Components and Props

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components – Các components cho phép bạn chia UI thành các phần độc lập, tái sử dụng, và suy nghĩ về từng phần một cách riêng biệt. Trang này cung cấp một giới thiệu về ý tưởng của các components.

You can find a detailed component API reference here

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called “props”) and return React elements describing what should appear on the screen. – Về mặt khái niệm, các components như các function JS. Chúng chấp nhận các đầu vào tùy ý (được gọi là “props”) và trả về React elements mô tả những gì sẽ xuất hiện trên màn hình.

Function and Class Components

The simplest way to define a component is to write a JavaScript function: - Các đơn giản nhất để định nghĩa một component là viết một JS function:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

This function is a valid React component because it accepts a single “props” (which stands for properties) object argument with data and returns a React element. We call such components “function components” because they are literally JavaScript functions. – Hàm này là một component React hợp lệ vì nó chấp nhận một đối số object “props” (viết tắt của properties) với dữ liệu và trả về một phần tử React. Chúng tôi gọi các components như vậy là các “function components” bởi chúng thực sự là các hàm JavaScript.

You can also use an ES6 class to define a component – Bạn cũng có thể sử dụng ES6 class để định nghĩa một component.

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

The above two components are equivalent from React's point of view. – Hai thành phần trên là tương đương nhau theo quan điểm của React.
Function and Class components both have some additional features that we will discuss in the next sections. – Cả hai Function và Class components đều có các chức năng bổ sung mà chúng ta sẽ thảo luận ở phần tiếp theo.

Rendering a Component

Previously, we only encountered React elements that represent DOM tags – Trước đây, chúng ta chỉ gặp các phần tử React đại diện cho các thẻ DOM:

```
const element = <div />;
```

However, elements can also represent user-defined components: - Tuy nhất, các elements cũng có thể đại diện cho các components do người dùng tự định nghĩa.

```
const element = <Welcome name="Sara" />;
```

When React sees an element representing a user-defined component, it passes JSX attributes and children to this component as a single object. We call this object "props". – Khi React nhìn thấy một element đang đại diện cho một components do người dùng tự định nghĩa, nó pass JSX attribute và phần tử con tới component đó như một object duy nhất. Chúng tôi gọi object đó là "props"

For example, this code renders "Hello, Sara" on the page:

```
function Welcome(props) {  return <h1>Hello, {props.name}</h1>;
}
const element = <Welcome name="Sara" />;
ReactDOM.render(element, document.getElementById('root'));
```

Let's recap what happens in this example: - Tóm tắt những gì đã xảy ra ở ví dụ trên:

1. We call `ReactDOM.render()` with the `<Welcome name="Sara" />` element. – Chúng ta call ReactDOM.render() với element <Welcome name="Sara" />

2. React calls the `Welcome` component with `{name: 'Sara'}` as the props. – React call tới component Welcome với object { name: 'Sara' } ở props
3. Our `Welcome` component returns a `<h1>Hello, Sara</h1>` element as the result. – component Welcome của chúng ta trả về một element `<h1>Hello, Sara</h1>` như một kết quả sau cùng.
4. React DOM efficiently updates the DOM to match `<h1>Hello, Sara</h1>`. – React DOM update hiệu quả DOM để match `<h1>Hello, Sara</h1>`.`<h1>Hello, Sara</h1>`.

Note: Always start component names with a capital letter. – Lưu ý: Luôn luôn bắt đầu tên các component với một chữ hoa.

React treats components starting with lowercase letters as DOM tags. For example, `<div />` represents an HTML div tag, but `<Welcome />` represents a component and requires `Welcome` to be in scope. – React xử lý các components bắt đầu chữ thường dưới dạng DOM tags. Ví dụ, đại diện cho một HTML div tag, nhưng <Welcome /> đại diện cho một component và yêu cầu Welcome phải nằm trong phạm vi.

To learn more about the reasoning behind this convention, please read JSX In Depth. – Để tìm hiểu thêm về lý do đằng sau quy ước này, hãy đọc JSX In Depth.

Composing Components

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. A button, a form, a dialog, a screen: in React apps, all those are commonly expressed as components. – Các components có thể tham chiếu với các components khác trong đầu ra của chúng. Điều này cho phép chúng tôi sử dụng cùng một abstraction cho bất kỳ mức độ chi tiết nào. Button, biểu mẫu, dialog, screen: trong React apps, tất cả những thứ đó thường được biểu thị dưới dạng các components.

For example, we can create an `App` component that renders `Welcome` many times: - Ví dụ, chúng tôi có thể tạo một App component để nó render Welcome nhiều lần.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
function App() {
  return (
    <div>
      <Welcome name="Sara" />
      <Welcome name="Cahal" />
      <Welcome name="Edite" />
```

```
</div>
  );
}

ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Typically, new React apps have a single `App` component at the very top. However, if you integrate React into an existing app, you might start bottom-up with a small component like `Button` and gradually work your way to the top of the view hierarchy. – Thông thường, các React apps mới có một App component duy nhất tại tầng cao nhất trên cùng. Tuy nhiên, nếu bạn tích hợp React vào trong một app đã tồn tại, bạn có thể bắt đầu từ dưới lên với một component như Button và dần dần hoạt động theo cách của bạn lên tới đầu hệ thống phân cấp.

