Chương 4

Giới thiệu JS, JSX và ReactNative

Mục lục

Mục lục

1.1 Javascript

- Javascript: ngôn ngữ lập trình kịch bản, được sử dụng rộng rãi trong phát triển các ứng dụng Web.
- Javascript Framework là một bộ thư viện được xây dựng dựa vào ngôn ngữ lập trình Javascript.
- Javascript thực thi ở phía máy khách (trên trình duyệt) và cả phía máy chủ (ví dụ NodeJS)
  - Phía máy khách: xử lý những đối tượng HTML trên trình duyệt, kiểm soát các dữ liệu đầu vào, xử lý các sự kiện, tạo các hiệu ứng,…
  - Phía máy chủ (backend): xử lý các logic nghiệp vụ của ứng dụng

```
<script type="text/javascript">
        //JavaScript goes here
</script>
```

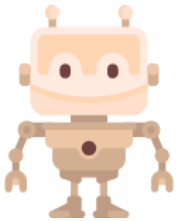
1.1 Javascript (2)

- Javascript engines
  - Mã JavaScript chạy trong một “engine" JavaScript.
- Babel
  - Babel là một trình biên dịch có thể cấu hình cho phép sử dụng các tính năng ngôn ngữ JavaScript mới (và các tiện ích mở rộng, như JSX), biên dịch "xuống" thành các phiên bản JavaScript cũ hơn được hỗ trợ trên nhiều loại công cụ
  - Khi khởi tạo một ứng dụng React Native, tệp cấu hình **babel.config.js** được tạo trong dự án

1.1 Javascript (3)

Viết mã JS theo cách truyền thống

```
<script type="text/javascript">
        //JavaScript goes here
</script>
```



“Modern” javascript

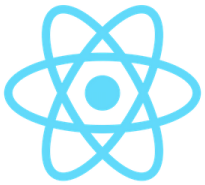
Babel

```
[1,2,3].map(n => n + 1);
```

Preprocessor →

```
[1, 2, 3].map(function (n) {
  return n + 1;
});
```

React Native sử dụng Babel làm bộ tiền xử lý



ES6 + JSX

ECMAScript 6 Extension

1.2 ES6 (ECMAScript6)

- Các điểm nhấn trong cú pháp của ES6:
  - Từ khoá `let` và `const`
  - Vòng lặp `for of`
  - Template literals
  - Giá trị mặc định cho tham số
  - Arrow Function
  - Xây dựng các class
  - Module
  - Rest Parameters (hay Rest Operator)
  - Toán tử Spread
  - Phép gán hủy cấu trúc

1.2 ES6 (ECMAScript6) (2)

- Từ khoá **`let`** và **`const`**
  - let để khai báo các biến

```
for (let i = 0; i < 5; i++) {
    console.log(i); // 0,1,2,3,4
}
console.log(i); // undefined
```

  - const giúp nghĩa các hằng

```
// Khai báo & Khởi tạo Hằng số PI
const PI = 3.14;
console.log(PI); // 3.14

// gán lại giá tri cho Hằng sẽ lỗi
PI = 10; // error
```

1.2 ES6 (ECMAScript6) (3)

- Vòng lặp **`for of`**
  - Lặp qua mảng hoặc lặp qua đối tượng dễ dàng

```
// Lặp qua mảng
let letters = ["a", "b", "c"];
for (let letter of letters) {
    console.log(letter);
}
```

- Template literals
  - Tạo chuỗi trên nhiều dòng và thực hiện nội suy chuỗi cho phép nhúng các biến hoặc biểu thức vào một chuỗi
  - Template literals được tạo bằng cách sử dụng cặp ký tự ` `

```
let str = `Chuỗi
    ở trên nhiều dòng!`;
let a = 6; let b = 9;
let result = `Tổng của ${a} và ${b} là: ${a+b}.`;
console.log(result); // Tổng của 6 và 9 là: 15.
```

1.2 ES6 (ECMAScript6) (4)

- Arrow Function: Cú pháp mũi tên (=>) là một cách viết tắt hàm.
  - `(param1, param2, …, paramN) => { statements }`
  - `(param1, param2, …, paramN) => expression`

    // tương đương với: `=> { return expression; }`
  - // Dấu ngoặc đơn là tùy chọn khi chỉ có một tên tham số:

    `(singleParam) => { statements }`

    `singleParam => { statements }`
  - // Danh sách tham số cho một hàm không có tham số phải được viết bằng một cặp dấu ngoặc đơn.

    `() => { statements }`

1.2 ES6 (ECMAScript6) (5)

- Arrow Function

```
// Function Expression
var sum = function(a, b) {
    return a + b;
}
console.log(sum(2, 3)); // 5
// Arrow function
var sum = (a, b) => a + b;
console.log(sum(2, 3)); // 5
```

- Giá trị mặc định cho tham số

```
function sayHello(name = "A") {
    var name = name;
    return `Xin chào ${name}!`;
}
console.log(sayHello()); // Xin chào A!
console.log(sayHello('B')); // Xin chào B!
```

1.2 ES6 (ECMAScript6) (6)

- Xây dựng các class

```
// Tạo một class
class Rectangle {
    // Hàm tạo (constructor)
    constructor(length, width) {
        this.length = length;
        this.width = width;
    }

    // Phương thức của class
    getArea() {
        return this.length*this.width;
    }
}
```

- Module
  - Mỗi module được biểu diễn bằng một tệp **.js** riêng biệt
  - Lệnh export hoặc import trong một module để xuất hoặc nhập các biến, hàm, class hoặc bất kỳ thực thể nào khác đến / từ các module hoặc tệp khác

1.2 ES6 (ECMAScript6) (7)

- Rest Parameters (hay Rest Operator)
  - Truyền một số tham số tùy ý cho hàm dưới dạng một mảng
  - Thêm vào phía trước tham số toán tử **...** (ba dấu chấm)

```
function sortNumbers(...numbers) {
    return numbers.sort();
}
console.log(sortNumbers(3, 5, 7));
console.log(sortNumbers(3, 5, 7, 1, 0));
```

- Toán tử Spread
  - Toán tử Spread (tức là chia nhỏ) một mảng và chuyển các giá trị vào hàm được chỉ định

```
let colors = ["Xanh", "Đỏ"];
let [a, b] = colors;
```

- Phép gán hủy cấu trúc
  - Biểu thức giúp dễ dàng trích xuất các giá trị từ mảng hoặc thuộc tính từ các đối tượng, thành các biến riêng biệt bằng cách cung cấp một cú pháp ngắn gọn.

