**ReactJS**

* Lợi ích của reactJS
* Tạo ra những component có thể dùng lại, dễ bảo trì, phát triển. Giúp cấu trúc code tốt hơn, xây dựng những ứng dụng lớn.
* Thay đổi DOM hiệu quả? Virtual DOM, DOM đc lưu trong bộ nhớ. Giống như cache, khi request, do DOM nằm trong JS nên browser ko cần lấy html từ server? VIệc thay đổi DOM (nếu phức tạp) sẽ đc tính toán tốt nhất
* ReactJS là thư viện js dùng để xây dựng giao diện người dùng.
* Component
* Khi dữ liệu thay đổi, React sẽ tự động gọi hàm render để update lại UI
* JSX: Javascript Syntax Extension
* Props, State: 2 kiểu data của react
  + State: private, sử dụng trong bản thân component, có thể thay đổi, dựa trên tương tác người dùng
  + Props: external, có thể truyền từ component cha xuống component con, ko thể thay đổi
* Inverse data flow
* Ref, React.findDOMNode(component)

**Tutorial**

* React có 1 số loại Component khác nhau.
* Component cho react biết cái cần hiển thị
* Component nhận vào các tham số, gọi là props, trả về các html thông qua phương thức render
* Phương thức render trả về 1 mô tả, react sẽ nhận mô tả và hiển thị ra giao diện
* Khi this.setState đc gọi, react sẽ update lại UI
* Khi muốn lấy thông tin từ các component con, hoặc các component con mún trao đổi thông tin, thì nên chuyển thông tin đó lên component cha.
* Giả sử 1 state của 1 component là 1 obj, theo nguyên tắc thì ko nên sửa vào obj đó, mà tạo 1 obj khác, sửa giá trị, rùi gán lại cái state. Khi đó, react chỉ cần so sánh tham chiếu là biết state đã thay đổi chứ ko cần so sánh từng giá trị
* Gán key phù hợp cho các thành phần (li) khi tạo danh sách (ul, ol)

**JSX: Javascript Syntax Extension**

* Jsx: cách mô tả UI, nhìn giống html

Const a = <p>keep learning</p>;

* Embedding expressions in JSX: dùng dấu {} để đưa biểu thức js vào jsx
* Sau khi compile, JSX là 1 đối tượng trong js, do đó, có thể dùng nó với các lệnh return, if, for, …. Babel sẽ chuyển jsx thành obj bằng lệnh React.createElement(), những obj như vậy gọi là React elements, mô tả cho cái mà mình muốn hiển thị. React đọc các obj đó và chuyển qua DOM
* Tạo attribute: dùng “” (cho string), hoặc {} (cho biểu thức), JSX qui ước dùng camelCase khi viết tên attr, như className, tabIndex, …
* Nếu 1 tag ko có tag đóng thì dùng /> cuối tag như XML
* JSX tự động chống injections, xss, tự động escape

**Rendering Elements**

* Có 1 thẻ div mà react sẽ quản lý mọi thứ ở trong nó, thẻ div đó gọi là root DOM node
* ReactDOM.render()
* Element là bất biến, ko thể thay đổi các tag, attr, text của nó, nếu muốn thì phải ra element mới và cập nhật lại. React chỉ cập nhật những cái thay đổi so với trc đó

**Components and Props**

* Component là những phần mô tả UI độc lập, có thể đc sử dụng lại
* Component giống như hàm js, nhận tham số truyền vào props, và trả về react elements
* 2 cách tạo component: hàm và class
* Element đại diện cho components: tên tag là tên của component.

Const elem = <Hello name=”thanh” />;

* Khi react gặp element như vậy, nó sẽ truyền props, lấy từ attr của element (name), cho component mà element đại diện
* Luôn bắt đầu tên component với chữ in hoa, như Welcome
* Trong output của component (element) có chứa component #
* Component phải chỉ trả về 1 element (1 tag), do đó nên bọc output bằng 1 thẻ div
* Đặt tên props từ góc độ của chính component chứ ko phải trong bối cảnh render nó ra
* Ko đc chỉnh sửa props trong component

**State and Lifecycle**

* Mounting: hoạt động thực hiện sau khi Component đc render
* Unmounting: hoạt động thực hiện sau khi Component bị hủy
* Lifecycle hooks:
  + ComponentDidMount: mounting
  + ComponentWillUnmount: unmounting
* Khi gọi hàm setState, react sẽ gọi hàm render lại
* setState:
  + ko sửa trực tiếp state mà phải dùng hàm setState thì react mới render lại. Chỗ duy nhất để gán giá
  + state, props có thể đc cập nhật bất đồng bộ, nên khi dùng state, props trc đó thì nên dùng preState

this.setState((prevState, props) => {…});

