo đường dẫn tương đối tới components hoặc src. Như vậy, một component ở file src/components/User/List.jsx sẽ có tên làUserList. Một component ở src/screens/User/List sẽ có tên là ScreenUserList.
* Nếu component ở trong folder cùng tên với nó thì không cần lặp lại trong tên component. Ví dụ, một component ở src/components/User/List/List.jsx sẽ có tên là UserList chứ ****KHÔNG**** phải là UserListList.