

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ- ĐHQGHN

Khoa Công nghệ thông tin

=====<><><>=====



Phát triển Ứng dụng di động

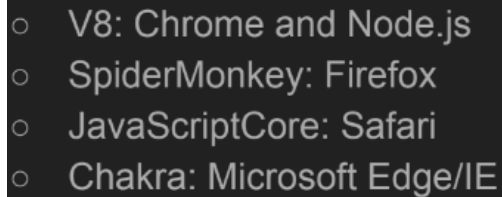
Họ tên: Nguyễn Đức Khánh
MSSV: 18020695
Khóa học: [React Native](#)
Mã môn học: 2021II_INT3120_3

=====<><><>=====

I. Overview, javascript

1. Javascript là ngôn ngữ thông dịch:

- Javascript là ngôn ngữ thông dịch. Mỗi loại trình duyệt có một nền tảng javascript riêng để chạy js:



- V8: Chrome and Node.js
- SpiderMonkey: Firefox
- JavaScriptCore: Safari
- Chakra: Microsoft Edge/IE

- Mỗi loại trên đều được triển khai theo tiêu chuẩn ECMAScript nhưng sẽ khác nhau ở các phần không có trong ECMAScript.

2. Cú pháp:

- Khi khai báo string có thể sử dụng cả ngoặc kép hoặc ngoặc đơn. Cuối câu lệnh có thể có dấu chấm phẩy hoặc không có đều được.

Ví dụ: **const a = "hello"** hoặc **const a = 'hello';**

=> Hai câu khai báo trên có tác dụng như nhau (gán biến a có giá trị là hello)

- Ta có thể truyền vào trong mảng js nhiều loại biến khác nhau; kể cả hàm:

Ví dụ: **const arr = ['string', 10, function() {console.log('hello')}]**

=> có thể chạy hàm trong mảng trên bằng lệnh **arr[2]()**

- Vòng for:

```
for(i = 0; i < arr.length; i++) {  
    console.log(arr[i])  
}
```

3. Kiểu dữ liệu

- JS là ngôn ngữ dynamic typing: khi khai báo một biến; ta không cần khai báo kiểu dữ liệu của biến đó. Js sẽ dựa vào giá trị được gán cho biến trong thời gian chạy để xác định kiểu dữ liệu của nó.

- Các kiểu dữ liệu cơ bản:

- **number** : các số bất kỳ loại nào: số nguyên hoặc dấu phẩy động.
- **string** : chuỗi. Chuỗi có thể có một hoặc nhiều ký tự, không có loại ký tự đơn riêng biệt.
- **boolean** : **true** / **false**.
- **null** : các giá trị không xác định – một loại độc lập có một giá trị duy nhất **null**.
- **undefined** : các giá trị chưa được gán – một kiểu độc lập có một giá trị duy nhất **undefined**.
- **object** : các cấu trúc dữ liệu phức tạp hơn.

4. Typecasting? Coercion.

- Trong quá trình thực thi; Js có thể chuyển đổi tự động kiểu dữ liệu của biến:
 - + **const x = 10** (gán x = 10; x là số nguyên)
 - + **const y = x + ''** (lúc này y có giá trị là '10' là một string. js đã tự động ép kiểu của x sang string trước khi tính toán)
- Cách lấy kiểu trong JS: dùng hàm typeof()
- typeof null trả về 'object'
- Toán tử '==' và '===':
 - + '==': So sánh hai giá trị nhưng không so sánh kiểu dữ liệu
 - + '===': So sánh cả giá trị và kiểu dữ liệu
- Các giá trị falsy:

```
○ undefined
○ null
○ false
○ +0, -0, NaN
○ ""
```

5. Đối tượng (Object):

- Tất cả mọi thứ trong js đều là đối tượng (do chúng kế thừa nguyên mẫu prototype)

6. Đối tượng và các kiểu dữ liệu cơ bản:

