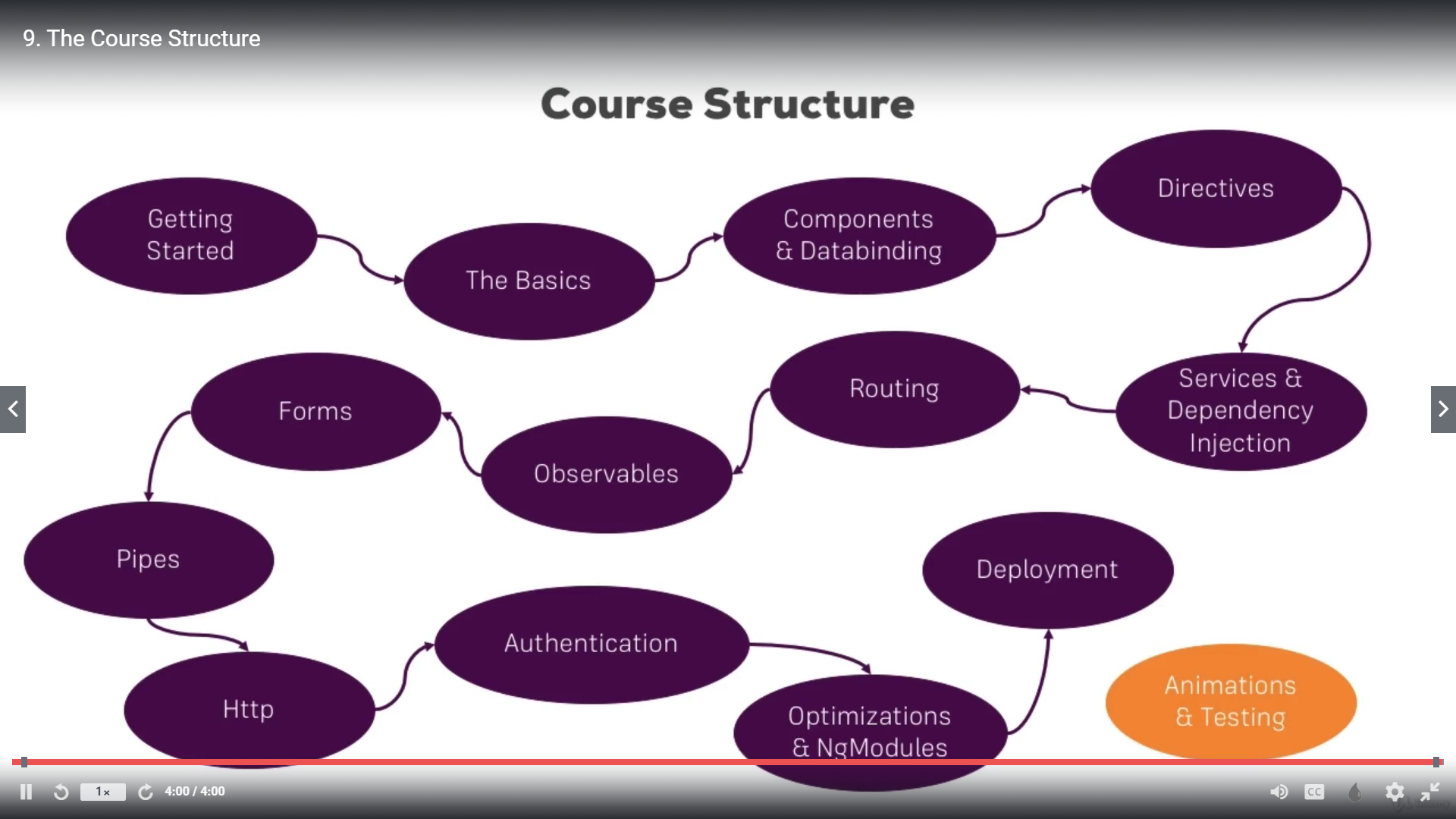
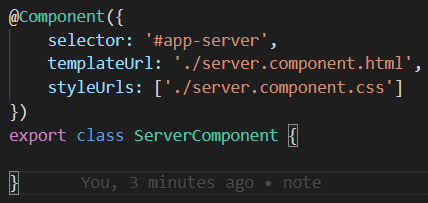
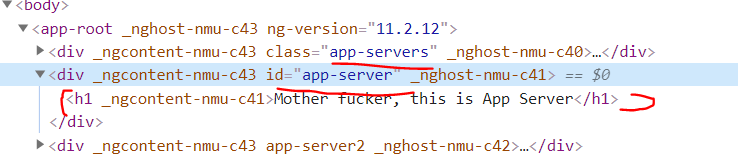
-ngModel là một thành phần của FormsModule, phải import FormsModule thì mới sử dụng được

- ngModel là một directive, dùng [(ngModel)] để gọi.

- ngModel sử dụng cơ chế two-way data binding, đồng bộ dữ liệu giữa component với dom và ngược lại

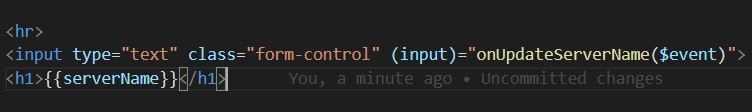
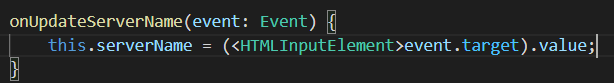
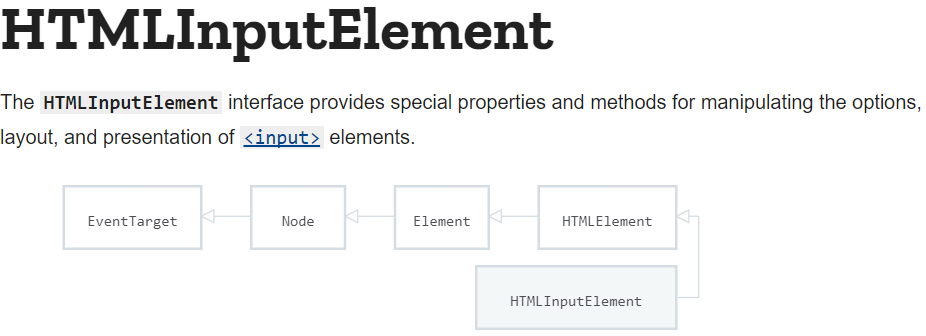
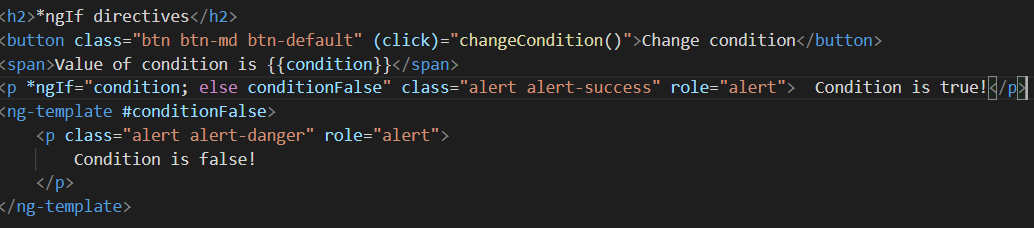