1.3 JSX

- **JSX = Javascript + XML**: Một phần mở rộng cho JavaScript mà sử dụng để xây dựng các giao diện UI.
- Cú pháp của JSX cũng tương tự như XML bao gồm các cặp: thẻ mở và thẻ đóng

```
<JSXElementName JSX_Attributes>

</JSXElementName>
```

- JSX chuyển đổi từ cú pháp giống XML thành JavaScript gốc
  - Các phần tử XML được chuyển đổi thành các lời gọi hàm
  - Các thuộc tính XML được chuyển đổi thành các đối tượng
- Bất kỳ biểu thức JavaScript nào cũng có thể được nhúng vào JSX bằng cách đặt nó trong dấu ngoặc nhọn: {……}

1.3 JSX (2)

- Ví dụ:

Code Javascipt

```
React.createElement("div", {className: "red"}, "Children Text");
React.createElement(MyCounter, {count: 3 + 5});

React.createElement(DashboardUnit, {"data-index": "2"},
  React.createElement("h1", null, "Scores"),
  React.createElement(Scoreboard, {className: "results", scores: gameScores})
);
```

Code JSX tương ứng

```
<div className="red">Children Text</div>;
<MyCounter count={3 + 5} />;

var gameScores = {
  player1: 2,
  player2: 5
};
<DashboardUnit data-index="2">
  <h1>Scores</h1>
  <Scoreboard className="results" scores={gameScores} />
</DashboardUnit>;
```

1.3 JSX (3)

- Nhúng biểu thức trong JSX

```
const name = 'A';
const element = <h1>Welcome to {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

```
function sayHi(name) {
    if(name) {
        return <p>Xin chào, {name} !</p>
    }else{
        return <p>Xin chào bạn !</p>
    }
}
```

- Chỉ định attributes với JSX

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

- JSX Object: sử dụng **React.createElement()** để compile một JSX object thành JSX thông thường

```
const element = React.createElement(
  "p",
  { className: "welcome" },
  "Welcome!"
);
const element = <p className="welcome">Welcome!</p>
```

Mục lục

1. Javascript và JSX
2. **Giới thiệu React và ReactNative**
3. Các thành phần của ứng dụng ReactNative
4. Các thành phần UI phổ biến
5. Truy xuất dữ liệu qua mạng trong ReactNative

2.1 Giới thiệu về React

- **React** là "một thư viện JS giúp xây dựng giao diện ứng dụng"
  - Thay vì sử dụng HTML template, react xây dựng các thành phần (component)
- React tuân theo mô hình phát triển theo hướng thành phần
  - Ý tưởng: chia giao diện người dùng thành một tập hợp các thành phần có thể tái sử dụng
  - Kết hợp và lồng ghép các thành phần để tạo giao diện người dùng hoàn chỉnh
- Những lợi ích:
  - Mã nguồn của các thành phần dễ dàng tái sử dụng
  - Khả năng đọc mã tốt hơn - các thành phần được lưu trữ trong các lớp riêng của chúng
  - Kiểm thử dễ dàng

2.1 Giới thiệu về React (2)

- Cài đặt môi trường phát triển React: Cài đặt node.js và các gói bổ sung khác - thông qua trình quản lý gói Node (Node Package Manager - npm).
- Các khái niệm cơ bản:
  - components
  - props
  - state
  - Life cycle
  - VirtualDOM

2.1 Giới thiệu về React (3)

- **Components** là những thành phần UI được chia nhỏ ra, độc lập và có thể tái sử dụng.
  - Component có thể là những function (stateless) hoặc class (stateful) trong JS.
  - Component sẽ có các thuộc tính props (properties) và state (nếu được định nghĩa bằng class).
  - Mỗi thành phần được thực thi và trả về bên trong hàm render()
  - Để phân biệt giữa React component và HMTL tag, tất cả các React components phải được viết kiểu **CamelCase** (các cụm từ được viết liền nhau và bắt đầu mỗi từ bằng chữ in hoa, không có khoảng cách hoặc dấu câu xen kẻ) và phải **bắt đầu bằng kí tự in hoa**.

2.1 Giới thiệu về React (4)

- **Props** là những thuộc tính được truyền vào một component và chỉ có thể đọc.
  - Ví dụ 1: `<button className="active">Sign up</button>`
    - Thuộc tính được truyền vào có thể truy xuất là className và child thông qua **props**.
    - Truy xuất bằng cú pháp props.className sẽ cho giá trị là "**active**" và props.child có giá trị là "**Sign up**".
  - Ví dụ 2:

```
class Welcome extends React.Component {
        render() {
                return <h1>Hello {this.props.name}</h1>;
        }
}
const element = <Welcome name="Sơn" />;
```

2.1 Giới thiệu về React (5)

- **State** là trạng thái thuộc về chính component đó, được quản lý bởi chính nó và không được truy xuất từ bên ngoài.
  - Trạng thái được sử dụng để lưu trữ thông tin (dữ liệu) về thành phần, có thể thay đổi theo thời gian.
  - Chỉ có thể sử dụng state khi dùng stateful component.
- Các hàm thao tác
  - Truy cập: **this.state.variable**
  - Thay đổi: **this.setState({variable: value});**

2.1 Giới thiệu về React (6)

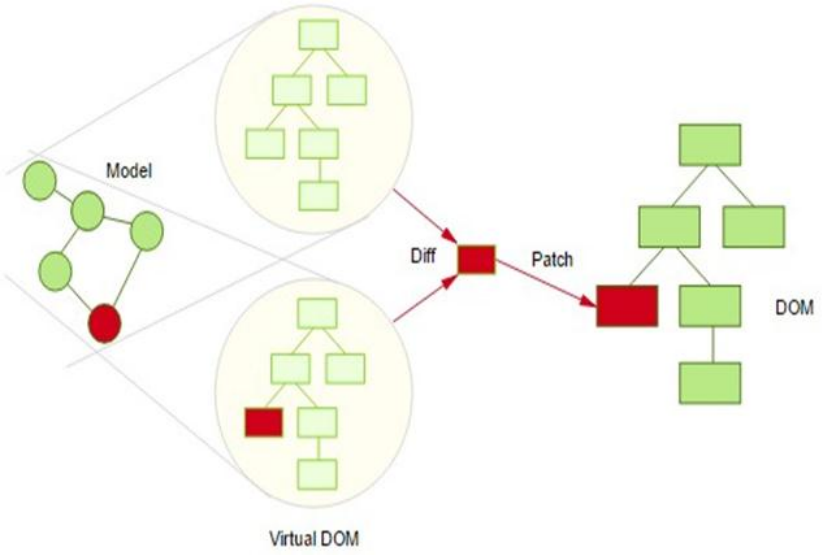
- **Life cycle** là một vòng đời của một React component từ lúc được render lần đầu tiên và mỗi lần render lại (mounting) và khi gỡ bỏ component (unmounting).
  - Hai phương thức được tự động gọi khi sự kiện mounting và unmounting xảy ra lần lượt là componentDidMount và componentWillUnmount.
  - Có thể ghi đè 2 phương thức này khi sử dụng stateful component (class).

2.1 Giới thiệu về React (7)

- **Virtual DOM**
  - DOM (Document Object Model) là một giao diện lập trình ứng dụng, cho phép Javascript hoặc một loại script khác đọc và thao tác nội dung của document.
    - Ứng dụng dạng Single Page Applications (SPA): DOM được tải về trình duyệt 1 lần
    - JS cập nhật DOM liên tục khi người dung thao tác ứng dụng
    - Mỗi khi DOM thay đổi, trình duyệt phải thực hiện render
  - → Điều này gây ra chậm trễ
  - React đưa ra VirtualDOM để tăng tốc ứng dụng, giảm thiểu thời gian khi trình duyệt render lại trang

2.1 Giới thiệu về React (8)

- **Virtual DOM**
  - Cơ chế hoạt động:
    - Khi component mới được thêm vào UI, một Virtual DOM sẽ được tạo ra
    - Khi state của component nào đó thay đổi, React sẽ cập nhật Virtual DOM đồng thời vẫn giữ phiên bản Virtual DOM trước để so sánh, điều này giúp tìm ra đối tượng Virtual DOM thay đổi.
    - Khi tìm được sự thay đổi, React chỉ cập nhật đối tượng đó trên DOM thật.

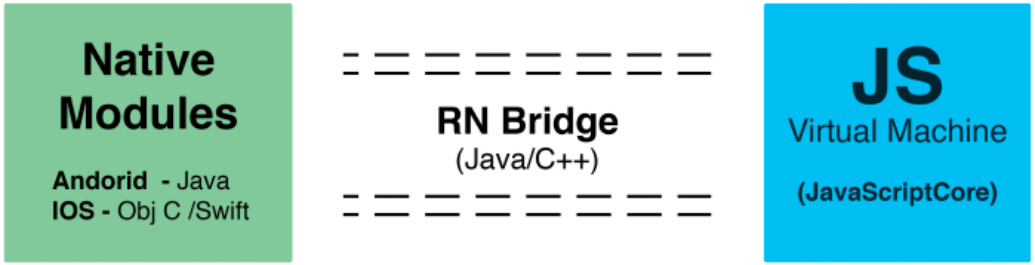


2.2 Giới thiệu về ReactNative

- React Native là framework JavaScript để viết các ứng dụng có thể dịch thành mã gốc trên nhiều nền tảng: iOS, Android
- Công bố lần đầu bởi Facebook vào năm 2015
  - Open source project - https://github.com/facebook/react-native
  - **Mô hình lập trình tương tự như React**: Component-based, JSX, stateful or stateless component, props, state,…
- React Native tạo ra các ứng dụng lai, trong đó mã JavaScript của người dùng được đóng gói và thông dịch bởi một công cụ JS engine cho mã gốc.
  - Thay vì sử dụng các phần tử HTML, React Native cung cấp một số component React có sẵn cho các nhà phát triển, điều này giúp tạo ra các phần tử native UI trên nền tảng đích.

2.2 Giới thiệu về ReactNative (2)

- Kiến trúc ReactNative



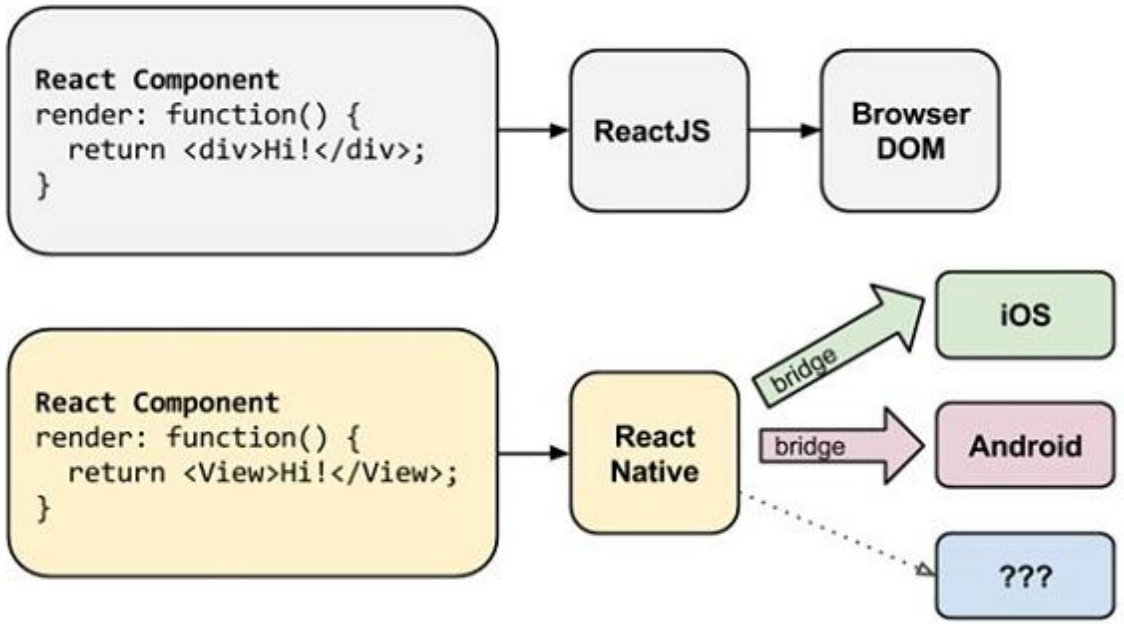
  - **Native Code/Modules**: viết bằng Swift và Java theo nền tảng gốc
  - **Javascript VM**: RN sử dụng JavaScriptCore (Safari)
    - Trong trường hợp của Android, RN gói JavaScriptCore cùng với ứng dụng
    - Trong trường hợp chế độ gỡ lỗi của Chrome, RN sử dụng engine V8 và giao tiếp với mã gốc qua WebSocket
  - **React Native Bridge**: một cầu nối C++/Java chịu trách nhiệm giao tiếp giữa thread native và Javascript
    - Một giao thức tùy chỉnh được sử dụng để truyền thông điệp

2.2 Giới thiệu về ReactNative (3)

- Nguyên lý cơ bản

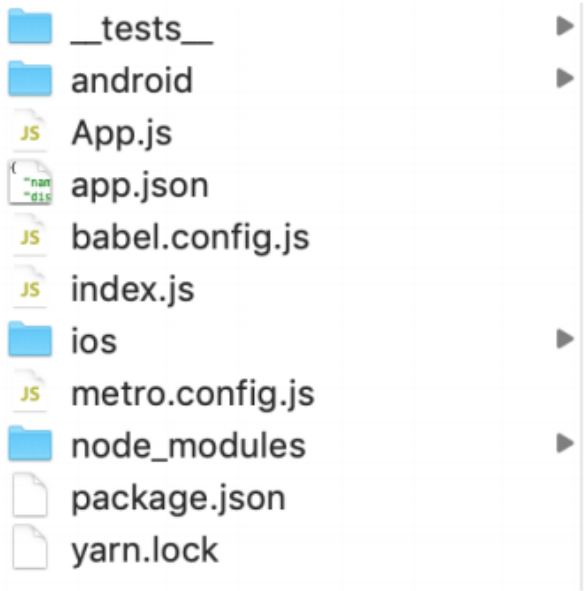
Các **React** component thông qua thư viện ReactJS được dịch thành các DOM của trình duyệt

Các **React Native** component thông qua framework React Native được dịch thành mã gốc tương ứng với nền tảng



2.3 Ứng dụng ReactNative đầu tiên

- Các công cụ phát triển ReactNative
  - node.js và trình quản lý gói Node (Node Package Manager - npm)
    - Cài đặt package RN: `npm install -g react-native-cli`
  - IDE: Visual Studio Code (or Atom,….), Xcode, Android Studio,…
- Cấu trúc thư mục dự án ReactNative



2.3 Ứng dụng ReactNative đầu tiên (2)

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

export default class HelloWorldApp extends Component {
        render() {
                return (
                        <View>
                                <Text>Hello world!</Text>
                        </View>
                );
        }
}
```

Phải import mọi thứ cần sử dụng; import là cú pháp ES6

Component: thường là những thứ nhìn thấy trên màn hình

từ khoá "class" và "extends"

render () hiển thị các thành phần

JSX: XML embedded in JavaScript

Đây là nội dung trong file: **App.js**

```
{ Component } from 'react';
// đây là cú pháp destructuring từ ES6
// tương đương với:
import React from "react";
let Component = React.Component;
```

Mục lục

1. Javascript và JSX
2. Giới thiệu React và ReactNative
3. **Các thành phần của ứng dụng ReactNative**
4. Các thành phần UI phổ biến
5. Truy xuất dữ liệu qua mạng trong ReactNative

3.1 ReactNative Components

- ReactNative Component có các đặc trưng giống với thành phần của React. Sự khác biệt là chúng liên kết với các phần tử giao diện người dùng native.
  - Component **`<View>`** là một vùng chứa chung của các phần tử giao diện người dùng khác
  - Component **`<Text>`** là một đoạn văn bản
  - Component **`<Image>`** là một hình ảnh
  - Component **`<Button>`** là nút bấm
  - …

Web (ReactJS)	React Native	Android	iOS
<input>	<TextInput>	EditText	UITextField
<p>	<Text>	TextView	UITextView
	<View>	Android.view	UIView