- Các biến có các kiểu dữ liệu cơ bản là bất biến. Chúng chỉ lưu trữ một giá trị và không thể thay đổi. Khi thay đổi; nó sẽ tạo ra một bản gốc mới thay thế cho cái cũ.
- Các đối tượng có thể sử dụng các tham chiếu để lưu trữ và có thể thay đổi.
 - + Các key trong object sẽ tự động được chuyển thành chuỗi
Ví dụ: **object = { 1: 'abc' }**
khi đó **object[1] = object['1'] = 'abc'**
 - + Ví dụ về object sử dụng tham chiếu:
o = {'1': 'abc'}; o2 = o; o[1] = 'new'
thì khi đó **o[1] = 'new'**
 - + Do đó; khi gán một object cho một biến khác; để tránh trường hợp trên ta cần có các hàm để gán như Object.assign() hoặc ta sẽ gán lần lượt từng key có chứa các loại giá trị cơ bản sang để copy.

7. Prototypal

- Các kiểu khác cơ bản có một vài thuộc tính / phương thức liên kết với nó như: **Array.prototype.push()**
- Mỗi đối tượng lưu trữ một tham chiếu đến nguyên mẫu của nó

- Hầu hết các biến cơ bản đều có các phương thức như: `String()`; `Number()`;...
- Ta có thể tự viết các phương thức kế thừa nguyên mẫu để sử dụng
 - + Ta có thể ghi đè các phương thức trong prototype của một biế:
vd: **`a.prototype.toString = function(){ return 'hello'}`**

8. Phạm vi

- `var`: từ khi khai báo đến khi function kết thúc
- `const`, `let`: cho đến dấu `}` tiếp theo
- Đối tượng toàn cục: trong trình duyệt có một đối tượng toàn cục là `window`.
khi khai báo một biến **`y = 10`**; nó sẽ xuất hiện trong `window` và có thể lấy ra bằng cách **`window.y`**

II. Javascript, ES6

1. Closures:

- Các hàm tham chiếu đến các biến do cha mẹ khai báo vẫn có quyền truy cập các biến đó (Do phạm vi của các biến trong JS)

Ví dụ:

```
for(i = 0; i <= 5; i++) {  
    arr.push(function() {  
        console.log(i)  
    })  
}
```

Khi đó nếu ta chạy **`arr[0]()`** thì kết quả sẽ in ra là 5 thay vì là 0 do sau khi đến cuối vòng lặp; giá trị của `i` là 5 và công việc của hàm được thêm vào mảng là in ra giá trị `i` đó.

2. Immediately-invoked function expression (IIFE)

- Là một dạng function expression định nghĩa một anonymous function được thực thi ngay sau khi nó được tạo ra.

```
(function(name) {  
    console.log(`Hello ${name}`)  
})('World')  
// Hello World
```

3. First-class function:

- Các hàm trong js có thể được xử lý giống như các biến thông thường. Chúng có thể được gán cho một biến khác; được truyền dưới dạng đối số cho hàm khác; hoặc được trả lại từ một hàm. Điều này có được là do mọi thứ trong JS đều có thể được coi như những đối tượng. Điều đó cũng đúng với các hàm trong JS.

- Ví dụ:

```
function createGenerator(prefix) {  
    let index = 0;  
    return function generateNewID() {  
        index++;  
        return prefix + index.toString();  
    }  
}
```

```
let generateNewID = createGenerator("btn");  
console.log(generateNewID()); // btn1
```

=> Ở trên ta thực hiện gán một hàm cho tham số và gán nó vào hàm console.log(). Hàm createGenerator cũng trả về một hàm generateNewID. Hàm này lưu trữ giá trị 'btn' khi chạy **let generateNewID = createGenerator("btn");** Do đó khi chạy **console.log(generateNewID());** Sẽ ra kết quả 'btn1'

4. Đồng bộ; Bất đồng bộ; Đơn luồng:

- Javascript là ngôn ngữ đồng bộ; đơn luồng. Và do đó nó có thể khiến cho trang web bị dừng lại và không phản hồi lại người dùng (Ví dụ: ta có thể viết một đoạn mã làm cho JS phải đợi 10p. Trong thời gian đó nó sẽ không xử lý được các công việc khác nữa.)
- Một chức năng mất nhiều thời gian để chạy sẽ gây ra tình trạng trang không phản hồi
- Tuy nhiên Js cũng có các hàm hoạt động không đồng bộ.