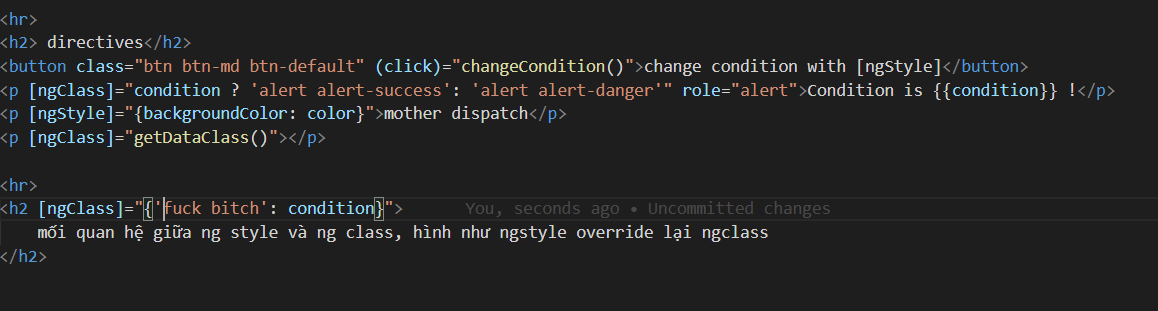
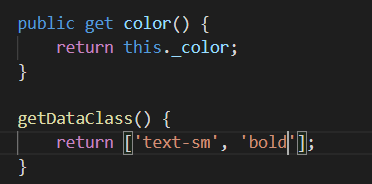
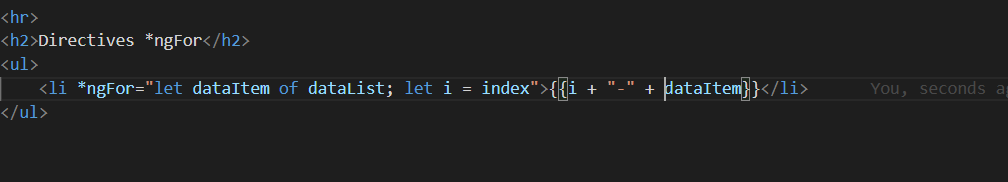
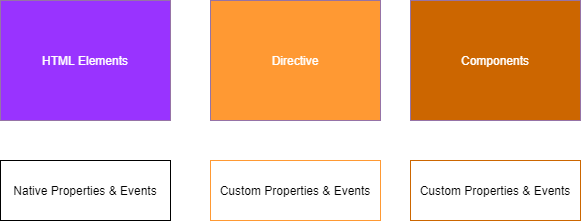
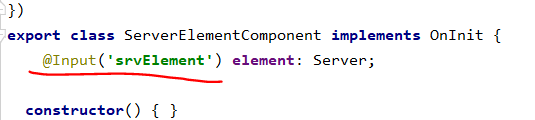
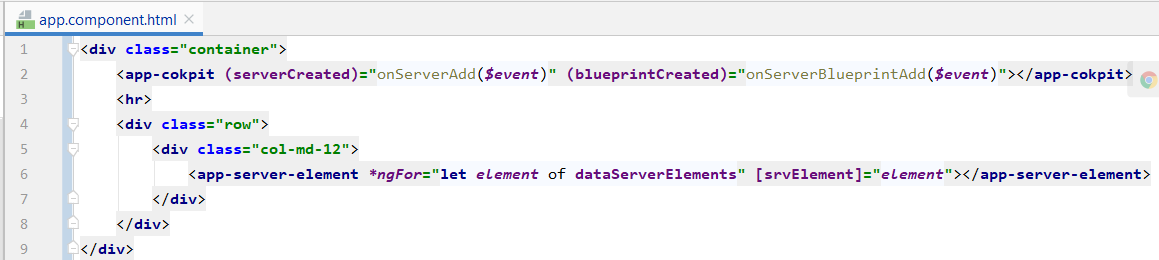
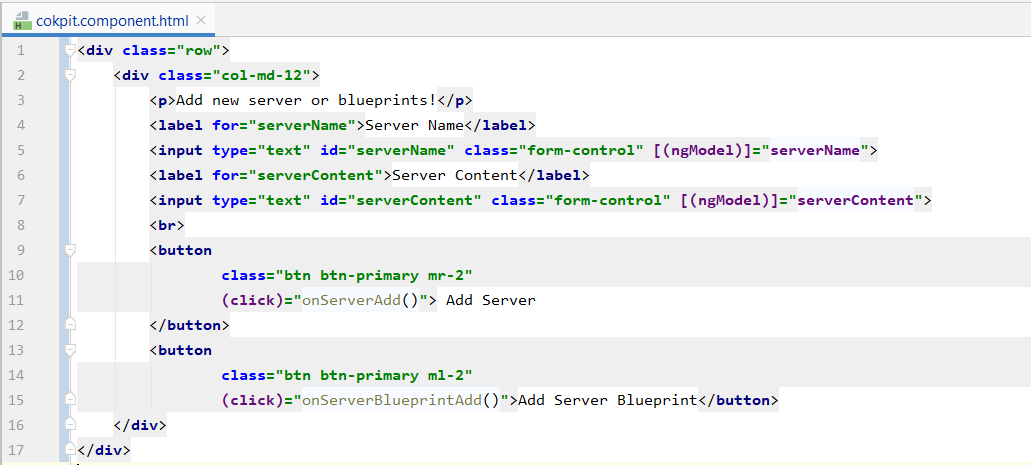
