

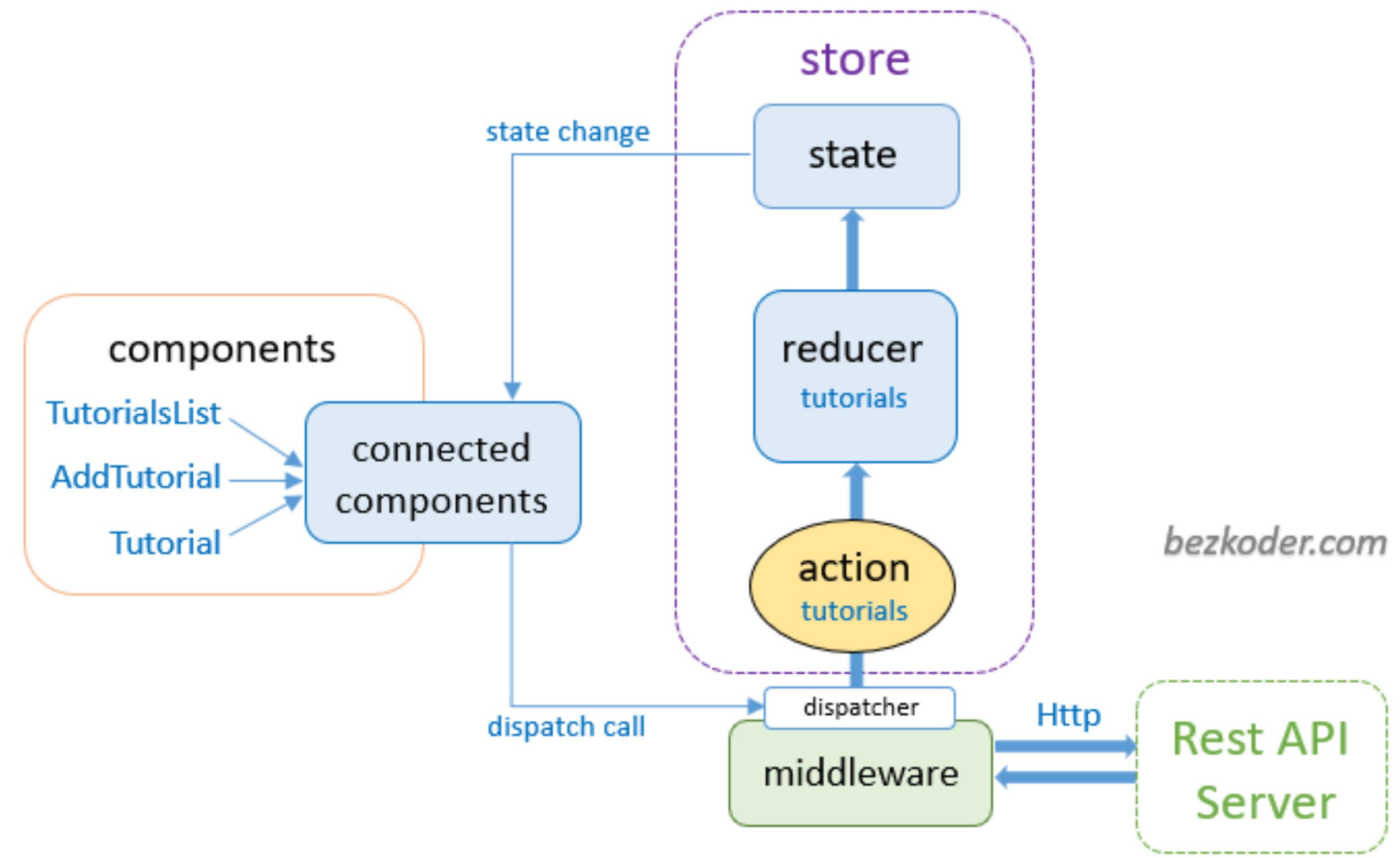
# Khoá học 6: Update một số công nghệ react

1/ Hướng dẫn sử dụng redux toolkit

2/ Typescript basic with react

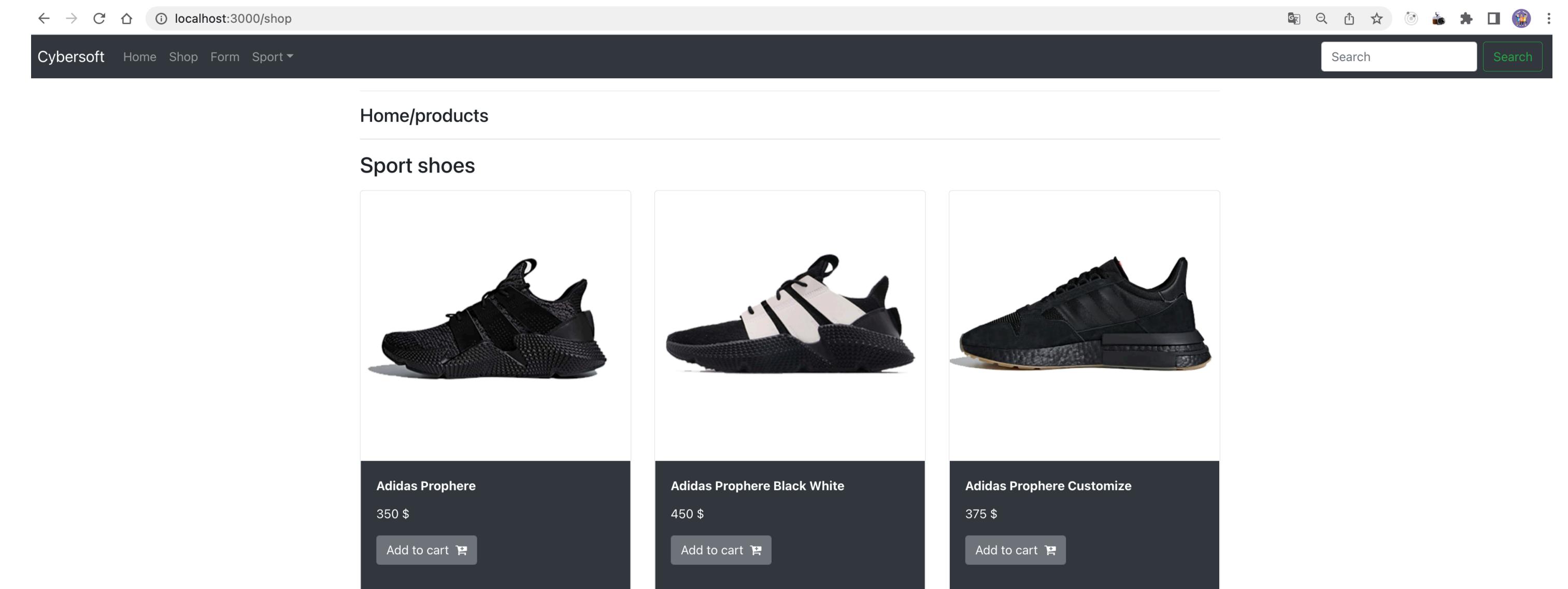
3/Hướng dẫn xây dựng dự án shoes shop tổng hợp

# REDUX TOOLKIT



# Hướng dẫn setup và xây dựng ứng dụng giỏ hàng thông qua api:

## <https://shop.cyberlearn.vn/swagger/index.html>



# REDUX TOOLKIT

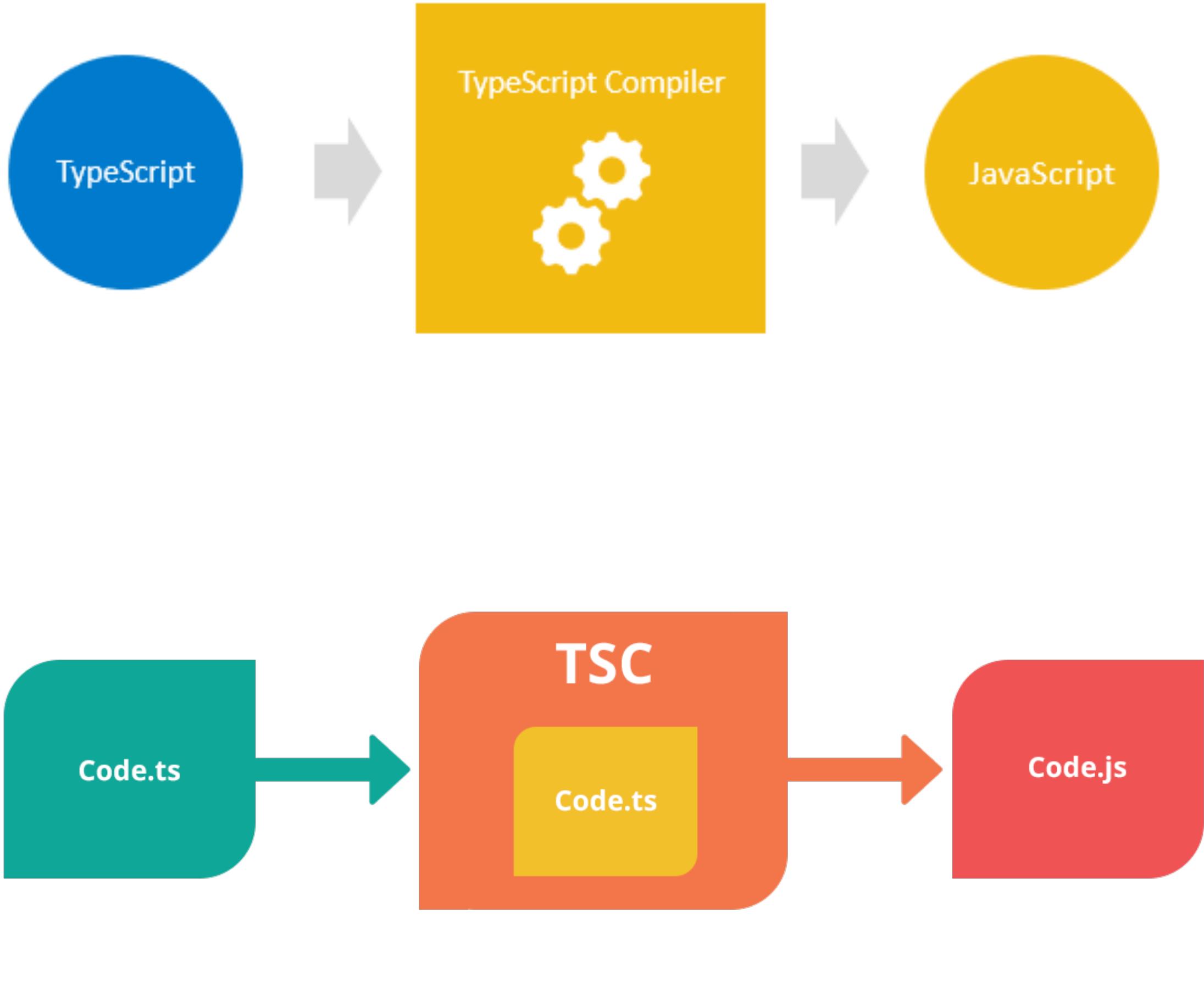
# Giới thiệu về typescript

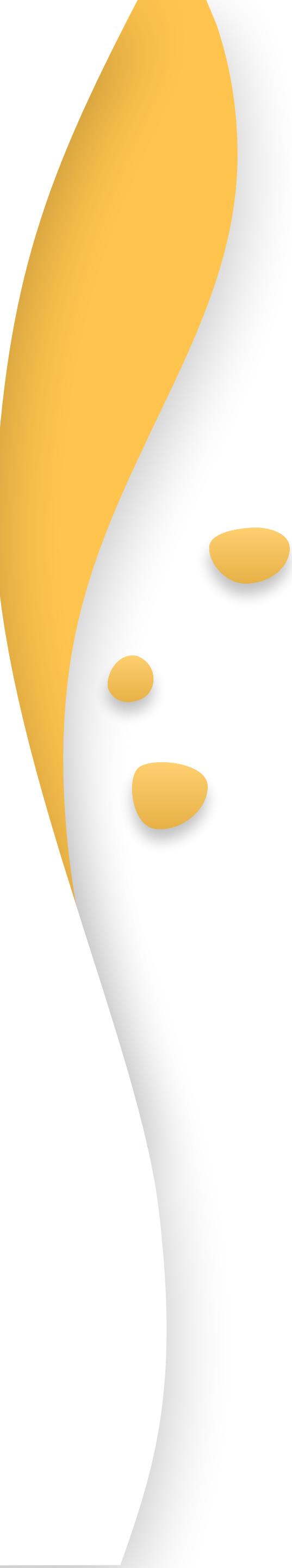
- **TypeScript là một dự án mã nguồn mở được phát triển bởi Microsoft**
- **TypeScript là một phiên bản nâng cao của javascript**
- **TypeScript bổ sung tùy chọn kiểu tĩnh và lớp hướng đối tượng mà điều này không có ở Javascript.**
- **TypeScript có thể sử dụng để phát triển các ứng dụng chạy ở clientside (Angular) và server-side (NodeJS).**
- **TypeScript sử dụng tất cả các tính năng của ECMAScript 2015 (ES6) như classes, modules.**

Types

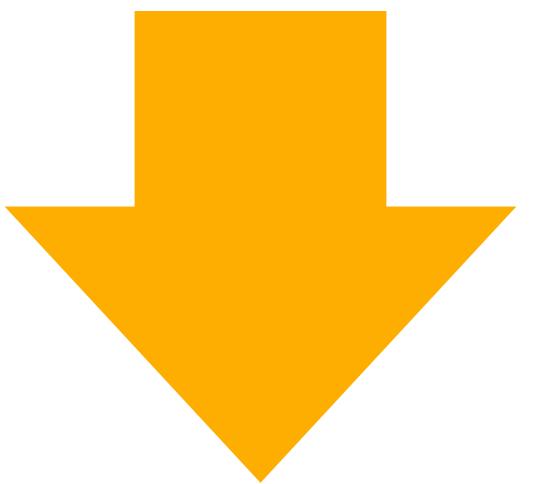
# Tại sao nên sử dụng TypeScript?

- Dễ phát triển dự án lớn: Với việc sử dụng các kỹ thuật mới nhất và lập trình hướng đối tượng nên TypeScript giúp chúng ta phát triển các dự án lớn một cách dễ dàng.
- Hỗ trợ các tính năng của Javascript phiên bản mới nhất: TypeScript luôn đảm bảo việc sử dụng đầy đủ các kỹ thuật mới nhất của Javascript, ví dụ như version hiện tại là ECMAScript 2015 (ES6).
- Là mã nguồn mở: TypeScript là một mã nguồn mở nên hoàn toàn có thể sử dụng miễn phí, bên cạnh đó còn được cộng đồng hỗ trợ lớn.
- Bản chất của Typescript là javascript: TypeScript sau khi biên dịch nó tạo ra các đoạn mã javascript vì vậy có thể chạy ở bất kỳ đâu mà javascript được hỗ trợ





# Type script



# Javascript

Một số tính năng mới: (Classes,  
Types, Interfaces ...)

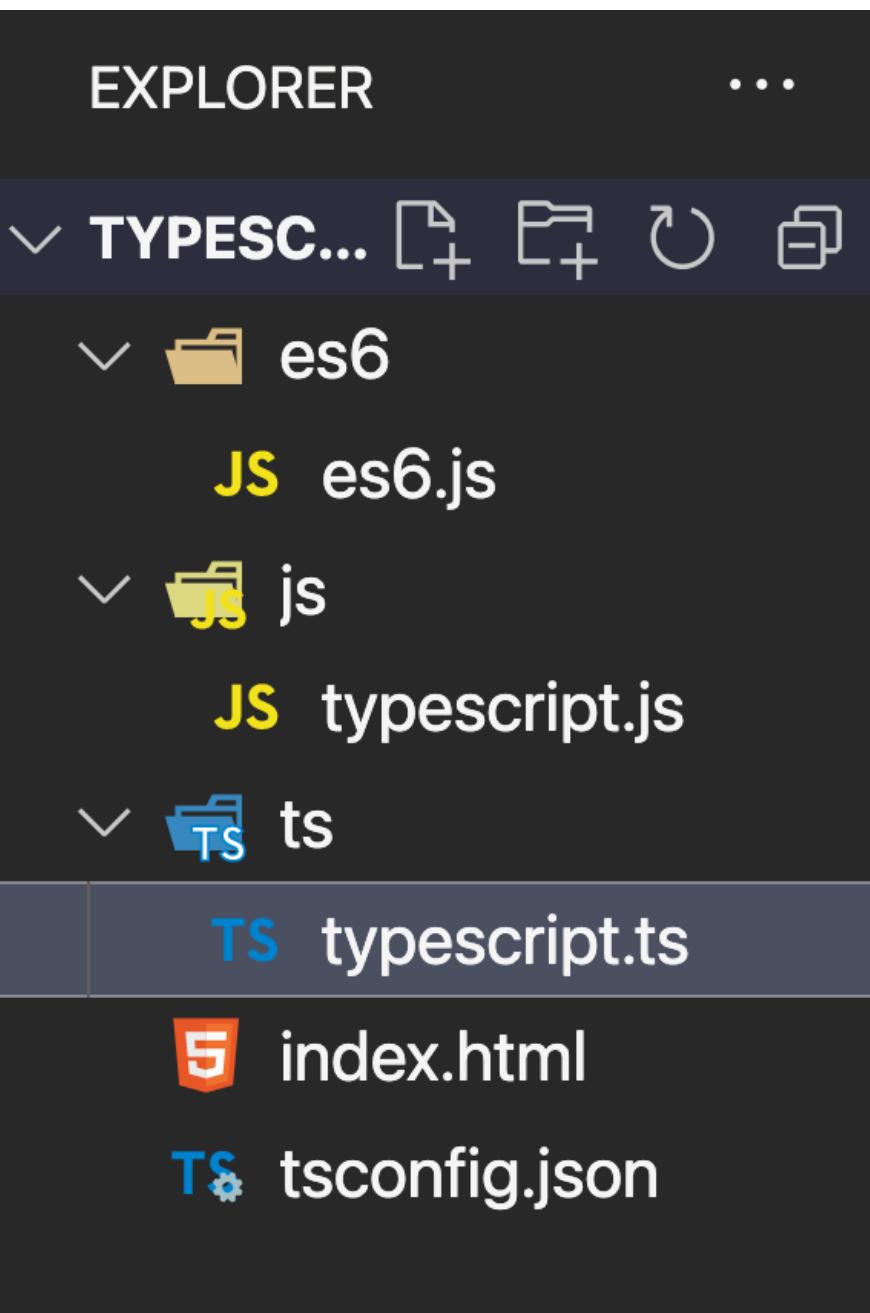
# Cài đặt typescript

**npm i typescript -g**

**Cấu hình:**

**tsc --init**

**Cấu trúc thư mục**



**tsconfig.json**

```
{  
  "compilerOptions": {  
    "target": "ES6",  
    "outDir": "./js",  
    "module": "CommonJS"  
  },
```

# Các tính năng của typescript hầu như đầy đủ so với es6 ngoài ra có 1 số điểm nổi bật

1. Cơ chế khai báo biến
2. Các kiểu dữ liệu (type)
3. Truyền tham số (Rest param, default params)
4. Xử lý đối với mảng, object (destructuring, spread operator)
5. String template
6. Duyệt Object, Array (For in, for of)
7. Object literal, dynamic key
8. Class object (access modifier)
9. Interfaces, type
10. Optional properties, optional chaining
11. implement interface
12. generic
13. union type - utility type

## 2. Type trong typescript (primitive value)

- TypeScript hỗ trợ type khi khai báo biến đối với các kiểu dữ liệu cơ sở (primitive value)

```
//Các kiểu dữ liệu primitive value (Kiểu cơ sở)
```

```
const title:string = 'cybersoft';
```

```
const price:number = 1000;
```

```
const valid:boolean = true;
```

```
const udf:undefined = undefined;
```

```
const ob:null = null;
```

```
//Các kiểu dữ liệu primitive value (Kiểu cơ sở)
```

```
const title = 'cybersoft';
```

```
const price = 1000;
```

```
const valid = true;
```

```
const udf = undefined;
```

```
const ob = null;
```

Typescript

Javascript (ES6)

## 2. Type trong typescript ( Reference value )

- Đối với mảng trong javascript thì các phần tử trong mảng có thể chứa được nhiều giá trị khác nhau về kiểu dữ liệu. Tương tự object trong javascript (ES6) không tường minh về các kiểu dữ liệu của thuộc tính —> đối với typescript ràng buộc tốt hơn về điều này với **interface** hoặc **type**.

```
//Reference value trong typescript  
  
//Array trong typescript  
  
const arrNumber:number[] = [1,2,3,4,5];  
  
const arrString:string [] = ['A','B','C','D','E'];
```

```
//Object trong typescript phải khai báo kiểu dữ liệu (interface hoặc type hoặc class)  
  
interface Product {  
    id:number,  
    name:string,  
    price:number
```

```
//Reference value trong typescript  
  
//Array trong typescript  
  
const arrNumber = [1,2,3,4,5];  
  
const arrString = ['A','B','C','D','E'];
```

```
//Object trong typescript  
  
const prod = {id:1,name:'Iphone',price:1000};
```

**TypeScript**

**Javascript (ES6)**

## 2. Type trong typescript ( Một số kiểu dữ liệu đặc biệt trong typescript)

- Any type đại diện cho bất kì value với kiểu dữ liệu nào : number, array, object, array ... Khi sử dụng any type chúng ta có thể thực hiện bất kỳ operation nào. Typescript sẽ bỏ qua việc kiểm tra type với any type value.
- Unknown cũng giống như any tuy nhiên khi thực hiện operation thì phải check kiểu dữ liệu.

```
//any dùng để chứa giá trị với bất kì kiểu dữ liệu nào  
let anyValue: any;  
  
anyValue = Math.random;  
  
anyValue = 1;  
  
anyValue = {};  
  
anyValue = [];  
  
anyValue = true;  
  
anyValue = "string";  
  
anyValue = null;  
  
anyValue = undefined;  
  
anyValue = Symbol("type");  
  
//Tương tự any unknown cũng có thể chứa bất kì giá trị nào. Tuy nhiên khi thực operation  
thì đối với unknown phải sử dụng type-checking (narrow type) thì mới cho thực hiện  
operation
```

```
// Any khá giống với các khai báo biến thuần của javascript  
let anyValue;  
  
anyValue = Math.random;  
  
anyValue = 1;  
  
anyValue = {};  
  
anyValue = [];  
  
anyValue = true;  
  
anyValue = "string";  
  
anyValue = null;  
  
anyValue = undefined;  
  
anyValue = Symbol("type");  
  
  
let numberAny = 1;  
console.log(numberAny + 1);
```

TypeScript

Javascript (ES6)

## 2. Type trong typescript ( Một số kiểu dữ liệu đặc biệt trong typescript)

- Thêm 1 ví dụ về type any và unkown . Tuy nhiên 2 kiểu dữ liệu này nên hạn chế sử dụng.

```
class SinhVien {  
    maSinhVien:string = "";  
    tenSinhVien:string = "";  
    constructor() {  
  
    }  
    hienThiThongTin () {  
        console.log('mã sinh viên: ', this.maSinhVien)  
    }  
}  
  
let sv = new SinhVien();  
sv.maSinhVien = '1';  
sv.tenSinhVien = 'Khải'  
//Tương tự any unknown cũng có thể chứa bất kì giá trị nào. Tuy nhiên khi thực operation thì đối với unknown phải sử dụng type-checking (narrow type) thì mới cho thực hiện  
operation  
let svUnknown:unknown = sv;  
if( svUnknown instanceof SinhVien) {  
    svUnknown.hienThiThongTin();
```

## 2. Type trong typescript ( Đối với callback function)

- Như các bạn biết callback function là function đóng vai trò là tham số truyền vào 1 function khác và được function đó gọi lại khi thực thi. Ở javascript kiểu dữ liệu của callback không được mô tả cụ thể dẫn đến khó hình dung → Tuy nhiên đối với typescript điều đó đã trở nên dễ dàng hơn !!! Ngoài ra typescript có thể nhận biết được tự động callback (đúng số lượng tham số) = **contextual typing**.

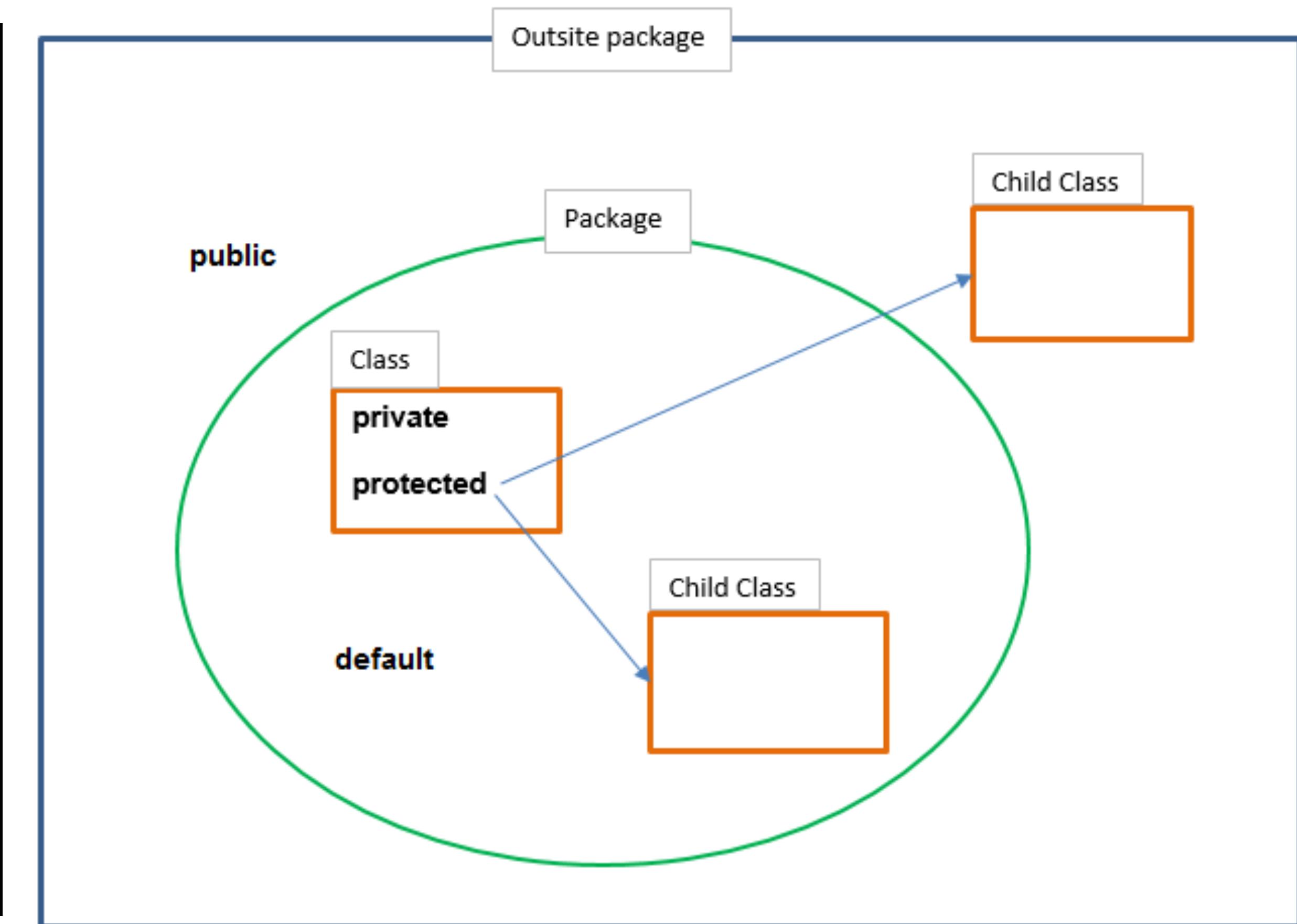
```
function main(callback: (title:string) => void) {  
    alert('Hello wellcome to cybersoft typescript !!!')  
    callback('Hello Khải master yi !!!');  
}  
  
//Callback  
function renderH1(title:string) {  
    document.querySelector('body').innerHTML += `<h1 class="bg-dark text-white display-4 p-3">${title}</h1>`;  
}  
//Callback  
function renderSection(title:string) {  
    document.querySelector('body').innerHTML += `<section class="bg-warning p-5 text-danger text-cetern">${title}</section>`;  
}  
//Gọi hàm main và truyền vào 1 trong 2 callback và kiểm tra kết quả !!!  
main(renderH1);
```

# TypeScript

## 8. Class object

- Class trong typescript sẽ có hỗ trợ access modifier (Phạm vi truy cập của thuộc tính or phương thức) với các từ khóa gần giống như các ngôn ngữ lập trình hướng đối tượng khác là public, private, protected

```
class Foo {  
    private x: number;  
    protected y: number;  
    public z: number;  
    saveData(foo: Foo): void {  
        this.x = 1; // ok  
        this.y = 1; // ok  
        this.z = 1; // ok  
  
        foo.x = 1; // ok  
        foo.y = 1; // ok  
        foo.z = 1; // ok  
    }  
}  
  
class Bar extends Foo {  
    getData(foo: Foo, bar: Bar) {  
        this.y = 1; // ok  
        this.z = 1; // ok  
        foo.z = 1; // ok  
  
        bar.y = 1; // ok  
        bar.z = 1; // ok  
        foo.z = 1; // ok  
  
        foo.x = 1; // Error, x only accessible within A  
        bar.x = 1; // Error, x only accessible within A  
        bar.y = 1; // Error, y only accessible through instance of B  
    }  
}
```



# TypeScript

## 9. Type

Từ khóa **type** trong **TypeScript** là một cách cung cấp **type aliases** (các tên thuộc tính) cho các variables (biến). Type có 3 cách sử dụng phổ biến là **Primitive types**, **Tuple** và **Union**

```
//Primitive type : Mình có thể tạo ra 1 alias type cho biến primitive dựa vào type
type name = string;
let hoTen:name = 'Hè lố mình đã tự tạo ra được kiểu dữ liệu rồi nè !!!';
```

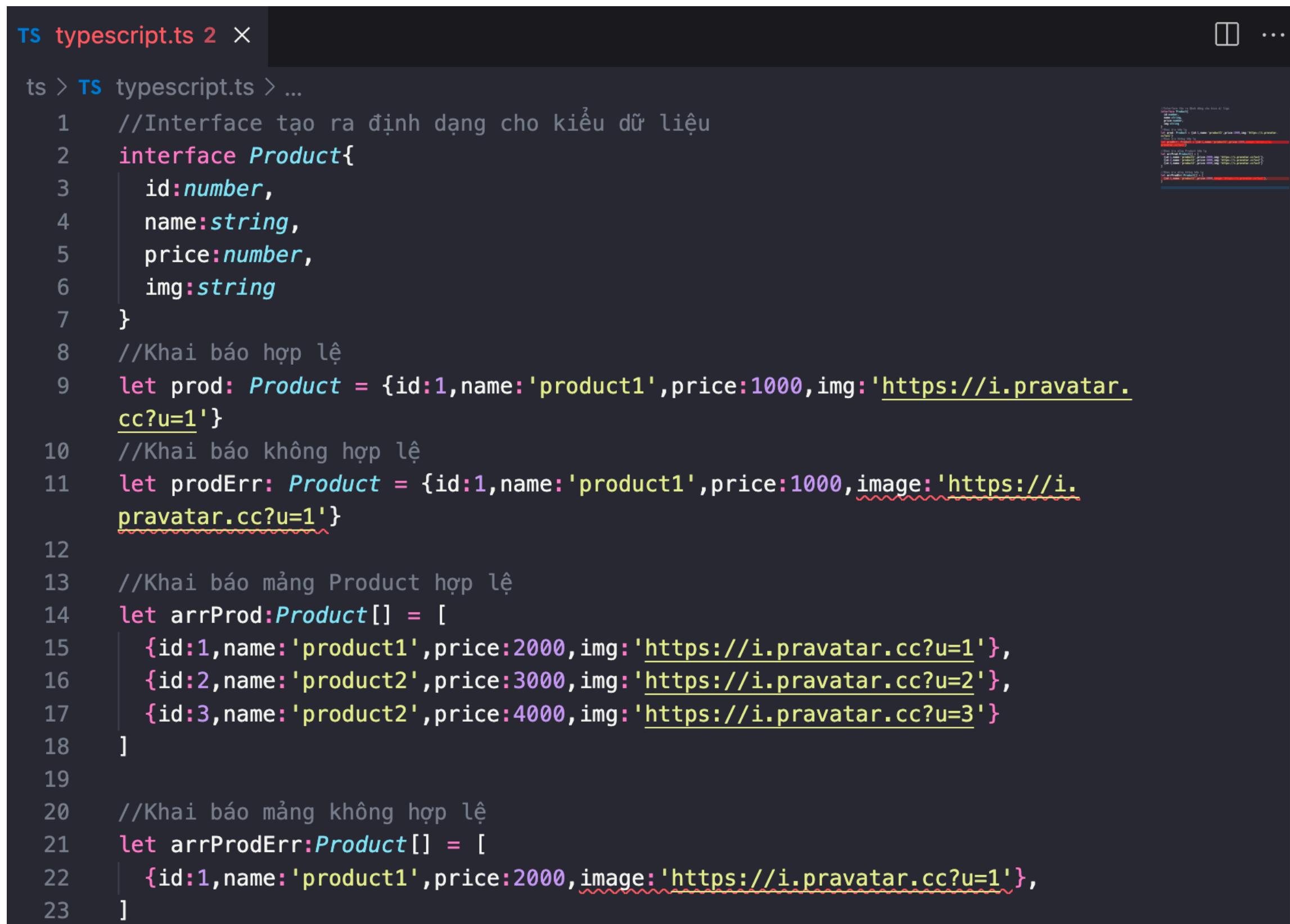
```
//Tuple type thường dùng cho các kiểu object được thể hiện dưới dạng mảng (hoặc các hook trong react)
type SinhVien = [number, string, () => void];
let [id, fName, showInfo]:SinhVien = [1, 'Nguyễn Văn A',
() => { console.log('Hello cybersoft')}]
];
showInfo();

//Ví dụ giống hook
type typeState = [number, (number:number) => void];
let [number, setNumber]:typeState = [1,
(newNumber:number) => {
    number = newNumber ;
}]
console.log(number);// 1
setNumber(number + 1);
console.log(number); // 2
```

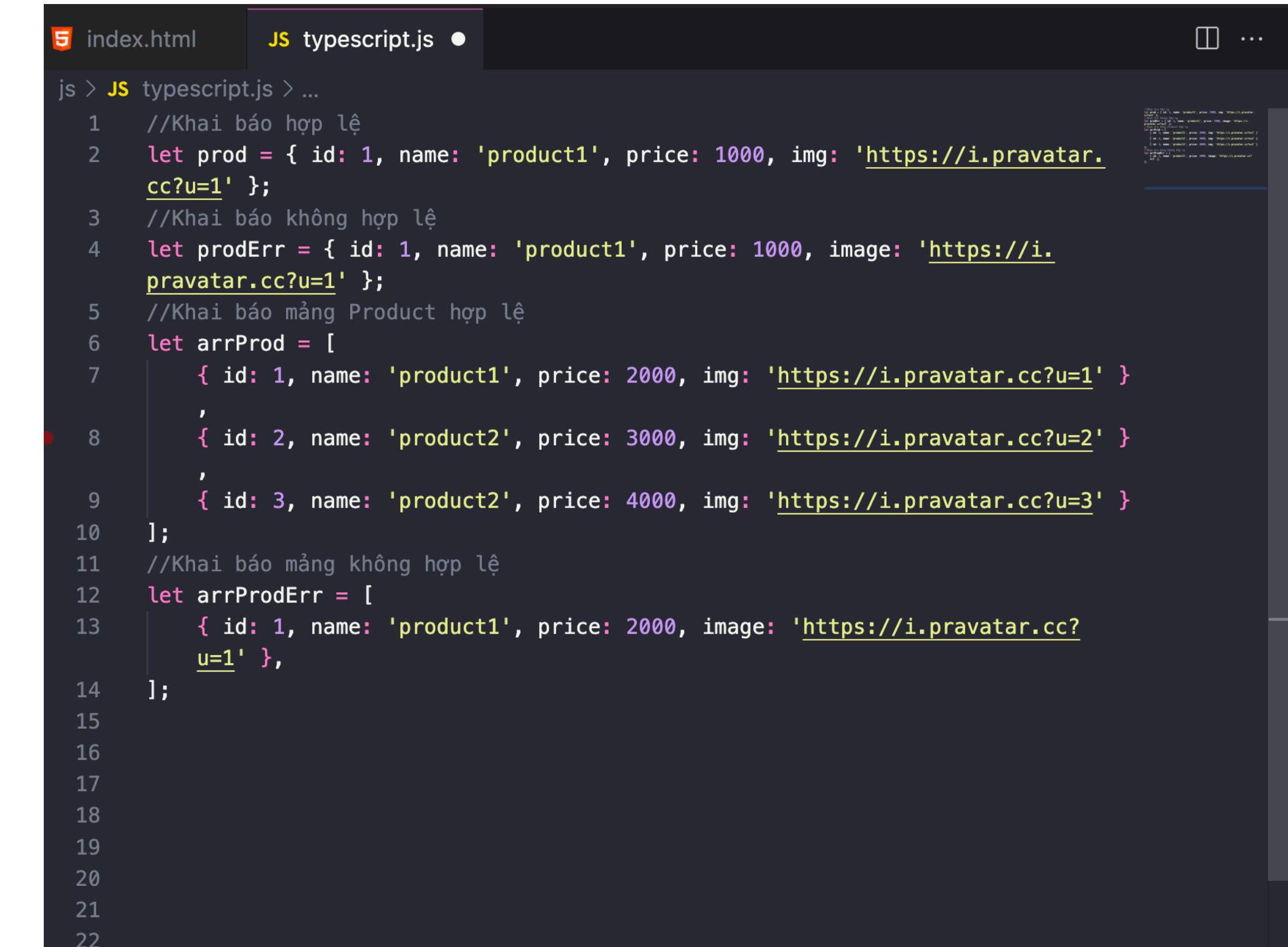
```
19   //union type
20   type key = number | string;
21   let index:key = 0;
22   let i :key = '0';
23   let ob = {
24     [index]: 1,
25     [i]:'1'
26   }
27
28   ob[index];
29   ob.i;
```

## 9. interface

Ngoài ra trong typescript còn hỗ trợ sử dụng interface để định dạng (format) dữ liệu. Interface tương tự như type tuy nhiên interface có thể merge (gộp lại được).



```
ts typescript.ts 2 ×
ts > TS typescript.ts > ...
1 //Interface tạo ra định dạng cho kiểu dữ liệu
2 interface Product{
3     id:number,
4     name:string,
5     price:number,
6     img:string
7 }
8 //Khai báo hợp lệ
9 let prod: Product = {id:1,name:'product1',price:1000,img:'https://i.pravatar.cc?u=1'}
10 //Khai báo không hợp lệ
11 let prodErr: Product = {id:1,name:'product1',price:1000,image:'https://i.pravatar.cc?u=1'}
12 //Khai báo mảng Product hợp lệ
13 let arrProd:Product[] = [
14     {id:1,name:'product1',price:2000,img:'https://i.pravatar.cc?u=1'},
15     {id:2,name:'product2',price:3000,img:'https://i.pravatar.cc?u=2'},
16     {id:3,name:'product2',price:4000,img:'https://i.pravatar.cc?u=3'}
17 ]
18
19 //Khai báo mảng không hợp lệ
20 let arrProdErr:Product[] = [
21     {id:1,name:'product1',price:2000,image:'https://i.pravatar.cc?u=1'},
22 ]
```



```
js index.html JS typescript.js ●
js > JS typescript.js > ...
1 //Khai báo hợp lệ
2 let prod = { id: 1, name: 'product1', price: 1000, img: 'https://i.pravatar.cc?u=1' };
3 //Khai báo không hợp lệ
4 let prodErr = { id: 1, name: 'product1', price: 1000, image: 'https://i.pravatar.cc?u=1' };
5 //Khai báo mảng Product hợp lệ
6 let arrProd = [
7     { id: 1, name: 'product1', price: 2000, img: 'https://i.pravatar.cc?u=1' },
8     { id: 2, name: 'product2', price: 3000, img: 'https://i.pravatar.cc?u=2' },
9     { id: 3, name: 'product2', price: 4000, img: 'https://i.pravatar.cc?u=3' }
10 ];
11 //Khai báo mảng không hợp lệ
12 let arrProdErr = [
13     { id: 1, name: 'product1', price: 2000, image: 'https://i.pravatar.cc?u=1' },
14 ];
```

## TypeScript

## Javascript

## 9. Optional properties - optional chaining

- **Optional properties** : Trong typescript **optional** properties cho phép ta định nghĩa các thuộc tính không bắt buộc đối với format của **interface** hoặc **type**.
- **Optional chaining** : Tương tự optional properties

```
ts typescript.ts < ...>
1 //Optional properties
2 interface IProduct {
3     id: number,
4     name:string,
5     price: number,
6     desc?:string
7 }
8
9 let prod:IProduct = {id:1,name:'product 1', price: 1000}
10
11 //Optional properties
12 type Product = {
13     id: number,
14     name:string,
15     price: number,
16     desc?:string
17 }
18
19 let prod1:Product = {id:1,name:'product 1', price: 1000}
```

```
ts typescript.ts < ...>
1 //Optional properties
2 class Product {
3     id: number;
4     name:string;
5     price: number;
6     info: Array<string>;
7     constructor(id,name,price){
8         this.id = id;
9         this.name = name;
10        this.price = price;
11    }
12 }
13
14 let prod = new Product(1,'product 1', 1000);
15
16 //Khi thuộc tính info không có giá trị (undefined) => không thực hiện được hàm map => báo lỗi
17 prod.info.map((item,index)=>{
18     console.log(item)
19 })
20
21 //Khi sử dụng optional chaining ở thuộc tính map => code vẫn chạy bình thường
22 prod.info.map((item,index)=>{
23     console.log(item)
24 });
25
26
27 console.log('continue!!!!');
```

# 10. Optional properties - optional chaining

- **Optional properties** : Trong typescript **optional** properties cho phép ta định nghĩa các thuộc tính không bắt buộc đối với format của **interface** hoặc **type**.
- **Optional chaining** : Tương tự optional properties

```
TS typescript.ts ×

ts > TS typescript.ts > ...
1 //Optional properties
2 interface IProduct {
3   id: number,
4   name:string,
5   price: number,
6   desc?:string
7 }
8
9 let prod:IProduct = {id:1,name:'product 1', price: 1000}
10
11 //Optional properties
12 type Product = {
13   id: number,
14   name:string,
15   price: number,
16   desc?:string
17 }
18
19 let prod1:Product = {id:1,name:'product 1', price: 1000}
```

```
TS typescript.ts × index.html

ts > TS typescript.ts > ...
1 //Optional properties
2 class Product {
3   id: number;
4   name:string;
5   price: number;
6   info: Array<string>;
7   constructor(id,name,price){
8     this.id = id;
9     this.name = name;
10    this.price = price;
11  }
12 }
13
14 let prod = new Product(1,'product 1', 1000);
15
16 //Khi thuộc tính info không có giá trị (undefined) => không thực hiện được hàm map => báo lỗi
17 prod.info.map((item,index)=>{
18   console.log(item)
19 })
20
21 //Khi sử dụng optional chaining ở thuộc tính map => code vẫn chạy bình thường
22 prod.info.map((item,index)=>{
23   console.log(item)
24 });
25
26
27 console.log('continue!!!!');
```

# 11. extends interface - implements interface

- **extends** cho phép ta có thể kế thừa lại từ 1 interface đã định nghĩa mà không cần viết lại (khá giống với merge những không cần đặt đúng tên interface)
- **implements** là chức năng cho phép 1 class hoặc 1 object định nghĩa cụ thể giá trị hoặc phương thức từ format của **interface**

```
TS typescript.ts ×

ts > TS typescript.ts > ...

1 enum FontWeight {bold = 'bold',normal = 'normal', italic='italic'}
2 interface IText {
3   color:string,
4   fontSize:number,
5   fontWeight:FontWeight
6 }
7
8 interface PaddingText extends IText {
9   padding:number
10}
11
12 let pText:PaddingText = {
13   color:'red',
14   fontSize:17,
15   fontWeight:FontWeight.bold,
16   padding: 15
17 }
```

```
23 interface IButton {
24   color: string,
25   fontSize: number,
26   fontWeight: string,
27   onClick: () => void,
28   onTouch: () => void,
29   onHover: () => void
30 }
31 //class implements interface
32 class GradientButton implements IButton {
33   color: string;
34   fontSize: number;
35   fontWeight: string;
36   onClick = (): void => {
37     console.log('onClick');
38   }
39   onTouch = (): void => {
40     console.log('onTouch');
41   }
42   onHover = (): void => {
43     console.log('onHover');
44   }
45 }
46 //object implements interface
47 const button: GradientButton = {
48   color: 'red',
49   fontSize: 17,
50   fontWeight: 'bold',
51   onClick: () => {
52     console.log('onTouch');
53   },
54   onTouch: (): void => {
55     console.log('onTouch');
56   },
57   onHover: (): void => {
58     console.log('onHover');
59   }
60 }
```

# 12. Generic

- **Generic** cho phép ta truyền động các type khi tạo ra interface hoặc class động

```
//1 Ví dụ của generic:  
let arrNumber = new Array<Number>(); // [1,2,3]  
let arrString = new Array<String>(); // ['A','B','C']
```

- **Định nghĩa Generic cho interface**

```
1  interface Product {  
2    id:number;  
3    name: string,  
4    price:number  
5  }  
6  interface UserLogin{  
7    id:number,  
8    name:string,  
9    phone:string  
10 }  
11  
12 //Định nghĩa generic với interface  
13 interface IList<T> {  
14   length:number;  
15   [index:number] : T  
16 }  
17  
18 const lstUser: IList<UserLogin> = [  
19   {id:1,name:'iphone',phone:'0909090909'},  
20   {id:2,name:'iphoneX',phone:'0808080808'},  
21   {id:3,name:'iphoneXS',phone:'0707070707'}  
22 ]  
23  
24 const lstProd: IList<Product> = [  
25   {id:1,name:'iphone',price:1000},  
26   {id:2,name:'iphoneX',price:2000},  
27   {id:3,name:'iphoneXS',price:3000}  
28 ]  
29
```

- **Định nghĩa Generic cho class**

```
32 //Định nghĩa generic cho class //T là đại diện cho 1 format object Type hoặc Interface (Muốn đặt tên gì cũng được)  
33 class List<T> {  
34   data: T[] = [];  
35   constructor() {  
36     }  
37   push(item:T){  
38     this.data.push(item);  
39   }  
40   del (index:number) {  
41     this.data.splice(index,1);  
42   }  
43   find(callback:(item:T,index:number) => T){  
44     return this.data.find(callback);  
45   }  
46 }  
47  
48  
49 //Khởi tạo 1 đối tượng object có kiểu dữ liệu T (Product)  
50 const prod:Product = {id:1,name:'phone',price:1000};  
51 //Khởi tạo Generic List với Type là product (T: Product)  
52 const lstProduct = new List<Product>();  
53 lstProduct.push(prod)  
54 console.log(lstProduct.data[0]);
```

## 13. union type - utility type

- **Union type:** Trong 1 số trường hợp giá trị trả về của chúng ta có thể có nhiều trường hợp chuỗi hoặc số hoặc object thậm chí là undefined vì vậy typescript cung cấp cho chúng ta cách khai báo union type để handle cho nhiều trường hợp.

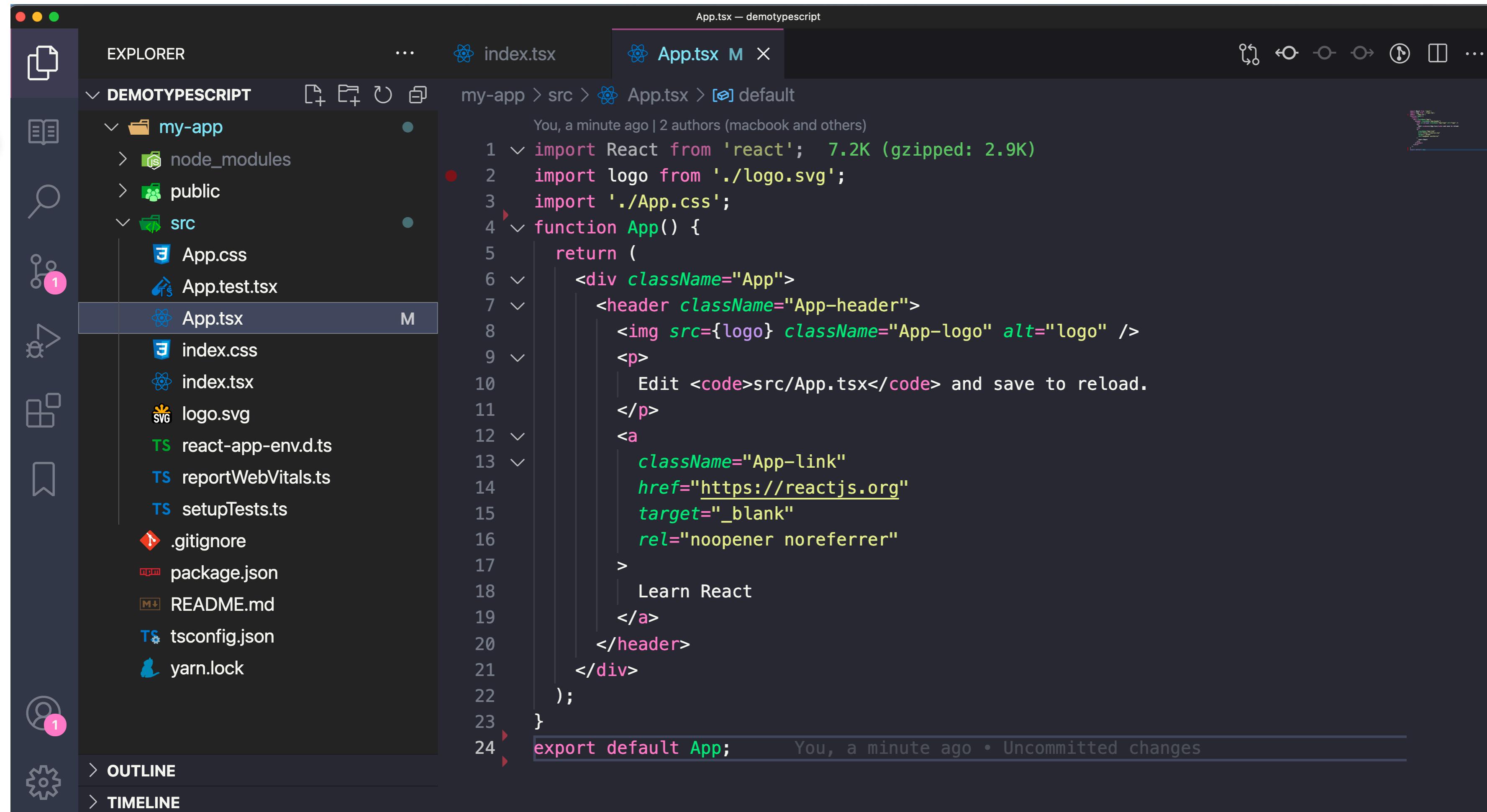
Ví dụ

```
interface Person {  
    id: string;  
    phone: string;  
    email: string;  
    gender: 'male' | 'female';  
}  
  
let result : number | string | undefined = 'ok';
```

```
enum Gender { male = 'male', female = 'female' }  
// interface  
interface IPerson {  
    id: string | number; // interface  
    phone: string;  
    email: string;  
    gender: Gender  
}
```

# Khởi tạo dự án reactjs với typescript với cú pháp

npx create-react-app my-app --template typescript



The screenshot shows a dark-themed instance of Visual Studio Code (VS Code) with the following details:

- File Explorer:** On the left, it shows a project structure named "DEMOTYPESCRIPT". Inside "my-app", there are "node\_modules", "public", and "src" folders. "src" contains "App.css", "App.test.tsx", and "App.tsx" (which is currently selected). Other files in "src" include "index.css", "index.tsx", "logo.svg", "react-app-env.d.ts", "reportWebVitals.ts", "setupTests.ts", ".gitignore", "package.json", "README.md", "tsconfig.json", and "yarn.lock".
- Code Editor:** The main editor area displays the content of "App.tsx". The code is as follows:

```
App.tsx – demotypescript
my-app > src > App.tsx > [?] default
You, a minute ago | 2 authors (macbook and others)
1 import React from 'react'; 7.2K (gzipped: 2.9K)
2 import logo from './logo.svg';
3 import './App.css';
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-logo" alt="logo" />
9         <p>
10           Edit <code>src/App.tsx</code> and save to reload.
11         </p>
12       <a
13         className="App-link"
14         href="https://reactjs.org"
15         target="_blank"
16         rel="noopener noreferrer"
17       >
18         Learn React
19       </a>
20     </header>
21   </div>
22 }
23 export default App; You, a minute ago • Uncommitted changes
```

The code uses standard React syntax with some TypeScript annotations (like `import` statements and the `function` keyword). A note at the bottom of the code editor says "Edit <code>src/App.tsx</code> and save to reload."

# Các điểm khác biệt khi sử dụng typescript và javascript

- **Khai báo state**

```
my-app > src > components > DemoState.tsx > [S] State
  1 import React, { Component } from 'react'  7.2K (gzipped: 2.9K)
  2
  3 type Props = {}
  4 type State = {
  5   number: number
  6 }
  7 export default class DemoState extends Component<Props, State> {
  8   state: State = {
  9     number: 1
 10   }
 11   increase = () => {
 12     this.setState({
 13       number: this.state.number + 1
 14     })
 15   }
 16   decrease = () => {
 17     this.setState({
 18       number: this.state.number - 1
 19     })
 20   }
 21   render() {
 22     return (
 23       <div className='container'>
 24         <h3>{this.state.number}</h3>
 25         <button className='btn btn-success' onClick={() => {
 26           this.increase();
 27         }}>+</button>
 28         <button className='btn btn-success ml-2' onClick={() => {
 29           this.decrease();
 30         }}>-</button>
 31       </div>
 32     )
 33   }
 34 }
```

# Các điểm khác biệt khi sử dụng typescript và javascript

- **Khai báo hook useState**

```
● ● ●

1 import {React, {useState} } from 'react'
2 export default function DemoState() {
3   const [number, setNumber] = useState<number>(1);
4   return (
5     <div className='container'>
6       <h3>{number}</h3>
7       <button className='btn btn-primary' onClick={() => {
8         setNumber(number + 1);
9       }}>+</button>
10      <button className='btn btn-primary ml-2' onClick={() => {
11        setNumber(number - 1);
12      }}>-</button>
13    </div>
14  )
15 }
```

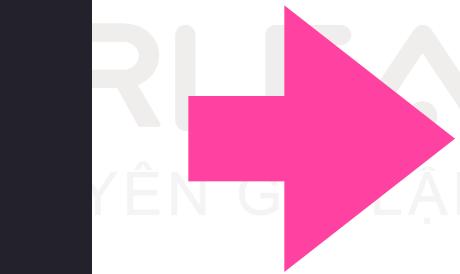
carbon  
carbon.now.sh

# Các điểm khác biệt khi sử dụng typescript và javascript

- Khai báo Props class component

```
●●●  
1 import './App.css';  
2 import Product from './components/Product';  
3  
4 function App() {  
5   const prod: ProductModel = {  
6     id: '1',  
7     name: 'iPhoneX',  
8     price: 1000,  
9     image: 'https://picsum.photos/id/1/200'  
10    };  
11    return (  
12      <div className="App">  
13        <Product product={prod}>  
14      </div>  
15    );  
16  }  
17  export default App;  
18
```

carbon  
carbon.now.sh



```
●●●  
1 import React, { Component } from 'react';  
2 interface ProductProps {  
3   product: ProductModel;  
4 }  
5 export default class Product extends Component<ProductProps> {  
6   render() {  
7     let { product } = this.props;  
8     return (  
9       <div className='card'>  
10         <img src={product.image} className="w-100" alt='...'/>  
11         <div className='card-body'>  
12           <p>{product.name}</p>  
13           <p>{product.price}</p>  
14         </div>  
15       </div>  
16     );  
17   }  
18 }
```

carbon  
carbon.now.sh

# Các điểm khác biệt khi sử dụng typescript và javascript

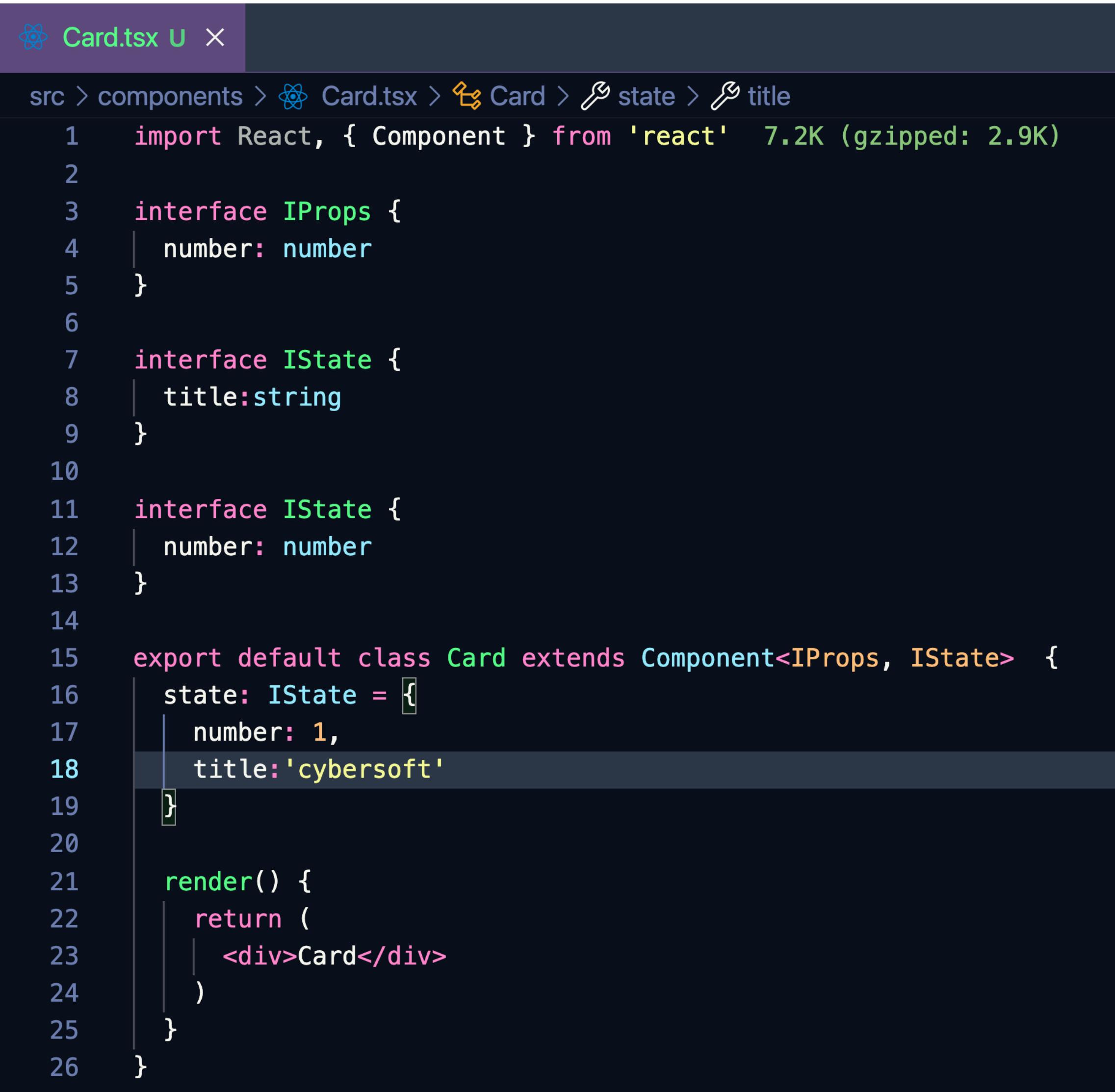
- **Khai báo props với react function component**

```
1 import React from 'react'  
2 type Props = {  
3   product: ProductModel,  
4 }  
5 export default function ProductFunc({product}: Props) {  
6   return (  
7     <div className="card">  
8       <img src={product.image} className="w-100" alt="..."/>  
9       <div className="card-body">  
10         <p>{product.name}</p>  
11         <p>{product.price}</p>  
12       </div>  
13     </div>  
14   )  
15 }
```

carbon  
carbon.now.sh

# Khai báo redux với typescript

- Khai báo với interface có thể merge



The screenshot shows a code editor window titled "Card.tsx" with a purple header bar. The file path is "src > components > Card.tsx". The code itself is as follows:

```
src > components > Card.tsx > Card > state > title
1 import React, { Component } from 'react' 7.2K (gzipped: 2.9K)
2
3 interface IProps {
4   number: number
5 }
6
7 interface IState {
8   title: string
9 }
10
11 interface IState {
12   number: number
13 }
14
15 export default class Card extends Component<IProps, IState> {
16   state: IState = {
17     number: 1,
18     title: 'cybersoft'
19   }
20
21   render() {
22     return (
23       <div>Card</div>
24     )
25   }
26 }
```

# Hướng dẫn xây dựng dự án với typescript