5. Javascript bất đồng bộ:

- Execution stack: khi có một hàm được gọi; nó sẽ được thêm vào stack thực thi. Sau khi thực thi xong; nó sẽ bị xóa khỏi stack để chạy các chức năng khác.
- Với những hàm không đồng bộ (như các hàm được cung cấp bởi trình duyệt) Chúng sẽ được đưa vào Browser APIs và thực hiện ở đó. Sau khi thực hiện xong; chúng sẽ được đưa trở lại Function Queue.
- Khi Execution Stack trống; nó sẽ kiểm tra trong Function Queue xem có hàm nào đang chờ hay không. Nếu có nó sẽ đưa hàm đó vào stack để thực hiện.
- Khi các ngăn xếp quá đầy có thể gây ra lỗi overflow.
- Bằng cách này ta có thể khiến cho JS hoạt động không đồng bộ khi cần thiết.

6. Các cách thực hiện không đồng bộ trong js:

- Callbacks: Đây là cách kiểm soát luồng với các cuộc gọi bất đồng bộ bằng cách thực thi một số hàm ngay sau khi cuộc gọi bất đồng bộ trả về giá trị. Tức là chương trình không phải dừng lại để đợi các giá trị đó. Nó có thể thực hiện các công việc khác và sau khi giá trị đó được trả về thì nó quay lại thực hiện công việc đã được cài đặt. Phương thức này có một nhược điểm đó là khi có quá nhiều callbacks được sử dụng; đoạn mã sẽ trở lên cồng kềnh và trông giống kim tự tháp.

- + Ví dụ:

```
function myDisplayer(s) {  
    console.log(s)  
}  
  
function myCalculator(num1, num2, myCallback) {  
    let sum = num1 + num2;  
    myCallback(sum);  
}  
  
myCalculator(5, 5, myDisplayer);
```

- + Ở trên: hành động console.log() sẽ được thực hiện khi nào sum được trả về.

- Promises: Đây là cách viết mã giả định rằng một giá trị sẽ được trả về. Ưu điểm lớn nhất của phương thức này là nó chỉ cần một trình xử lý lỗi duy nhất.

- + Ví dụ:

```
fetch(url).then(res => {  
    console.log(res)  
}).catch(err => {  
    console.log(err)  
})
```

- + Ta có thể viết nhiều hàm then hơn trong promises.

- Async/Await: Cho phép người dùng có thể viết các đoạn code bất đồng bộ như những đoạn code đồng bộ thông thường. Để có thể sử dụng await trong một function ta cần phải khai báo function đó là một async function:

- + **async function () {**
 let res = await fetch(url)
 console.log(res)
}

- + Async/Await giúp đoạn code trở nên sạch và dễ hiểu hơn so với 2 loại trên.

III. React; Props; State

1. Classes:

- Được giới thiệu trong ES6
- Classes là một khái niệm trừu tượng mà ta có thể khai báo sẽ có các thuộc tính; phương thức; phương thức tĩnh (Phương thức tĩnh là phương thức mà nó không quan tâm đến thể hiện của class đó. Thay vào đó nó quan tâm đến tất cả các thể hiện của lớp và ta có thể sử dụng tên lớp để gọi phương thức đó ra. ví dụ: **Date.now()**. Tương tự với các thuộc tính tĩnh). Sau đó ta có thể khai báo các đối tượng được gọi là thể hiện của lớp đó. Ví dụ **Date** là một lớp trong JS. Để tạo một thể hiện của lớp **Date** ta dùng như sau: **let date = new Date()**
- Sử dụng các từ khóa: new, constructor, extends, super.