3.1 ReactNative Components (2)

- Các components cơ bản của React Native:
  - https://facebook.github.io/react-native/docs/components-and-apis.html
  - Basic Components
  - User Interface
  - List Views
  - …

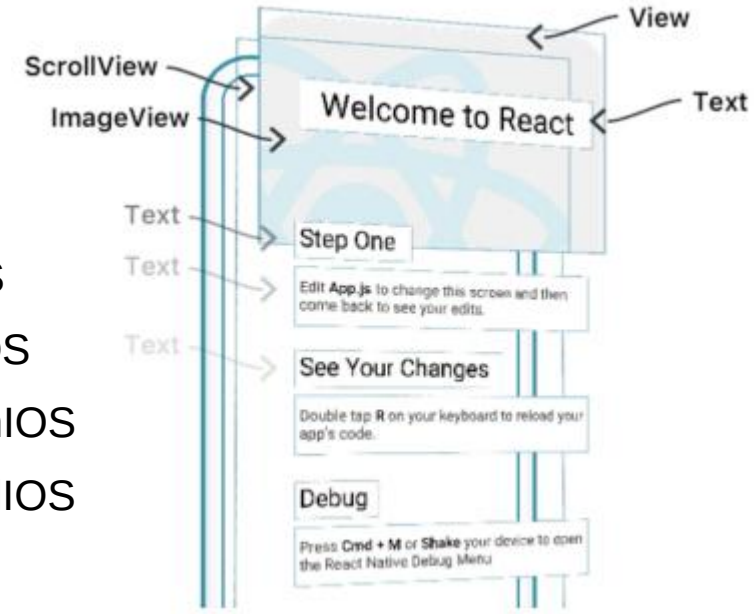
- Most common
  - View
  - Text
  - Image
  - TextInput
  - ScrollView
  - StyleSheet
- UI
  - Button
  - Switch
  - Picker
  - Slider
- On screen list views
  - FlatList
  - SectionList

- Các components chia sẻ bởi các lập trình viên khác:
  - http://www.awesome-react-native.com/#components

3.1 ReactNative Components (3)

- Các components riêng theo nền tảng Android hoặc iOS:
  - iOS-specific
  - Android-specific

- iOS components
  - ActionSheetIOS
  - AlertIOS
  - DatePickerIOS
  - ImagePickerIOS
  - ProgressViewIOS
  - PushNotificationIOS
  - SegmentControlIOS
  - TabBarIOS



- Android components
  - BackHandler
  - DatePickerAndroid
  - DrawerLayoutAndroid
  - PermissionsAndroid
  - ProgressBarAndroid
  - TimePickerAndroid
  - ToastAndroid
  - ToolbarAndroid
  - ViewPagerAndroid

3.1 ReactNative Components (4)

- Cấu trúc chung của một component:

- Để xây dựng một thành phần trước tiên cần import Component từ bộ thư viện react
- Cú pháp :

import React,{Component} from 'react'

- Sau đó tạo một class kế thừa Component

```
class MyComponent extends Component {
}
```

```
import React from 'react';

class MyComponent extends React.Component {
    constructor() {
        super();
     // This constructor defines
     // the state of the component
    }

    render() {
        return(
            // This defines the reactive UI
            // associated with the component
        )
    }

    my_method() {
        //This is a custom method
    }
}
```

3.2 Vòng đời ReactNative Components

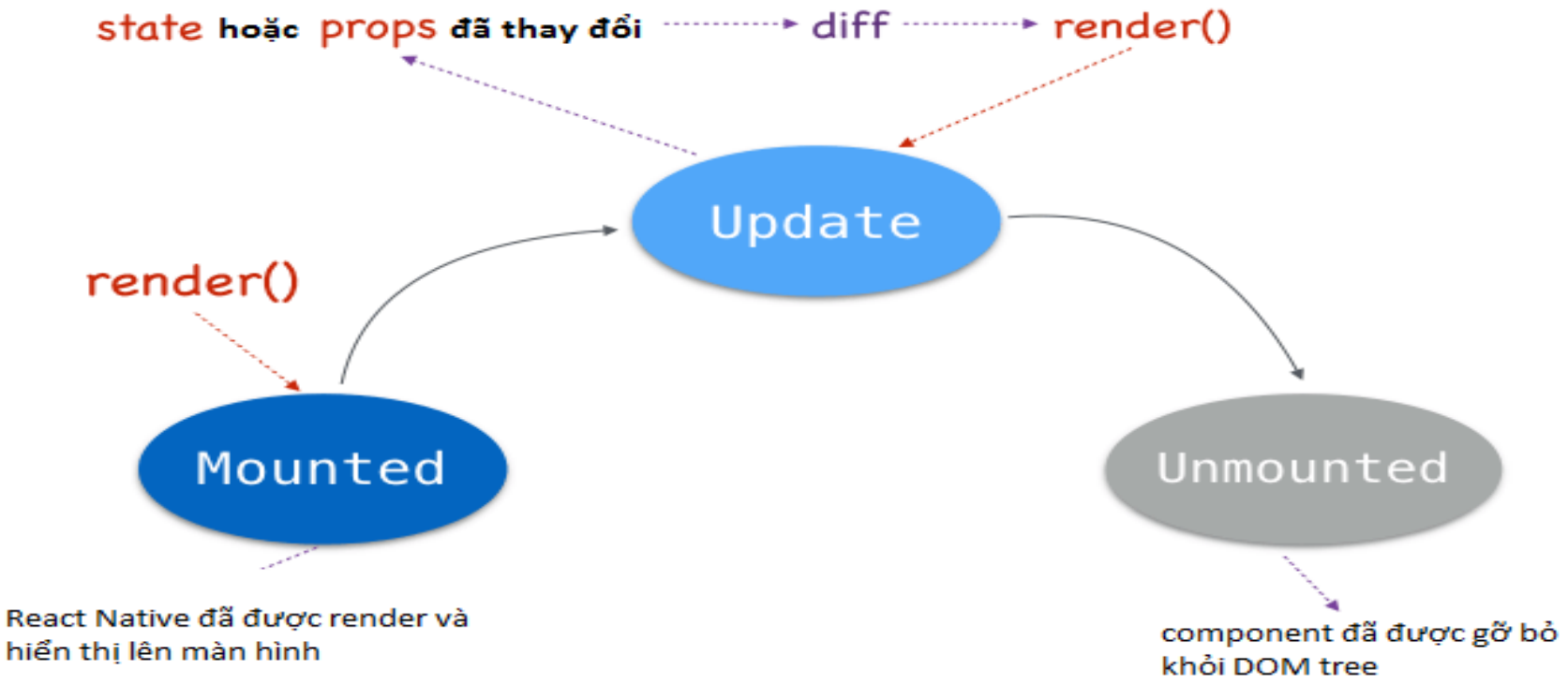
- Mỗi thành phần ReactNative sẽ có vòng đời riêng của nó. Vòng đời này được thực thi tương ứng với quá trình mà người dùng tương tác với ứng dụng
- Một vòng đời của Component bao gồm 3 giai đoạn chính:

Mounted / Update / UnMounted

  - **Mounted**: Chứa các phương thức được gọi khi Component được khởi tạo và Insert vào DOM.
  - **Update**: Chứa các phương thức được gọi khi một Component được render lại.
  - **UnMounted**: Chứa các phương thức được gọi khi một Component được gỡ bỏ khỏi DOM