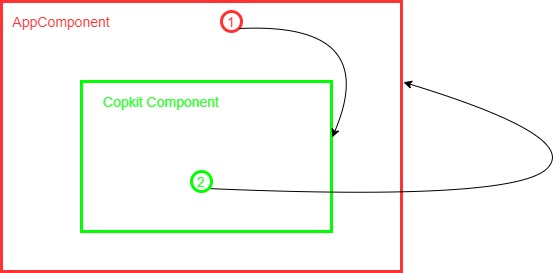
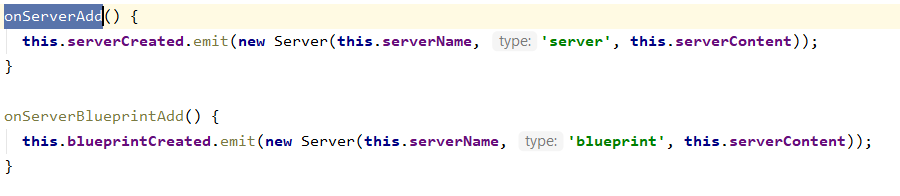
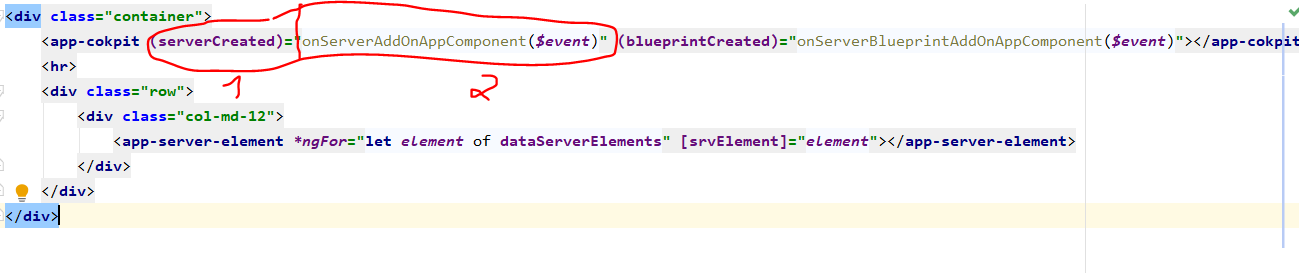
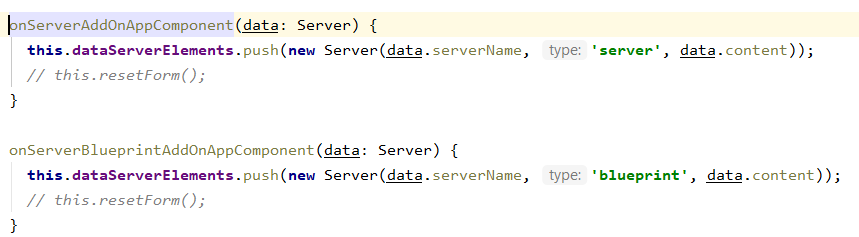
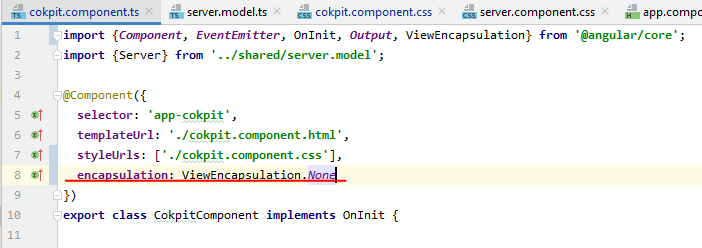
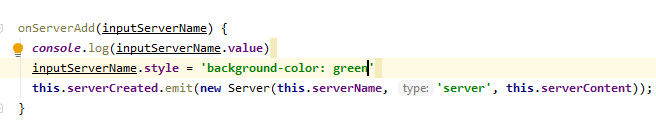