2. React:

- Là một thư viện JS cho phép viết các dạng view có thể tự động phản ứng với các thay đổi trong dữ liệu.
- Cho phép chia nhỏ vấn đề ra thành các component nhỏ hơn
- Cho phép viết code đơn giản hơn nhưng vẫn đạt hiệu suất cao.
- React cung cấp một API để khai báo để ta không cần quan tâm đến những thay đổi

3. Imperative vs Declarative

- Imperative phải phác thảo một loạt các bước để đạt được những thứ ta cần còn Declarative chỉ cần nói những gì ta muốn và nó chỉ là một chi tiết triển khai về cách lấy nó.
- Ví dụ trong Html ta phải cho trình duyệt biết cách chính xác những thứ cần hiển thị. Do đó Html là một ngôn ngữ khai báo (Declarative).
- Thay vào đó, với JS ta phải nói chính xác cách ta thực hiện (Ví dụ: đầu tiên phải tạo một thẻ div; thêm class;... gán nó vào DOM). Do đó nó là một ngôn ngữ Imperative.
- React is Declarative. Điều đó giúp cho chúng ta không cần lo lắng về những thứ khác mà chỉ cần quan tâm đến trạng thái của nó.
- React thường được tổ chức thành các component nhỏ hơn. Ta có thể tái sử dụng chúng cho nhiều trang khác nhau. Từ đó chúng ta có thể dễ dàng thay đổi giao diện mà không cần phải chỉnh sửa quá nhiều dòng code.

4. React Component:

- Có hai cách khai báo component đó là class component và function component:

- + Class component:

```
class VD extends React.Component {  
    render() {  
        return(  
            <Text>Ví dụ</Text>  
        )  
    }  
}
```

- + Function component:

```
const VD = () => {  
    return (  
        <Text>Ví dụ</Text>  
    )  
}
```

- Khi dữ liệu thay đổi; chỉ những phần cần thiết được thay đổi thay vì thay đổi lại tất cả mọi thứ trong DOM.
- React được viết bằng JSX (là viết tắt của Javascript và JSX; về cơ bản thì nó giống với XML) sau đó nó sẽ được biên dịch lại thành JS. Do đó các thẻ được định nghĩa trong JSX, XML giống hệt với các thẻ trong Html

5. Props:

- Truyền Object vào component khác để thực hiện tính toán
- Những thay đổi trong props sẽ gây ra sự thay đổi trong views
- Chúng có thể là bất kỳ kiểu giá trị Javascript nào

Ví dụ với component VD trên; ta có thể truyền props = 1 vào như sau:

```
<VD props={1}/>
```

6. State

- Nơi lưu trữ các trạng thái nội bộ của một thành phần (component)
- Truy cập state của một component bằng câu lệnh "this.state".
- Cập nhật state của một component bằng câu lệnh "this.setState()"
- Khi thay đổi state, thì hệ thống sẽ tự động render lại view.

IV. React Native

1. Giới thiệu:

- Là một framework dựa trên React core
- Cho phép ta xây dựng ứng dụng mobile chỉ bằng Js.
- Hỗ trợ trên cả IOS và Android

2. Cách React Native hoạt động:

- Javascript được đóng gói; vận chuyển và thu nhỏ
- Có các luồng riêng biệt cho UI, Layout và JavaScript.
- Giao tiếp không đồng bộ thông qua bridge:
 - + JS thread sẽ yêu cầu các phần tử UI được hiển thị. Và khi có tương tác với UI thì tương tác đó cũng được truyền về cho JS biết thông qua bridge.
 - + JS thread có thể bị chặn và UI vẫn hoạt động do JS và UI hoạt động trên 2 luồng khác nhau.

3. Sự khác biệt giữa React Native và Web:

a. Các components cơ bản.

- Các component được khai báo trong 'react-native'
- div -> View
- Tất cả các string phải được sử dụng trong thẻ Text
- button -> Button
- ScrollView
- Trong thực tế; các component trên không có trong mobile; ta phải import nó từ thư viện react native

b. Style

- React Native sử dụng JS object để định dạng style
- Các key định dạng dựa trên css
- Lengths (đơn vị độ dài) là các số thay vì các đơn vị như pixel;.. trong web.
- Khởi tạo style bằng StyleSheet.create()

c. Event Handling

- Không giống như web; không phải mọi thành phần trong React Native đều có thể tương tác
- Chỉ một số thành phần là có thể tương tác; chạm vào như button;...

d. Components

- Trả về một node (một thứ có thể hiển thị ra màn hình)
- Hiển thị một phần giao diện
- Gồm hai loại: SFC và React.Component
 - + SFC: thành phần thường được sử dụng khi không cần state. Nó có thể là một function nhận vào props và trả về node. Mỗi khi props thay đổi sẽ làm cho nó thay đổi.