3.2 Vòng đời ReactNative Components (2)

- Mô tả các giai đoạn vòng đời RN Component



3.2 Vòng đời ReactNative Components (3)

- Các phương thức giai đoạn Mounted:
  1. **constructor(object Props)**: phương thức khởi tạo của một Component thông thường phương thức này luôn nhận vào một props hoặc các tham số khác.
  2. **componentWillMount()**: phương thức này xảy ra một lần sau constructor và ở phương thức này chúng ta sẽ không reder giao diện cho Component
  3. **render()**: hoạt động sau componentWillMount , phương thức này dùng để render các Element của react ra giao diện cho người dùng. Tất cả những gì trong phương thức này đều sẽ hiển thị ra màn hình.
  4. **componentDidMount()**: phương thức này chỉ được gọi một lần sau khi render xảy ra , lúc này các Element đã được hiển thị trên màn hình và có thể được truy cập bằng cách this.refs. Thông thường chúng ta dùng phương thức này để viết các đoạn code liên quan tới delay hoặc đồng bộ API

3.2 Vòng đời ReactNative Components (4)

- Các phương thức giai đoạn UnMounted:

1. **componentWillReceiveProps(onject nextProps)** : Cha của component này sẽ truyền vào một props mới và phương thức này sẽ khởi tạo lại giao diện . Chúng ta có thể cập nhật lại state nội bộ thông qua phương thức this.setState() trước khi phương thức render được gọi.
2. **shouldComponentUpdate(object nextProps, object nextState)** : Phương thức này có giá trị trả về là kiểu boolean.
   - Một Component có thể được render hoặc không được render, phương thức này nhằm đảm bảo component sẽ được render lại nếu như trả về true. Chúng ta thường dùng phương thức này để kiểm tra props hoặc state có thay đổi hay không nếu như có thay đổi thì chúng ta trả ra true để chạy tiếp render() và trả ra false để ko chạy phương thức render().
3. **render()** : phương thức này chỉ được gọi lại khi shouldComponentUpdate trả ra true.
4. **componentDidUpdate(object prevProps,object prevState)** : Phương thức này chỉ thực hiện khi render() xảy ra . Lúc này giao diện đã được cập nhật

3.2 Vòng đời ReactNative Components (5)

- Các phương thức giai đoạn Update:
  - **componentWillUnmount()** : Phương thức này chỉ xảy ra khi ứng dụng bị đóng hoàn toàn.

3.3 Quản lý trạng thái

- Trạng thái trong ứng dụng React Native có thể được định nghĩa là một tập hợp các giá trị mà một thành phần sử dụng và quản lý.
- Trạng thái thành phần là một đối tượng JavaScript được khai báo khi một component được tạo.
- RN component giống như React component:
  - props = properties: các thông số có thể truyền vào khi khởi tạo component, sau đó props không thể bị thay đổi
  - state: giá trị ban đầu của trạng thái được khởi tạo trong hàm khởi tạo, sau đó trạng thái có thể thay đổi qua hàm setState()

3.3 Quản lý trạng thái (2)

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {
   render() {
      return (
         <Text>Hello {this.props.name}!</Text>
      );
   }
}

export default class LotsOfGreetings extends Component {
   render() {
      return (
         <View style={{alignItems: 'center'}}>
            <Greeting name='Rexxar' />
            <Greeting name='Jaina' />
            <Greeting name='Valeera' />
         </View>
      );
   }
}
```

- component **Greeting** khi tạo ra sẽ được gán giá trị từ bên ngoài vào **this.props.name**
- Sau đó giá trị nàu không thể thay đổi

- Tạo 3 component **Greeting**
- Giá trị được truyền vào qua thể hiện "name"
- Kết quả là 3 lời chào hiển thị trên màn hình

```
import React, {Component} from "react";
import {Button, Text, View, SafeAreaView} from "react-native";
class Counter extends Component {
        state = { counter: this.props.value };
        render(props) {
                return (
                        <View>
                                <Text>{this.state.counter}</Text>
                                <Button
                                        onPress={() => {this.setState({counter:
                                        this.state.counter + 1});}}
                                        title={"Click here"}
                                />
                        </View>
                );
        }
}
class MyApp extends Component {
        render() {
                return (
                        <SafeAreaView>
                                <Counter value={10}/>
                        </SafeAreaView>
                );
        }
}
export default MyApp;
```

3.3 Quản lý trạng thái (4)

- Hooks: là một bổ sung mới trong React 16.8, ReactNative hỗ trợ Hooks kể từ bản phát hành 0.59
  - Là các hàm cho phép "nhập" vào trạng thái React và các tính năng vòng đời từ các component viết theo dạng function. Hook không hoạt động bên trong các lớp.
- React cung cấp một vài Hooks tích hợp sẵn
  - useEffect thêm khả năng thực hiện các tác dụng phụ từ một thành phần chức năng
  - useContext cho phép đăng ký bối cảnh React
  - useReducer cho phép quản lý trạng thái cục bộ của các thành phần phức tạp bằng reducer
- Lập trình viên có thể tạo Hook của riêng mình

3.3 Quản lý trạng thái (5)

- Hooks là các hàm JavaScript, nhưng chúng áp đặt hai quy tắc bổ sung
- Chỉ gọi Hook ở cấp cao nhất
  - Không gọi các Hook bên trong các vòng lặp, điều kiện hoặc các hàm lồng nhau
- Chỉ gọi các Hook từ các component viết theo dạng function của React
  - Không gọi các Hook từ các hàm JavaScript thông thường

```
import React, { useState } from "react";
import { Button, Text, View, SafeAreaView } from "react-native";
const Counter = props => {
        const [counter, setCounter] = useState(props.value);
        return (
                <View>
                        <Text>{counter}</Text>
                        <Button
                                onPress={() => {setCounter(counter+1);}}
                                title={"Click here"}
                        />
                </View>
        );
}
const MyApp = () => {
        return (
                <SafeAreaView>
                        <Counter value={10} />
                </SafeAreaView>
        );
}
export default MyApp;
```

3.3 Quản lý trạng thái (6)

- So sánh props và state

State	Props
Internal data	external data
Mutable	Immutable
Created in the component	Inherited from a parent
Can only be updated in the component	Can be updated by parent component
Can be passed down as props	Can be passed down as props
Can set default values inside Component	Can set default values inside Component

Mục lục

1. Javascript và JSX
2. Giới thiệu React và ReactNative
3. Các thành phần của ứng dụng ReactNative
4. **Các thành phần UI phổ biến**
5. Truy xuất dữ liệu qua mạng trong ReactNative

4.1 Các thành phần UI phổ biến

- <View> - Tương tự như HTML , không có giao diện mặc định nhưng có thể được tạo kiểu. Cũng thường được sử dụng để bố trí các thành phần con.
- <Text> - Hiển thị văn bản ra màn hình. Chúng có thể được lồng vào nhau - các kiểu áp dụng cho tổ tiên cũng sẽ áp dụng cho con cháu:

```
<Text style={{ fontWeight: 'bold' }}>
    This is some bold text.
    <Text style={{ fontStyle: 'italic'
    }}>
        This is some bold and italic
    text.
    </Text>
</Text>
```

4.1 Các thành phần UI phổ biến (2)

- <Image> - hiển thị hình ảnh trên màn hình.
  - Có thể kết xuất hình ảnh cục bộ và mạng.
  - Có thể đặt chiều rộng và chiều cao thông qua thuộc tính style. Điều này phải được thực hiện khi tải hình ảnh mạng.
  - Nếu chiều rộng và chiều cao được cung cấp không khớp với chiều rộng và chiều cao thực tế, thì việc điều chỉnh tỷ lệ hoặc kéo dài sẽ xảy ra tùy thuộc vào thuộc tính resizeMode của hình ảnh.
- Hình ảnh cục bộ: tải hình ảnh có tên 'Trex.png' nằm trong cùng thư mục với tệp JavaScript hiện tại:

```
<Image source={require('./TRex.png')}/>
```

- Hình ảnh từ xa: tải hình ảnh tại URL được chỉ định:

```
<Image style={{ width: 150, height: 200 }}
source={{ uri: 'http://via.placeholder.com/150x200' }} />
```

4.1 Các thành phần UI phổ biến (3)

- <Image> - hiển thị hình ảnh trên màn hình.
  - Trong React Native có thể chỉ định hình ảnh riêng cho từng Platform bằng cách chỉ định đuôi mở rộng image.ios.png và image.android.png
  - Ngoài ra có thể cung cấp hình ảnh cho từng thiết bị có kích thước khác nhau.
  - Cú pháp : @2x hoặc @3x

```
.
├── button.js
└── img
    ├── check@2x.png
    └── check@3x.png
```

4.1 Các thành phần UI phổ biến (4)

- <TextInput> - cho phép người dùng nhập văn bản.
  - Thuộc tính trình xử lý sự kiện onChangeText cho phép nhà phát triển phản hồi bất kỳ thay đổi nào mà người dùng thực hiện.

