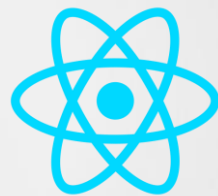


Áp dụng typescript vào react



Nội dung

Đây là những gì bạn sẽ tìm thấy trong slide này:

- I. Ngôn ngữ Typescript và các cú pháp cơ bản
- II. Cài đặt thủ công sử dụng typescript vào react
- III. Cấu hình, cài đặt và sử dụng typescript với reactjs qua create-react-app



Ngôn ngữ Typescript và các cú pháp cơ bản

Giới thiệu ngôn ngữ Typescript và các cú pháp cơ bản

Typescript là gì

Khái niệm, nguồn gốc, cách triển khai, ứng dụng

01

Tại sao nên sử dụng Typescript

Lý do, ưu nhược điểm

02

Các kiến thức cơ bản về Typescript

Các kiến thức cơ bản về Typescript

03

TypeScript so với JavaScript

Lợi ích, khác biệt khi sử dụng js

04

Typescript

3.1 Cài đặt và chạy chương trình đầu tiên

Cài đặt, môi trường, cú pháp

3.2 Kiểu dữ liệu và khai báo biến

Kiểu dữ liệu, khai báo biến

3.3 Function trong TypeScript

Cú pháp, kiểu dữ liệu trả về

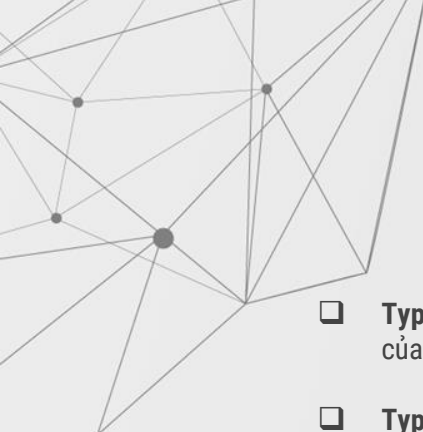
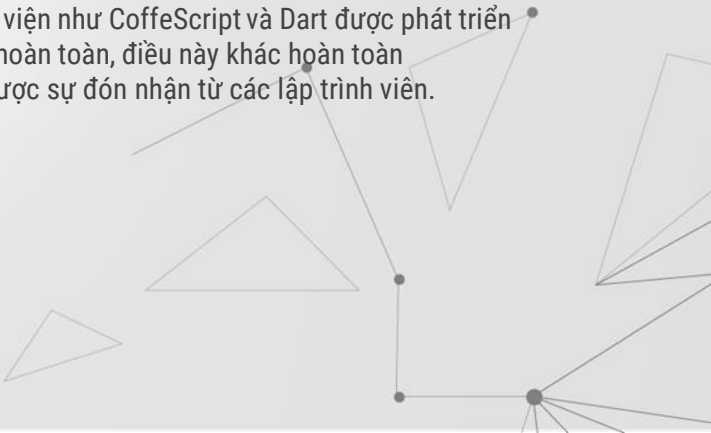
3.4 Hướng đối tượng trong Typescript

Tính đóng gói, Tính đa hình, Tính trừu tượng, Interface

01

Typescript là gì

TypeScript là một dự án mã nguồn mở được phát triển bởi **Microsoft**. Nó có thể được coi là một phiên bản nâng cao của Javascript bởi việc bổ sung tùy chọn **kiểu tĩnh** và **lớp hướng đối tượng** mà điều này không có ở Javascript.

- 
-
- ❑ **TypeScript** là một dự án mã nguồn mở được phát triển bởi **Microsoft**. Nó có thể được coi là một phiên bản nâng cao của Javascript bởi việc bổ sung tùy chọn **kiểu tĩnh** và **lớp hướng đối tượng** mà điều này không có ở Javascript.
 - ❑ **TypeScript** có thể sử dụng để phát triển các ứng dụng chạy ở client-side (Angular) và server-side (NodeJS).
 - ❑ **TypeScript** sử dụng tất cả các tính năng của của ECMAScript 2015 (ES6) như classes, modules.
 - ❑ Trưởng nhóm dự án này là **Anders Hejlsberg**, người đã đóng góp cũng như tạo ra các ngôn ngữ khác C#, Turbo Pascal và Delphi.
 - ❑ **TypeScript** không phải ra đời đầu tiên mà trước đây cũng có một số thư viện như CoffeScript và Dart được phát triển bởi Google, tuy nhiên điểm yếu là hai thư viện này sử dụng cú pháp mới hoàn toàn, điều này khác hoàn toàn với **TypeScript**, vì vậy tuy ra đời sau nhưng **TypeScript** vẫn đang nhận được sự đón nhận từ các lập trình viên.
- 

Sự quan tâm theo thời gian

● Typescript
Ngôn ngữ lập trình

+ So sánh

Trên toàn thế giới ▾

07/09/2010 - 07/10/2019 ▾

Tất cả danh mục ▾

Tìm kiếm trên web ▾

Sự quan tâm theo thời gian ?