- + `React.Component`: là một lớp trừu tượng có thể mở rộng. Nó có các instances; state; các lifecycle method. Nó duy trì trạng thái (state) bên trong nó.
- **Component lifecycle**: Một component có vòng đời của riêng nó đi từ mount đến update cho đến khi bị unmount. ta có thể hiểu là một component sẽ được bắt đầu với việc tạo ra nó (mount). Sau đó nó được hiển thị ra màn hình. Khi có props hoặc state thay đổi; nó sẽ cập nhật lại (update). Cuối cùng khi nó không còn cần thiết nữa thì sẽ bị xóa đi (unmount).
- **Mount**: Về cơ bản Mount là một loạt các bước xảy ra khi một component được gắn kết và hiển thị. Điều đầu tiên xảy ra trong quá trình này là hàm khởi tạo constructor của class được gọi. Ta có thể khởi tạo state; thêm một số phương thức... ở hàm này. Tiếp sau đó là hàm `render()` được chạy và nó sẽ trả về một node. Sau đó đến `ComponentDidMount()`. Ở đây ta có thể làm những thứ không liên quan đến UI như các hành vi bất đồng bộ (gọi api,...); sau đó có thể cập nhật trạng thái cho phù hợp. Setting state ở đây sẽ làm cho component render lại mà không cập nhật UI.
- **Update**: Mỗi khi một instance của component được update, nó sẽ tự động gọi lần lượt 5 methods: `componentWillReceiveProps`, `shouldComponentUpdate`, `componentWillUpdate`, `render`, và `componentDidUpdate`.
 - + `componentWillReceiveProps(nextProps)`: khi một component được update; hàm này sẽ được gọi trước khi render. Và nó chỉ được gọi nếu component nhận một props. nextProps là giá trị props sắp được truyền vào component.
 - + `shouldComponentUpdate`: Khi một component update; hàm này sẽ chạy trước hàm trên và trước render. Nó thường được dùng để so sánh giá trị thay đổi (state, props). Nếu trả về true thì việc update diễn ra bình thường. Ngược lại tất cả method còn lại của updating cycle method sẽ không được gọi nữa, kể cả render và component sẽ không được update, `shouldComponentUpdate` tự động nhận vào 2 argument là: nextProps và nextState, nextProps và nextState là prop và state được truyền từ một component khác đến, hay chính là giá trị prop và state nếu component được update,

shouldComponentUpdate sẽ kiểm tra 2 giá trị này với prop và state hiện tại của nó. Nếu khác nhau sẽ tiến hành update component, còn không thì dừng hẳn lại update component.

- + componentWillUpdate: Nó được gọi giữa hai hàm shouldComponentUpdate và render. Mục tiêu chính của componentWillUpdate là tương tác những thứ bên ngoài kiến trúc React. Nó thường được sử dụng để tương tác API.
- + Tiếp theo là method render và cuối cùng trong updating lifecycle method là componentDidMount. Khi một component instance update, componentDidMount sẽ được gọi sau khi render được chạy xong.
 - unmount: Khi một component bị remove ra khỏi DOM. componentWillUnmount sẽ được gọi trước khi điều đó xảy ra. Hàm này thường được sử dụng để clean up;...
- e. Import/Export: Trong một dự án; thay vì viết hết tất cả component trong một file ta có thể chia nhỏ nó ra và chia vào các file nhỏ khác nhau; sau đó sử dụng import - export để sử dụng component khi cần thiết. Điều đó giúp ta tổ chức dự án tốt hơn. Các sử dụng: export từ một file và import nó từ một file khác thông qua đường dẫn.