```
<View style={{ padding: 10, flex: 1, justifyContent: 'center' }}>
<TextInput
style={{ height: 40 }}
placeholder="Type here to translate!"
onChangeText={(text) => this.setState({ text })}
/>
<Text style={{ padding: 10, fontSize: 42 }}>
          {this.state.text.split(' ').map((word) => word && '□ ').join(' ')}
</Text>
</View>
```

4.1 Các thành phần UI phổ biến (5)

```
import React, { Component } from 'react';
import { AppRegistry, Text Input } from 'react-native';
export default class UselessTextInput extends Component {
    constructor(props) {
        super(props);
        this.state = { text: 'Useless Placeholder' };
    }
    render() {
        return (
          <Text Input
       style={{height: 40, borderColor: 'gray', borderWidth: 1}}
       onChangeText={(text) => this.setState({text})}
       value={this.state.text}
          />
        );
    }
}
```

Đây là tên của biến trạng thái sẽ nhận văn bản từ hộp Text Input

Thành phần Button

```
<Button
  onPress={() => {
   Alert.alert('You tapped the button!');
  }}
  title="Press Me"
/>
```

- Kết xuất (render)
  - Nhãn màu xanh trên iOS
  - Hình chữ nhật bo tròn màu xanh với văn bản màu trắng trên Android
- Hàm "onPress"
  - Có thể gọi một trình xử lý sự kiện
  - Có thể là hàm ẩn danh
  - Có thể chỉ định một thuộc tính “color" để thay đổi màu của nút
- Xây dựng button riêng bằng cách sử dụng bất kỳ thành phần "Có thể chạm" ("Touchable" component)
  - Các thành phần " Touchable" cung cấp khả năng ghi lại các cử chỉ chạm, và hiển thị phản hồi khi một cử chỉ được nhận dạng.
  - Không cung cấp kiểu dáng mặc định, phải định nghĩa kiểu riêng

4.2 ScrollView

- ScrollView là một vùng chứa cuộn chung có thể lưu trữ nhiều component và view
  - Các item không cần phải đồng nhất
  - Có thể cuộn theo cả chiều dọc và chiều ngang
  - Tất cả các phần tử của ScrollView đều được hiển thị
- ScrollViews có thể được định cấu hình để cho phép phân trang qua các view bằng cách sử dụng cử chỉ vuốt (**prop pagingEnabled**)
  - Trên Android, có thể thực hiện vuốt theo chiều ngang giữa các view với ViewPager
  - Trên iOS, ScrollView với một mục duy nhất có thể được sử dụng để cho phép người dùng thu phóng nội dung

4.2 ScrollView (2)

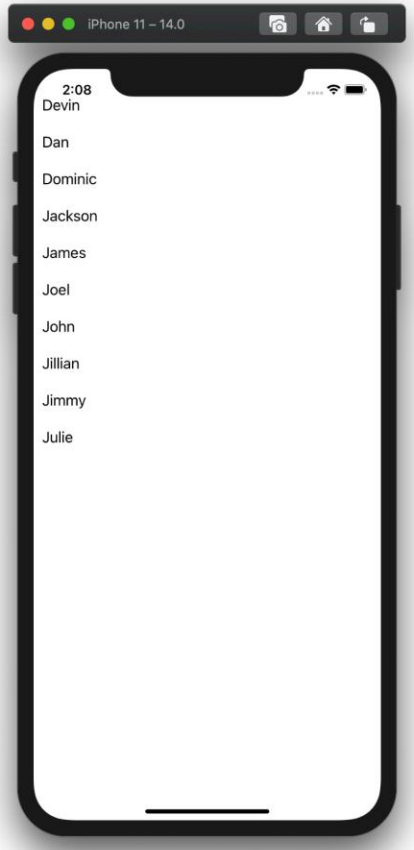
```
const logo = {
  uri: 'https://reactnative.dev/img/tiny_logo.png',
  width: 64,
  height: 64
};

export default App = () => (
  <ScrollView>
    <Text style={{ fontSize: 96 }}>Scroll me plz</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Text style={{ fontSize: 96 }}>If you like</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Text style={{ fontSize: 96 }}>Scrolling down</Text>
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    <Image source={logo} />
    …
  </ScrollView>
);
```

Example

4.2 ListView

- React Native cung cấp các thành phần để trình bày danh sách dữ liệu, sử dụng **FlatList** hoặc **SectionList**
- Thành phần **FlatList** hiển thị danh sách cuộn
  - Hoạt động tốt cho danh sách dữ liệu dài, trong đó số lượng mục có thể thay đổi theo thời gian
  - Không giống như ScrollView, FlatList chỉ render các phần tử hiện đang hiển thị trên màn hình, không phải tất cả các phần tử cùng một lúc
- Thành phần FlatList yêu cầu hai prop:
  - **Data**: nguồn thông tin cho danh sách
  - **renderItem**: hàm nhận một mục từ nguồn và trả về thành phần được định dạng hiển thị
- **SectionList** hiển thị một tập hợp dữ liệu được chia thành các phần logic

```
import React from 'react';
import { FlatList, StyleSheet, Text, View } from 'react-native';

const FlatListBasics = () => { Example (II)
        return (
                <View style={styles.container}>
                <FlatList
                data={[
                        { key: 'Devin' },
                        { key: 'Dan' },
                        { key: 'Dominic' },
                        { key: 'Jackson' },
                        { key: 'James' },
                        { key: 'Joel' },
                        { key: 'John' },
                        { key: 'Jillian' },
                        { key: 'Jimmy' },
                        { key: 'Julie' },
                ]}
                renderItem={({ item }) =>
                <Text style={styles.item}>{item.key}</Text>}
                />
                </View>
        );
}
export default FlatListBasics;
```



Mục lục

1. Javascript và JSX
2. Giới thiệu React và ReactNative
3. Các thành phần của ứng dụng ReactNative
4. Các thành phần UI phổ biến
5. **Truy xuất dữ liệu qua mạng trong ReactNative**

5.1 Giao tiếp qua mạng trong ReactNative

- RN cung cấp hàm fetch cho các xử lý liên quan đến kết mạng (networking)
  - Tương tự như XMLHttpRequest hoặc các API networking khác
- Các thao tác networking là không đồng bộ
  - Phương thức **fetch()** trả về một **Promise** giúp dễ dàng viết các đoạn mã hoạt động không đồng bộ
  - Một đối tượng **Promise** đại diện cho việc hoàn thành (hoặc thất bại) cuối cùng của một hoạt động không đồng bộ và giá trị kết quả của nó

5.2 Hàm fetch()

- Đặc tả fetch khác với jQuery.ajax() ở hai điểm chính:
  - Promise được trả về từ fetch() sẽ không từ chối với trạng thái lỗi HTTP ngay cả khi phản hồi là HTTP 404 hoặc 500
    - Thay vào đó, nó sẽ giải quyết bình thường (với trạng thái ok được đặt thành false) và nó sẽ chỉ từ chối khi lỗi mạng hoặc nếu có bất kỳ điều gì ngăn cản yêu cầu hoàn thành
  - Theo mặc định, fetch sẽ không gửi hoặc nhận bất kỳ cookie nào từ máy chủ
    - Nếu trang web dựa vào việc duy trì phiên người dùng có thể dẫn đến các request chưa được xác thực (để gửi cookie, tùy chọn init thông tin xác thực phải được thiết lập)

5.2 Hàm fetch() (2)

- Cú pháp
  - Fetch nhận đầu vào là một URL và truy xuất dữ liệu từ URL
  - Tham số thứ hai tùy chọn cho phép tùy chỉnh request HTTP: ví dụ chỉ định các tiêu đề bổ sung hoặc phương thức thực hiện request GET/POST