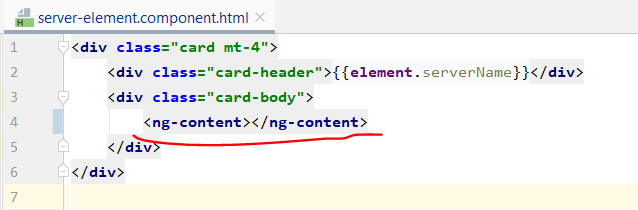
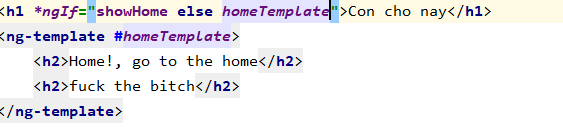
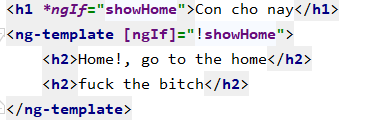
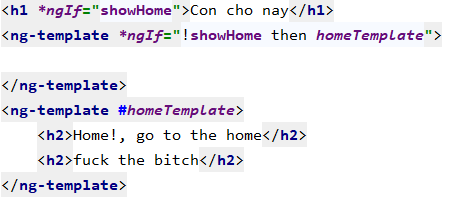
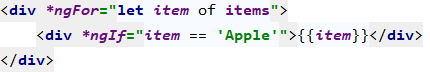
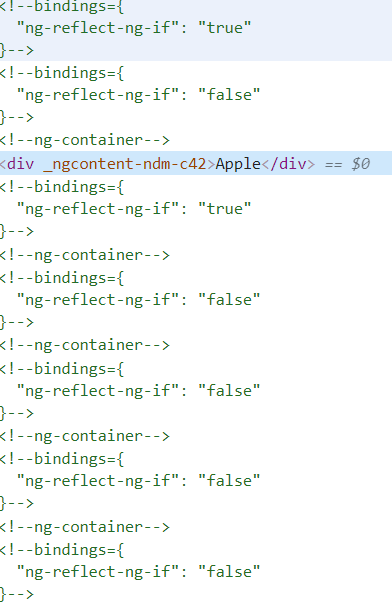
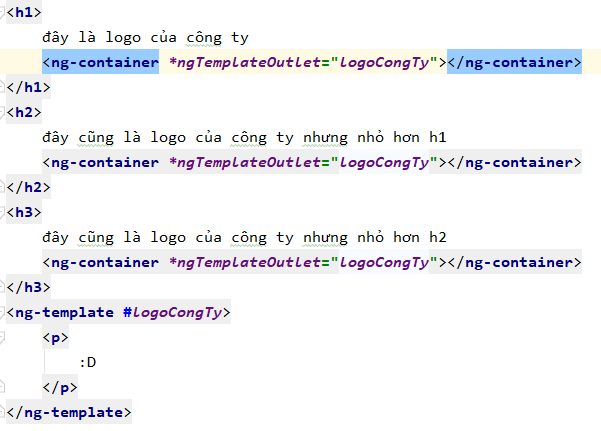
