Hướng dẫn ôn tập Angular

Creator: dien.nguyen

Nội dung

- Giới thiệu về Angular
- Các thành phần chính để xây dựng ứng dụng Angular
- Q&A

1. Giới thiệu

Angular là một nền tảng và khuôn khổ để xây dựng các ứng dụng client bằng HTML và TypeScript.

➔ **Angular là 1 Javascript framework.**

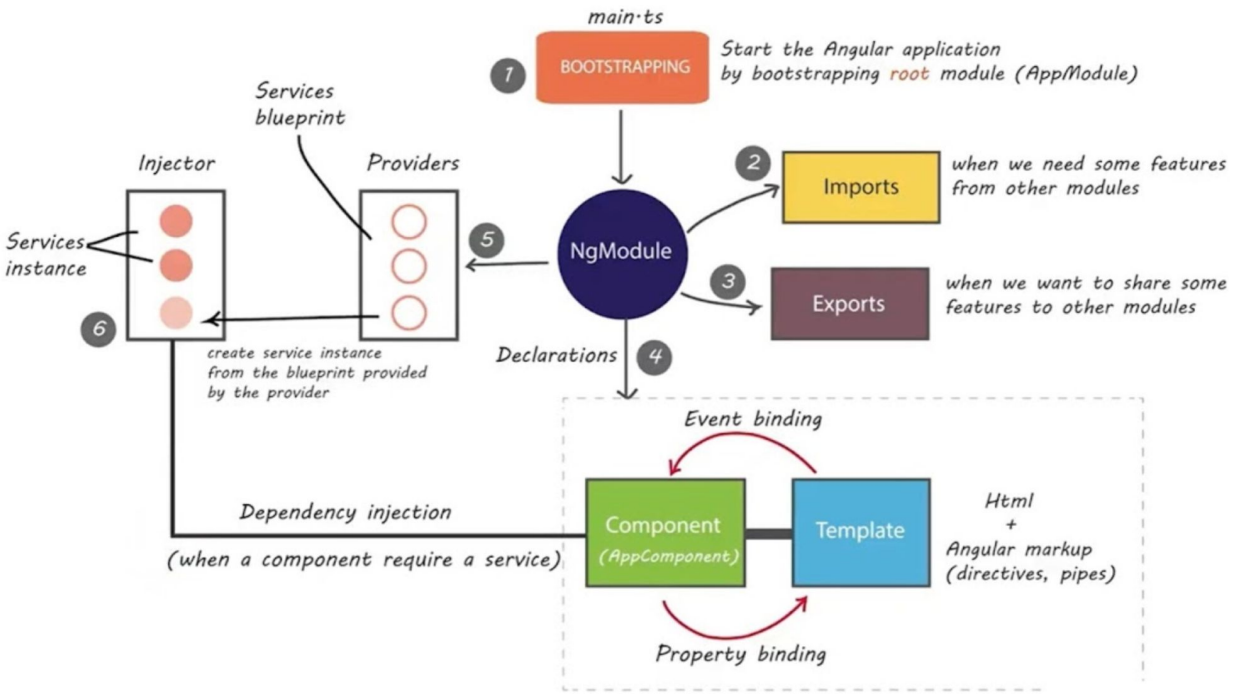
➔ **Angular được sử dụng để xây dựng các ứng dụng client-side (web-drjoy, web-admin)**

➔ **Angular code được viết bằng Typescript**

- Typescript được biên dịch thành Javascript
- Javascript được sử dụng trong HTML

Các thành phần chính để xây dựng ứng dụng Angular

- Modules
- Components
- Data binding
- Pipes
- Directives
- Services
- Routing
- Form

