HCMUTE

KHOA CÔNG NGHỆ THÔNG TIN

GIỚI THIỆU HỌC PHẦN
CÀI ĐẶT MÔI TRƯỜNG

Khoa Công nghệ Thông tin
Đại học Sư phạm Kỹ thuật TP.HCM

ThS. Nguyễn Hữu Trung

Cài đặt môi trường

- Cài JDK 11+
- Cấu hình môi trường cho JDK
- Cài NodeJS
- Cài Visual Studio Code
- Cài Extension: ES7
- Cài Android Studio Hoặc Xcode (IOS)

Các ứng dụng sử dụng React Native

1. Facebook
2. Skype
3. Facebook Ads
4. Instagram
5. Walmart
6. Airbnb
7. SoundCloud Pulse
8. Yeti Smart Home
9. Uber Eats
10. . . .

Lộ Trình học ReactNative

1. Typescript và cài đặt typescript
2. Interface và Class
3. Datetime
4. Regex
5. Callback functions
6. Async/await functions

Lộ Trình học ReactNative

1. Cài đặt môi trường, build, run
2. Giới thiệu về App, State, Props
3. Class component & Function component
4. View, Buttons, Text
5. Styling, Color, App Theme
6. Images & Icon, Flexbox
7. ListView
8. Text Input & validate
9. ScrollView
10. Router, navigation, statusBar
11. Libraries & package
12. HTTPs
13. Layout
14. React hooks: useEffect, useState

Lộ Trình học ReactNative

1. React hooks: useCallback, useMemo
2. Network: Axios
3. Quản lý state: Redux
4. Quản lý state: Redux saga
5. Local database (Realm), share preferences, keychain
6. Cloud database: Firebase
7. Animations
8. Thay đổi splash screen & launcher icon app

HCMUTE

KHOA CÔNG NGHỆ THÔNG TIN

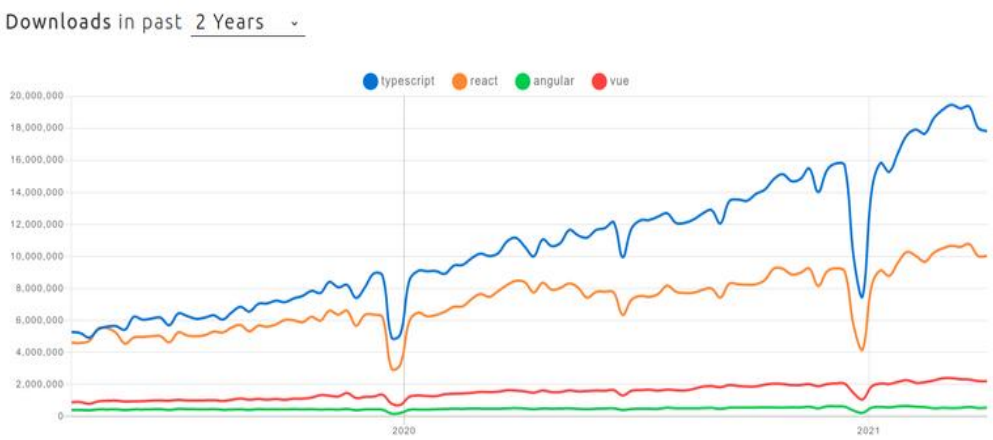
TỔNG QUAN VỀ TYPESCRIPT

Khoa Công nghệ Thông tin
Đại học Sư phạm Kỹ thuật TP.HCM

ThS. Nguyễn Hữu Trung

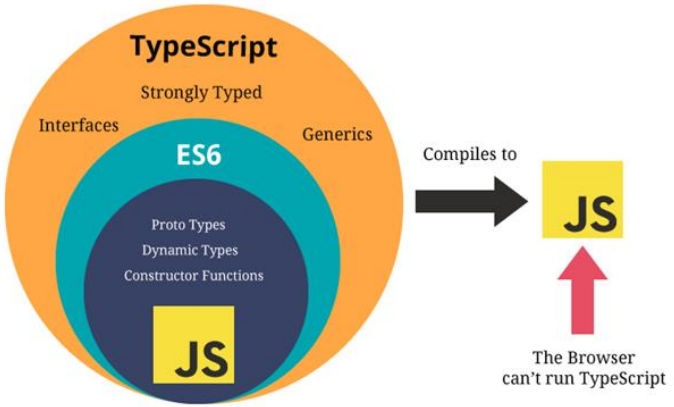
Giới thiệu TypeScript

- TypeScript là một ngôn ngữ lập trình mã nguồn mở, được xây dựng dựa trên JavaScript. Phần mở rộng **.ts**



Stats

	stars	issues	updated	created	size
typescript	70,101	5,129	Apr 15, 2021	Jun 17, 2014	minzipped size 712.2 KB
react	166,896	696	Apr 15, 2021	May 24, 2013	minzipped size 2.8 KB
angular	59,596	464	Mar 31, 2021	Jan 6, 2010	minzipped size 62.3 KB
vue	181,993	542	Apr 13, 2021	Jul 29, 2013	minzipped size 22.9 KB



Cài đặt TypeScript

- **Node.js** : Node.js là môi trường.
- **TypeScript compiler**: Là một module Node.js sẽ biên dịch code TypeScript thành code JavaScript. npm install -g typescript / tsc --v
- **Visual Studio Code(VS code):** Là một editor hỗ trợ code TypeScript. **Live Server** : cho phép bạn khởi chạy một server local cho việc phát triển.

Khởi tạo Project TypeScript

- Bước 1: Tạo một folder để lưu code, ví dụ folder là: **vidu1**
- Bước 2: Chạy VS Code và mở folder đó. Bước 3: Tạo một file TypeScript gọi là **app.ts** với extension của file là **.ts**
- Bước 4: Thêm code bên dưới vào file app.ts

```
let message: string = 'Hello, World!';
console.log(message);
```

- Bước 5: Mở Terminal trong VS Code bằng keyboard shortcut **Ctrl+`** hoặc theo menu **Terminal > New Terminal**
- Bước 6: Biên dịch **tsc app.ts** và chạy **node app.js**

Type Annotations

- TypeScript sử dụng **type annotations**(chú thích kiểu dữ liệu) để chỉ định rõ ràng các kiểu dữ liệu cho các định danh như variables, function, objects, etc.
- TypeScript sử dụng cú pháp **: type** sau một định danh để làm type annotations, và **type** có thể là bất kỳ loại dữ liệu hợp lệ nào.

```
let variableName: type;
let variableName: type = value;
const constantName: type = value;
let arrayName: type[];
```

Ví dụ:

```
let name: string = 'John';
let age: number = 25;
let active: boolean = true;
let names: string[] = ['John', 'Jane', 'Peter'];
let person: { name: string; age: number };
```

Number, String, Boolean trong TypeScript

- Tất cả các số trong TypeScript đều là giá trị floating-point(dấu phẩy động) hoặc số nguyên lớn. Các số dấu phẩy động có kiểu **number** trong khi số nguyên lớn có kiểu là **big int**.
- TypeScript cũng sử dụng cặp dấu nháy kép (") hoặc dấu nháy đơn (') để bao quanh các chuỗi ký tự
- Kiểu **boolean** cho phép 2 giá trị: **true** và **false**.

Ví dụ:

```
let name: string = 'John’;
let age: number = 25;
let active: boolean = true;
let names: string[] = ['John', 'Jane', 'Peter’];
let person: { name: string; age: number };
```

Array trong TypeScript

- Một mảng trong TypeScript là một danh sách dữ liệu có thứ tự: let arrayName: type[];

Ví dụ:

```
let names: string[] = ['John', 'Jane', 'Peter'];
```

```
let series = [1, 2, 3];
console.log(series.length);
```

```
let series = [1, 2, 3];
let doubleIt = series.map(e => e* 2);
console.log(doubleIt);
```

```
let scores : (string | number)[];
 scores = ['Programming', 5, 'Software Design', 4]; console.log(scores);
```

Enum trong TypeScript

- Enum là một nhóm tên các giá trị constant. Enum là kiểu liệt kê.

```
enum name {constant1, constant2, ...};
```

Ví dụ:

```
enum Month { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
```

```
function isItSummer(month: Month) {
    let isSummer: boolean;
    switch (month) {
    case Month.Jun:
    case Month.Jul:
    case Month.Aug:
        isSummer = true; break;
    default: isSummer = false; break;
    }
    return isSummer;
}
```

Void, Any TypeScript

- Kiểu **void** trong TypeScript được sử dụng để trả về loại của hàm, mà trong đó hàm không trả về bất kỳ giá trị nào

Ví dụ:

```
function log(message): void {
        console.log(messsage);
}
```

- Kiểu **any** cho phép bạn lưu trữ một giá trị thuộc bất kỳ kiểu nào.

```
// json may come from a third-party API
const json = `{"latitude": 10.11, "longitude":12.12}`;
// parse JSON to find location
const currentLocation = JSON.parse(json);
console.log(currentLocation);
```

```
let result: any;
result = 10.123;
console.log(result.toFixed());
result.willExist(); //
```

Type Assertions trong TypeScript

- Khi sử dụng Type Assertions, trình biên dịch sẽ coi một giá trị là một kiểu được chỉ định cụ thể. Sử dụng từ khóa **as**.

Ví dụ:

```
let netPrice = getNetPrice(100, 0.05, false) as number;
console.log(netPrice);

let netPrice = <number>getNetPrice(100, 0.05, false);
```

- Bạn cũng có thể sử dụng cú pháp ngoặc nhọn **<>** để xác nhận một kiểu, như sau: <targetType> value. Lưu ý rằng bạn không thể sử dụng cú pháp ngoặc nhọn **<>** với một số thư viện như React. Vì lý do này, bạn nên sử dụng từ khóa **as** cho các xác nhận kiểu dữ liệu

Interface trong TypeScript

- Interfaces trong TypeScript định nghĩa một tiêu chuẩn trong code của bạn. Chúng cung cấp các tên rõ ràng để kiểm tra loại dữ liệu. Một interface có thể có các thuộc tính tùy chọn. Để khai báo một thuộc tính tùy chọn, bạn sử dụng dấu (**?**) đặt ở cuối tên thuộc tính trong khai báo

```
function getFullName(
        person: {
                firstName: string; lastName: string
        }) {
        return `${person.firstName} ${person.lastName}`;
}
let person = { firstName: 'John', lastName: 'Doe' };
console.log(getFullName(person));
```

```
interface Person {
        readonly ssn: string;
        firstName: string;
        lastName: string;
        middleName?: string;
}
```

```
function getFullName(person: Person) {
 return `${person.firstName} ${person.lastName}`;
}
let john = { firstName: 'John', lastName: 'Doe' };
console.log(getFullName(john));
```

