**NODE**

1. Cài thư viện**: npm install tên\_thư\_viện\_muốn\_cài**
2. Các thư viện được cài trong thư mục **node\_modules** và được khai báo trong file **package.json**
3. Tạo server
4. Phương thức GET
   1. res.send: Trả về một text
   2. res.sendFile: Trả về text trong một file
   3. fs.readFile: Đọc dữ liệu trong file và trả về bằng lệnh res.end()
5. Phương thức POST
   1. Sử dụng thư viện **body-parser** để nhận các thông tin khi gửi lên
   2. Thêm đoạn lệnh

**app.use((req, res, next) => {**

**res.header('Access-Control-Allow-Origin', '\*');**

**res.header('Access-Control-Allow-Headers', 'Content-Type,X-Requested-With');**

**next();**

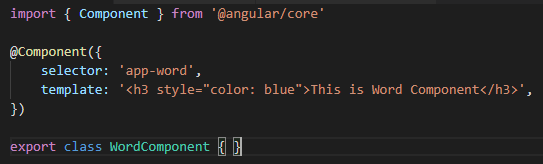
**});**

Để cho phép truy cập server từ một địa chỉ localhost khác

1. Kết nối Database:
   1. Tạo Database
   2. Tạo Bảng trong DB
   3. Insert dữ liệu vào trong bảng
   4. Select dữ liệu trong bảng (chỉ lấy 1 dòng đầu tiên)
   5. Select dữ liệu trong bảng (lấy tất cả các dòng)
2. Tạo DB từ file excel
3. Tạo file pdf và pdf sample sử dụng thư viện được tạo sẵn
4. Dùng dữ liệu dạng json để ghi ra file excel
5. Nhân bản một file excel
6. Tạo server nodeJS
   1. Sử dụng phương thức get để lấy dữ liệu từ server
   2. Sử dụng phương thức post để post dữ liệu lên server và nhận dữ liệu từ server trả về
7. Đưa ứng dụng lên heroku
   1. Chạy lệnh **ionic** **cordova build browser --prod –release** để dịch toàn bộ thư mục nguồn (src) thành thư mục www
   2. Push thư mục www và toàn bộ những thư mục liên quan đến server lên heroku và chạy

**ANGULAR**

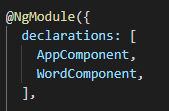
1. **Những tính năng nổi bật của Angular:**
   * + Tính năng tự động reload browser khi code thay đổi
     + Giúp code có tính chặt chẽ cao hơn và ngắn gọn hơn
     + Phiên bản Angular 4 là phiên bản mới, được nâng cấp từ phiên bản Angular 2
     + Tài liệu tham khảo: <https://angular.io/>
2. **Cài đặt môi trường lập trình**
   * + Cài nodeJS: truy cập đường dẫn sau để download NodeJS phiên bản phù hợp với máy <https://nodejs.org/en/>
     + Cài đặt Code editor: VS Code: <https://code.visualstudio.com/download>
3. **Cài đặt Angular CLI**
   * + Mở Terminal và gõ lệnh : **npm install -g @angular/cli**
     + Kiểm tra xem đã cài đặt thành công chưa: **ng -v**
4. **Tạo không gian làm việc và ứng dụng đầu tiên**
   * + Tạo project mới**: ng new ProjectName**
     + Chạy project lên: **ng serve --open**
     + Mở trình duyệt với địa chỉ: <http://localhost:4200/> để xem kết quả
5. **Các thành phần của Angular**
   * + ./src/app/app.component.ts.
     + ./src/app/app.component.css
     + ./src/app/app.component.html
     + ./src/app/app.module.ts
     + ./src/index.html
6. **Khái niệm component**
   1. Component là gì?
      * Component là các thành phần được hiển thị trên màn hình
      * Điều chỉnh một phần của màn hình (view)
      * Component có thể chứa nhiều thẻ HTML
      * Component có thể chứa Component khác
      * Component có thể nhận tham số
      * Component có thể rất linh hoạt
   2. Lợi ích của việc sử dụng Component
      * Dễ quản lý code
      * Tái sử dụng code
7. **Tạo component đơn giản**
   * + Việc viết ra một component là tạo ra một HTML tag, để có thể sử dụng trong các component khác
     + Quy tắc đặt tên của component là **tên\_component.component.ts**
     + Ví dụ:
       1. Trong thư mục **./src/app**/ tạo ra file **word.component.ts**
       2. Trong file **word.component.ts**, import các thư viện cần thiết vào và tạo ra các thành phần cần thiết cho một component như sau:



* + - 1. Import component này vào file **app.module.ts**



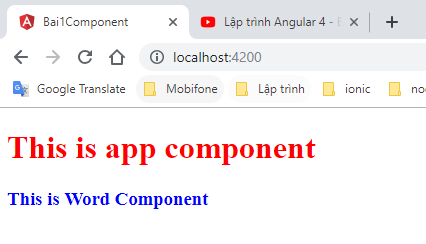
* + - 1. Khai báo component này trong mảng **declarations** của file **app.module.ts,** như sau:



* + - 1. Sử dụng component vừa tạo trong file **app.component.html**, như sau:



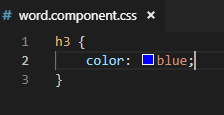
* + - 1. Truy cập trình duyệt với port 4200, ta được giao diện như sau:



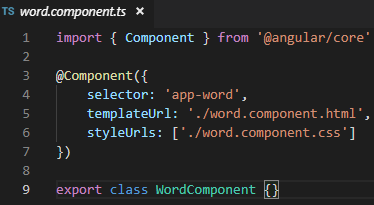
1. **Component decorators**
   * + Có thể tách riêng các template, style ra thành các file riêng sau đó link vào trong file ts để sử dụng, như vậy code sẽ gọn hơn
       - 1. Tạo các file **word.component.css**, **word.component.html** để định nghĩa các thành phần này trong các file riêng biệt
     + File **word.component.html**:



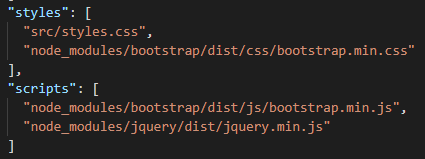
* + - File **word.component.css**



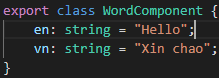
* + - * 1. Khai báo các file này vào file **word.component.ts** để sử dụng



1. **Nhúng bootstrap và jquery vào project**
   * + Cài bootstrap và jquery: **npm i bootstrap jquery –save**
     + Thêm đường dẫn **bootstrap.min.css** vào file **angular.json**
     + Thêm đường dẫn **bootstrap.min.js** và **jquery.min.js** vào file **angular.json**



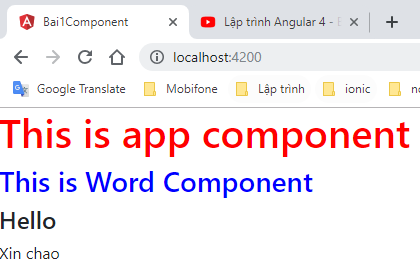
1. **String interpolation**
   * + Truyền giá trị từ class sang file HTML để hiển thị ra trình duyệt
     + Trong file **word.component.ts**, tạo các biến **en** và **vn** như sau:



* + - Trong file **word.component.html**, dùng string interpolation để hiển thị giá trị các biến được định nghĩa ở file **word.component.ts**, như sau:

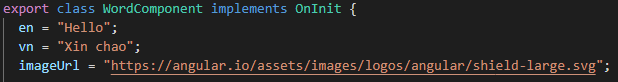


* + - Kết quả hiển thị ở trình duyệt



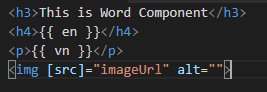
**Nhận xét:** Nếu để các file chung ngoài thư mục **.src/app/** thì sẽ có rất nhiều file lộn xộn, để cho cấu trúc project gọn gàng và dễ quản lý, ta tạo thư mục **word** rồi di chuyển các file của component word vào trong thư mục này, sau đó sửa đường dẫn import trong **app.module.ts** cho đúng.

1. **Tạo component sử dụng Angular CLI**
   * + Dùng lệnh Angular CLI để tạo các component: **ng g c tên\_component**
     + Một folder mới được tạo ra chứa 4 file của component mới là **css**, **html**, **spec.ts** và **ts**
     + File **.spec.ts** là file dùng để test component
     + Ngoài ra thì component vừa tạo này cũng tự động import vào file **app.module.ts**
     + Lúc mới tạo component thì Angular cũng tự động implement một interface OnInit vào, vì vậy trong class phải có hàm **ngOnInit(){},** hàm này sẽ tự động chạy khi gọi component, khi component thay đổi trang thái, khi component khởi tạo, có thể dùng hàm này để lấy dữ liệu từ server, chi tiết xem thêm tại document
2. **Property binding**
   * + Tạo các biến trong class **WordComponent**, trong đó biến **imageUrl** là đường dẫn của một hình ảnh, như sau:

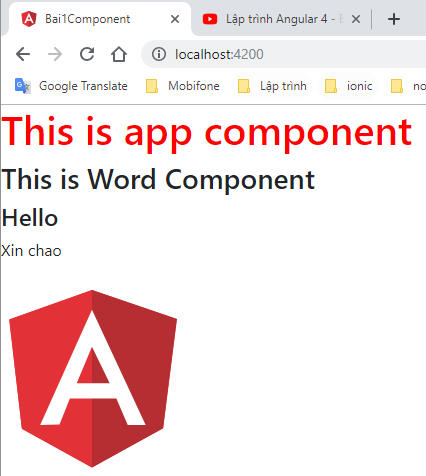


* + - Sử dụng các biến này trong file html, trong đó, thuộc tính **src** để trong dấu ngoặc vuông, còn giá trị **imageUrl** được để trong dấu ngoặc kép, khi đó Angular sẽ hiểu thuộc tính **src** của thẻ img sẽ có giá trị là giá trị của biến **imageUrl**:

**(**thuộc tính **src** sử dụng property binding)



* + - Kết quả hiển thị ra trình duyệt



* + - Thêm biến **forgot = “true”** vào class **WordComponent**
    - Sử dụng property binding biến **forgot** trong file html, như sau:



* + - Khi đó thẻ **<h4>** sẽ bị ẩn đi, khi biến **forgot** có giá trị **true**

1. **Event binding**
   * + Qua những bài học vừa rồi, ta đã sử dụng **string interpolation**, **property binding** để gán giá trị của biến từ file ts sang file html. Bây giờ, dùng event binding để gán giá trị từ file html sang file ts
     + Thêm một thẻ button vào file html, tạo event binding cho sự kiện click, để khi kích vào thì phủ định lại giá trị của biến **forgot**, như sau:



* + - Thông thường ta không gán event binding với một phép toán như trên mà gán với một hàm
    - Định nghĩa hàm này trong file ts:



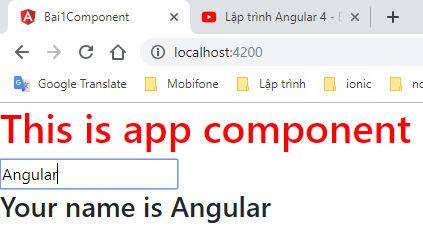
1. **Xử lý text input**
   * + Trong phần này, ta sẽ sử dụng kết hợp **property binding** và **event binding**
     + Dùng Angular CLI, tạo một component mới tên là user-form: **ng g c user-form**
     + ở file html, tạo một thẻ input để nhập dữ liệu và có một thẻ h3 để hiển thị dữ liệu, như sau:



* + - Mỗi khi nhập dữ liệu vào ô input thì sự kiện **keyup** sẽ thực hiện hàm **showEvent($event),** định nghĩa hàm **showEvent($event)** trong file ts, như sau:



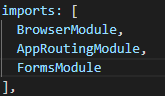
* + - Tham số **$event** là biến chứa nhiều thông tin về event mà mình vừa làm, biến này chứa nhiều thuộc tính, sẽ được truy cập để lấy các giá trị cần thiết
    - Khi chạy chương trình, mỗi khi nhập dữ liệu vào ô input thì giá trị vừa nhập sẽ được gán cho biến **name**, và được hiển thị trong thẻ h3, như sau:



1. **Two way binding**
   * + Để sử dụng được two way binding, cần import vào **app.module.ts** như sau:



* + - Thêm vào mảng import



* + - Thêm thuộc tính **[(ngModel)]** vào thẻ input như sau:



* + - Như vậy thì biến name sẽ được binding 2 chiều, từ file ts sang file html và ngược lại

1. **Template reference variable**
   * + Chỉnh sửa lại file html, thêm biến template reference, và hiển thị ra như sau, nhớ phải có thuộc tính **ngModel**



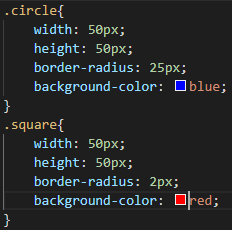
1. **Style binding và class binding**
   * + Thêm thuộc tính **style.color** là “red” vào thẻ h3 để đoạn text trong này hiển thị màu đỏ theo ý thích, như sau:



* + - Có thể thêm biểu thức vào cho thuộc tính **style.color** để khi chiều dài giá trị của biến name là chẵn thì hiển thị chữ màu đỏ, ngược lại thì hiển thị chữ màu đen, như sau:



* + - Trong file css, tạo ra các định dạng css cho các biến circle và square như sau:



* + - Trong file html, sử dụng class binding để khi giá trị biến isHighLight là true thì áp dụng class **circle**, ngược lại thì áp dụng class **square**



**(**nếu dùng binding theo cách này thì giá trị ở trong dấu “” là một biểu thức hoặc giá trị của một biến)

* + - Có thể dùng class binding theo cách này:



1. **Cơ bản về directive, ngStyle**
   * + Có 3 kiểu directive trong Angular:
       1. Component: directive với 1 Template
       2. Attribute directive: thay đổi việc hiển thị hoặc hành vi của các DOM, component hoặc directive khác
       3. Structural directive: thường thêm bớt các DOM **(ngIf, ngFor)**
     + Dùng **[ngStyle]** để định nghĩa nhiều thuộc tính style cho một tag



* + - Có thể định nghĩa biến **currentStyle** ở file ts, để gán các thuộc tính cho biến
    - Sau đó sử dụng biến này ở file html



* + - Có thể định nghĩa nhiều biến chứa các giá trị khác nhau của style



* + - Sau đó sử dụng các biến này ở file html với các điều kiện khác nhau:



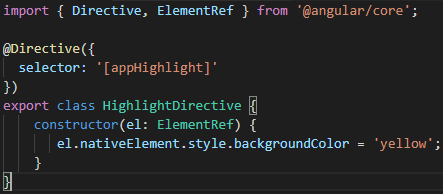
* + - Thuộc tính **ngClass** cũng tương tự như thuộc tính **ngStyle**
    - Giá trị truyền vào của các thuộc tính **ngStyle** và **ngClass** là một **Object**



**Ví dụ:** Xây dựng một **Attribute directive** đơn giản

Xây dựng một **Attrubute directive** để đặt màu nền cho một phần tử khi người dùng di chuyển chuột qua nó

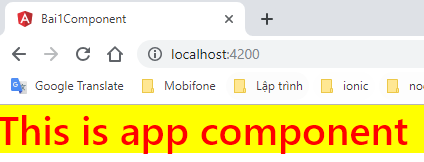
* + - 1. Khởi tạo lớp **directive** mới bằng lệnh Angular CLI: **ng g d highlight**
      2. Import thư viện **ElementRef**
      3. Chỉnh sửa lại file **highlight.directive.ts** như sau:



* + - 1. Để sử dụng directive vừa tạo ra, thêm nó vào một thành phần trong file html



* + - 1. Kết quả hiển thị ra trình duyệt như sau:

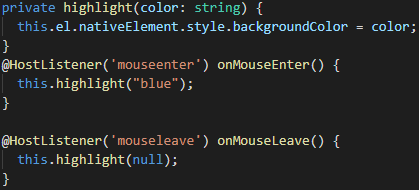


Tiếp theo, làm thêm để khi người dùng di chuyển chuột qua thành phần thì nó sẽ có hiệu ứng

* + - 1. Import thêm thư viện HostListener vào



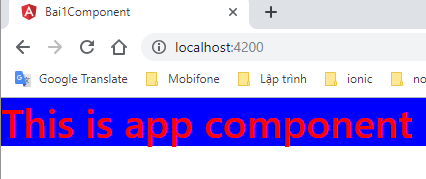
* + - 1. Sau đó, thêm hai trình xử lý sự kiện phản hồi khi người dùng di chuyển chuột vào và rời chuột khỏi



* + - 1. Thêm biến ElementRef vào constructor



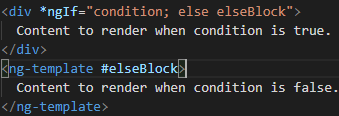
* + - 1. Chạy ứng dụng lên để xem hiệu ứng khi di chuyển chuột vào thì sẽ như sau:



1. **ngIf**
   * + **\*ngIf** là một **Structural directive**
     + Cú pháp đơn giản của **\*ngIf**



* + - \***ngIf** với else:



1. **ngFor đơn giản**
   * + \***ngFor** để lặp một dãy các giá trị và hiển thị ra
     + Khai báo một mảng các môn học trong file ts:



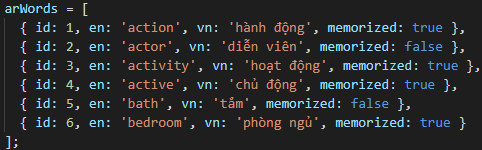
* + - Dùng \***ngFor** để hiển thị ra trong file html



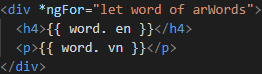
* + - Kết quả hiển thị ở trình duyệt



1. **ngFor với mảng đối tượng**
   * + Dùng \***ngFor** để lặp các phần tử trong một mảng các **object**
     + Tạo một component mới có tên **words**
     + Trong file ts, tạo một mảng các **words** như sau:



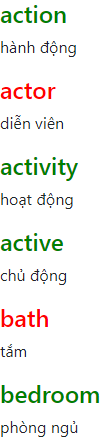
* + - Trong file html, dùng lệnh \***ngFor** để lặp các phần tử trong mảng và hiển thị ra như sau:



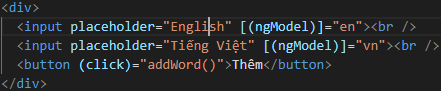
* + - Thêm thuộc tính [**ngStyle**] để định dạng các thẻ h4 có màu xanh khi thuộc tính **memorized** là **true** và màu đỏ khi thuộc tính **memorized** là **false**



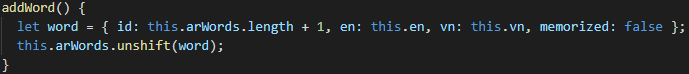
* + - Kết quả hiển thị ra trình duyệt như sau:



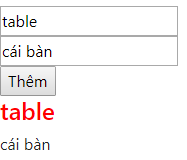
1. **Thêm phần tử vào mảng kết hợp ngFor**
   * + Tạo các thẻ input để nhập các dữ liệu ở file html
     + Tạo button để khi kích vào thì thực hiện hàm **addWord**()



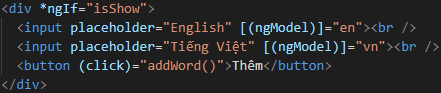
* + - Khai báo các biến **en** và **vn** để lưu các giá trị nhập vào từ file html
    - 
    - Định nghĩa hàm **addWord**() thực hiện thêm từ mới vào mảng **arWords**



* + - Kết quả hiển thị ra trình duyệt



1. **Ẩn hiện form**
   * + **Mô tả:**
       1. Lúc mới khởi động trình duyệt lên thì không hiện form thêm từ mới, chỉ hiện danh sách các từ và button **thêm từ mới**
       2. Khi kích vào button **thêm từ mới** thì hiện form để thêm từ mới
       3. Sau khi thêm từ mới xong thì ẩn form đi
     + **Các bước thực hiện:**
       1. Tạo button thêm từ mới: 
       2. Tạo biến **isShow=false** để ban đầu thì không hiện form 
       3. Sử dụng \***ngIf** để ẩn form đi

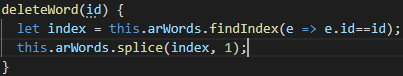


* + - 1. Trong định nghĩa hàm **addWord**(), sau khi thêm từ mới xong thì thay đổi giá trị biến **isShow** để ẩn form đi 

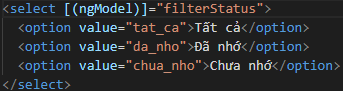
1. **Xóa phần tử trong mảng hiển thị**
   * + Thêm định dạng trong file css, để tách riêng mỗi phần tử hiển thị cho đẹp
     + Với mỗi phần tử, thêm một button để khi kích vào thì thực hiện hàm **deleteWord** xóa phần tử đó khỏi mảng, tham số truyền vào là **id** của phần tử muốn xóa



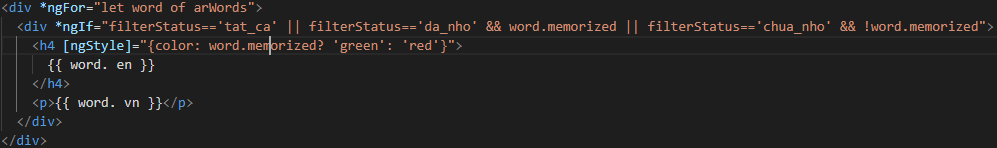
* + - Định nghĩa hàm **deleteWord**, như sau:



1. **Lọc mảng hiển thị**
   * + 1. Thêm thẻ **select** để xác định các từ muốn lọc



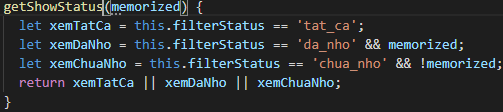
* + - 1. Dùng \***ngIf** để hiển thị những từ muốn lọc dựa vào **filterStatus** và thuộctính **memorized** của từ



* + - 1. Có thể tách riêng câu lệnh điều kiện ở \***ngIf** vào hàm **getShowStatus** cho gọn như sau:



* + - Định nghĩa hàm **getShowStatus** ở file ts



1. **Thay đổi trạng thái phần tử trong mảng hiển thị**
   * + Tạo button để khi kích vào thì thay đổi thuộc tính của đối tượng trong mảng hiển thị



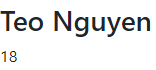
1. **Sử dụng Input để truyền tham số vào cho các component**
   * + Tạo ra các component có thể truyền tham số vào khi sử dụng, như vậy component sẽ linh hoạt hơn, và có tính tái sử dụng
     + Tạo component mới có tên **person**
     + Trong file html, sử dụng các thẻ h3 và p để hiển thị tên và tuổi



* + - Sử dụng component **person** trong file **app.component.html**



* + - Kết quả hiện thị ra trình duyệt



* + - Nhận thấy, component **person** chỉ được sử dụng với giá trị gán cứng và không thể gán giá trị khác cho component này được. Bây giờ sử dụng **Input** để gán giá trị cho component, như vậy component sẽ linh hoạt hơn, có thể tái sử dụng được
    - Trong file **app.component.html**, truyền thêm các biến vào cho component **person**:



* + - Trong file **person.component.ts**, import thêm thư viện **Input**



* + - Cũng trong file này, nhận các biến được truyền vào từ component



* + - Trong file **person.component.html**, dùng string interpolation để hiển thị giá trị các biến ra



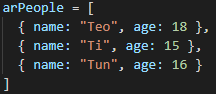
* + - Kết quả hiển thị ra trình duyệt cũng tương tự như trên, nhưng bây giờ, component **person** đã có thể tái sử dụng bằng cách truyền tham số vào cho nó:



* + - Kết quả hiển thị ra trình duyệt



1. **ngFor kết hợp Input**
   * + Sử dụng \***ngFor** để lặp mảng đối tượng và hiển thị ra, dùng **input** để truyền tham số vào cho component
     + Tạo component mới **list-person**
     + Trong file ts, khai báo mảng **arPeople** như sau:



* + - Trong file html, dùng \***ngFor** để lặp mảng **arPeople (**truyền tham số vào cho các biến) và hiển thị ra như sau:



Chú ý:

các tham số **name**, **age** phải dùng property binding hoặc string interpolation

Trong lệnh \***ngFor** ở trên, nếu để câu lệnh \***ngFor** trong thẻ div thì sẽ hình thành một thẻ div bao component **person** lại (bị dư ra một thẻ div),



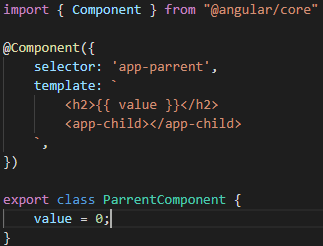
để tránh điều này, sử dụng **<ng-container>** thay cho thẻ div, như sau:



Lúc này thì không bị dư thẻ div nữa



1. **Output 1:** setup
   * + Mô tả: Setup để **parrent-component** hiển thị **value** của nó và render **child component** vào
     + Các bước thực hiện:
       1. Tạo **parrent-component.ts** như sau, nhớ import vào **app.module.ts**:



* + - 1. Tạo **child-component.ts** như sau, nhớ import vào **app.module.ts**:



* + - 1. Sử dụng **parrent** component trong file **app.component.html**



* + - 1. Kết quả hiển thị ra trình duyệt:



1. **Output 2:** tiếp cận các thuộc tính và phương thức của Component cha từ Component con
   * + **Các bước thực hiện:**
       1. Trong **parrent.component.ts**, khai báo một sự kiện **myClick** ở component **app-child**, sự kiện này do mình tự đặt tên



* + - 1. Trong **child.component.ts**:
         1. Import thêm thư viện **Output** và **EventEmitter** vào



* + - * 1. Nhận sự kiện **myClick**:



* + - * 1. Thêm sự kiện **click** vào button của **child.component**



* + - * 1. Định nghĩa hàm **addForParrent()** để khi kích vào button thì phát sinh sự kiện **myClick**



* + - 1. Kết quả hiển thị ở trình duyệt, mỗi khi kích vào button **ADD** thì phát sinh sự kiện **myClick** để tăng giá trị **value** lên 1



1. **Ouput có tham số**

# Mô tả: Cách làm thế nào để truyền tham số vào cho các sự kiện Output

# Các bước thực hiện:

# + Sử dụng hàm

# Thay đổi sự kiện myClick ở parrent component là một hàm changeValue

# 

# Định nghĩa hàm changeValue ở trong class parrent component

# 

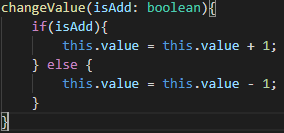
# Kết quả hiển thị ra trình duyệt cũng tương tự như trên

# + truyền tham số (1)

# Truyền tham số vào cho hàm changeValue là true (hoặc false)

# 

# Định nghĩa hàm changeValue để khi tham số là true thì tăng giá trị biến value lên 1, ngược lại thì giảm đi 1



1. Kết quả hiện thị ra trình duyệt, mỗi khi kích vào button thì phát sinh sự kiện **myClick** để thực hiện hàm **changeValue**, và tăng giá trị biến **value** lên 1, vì truyền vào tham số là **true, (**nếu truyền vào tham số là **false** thì sẽ giảm **value** đi 1)



1. Có thể truyền vào hàm **changeValue** tham số là **$event**



1. ở file **child.component.ts**, truyền giá trị **true** (hoặc **false**) vào sự kiện emit



1. kết quả hiện thị ra trình duyệt cũng tương tự như trên

# + truyền tham số (2)

1. ở file **child.component.ts**, tạo thêm một button để khi kích vào thì thực hiện hàm **subForParrent**



1. định nghĩa hàm **subForParrent**, truyền giá trị **false** vào sự kiện emit
2. Kết quả hiện ra ở trình duyệt, khi nhấn button ADD thì sẽ thực hiện hàm **changeValue**, tham số nhận vào là **true**, do hàm **addForParrent** ở **child**.component emit sang, còn nếu nhấn button SUB thì ngược lại

\*Lưu ý: Nếu chỉ muốn truyền tham số kiểu boolean thì phải thêm ràng buộc này vào EventEmitter



1. **Bài tập Output**

# Mô tả: sử dụng lại component person và list-person, tạo thêm nút xóa ở mỗi person, khi kích vào nút nào thì sẽ xóa phần tử đó ra khỏi mảng arPeople

# Các bước thực hiện:

# ở file list-person.component.html, thêm sự kiện removePerson ở trong thẻ app-person, để khi kích vào nút này thì thực hiện hàm removePersonByName

# 

# trong file list-person.component.ts, định nghĩa hàm removePersonByName tham số truyền vào là name, thực hiện xóa phần tử trong mảng arPeople dựa vào name

# 

# trong file person.component.ts, nhận sự kiện removePerson

# 

# Trong file person.component.html, tạo một button Xóa, khi kích vào thì chạy hàm removeByClick để phát sinh sự kiện removePerson

# 

# Trong file person.component.ts, định nghĩa hàm removeByClick để phát sinh sự kiện removePerson, tham số truyền vào là biến name từ input

# 

# Kết quả hiển thị ra trình duyệt, khi kích vào button Xóa thì xóa phần tử đó ra khỏi mảng

# 

1. **Viewchild 1:** setup

# Mô tả:

# tạo child component, có thẻ h3 hiển thị giá trị value

# tạo parrent component, có button add for child, và render child component

# Các bước thực hiện

# Tạo file child.component.ts, có biến value và hiển thị giá trị của biến trong thẻ h3

# 

* + - 1. Tạo file **parrent.component.ts**, có button **add for child** và render **child** component vào



1. **ViewChild 2:** tiếp cận các thuộc tính và phương thức của Component con từ Component cha

# Mô tả: Làm thế nào để khi kích vào button ở parrent component thì có thể thay đổi được giá trị của child component

# Các bước thực hiện:

# \*cách 1: Khai báo biến variable ở trong app-child, khi đó có thể dùng biến này để lấy các thuộc tính và phương thức của child component, vì vậy có thể dùng biến này để tăng giá trị value lên 1

# 

# Kết quả hiển thị ra trình duyệt, mỗi khi kích vào button thì sẽ tăng giá trị lên 1

# 

# \*cách 2:

# 1. trong parrent.component.ts, import thư viện ViewChild, và ChildComponent vào

# 

# 2. Nhận component ViewChild và khởi tạo biến myChild thuộc lớp ChildComponent

# 

# 3. Tại templates, khi kích vào button thì thực hiện hàm onAddForChild

# 

# 4. Tại class, định nghĩa hàm onAddForChild để tăng giá trị biến value lên 1

# 

# 5. Kết quả hiển ra ở trình duyệt cũng tương tự như trên

1. **ngContent:** tạo ra những component làm khung sườn cho những component khác
   * + Tạo component **card.component.ts** hiển thị một thẻ card, như sau:



* + - Định nghĩa định dạng CSS cho class **card**
    - Kết quả hiển thị ra trình duyệt là một card với nội dung được gán cứng

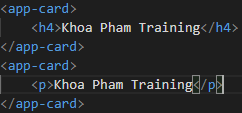


* + - Bây giờ, không truyền nội dung gán cứng cho component **card** nữa mà truyền cho nó một thẻ

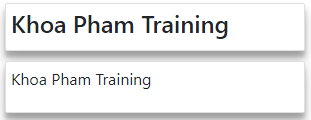
**<ng-content>**



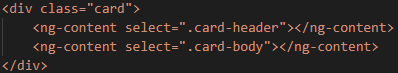
* + - Sử dụng component này trong file **app.component.html**, truyền nội dung cho **card** component, như vậy việc sử dụng component **card** sẽ linh động hơn, có thể tái sử dụng các component



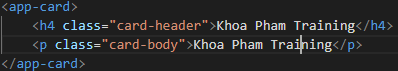
* + - Kết quả hiển thị ra trình duyệt



* + - Tiếp theo, truyền cho các thẻ **ng-content** thuộc tính **select** là class được khai báo ở **app.component.html**



* + - Sử dụng component này trong file **app.component.html**, nhớ định nghĩa các class



\*Lưu ý: Thứ tự hiển thị các thẻ phụ thuộc vào thứ tự ở trong component **card**

1. **Build in pipes**
   * + Pipe là công cụ để định dạng lại dữ liệu để show ra trình duyệt cho người dùng xem
     + Hầu hết các ứng dụng đều làm 3 công việc như sau:
       1. Lấy dữ liệu (từ server hoặc giá trị các biến trong component)
       2. Chuyển đổi dữ liệu đó thành dạng thân thiện với người dùng để người dùng có thể đọc hiểu được
       3. Show dữ liệu cho người dùng xem
     + Ví dụ:
       1. Tạo component mới đặt tên là **learn-pipe**
       2. Trong file ts, tạo biến **birthday** có kiểu dữ liệu **Date**



* + - 1. Trong file html, hiển thị giá trị biến **birthday**



* + - 1. Kết quả hiển thị ra trình duyệt là dữ liệu ở dạng thô



* + - Bây giờ dùng công cụ Pipe để chuyển đổi dữ liệu từ dạng thô sang dạng thân thiên hơn để người dùng có thể đọc hiều được
    - Truy cập vào <https://angular.io/api?query=pipe> để xem các Pipe được tạo sẵn của Angular và sử dụng
      1. Chỉnh sửa lại file html, sử dụng pipe để chuyển đổi sang dạng date



* + - 1. Kết quả hiển thị ra trình duyệt sẽ được dữ liệu thân thiện hơn



* + - 1. Có thể truyền tham số cho Pipe



* + - 1. Kết quả hiển thị ra trình duyệt



* + - Truy cập vào trang **Angular.io** để xem các pipe khác do Angular tạo sẵn
      1. Trong file ts, tạo biến json như sau:



* + - 1. Trong file html, hiển thị biến này ra, sử dụng pipe



* + - 1. Kết quả hiển thị ra trình duyệt



* + - 1. Có thể sử dụng kết hợp nhiều Pipe liên tiếp nhau



* + - 1. Kết quả hiển thị ra trình duyệt



* + - 1. Tạo biến kiểu Promise (là kiểu từ server trả về)



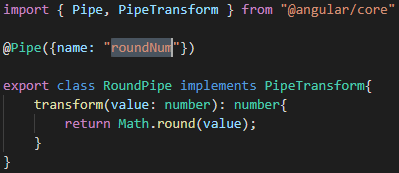
* + - 1. Dùng Pipe để chuyển đổi từ kiểu Promise về kiểu bình thường để có thể xem được



* + - 1. Kết quả hiển thị ra trình duyệt



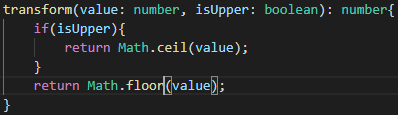
1. **Custom pipe**
   * + Định nghĩa ra các **pipe** và sử dụng
     + Ví dụ:
       1. Tạo pipe mới tên là: **round.pipe.ts** để làm tròn số, nhớ import vào **app.module.ts**
       2. Khai báo pipe này:



* + - 1. Sử dụng pipe này trong file **app.component.html**



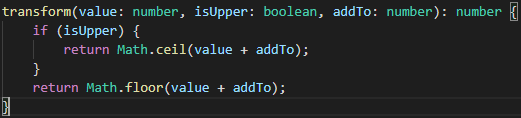
* + - 1. Kết quả hiển thị ra trình duyệt là số 1.9 được làm tròn thành 2
    - Truyền tham số cho các pipe
      1. Khai báo để truyền thêm tham số cho pipe



* + - 1. Sử dụng pipe và truyền thêm tham số vào



* + - 1. Truyền thêm nhiều tham số:



* + - 1. Sử dụng Pipe



* + - 1. Kết quả hiển thị ra trình duyệt là số 1.9 cộng với 10 rồi sau đó làm tròn lên được 12, số 3.5 cộng với 1 rồi sau đó làm tròn xuống được 4

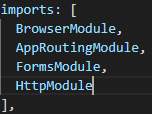
1. **Khái niệm service và http service**
   * + Service dùng để lấy dữ liệu từ server thông qua một API nào đó
     + Thêm, xóa, sửa dữ liệu thông qua API
     + Login service
     + Có thể viết service chung trong component, nhưng việc tách các service ra làm cho chương trình dễ quản lý và tập trung vào các công việc cụ thể, cũng giúp dễ dàng hơn khi test các service
     + Ví dụ: Tạo một Component để lấy dữ liệu từ server, viết chung service trong component luôn
       1. Tạo component mới, đặt tên là **ip.component.ts,** nhớ import component này vào **app.module.ts**
       2. Để lấy được dữ liệu từ server thì phải import thêm phương thức **Http** như sau



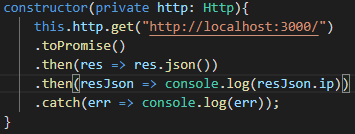
Cũng nhớ import **Http** vào **app.module.ts** luôn



Thêm vào mảng import:



* + - 1. Trong file **ip.component.ts**, khởi tạo constructor để get dữ liệu từ server như sau:



* + - 1. Kết quả là trình duyệt sẽ log ra kết quả ip trả về từ đường dẫn <http://localhost:3000/>
      2. Để lấy được giá trị từ server thì khai báo biến **ip** rồi gán như sau:

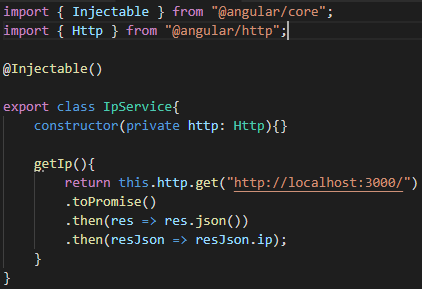


* + - 1. Sau đó dùng string interpolation để hiển thị giá trị **ip** để xem

+ Chú ý:

* + - 1. phương thức **http.get()** trả về dữ liệu dạng **Observable**, nên phải dùng **toPromise()** để chuyển về dạng Promise thì mới có thể xử lý được
      2. Lệnh **then** thứ nhất trả về dữ liệu kiểu json, dùng phương thức **.json** để xử lý nó
      3. Lệnh **then** thứ hai để log kết quả ra
      4. Lệnh **catch** để nếu xảy ra lỗi thì log lỗi ra

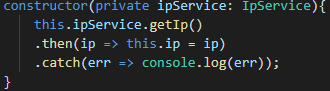
1. **Tách service**
   * + Việc viết service chung trong Component làm cho code khá dài, và gây khó khăn cho việc test
     + Cần phải tạo riêng một service để thực hiện chức năng get dữ liệu từ server
     + Ví dụ:
       1. Tạo file **ip.service.ts** để lấy dữ liệu từ server
       2. Khai báo service này, có phương thức **getIp** để trả về ip lấy được:



* + - 1. Khai báo service này vào mảng **providers** của component cần lấy ip



* + - 1. Sử dụng service này trong component để lấy **ip**



* + - 1. Kết quả hiển thị ra trình duyệt cũng tương tự như trên

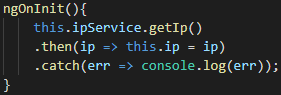
1. **Sử dụng ngOnInit**
   * + Việc sử dụng constructor để lấy dữ liệu từ service như trên là không đúng lắm, vì vậy phải sử dụng hàm ngOnInit để lấy dữ liệu, hàm constructor chỉ nên dùng để khai báo các biến mà thôi
     + Các bước thực hiện:
       1. Để sử dụng được phương thức **ngOnInit**, phải import thêm thư viện **OnInit**



* + - 1. Trong **IpComponent**, class **IpComponent** phải implements **OnInit**



* + - 1. Trong class IpComponent, định nghĩa hàm **ngOnInit** để lấy dữ liệu từ server



* + - 1. Cần thêm service vào mảng provides ở file **app.module.ts** để có thể sử dụng service này ở component nào cũng được 



1. **Giới thiệu bài tập thời tiết**
   * + Tạo giao diện nhập vào tên một thành phố bất kỳ, hiển thị ra nhiệt độ của thành phố đó

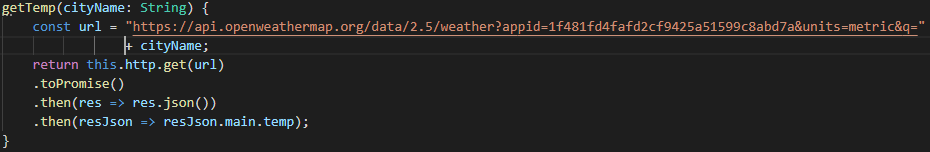


* + - Nếu nhập vào thành phố không tồn tại thì cũng thông báo cho người dùng biết
    - Nhiệt độ được lấy từ trang web: <https://api.openweathermap.org/data/2.5/weather?appid=1f481fd4fafd2cf9425a51599c8abd7a&units=metric&q=hanoi>, cung cấp nhiệt độ của một thành phố nào đó

1. **API open weather map**
2. **Weather service**
   * + Tạo giao diện, tạo ra service để lấy nhiệt độ
     + **Các bước thực hiện:**
       1. Tạo component mới: **ng g c weather**
       2. Trong file html, tạo giao diện cho chương trình



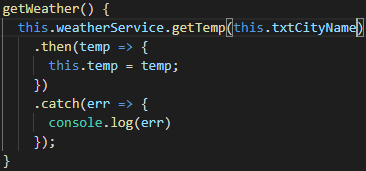
* + - 1. Trong component **weather** vừa tạo, tạo một file mới tên **weather.service** để viết service, nhớ thêm service này vào mảng **providers** của **app.module.ts**
      2. Trong weather service này, khởi tạo một hàm **getTemp** để trả về nhiệt độ của một thành phố, tham số nhận vào là tên một thành phố (**cityName**)



1. **Lấy nhiệt độ và hiển thị**
   * + Thêm sự kiện **(click)=”getWeather()”** cho component weather để khi kích vào thì thực hiện lấy nhiệt độ
     + Thêm thuộc tính **[(ngModel)]** để binding giá trị cho biến **txtCityName**



* + - Định nghĩa hàm **getWeather** để lấy nhiệt độ của một thành phố, sử dụng hàm **getTemp** của weather service đã định nghĩa, tham số truyền vào là giá trị **txtCityName**



* + - ở file html, thêm điều kiện để hiện thông báo yêu cầu nhập hoặc nhiệt độ đã lấy được



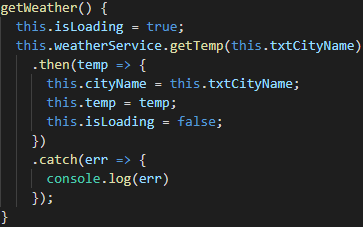
+ Chú ý: Nhớ khai báo các biến **temp**, **txtCityName, cityName**

* + - Kết quả hiển thị ra trình duyệt

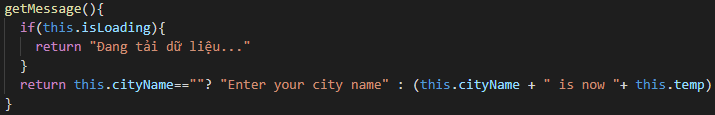


1. **Tính năng loading và xử lý lỗi**
   * + Trong lúc đợi lấy dữ liệu từ server thì hiển thị thông báo cho người dùng biết là đang tải dữ liệu
     + Sau khi lấy xong thì hiển thị nhiệt độ cho người dùng biết, đồng thời xóa sạch ô input ban đầu
     + Nếu xảy ra lỗi thì thông báo cho người dùng biết
     + Các bước thực hiện:
       1. Trong file ts, tạo thêm biến 
       2. Khi bắt đầu get dữ liệu gán biến ,
       3. Sau khi lấy dữ liệu xong, gán lại biến 

Cụ thể hàm getWeather sẽ như sau:



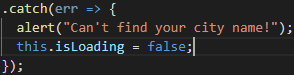
* + - 1. Định nghĩa hàm **getMessage** để hiện thị thông báo



* + - 1. Trong file html, hiển thị thông báo bằng hàm **getMessage**



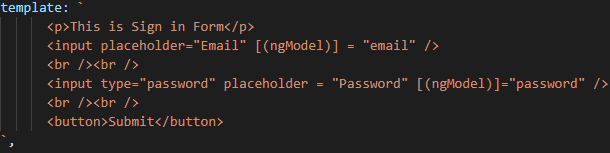
* + - 1. Nếu nhập vào thành phố không tồn tại, biến isLoading luôn có giá trị true, nên cứ loading mãi, để tránh điều này, cần xử lý ở phần catch như sau:



* + - 1. Đồng thời sau khi hoàn thành get dữ liệu, phải gán lại giá trị ban đầu cho các biến



1. **Giới thiệu về form**
   * + Hầu hết các ứng dụng đều thu thập thông tin của người dùng thông qua các form
     + Angular hỗ trợ việc xử lý form input rất tốt
     + Ví dụ:
       1. Tạo một component mới tên là **sign-in.component.ts**
       2. Trong **template** của component này, hiển thị 2 thẻ input để nhập **email** và **password**, và 1 button để thực hiện submit form

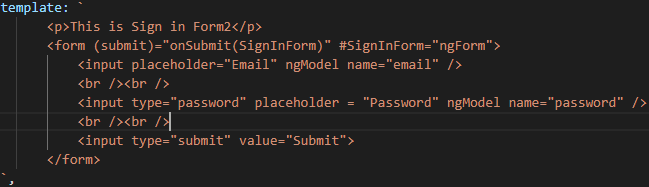


* + - 1. Thêm sự kiện (click) cho button Submit, và định nghĩa hàm **onSubmit,** để log ra giá trị email và password nhập vào từ ô input



* + - 1. Kết quả, sau khi nhập thông tin vào ô input và nhấn submit, trình duyệt sẽ log ra giá trị email và password nhập vào từ ô input

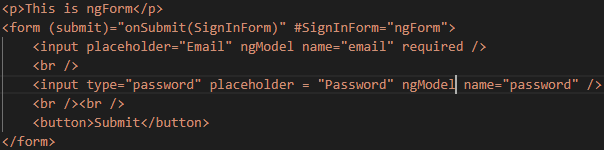
1. **ngForm:** tài liệu tham khảo: <https://angular.io/guide/forms>
   * + có 2 kiểu form thường được sử dụng là **Template-driven** Form và **Reactive** Form
     + Trong bài này sẽ tìm hiểu về **Template-driven Form**
     + Trong form thì có các thẻ input để nhập thông tin
     + Có sự kiện **(submit)** để submit giá trị của form
     + Có thuộc tính **ngModel,** và kèm theo là thuộc tính **name**
     + Mỗi form đều được gắn 1 giá trị là **ngForm**
     + **Ví dụ:**
       1. Tạo một component mới tên là **sign-in2**
       2. Khai báo 1 form chứa 2 thẻ input, mỗi thẻ input có thuộc tính **[(ngModel)]** để binding giá trị cho ô input và kèm theo thuộc tính **name**
       3. Gán sự kiện **(submit)=”onSubmit()”** cho thẻ form, gán biến **#SignInForm=”ngForm”**



* + - 1. Định nghĩa hàm **onSubmit** để log ra giá trị nhập vào từ ô input



* + - 1. Kết quả, sau khi nhập thông tin vào ô input và nhấn submit, trình duyệt sẽ log ra giá trị email và password nhập vào từ ô input

1. **Validate template form**
   * + Validate là kiểm tra dữ liệu nhập vào trước khi submit lên server
     + Giúp cho người dùng biết được trường nào họ nhập vào có thể sai
     + **Ví dụ**: có những trường yêu cầu phải nhập, có trường nhập sai định dạng…
     + **Ví dụ áp dụng:**
       1. Tạo component mới tên là **ng-form.component.ts,** nhớ import vào app.module.ts
       2. Tạo form có 2 thẻ input để nhập **email** và **password**
       3. Thêm thuộc tính **required** cho thẻ input **email**
       4. Kết quả hiển thị ra trình duyệt



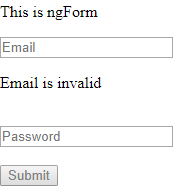
* + - 1. Khi chưa nhập thông tin cho thẻ email thì giá trị của thuộc tính invalid của thẻ này là false, do đó, giá trị thuộc tính của form cũng là form
      2. Thêm thuộc tính [**disabled**] vào thẻ button để khi giá trị form invalid thì ẩn nút submit đi



* + - 1. Thêm điều kiện để nếu trường email **có lỗi** thì hiện thông báo



* + - 1. Kết quả hiển thị ra trình duyệt, nếu trường email chưa nhập thì hiện thông báo lỗi, và nếu form invalid thì ẩn nút submit đi



\*chú ý: Phải thêm điều kiện  để nếu email là **null** hoặc **undefined** thì không truy cập tiếp (khỏi xảy ra lỗi)

1. **ngSubmit**

# các thuộc tính của ngForm thường đi theo cặp, và có giá trị đối nhau

# dirty: true ⬄ pristine: false ô input đã nhập chưa?

# touched: true ⬄ untouched: false: đã chạm vào ô này chưa?

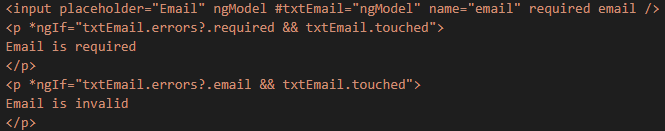
# valid: true ⬄ invalid: false: đã valid ô này chưa?

# submitted: true ⬄ pedding: false: đã submit chưa?

# Khi thực hiện throw new Error("Form is invalid!"); để ném ngoại lệ, thì giá trị của form sẽ được submit đi

# Sử dụng (ngSubmit) để khi thêm câu lệnh throw ở file ts thì form chỉ ném ra ngoại lệ ở console mà không cho submit, tức là không tải lại trang

1. **Xử lý style các input không hợp lệ**
   * + 1. Tạo biến template variable để giúp câu lệnh truy cập đến các thuộc tính ngắn gọn hơn
       2. Thêm validate cho thẻ input là phải đúng định dạng email
       3. Nếu ô input này invalid và đã touched vào thì hiển thị câu thông báo lỗi

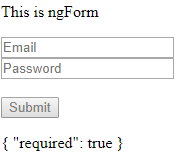
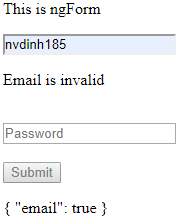


* + - 1. Cuối cùng hiện thông báo lỗi ở dưới cho người dùng biết



* + - 1. Kết quả hiển thị ra trình duyệt, ban đầu thì chỉ hiện thông báo lỗi ở dưới là required: true,

Nếu touched vào ô email thì hiển thị thông báo lỗi ở dưới ô này, nếu nhập email sai định dạng cũng hiện thông báo lỗi



* + - Xử lý style các input không hợp lệ
      1. Vào inspect để biết class của những ô không hợp lệ



* + - 1. Dựa vào class để định dạng CSS cho các thẻ này trong file **style.css**



* + - 1. Kết quả hiển thị ra trình duyệt, khi ô input invalid và đã touched vào thì sẽ bị tô màu viền đỏ

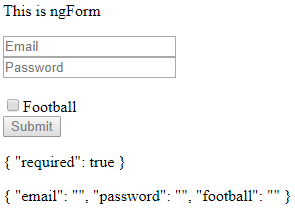
1. **Xử lý checkbox và ngModelGroup**
   1. **checkbox**
      * Thêm thẻ **checkbox** vào cho form



* + - Hiển thị giá trị form ngay dưới thẻ form luôn



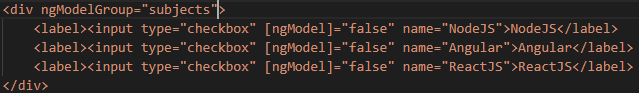
* + - Kết quả hiển thị: ngay khi mở trình duyệt lên thì giá trị đều là null



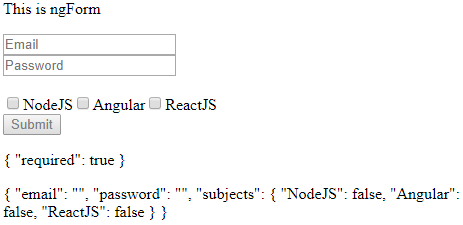
* + - Dùng event-binding để gán giá trị ban đầu là false cho thẻ checkbox



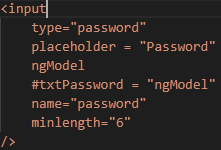
* 1. **ngModelGroup**
     + Tạo 1 thẻ div chứa 3 thẻ checkbox là tên 3 môn học
     + Thêm thuộc tính **ngModelGroup** cho thẻ div
     + Khi đó hình thành một đối tượng mới chứa các môn học



* + - Kết quả hiển thị ra trình duyệt:



* 1. **Các loại validator**
     + Vào đường dẫn <https://angular.io/api?query=validator> để xem các loại validator của Angular
     + Thêm validator **minlength** cho input **password**



* + - Hiện lỗi ngay dưới form để xem



* + - Kết quả: Nếu nhập vào password bé hơn 6 ký tự thì sẽ hiển thị thông báo lỗi

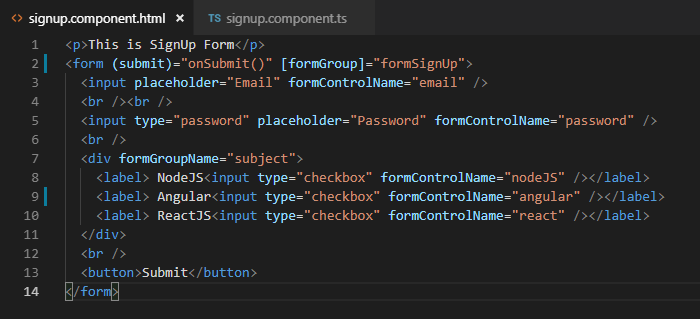


* + - Thêm validator **pattern = “[a-z]\*”** cho ô input **password**, có nghĩa là ô này chỉ được nhập các chữ cái từ a đến z 
    - Kết quả: nếu nhập vào password không thỏa mãn thì sẽ hiển thị 2 lỗi



Nếu nhập vào password thỏa mãn thì lỗi là **null**

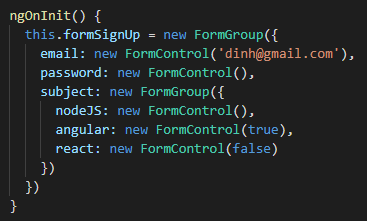
1. **Reactive form**
   1. **Reactive Forms là gì?**
      * **Reactive** Forms là một phương pháp để tạo form trong Angular
      * Trong **Reactive** forms, chúng ta tạo toàn bộ form control ở trong code (khởi tạo ngay, khởi tạo trong constructor, hoặc khởi tạo trong ngOnInit), nên có thể dễ dàng truy cập các phần tử của form ngay tức thì.
   2. **Các thành phần cơ bản của form**
      * AbstractControl là một abstract class cho 3 lớp con: **FormControl**, **FormGroup**, và FormArray.
        1. **FormControl** là đơn vị nhỏ nhất, nó lưu giữ giá trị và trạng thái hợp lệ của một form control. Tương ứng với một HTML form control như **input**, **select**.
        2. **FormGroup** nó lưu giữ giá trị và trạng thái hợp lệ của một nhóm các đối tượng thuộc AbstractControl – có thể là **FormControl**, **FormGroup**, hay FormArray
        3. **FormArray** nó lưu giữ giá trị và trạng thái hợp lệ của một mảng các đối tượng thuộc AbstractControl giống như FormGroup.
   3. **Tạo file templates:**



* 1. **Import APIs cho Reactive forms**
     + Để có thể sử dụng các APIs mà Angular cung cấp cho việc thao tác với Reactive forms, chúng ta cần import NgModule là **ReactiveFormsModule** từ package **@angular/forms** như sau:



* 1. **Khởi tạo form trong Component**
     + khởi tạo trong ngOnInit.



* 1. **Reactive Forms Validator**
     + Việc thêm Validator vào cho một control rất đơn giản, việc của bạn là thêm giá trị tham số tiếp theo như sau:



Ở đoạn code trên, chúng ta gộp required và minLength(3) cho control email, hoặc có thể thêm nhiều các validator function nữa theo ý muốn.

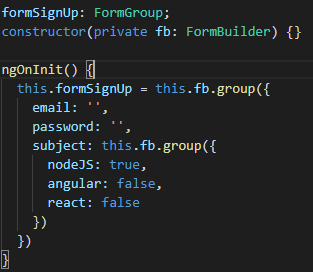
* + - Để hiển thị thông báo cho người dùng biết về lỗi, chúng ta thêm đoạn code sau trong file html



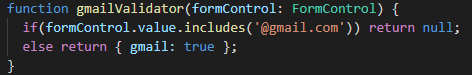
* 1. **Form Builder**
     + **Form Builder** giúp tạo form model một cách nhanh chóng và tiện lợi
     + **Import** FormBuilder vào class muốn tạo form



* + - gọi các API như group, array, control để tạo form.



* 1. **Tùy chỉnh ngoại lệ Reactive form**
     + Là tạo ra các **validator** do người lập trình quy định
     + Tạo ra các Validator cũng như tạo các function bình thường khác
     + Ví dụ dưới đây tạo một Validator cho một formControl, để quy định email nhập vào phải đúng định dạng gmail



* + - Việc thêm Validator vào cho một control tương tự như thêm các Validator mặc định khác



* + - Muốn hiển thị xem Validator có lỗi gì không, ta thêm đoạn code này vào file html



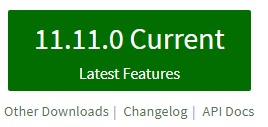
* + - Muốn sử dụng nhiều Validator cho một biến nào đó, thêm vào mảng Validator như sau:



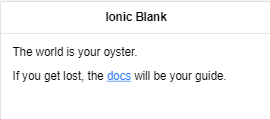
**IONIC**

1. **Khái niệm**
   * + Ionic framework là một framework sử dụng HTML5 và Angular để xây dựng các ứng dụng moble chạy đa nền tảng.
     + Ứng dụng xây dựng bằng ionic framework có thể chạy được trên web, trên android, trên ios
     + Xây dựng một lần duy nhất và sau này có thể chạy trên tất cả nền tảng
     + Trang chủ của ionic framework: https://ionicframework.com/
2. **Cài đặt môi trường phát triển**
   1. **Cài đặt nodeJS:**

- vào trang chủ của nodeJS, download và tiến hành cài đặt https://nodejs.org/en/



* + - Kiểm tra cài đặt thành công hay chưa: **node -v**
  1. **Cài đặt ionic-cordova CLI**
     + Đây là công cụ hỗ trợ thực hiện tạo project, tạo page, tạo component ở trên một project ionic
     + Chạy lệnh **npm install -g ionic cordova** để tiến hành cài đặt ionic và cordova
  2. **Tạo project mới** 
     + **ionic start tên\_project loại\_giao\_diện**
     + có các loại giao diện là: **blank**, **tabs**, **sidemenu**…
  3. **Khởi chạy project:** 
     + **ionic serve (–port ????)**
     + Kết quả hiển thị trên trình duyệt, chọn môi trường ios:



1. **Cấu trúc thư mục**
   1. **Giới thiệu cấu trúc thư mục**
      * Trong project vừa tạo, cũng có các thành phần tương tự như của Angular
      * File src/index.html là file chính của dự án
      * Mục đích của nó là thiết lập các tập lệnh, khai báo CSS, bootstrap và component root
      * Toàn bộ code của dự án được viết trong thư mục **./src/**
      * File **app.module.ts** là module chạy đầu tiên của ứng dụng
        1. Trong file này có phần khai báo các Page, component, service…
        2. Khi khai báo một page thì cũng khai báo vào entryComponent
        3. Mảng providers dùng để khai báo các service
        4. Bootstrap: khai báo component chạy đầu tiên
      * Thư mục assets để khai báo các thành phần CSS, các hình ảnh…
      * Thư mục theme chứa file **variables.scss**, dùng để định nghĩa các tài nguyên
2. **Cơ chế navigation: NavController**
   * + Ionic không sử dụng Router như trong Angular
     + Sử dụng navigation để chuyển trang theo kiểu ngăn xếp (Stack of Pages)
     + Ví dụ 1: chuyển trang đơn giản
       1. Tạo trang mới: **ionic g page users,** một thư mục mới được tạo ra chứa 4 file: html, scss, module.ts và ts. Trong 1 trang gồm có các thành phần: **header**, **content**, **footer**
       2. Từ trang **home**, muốn chuyển sang trang **users**, ta làm như sau:
          1. Trong file **home.html**, tạo một button để khi kích vào thì thực hiện hàm **goToUsers**



* + - * 1. Trong file **home.ts,**

Import page **users** vào



định nghĩa hàm **goToUsers**, để chuyển sang trang users



* + - * 1. Kết quả, ở trang home, khi nhấn vào button thì sẽ chuyển sang trang **users**
    - Ví dụ 2: chuyển trang có truyền tham số
      1. Trong file **home.ts**, tạo danh sách **users**:



* + - 1. Để chuyển sang trang users, có truyền tham số là danh sách users, thêm thuộc tính này vào hàm **goToUsers**



* + - 1. ở file **users.ts**, nhận giá trị này bằng hàm **ngOnInit**, như sau:



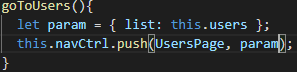
* + - 1. ở file **users.html**, dùng \***ngFor** để lặp danh sách **users** và hiển thị ra trình duyệt



* + - 1. kết quả, khi nhấn button chuyển sang trang **users** thì hiển thị danh sách users
      2. **Nếu truyền tham số là một object:**



**Hoặc**



* + - 1. **Nhận tham số như sau:**



* + - Ví dụ 3: Tạo trang user detail, để khi kích vào từng user ở list users thì sẽ chuyển sang thông tin chi tiết của một user
      1. Tạo trang mới user: **ionic g page user**
      2. Tại file users.html, tạo một sự kiện, để khi kích vào thì chuyển sang trang chi tiết



* + - 1. **Tại file users.ts, định nghĩa hàm goToUserDetail để chuyển sang trang user**



* + - 1. **Tại file user.ts, dùng hàm ngOnInit để nhận dữ liệu từ Users chuyển sang**



* + - 1. **Hiển thị thông tin ra ở file user.html**



* + - Ví dụ 4: Trở về trang trước
    - Mặc định **NavController** tạo một button back để trở lại trang trước
    - Mở rộng: trở về trang trước bằng cách khác
      1. **NavCtrl.pop()**
      2. **NavCtrl.popToRoot()**
    - Ví dụ áp dụng:
      1. Tại file **user.html**, tạo sự kiện để khi kích vào thì trở về trang users



* + - 1. Tại file **user.ts**, định nghĩa hàm **goToUsers**



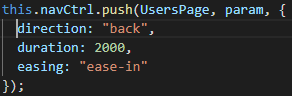
* + - Ví dụ 5: Trở về trang chủ
      1. Tại file **user.html**, tạo sự kiện để khi kích vào thì trở về trang chủ



* + - 1. Tại file **user.ts,** định nghĩa hàm **goToHome**

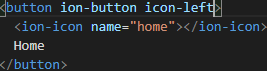
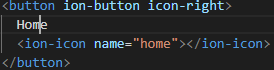
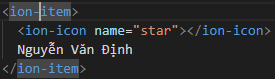


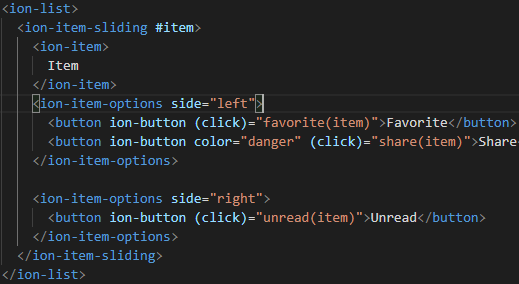
* + - Chuyển trang sử dụng **directive**
      1. Tạo đối tượng thuộc trang cần chuyển => [navPush] = “đối tượng trang cần chuyền”
      2. Truyền tham số: [navParams]=”data”
    - Trở về: sử dụng directive: navPop
    - Chuyển trang sử dụng **animation**: (Tham số thứ 3 của NavController.push)
      1. direction: “back” hoặc “forward” (mặc định) => nếu dùng direction: ”back” thì sẽ gỡ trang hiện tại ra và gắn trang mới vào
      2. duration: 2000 =>, thời gian chuyển trang
      3. easing: “ease-out”/”ease-in”



1. **Các component thường sử dụng**

Truy cập trang chủ của ionic framework để sử dụng các component: https://ionicframework.com/docs/v3/components/

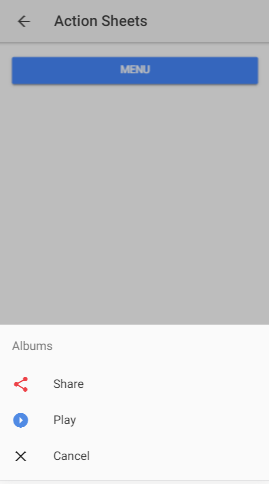
* 1. **Button**
     + **Button** được tạo ra bằng thẻ **button** trong html kết hợp với directive **ion-button** của ionic 3
     + Thông qua button có thể bắt các sự kiện và xử lý bằng event binding, ví dụ click, double click, long click,…
     + Để thay đổi màu sắc button ta sử dụng thuộc tính **color**, các biến chứa mã màu được lưu trữ tại **theme/variables.scss**. Các màu cơ bản: **primary**, **secondary**, **danger**, **light**, **dark**. Mặc định **primary**
     + Thuộc tính **outline**: tạo ra button chỉ có đường viền và thiết lập màu trước khi kích và sau khi kích
     + Thuộc tính **clear**: xóa toàn bộ màu viền và màu nền
     + Thuộc tính **round**: tạo button có đường viền bo tròn ở bốn góc
     + Thuộc tính **block**: hiển thị theo 1 hàng độc lập
     + Thuộc tính chỉnh kích cỡ: **small**, **medium**, **large**
     + Ví dụ:
       1. **Default**:  
       2. **Light:**  
       3. **Outline:** 
       4. **Clear:**  
       5. **Block:**  
       6. **Small:**  
       7. **Icon in button:** 
       8. **Icon-only:**  tăng kích thước icon:  
       9. **Icon-left:**  
       10. **Icon-right:**  
  2. **List**
     + Dùng để hiển thị các dòng thông tin: danh bạ, danh sách nhạc, menu
     + Cú pháp: **ion-list**
     + Bên trong list có thể là một
       1. Ion-item
       2. Button
     + Theo mặc định, list sẽ có đường ngăn giữa các thành phần, để bỏ đi, thêm thuộc tính no-lines
     + Mỗi thành phần  là một nút có mũi tên bên phải ở chế độ ios
     + Thông thường sử dụng các mục như này: 
     + Icon in list:  
     + Thẻ <ion-item-sliding> ở trong <ion-list> để hiện tùy chọn khi trượt sang trái hoặc phải



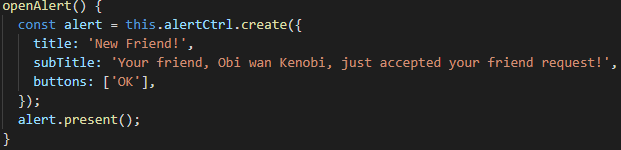
* 1. **ActionSheet**
     + **ActionSheet** là bảng thông báo trượt lên từ cạnh dưới của thiết bị và hiển thị các tùy chọn
     + Các thành phần: **title, button** (trong button có các thành phần như **text**, **handle**, **icon**…), …
     + Muốn sử dụng **ActionSheet**, import vào ****
     + Khởi tạo: ****
     + Ví dụ:

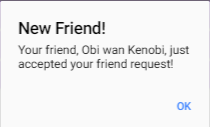


* + - Kết quả hiển thị ra trình duyệt:

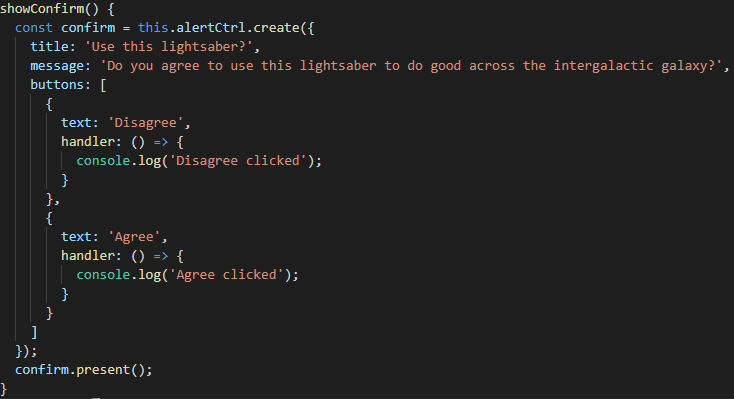


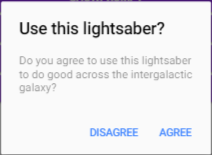
* 1. **Alert**
     + Alert cung cấp bảng chọn để người dùng xác nhận hoặc có nhiều tùy chọn khác nhau
     + Các thành phần: Prompt Alerts, Confirmation Alerts, Radio Alerts, Checkbox Alerts
       1. Alert chỉ được chọn OK





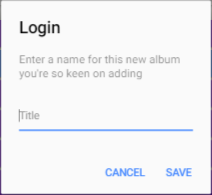
* + - 1. Alert để xác nhận (Đồng ý hoặc không?)



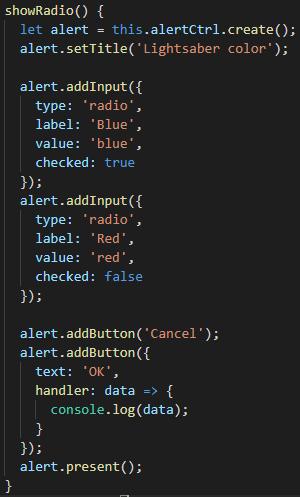


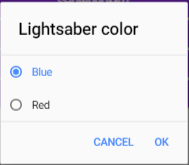
* + - 1. Alert để nhập thông tin



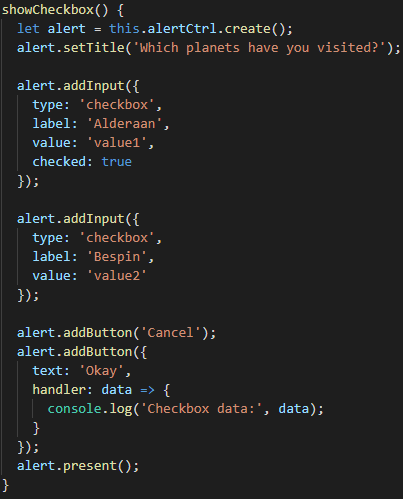


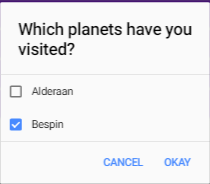
* + - 1. Alert hiện các radio để chọn (chỉ được chọn 1)





* + - 1. Alert hiện các checkbox để chọn (0, 1 hoặc nhiều)

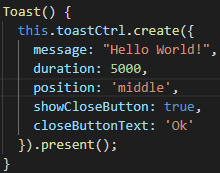




* 1. **Badges**
* Badges là các thành phần nhỏ thường truyền đạt một giá trị số cho người dùng. Chúng thường được sử dụng trong <ion-item>.
* **Ví dụ:**

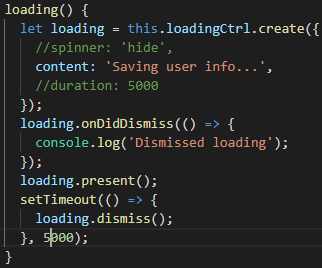


* 1. **Toast**
* Toast là một thông báo xuất hiện trên đầu nội dung của ứng dụng.
* Thông báo Toast sẽ tự động mất đi sau một khoảng thời gian nhất định
* Các thành phần:
  + Message: thông báo
  + Duration: thời gian diễn ra (ms)
  + Position: vị trí (top/bottom/middle)
  + showCloseButton: hiện nút để click close (true/false)
  + closeButtonTex





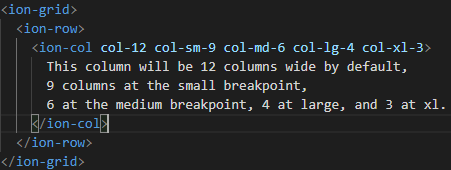
* 1. **Loading**
* Loading là lớp phủ ngăn chặn sự tương tác của người dùng trong khi chỉ ra hoạt động
* Loading diễn ra khi đang trong quá trình tải dữ liệu,…
* Các thành phần:
  + Content: nội dung hiển thị
  + Duration: thời gian hiển thị
  + spinner: 'hide' => ẩn biểu tượng loading
* Dùng lệnh **loading.dismiss();** để dừng loading
* Hàm **loading.onDidDismiss()** thực hiện lệnh sau khi đã loading xong
* Lệnh **loading.present()** để thực thi lệnh loading





* 1. **Grid**
* Hệ thống lưới được tạo thành từ 12 cột và mỗi ion-col có thể được định kích thước bằng cách

đặt thuộc tính **col- <width>**



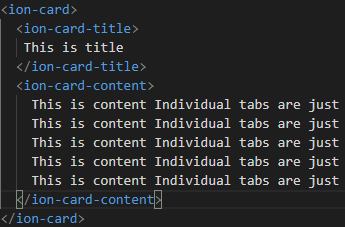
* 1. **List mở rộng**
* Sự kiện **reorder="true"** và **ionItemReorder** cho phép dùng chuột kéo/thả để sắp xếp lại danh sách
* Sự kiện ionRefresh cho phép refresh lại danh sách khi kéo chuột xuống, sau khi refresh thì danh sách được cập nhật lại
* Sự kiện **ionInfinite** cho phép tải thêm dữ liệu nếu kéo chuột xuống
  1. **Slide**
* Khai báo **@ViewChild(Slides) slides: Slides;** để có biến slides làm việc với slide
* Lệnh **this.slides.slideTo(1, 500);** để chuyển đến slide mong muốn
* Lệnh **this.slides.getActiveIndex();** để lấy chỉ số slide hiện tại
* Lệnh **this.slides.slideNext(500, true)** để chuyển đến slide tiếp theo
* Sự kiện **ionSlideDidChange** thực hiện khi slide được thay đổi
* Lệnh **this.slides.lockSwipes(true);** cho phép chuyển giữa các slide
* Thuộc tính **centeredSlides="true"** để canh giữa slide
* Thuộc tính **autoplay="2000"** để tự động chuyển slide sau 2 giây
  1. **Items**
* Một **item** cơ bản phải được viết dưới dạng phần tử **<ion-item>**
* ion-item có thể được thêm vào <button>
* Theo mặc định, <button> và <a> với thuộc tính ion-item sẽ hiển thị biểu tượng mũi tên phải trên chế độ ios.
* Để ẩn/hiện mũi tên bên phải, thêm các thuộc tính **detail-none/detail-push**
* Thuộc tính **text-wrap** giúp đoạn text tự động xuống hàng khi quá dài
* Thuộc tính **disabled** để vô hiệu hóa các button
  1. **Segment**
* **Segment** là một nhóm các nút, đôi khi được gọi là Điều khiển phân đoạn, cho phép người dùng tương tác với một nhóm nhỏ gọn của một số điều khiển.
* **ngSwitch/ ngSwitchCase** để chọn ra các trường hợp điều kiện

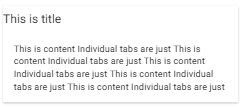


* **Kết quả hiển thị ra trình duyệt**

* 1. **card**





**XỬ LÝ BẤT ĐỒNG BỘ TRONG JAVASCRIPT**

**1. Khai báo biến với từ khóa let, biến chỉ có hiệu lực trong phạm vi cục bộ**

**2. Khai báo hàm kiểu mới** : arrow function

**3. Bất đồng bộ:** Câu lệnh trước chưa chạy xong thì đã chạy câu lệnh sau

\* Lưu ý: Một hàm có thể là đồng bộ hoặc bất đồng bộ, muốn biết phải đọc document tại

https://nodejs.org/api/fs.html

**4. CallBack function:**

- số lượng, thứ tự tham số của callback do người viết hàm quy định

- nếu giá trị underfined trong return callback nằm sau thì có thể không cần viết

- lúc sử dụng phải lưu ý thứ tự các tham số trong callback

\* Lưu ý: sau câu lệnh log thì cộng thêm dấu '' để tránh hiện thông báo lỗi dài dòng

Ví dụ: Viết hàm add để tính tổng 2 số, có sử dụng callback,

sau đó sử dụng hàm add() và xử lý err, result

**5. CallBack Hell:**

- Có thể sử dụng các hàm callback lồng nhau. khi đó, có thể sử dụng có kết quả của

hàm callback trước làm tham số cho hàm callback sau

- Chú ý thứ tự các tham số của hàm callback

Ví dụ:

- Viết hàm tính chu vi hình chữ nhật, truyền vào tham số a, b, h

- Viết hàm tính diện tích hình thang, truyền vào tham số a, b, h và

hàm cb là một callback function, sử dụng các hàm lồng nhau để tính được kết quả cuối cùng

**6. Promise,** trả về 2 kết quả là resolve và reject tương ứng với 2 trạng thái thành công

hoặc thất bại. sau đó, dùng lệnh then và catch để thực thi kết quả Promise,

\* Lưu ý: Nếu vừa có lệnh resolve vừa có lệnh reject thì câu lệnh nào viết trước

sẽ được thực hiện

**7. Tái sử dụng Promise** có nghĩa là sử dụng Promise trong 1 hàm, vậy thì hàm này sẽ

trả về 1 Promise nên có thể xử lý như một Promise và có thể truyền tham số vào cho nó

Ví dụ: Viết hàm read, tham số truyền vào là filename, sử dụng Promise để xử lý.

Sau đó sử dụng hàm read

**8. Promise liên tiếp**: có thể sử dụng các Promise liên tiếp nhau,

khi đó kết quả của câu lệnh trước sẽ dùng làm tham số cho câu lệnh sau,

Sau cùng là một lệnh catch để bắt ngoại lệ (nếu xảy ra lỗi thì sẽ nhảy đến lệnh này)

\* Lưu ý: - Chỉ dùng Promise cho một hàm con trong cùng

- chỉ dùng catch cho hàm ngoài cùng

**9. Promise status và Promise value**

- Promise (Status) có 2 trạng thái, đang chờ (pending) và hoàn thành (resolve/reject)

- Promise (Value) có thể có giá trị underfined hoặc giá trị nào đó (do hàm then trả về)

- Câu lệnh then luôn trả về Promise có Status là resolve cho dù có bị xử lý lỗi hay không?

Vì khi nhảy vào then thì kết quả đã được xử lý rồi (có lỗi hoặc không)

còn giá trị của Promise trả về chính là bằng giá trị của câu lệnh then return ra

(nếu không có câu lệnh trả về (return) thì giá trị trả về là underfined)

- có thể có nhiều lệnh then lồng nhau, giá trị của Promise chính là giá trị trả về

của lệnh then sau cùng, và kết quả của lệnh then trước sẽ làm tham số cho lệnh then sau

**10. Phương thức resolve và reject**

- Tạo ra các biến Promise trả về phương thức resolve

- Tham số truyền vào của resolve có thể là một hàm Promise, khi đó trạng thái của

biến là pending, nên cần phải xử lý then với nó

- Nếu phương thức của Promise là reject thì phải xử lý catch với nó

**11. Phương thức all và race**

+ tham số truyền vào là một mảng các Promise

- Phương thức all:

+ trả về một mảng các giá trị của các Promise

+ nếu một tham số bị lỗi thì sẽ nhảy đến catch và báo lỗi của Promise bị lỗi

+ nếu có nhiều tham số bị lỗi thì tham số bị lỗi nào kết thúc trước

thì sẽ hiện lỗi của tham số đó

- Phương thức race:

+ trả về kết quả của Promise kết thúc trước

**12. Sử dụng Babel CLI**

- Để dịch những syntax javascript mới mà chưa được hỗ trợ thành javascript hiện tại

để có thể chạy được trên máy => Phải sử dụng thư viện babel

- Tạo folder mới để chứa dự án, mở folder này với VS Code

- Tạo file package.json: npm init -y

- Tạo file index.js trong dự án

- Cài đặt thư viện babel: npm i --save-dev babel-cli

- Chỉnh sửa lại file package.json, thêm đoạn lệnh "start": "babel-node index.js"

để khi chạy lệnh npm start thì sẽ chạy dự án lên

- Cài preset cho babel để nó dịch file index này qua javascript bình thường mà

node có thể chạy được: npm i --save-dev babel-preset-es2015 babel-preset-es2017

- Tạo file config .babelrc để dịch được file index thành javascript phiên bản cũ hơn

- Nội dung của file babelrc là {"presets": ["es2015", "es2017"]}

- Sau đó, chạy lệnh npm start để chạy dự án lên

**13. Làm quen với async – await**

- Từ khóa await chỉ được sử dụng trong async function

- Từ khóa await sẽ dừng hàm async lại (không chạy các câu lệnh tiếp theo ở dưới),

nó sẽ chờ Promise trả về kết quả rồi gán vào biến, sau đó mới chạy tiếp các câu lệnh

tiếp theo

**14. Sử dụng await liên tiếp**

- Viết hàm tính diện tích hình thang, sử dụng async-await

- Sử dụng try...catch để xử lý nếu xảy ra lỗi

- Để tái sử dụng async function thì có thể dụng callback function

**15. Async function trả về promise**

- Viết lại hàm tính diện tích hình thang, nhưng không sử dụng callback function

mà sử dụng Promise, trả về Promise.resolve hoặc Promise.reject

- Gọi hàm tính diện tích, sau đó xử lý then và catch