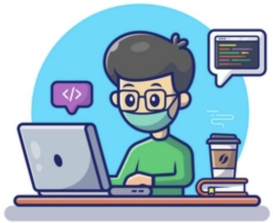


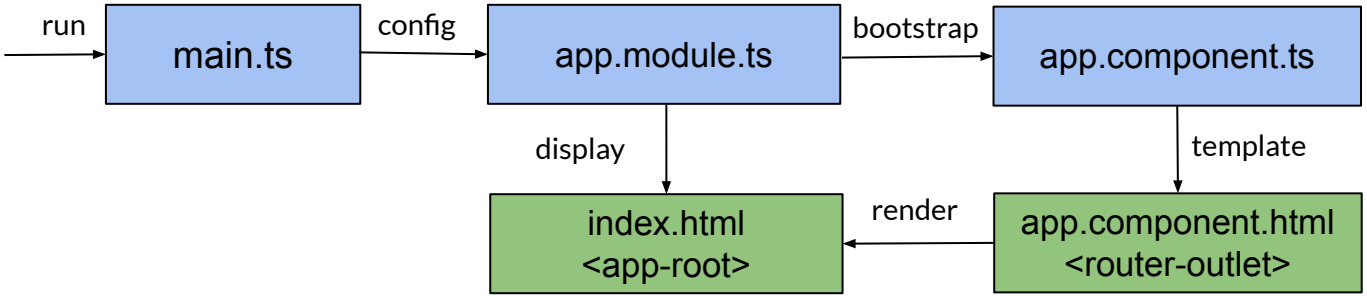
Angular module

- Một ứng dụng thì có nhiều module chức năng
- Một module có thể sử dụng một module khác như FormsModule, RouterModule...
- Các thành phần chính của 1 module là:
  + **Components**: xử lý cho việc hiển thị và control
  + **Directives**: dùng cho databinding
  + **Pipes**: dùng để format dữ liệu
  + **Services**: dùng cho các hoạt động có thể tái sử dụng

Luồng thực thi module

- Ứng dụng thực thi file main.ts
- File main.ts sử dụng cấu hình ứng dụng trong app.module.ts
- File app.module.ts định nghĩa module
- Ứng dụng sẽ được hiển thị trong index.html
- File index.html sử dụng bootstraps root component từ app.component.ts

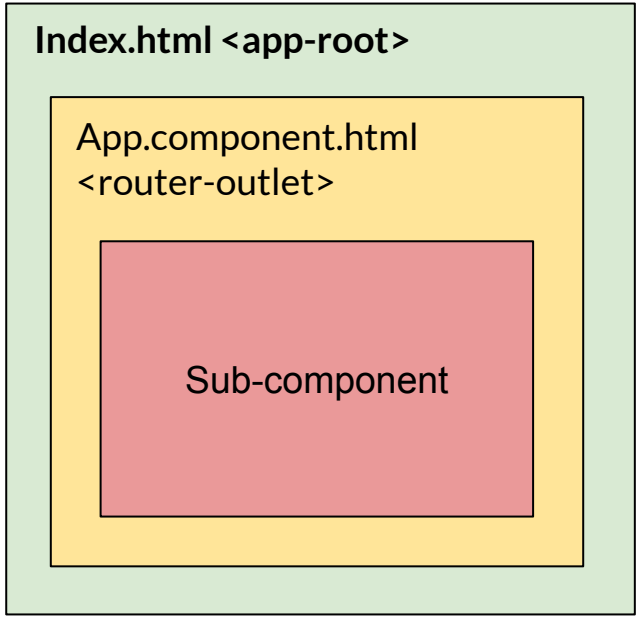




index.html

- src/index.html: Đây là tệp đầu tiên thực thi cùng với main.ts khi tải trang

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="robots" content="noindex,nofollow"/>
  <title></title>
  <base href="/">
  <link id="mask-icon" rel="mask-icon" href="" color="">
</head>
<body>
<app-root></app-root>
</body>
</html>
```



Lưu ý khi xây dựng 1 module mới

- Sử dụng lazy load cho module
- Chỉ import thành phần dùng chung cho tất cả các màn vào Share Module
- Import các thành phần dùng chung của mỗi module vào Share Module con của từng module

```
const routes: Routes = [
  {
    path: 'ch0003',
    loadChildren: 'app/module/ch/ch0003/ch0003.module#Ch0003Module'
  },
  {
    path: '**',
    redirectTo: '/NotFound',
    pathMatch: 'full',
  }
];