- JavaScript hỗ trợ các **default parameters** kể từ ES2015 (hoặc ES6) với cú pháp sau:

```
function name(parameter1=defaultValue1,...) {
// do something }
```

```
function applyDiscount(price, discount = 0.05)
{ return price * (1 - discount); }
console.log(applyDiscount(100)); // 95
```

- Một **Rest parameter**(các tham số còn lại) cho phép một hàm chấp nhận không hoặc nhiều đối số của kiểu được chỉ định. Trong TypeScript các rest parameter tuân theo các quy tắc sau:
  - Một hàm chỉ có một rest parameter
  - Rest parameter xuất hiện ở cuối danh sách các tham số
  - Loại của rest parameter là một loại mảng

```
function fn(...rest: type[]) { //... }
```

```
function getTotal(...numbers: number[]): number {
let total = 0; numbers.forEach((num) => total +=
num); return total; }
```

Functions trong TypeScript

HCMUTE

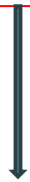
- Khai báo

```
function name(parameter: type, parameter:type,...): returnType {
        // do something
}
```

```
function add(x: number, y: number): number { return x + y; }
let add2 = function (x: number, y: number): number { return x + y; };
console.log(add(10, 20)); //output 30
console.log(add2(10, 20)); //output 30
//Sử dụng với arrow function(mũi tên (=>) xuất hiện giữa các tham số và kiểu trả về.)
let add3 = (x: number, y: number) : number => { return x + y; }
let add4 = (x: number, y: number) => { return x + y; }
 let add5 = (x: number, y: number) => x + y;
let add6: (a: number, b: number) => number = function (x: number, y: number) {
return x + y; };
console.log(add3(10, 20)); //output 30
console.log(add4(10, 20)); //output 30
console.log(add5(10, 20)); //output 30
console.log(add6(10, 20)); //output 30
```

Function Overloadings trong TypeScript


```
function addNumbers(a: number, b: number):
number { return a + b; }
function addStrings(a: string, b: string): string {
return a + b; }
```



```
function add(a: number | string, b: number | string): number | string {
if (typeof a === 'number' && typeof b === 'number')
            return a + b;
 if (typeof a === 'string' && typeof b === 'string')
            return a + b;
}
```

Function Overloadings trong TypeScript

- Để mô tả tốt hơn các mối quan hệ giữa các kiểu được sử dụng bởi một hàm, TypeScript hỗ trợ **function overloadings**(nạp chồng hàm). Ví dụ

```
function add(a: number, b: number): number;
function add(a: string, b: string): string;
function add(a: any, b: any): any {
        return a + b;
}
```

```
function sum(a: number, b: number): number;
function sum(a: number, b: number, c: number): number;
function sum(a: number, b: number, c?: number): number {
        if (c) return a + b + c; return a + b;
 }
```

- Khi một hàm là thuộc tính của một lớp, nó được gọi là một phương thức. TypeScript cũng hỗ trợ phương thức overloading

```
class Counter {
        private current: number = 0;
        count(): number;
        count(target: number): number[];
        count(target?: number): number | number[] {
                if (target) {
                        let values = [];
                        for (let start = this.current; start <= target; start++) {
                                values.push(start);
                         }
                this.current = target;
                return values;
        }
         return ++this.current;
} }
```

Class trong TypeScript

- Trong JavaScript thì không có khái niệm class như các ngôn ngữ Java, C#. Nhưng trong phiên bản ES5 bạn có thể sử dụng một hàm khởi tạo và kế thừa nguyên mẫu để tạo một class

```
function Person(ssn, firstName, lastName) {
        this.ssn = ssn;
       this.firstName = firstName;
       this.lastName = lastName;
}
```

Bây giờ, bạn có thể định nghĩa một phương thức prototype để lấy full name của person như bên dưới.

```
Person.prototype.getFullName = function () {
                return `${this.firstName} ${this.lastName}`;
}
```

Tiếp theo, bạn có thể sử dụng class Person bằng cách tạo một new object:

```
let person = new Person('171-28-0926','John','Doe');
console.log(person.getFullName());
```

Class trong TypeScript

- **Sử dụng class trong ES6, ví dụ:**

```
class Person {
        ssn; firstName; lastName;
        constructor(ssn, firstName, lastName) {
                this.ssn = ssn;
                this.firstName = firstName;
                this.lastName = lastName;
} }
```

```
let person = new Person('171-28-0926','John','Doe');
console.log(person.getFullName());
```

```
class Person { ssn; firstName; lastName;
constructor(ssn, firstName, lastName) {
this.ssn = ssn; this.firstName = firstName; this.lastName = lastName;
}

getFullName() {
return `${this.firstName} ${this.lastName}`;
} }
```

Class trong TypeScript

- Class trong TypeScript sẽ thêm type annotations(chú thích kiểu) đến các thuộc tính và phương thức trong class

```
class Person {
        ssn: string; firstName: string; lastName: string;
        constructor(ssn: string, firstName: string, lastName: string) {
        this.ssn = ssn; this.firstName = firstName; this.lastName = lastName;
}
getFullName(): string { return `${this.firstName} ${this.lastName}`; } }
```

Access Modifiers trong TypeScript

- Access modifiers có nhiệm vụ thay đổi quyền truy cập các thuộc tính và phương thức trong **class**. TypeScript cung cấp 3 mức truy cập là:
  - **private** chỉ cho phép truy cập bên trong class đó
  - **protected** chỉ cho phép truy cập bên trong class đó và bên trong class kế thừa(class con)
  - **public** cho phép truy cập ở bất kỳ vị trị nào
- Chú ý: TypeScript sẽ kiểm soát truy cập trong thời gian biên dịch chứ không phải trong thời gian chạy.

```
class Person { private ssn: string; private
firstName: string; private lastName: string; //
... }
```

```
class Person { protected ssn:
string; // other code }
```

```
class Person { // ... public getFullName(): string { return
`${this.firstName} ${this.lastName}`; } // ... }
```

Getters và Setters trong TypeScript

```
class Person {
        public age: number;
        public firstName:
        string; public lastName: string;
}
```

Để truy cập các thuộc tính của class **Person**, bạn có thể làm như sau:

```
let person = new Person();
person.age = 26;
```

Nhập dữ liệu

```
if( inputAge > 0 && inputAge < 200 ) {
        person.age = inputAge;
}
```

Getters và Setters trong TypeScript


```
class Person {
        private _age: number;
        private _firstName: string;
        private _lastName: string;
public get age() { return this._age; }
public set age(theAge: number) {
        if (theAge <= 0 || theAge >= 200) {
                throw new Error('The age is invalid');
        }
        this._age = theAge;
}
public get firstName() { return this._firstName; }
public set firstName(theFirstName: string) { if (!theFirstName) { throw new
Error('Invalid first name.'); } this._firstName = theFirstName; }
public get lastName() { return this._lastName; }
public set lastName(theLastName: string) { if (!theLastName) { throw new
Error('Invalid last name.'); } this._lastName = theLastName; }
public getFullName(): string { return `${this.firstName} ${this.lastName}`; } }
```

Inheritance (kế thừa) trong TypeScript

- Một **class** có thể tái sử dụng các thuộc tính và phương thức của class khác. Cái này gọi là sự **inheritance** (kế thừa) trong TypeScript.
- Class kế thừa các thuộc tính và phương thức được gọi là **child class**(lớp con). Và class có các thuộc tính và phương thức được kế thừa được gọi là **parent class**(lớp cha).
- Sử dụng từ khóa **extends** để cho phép một lớp kế thừa từ một lớp khác
- Sử dụng **super()** trong constructor của lớp con để gọi constructor của lớp cha. Bạn cũng có thể sử dụng cú pháp **super.methodInParentClass()** để gọi **methodInParentClass()** trong phương thức của class con.

Inheritance (kế thừa) trong TypeScript


```
class Person {
        constructor(private firstName: string, private lastName: string) {
                this.firstName = firstName;
                this.lastName = lastName;
        }
        getFullName(): string {
                return `${this.firstName} ${this.lastName}`;
        }
       describe(): string {
                return `This is ${this.firstName} ${this.lastName}.`;
        }
}
```

```
class Employee extends Person { //.. }
```

Để gọi constructor của lớp cha trong constructor của lớp con, sử dụng cú pháp **super()**, ví dụ:

```
class Employee extends Person {
        constructor( firstName: string, lastName: string, private jobTitle: string)
{
                // call the constructor of the Person class:
                super(firstName, lastName);
        }
}
```

```
let employee = new Employee('John', 'Doe', 'Web Developer');
console.log(employee.getFullName());
console.log(employee.describe());
```

Abstract Classes trong TypeScript

- Một abstract class(lớp trừu tượng) là một lớp cơ sở, thường được sử dụng để định nghĩa các hành vi chung cho các lớp dẫn xuất(lớp kế thừa). Không giống như lớp bình thường, một lớp abstract không thể được khởi tạo trực tiếp.
- Để khai báo một lớp abstract, bạn sử dụng từ khóa **abstract** :

```
abstract class Employee { //... }
```

  - Các lớp trừu tượng không thể được khởi tạo.
  - Một lớp trừu tượng có ít nhất một phương thức trừu tượng.
  - Để sử dụng một lớp trừu tượng, bạn cần kế thừa nó và cung cấp thực hiện các xử lý cho các phương thức trừu tượng

Abstract Classes trong TypeScript

- Thông thường, một lớp abstract chứa một hoặc nhiều phương thức abstract .
- Một phương thức của lớp abstract không chứa các thực thi code bên trong. Nó chỉ định nghĩa tên của phương thức mà không thực thi gì. Các thực thi của phương thức abstract sẽ được thực hiện bên trong lớp dẫn xuất.

```
abstract class Employee {
constructor(private firstName: string, private
lastName: string) { }
abstract getSalary(): number
get fullName(): string { return
`${this.firstName} ${this.lastName}`; }
 compensationStatement(): string { return
`${this.fullName} makes ${this.getSalary()} a
month.`;
} }
```

Giải thích lớp **Employee**

- Constructor khai báo các thuộc tính **firstName** và **lastName**.
- Phương thức **getSalary()** là một phương thức abstract. Lớp dẫn xuất sẽ thực thi logic dựa trên loại của employee.
- Phương thức **getFullName()** và **compensationStatement()** chứa chi tiết các thực thi.
- Lưu ý rằng phương thức **compensationStatement ()** gọi phương thức **getSalary()**.