V. Lists, User Input

1. Lists

Trong web, các trình duyệt sẽ tự động hiện thanh scroll khi mà nội dung hiển thị dài quá độ rộng màn hình. Trong thiết bị di động, chúng ta phải cài đặt việc đó theo cách thủ công: ScrollView, ListView, FlatList, SectionList

a. ScrollView

- Đây là chế độ xem cuộn cơ bản nhất
 - Sẽ hiển thị tất cả các thành phần con của nó trước khi xuất hiện
 - Dùng .map() để hiển thị một mảng dữ liệu. Các thành phần con trong mảng cần một khóa duy nhất để định danh

b. FlatList

- Đây là chế độ xem cuộn hiệu quả để hiển thị dữ liệu
 - “Virtualized”: Chỉ hiển thị những gì cần thiết tại một thời điểm
 - Truyền một mảng hoặc dữ liệu dưới dạng props
 - Chỉ cập nhật lại khi mà props được cập nhật

c. SectionList

- Giống như FlatList nhưng thêm các thành phần sau
 - Thay vì sử dụng props, nó xác định các section

* * Để sử dụng list ta thường render một mảng bằng cách sử dụng map giống như **arr.map(element => return <React Component/>)**. Đoạn code trên sẽ trả về một mảng các phần tử react. Tuy nhiên cần lưu ý mỗi react component trong đó nên có một thuộc tính key để phân biệt trong mảng.

2. User Input

React Native khuyến nghị luôn sử dụng các thành phần được kiểm soát (controlled components)

VI. User Input, Debugging

1. User Input:

- React Native cung cấp component TextInput dùng để nhận dữ liệu nhập vào từ bàn phím.
- TextInput cần truyền value và onChangeText vào để có thể gán giá trị nhập vào cho biến.
- Trong React Native không có Form nên cần định nghĩa một phương thức khi submit dữ liệu
- Validate dữ liệu: có thể validate trước khi submit hoặc validate khi dữ liệu thay đổi thông qua componentDidUpdate.
- Trong thực tế khi dùng Input; lúc ấn vào sẽ có một bàn phím ảo hiện lên và ta cần tạo ra một khoảng padding để tránh bàn phím đó đè lên input.

2. Debugging

a. React errors and warnings

- Hiện thị lỗi dưới dạng cảnh báo trên toàn màn hình: console.error()
- Cảnh báo với biểu mẫu màu vàng: console.warn() (Không nên sử dụng trong môi trường product)

b. Chrome Devtools

- Ta có thể chạy js trong chrome tab

VII. Navigation

1. Navigation là gì

- Navigation là thuật ngữ liên quan đến việc di chuyển giữa các màn hình trong ứng dụng.
- Khác với web sử dụng URL để điều hướng thì các ứng dụng di động không sử dụng URL.
- Trong React Native có một số các thư viện cung cấp Navigation như React Navigation.

- Có nhiều sự khác biệt giữa điều hướng trong IOS và Android.
- Có nhiều thư viện trong react native cung cấp chức năng này. Trong đó thường được sử dụng là react-navigation.

2. Cài đặt Navigation:

```
npm install react-navigation@2.0.0-beta.5 --save
```

3. Navigators, routes, and screen components

- Navigator là một component triển khai một navigation pattern.
- Mỗi navigator sẽ cần có một hoặc nhiều routes. navigator là cha của routes và routes là con của navigator.
- Mỗi routes phải có tên và thành phần màn hình. Tên thường không được trùng nhau. Thành phần màn hình là React Component được hiển thị khi routes được chọn. Thành phần màn hình cũng có thể là một navigator khác.

4. Các loại navigator chính:

- a. Switch Navigator
- b. Stack Navigator

- Ví dụ từ trang [react-navigation](#):

```
import * as React from 'react';
import { View, Text } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

function HomeScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Home Screen</Text>
    </View>
  );
}

const Stack = createStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen name="Home" component={HomeScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}

export default App;
```