* + State mới sẽ đc merge vào obj state
* State đc coi như biến local
* Component cha có thể truyền data cho component con, ngc lại thì ko đc (?), đó gọi là luồng top-down hay hình cây
* Tưởng tượng cây component như là dòng thác các props, các state là nguồn nc thêm vào dòng chảy đó

**Handling Events**

* Để preventDefault thì dùng e.preventDefault() chứ ko dùng return false;
* React hỗ trợ cho nhiều trình duyệt khác nhau nên ko cần quan tâm điều đó
* Ko cần sử dụng addEventListner
* Sử dụng nhiều arrow function trong event handler sẽ làm giảm performance, vì mỗi lần render lại thì 1 callback khác sẽ đc tạo ra (có phải là vì arrow function ko có tên, nên ko thể gọi lại tên của nó (địa chỉ func))

**Conditional Rendering**

* Cấu trúc điều khiển
* Hiện thị component dựa vào điều kiện
* Sử dụng “if” hoặc “? :”
* Sử dụng biến để lưu component:

If (a) {

X = <A />

} else {

X = <B />

}

Render() {

<C />

{X}

}

* Dùng && trong JSX: {a > 0 && <p>thanh dep trai</p>}

Nếu thỏa đk thì element đằng sau sẽ hiển thị

* Dùng {conditions ? true : false}
* Nếu component trả về null (return null) thì sẽ ko render component đó
* , các phương thức lifecycle

**Lists and Keys**

* Các element có thể nằm trong 1 mảng (list)
* Khi render list đó ra thì sẽ render tuần tự các element ra
* **Keys:**
  + Key giúp react pik đc phần nào thay đổi, thêm vào, bỏ đi
  + Chỉ ra các khi tạo mảng (ví dụ dùng với hàm map)
  + Chỉ khác nhau trong cùng ngữ cảnh (cùng array, các <li> cùng <ul>)
  + Key ko thể truyền vào component như props

**Forms**

* Những thành phần form có sẵn những state
* Controlled Component:
  + Là những thành phần (form) mà giá trị của nó đc điều khiển bởi react
  + Viết 1 js để lấy data user nhập vào form và submit form
  + Đồng bộ giữa các value của form với các state của component
* Những thành phần <input>, <select>, … duy trì state của chính nó và update khi user nhập liệu
* **Textarea:** Dùng value thay vì viết text thuần giữa tag
* **Select:** dùng attr value={this.state.value} ở tag select thay vì dùng selected

**Lifting State Up**

**Composition vs Inheritance**

* Để sử dụng lại code giữa các component thì nên dùng composition thay vì inheritance
* Composition:
  + Khi tạo 1 component, trong đó có chỗ render từ props.children của component đó thì khi sử dụng, ta dùng 2 tag đóng, mở cho component đó,

<Com>

<p>haha</p>

</Com>

Ở giữa chính là props.children

* **Specialization**: 1 component là trường hợp đặc biệt của component khác, như WelcomeDialog là trường hợp đặc biệt của Dialog.
  + Dialog là component với các props như title, message, …. WelcomeDialog sẽ dùng Dialog với các props cụ thể

**Thinking in React**

* Nên giữ số lượng state ít nhất có thể.
* Để xác định dữ liệu đó có phải state, cần xác định nó có phải đc nhận từ props của parent, có thay đổi theo thời gian, có thể tính đc từ các props, state khác.

**JSX in Depth**

* JSX là cách viết ngắn gọn cho React.createElement(Component, props, …children)
* Thư viện react phải có cùng phạm vi với jsx
* Định nghĩa 1 obj chứa nhiều component, khi export thì chỉ cần export obj đó.
* Phải viết hoa component mà mình định nghĩa
* Ko thể dùng công thức trong tên component khi gọi component, phải gán vào 1 biến viết hoa trước

*const components = {*

*photo: PhotoStory,*

*video: VideoStory*

*};*

*function Story(props) {*

*// Wrong! JSX type can't be an expression.*

*return <components[props.storyType] story={props.story} />;*

*}*

* Khi truyền props là string thì react sẽ ko escape

*<MyComponent message="&lt;3" />*

*<MyComponent message={'<3'} />*

* Nếu ko truyền gì cho props, mặc định nó là true

*<MyTextBox autocomplete />*

* Spread attributes: dùng <A …someObj /> để truyền prop cho component
* Props.children có thể là 1 hàm
* Những giá trị bool, null, undefined sẽ ko đc render

**Typechecking With PropTypes**

* React.PropTypes đã lỗi thời, nên dùng thư viện prop-types
* Kiểm tra kiểu của props đưa vào component