Generics trong TypeScript

- Sử dụng generics trong TypeScript cho phép bạn viết các hàm, class, interfaces có thể tái sử dụng và tổng quát hóa.

```
function getRandomElement<T>(items: T[]): T {
            let randomIndex = Math.floor(Math.random() * items.length);
 return items[randomIndex];
}
```

Modules Trong TypeScript

- Một module trong TypeScript có chứa cả khai báo và code. Một module thực thi trong phạm vi riêng của nó, không phải trong phạm vi toàn cục. Điều đó có nghĩa là khi bạn khai báo các biến, hàm, lớp, giao diện, v.v., trong một module, chúng không hiển thị bên ngoài module, trừ khi bạn xuất chúng một cách rõ ràng bằng cách sử dụng câu lệnh **export**
- Mặt khác, nếu bạn muốn truy cập các biến, hàm, lớp, v.v., từ một module, bạn cần nhập chúng bằng cách sử dụng câu lệnh **import**

```
import { Validator } from './Validator';
```

```
import * from 'module_name';
```

```
export interface Validator {
        isValid(s: string): boolean
}
```

```
interface Validator {
        isValid(s: string): boolean
}
export { Validator };
```

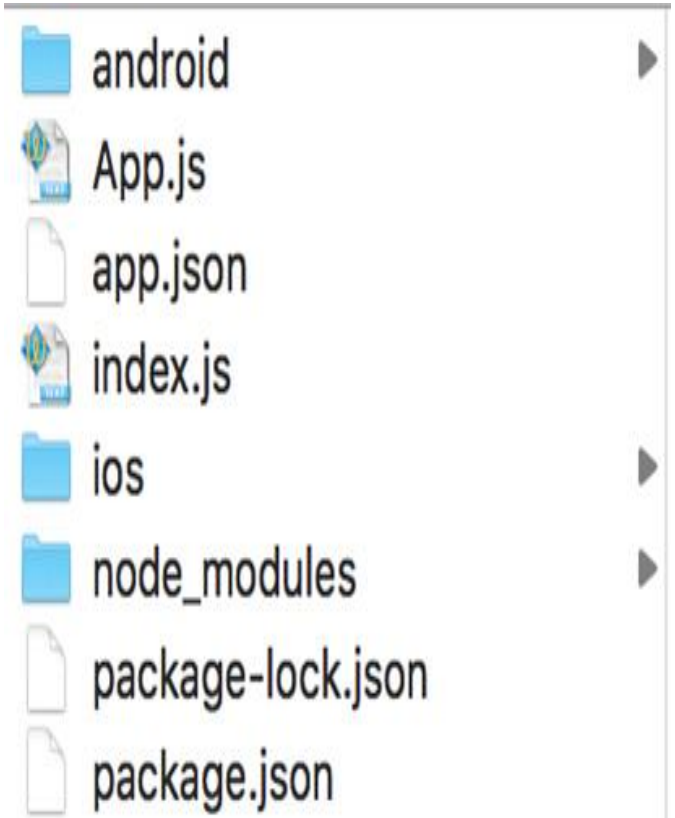
```
interface Validator { isValid(s: string): boolean }
export { Validator as StringValidator };
```

HCMUTE

KHOA CÔNG NGHỆ THÔNG TIN

React Native

Các thành phần cơ bản của dự án

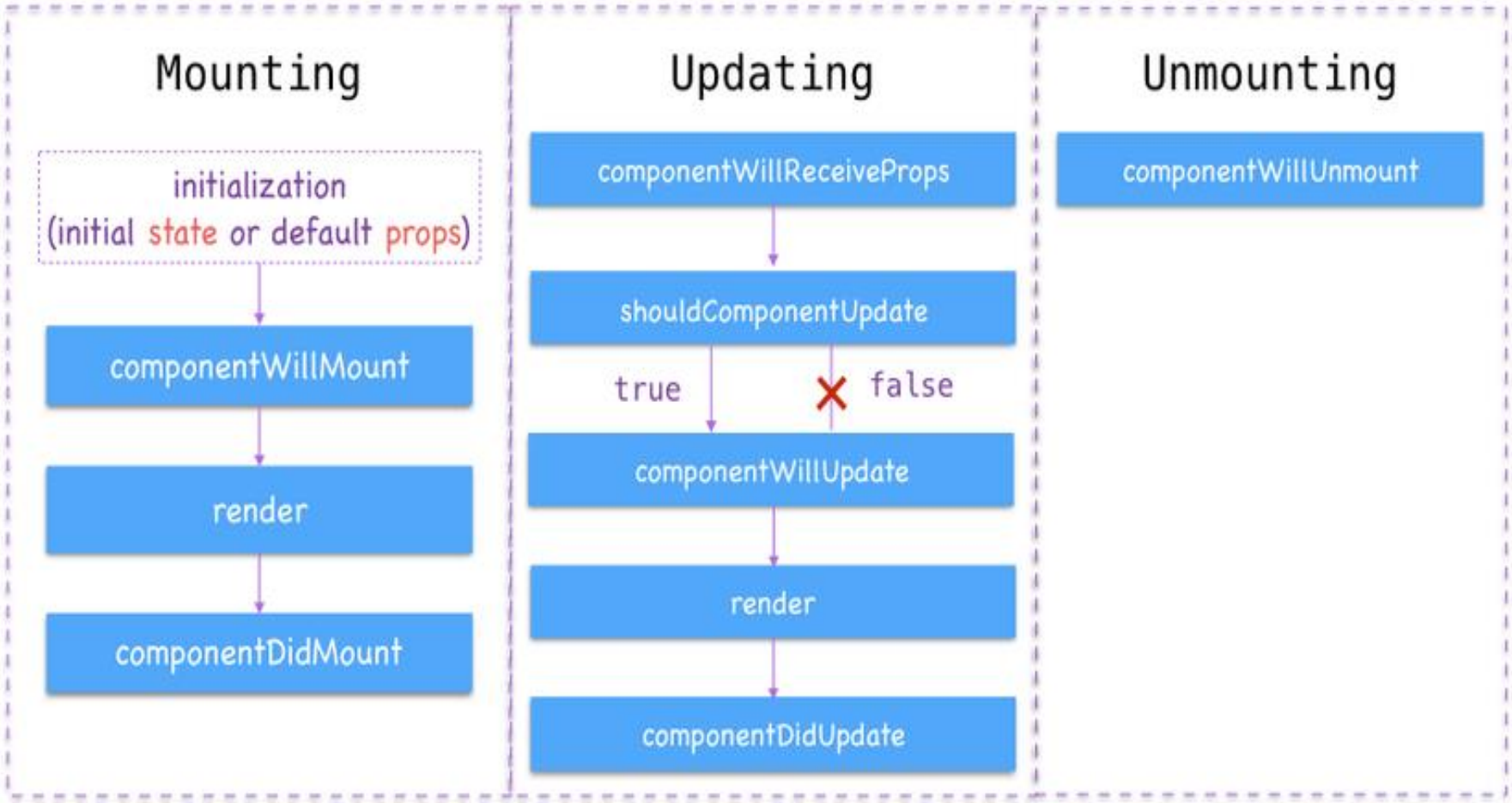


- **Thư mục Android**: chứa toàn bộ source build ứng dụng Android.
- **Thư mục IOS**: chứa toàn bộ source build ứng dụng IOS.
- **Thư mục node_modules**: chứa toàn bộ các package (thư viện) cần để chạy một ứng dụng react-native.
- **File package.js**: file quản lý các package nodejs đi kèm với dự án, sử dụng dòng lệnh npm install để tải toàn bộ thư viện yêu cầu của dự án về.
- **File package-lock.js** file được general sau khi chạy cài đặt npm install
- **File index.js**: file đầu tiên được binding khi chạy ứng dụng. File này sẽ đăng ký một component, component này sẽ được load lên đầu tiên khi chạy, mặc định ứng dụng sẽ đăng ký component trong App.js
- **File app.json**: file config tên ứng dụng và tên hiển thị.
- **File App.js** là một component mặc định có sử dụng một số Component khác như Text, View...

Component trong React-Native

- Component là một thành phần cơ bản trong ứng dụng react-native. Mọi view, screen đều được kế thừa từ lớp component này
- **Các thành phần cơ bản của component**
  - State
  - Props

Vòng đời của component



Các thành phần cơ bản của component


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

export default class App extends Component {

    constructor(props) {
        super(props);
        this.state = {
            message: "Welcome React-native"
        }
    }
    render() {
        return (
            <View>
                <Text>{this.state.message}</Text>
            </View>
        );
    }
}
```

- **State** - là biến điều khiển trạng thái nội bộ của 1 component. State có thể thay đổi bằng cách gọi hàm this.setState({...}). Mỗi lần thay đổi state hàm render sẽ được gọi lại ngay sau đó (hàm render chỉ thay đổi những thành phần có liên quan đến những giá trị trong state bị thay đổi). Chúng ta nên bỏ các biến có liên quan đến UI vào trong state này, để khi state thay đổi, UI màn hình sẽ được vẽ lại và thay đổi theo.
- **Lưu ý:** Không được thay đổi state trực tiếp bằng cách gọi this.state = {...} nếu sử dụng thay đổi state trực tiếp toàn bộ component này sẽ không còn hoạt động đúng như mong muốn nữa.
- **Props** - là các thuộc tính được thằng sử dụng truyền vào. Đây là các thông số được truyền vào để tùy chỉnh theo ý muốn của người xây dựng Component. Khác với state chúng ta không được thay đổi props ở trong chính bản thân của nó. Chúng ta chỉ nên đọc các thuộc tính được truyền vào để sử dụng mà thôi.
  Ví dụ sử dụng props: cũng là ví dụ trên nhưng chúng ta custom một số thứ để bạn có thể hiểu rõ hơn về props.

Các thành phần cơ bản của component


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

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

    render() {
        return (
            <Text>{this.props.message}</Text>   /*Sử dụng props được truyền từ ngoài vào.*/
        );
    }
}

export default class App extends Component {
    constructor(props) {
        super(props);
        this.state = {
            message: "Welcome to Code 101 - React-native"
        }
    }

    render() {
        return (
            <CustomText message={this.state.message} /> /*truyền 1 props vào cho thằng con sử dụng.*/
        );
    }
}
```

Một số hàm đặc biệt

- **Hàm this.setState()** - Hàm dùng để thay đổi state của component. Đây là phương thức chính để cập nhật giao diện người dùng. Khi hàm này thực thi xong thì hàm **render()** sẽ được tự động gọi lại. **Những giá trị nào của state thay đổi thì chỉ những thành phần có sử dụng biến state tương ứng đó được gọi để vẽ lại UI**.

  Lưu ý: hàm này chạy bất đồng bộ nên chúng ta không nên đọc giá trị sau khi gọi hàm này.

```
this.setState({
        message: "Chào mừng",
        key: "Value",
})
console.log(this.state.message) //không nên
// không sử dụng this.state ngay sau khi vừa set xong
// biến truyền vào cho hàm setState là một đối tượng có dạng key: value.
```

Một số hàm đặc biệt

- Có thể sử dụng callback để check dữ liệu hoặc xử lý một số tác vụ sau khi thay đổi trạng thái.

```
this.setState({
        message: "Chào mừng"
}, ()=>{
        console.log(this.state.message)   // kết quả: Chào mừng
})
```

•**Hàm forceUpdate()** - Mặc định hàm render() sẽ được gọi khi props hoặc state thay đổi. Nhưng nếu một vài thành phần UI sử dụng một số dữ liệu khác state hoặc prop muốn thay đổi, thì chúng ta cần thông báo cho React biết để vẽ lại toàn bộ bằng cách gọi hàm forceUpdate().