02

Tại sao nên sử dụng Typescript



Tại sao nên sử dụng Typescript

- **TypeScript** giúp chúng ta phát triển các dự án lớn một cách dễ dàng.
- Hiện nay có nhiều Javascript Framework khuyến khích sử dụng **Typescript**. Ví dụ: AngularJS, Ionic ...
- Hỗ trợ các tính năng của Javascript phiên bản mới nhất.
- **TypeScript** là một mã nguồn mở nên bạn hoàn toàn có thể sử dụng mà không mất phí
- Bản chất của **Typescript** vẫn là **Javascript** - **TypeScript** được biên dịch tạo ra các đoạn mã **javascript** nên bạn có thể chạy bất kì ở đâu miễn ở đó có hỗ trợ biên dịch **Javascript**. Ngoài ra bạn có thể sử dụng trộn lẫn cú pháp của Javascript vào bên trong **TypeScript**, điều này giúp các lập trình viên tiếp cận **TypeScript** dễ dàng hơn.



TS

Tại sao nên sử dụng Typescript

Code Suggestions

- Nếu một nhà phát triển mới gia nhập nhóm và anh ta đang sử dụng lại một thành phần, TypeScript sẽ gợi ý các đạo cụ cần có.

```
1  import * as React from 'react';
2  interface IProps {
3    title: string;
4    status: boolean;
5  }
6  export default class Item extends React.Component<IProps> {
7    render() {
8      return (
9        <div>
10         <h2>{this.props.}</h2>
11         </div>
12       );
13     }
14   }
15
```

children?

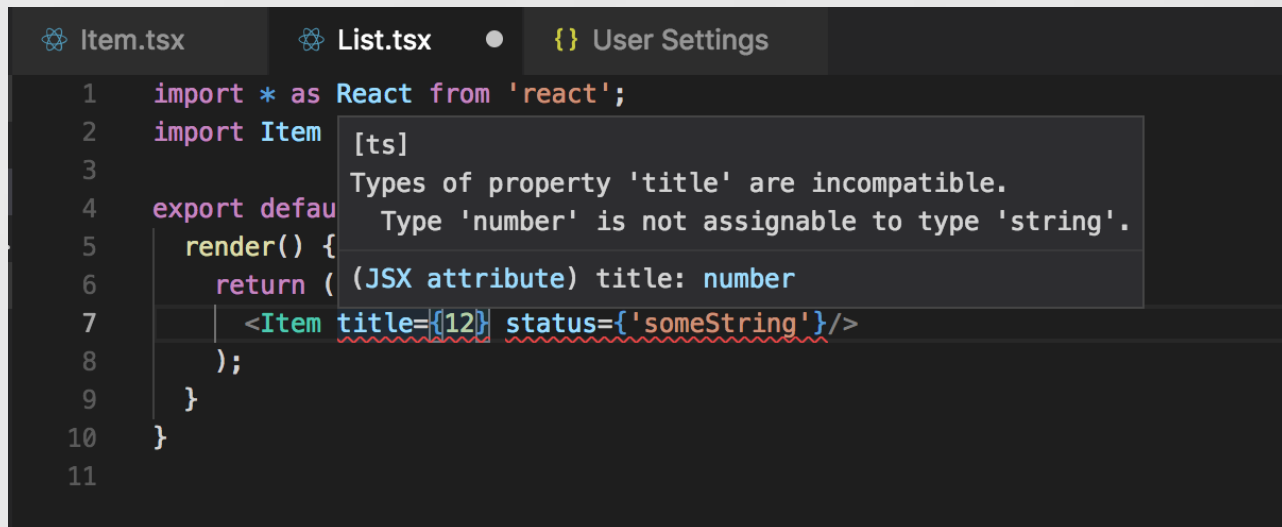
status (property) status: boolean ⓘ

title

Tại sao nên sử dụng Typescript

Đánh dấu các lỗi càng sớm càng tốt

- Các lỗi sẽ được báo sớm hơn để kịp thời sửa chữa



```
Item.tsx  List.tsx  {} User Settings

1  import * as React from 'react';
2  import Item
3
4  export default
5    render() {
6      return (
7        <Item title={12} status={'someString'}/>
8      );
9    }
10 }
11
```

Tại sao nên sử dụng Typescript

Self-documenting code

- The type, interfaces, enums, types for function parameter and its return type, etc tất cả được thể hiện 1 cách tường minh

```
class SinhVien {  
    HoTen: string;  
    Tuoi: number;  
    Lop: string;  
  
    constructor(hoten: string, tuoi: number, lop: string) {  
        this.HoTen = hoten;  
        this.Tuoi = tuoi;  
        this.Lop = lop;  
    }  
  
    show(): string {  
        return this.HoTen + ' - ' + this.Lop + ' - ' + this.Tuoi;  
    }  
}  
  
var sv = new SinhVien("Dinh Phuc", 21, "CNPM");  
console.log(sv.show());  
  
//extends
```



SinhVien
+ string: HoTen + number: Tuoi + string: Lop
+ string show()



3.1

Cài đặt và chạy chương trình đầu tiên

Cài đặt



Node js

Cài đặt node js

npm install -g typescript

Typescript



Compile

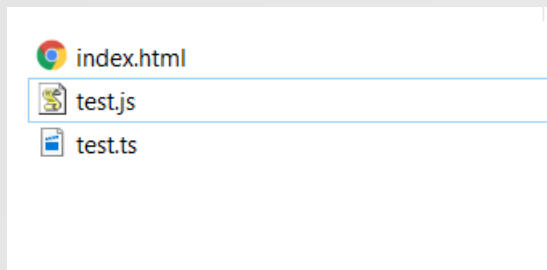
- **Typescript** có đuôi mở rộng là **.ts**
- Để biên dịch một file **Typescript** sang **javascript** ta chạy lệnh **tsc tên_file -watch**

Ví dụ

File html: 1.html

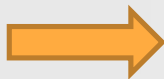
```
1.<!DOCTYPE html>
2.<html lang="en">
3.<head>
4.    <meta charset="UTF-8">
5.    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6.    <meta http-equiv="X-UA-Compatible" content="ie=edge">
7.    <title>Hello word</title>
8.    <script src="test.js"></script>
9.    <script src="sinhvien.js"></script>
10.</head>
11.<body>
12.    <center><h1>Áp dụng typescript vào react </h1></center>
13.</body>
14.</html>
```

Ví dụ



File html: test.ts

```
1. //init
2. console.log('Hello word');
3. var a: number;
4. a = 1;
5. console.log(a);
```



File html: test.js

```
1. console.log('Hello word');
2. var a;
3. a = 1;
4. console.log(a);
```




3.2

Kiểu dữ liệu và khai báo biến

3.2) Kiểu dữ liệu và khai báo biến

	Javascript	Typescript
Kiểu dữ liệu cơ bản	number, string, Boolean, array, enum, tuple, any, void ...
Cú pháp khai báo	var a = 123;	var test : int;

- Như đã nói ở trên thì bản chất của **Typescript** vẫn là Javascript nên các kiểu dữ liệu cơ bản của java script thì **Typescript** đều có ngoài ra **Typescript** còn có một số kiểu dữ liệu khác như là enum, tuple, any, void ...
- Còn về mặt cú pháp khai báo biến trong **Typescript** hơi khác một chút đó là khi khai báo chúng ta cần khai báo thêm cho nó xem là thuộc kiểu dữ liệu nào.

3.2) Kiểu dữ liệu và khai báo biến

```
1.//Kiểu dữ liệu
2.// string
3.var string1 : string;
4.string1 = '1001';
5.// KDL number
6.var number1 : number = 10;
7.
8.//KDL mảng string
9.var arrString : string[];
10.arrString = ['teo', 'ty', 'tun'];
11.
12.// KDL mảng number
13.var arrNumber : number[];
14.arrNumber = [1, 2, 3];
```

```
1.//KDL boolean
2.var boolean1 : boolean = true;
3.console.log(boolean1);
4.
5.// KDL enum
6.enum Color {Red, Green, Blue}
7.var c: Color = Color.Green;
8.
9.// KDL tuple - kiểu mảng dữ liệu hỗn tạp
10.var x: [string, number];
11.
12.x = ['string', 10];
13.for (let i = 0; i < x.length; i++) {
14.    console.log(x[i]);
15.}
16.
17.// KDL any
18.var xyz : any;
19.xyz = 'string';
20.console.log(xyz);
```

3.3

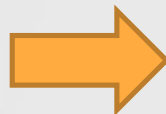
Function trong TypeScript



3.3) Function trong TypeScript

- Trong **typescript** các hàm đều trả về một kết quả, kết quả đó sẽ thuộc về một kiểu dữ liệu nào đó (vd: number, string, void, ...).
- Ta phải chỉ định luôn kiểu dữ liệu trả về cho hàm ngay từ đầu.

```
1. // number
2. function sum (x: number, y: number) : number {
3.     return x + y;
4. }
5. console.log(sum(2, 2));
6. // string
7. function showString() : string {
8.     return 'hello';
9. }
10. console.log(showString());
11. // array
12. function showArrNumber() : number[] {
13.     return [1, 2, 3];
14. }
15. console.log(showArrNumber());
```



```
4
hello
▼ (3) [1, 2, 3] ⓘ
  0: 1
  1: 2
  2: 3
  length: 3
  ► __proto__: Array(0)
```

3.4

Hướng đối tượng trong Typescript



3.4) Hướng đối tượng trong Typescript

Trong Typescript hỗ trợ chúng ta các tính chất về hướng đối tượng như:

Class

```
1. class SinhVien {  
2.   HoTen: string;  
3.   Tuoi: number;  
4.   Lop: string;  
5.  
6.   constructor(hoten: string, tuoi: number, lop: string) {  
7.     this.HoTen = hoten;  
8.     this.Tuai = tuoi;  
9.     this.Lop = lop;  
10.  }  
11.  
12.  show() {  
13.    return this.HoTen + ' - ' + this.Lop + ' - ' + this.Tuai;  
14.  }  
15.}  
16. var sv = new SinhVien("Dinh Phuc", 21, "CNPM");  
17. console.log(sv.show());
```



Dinh Phuc - CNPM - 21

> |

3.4) Hướng đối tượng trong Typescript

Trong Typescript hỗ trợ chúng ta các tính chất về hướng đối tượng như:

Access Modifier

Phạm vi truy cập trong class: Private, protected, public. Mặc định khi không khai báo thì là public

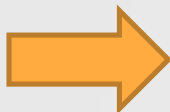
```
class SinhVien {  
  public HoTen: string;  
  private Tuoi: number;  
  Lop: string;  
  
  constructor(hoten: string, tuoi: number, lop: string) {  
    this.HoTen = hoten;  
    this.Tuoi = tuoi;  
    this.Lop = lop;  
  }  
  
  show() {  
    return this.HoTen + ' - ' + this.Lop + ' - ' + this.Tuoi;  
  }  
}  
  
var sv = new SinhVien("Dinh Phuc", 21, "CNPM");  
sv.  
con HoTen (property) SinhVien.HoTen: string  
    Lop  
    show
```


3.4) Hướng đối tượng trong Typescript

Trong Typescript hỗ trợ chúng ta các tính chất về hướng đối tượng như:

Tính kế thừa

```
1. class SinhVienIT extends SinhVien{
2.     Skill: string;
3.
4.     constructor(hoten: string, tuoi: number, lop: string, skill: string) {
5.         super(hoten, tuoi, lop);
6.         this.Skill = skill;
7.     }
8.     getFullInfor(){
9.         return this.HoTen + ' - ' + this.Lop + ' - '
10.        + this.Tuoi + " | Ky nang: " + this.Skill;
11.    }
12. var svIt = new SinhVienIT("Dinh Phuc", 21, "CNPM", "C#,C++,JS");
13. console.log(svIt.getFullInfor());
```



Dinh Phuc - CNPM - 21 | Ky nang: C#,C++,JS

> |

I

3.4) Hướng đối tượng trong Typescript

Trong Typescript hỗ trợ chúng ta các tính chất về hướng đối tượng như:

Interface

Interface trong TypeScript thì có thể khai báo được cả property

```
1.interface nguoi {  
2.   Ten: string;  
3.   Tuổi : number;  
4.}  
5.function xemtt(n : nguoi) : void {  
6.   console.log(`Xin chào ${n.Ten} bạn ${n.Tuoi} tuổi phải không`);  
7.}  
8.  
9.xemtt({Tuoi : 21, Ten : 'phucnd'});
```



Xin chào phucnd bạn 21 tuổi phải không

>

3.4) Hướng đối tượng trong Typescript

Trong Typescript hỗ trợ chúng ta các tính chất về hướng đối tượng như:

Abstract

```
1. //Abstract
2. interface Animal {
3.     Height: number;
4.     Weight: number;
5.
6.     Talk(): void;
7. }
8. class Cat implements Animal {
9.     Height: number;
10.    Weight: number;
11.
12.    constructor(height: number, weight: number) {
13.        this.Height = height;
14.        this.Weight = weight;
15.    }
16.
17.    Talk(): void {
18.        console.log("Meo");
19.    }
20. }
21. var cat = new Cat(20,3);
22. cat.Talk();
```

Namespace

- Đặt tên class mà không sợ bị trùng từ khóa. Quản lý theo một nhóm gọi là module có hệ thống.

```
1. //Namespace
2. module NuocNgot {
3.     export class String {
4.
5.     }
6.     export class Number {
7.
8.         test () : void {
9.             console.log('1 lit');
10.        }
11.    }
12.}
13.
14. var coca = new NuocNgot.String();
15. var pessi = new NuocNgot.Number();
16. pessi.test();
```

Generic

```
1.//Generic
2.//#Normal
3.function getDataNumber(x : number) : number {
4.    return x;
5.}
6.function getDataString(x : string) : string {
7.    return x;
8.}
9.
10.function getDataBoolean(x : boolean) : boolean {
11.    return x;
12.}
13.console.log(getDataNumber(9));
14.
15.//#Use Generic
16.function getData<T>(x : T) : T {
17.    return x;
18.}
19.console.log(getData<string>("day la string"));
```

```
1.class MayTinh {
2.
3.    static XemThongTin<T>(x : T[]) : void {
4.        for (var i = 0; i < x.length; i++) {
5.            console.log(x[i]);
6.        }
7.    }
8.}
9.MayTinh.XemThongTin<string>(['HP', 'Dell', 'Asus']);
10.MayTinh.XemThongTin<any>(['HP', 9000000, 'Asus']);
```



04

TypeScript so với JavaScript

4) TypeScript so với JavaScript

Chú thích kiểu

Một trong những khó khăn lớn nhất của JavaScript là việc theo dõi các vấn đề vì thiếu kiểm tra kiểu kết hợp với những thứ như ép buộc loại có thể gây ra kết quả không mong muốn cho những người không quen với JavaScript phức tạp. Khi sử dụng các ràng buộc về dữ liệu có thể tránh được tại thời gian biên dịch **bằng cách** sử dụng chú thích kiểu của TypeScript

```
1. // JavaScript
2. function getPassword(pass) {
3.     if(pass) {
4.         return 'password';
5.     }
6.     return '*****';
7. }
8.
9. let password = getPassword('false')
; // "password"
```

```
1. // TypeScript
2. function getPassword(pass: boolean) : string {
3.     if(clearTextPassword) {
4.         return 'password';
5.     }
6.     return '*****';
7. }
8.
9. let password = getPassword('false');
10. // throws: error TS2345: Argument of type '"false"' is not assignable to parameter
11. of type 'boolean'.
```

4) TypeScript so với JavaScript

Tính năng ngôn ngữ

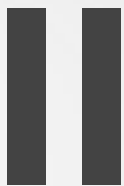
Ngoài việc phân tích kiểu tĩnh, TypeScript cũng thêm các tính năng sau vào JavaScript

- Interfaces
- Generics
- Namespaces
- Null checking
- Access Modifiers

4) TypeScript so với JavaScript

Khi nào cần lựa chọn: Typescript vs. JavaScript

JavaScript	TypeScript
<ul style="list-style-type: none">• Xây dựng các công cụ cần thiết : TypeScript đòi hỏi một bước xây dựng để tạo ra JavaScript cuối cùng được thực thi. Tuy nhiên, ngày càng trở nên hiếm hoi để phát triển các ứng dụng JavaScript mà không cần xây dựng các công cụ dưới bất kỳ hình thức nào.• Các dự án nhỏ : TypeScript có thể quá mức cần thiết cho các nhóm nhỏ hoặc các dự án lượng code nhỏ.• Typescript ra đời để giúp giải quyết vấn đề javascript, Angular hay bất cứ framework front-end nào đều chạy trên javascript, và lại Typescript cũng chỉ là superset của js, và nó cũng chỉ gói gọn trong việc học syntax nên bạn muốn theo sâu mảng này, kiến thức sâu về js là 1 điều bắt buộc.	<ul style="list-style-type: none">• Điểm cộng thứ nhất rõ ràng là với typescript code của chúng ta đẹp để và dễ đọc hơn rất nhiều.• Làm việc với thư viện mới hoặc Framework: Giả sử bạn đang dùng React cho một dự án mới. Bạn không quen thuộc với các API của React, nhưng vì chúng cung cấp các định nghĩa kiểu, bạn có thể nhận được intellisense sẽ giúp bạn điều hướng và khám phá các giao diện mới.• Các dự án lớn hoặc nhiều nhà phát triển: TypeScript có ý nghĩa nhất khi làm việc trên các dự án lớn hoặc bạn có một số nhà phát triển làm việc cùng nhau. Việc sử dụng các giao diện của TypeScript và các công cụ sửa đổi truy cập có thể là vô giá trong việc giao tiếp các API (các thành viên của một lớp có sẵn để sử dụng).



Cài đặt thủ công sử dụng typescript vào react

Cách cài đặt thủ công sử dụng typescript vào react

1) Cài đặt dự án

Hãy bắt đầu bằng cách tạo ra một dự án React mới và tích hợp TypeScript. Chạy các lệnh sau để bắt đầu dự án:

- 1.# Make a new directory
- 2.\$ mkdir react-typescript
- 3.
- 4.# Change to this directory within the terminal
- 5.\$ cd react-typescript
- 6.
- 7.# Initialise a new npm project with defaults
- 8.\$ npm init -y
- 9.
- 10.# Install React dependencies
- 11.\$ npm install react react-dom

1) Cài đặt dự án

Hãy bắt đầu bằng cách tạo ra một dự án React mới và tích hợp TypeScript. Chạy các lệnh sau để bắt đầu dự án:

1. # Make index.html and App.tsx in src folder
2. \$ mkdir src
3. \$ cd src
4. \$ touch index.html
5. \$ touch App.tsx
- 6.
7. # Open the directory in your favorite editor
8. \$ code .

1) Cài đặt dự án

Sau đó tạo một file index.html với nội dung sau:

```
1.<!DOCTYPE html>
2.<html>
3.<head>
4.  <meta charset="utf-8" />
5.  <meta http-equiv="X-UA-Compatible" content="IE=edge">
6.  <title>React + TypeScript</title>
7.  <meta name="viewport" content="width=device-width, initial-
  scale=1">
8.</head>
9.<body>
10.  <div id="main"></div>
11.  <script src="./App.tsx"></script>
12.</body>
13.</html>
```

1) Cài đặt dự án

Sử dụng **Parcel** như bundler trong dự án (có thể chọn sử dụng **webpack** hoặc **bundler** khác nếu muốn).



1. # Install Parcel to our DevDependencies
2. \$ npm i parcel-bundler -D
- 3.
4. # Install TypeScript
5. \$ npm i typescript -D
- 6.
7. # Install types for React and ReactDOM
8. \$ npm i -D @types/react @types/react-dom

1) Cài đặt dự án

We can update our package.json with a new task that will start our development server:

```
1. {
2.   "name": "react-typescript",
3.   "version": "1.0.0",
4.   "description": "",
5.   "main": "index.js",
6.   "scripts": {
7.     "dev": "parcel src/index.html",
8.     "test": "echo \"Error: no test specified\" && exit 1"
9.   },
10.  "keywords": [],
11.  "author": "",
12.  "license": "ISC",
13.  "dependencies": {
14.    "react": "^16.10.2",
15.    "react-dom": "^16.10.2"
16.  },
17.  "devDependencies": {
18.    "@types/react": "^16.9.5",
19.    "@types/react-dom": "^16.9.1",
20.    "parcel-bundler": "^1.12.4",
21.    "typescript": "^3.6.3"
22.  }
23. }
```

1) Cài đặt dự án

Bây giờ chúng ta có thể tạo một file Counter.tsx với một bộ đếm đơn giản:

```
1. import * as React from 'react';
2.
3. export default class Counter extends React.Component {
4.   state = {
5.     count: 0
6.   };
7.
8.   increment = () => {
9.     this.setState({
10.       count: (this.state.count + 1)
11.     });
12.   };
13.
14.   decrement = () => {
15.     this.setState({
16.       count: (this.state.count - 1)
17.     });
18.   };
19.
20. }
```


1) Cài đặt dự án

Bây giờ chúng ta có thể tạo một file Counter.tsx với một bộ đếm đơn giản:

```
1. render () {  
2.   return (  
3.     <div>  
4.       <h1>{this.state.count}</h1>  
5.       <button onClick={this.increment}>I  
ncrement</button>  
6.       <button onClick={this.decrement}>D  
ecrement</button>  
7.     </div>  
8.   );  
9. }
```

1) Cài đặt dự án

Sau đó, bên trong **App.tsx**, chúng ta có thể load Counter:

```
1. import * as React from 'react';  
2. import { render } from 'react-dom';  
3.  
4. import Counter from './Counter';  
5.  
6. render(<Counter />, document.getElementById('main'));
```

1) Cài đặt dự án

Khởi chạy dự án: **npm run dev**

```
PS D:\Topic\ReactTypeScript\App\test2\react-typescript> npm run dev

> react-typescript@1.0.0 dev D:\Topic\ReactTypeScript\App\test2\react-typescript
> parcel src/index.html

Server running at http://localhost:1234
✓ Built in 131ms.
█
```



Cấu hình, cài đặt và sử dụng typescript với reactjs qua create- react-app

Cấu hình, cài đặt và sử dụng typescript với reactjs qua **create-react-app**

1) Cài đặt create-react-app

Create-react-app là một công cụ cho phép các lập trình viên có thể tạo mới **một ứng dụng** React với các giá trị mặc định mà **không cần** quan tâm tới việc cấu hình như thế nào.

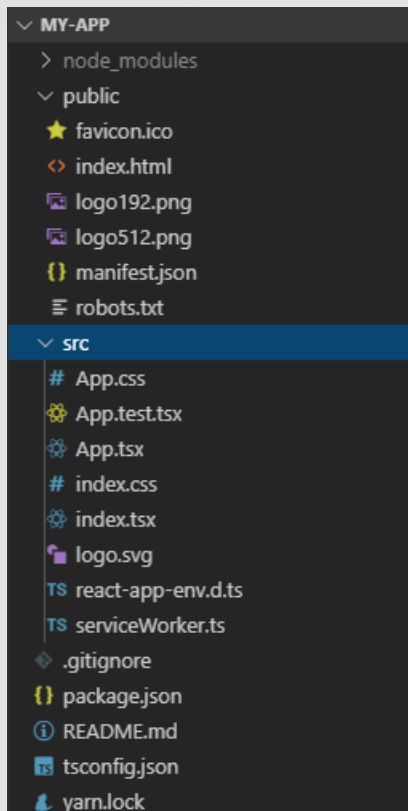
1. `# install`

2. `$ npm install -g create-react-app`

Sử dụng công cụ **create-react-app** với tùy chọn **--typescript** để tạo ra một ứng dụng React sử dụng **Typescript** làm cú pháp mặc định.

```
create-react-app my-app --typescript_
```

1) Cài đặt create-react-app



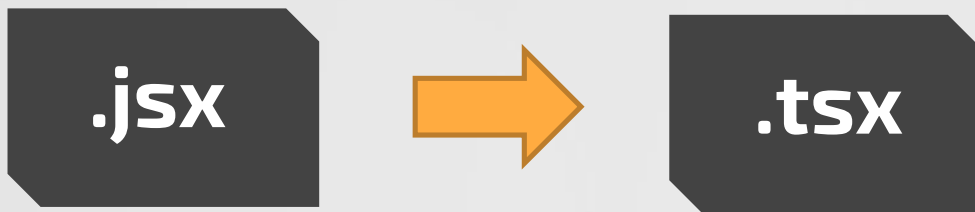
Ở đây thêm tùy chọn `--typescript`. Tùy chọn này sẽ quy định CRA sử dụng Typescript làm cú pháp mặc định và các cấu hình phù hợp để ứng dụng có thể được build và chạy.

Một số file, thư mục mà chúng ta quan tâm:

- *tsconfig.json* khai báo các tùy chọn cho việc **biên dịch** Typescript sang Javascript.
- *tslint.json* khai báo các cài đặt được sử dụng bởi TSLint (Một công cụ kiểm tra code Typescript)
- `public` thư mục chứa những tài nguyên tĩnh như file `index.html`
- `src` thư mục chứa code giao diện, logic của ứng dụng. Nó bao gồm các **components** viết bằng **Typescript** và **CSS** file. Trong thư mục này file **index.js** sẽ được thay thế bằng **index.tsx**

2) Những thay đổi khi dùng Typescript

Để xây dựng React với Typescript, chúng ta sẽ phải thực hiện một số thay đổi so với cách truyền thống.



Typescript có đuôi mở rộng cho file là **.ts**, có thể vì lý do này mà những file trong dự án **React** sẽ có đuôi mở rộng là **.tsx**. Trong những file này, chúng ta sẽ sử dụng những cú pháp của **Typescript** và **JSX**. Bạn luôn phải khai báo giá trị **"jsx": "preserve"** trong file **tsconfig.json**, việc này được đặt làm mặc định. Bạn có thể tìm hiểu thêm [tại đây](#).

3) Xây dựng các component

Một trong những **điểm lợi thế lớn nhất** của việc sử dụng **Typescript** có thể việc **không phải** sử dụng gói prop-types. Chúng ta sẽ sử dụng các **đặc điểm** của **Typescript** để xây dựng và quản lý **props** và **state**, nếu bạn làm chủ được vấn đề này, bạn sẽ thấy nó mạnh hơn các phương pháp mặc định của **React**.

Chúng ta sẽ định nghĩa một **props interface** (phương thức của **Typescript**) cho từng **component**, nơi mà bạn sẽ truyền các giá trị **props** khi khởi tạo. Việc này định nghĩa cấu trúc của đối tượng sẽ được truyền vào, bao gồm cả kiểu dữ liệu và các key.

Ngoài ra chúng ta có thể khai báo một **interface** cho **state** của **components**.

```
1. interface IProps{  
2.   count?: number;  
3. }  
4.  
5. interface IState{  
6.   count: number;  
7. }
```


4) Function component

Để lấy ví dụ cho **function component** với **props**, chúng ta sẽ thêm thẻ <Header/> giới thiệu trong file App.tsx

Bắt đầu bằng việc tạo file ./src/Header.tsx. function components này sẽ nhận vào một prop với key là name

```
1. import React from 'react'
2.
3. interface IProps{
4.   name?: string;
5. }
6. const Header: React.FC<IProps> =(props: IProps) =>{
7.   return <h1>{props.name} ! Wellcome to my project.</h1>
8. }
9.
10. Header.defaultProps = {
11.   name: 'Phucnd',
12. }
13. export default Header;
```

5) Class component

Để mô tả những nguyên tắc cơ bản của một class component, chúng ta sẽ thêm một thành phần vào nội dung của file App.tsx với việc xử lý một sự kiện đơn giản.

Tạo ra file `./src/Counter.tsx` với nội dung:

```
1. import * as React from 'react';
2. interface IProps {
3.   count?: number;
4. }
5. interface IState {
6.   count: number;
7. }
8. class Counter extends React.Component<IProps, IState> {
9.   public static defaultProps: Partial<IProps> = {
10.     count: 0,
11.   }
12.   public state: IState = {
13.     count: 0
14.   };
```

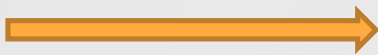
5) Class component

```
1.   this.setState({
2.     count: (this.state.count + 1)
3.   });
4. };
5. public decrement = () => {
6.   this.setState({
7.     count: (this.state.count - 1)
8.   });
9. };
10. public increment = () => {
11.   public render(): React.ReactNode {
12.     return (
13.       <div>
14.         <h1>{this.state.count}</h1>
15.         <button onClick={this.increment}>Increment</button>
16.         <button onClick={this.decrement}>Decrement</button>
17.       </div>
18.     );
19.   }
20. }
21. export default Counter;
```

6) Sử dụng Typescript component cho React app

Bên trong file **App.tsx**, chúng ta sẽ import những **component** vừa mới tạo ở trên, sửa lại nội dung mặc định của file.

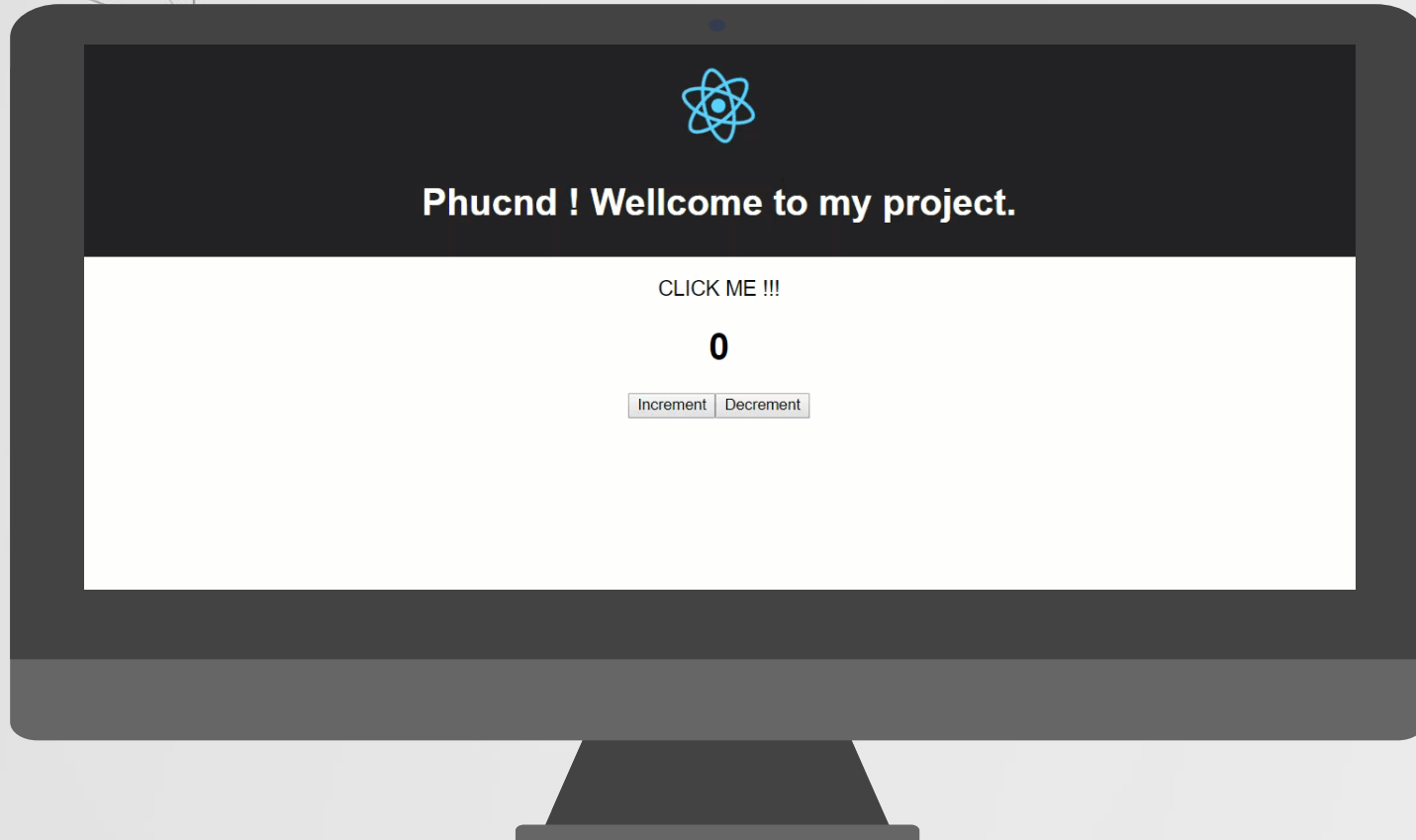
Các giá trị prop name (của Header) nên được gán giá trị có cùng kiểu đã định nghĩa trong component, các giá trị này đều là tùy chọn, chúng ta có thể không cần gán giá trị cho chúng, khi đó **defaultProps** sẽ được sử dụng.



6) Sử dụng Typescript componet cho React app

```
1. import * as React from 'react';
2. import Counter from './Counter'
3. import Header from './Header'
4. import './App.css';
5. import logo from './logo.svg';
6. class App extends React.Component {
7.   public render() {
8.     return (
9.       <div className="App">
10.        <header className="App-header">
11.          <img src={logo} className="App-logo" alt="logo" />
12.          <Header />
13.        </header>
14.        <p className="App-intro">
15.          CLICK ME !!!
16.        </p>
17.        <Counter />
18.      </div>
19.    );
20.  }
21.}
22.
23. export default App;
```

7) Application





THANKS

Does anyone have any questions?

Phucnd.zit@gmail.com
<https://dinhphuc.github.io>

RESOURCES

- [Using TypeScript with React](#)
- [Why and How to use TypeScript in your React App?](#)
- [Is React TypeScript a Good Combination?](#)
- [Xây dựng ứng dụng bằng React sử dụng Typescript](#)
- [TypeScript so với JavaScript](#)
- [Getting started with TypeScript](#)
- [TypeScript React Starter](#)