Extracting Components – Giải nén các Components

Don't be afraid to split components into smaller components. – Đừng ngại chia các components thành các components nhỏ hơn.

For example, consider this `Comment` component:

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <img className="Avatar" src={props.author.avatarUrl}
                alt={props.author.name}
        />
        <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

It accepts `author` (an object), `text` (a string), and `date` (a date) as props, and describes a comment on a social media website. – Nó chấp nhận author (một object), text (một string), và date (một date) dưới dạng props, và mô tả một comment trên một mạng xã hội.

This component can be tricky to change because of all the nesting, and it is also hard to reuse individual parts of it. Let's extract a few components from it. – Component này có thể khó khay đổi vì tất cả các phần chúng lồng vào nhau và cũng khó sử dụng lại các thành phần riêng lẻ của nó. Hãy trích xuất một vài thành phần từ nó.

First, we will extract `Avatar`:

```
function Avatar(props) {
  return (
    <img className="Avatar"
         src={props.user.avatarUrl}
         alt={props.user.name}
    />  );
}
```

The `Avatar` doesn't need to know that it is being rendered inside a `Comment`. This is why we have given its prop a more generic name: `user` rather than `author`. – Avatar không cần biết rằng nó đang được render bên trong một Comment. Đây là lý do tại sao chúng tôi đã đặt cho phần prop của nó một cái tên chung chung hơn: user thay vì author.

We recommend naming props from the component's own point of view rather than the context in which it is being used. – Chúng tôi khuyên bạn nên dặt tên props theo quan điểm riêng của component thay vì ngữ cảnh mà nó đang được sử dụng.
We can now simplify `Comment` a tiny bit: - Bây giờ chúng ta đã có thể đơn giản hóa Comment đi một chút:

```
function Comment(props) {
  return (
    <div className="Comment">
      <div className="UserInfo">
        <Avatar user={props.author} />
      <div className="UserInfo-name">
          {props.author.name}
        </div>
      </div>
      <div className="Comment-text">
        {props.text}
```

```
      </div>
      <div className="Comment-date">
        {formatDate(props.date)}
      </div>
    </div>
  );
}
```

Next, we will extract a `UserInfo` component that renders an `Avatar` next to the user's name: - Tiếp theo, chúng tôi sẽ nén component UserInfo render Avatar bên cạnh tên người dùng:

```
function UserInfo(props) {
  return (
    <div className="UserInfo">
        <Avatar user={props.user} />
        <div className="UserInfo-name">
                {props.user.name}
        </div>
    </div>
  );
}
```

This lets us simplify `Comment` even further: - Điều này cho phép chúng tôi đơn giản hóa Comment component hơn nữa:

```
function Comment(props) {
  return (
    <div className="Comment">
        <UserInfo user={props.author} />
        <div className="Comment-text">
                {props.text}
        </div>
        <div className="Comment-date">
                {formatDate(props.date)}
        </div>
    </div>
  );
}
```

Extracting components might seem like grunt work at first, but having a palette of reusable components pays off in larger apps. A good rule of thumb is that if a part of your UI is used several times (`Button`, `Panel`, `Avatar`), or is complex enough on its own (`App`, `FeedStory`, `Comment`), it is a good candidate to be extracted to a separate component. – Việc nén các component thoạt đầu có vẻ như không hoạt động, nhưng

việc có một bảng các component có thể tái sử dụng sẽ mang lại hiệu quả trong các ứng dụng lớn hơn. Một nguyên tắc chung là nếu một phần của giao diện người dungf của bạn được sử dụng nhiều lần (Button, Panel, Avatar), hoặc tự nó đủ phức tạp (App, FeedStory, Comment), thì nó là một ứng cử viên tốt để được nén vào thành một component riêng biệt.

Props are Read-Only

Whether you declare a component as a function or a class, it must never modify its own props. Consider this `sum` function: - Cho dù bạn khai báo một component dưới dạng một function hay một class, nó không bao giờ sửa đổi được props. Hãy xem hàm sum này:

```
function sum(a, b) {
  return a + b;
}
```

Such functions are called "pure" because they do not attempt to change their inputs, and always return the same result for the same inputs. – Các functions như vậy được gọi là "thuần túy" bởi vì chúng không cố gắng thay đổi các input của mình và luôn trả về cùng một kết quả cho các đầu vào giống nhau.
In contrast, this function is impure because it changes its own input: - Ngược lại, hàm này là không thuần túy vì nó thay đổi đầu vào của chính nó:

```
function withdraw(account, amount) {
  account.total -= amount;
}
```

React is pretty flexible but it has a single strict rule: - React khá linh hoạt nhưng nó có một quy tắc nghiêm ngặt duy nhất:
All React components must act like pure functions with respect to their props. – Tất cả các React components cần phải hoạt động giống như các pure function đối với các props của chúng.
Of course, application UIs are dynamic and change over time. In the next section, we will introduce a new concept of "state". State allows React components to change their output over time in response to user actions, network responses, and anything else, without violating this rule. – Tất nhiên, giao diện người dùng ứng dụng rất động và thay đổi theo thời gian. Trong phần tiếp theo, chúng tôi sẽ giới thiệu khái niệm mới về "state". State cho phép các thành phần React thay đổi đầu ra của chúng theo thời gian để phản hồi lại các hành động của người dùng, phản hồi network và bất kỳ thứ gì khác mà không vi phạm quy tắc ở trên.