Một vài lưu ý nhỏ khi dùng React-Native

- Dữ liệu cần in ra màn hình và cần thay đổi lại UI khi nó thay đổi thì đặt vào state.
- Dữ liệu không cần thay đổi UI khi nó thay đổi thì có thể dùng this.xxx như vậy biến này có thể thực hiện thao tác = (gán) và sử dụng trực tiếp như các biến thông thường.
- Dữ liệu trong prop thì không nên thay đổi.
- Trong **state** chỉ nên chứa dữ liệu, không nên chứa các **View / Component** vào trong state. Làm như vậy có thể gây double dữ liệu và việc quản lý UI trở nên phức tạp hơn và khó tùy biến sau này.

View

Là một component được sử dụng với mục đích chia các view con theo hàng dọc hoặc hàng ngang dựa vào thuộc tính flexDirection trong style là 'column/row' (dọc / ngang), hoặc sử dụng để chứa nhiều view con hoặc khi cần in ra màn hình một view không hiển thị gì hết ví dụ như trong cấu trúc toán tử:

```
{
        (Điều kiện) ?  <Text> Text Message <Text> : <View/>
}
```

flex: 1 - ở style sẽ giúp kéo view rộng hết khung chứa có thể.

Các Component thường sử dụng

HCMUTE

Text

Dùng để hiển thị 1 message lên màn hình. Có thể sử dụng text cố định hoặc in nội dung của một biến lên màn hình

```
<Text>Message Here<Text>
<Text>{variable_here}<Text>
```


CN TT

Các Component thường sử dụng

HCMUTE

Image

Dùng để hiển thị hình ảnh lên màn hình. Có 3 cách hiển thị:

Hiển thị ảnh Local:

```
<Image
        source={require('/react-native/img/favicon.png')}
/>
```

Hiển thị ảnh từ url:

```
<Image
        style={{width: 50, height: 50}}
        source={{uri: 'https://facebook.github.io/react-native/docs/assets/favicon.png'}}
/>
```

Các Component thường sử dụng

- Hiển thị ảnh base 64:

```
<Image
        style={{width: 66, height: 58}}
        source={{uri: '
/>
```

Resize Mode quen thuộc:

- cover: (mặc định) Hình ảnh sẽ giữ nguyên tỷ lệ. Ảnh sẽ lớn hơn hoặc bằng khung chứa.
- contain: Hình ảnh vẫn giữ nguyên tỷ lệ. Ảnh sẽ nhỏ hơn hoặc bằng khung chứa
- center: Căn giữa hình ảnh theo 2 chiều. Lấy phần ở giữa, gần giống với cover.
- repeat: Lặp lại hình ảnh để che hết phần kích thước ô chứa.
- stretch: Thay đổi tỷ lệ hình ảnh để kéo dãn bằng với ô chứa.

Các Component thường sử dụng

TouchableOpacity

Thay thế cho việc sử dụng Button để việc định dạng style giống nhau cho cả android và ios, TouchableOpacity có thể chứa bất kỳ view con nào, và nhớ lưu ý cách dùng sự kiện onPress giống như Button nhé.

```
<TouchableOpacity
        style={Styles.btnStyle}
        onPress={() => this.onPressTouchableOpacityDemo()}>
                <Text style={Styles.textAction}>Click Me</Text>
</TouchableOpacity>
```

Flatlist

Được sử dụng để hiển thị 1 danh sách lên màn hình.

```
<FlatList
        data={this.state.listData}
        renderItem={({ item }) => this.renderItem(item)}
        keyExtractor={(item, index) => index.toString()}
/>

/* Hiển thị chi tiết 1 item như thế nào */
renderItem(item) {
    return (
        <View style={Styles.containerItem}>
            <Image
                style={Styles.imgLogo}
                resizeMode={'contain'}
                source={item.image}
            />
            <Text>{item.title}</Text>
        </View>
    )
}
```

Một vài lưu ý khi sử dụng Flatlist:

- Khi một thành phần data (ví dụ data[0] = ...) của bạn thay đổi thường thì không vẽ lại UI cho nên bạn sẽ cần thêm một thuộc tính là extraData={this.state}. Lúc này mỗi lần state thay đổi thì danh sách lại được vẽ lại.
- Có thể sử dụng Flatlist để làm như GridView trong android dựa vào thuộc tính numColumns={colum} (colum là số cột).

Chuyển đổi giữa các màn hình


```
import React, { Component } from 'react';
import { StackNavigator } from 'react-navigation';
import { StyleSheet, View } from 'react-native';

// import toàn bộ các class Screen từ modules/screens (những class được xuất thông qua file modules/scr
import * as Screens from './modules/screens';


//Tạo StackNavigator từ thư viện react-navigation
const AppNavigator = StackNavigator({
    HOME: {
        screen: Screens.Home
    },
    STYLES: {
        screen: Screens.StyleDemo
    },
    COMPONENT: {
        screen: Screens.Components
    },
    PROPS: {
        screen: Screens.Props
    }
}, {
        headerMode: "screen"
    });

export default class App extends Component {
    render() {
        return (
            <View style={styles.container}>
```

Sử dụng thư viện react-navigation để chuyển đổi giữa các màn hình. Các bạn có thể tìm hiểu thêm về thư viện này tại (https://reactnavigation.org)

- Cài đặt thư viện: Vào dự án bạn tạo và chạy dòng lệnh sau để cài đặt thư viện

npm install --save react-navigation

Chuyển đổi giữa các màn hình

HCMUTE

Chuyển đổi màn hình: có 2 cách chuyển màn hình:

Chuyển đổi và xóa toàn bộ màn hình trước đó: params: {} - Đây là phần để bạn truyền dữ liệu qua màn hình kế tiếp. Bạn có thể truyền qua cho màn hình tiếp theo một đối tượng theo cú pháp này.

```
// chuyển qua màn hình PROPS đã khai báo trong App StackNavigator
let pageContinue = NavigationActions.reset({
        index: 0,
        actions: [NavigationActions.navigate({ routeName: "PROPS", params: {} })]
});
this.props.navigation.dispatch(pageContinue);
```

Chuyển đổi và giữ lại màn hình trước để quay lại: {} - Đây cũng là cách để bạn truyền một đối tượng qua cho màn hình kế tiếp. Mặc định nếu bạn hiển thị Status bar thì sẽ có phím quay về, nhưng nếu cần thiết có thể quay về bằng cách gọi hàm sau đây:

```
// chuyển qua màn hình PROPS đã khai báo trong App StackNavigator
this.props.navigation.navigate("PROPS");
//or
this.props.navigation.navigate("PROPS", {});
```

```
this.props.navigation.goBack();
```

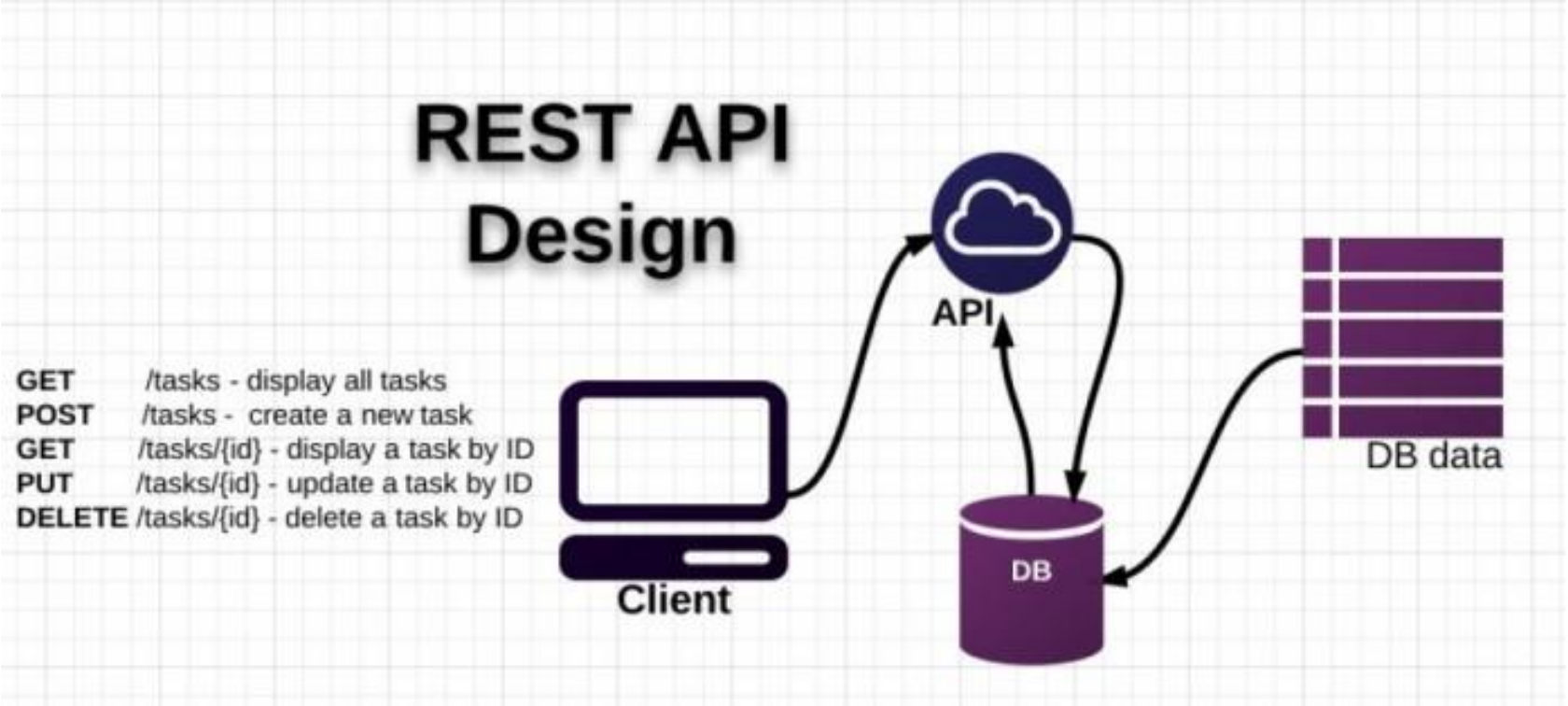
Chuyển đổi giữa các màn hình

Hiển thị Status bar:

```
static navigationOptions = ({ navigation }) => {
        return {
        title: "PROPS",
        headerStyle: {
                backgroundColor: Colors.primary
        },
        headerTintColor: Colors.white,
        headerTitleStyle: {
                alignSelf: 'center'
        }
        };
};
```

Giao tiếp Client vs Server

Giao tiếp Client vs Server



1. fetchData
2. Axios
3. websocket

Giao tiếp Client vs Server


```
import { getBaseURL } from '../configs/config';

let networkError = {
    error_code: -1,
    message: 'Network error',
    data: {}
};

export class RESTFulAPI {

    //Định nghĩa một api lấy language từ server.
    // Public api có sẵn tại https://api.ice5.skyx.app/get_languages

    getLanguage() {
        let api = getBaseURL() + "get_languages";
        return this.fetchData(api);
    }
```

Giao tiếp Client vs Server


```
    //Định nghĩa một hàm bất đồng bộ hỗ trợ các phương thức, GET, POST, PUT, DELETE (mặc định là GET)
    async fetchData(api, method = 'GET', body) {
        let headers = {
            Accept: 'application/json',
            'Content-Type': 'application/json',
        };
        try {
            let response = await fetch(api, {
                method: method,
                headers: headers,
                body: JSON.stringify(body)
            });
            let responseJson = await response.json();
            return responseJson;
        } catch (error) {
            return networkError;
        }
    }
}

export default RESTClient = new RESTFulAPI();
```

Lưu trữ dữ liệu

Lưu trữ dữ liệu

- React-Native mặc định hỗ trợ chức năng lưu trữ thông qua AsyncStorage được cung cấp mặc định trong gói thư viện react-native.
- Ngoài ra lưu trữ theo dạng dữ liệu có cấu trúc sử dụng realm database

HCMUTE

KHOA CÔNG NGHỆ THÔNG TIN

UTEx

Quy chuẩn tên biến và cấu trúc chương trình

- Quy chuẩn để:
  - Khi đọc lại bớt bỡ ngỡ (trước mình code cái gì vậy)
  - Người khác đọc vào biết bạn đang làm gì?
  - Có thể bạn khác join vào dự án biết cách sửa đổi.
  - Làm dự án lớn nhiều người tham gia.
  - ....

- **Tên biến và hàm:**
- **Một vài quy chuẩn tên biến mà mình cần tuân thủ như**:
  - Tên biến phải bắt đầu bằng ký tự viết thường.
  - Tên biến không được bắt đầu bằng số hoặc ký tự đặc biệt.
  - Những chữ cái đầu của mỗi từ đều viết hoa.
  - Tên biến phải mang ý nghĩa rõ ràng.
  - Nếu là style thì nên thêm viết tắt của view ở phía trước
  - **Một vài ví dụ về tên biến**:
    - maxNumber
    - minNumber
    - textMessageAnswer
    - btnActionAgree
- **Một vài quy chuẩn tên hàm**:
  - Tên hàm cũng bắt đầu bằng ký tự viết thường.
  - Tên hàm không chứa các ký tự đặc biệt.
  - Những chữ cái đầu của mỗi từ đều viết hoa.
  - Tên hàm phải mang ý nghĩa rõ ràng và thể hiện được chức năng của hàm.
  - **Một vài ví dụ về tên hàm**:
    - findMinOfTowNumber(firstNumber, secondNumeber){}
    - onPressBtnLanguage(){}
    - onPressNegativeAction(){}

Quy chuẩn tên biến và cấu trúc chương trình

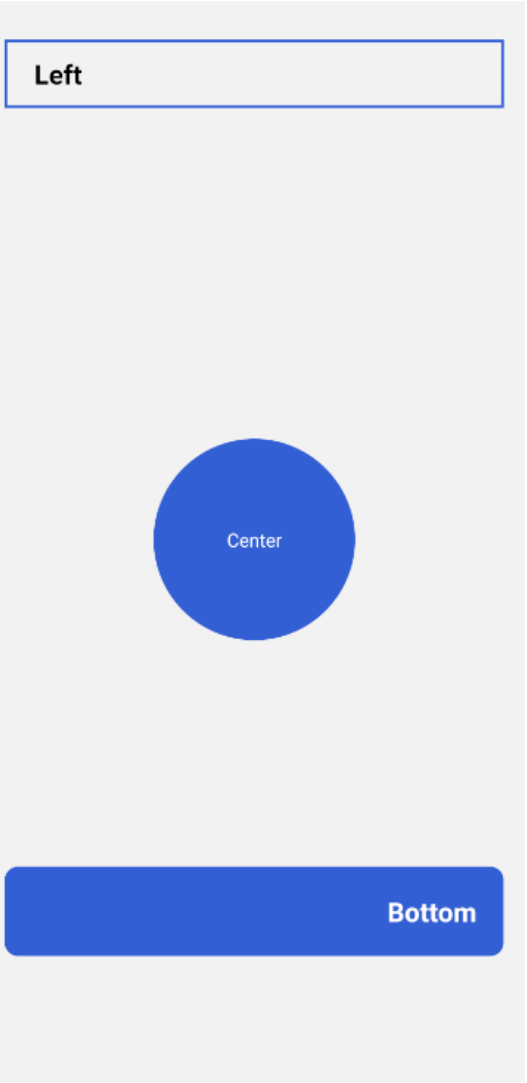
- **Cấu trúc chương trình:** Toàn bộ source code của chương trình sẽ được đặt trong thư mục app:
- **assets** là thư mục chứa resource của mình bao gồm các resource như custom font (fonts), hình ảnh (images), ngôn ngữ (languages)
- **configs** là thư mục chứa các cấu hình của ứng dụng: bao gồm các cấu hình server, link, màu sắc cơ bản.
- **libs** là thư mục chứa các thư viện cơ bản của mình để xử lý một số vấn đề nội bộ như:
  - **Database** (xử lý lưu trữ dữ liệu bằng database)
  - **Storage** (xử lý lưu trữ dữ liệu bằng storage)
  - **Language** (Cấu hình xử lý đa ngôn ngữ trong ứng dụng)
  - **RESTClient** (Cấu hình, danh sách các api truy cập hệ thống server)
  - **SoundPlayer** (Điều khiển âm thanh)
  - **Inapp** (Một vài cấu hình, xử lý thanh toán mua bán với store)
  - **Ads** (Cấu hình hiển thị quảng cáo từ bên thứ 3)
- **models** là thư mục chứa các model do mình định nghĩa, có thể là định nghĩa các đối tượng hoặc các loại của đối tượng
- **modules** là thư mục chứa các module do mình định nghĩa hoặc tùy biến lại. Trong đó bao gồm:
  - **screens** - module chứa toàn bộ xử lý màn hình của ứng dụng
  - **views** - module chứa toàn bộ view đã được custom.
  - Và một số module mình muốn chỉnh sửa từ thư viện, thì có thể thêm vào đây để tùy biến.

HCMUTE

KHOA CÔNG NGHỆ THÔNG TIN

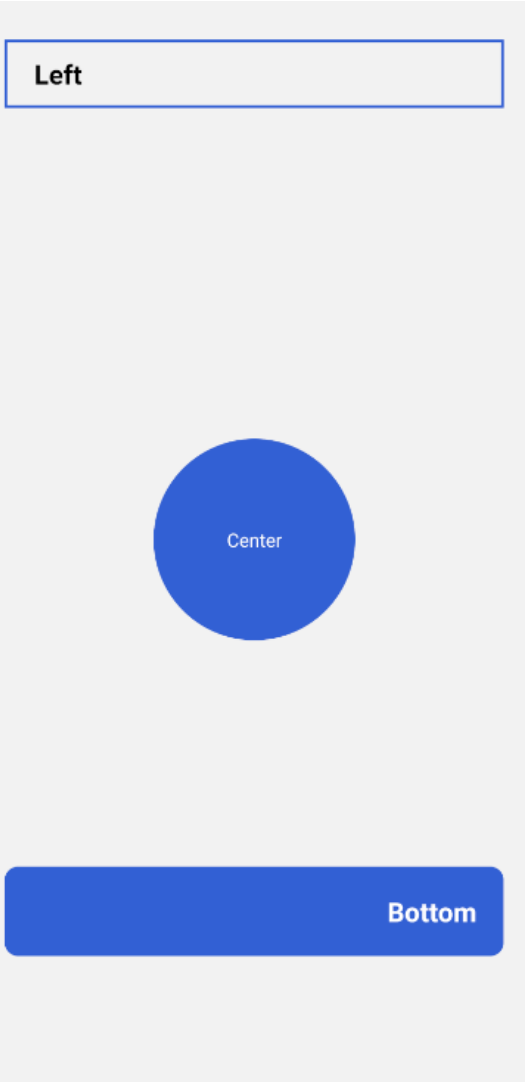
Bài tập 01: View and Style

Mô tả



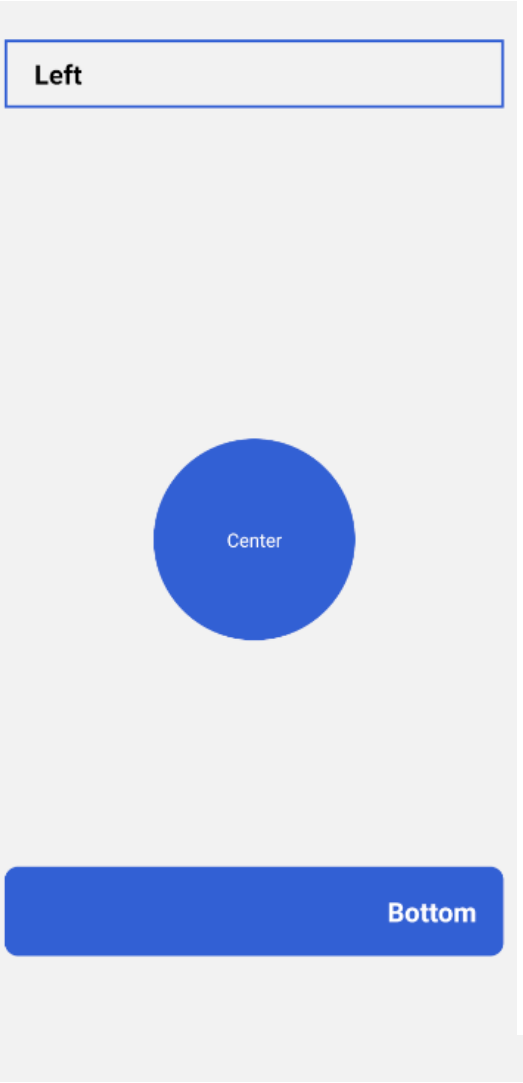
- Có 03 view: left, center, bottom nằm trong 01 view cha.
- Trong mỗi view sẽ có 01 component Text
- Thiết lập các stylesheet để có kết quả như trên.

Các bước




```
1   import { StatusBar } from 'expo-status-bar';
2   import { StyleSheet, Text, View } from 'react-native';
3
4   export default function App() {
5     return (
6       <View style={styles.container}>
7         <View style={styles.containerTop}>
8           <View style={styles.top}>
9             <Text style={styles.topText}>Left</Text>
10          </View>
11        </View>
12        <View style={styles.containerCenter}>
13          <View style={styles.center}>
14            <Text style={styles.centerText} >Center</Text>
15          </View>
16        </View>
17        <View style={styles.containerBottom}>
18          <View style={styles.bottom}>
19            <Text style={styles.bottomText}>Bottom</Text>
20          </View>
21        </View>
22        <StatusBar style="auto" />
23      </View>
24    );
25  }
```

Các bước làm



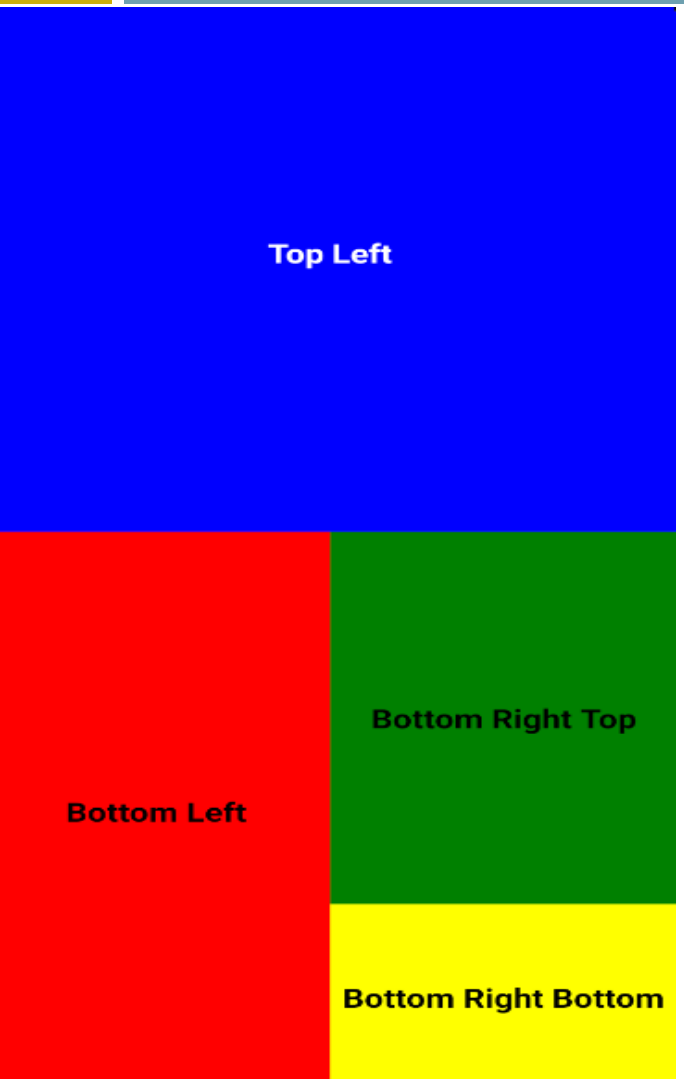

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#F2F2F2',

  },
  containerTop: {
    flex: 1,
    backgroundColor: '#F2F2F2',

  },
  containerCenter: {
    flex: 1,
    backgroundColor: '#F2F2F2',
    justifyContent:'center',
    alignItems:'center'

  },
  containerBottom: {
    flex: 1,
    backgroundColor: '#F2F2F2',
    justifyContent:'center'

  },
  top:{
    borderColor:'#3260D4',
    marginTop:50,
    marginHorizontal:20,
    paddingHorizontal:20,
    paddingVertical:10,
    borderWidth:2
  },
  topText:{
    color:'#000000',
    fontSize:20,
    fontWeight:'bold'
  },
  center:{
    width:150,
    height:150,
    justifyContent:'center',
    alignItems:'center',
    backgroundColor:'#3260D4',
    borderRadius:100,
    paddingHorizontal:50,
    paddingVertical:50
  },
  centerText:{
    color:'white'
  },
  bottom:{
    marginBottom:10,
    marginHorizontal:20,
    paddingHorizontal:20,
    paddingVertical:20,
   backgroundColor:'#3260D4',
    borderRadius:10
  },
  bottomText:{
    textAlign:'right',
    color:'#ffffff',
    fontSize:20,
    fontWeight:'bold'
  }
});
```

HCMUTE

KHOA CÔNG NGHỆ THÔNG TIN

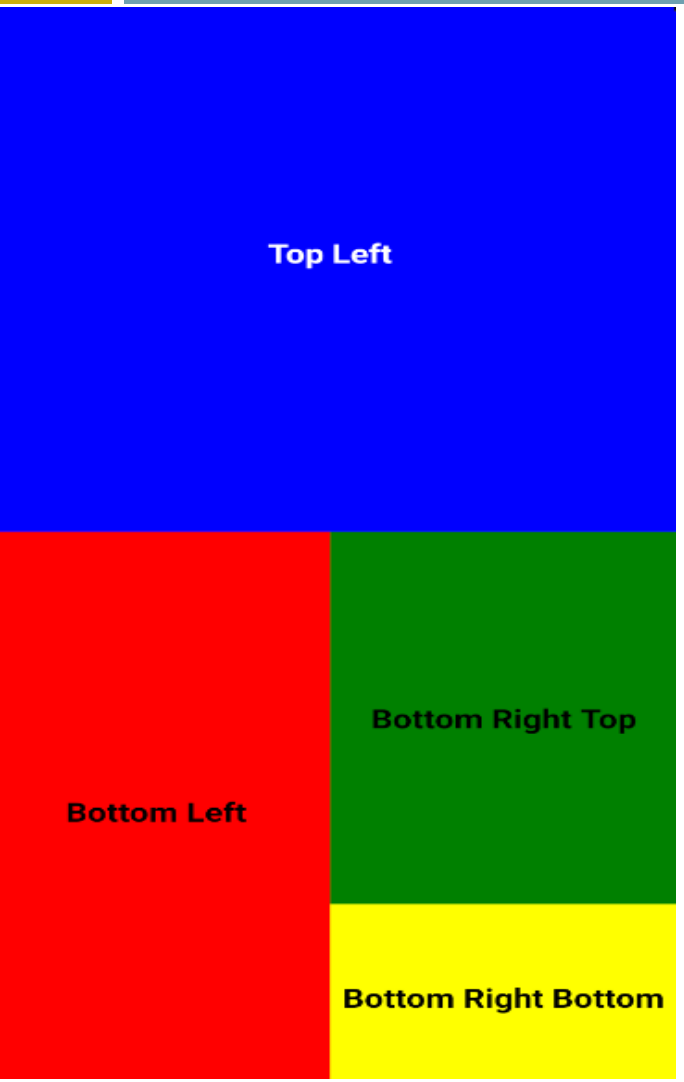
Bài tập 02: Hiểu về Flex

Mô tả



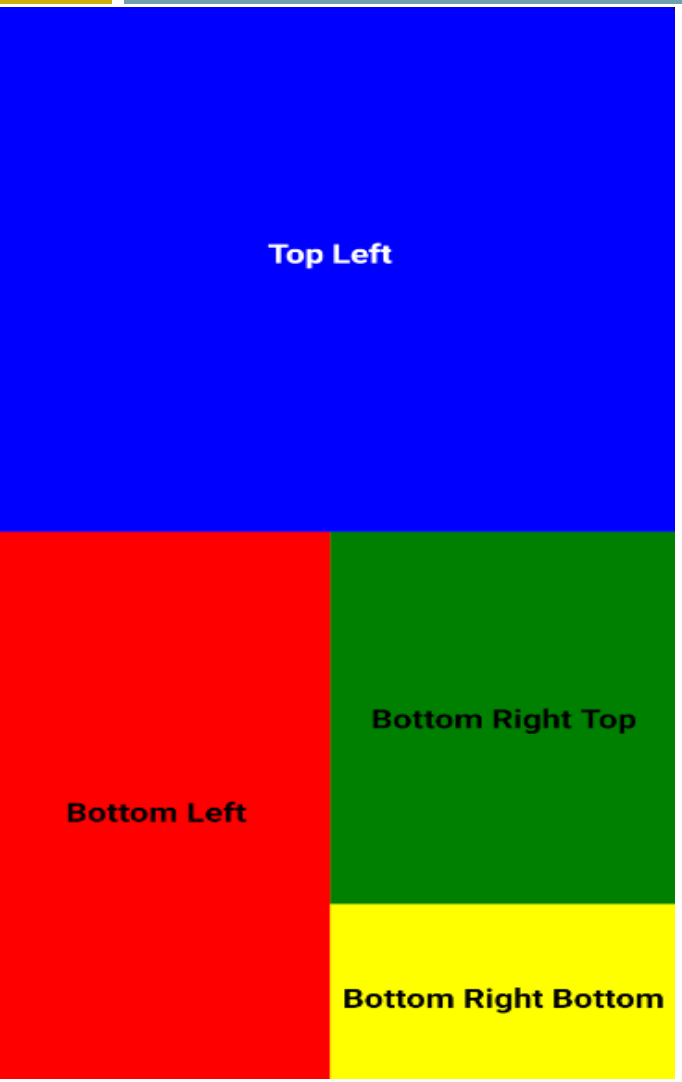
- Sử dụng thuộc tính Flex để xác định kích thước các View

Các bước




```jsx
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <View style={styles.containerTop}>

        <Text style={styles.Text}>Top Left</Text>

      </View>
      <View style={styles.containerBottom}>
        <View style={styles.bottomLeft}>
          <Text style={[styles.Text,styles.bottomText]}>Bottom Left</Text>
        </View>
        <View style={styles.bottomRight}>
          <View style={styles.bottomRightTop}>
            <Text style={[styles.Text,styles.bottomText]}>Bottom Right Top</Text>

          </View>
          <View style={styles.bottomRightBottom}>
            <Text style={[styles.Text,styles.bottomText]}>Bottom Right Bottom</Text>

          </View>
        </View>

      </View>
      <StatusBar style="auto" />
    </View>
  );
}
```

Các bước




```
const styles = StyleSheet.create({
  container: {
    flex: 1
  },
  containerTop: {
    flex: 1,
    backgroundColor: 'blue',
    justifyContent:'center',
    alignItems:'center'
  },
  containerBottom: {
    flex: 1,
    flexDirection:'row'
  },
  Text:{
    color:'#ffffff',
    fontSize:20,
    fontWeight:'bold'
  },
  bottomLeft:{
    flex:1,
    backgroundColor: 'red',
    justifyContent:'center',
    alignItems:'center'
  },
  bottomText:{
    color:'#000000',
  },
  bottomRight:{
    flex:1,
    flexDirection:'column'
  },
  bottomRightTop:{
    flex:2,
    backgroundColor:"green",
    justifyContent:'center',
    alignItems:'center'
  },
  bottomRightBottom:{
    flex:1,
    backgroundColor: 'yellow',
    justifyContent:'center',
    alignItems:'center'
  }

});
```

HCMUTE

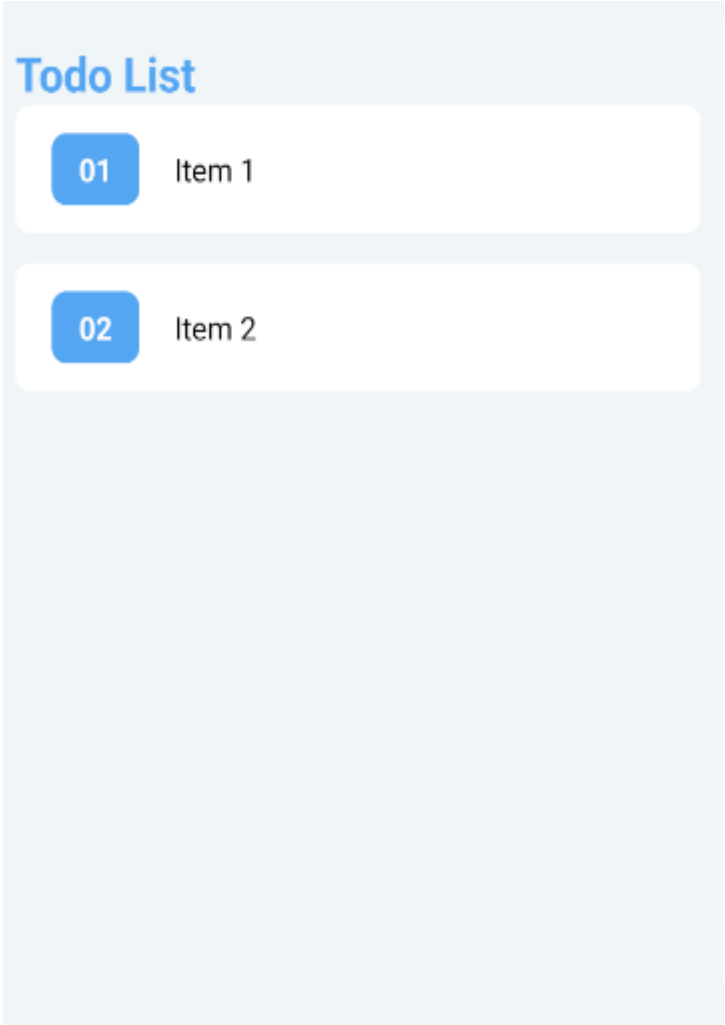
KHOA CÔNG NGHỆ THÔNG TIN

Bài tập 03: App công việc

Giới thiệu

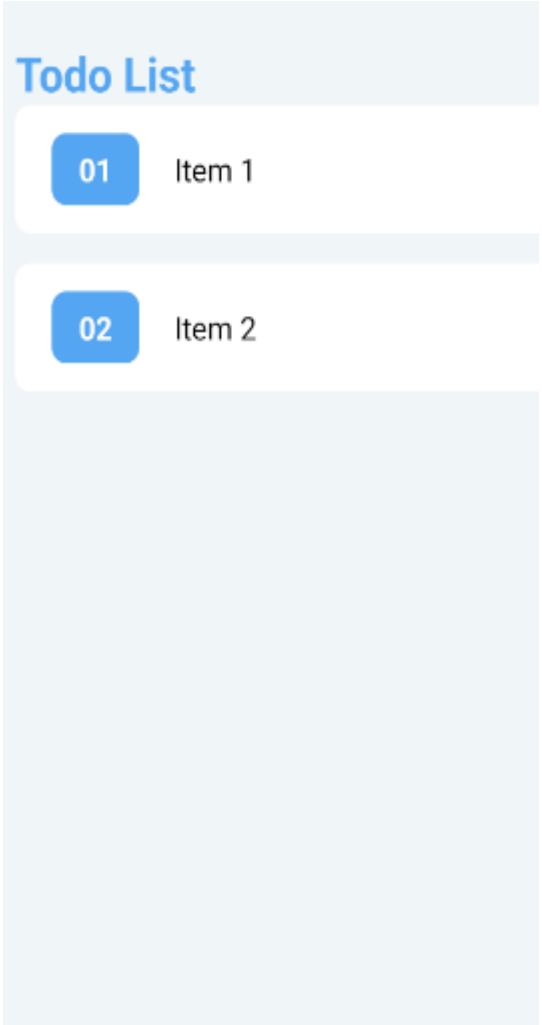
- Sử dụng View, Text, Input Text, stylesheet, ScrollView, TouchpadOpacity

Giao diện phần Task



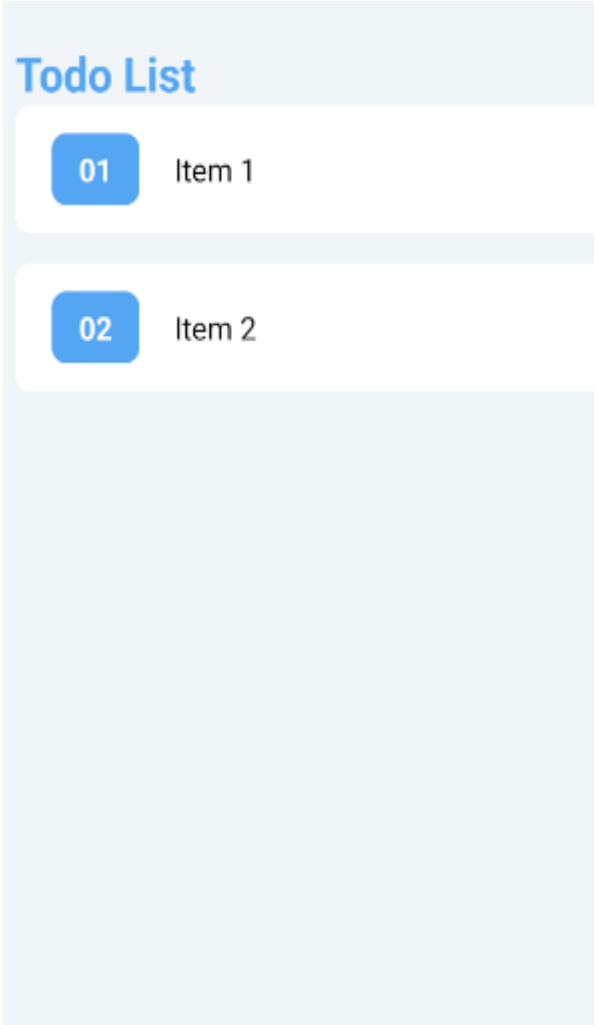

```
1  import { StyleSheet, Text, View } from 'react-native';
2
3  export default function App() {
4    return (
5      <View style={styles.container}>
6        <View style={styles.containerTop}>
7          <Text style={styles.Text}>Todo List</Text>
8          <View style={styles.item}>
9            <View style={styles.square}>
10             <Text style={styles.number}>01</Text>
11           </View>
12           <Text style={styles.content}>Item 1</Text>
13         </View>
14         <View style={styles.item}>
15           <View style={styles.square}>
16             <Text style={styles.number}>02</Text>
17           </View>
18           <Text style={styles.content}>Item 2</Text>
19         </View>
20       </View>
21       <View style={styles.containerBottom}>
22       </View>
23     </View>
24   );
25 }
```

Giao diện phần Task



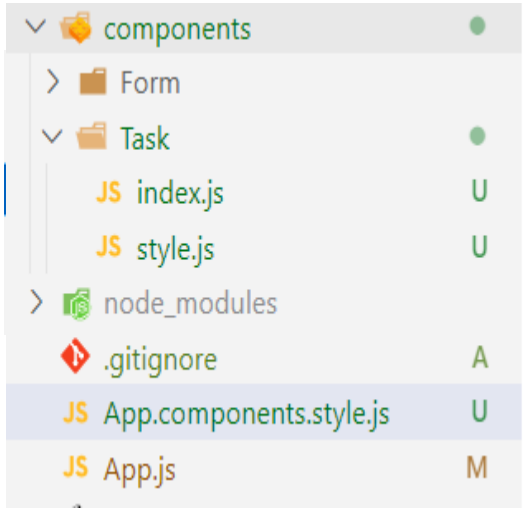

```
27  const styles = StyleSheet.create({
28    container: {
29      flex: 1,
30      backgroundColor:'#F0F6F8'
31    },
32    containerTop: {
33      flex: 1,
34      paddingTop:50,
35      paddingHorizontal:18
36    },
37    Text:{
38      color:'#54A6F2',
39      fontSize:24,
40      fontWeight:'bold'
41    },
42    item:{
43      flexDirection:'row',
44      backgroundColor:'#ffffff',
45      marginBottom:15,
46      paddingHorizontal:20,
47      paddingVertical:14,
48      borderRadius:8,
49      alignItems:'center',
50      justifyContent:'space-between'
51    },
52    square:{
53      width:48,
54      height:36,
55      borderRadius:8,
56      backgroundColor:'#54A6F2',
57      alignItems:'center',
58      justifyContent:'center'
59    },
60    number:{
61      fontSize:16,
62      fontWeight:'700',
63      color:'#fff'
64    },
65    content:{
66      width:'80%',
67      fontSize:16
68    },
```

Touch và ScrollView




```jsx
import { StyleSheet, Text, View, TouchableOpacity, ScrollView } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <View style={styles.containerTop}>
        <Text style={styles.Text}>Todo List</Text>
        <ScrollView style={styles.items}>
          <TouchableOpacity>
          <View style={styles.item}>
              <View style={styles.square}>
                <Text style={styles.number}>01</Text>
              </View>
              <Text style={styles.content}>Item 1</Text>
          </View>
          </TouchableOpacity>
          <TouchableOpacity>
          <View style={styles.item}>
              <View style={styles.square}>
                <Text style={styles.number}>02</Text>
              </View>
              <Text style={styles.content}>Item 2</Text>
          </View>
          </TouchableOpacity>

        </ScrollView>
      </View>
      {/* <View style={styles.containerBottom}>
      </View> */}
    </View>
  );
}
```

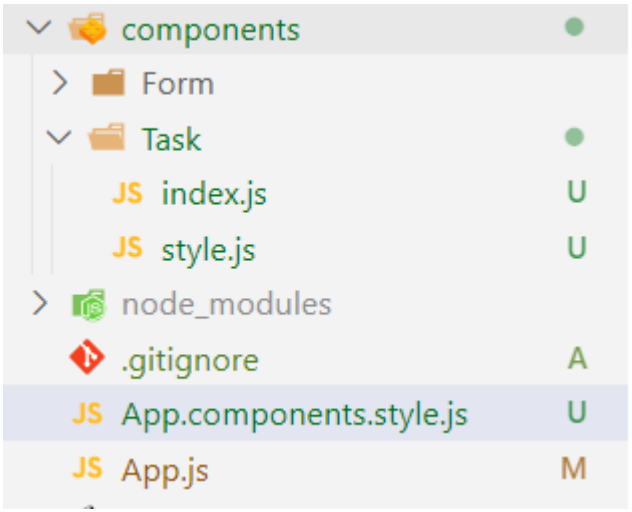
Phân tách thành phần



Tập tin App.js