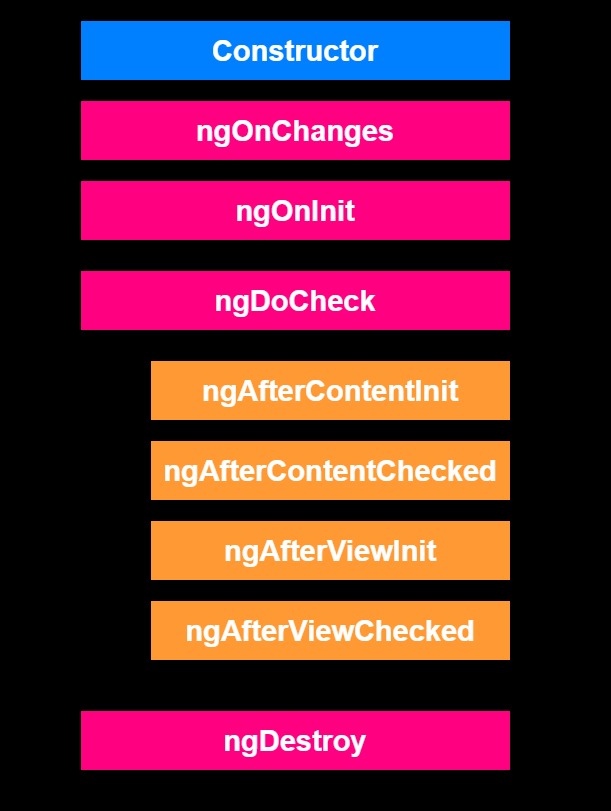
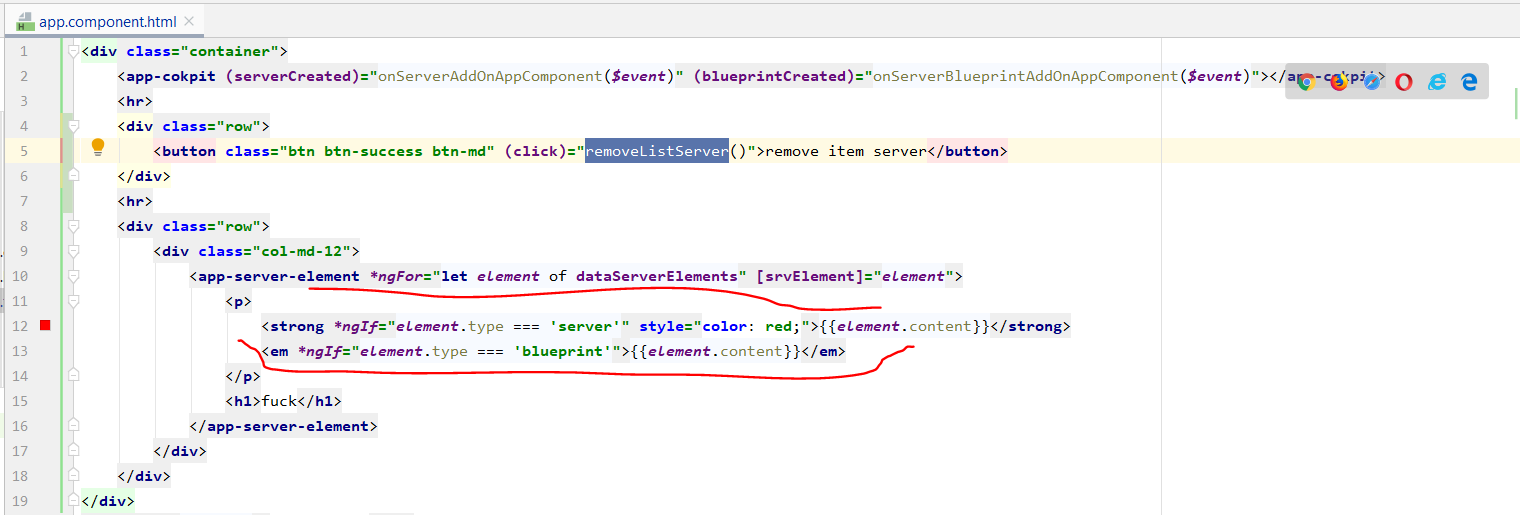
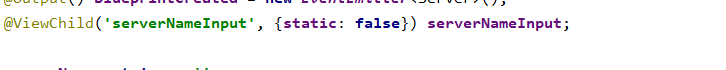
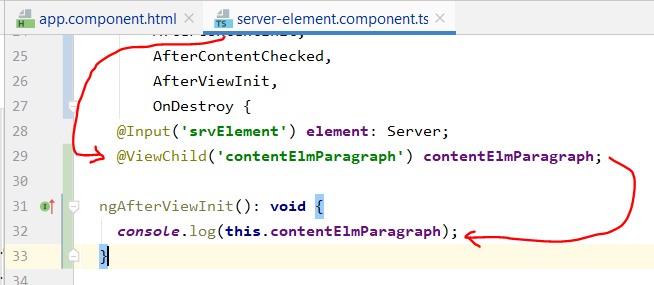
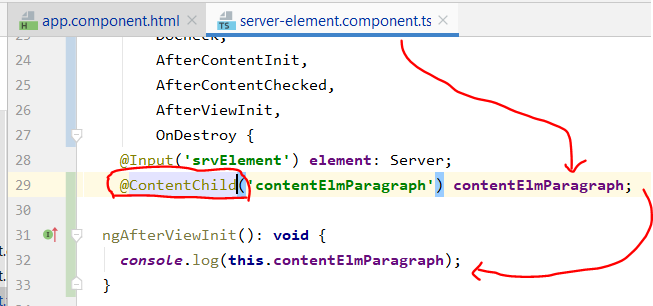
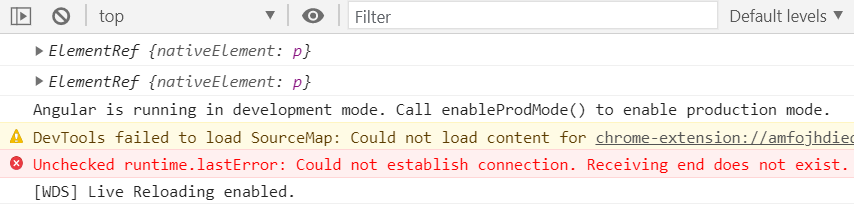
  
selector trong angular component, có 4 kiểu đặt selector:

1. Kiểu mặc định  
     
   Khi gọi component này, ta sử dụng cú pháp : <app-root></app-root>
2. Kiểu thứ 2 là sử dụng mảng:  
     
   Khi khai báo selector kiểu này, ta khai báo với cú pháp  
   <div app-server2></div>
3. Kiểu thứ 3 là dưới dạng #Selectors  
     
   Ta sử dụng cú pháp <div id=”app-server”></div>
4. Kiểu cuối cùng là dạng class như Jquery: .Selector  
     
   ta sử dụng cú pháp : <div class=”app-server”></div>

Đây là các cú pháp selector, tuỳ vào cách khai báo selector mà angular detect theo cách đó mà tìm chỗ khai báo component để rồi ném template của component vào (component sẽ là element con của các thẻ có id hoặc class hoặc attribute ở trên). Giả sử khai báo dạng id thì angular tự tìm element có id là app-server, dạng class thì tự tìm class,…  
  
Nhưng kiểu khuyên dùng vẫn là dạng normal là selector:‘app-root’  
+> Binding attribute , ta sử dụng ngoặc vuông : [disabled] =”allowDisable”

+> String interpolation: {{ dataOutput }} hiển thị data ra ngoài màn hình, khác với Property Binding, bind data ra attribute của HTML Element. Tuy nhiên ta cũng có thể binding data ra màn hình với property binding là [innerText]. Không được mix String interpolation và Data binding. String Interploation cuối cùng cũng sẽ convert về Property Binding. Ta có thể dùng String Interpolation ở property, tuy nhiên là không nên dùng như vậy:  
<button disabled={{allowDisable}}>Login </button> (chỉ support một số các property built in của Angular)

string interpolation thì convert data sang string, còn property binding thì không

1. Event binding ta sẽ dùng dấu ngoặc tròng (click)=”data()”  
     
   Khi mà click thì angular sẽ bind event vào trong hàm onUpdateServerName.  
     
   Mặc định hàm event.target không có chưa function value, để get value của input bằng event, ta phải ép kiểu target về HTMLInputElement.  
     
   \*\* FormModule cần phải có để two-way data binding hoạt động, cần phải enalbe ngModel
2. Directives: là một đối tượng giúp ta thay đổi một đối tượng khác (Directive is instruction in the DOM)  
   có các loại directives:  
   - Components directive: Có thể gọi Component là Directive cmnl, Directive có selector, và nhúng vào html với selector tag  
   - Structural directives: \*ngFor, \*ngIf  
   - Attribute directives: \*ngStyle  
   \*\* Đối với ngIf, template sẽ re-render lại cái đoạn nằm trong if chứ ko hidden đi  
   - Đối với Directives \*ngIf , ngoài cách else bằng (condition ? a : b), ta dùng **ng-template** và **local reference** để else data như sau :  
     
   trỏ tới ng-template bằng local reference  
   - Không giống như Structual Directives: attribute directives chỉ add hoặc remove cái element mà nó điều khiển, ko có add vs remove nhiều elements khác.  
     
     
   Ta có thể cấu hình style bằng object hoặc chuỗi string, cấu hình class bằng 1 object, 1 mảng hoặc một chuỗi string  
   - Directives: \*ngFor  
   
3. **Properties Binding  
   **Chuyên mục truyền dữ liệu qua lại giữa các Components
   1. Concept đầu tiên của Binding Properties là @Input()  
        
      Tạo một property binding mới của ng-component, cho phép gán giá trị cho sererElements từ component khác (appComponent gọi cái thằng AppServerElementComponent và nó truyền data cho thằng AppServerElementComponent). Việc bind data nó sẽ chạy trước khi AppServerElementComponent  
        
      Đôi khi ta muốn đổi tên của @Input, ta dùng cú pháp @Input(‘aliasName’)  
        
      Ngoài ra, ta có thể gán Input ở cái @Component chứ ko cần dùng annotation @Input, tuy nhiên là éo nên dùng  
      Ta có thể dùng @Input ở hàm setter
   2. Component Event với EventEmitter và @Output  
        
      Copkit Component được gọi trong AppComponent.  
        
        
        
      \*\* Chu trình gọi của EventEmitter  
      ****  
      Khi nhấn nút OnServerAdd, sự kiện nhấn sẽ gọi tới hàm onServerAdd() và onBlueprintAdd() của CopkitComponent. Tại đây, với sự khai báo của @Output và EventEmitter  
        
      Sẽ emit data ngược lại AppComponent dưới dạng $event  
        
      và khi đó, hàm onServerAddOnAppComponent() và onServerBlueprintAddOnAppConponent() khai báo ở AppComponent được trigger và gửi data vào  
        
      Tương tự như @Input thì @Output cũng có thể change Alias name để phù hợp với ngữ cảnh code
   3. View Encapsulation: Đóng gói css  
      Có 3 loại style của view Encapsulation là : ShowDow, None và Emulated  
        
      có nghĩa là : quản lý việc implement css của các component, Component này style sẽ không ảnh hưởng tới component kia hoặc là có ảnh hưởng phụ thuộc vào việc config encapsulation này
   4. Local Reference Template  
      LocalReference : dùng để đánh dấu template element với dấu #  
        
      nó như một biến ElementRef (hoặc getElementById) trỏ tới element đó  
        
      Ta có thể get value hay làm gì đó với nó  
      
   5. Tiếp theo là đến ViewChild  
      Trước kia, chưa có cách nào để truy cập trực tiếp từ Typescript code tới LocalReference Template, nay ta có thể dùng ViewChild  
      Khai báo một ViewChild  
        
      Trong khi LocalReference Template console log ra 1 cái input html thì ViewChild console log ra ElementRef , ta có thể sử dụng nhiều hàm của đối tượng ElementRef này  
        
        
      ViewChild có thuộc tính static, khi mà static = false thì ở Lifecycle Hook ngOnInit() ta không thể dùng được thằng this.serverNameInput, console log sẽ ra undefined, ta chỉ có thể truy cập sớm nhất ở Hook AfterViewInit()  
      Khi để static = true, ta có thể truy cập ViewChild ở ngOnInit()
   6. Sử dụng component linh hoạt với ng-content  
        
      Ta truyền nội dung HTML vào trong thân của <app-server-element> từ component cha là AppComponent  
        
      sau đó sử dụng <ng-content> ở component con. Lúc đó ServerElementComponent sẽ lấy toàn bộ content mà AppComponent truyền vào để render  
      Đôi khi ta không muốn lấy toàn bộ nội dung thẻ mà Component cha truyền xuống, ta có thể sử dụng thuộc tính select để select thẻ mà ta mong muốn như sau:  
        
      Ta chỉ có thể select từ thằng root như <p> hay <h1> chứ không thể select <strong> hay <em>  
      Đi cùng với <ng-content> ta có <ng-container>, <ng-template> và \*templateOutlet  
      \*\* <ng-template> là một đối tượng Template trong Angular, có thể sử dụng kèm với các directive như \*ngIF, \*ngFor, \*ngSwitch và Custom Directive  
        
      Không thể dùng directive \*ngIf trên bản thân <ng-template> được, ta phải dùng @Input [ngIf], cách viết trên có thể được viết lại như sau:  
        
      Nếu vẫn muốn cố chấp sử dụng \*ngIf, ta có thể dùng cú pháp sau  
        
      Tuy nhiên cách này dài và không hiệu quả, chúng ta nên sử dụng Property Binding là [ngIf]  
      \*\* <ng-container>  
        
      Khi chúng ta gặp trường hợp một số directive không được cùng nằm trên 1 element, chúng ta phải viết ntn. Tuy nhiên, ta có thể thấy  
        
      Nó tạo ra rất nhiều thẻ div, mặc dù ta chỉ hiển thị mỗi cái thằng Apple, khi mà data nhiều lên và ứng dụng có nhiều component thì sẽ tạo ra rất nhiều DOM element, để giải quyết vấn đề này, ta dùng <ng-container>  
        
        
      Ở đây chúng ta chỉ có một thẻ div thôi  
      \*\* Tái sử dụng code ở nhiều chỗ với \*ngTemplateOutlet  
        
      Ta sử dụng lại template logoCongTy ở 3 chỗ
   7. Component Lifecycle  
      Tất cả các Component của Angular đều có một vòng đời khép kín (constructor không nằm trong Lifecycle hook)  
        
      - Khi mà Angular thiết lập các thuộc tính đầu vào ràng buộc dữ liệu. Được gọi trước ngOnInit() và bất cứ khi nào thuộc tính đầu vào thay đổi.  
        
      Giả sử như ta for cái <app-server-element>, mỗi lần for là nó new mới 1 cái component, chạy từ constructor với ngOnChanges, ngOnInit, ….. Khi ta bấm nút xoá để xoá 1 phần tử của mảng Data thì Angular sẽ phải destroy cái Component bị xoá chứ không re-render toàn bộ.  
      Do đó, vòng đời nó liên quan đến việc Template Access thông qua ViewChild  
        
      nếu console log viewChild ở OnInit thì nó sẽ ra undefined, nếu console log ở AfterViewInit thì nó mới ra dữ liệu
   8. Ta đến với @ContentChild  
      Đối với những Template nằm trong <ng-content>, ta không thể dùng ViewChild để access tới Template đó mà phải dùng ContentChild  
        
      Nếu như dùng như thế này  
        
      ta sẽ không bao giờ truy cập vào được thẻ <p> với ViewChild mà phải dùng @ContentChild  
        
        
      Cũng tương tự như ViewChild, để truy cập được viewChild trong Hook ngOnInit thì ta phải cấu hình static = true.