*import PropTypes from 'prop-types';*

*class Greeting extends React.Component {*

*render() {*

*return (*

*<h1>Hello, {this.props.name}</h1>*

*);*

*}*

*}*

*Greeting.propTypes = {*

*name: PropTypes.string*

*};*

* Chỉ nên dùng ở mode develop, để ko ảnh hưởng đến performance
* isRequired: kiểm tra prop có đc đưa vào ko
* Component.defaultProps: giá trị mặc định cho prop, kiểm tra loại áp dụng cho cái này luôn

**Refs and the DOM**

* Trường hợp cần dùng refs:
  + Focus, chọn text, media playback
  + Trigger imperative animation
  + Dùng thư viện DOM của người #
* Tránh sử dụng refs
* React hỗ trợ 1 đặc tính ref cho các component, là 1 callback đc gọi sau khi component đc render hoặc sau khi component bị hủy
* Dùng ref để tạo property của component tham chiếu đến DOM

**Uncontrolled Components**

* Controlled Components: form đc kiểm soát bởi react
* Nên dùng controlled components, tránh uncontrolled

**Optimizing Performance**

* Sử dụng phiên bản production của react

**Redux**

* Sự phức tạp khó giải quyết là vì ta trộn lẫn (nhầm lẫn?) 2 khái niệm: biến đổi và bất đồng bộ.

**Reducers**

* Coi obj state như 1 db, đừng lồng quá nhiều cấp.
* Những thứ ko đc làm trong reducer:
  + Biến đổi tham số
  + Gọi API,
  + Gọi các hàm non-pure: Date.now(), Math.random()
* Reducer chỉ có nhiệm vụ tính toán
* Reducer nhận 2 tham số là prevState và action, lần gọi đầu tiên thì prevState là undefined, ta sẽ trả về giá trị state ban đầu trong lần gọi này, hoặc dùng giá trị mặc định
* Cho những action ko pik, ta nên trả về prevState.
* Đ/v app lớn thì nên chia nhỏ reducer ra, reducer lớn 1 sẽ truyền cho các reducer nhỏ 1 phần của state, các reducer nhỏ sẽ chỉ cập nhật phần đó. Gọi là reducer composition
* Const mainReducer = combineReducers({a, b})

= function mainReducer(state = {}, action) {

Return {

a: a(state.a, action),

b: b(state.b, action)

}

}

* Có thể chỉ ra những hàm, key khác trong combineReducers()

**Store**

* Store là 1 obj nối action, reducer, có những nhiệm vụ sau:
  + Giữ state của ứng dụng
  + Cho phép lấy state = getSTate()
  + Cho phép thay đổi state = dispatch(action)
  + Đăng ký listener = subscribe(listener)
  + Xử lý hủy đăng ký những listener qua hàm trả về từ subscribe
* Chỉ có duy nhất 1 store trong ứng dụng
* createStore(reducer, state): tham số state là tùy chọn, chỉ ra giá trị state ban đầu (nếu ko có là undefined?)
* a = store.subscribe(callback): callback sẽ đc gọi khi store đc update. Muốn ngừng lắng nghe thì gọi a().

**Data Flow**

* Vòng data trong redux trải qua 4 bước:
  + Gọi store.dispatch(action)
  + Redux sẽ gọi reducer mà mình truyền vào
  + Reducer trả về kết quả
  + Redux lưu lại state mới đó
* Pure Function: gọi hàm đó nhiều lần với cùng 1 input thì output trả về ko thay đổi

**Usage with React**

* Redux có thể đc dùng chung với React, Angular, Ember, jQuery, JS thuần
* **Presentational and Container Components**
* --------------------------------------
* Chia các component thành 2 loại: presentational, container
  + **Presentational**: hình dạng. có thể chứa presentational, container. Có style riêng. Containment, this.props.children. Ko có thuộc vào phần còn lại của app, như store. Ko chỉ ra cách data đc tải hay thay đổi. Nhận data, callback duy nhất thông qua props. Hiếm khi có state. Đc viết theo kiểu function. VD: Page, SideBar, UserInfo
  + **Container**: cách thức. có thể chứa presentational, container. Ko có style, có thể có DOM mark up. Cung cấp data, callback cho presentational, container #. Gọi action. Stateful, như 1 data source. Đc tạo bởi những component cấp cao như connect()
* **Khi nào sử dụng container:**
* Khi tạo app chỏ với các presentational components, mình sẽ phải truyền nhiều props qua các component trung gian, 1 số component sẽ chỉ chuyển tiếp props chứ ko sử dụng, và nếu component con cần nhiều data hơn thì ta phải chỉnh sửa lại component trung gian
* Dùng container component
* ----------------------------------------
* Hầu hết các component sẽ là presentational, các container component sẽ kết nối đến redux store
* Có thể dùng store.subscribe() để tạo container, nhưng ko nên. Nên dùng connect().