```
import { StyleSheet, Text, View, TouchableOpacity, ScrollView } from 'react-native';
import Task from './components/Task';
import styles from './App.components.style';
export default function App() {
  return (
    <View style={styles.container}>
      <View style={styles.containerTop}>
          <Text style={styles.Text}>Todo List</Text>
        <ScrollView style={styles.items}>
          {/* gọi components Task */}
          <Task />

        </ScrollView>
      </View>
      {/* <View style={styles.containerBottom}>
      </View> */}
    </View>
  );
}
```

Phân tách thành phần

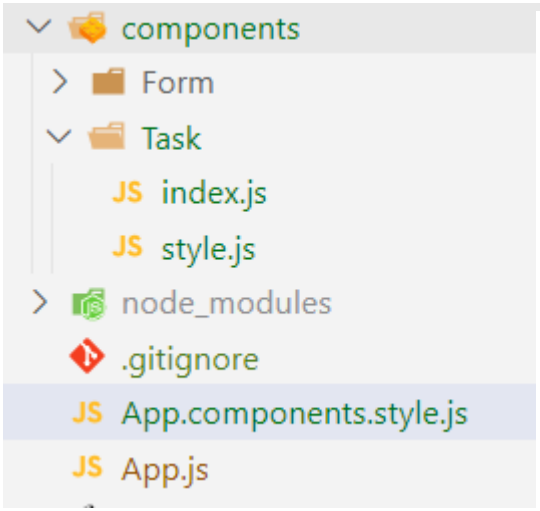


Tập tin
App.components.style.js

App.components.style.js > default

```
1   import { StyleSheet } from 'react-native'
2   const styles = StyleSheet.create({
3       container: {
4           flex: 1,
5           backgroundColor:'#F0F6F8'
6         },
7         containerTop: {
8           flex: 1,
9           paddingTop:50,
10          paddingHorizontal:18
11        },
12        Text:{
13          color:'#54A6F2',
14          fontSize:24,
15          fontWeight:'bold'
16        },
17        items:{
18          marginTop:25
19        },
20
21        containerBottom: {
22          flex: 1,
23          flexDirection:'row'
24        }
25
26  export default styles;
27
```

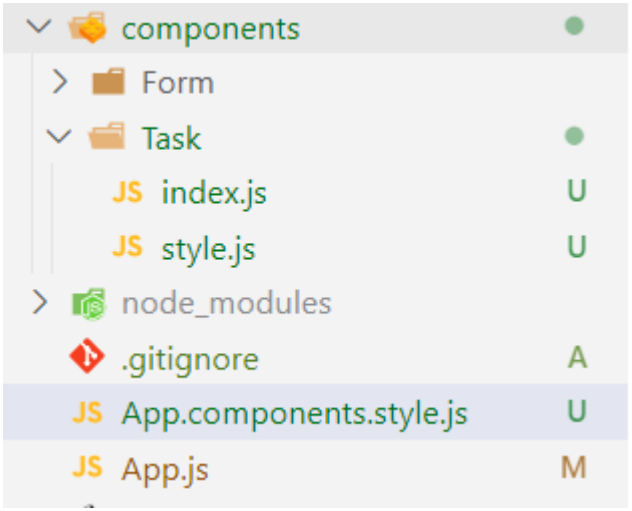
Phân tách thành phần




```
1  import { View, Text, TouchableOpacity } from 'react-native'
2  import React from 'react'
3  import styles from './style'
4  const Task  = () => {
5    return (
6      <TouchableOpacity>
7              <View style={styles.item}>
8                  <View style={styles.square}>
9                    <Text style={styles.number}>01</Text>
10                 </View>
11                 <Text style={styles.content}>Item 1</Text>
12             </View>
13     </TouchableOpacity>
14   )
15 }
16 export default Task;
```

Tập tin Task/index.js

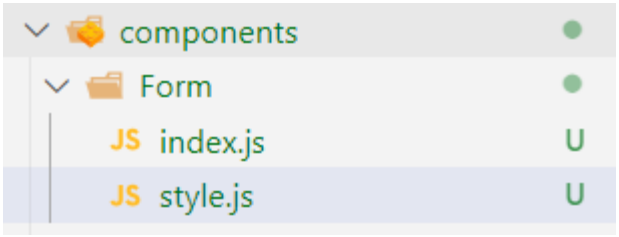
Phân tách thành phần

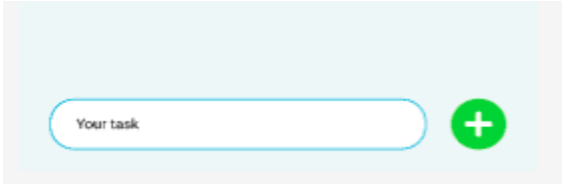


Tập tin Task/style.js

```
1  import { StyleSheet} from 'react-native';
2
3  const styles = StyleSheet.create({
4    item:{
5      flexDirection:'row',
6      backgroundColor:'#ffffff',
7      marginBottom:15,
8      paddingHorizontal:20,
9      paddingVertical:14,
10     borderRadius:8,
11     alignItems:'center',
12     justifyContent:'space-between'
13   },
14   square:{
15     width:48,
16     height:36,
17     borderRadius:8,
18     backgroundColor:'#54A6F2',
19     alignItems:'center',
20     justifyContent:'center'
21   },
22   number:{
23     fontSize:16,
24     fontWeight:'700',
25     color:'#fff'
26   },
27   content:{
28     width:'80%',
29     fontSize:16
30   },
31
32 });
33 export default styles;
```

Giao diện phần Form





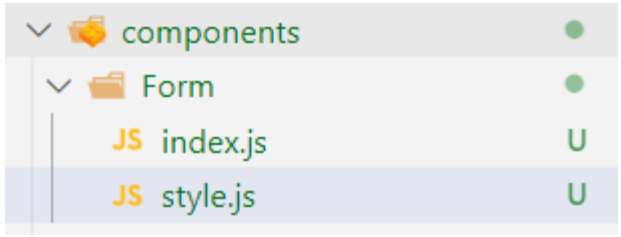

```
import Form from
'./components/Form'
Trong App.js
```

Tập tin Form/Index.js

components > Form > JS index.js > Form

```
1  import { View, Text, TextInput, TouchableOpacity, KeyboardAvoidingView } from 'react-native'
2  import React from 'react'
3  import styles from './style'
4  import { Platform } from 'react-native'
5
6  const Form = () => {
7    return (
8      <KeyboardAvoidingView
9        behavior={Platform.OS==="ios"?"padding":"height"}
10       keyboardVerticalOffset={10}
11       style={styles.containerBottom}>
12       <TextInput placeholder="You task" style={styles.inputText} />
13       <TouchableOpacity>
14           <View style={styles.addButton}>
15             <Text style={styles.addText}>+</Text>
16           </View>
17       </TouchableOpacity>
18     </KeyboardAvoidingView>
19   )
20 }
21 export default Form;
```

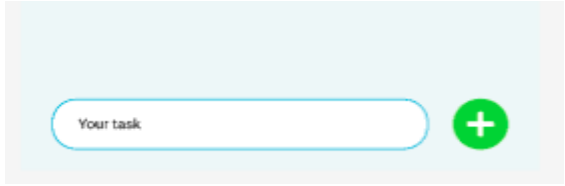
Giao diện phần Form



Tập tin Form/style.js

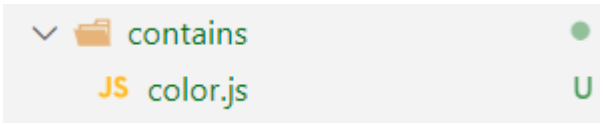
components > Form > style.js > styles > inputText

```
import { StyleSheet} from 'react-native'
const styles = StyleSheet.create({
    containerBottom: {
      flexDirection:'row',
        bottom:30,
        paddingHorizontal :20,
        width:'100%',
        justifyContent:'space-between',
        alignItems:'center'
      },
      addButton:{
        width:44,
        height:44,
        backgroundColor:'green',
        borderRadius:44,
        justifyContent:'center',
        alignItems:'center',
        borderWidth:1,
        borderColor:'#21a3d8'
      },
      addText:{
        fontSize:24,
        fontWeight:'bold',
        color:'#fff'
      },
      inputText:{
        height:44,width:'80%', backgroundColor:'#fff',
        borderRadius:20, borderWidth:1,
        borderColor:'#21a3d8',
        paddingHorizontal:20
      }
})
export default styles;
```




```
import Form from
'./components/Form'
Trong App.js
```

Cấu hình màu chung



contains > JS color.js > default

```
1   export default {
2       background:'#F0F6F8',
3       primary: '#21a3d0',
4       second : '#53d6f2',
5       white : '#fff',
6       black: '#000',
7       text:'#54A6F2'
8
9   }
10
```

```
import color from '../../contains/color'

const styles = StyleSheet.create({
    containerBottom: {
      flexDirection:'row',
        bottom:30,
        paddingHorizontal :20,
        width:'100%',
        justifyContent:'space-between',
        alignItems:'center'
      },
      addButton:{
        width:44,
        height:44,
        backgroundColor:'green',
        borderRadius:44,
        justifyContent:'center',
        alignItems:'center',
        borderWidth:1,
        borderColor:color.primary
      },
```

Xử lý Logic: Add Task

- Sử dụng onPress trong TouchableOpacity

```
<TouchableOpacity
        onPress={()=> alert('Xin chào')}>
          <View style={styles.addButton}>
            <Text style={styles.addText}>+</Text>
          </View>
</TouchableOpacity>
```

```
Viết 01 function
const handleAddTask = () =>{
    alert('Xin chào');
  }
```

```
Gọi function
onPress={handleAddTask}
```

Xử lý Logic: Add Task

- Sử dụng local state: useState để lưu thông tin

```
import { View, Text, TextInput, TouchableOpacity, KeyboardAvoidingView } from 'react-native'
import React, {useState} from 'react'
import styles from './style'
import { Platform } from 'react-native'

const Form = () => {
  const [task,setTask]  = useState('')
  const handleAddTask = () =>{
    if(task.length===0){
      alert("Bạn vui lòng nhập công việc")
      return false;
    }
    alert(task);
  }
  return (
    <KeyboardAvoidingView
      behavior={Platform.OS==="ios"?"padding":"height"}
      keyboardVerticalOffset={10}
      style={styles.containerBottom}>
      <TextInput placeholder="You task" style={styles.inputText}
        onChangeText={(text) => setTask(text)}

      />
      <TouchableOpacity
        onPress={handleAddTask}>
          <View style={styles.addButton}>
            <Text style={styles.addText}>+</Text>
          </View>
      </TouchableOpacity>
    </KeyboardAvoidingView>
  )
}
export default Form;
```

Thông tin liên lạc

- Nguyễn Hữu Trung
- 0908617108
- trungnh@hcmute.edu.vn
- utex.hcmute.edu.vn