```
fetch('https://mywebsite.com/endpoint/', {
      method: 'POST',
      headers: {
            Accept: 'application/json',
            'Content-Type': 'application/json',
},

body: JSON.stringify({
      firstParam: 'yourValue',
      secondParam: 'yourOtherValue', }),
});
```

Phương thức gửi request: GET hoặc POST

Sử dụng JSON để chuyển thông tin qua web
JSON là một giao thức javascript đơn giản để chuyển cấu trúc dữ liệu thành chuỗi

5.2 Hàm fetch() (3)

- Xử lý phản hồi: đối tượng response
  - Một lời gọi fetch sẽ trả về một promise
  - Khi promise hoàn thành, nó trả về một đối tượng response
  - Một đối tượng phản hồi có các thuộc tính và phương thức
    - Phương thức quan trọng: Body.json()
    - Nhận Response stream và đọc nó cho đến khi hoàn thành.
    - Kết quả là văn bản nội dung dưới dạng JSON

```
const json = await response.json()

const text = await response.text()
```

5.3 Promise

- Một promise là một đối tượng được trả về để đính kèm các lệnh gọi lại (callback), thay vì chuyển các callback vào một hàm

```
function getMoviesFromApiAsync() {
   return fetch('https://facebook.github.io/react-native/movies.json')
      .then((response) => response.json())
      .then((responseJson) => {
         return responseJson.movies;
       })
      .catch((error) => {
         console.error(error);
      });
}
```

Promise: Đối tượng **Promise** đại diện cho kết quả cuối cùng (hoàn thành hoặc thất bại) của một hoạt động không đồng bộ và giá trị kết quả của nó.

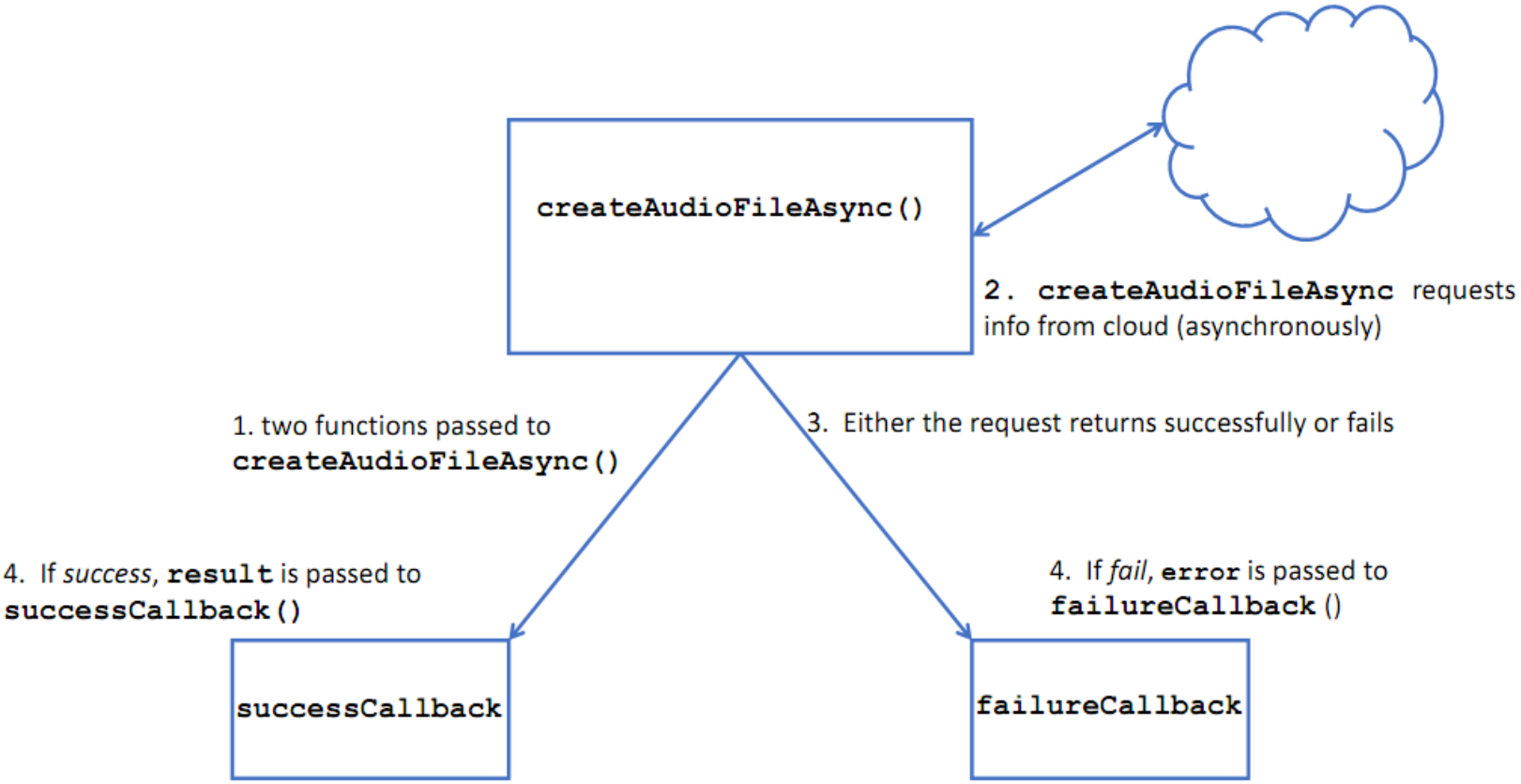
5.3 Promise (2)

- Ví dụ: chúng ta cần xây dựng hàm **createAudioFileAsync()**
  - Hàm này sẽ thực hiện thao tác không đồng bộ tạo tệp âm thanh
  - Hàm nhận đầu vào là 3 tham số: một bản ghi cấu hình và hai hàm callback (một được gọi nếu tệp âm thanh được tạo thành công và một được gọi nếu xảy ra lỗi)
  - Thông thường hàm createAudioFileAsync() sẽ được xây dựng như sau:

```
function successCallback(result) {
        console.log("Audio file ready at URL: " + result);
}

function failureCallback(error) {
        console.log("Error generating audio file: " + error);
}

createAudioFileAsync(audioSettings, successCallback, failureCallback);
```

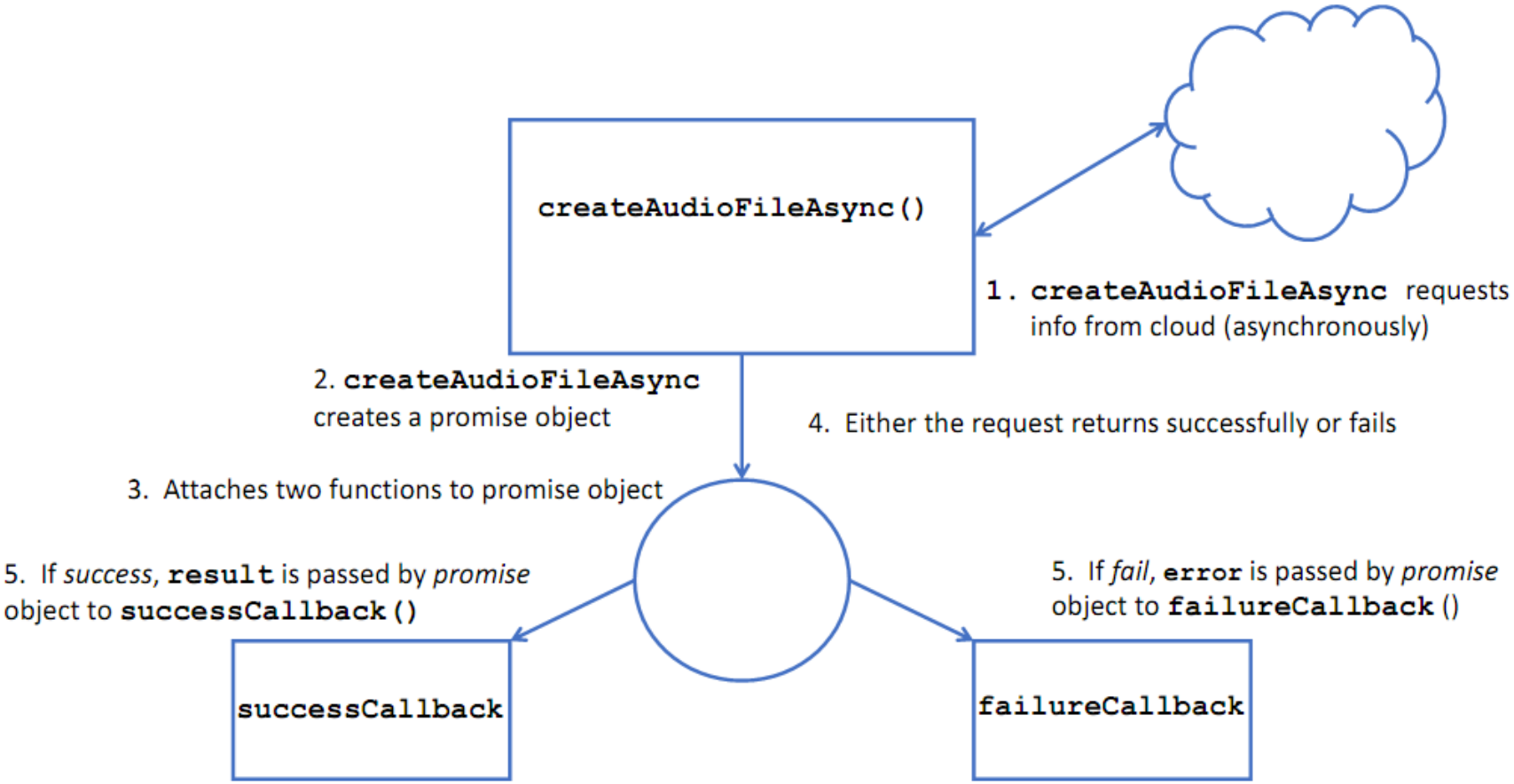
5.3 Promise (3)

- Cài đặt thông thường, khi chưa có promise



5.3 Promise (4)

- Cài đặt sử dụng promise



5.3 Promise (5)

- Chúng ta có thể đính kèm các lệnh callback của mình qua đối tượng promise trả về

```
const promise = createAudioFileAsync(audioSettings);
promise.then(successCallback, failureCallback);
```

- Có thể dùng cách viết rút gọn như sau:

```
createAudioFileAsync(audioSettings).then(successCallback, failureCallback);
```

- Đây được gọi là các lời gọi hàm không đồng bộ. Không giống như các lệnh callback truyền theo kiểu cũ, một promise đi kèm với một số đảm bảo:
  - Các lệnh callback sẽ không bao giờ được gọi trước khi hoàn thành quá trình thực thi hiện tại của vòng lặp sự kiện JavaScript
  - Các lệnh gọi lại được thêm vào với cú pháp **then()** ngay cả khi hoạt động không đồng bộ thành công hay thất bại
  - Có thể thêm nhiều lệnh callback bằng cách gọi **then()** nhiều lần → được gọi là chuỗi hàm (chaining)

5.3 Promise (6)

- Promise chain
  - Nhu cầu chung là thực hiện hai hoặc nhiều hoạt động không đồng bộ liền nhau
    - mỗi hoạt động tiếp theo bắt đầu khi hoạt động trước đó thành công
    - kết quả từ bước trước là promise cho bước tiếp theo
  - Thực hiện điều này bằng cách tạo ra một chuỗi promise (promise chain).

Cú pháp kiểu cũ

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Cú pháp mới

```
doSomething().then(function(result) {
  return doSomethingElse(result);
})
.then(function(newResult) {
  return doThirdThing(newResult);
})
.then(function(finalResult) {
  console.log('Got the final result: ' + finalResult);
})
.catch(failureCallback);
```

5.3 Promise (7)

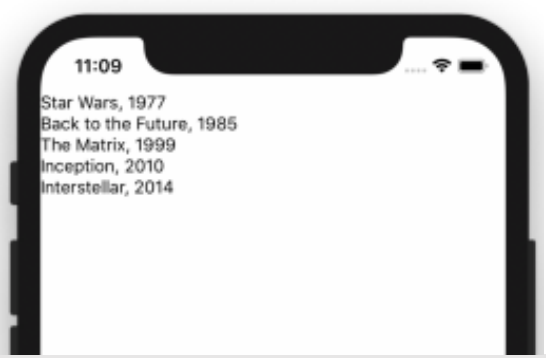
- Promise chain
  - Các đối số của **then()** là tùy chọn và **catch(failureCallback)** là viết tắt của **then(null, failureCallback)**.
  - Cách viết được thể hiện bằng các hàm mũi tên (=>):

```
doSomething()
.then(result => doSomethingElse(result))
.then(newResult => doThirdThing(newResult))
.then(finalResult => {
        console.log(`Got the final result: ${finalResult}`);
})
.catch(failureCallback);
```

Ví dụ

```
import React from 'react';
import { FlatList, ActivityIndicator,
        Text, View, SafeAreaView } from 'react-native';

export default class FetchExample extends React.Component {

  constructor(props) {
    super(props);
    this.state = { isLoading: true }
  }

  componentDidMount() {
    return fetch('https://facebook.github.io/react-native/movies.json')
      .then((response) => response.json())
      .then((responseJson) => {

        this.setState({
          isLoading: false,
          dataSource: responseJson.movies,
        }, function () {

        });

      })
      .catch((error) => {
        console.error(error);
      });
  }
```

Ví dụ (2)




```
render() {
    if (this.state.isLoading) {
      return (
        <View style={{ flex: 1, padding: 20 }}>
          <ActivityIndicator />
        </View>
      )
    }

    return (
      < SafeAreaView style={{ flex: 1, paddingTop: 20 }}>
        <FlatList
          data={this.state.dataSource}
          renderItem={({ item }) => <Text>{item.title}, {item.releaseYear}</Text>}
          keyExtractor={({ id }, index) => id}
        />
      </SafeAreaView>
    );
  }
}
```

HẾT TUẦN 4

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions !