REACTJS

Ngo Thuc Dat

Brief

1. Tổng quan về REACTJS
2. Thư viện ReactJS
3. ReactJS Core

Tổng quan về ReactJS

- ReactJS là gì?
- Tại sao dùng ReactJS
- Ai đang dùng ReactJs
- Xu thế công nghệ và ReactJS

Tổng quan về ReactJS

- Là một bộ thư viện do Facebook phát triển từ năm 2013 và Open-source từ năm 2015
- Được build từ JavaScript - ngôn ngữ phổ biến nhất thế giới
- Tư tưởng cải tiến, cho phép phá vỡ giao diện thành các “Component” - một khái niệm nổi bật của ReactJS - Tránh lại tải trang
- Sử dụng Dom Ảo cùng quản lý state giúp tối ưu performance cho ứng dụng web, kết hợp với việc xử lý các use-case về mặt UX - cảm nhận trực tiếp của người dùng ảnh hưởng đến sự tồn vong của một ứng dụng
- Sử dụng JSX - giúp lập trình viên viết code dễ dàng đưa trực tiếp HTML code vào javascript

Tổng quan về ReactJS

- ReactJS tối ưu hóa được mặc UI - giảm tải cho server - cải thiện được mặc UX - tăng tỉ lệ chuyển đổi các khách hàng
- ReactJS được ứng dụng lập trình rộng rãi với hàng ngàn websites thuộc các lĩnh vực khác nhau, bao gồm Lazada, Shopee, Sendo, Bamboo Airways, Instagram and Airbnb và dĩ nhiên là Facebook.
- ReactJS một xu hướng của công nghệ hiện đại trong làng web-application
- Thị trường cầu rất lớn - hiện tại nguồn cung rất hạn chế => cơ hội việc làm rất nhiều
- Cần nắm bắt cơ hội - đi trước đón đầu

Thư viện ReactJS

- Cnd links: https://reactjs.org/docs/cdn-links.html
- Bản chất reactjs là một thư viện javascript giống như các thư viện khác trong javascript (chẳng hạn JQuery) nên phải cần nhiều thư viện và plugin khác để tạo thành một framework
- Cách dùng đơn giản là nhúng file url cdn vào html file
- Chương trình hello world đầu tiên với library ReactJS

```
<head>
    <meta charset="UTF-8">
    <script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
</head>
```

Hello world với reactjs cdn

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
</head>
<body>
    <div id="app"></div>
    <script src="index.js"></script>
</body>
</html>
```

```js
var element = React.createElement('div', null, 'Hello world')
ReactDOM.render(
    element,
    document.getElementById("app")
)
```

Giới thiệu về JSX

- JSX là gì?
  Javascript Extension
- Tại sao JSX ra đời
  ReactJS không bắt buộc dùng JSX, nhằm giúp developer có cách nhìn UI trực quan khi viết code Javascript cũng những thấy message liên quan đến warning và error
- Ví dụ về JSX - Giới thiệu về babel
  - Try it out: https://babeljs.io/repl
  - Cdn link: https://cdnjs.com/libraries/babel-standalone
- Các kỹ thuật cơ bản và nâng cao với JSX

JSX Example

```
var element = <div> Hello world </div>
// var element = React.createElement('div', null, 'Hello world')
ReactDOM.render(
    element,
    document.getElementById("app")
)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <script crossorigin src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@16/umd/react-dom.development.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.26.0/babel.js"></script>
</head>
<body>
    <div id="app"></div>
    <script type="text/babel" src="index.js"></script>
</body>
</html>
```

Embedding Expressions in JSX

```
const name = 'Josh Perez';
const element = <h1>Hello,
{name}</h1>;
```

Render element - Class render()

- **React.Component**
  Là một class được định nghĩa của ReactJS, một instance được kế thừa sẽ mang thuộc tính và phương thức của Component

- **render()**
  Là một phương thức của component, hàm sẽ trigger render lên browser

```
class App extends React.Component {
    render() {
        return (
            <div>
                <h1> Hello class </h1>
            </div>
        )
    }
}

ReactDOM.render(<App/>,
document.getElementById("app"))
```

Render element - ReactDOM.render()

React Only Updates What's Necessary

```
function tick() {
  const element = (
    <div>
      <h1>Hello, world!</h1>
      <h2>It is {new Date().toLocaleTimeString()}.</h2>
    </div>
  );
  ReactDOM.render(element, document.getElementById("root"));
}

setInterval(tick, 1000);
```

https://reactjs.org/c158617ed7cc0eac8f58330e49e48224/granular-dom-updates.gif

State in a Component

- Là data trong một component và có full control bởi component (reactComponent)
- State được mapping với data khi render: state change => trigger render() => data hiển thị trên browser change
- Thay đổi state của component thông qua setState() - tránh immutable state

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
    this.timerID = setInterval(() => this.tick(),
1000);
  }

  tick() {
    this.setState({
      date: new Date(),
    });
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is
{this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(<Clock />,
document.getElementById("app"));
```

Event Handling

- Likely javascript event handler, we can catch event by user keyword type event in directly on tab (example below)
- Usage of bind
- Passing Arguments to Event Handlers

For example, the HTML:

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

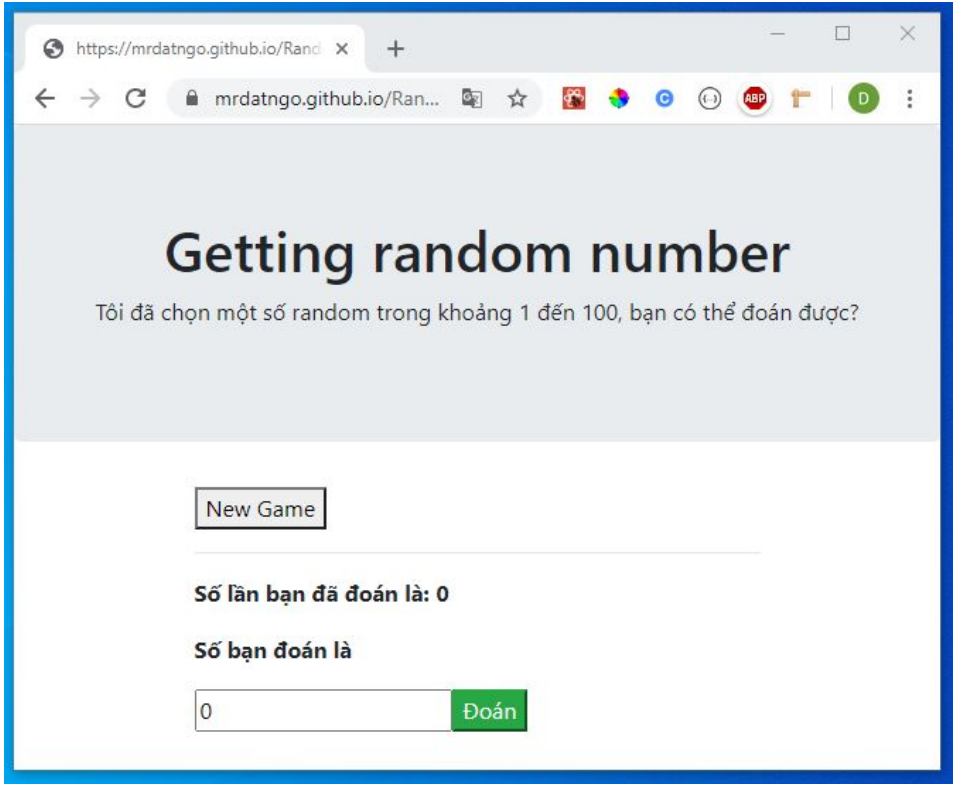
is slightly different in React:

```
<button onClick={activateLasers}>
 Activate Lasers
</button>
```

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isToggleOn: true };
    // This binding is necessary to make `this` work
in the callback
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState((state) => ({
      isToggleOn: !state.isToggleOn,
    }));
  }
  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? "ON" : "OFF"}
      </button>
    );
  }
}
ReactDOM.render(<Toggle />,
document.getElementById("app"));
```

Practice Random Game

https://mrdatngo.github.io/RandomGame/index.html



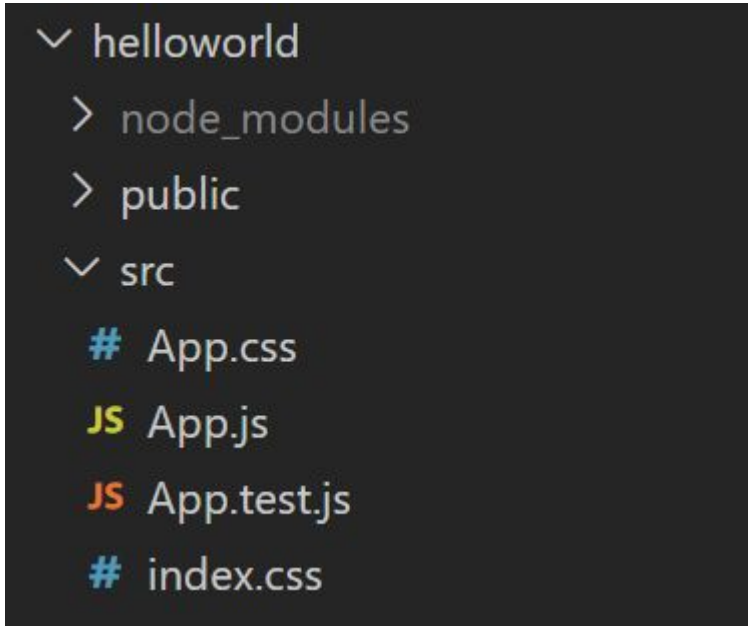
ReactJS with other libraries, plugins

To become a framework to development an app

Toolchain - yarn create react-app

Yarn create react-app helloworld

- We can create react-app with many cli
  - yarn create react-app project-name
  - npx create-react-app project-name
  - npm init react-app project-name
- Run app:
  - cd project-name
  - yarn start
- Explore structure pattern



Component LifeCycle

Adding Lifecycle Methods to a Class

- Trong reactjs app có rất nhiều component, vì vậy rất quan trọng trong việc giải phóng resource của component khi component bị destroyed
- Component render to DOM for the first-time: mounting
- Component removed from the DOM: unmounting
  => componentDidMount() {} && componentWillUnmount() {}

Clock Example:

```
componentDidMount() {
  this.timerID = setInterval(
    () => this.tick(),
    1000
  );
}
```

```
componentWillUnmount() {
  clearInterval(this.timerID);
}
```

Component props

- Props are Read-Only
- Reveal: pass arguments by value and pass arguments by references
- All React components must act like pure functions with respect to their props.

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

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

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

Event Handling - binding

- Likely javascript event handler, we can catch event by user keyword type event in directly on tab (example below)
- Usage of bind
- Passing Arguments to Event Handlers

For example, the HTML:

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

is slightly different in React:

```
<button onClick={activateLasers}>
 Activate Lasers
</button>
```

```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = { isToggleOn: true };
    // This binding is necessary to make `this` work in
the callback
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.setState((state) => ({
      isToggleOn: !state.isToggleOn,
    }));
  }
  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? "ON" : "OFF"}
      </button>
    );
  }
}
ReactDOM.render(<Toggle />,
document.getElementById("app"));
```

Conditional - advance

- Inline If with Logical && Operator

```
> true && 5 * 3
<· 15
> true && "Show if it's true"
<· "Show if it's true"
```

- Inline If-Else with Conditional Operator

```
> var condition = true; // false
  condition ? "condition is true":"condition is false"
<· "condition is true"
>
```

- Arrays.map()

```
> ['nguyen van a', 'nguyen van b'].map(name => {return { name: name }})
<· ▼(2) [{…}, {…}]
    ▶0: {name: "nguyen van a"}
    ▶1: {name: "nguyen van b"}
     length: 2
```

Các kỹ thuật cơ bản và nâng cao JSX

- Specifying Attributes with JSX

```
const element = <img src={user.avatarUrl} />;

<h1 className="red--color" style={{color: "red"}} > Hello world </h1>
```

- Specifying Children with JSX
  - If tag self close
  - If tag have children

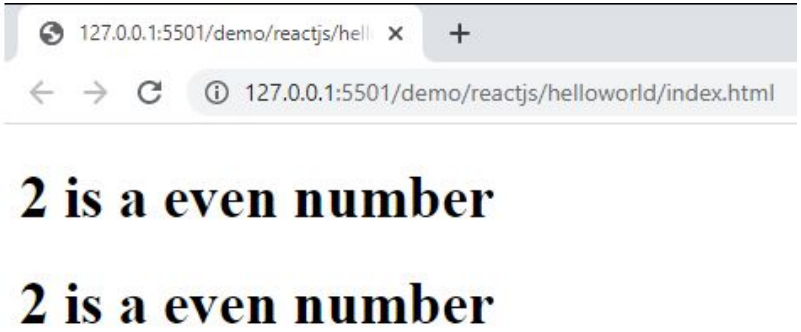
```
const element = <img src={user.avatarUrl} />

const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

Các kỹ thuật nâng cao JSX - If

- Embedding Expressions in JSX

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

- Thao tác với if/else



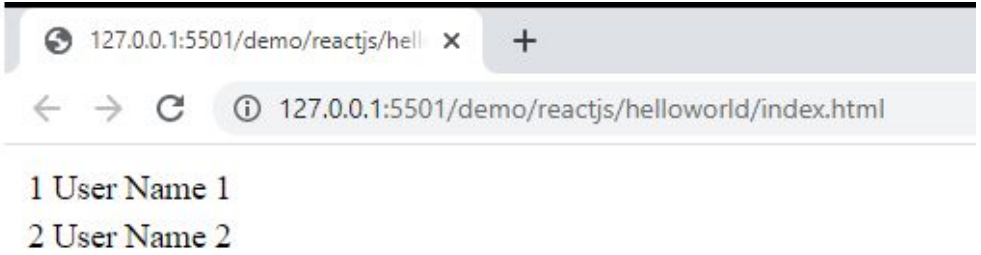

```
function checkElement(number) {
    if (number % 2 === 0) {
        return <h1> { number } is a even number </h1>
    } else {
        return <h1> { number } is a odd number </h1>
    }
}
var number = 2
const element = (
  <div>
    {checkElement(number)}
    { number % 2 === 0 ?
        <h1> { number } is a even number </h1>
        :
        <h1> { number } is a odd number </h1>
    }
  </div>
);
ReactDOM.render(element, document.getElementById("app"));
```

Các kỹ thuật nâng cao JSX - List

Thao tác với array

```
function getRowsElement(users) {
  var rows = [];
  for (var i = 0; i < users.length; i++) {
    rows.push(
      <tr>
        <td>{users[i].id}</td>
        <td>{users[i].name}</td>
      </tr>
    );
  }
  return rows;
}
var users = [
  {
    id: 1,
    name: "User Name 1",
  },
  {
    id: 2,
    name: "User Name 2",
  },
];
var element = <table>{getRowsElement(users)}</table>;
console.log(element)
ReactDOM.render(element,
document.getElementById("app"));
```

```
var users = [...];
var element = (
  <table>
    <tbody>
      {users.map((user) => (
        <tr key={user.id}>
          <td>{user.id}</td>
          <td>{user.name}</td>
        </tr>
      ))}
    </tbody>
  </table>
);
console.log(element);
ReactDOM.render(element,
document.getElementById("app"));
```



Lists and Keys

- Keys giúp React xác định item nào thay đổi, item nào added hoặc removed.
- Keys nên được đưa vào elements inside a array với stable identity
- Cách tốt để chọn key là dùng một chuỗi định danh duy nhất trong list item trong số các element siblings (anh chị em gần kề).
- Nếu không có key ổn định thì có thể dùng index trong map
- Key chỉ cần unique trong siblings.

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>
    {number}
  </li>
);
```

Forms

- Different with handle html form
  - Html form: use default onsubmit form
  - React form: handle by control
- Control Component
- Uncontrol Component

```
<form>
  <label>
    Name: <input type="text" name="name" />
  </label>
  <input type="submit" value="Submit" />
</form>
```

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = { value: "" };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }

  handleChange(event) {
    this.setState({ value: event.target.value });
  }

  handleSubmit(event) {
    alert("A name was submitted: " + this.state.value);
    event.preventDefault();
  }

  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input
            type="text"
            value={this.state.value}
            onChange={this.handleChange}
          />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

Forms

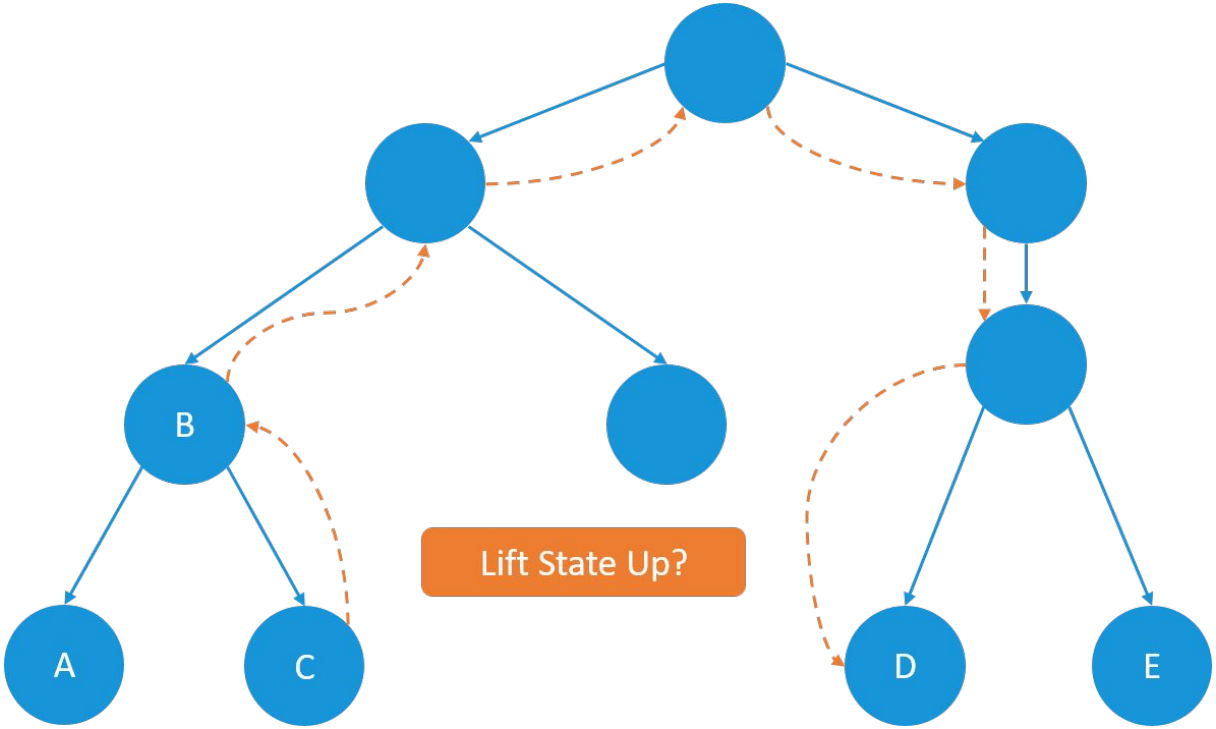
- <textarea></textarea>
- <select></select>
- checkbox

```
<form onSubmit={this.handleSubmit}>
  <label>
    Essay: <textarea value={this.state.value}
onChange={this.handleChange} />
  </label>
  <input type="submit" value="Submit" />
</form>
```

```
<form onSubmit={this.handleSubmit}>
  <label>
    Pick your favorite flavor:
    <select value={this.state.value} onChange={this.handleChange}>
      <option value="grapefruit">Grapefruit</option>
      <option value="lime">Lime</option>
      <option value="coconut">Coconut</option>
      <option value="mango">Mango</option>
    </select>
  </label>
  <input type="submit" value="Submit" />
</form>
```

```
<input
  name="isGoing"
  type="checkbox"
  checked={this.state.isGoing}
  onChange={this.handleInputChange} />
```

Lifting State up



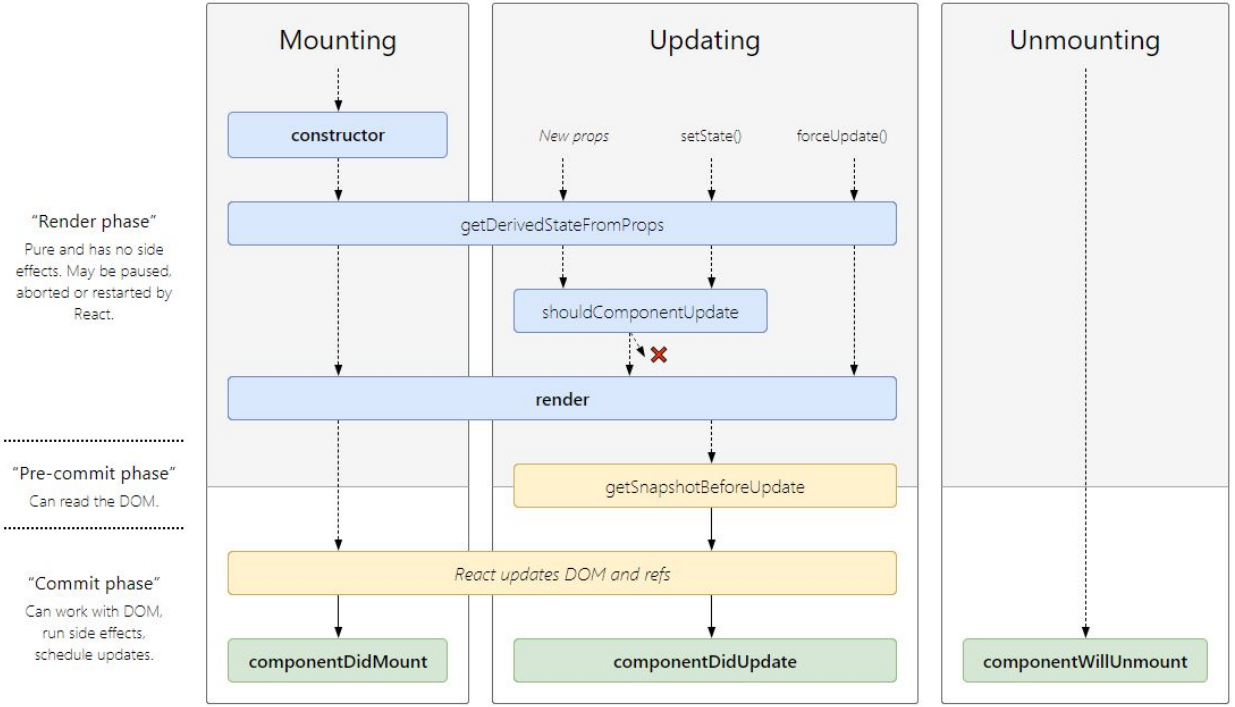
Composition vs Inheritance

```
function FancyBorder(props) {
  return (
    <div className={'FancyBorder FancyBorder-' +
props.color}>
      {props.children}
    </div>
  );
}

function WelcomeDialog() {
  return (
    <FancyBorder color="blue">
      <h1 className="Dialog-title">
        Welcome
      </h1>
      <p className="Dialog-message">
        Thank you for visiting our spacecraft!
      </p>
    </FancyBorder>
  );
}
```

```
function SplitPane(props) {
  return (
    <div className="SplitPane">
      <div className="SplitPane-left">
        {props.left}
      </div>
      <div className="SplitPane-right">
        {props.right}
      </div>
    </div>
  );
}

function App() {
  return (
    <SplitPane
      left={
        <Contacts />
      }
      right={
        <Chat />
      } />
  );
}
```

Component LifeCycle advance

https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/



Component LifeCycle advance

- The `render()` method is the only required method in a class component.
- constructor() : If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component. You should not call `setState()` in the `constructor() -` you should assign `this.state` directly
- Avoid copying props into state! This is a common mistake:

- You may call setState() immediately in componentDidMount(). It will trigger an extra rendering, but it will happen before the browser updates the screen
- componentDidUpdate() will not be invoked if shouldComponentUpdate() returns false.
- componentWillUnmount() is invoked immediately before a component is unmounted and destroyed. You should not call `setState()` in `componentWillUnmount()` because the component will never be re-rendered.
- Use `shouldComponentUpdate()` to let React know if a component's output is not affected by the current change in state or props. The default behavior is to re-render on every state change, and in the vast majority of cases you should rely on the default behavior.

LocalStorage - Definitions

- The localStorage and sessionStorage properties allow to save key/value pairs in a web browser.
- The localStorage object stores data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.
- The localStorage property is read-only.

LocalStorage - Syntax

- **window.localStorage**
- Syntax for SAVING data to localStorage:
  - **localStorage.setItem("key", "value");**
- Syntax for READING data from localStorage:
  - **var lastname = localStorage.getItem("key");**
- Syntax for REMOVING data from localStorage:
  - **localStorage.removeItem("key");**

Children

- Content in between tags Component will pass to Component is called through props with **props.children**

```
import React, { Component } from 'react'

export default class Childrens extends Component {
    render() {
        return (
            <div>
                <Child>
                    <h3>Child</h3>
                    <p>Bring me to Child</p>
                </Child>
            </div>
        )
    }
}

class Child extends Component {
    render() {
        return (
            <div>
                Child: { this.props.children }
            </div>
        )
    }
}
```

Class component & function component

- Ease to code and test
- Less code
- Don’t have setState() in your component
- Performance boost in the future versions

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

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

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

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

React.Memo & React.PureComponent

- Help define our **shouldComponentUpdate()** lifecycle
- Only render when state or props change

```
const Counter = React.memo(({ value,
children }) => {
    console.log("value: ", children)
    return (
        <div>
            {children}: {value}
        </div>
    )
})
```

```
class Counter extends React.PureComponent
{
    render() {
        return (
            <div>
                {this.props.children}
                :
                {this.props.value}
            </div>
        );
    }
}
```

REACT ROUTER DOM

- Library:
- Install:
  - npm install react-router-dom
  - yarn add react-router-dom

REACT ROUTER DOM

- BrowserRouter
- HashRouter
- Route
- Link
- Switch
- Redirect

```
<Router>
    <div>
        <nav>
            <ul>
                <li>
                    <Link to="/">Home</Link>
                </li>
                <li>
                    <Link to="/login">Login</Link>
                </li>
            </ul>
        </nav>
    </div>
    <Switch>
        <Route path="/login">
            ( isLogin ? <Redirect to="/" /> : <Login></Login> )
        </Route>
        <Route path="/">
            <Home></Home>
        </Route>
    </Switch>
</Router>
```

React Router DOM - Private Route

- All private link need a authen to load data
- Every Route need a authen should check authen before render data => Create Custom Route

```
export default function PrivateRoute({ children, ...rest }) {
    return (
        <Route
            {...rest}
            render={({ location }) => {
                return !FakeAuth.isAuthenticated ? (
                    <Redirect
                        to={{
                            pathname: "/login",
                            state: { location },
                        }}
                    />
                ) : (
                    children
                );
            }}
        />
    );
}

<PrivateRoute path="/protected">
    <ProctectedPage></ProctectedPage>
</PrivateRoute>
```

Lazy-loading

- **Note:**

  `React.lazy` and Suspense are not yet available for server-side rendering. If you want to do code-splitting in a server rendered app, we recommend Loadable Components.
- The lazy component should then be rendered inside a `Suspense` component, which allows us to show some fallback content

Before:

```
import OtherComponent from './OtherComponent';
```

After:

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));
```

```
import React, { Suspense } from 'react';

const OtherComponent = React.lazy(() => import('./OtherComponent'));
const AnotherComponent = React.lazy(() => import('./AnotherComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <section>
          <OtherComponent />
          <AnotherComponent />
        </section>
      </Suspense>
    </div>
  );
}
```

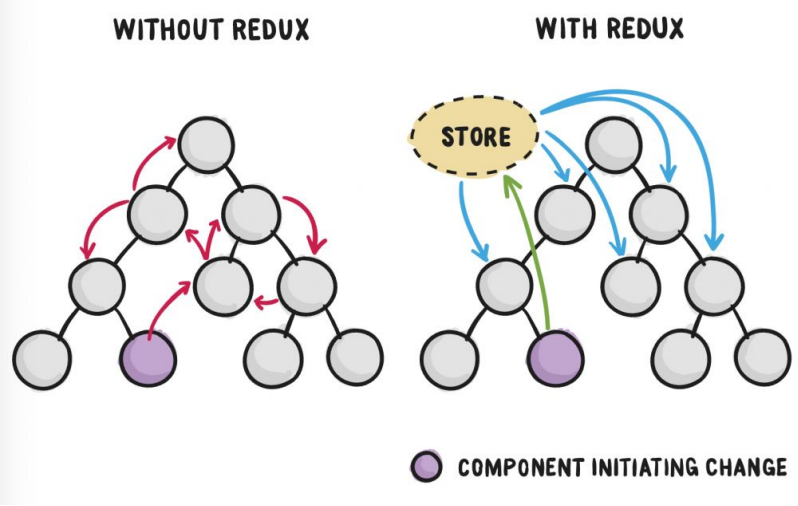
Axios - Config

- Set bearer token header

```
import axios from 'axios';

export default (token = null) => {
    if (token) {
        axios.defaults.headers.common.authorization = `Bearer ${token}`;
    } else {
        delete axios.defaults.headers.common.authoriztion;
    }
}
```

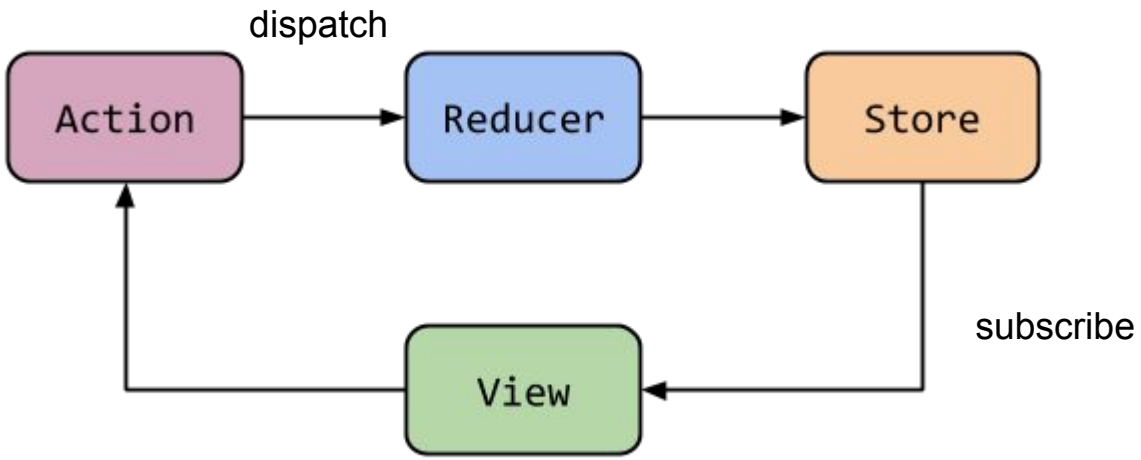
REDUX

- Data và UI state => lưu trong state tree (**store**) => State của ứng dụng sẽ **to ra** và **khó kiểm soát** => Tạo ra một nơi lưu trữ **data chung** và **đáng tin cậy**
- State **chỉ được phép đọc** và chỉ được thay đổi thông qua **Actions** => Mọi thay đổi sẽ được kiểm soát
- Gồm: getState(), dispatch(), subscribe()



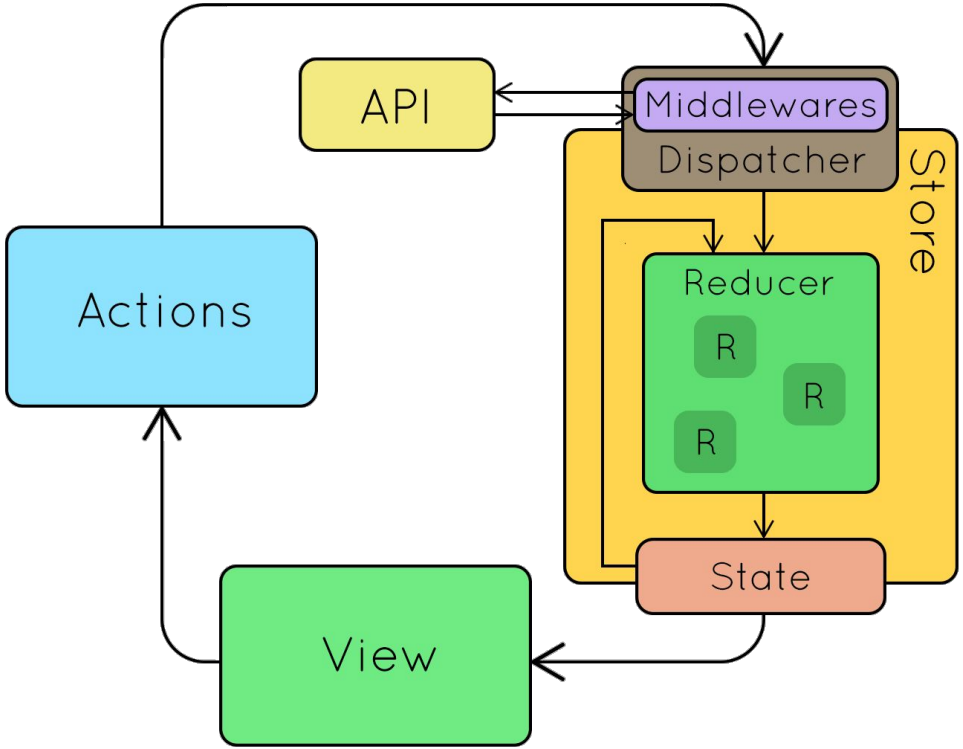
REDUX

- Diagram



REDUX - Event and data flow

- https://miro.medium.com/max/2880/1*QERgzuzphdQz4e0fNs1CFQ.gif



Redux - example

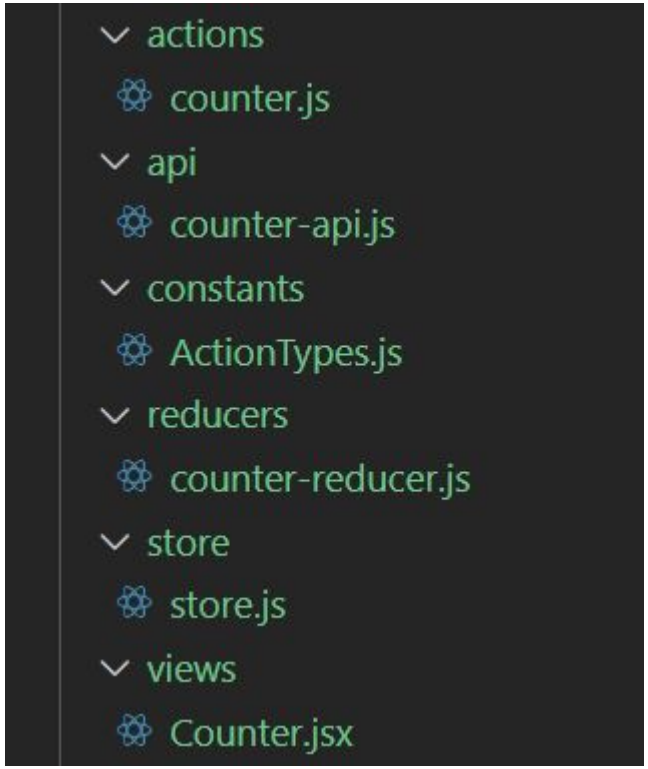
```
import React, { Component } from "react";
import { createStore } from "redux";

const counterReducer = (state, action = {}) =>
{
    switch (action.type) {
        case "INCREASE":
            console.log("HIHI");
            return state + 1;
        case "DECREASE":
            return state - 1;
    }
    return state;
};

let initialState = 0;

const store = createStore(counterReducer,
initialState);
```

```
class Counter extends Component {
    constructor(props) {
        super(props);
        this.state = {
            counter: 0,
        };
        store.subscribe(this.subcribeChange)
    }

    subcribeChange = () => {
        this.setState({ counter: store.getState() });
    }

    increase = () => {
        // this.setState({ counter: this.state.counter + 1 })
        store.dispatch({ type: "INCREASE" });
    };

    decrease = () => {
        // this.setState({ counter: this.state.counter - 1 })
        store.dispatch({ type: "DECREASE" });
    };

    render() {
        return (
            <div>
                Counter: {this.state.counter}
                <button
onClick={this.increase}>Increase</button>
                <button
onClick={this.decrease}>Decrease</button>
            </div>
        );
    }
}
export default Counter;
```

REDUX - IN PROJECT

- Project structures

src folder



REDUX - IN PROJECT

- actions/counter.js

```
import api from "../api/api";
import { INCREASE } from
"../constants/ActionTypes";

export const increasing = (user) => ({
    type: INCREASE,
    user,
});

export const increase = (data) => (dispatch) =>
    api.counter.getCounter(data).then((counter)
=> {
        return dispatch(increasing(counter));
    });
```

- api/counter-api.js

```
import axios from "axios";
export default {
    counters: {
        login: (data) =>
            axios
                .post(api gateway + "counter",
{ counter: data.username })
                .then((res) => {
                    return res.data;
                }),
    },
};
```

REDUX - IN PROJECT

- constants/ActionTypes.js

```
export const INCREASE = "INCREASE";
```

- reducers/counter-api.js

```
import { INCREASE } from
'../constants/ActionTypes';

export default function counter(state = {},
action) {
  switch (action.type) {
    case INCREASE:
      return { ...state, counter:
action.counter };
    default:
      return state;
  }
}
```

- store/store.js

```
import { createStore, applyMiddleware } from
'redux';
import thunk from 'redux-thunk';
import { composeWithDevTools } from
'redux-devtools-extension';
import reducerAll from '../reducers';

const store = createStore(reducerAll,
composeWithDevTools(applyMiddleware(thunk)));

export default store
```

REDUX - IN PROJECT

- views/Counter.js

```
import React, { Component } from 'react';
import { increase } from '../actions/counter';
import { connect } from 'react-redux';

class Counter extends Component {
    constructor(props) {
        super(props);
    }

    increase = () => {
        this.props.increase()
    };

    render() {
        return (
            <div>
                Counter: {this.state.counter}
                <button onClick={this.increase}>Increase</button>
            </div>
        );
    }
}

function mapStateToProps(state) {
    return {
        counter: state.counter
    }
}

export default connect(mapStateToProps, { increase })(Counter);
```

REDUX - IN PROJECT

- Adding provide to the top of component
- One app should only have one store

```
import store from './redux/store/store';
import { Provider } from 'react-redux';

const Components = () => {
  return (
    <Provider store={store}>
        <Route component={App} />
    </Provider>
  )
}
```

Typechecking With PropTypes

- Install:
  - Yarn add prop-types
  - Npm install prop-types
- Usage:
  - https://www.npmjs.com/package/prop-types

```
import PropTypes from 'prop-types';

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

Greeting.propTypes = {
  name: PropTypes.string
};
```

```
import PropTypes from 'prop-types';
MyComponent.propTypes = {
  // You can declare that a prop is a specific
JS type. By default, these
  // are all optional.
  optionalArray: PropTypes.array,
  optionalBool: PropTypes.bool,
  optionalFunc: PropTypes.func,
  optionalNumber: PropTypes.number,
  optionalObject: PropTypes.object,
  optionalString: PropTypes.string,
  optionalSymbol: PropTypes.symbol,

  // Anything that can be rendered: numbers,
strings, elements or an array
  // (or fragment) containing these types.
  optionalNode: PropTypes.node,

  // A React element.
  optionalElement: PropTypes.element,

  // A React element type (ie. MyComponent).
  optionalElementType: PropTypes.elementType,
  // An object taking on a particular shape
  optionalObjectWithShape: PropTypes.shape({
    optionalProperty: PropTypes.string,
    requiredProperty:
PropTypes.number.isRequired
  }),
};
```

Map Login Page With Redux

Authen Workflow

- Practice make an authen form with redux

HOOKS

- useState() https://reactjs.org/docs/hooks-state.html

```
import React, { useState } from 'react'

export default function UseState() {
    const [counter, setCounter] = useState(0)

    return (
        <div>
            <p>{ counter }</p>
            <button onClick={() => { setCounter(counter + 1)
}}>Increase</button>
        </div>
    )
}
```

HOOKS

- useEffect()
  https://reactjs.org/docs/hooks-effect.html

```
import React, { useState, useEffect } from "react";

export default function UseEffect() {
    const [counter, setCounter] = useState(0);
    useEffect(() => {
        document.title = `You click on title in ${counter} times`;
        console.log("Component before update");
        return () => {
            console.log("Component updated");
        };
    });

    // with useEffect()
    useEffect(() => {
        console.log("Component init");
        return () => {
            console.log("Component did mount / will unmount");
        };
    }, []);

    return (
        <div>
            <p>{counter}</p>
            <button onClick={() => { setCounter(counter + 1) }}>
                Increase
            </button>
        </div>
    );
}
```

HOOKS

- useContext()
  https://reactjs.org/docs/hooks-effect.html

```
const AppContext = React.createContext({
    state: {},
});

const StateProvider = ({ children }) => {
    const [state, dispatch] = useReducer(reducer, {
        theme: "dark",
    });
    return (
        <AppContext.Provider value={{state, dispatch}}>
            {children}
        </AppContext.Provider>
    );
};
const { state, dispatch } = useContext(AppContext);

<StateProvider>
    <MyComponent />
</StateProvider>
```

HOOKS

- useReducer()

```
function reducer(state, action) {
    console.log("Reducer: ", state)
    switch (action.type) {
        case "change":
            var newTheme = "light"
            if (state.theme == "light") {
                newTheme = "dark"
            }
            return { theme: newTheme };
        default:
            throw new Error();
    }
}
const StateProvider = ({ children }) => {
    const [state, dispatch] = useReducer(reducer, {
        theme: "dark",
    });
    return (
        <AppContext.Provider value={{state, dispatch}}>
            {children}
        </AppContext.Provider>
    );
};
```

HOOKS

- Custom hook

```
const useFormInput = (name) => {
    const [value, setValue] = useState(name);
    return {
        value,
        onChange: (event) => setValue(event.target.value),
    };
};

<Input
   // value={name}
   // onChange={(event) => setName(event.target.value)}
   {...name}
   name="name"
/>
const name = useFormInput("Harry");
```

Fragments and Refs

- <fragment></fragment> is the same as <></>
- Refs to do task that can’t not use state

```
constructor(props) {
    super(props);
    this.userNameInput = null;
}
```

```
this.userNameInput.focus();
```

```
<Input
    name="username"
    value={data.username}
    onChange={this.onDataChange}
    ref={(input) => {
        this.userNameInput = input;
    }}
/>
```

setState

- ReactJs setState not update your state immediately
- Before performance, it use collect all changes and update to render one time
- So, sometime you action affect state with quite a lot of source, you state maybe go wrong.
- Use callback to correct you state

Instead of:

```
this.setState({ counter: this.state.counter + 1 });
```

Use:

```
this.setState((state) => {
            return { counter: state.counter + 1 };
        });
```

HOCS

Style Component

Add style for component

https://create-react-app.dev/docs/adding-a-stylesheet

https://www.sitepoint.com/react-components-styling-options/

Redux Saga

PUSH HEROKU APP

NextJS

- Toolchain that’s support both SSR (Server Side Rendering and CSR (Client Side Rendering)
- Support for webs page that need SEO
- Tutorials: https://nextjs.org/learn/basics/create-nextjs-app

TypeScript

Code Splitting

Accessibility

Advance Guide

Context

Redux persist

Error Boundaring