- + Trong ví dụ trên ta đã khởi tạo một StackNavigator và một tuyến routes đi đến màn Home (Home là một screen component).
- + Ta có thể thêm nhiều màn hình hơn vào nếu muốn:

```
function DetailsScreen() {
  return (
    <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
      <Text>Details Screen</Text>
    </View>
  );
}

const Stack = createStackNavigator();

function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator initialRouteName="Home">
        <Stack.Screen name="Home" component={HomeScreen} />
        <Stack.Screen name="Details" component={DetailsScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

- + Ở đây ta sẽ có hai màn hình tương ứng với 2 routes là Home và Detail. thuộc tính `initialRouteName` xác định routes được khởi tạo. Như trong ví dụ là `HomeScreen` do đó khi mới chạy màn hình Home sẽ hiện ra.
- + Để chuyển đổi giữa các màn hình; một thuộc tính là **navigation** sẽ được truyền vào screen dưới dạng props. và ta có thể dùng hàm **navigation.navigate()** để chuyển sang màn hình khác. Ví dụ: **navigation.navigate('Details')** với biến truyền vào hàm là **name** của screen đã khai báo ở **Stack.Screen** phía trên.

VIII. Data

- Các ứng dụng thường cần phải giao tiếp với các tài nguyên khác để có thể làm nhiều các tác vụ khác nhau. Để giao tiếp với các tài nguyên khác chúng thường sử dụng các API (Application Programming Interface). Các API là các hành vi bất đồng bộ nên ta cần viết các đoạn code bất đồng bộ để gọi các API khi cần thiết.
- Tạo network request: sử dụng hàm `fetch()`
- Gọi API bằng Promise: Cho phép viết code bất đồng bộ bằng `then()` và `catch()`
- Async/Await: Cho phép viết code bất đồng bộ giống như code đồng bộ; sử dụng `try catch` để xử lý lỗi.
- Transforming Data: Đôi khi dữ liệu API trả về chưa phải là tốt; khi đó ta cần chuyển đổi dữ liệu thành dạng như ta mong muốn.

- Authentication: là tiến trình xác thực người dùng xem ai đang dùng ứng dụng của ta. Thường sử dụng tên và mật khẩu để xác thực.
- HTTP methods: gồm 2 loại chính là GET() và POST().
- Các loại HTTP response code:

- 200: OK
- 400: Bad Request
- 403: Forbidden
- 404: Not Found
- 500: Internal Server Error
- 418: I'm a teapot

IX. Project trong khóa học: [link github](#)

1. Project 0:

My TODO App

Item count: 2

new 1

Unchecked count: 1

New TODO

☐ new

☒ new 1

2. Project 1:

Work Time

00:00



















Work Time: Mins: Secs:

Break Time: Mins: Secs:

Start

Reset

3. Project 2: - 

Homes	Homes
<div>Search...</div> <div><div><div>The ABC Murders 2018 (series)</div></div><div><div><div>ABC Afterschool Specials 1972–1997 (series)</div></div><div><div><div>ABC Africa 2001 (movie)</div></div><div><div><div>ABC TGIF 1989–2018 (series)</div></div><div><div><div>ABC News Nightline 1980– (series)</div></div><div><div><div>ABC World News Tonight with David Muir 1953– (series)</div></div><div><div><div>ABC Weekend Specials 1977–1995 (series)</div></div><div><div><div>Garotas do ABC 2003 (movie)</div></div><div><div><div>ABC da Greve 1990 (movie)</div></div></div></div></div></div></div></div></div></div></div>	<div>123</div> <div><div><div>The Taking of Pelham 123 2009 (movie)</div></div><div><div><div>Chelmsford 123 1988–1990 (series)</div></div><div><div><div>Sesame Street: 123 Count with Me 1997 (movie)</div></div><div><div><div>123 2002 (movie)</div></div><div><div><div>No Time to Lose: The Making of 'Pelham 123' 2009 (movie)</div></div><div><div><div>UFC 123: Rampage vs. Machida 2010 (movie)</div></div><div><div><div>Fatboy Slim Live at Ibiza 123 Rocktronic Festival 2012 (movie)</div></div><div><div><div>Ludwig van Beethoven: Missa solemnis op. 123 1979 (movie)</div></div><div><div><div>Sesame Street: 123 1988 (game)</div></div></div></div></div></div></div></div></div></div></div>

[Homes](#) The Taking of Pelham 123



The Taking of Pelham 123 (2009)

R, 106 min

Armed men hijack a New York City subway train, holding the passengers hostage in return for a ransom, and turning an ordinary day's work for dispatcher Walter Garber into a face-off with the mastermind behind the crime.

Internet Movie Database (6.4/10):

Rotten Tomatoes (51%):

Metacritic (55/100):