@NgModule({
  imports: [
    RouterModule.forChild(routes)
  ],
  declarations: []
})
```

Angular components

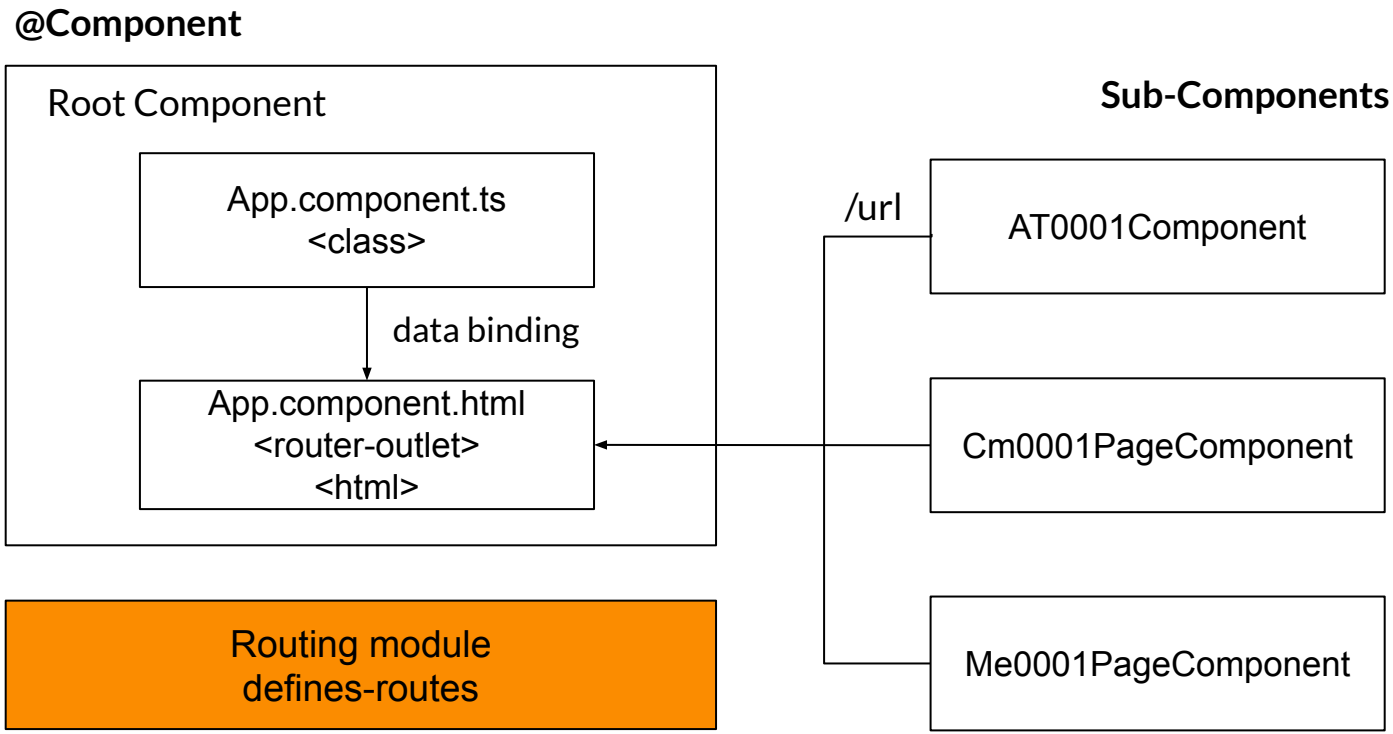
- Một ứng dụng Angular đều có ít nhất 1 component
- Mỗi component xác định lớp chứa dữ liệu ứng dụng và logic và nó được liên kết với mẫu HTML xác định việc hiển thị.
- Component chứa các file sau:
  + **Controller**: *.ts
  + **View**: *.html
  + **Style**: *.scss, .css tùy vào cấu hình của từng ứng dụng
  + **Unit test**: *.spec.ts

Angular components

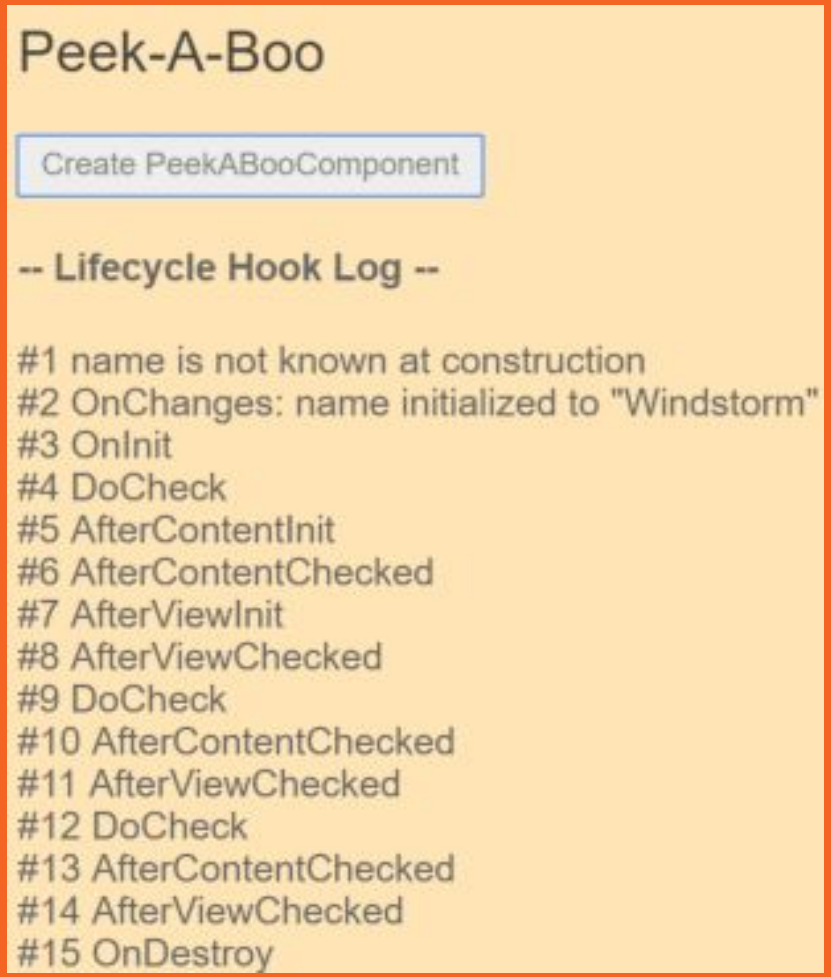
- **@Component**: là trình trang trí thêm siêu dữ liệu vào lớp AppComponent
- **selector:** Angular dùng để nhận biết điền vào <app-root></app-root> hoặc gọi child component vào các component khác
- **templateUrl**: nơi lưu file HTML sử dụng để hiển thị
- **styleUrls**: nơi lưu file SCSS, CSS format giao diện hiển thị

```
import {Component, OnInit} from '@angular/core';

@Component({
  selector: 'app-cm0001-page',
  templateUrl: './cm0001-page.component.html',
  styleUrls: ['./cm0001-page.component.scss']
})
export class Cm0001PageComponent implements OnInit {
  public dataList = [];

  ngOnInit() {
  }
}
```

Luồng thực thi component

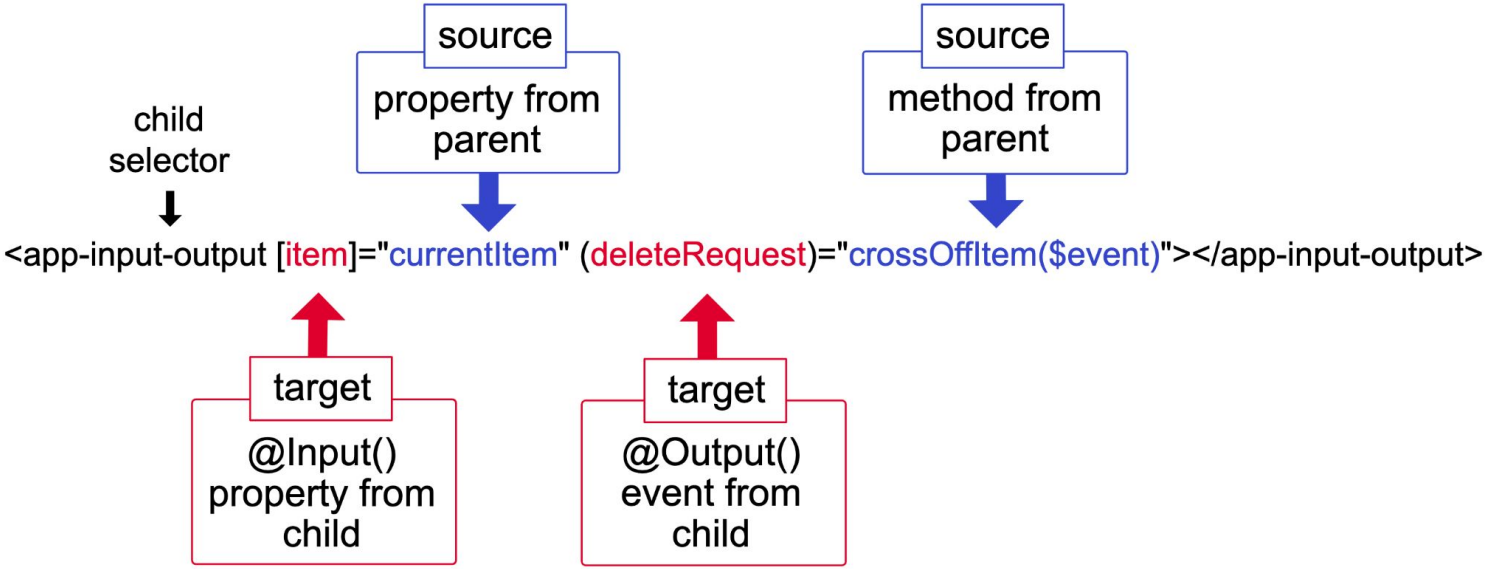


Life cycle trong 1 component



Sharing data between child and parent

- @Input() and @Output() cho phép child component giao tiếp với parent component và ngược lại. @Input() cho phép parent component cập nhật data tới child component. @Output() cho phép child component gửi dữ liệu tới parent



Lưu ý khi xây dựng 1 component

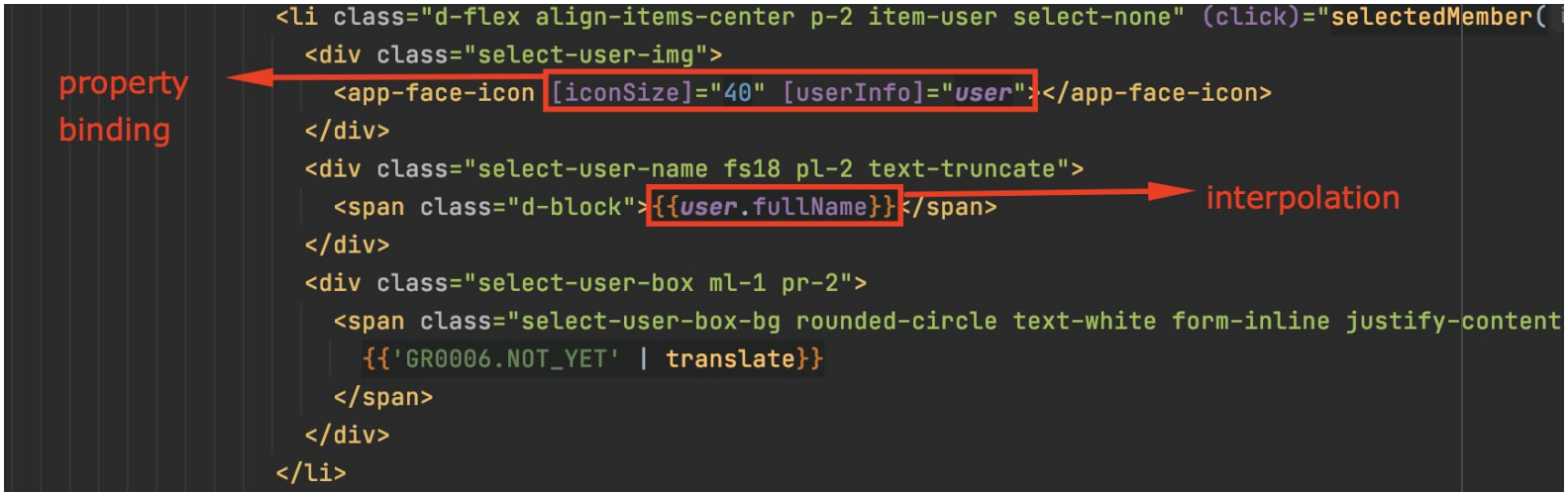
- Sử dụng **ngOnDestroy** để hủy các sự kiện tránh nguy cơ bị rò rỉ bộ nhớ:
  + Unsubscribe from Observables and DOM events.
  + Stop interval timers.
- Không nên gọi function trong file .*html đối với các directive:
  NgClass, NgIf, NgSwitch, NgFor… vì dẫn đến tình trạng 1 sự kiện kích hoạt các function được gọi lại nhiều lần
- Chú ý về coding convention trong file .ts, .html, .scss
- Tên lớp thành phần có bắt có suffix là Component: Cm0001Page**Component**
- **Selector** bắt đầu bằng **app-xxxx** có thể được sửa đổi ở thuộc tính prefix trong file **angular-cli.json**

Data binding

- Hỗ trợ đồng bộ dữ liệu view(*.html) and controller(*.ts)
- Data Binding có thể là One-way, phản ánh sự thay đổi dữ liệu từ controller (.ts) tới view(.html), hoặc Two-way, phản ánh sự thay đổi dữ liệu giữa controller tới view và ngược lại
- Có những binding sau được hỗ trợ bởi Angular:
  + One-way binding:
  → Interpolation - {{attribute-name}}
  → Property Binding - [attribute-name]
  + Event Binding - (event)
  + Two-way binding - [(attribute-name)]

One-way binding

- **Interpolation**: các attributes được định nghĩa trong controller được hiển thị trong html sử dụng {{}}
- **Property binding:** liên kết attribute của controller với thuộc tính DOM của phần tử HTML
- Lưu ý: không gọi function trong one-way binding



Event binding

- Một số các sự kiện dùng cho event binding:
  + **(click)** : called on button click event
  + **(change)** : called on value change event
  + **(keyup)** : called on key up event
  + **(blur)** : called on blur event

```
@Component({
  selector: 'app-popup-webmeeting',
  templateUrl: './popup-webmeeting.component.html',
  styleUrls: ['./popup-webmeeting.component.scss']
})
export class PopupWebmeetingComponent {
  gotoDocument() {
    const $modalSelectFile = $('#modal-select-file');
    if ($modalSelectFile.is(':visible')) {
      $modalSelectFile.modal('hide');
    }
  }
}
```

```
<div class="d-lg-flex mb-3">
  <ul class="fs14 mb-0 pl-4">
    <li>{{'WM0001.SHARE_FILE.LEAD_BLOCK.LEAD_TEXT1' | translate}}</li>
    <li>{{'WM0001.SHARE_FILE.LEAD_BLOCK.LEAD_TEXT2' | translate}}</li>
  </ul>
  <div class="pl-2 ml-auto text-right align-self-center">
    <button type="button" class="btn btn-success" (click)="gotoDocument()">
      {{'WM0001.SHARE_FILE.LEAD_BLOCK.BTN' | translate}}
    </button>
  </div>
</div>
```

Two-way binding

- Trong two-way binding, nếu thay đổi dữ liệu ở view thì controller sẽ được thay đổi. Nếu dữ liệu ở controller thay đổi thì ở view cũng sẽ được thay đổi
- Directive **[(ngModel)]** được Angular hỗ trợ two-way binding
- Kết hợp @Input với @Output bằng cú pháp [(event)]

```
<app-to-do-list [(thingsToDo)]="thingsToDo"></app-to-do-list>
```

```
<div class="form-group">
  <div *ngIf="f.submitted && (model.inviteMessage.length > maxInviteMsg)"
       class="error-message text-danger fs14">
    * {{'MSG.CH.MESSAGE_MAX_LENGTH' | translate}}
  </div>
  <textarea #inviteMessageRef
    name="inviteMessage" id="invite-message" class="form-control invite-message" rows="4"
    appAutoSize [(ngModel)]="model.inviteMessage" [maxLength]="maxInviteMsg + 1">
  </textarea>
```

Angular Pipes

- Pipe được sử dụng để format data, được đặt trong {{...}}
- Pipe có thể được sử dụng để thay đổi dữ liệu từ định dạng này sang định dạng khác
- Ký tự Pipe (|) được sử dụng để áp dụng cho attribute
- Angular hỗ trợ mặc định 1 số Pipe sau: LowercasePipe, UppercasePipe, DatePipe, CurrencyPipe, JsonPipe, PercentPipe, DecimalPipe, SlicePipe, TranslatePipe, AsyncPipe

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({
  name: 'selectMaxText',
  pure: true
})
export class SelectMaxTextPipe implements PipeTransform {
  transform(value: any, maxLength?: number, sign : string  = '...'): any {
    return value && value.length > maxLength ? value.substr( from: 0, maxLength) + sign : value;
  }
}
```

Lưu ý khi xây dựng Pipe

- Có thể tạo các Pipe custom ngoài các Pipe mà angular đã hỗ trợ
- Sử dụng meta data **pure = true** để tăng hiệu năng trên view: chỉ render lại tính toán của Pipe khi giá trị đầu vào thay đổi

```
<tr [ngClass]="{'py-19': status === CsvStatus.CREATING}">
  <td>
    <span [ngbTooltip]="departmentName">
      {{departmentName | selectMaxText: maxLength: 13}}
      {{headerTitle | uppercase}}
    </span>
  </td>
  <td>{{createdAt | timePipe: format: 'YYYY/MM/DD HH:mm'}}</td>
  <td class="text-center">
    <ng-container *ngIf="status === CsvStatus.CREATING">
      {{'AT0029.CREATING_STATUS_TEXT' | translate}}
    </ng-container>
  </td>
</tr>
```

Angular directives

- Directives được sử dụng để thay đổi cấu trúc DOM của 1 trang html
- Angular hỗ trợ nhiều directive như *ngFor và *ngIf
- Có 2 kiểu của directive:
  + **Attribute directives:** Thay đổi hình thức hoặc hành vi của một element, component hoặc một directive khác (NgClass, NgStyle, NgModel)
  + Structural directives: Thay đổi bố cục DOM bằng cách thêm và xóa các phần tử DOM (NgIf, NgFor, NgSwitch)

Attribute directives

Các attribute directives phổ biến nhất như sau:

- **NgClass**—Thêm và xóa một tập hợp các class CSS.
- **NgStyle**—Thêm và xóa một tập hợp các style vào HTML.
- **NgModel**—Thêm liên kết dữ liệu hai chiều vào một phần tử biểu mẫu HTML.

```
<div class="modal-dialog" [ngClass]="dialog.size" role="document">
  <div class="modal-content" [ngStyle]="{'overflow': 'hidden'}" >
    <!--head-->
    <div class="modal-header modal-header-custom">
      <!--title-->
      <ng-container *ngIf="dialog.type">
        <h5 class="modal-title" *ngIf="dialog.title">{{dialog.title}}</h5>
      </ng-container>
      <button type="button" class="close" data-dismiss="modal" aria-label="Close" *ngIf="dialog.clos
        <span aria-hidden="true">×</span>
      </button>
    </div>
  </div>
</div>
```

Structural directives

Structural directive chịu trách nhiệm về bố cục HTML. Chúng định hình lại cấu trúc của DOM, thường bằng cách thêm, xóa và thao tác các element mà chúng được gắn vào.
Các Structural directive tích hợp sẵn phổ biến nhất:
- **NgIf**—Điều kiện tạo hoặc loại bỏ các lượt xem phụ từ mẫu
- **NgFor**—Lặp lại một nút cho từng mục trong danh sách.
- **NgSwitch**—Một tập hợp các lệnh chuyển đổi giữa các chế độ xem thay thế:

```
<div class="maxh-200 content-list">
  <div class="row mb-2 ml-4" *ngFor="let data of listData">
    <div class="custom-control-checkbox">
      <label class="custom-control custom-radio check-box-list">
        <input type="radio" class="custom-control-input" [(ngModel)]="value" [value]="data.value">
        <span class="custom-control-indicator"></span>
        <span *ngIf="data.text" class="custom-control-description">{{data.text | translate}}</span>
      </label>
    </div>
  </div>
</div>
```

Lưu ý khi sử dụng directives

- Không gọi function trong directive, sẽ dẫn đến việc gọi lại và tính toán lại function đấy nhiều lần khi có 1 sự kiện hoặc nhiều sự kiện tương tác
- Có thể tạo custom directive tùy theo logic

```
import {AfterViewInit, Directive, ElementRef} from '@angular/core';

@Directive({
  selector: '[appAutofocusInput]'
})
export class AutoFocusInputDirective implements AfterViewInit {

  constructor(private el: ElementRef) {
  }

  ngAfterViewInit() {
    this.el.nativeElement.focus();
  }
}
```

```
<div class="mb-2 p-0" *ngIf="attachmentFile.length > 0">
  <ng-container *ngFor="let file of attachmentFile.slice(0, 3); let i = index">
    <div class="select-file-info-row align-items-center file-box px-1 mb-1 d-inline-flex" *ngIf="i < 3
      <span class="fs24 pr-1"
            *ngIf="helper.getFileClass(file) && helper.getFileClass(file) !== 'application'">
        <i class="fa text-link" [ngClass]="'fa-file-' + helper.getFileClass(file) + '-o'"
           aria-hidden="true"></i>
      </span>
      <span class="fs24 pr-1"
            *ngIf="!helper.getFileClass(file) || helper.getFileClass(file) === 'application'">
        <i class="fa text-link" [ngClass]="'fa-file-o'" aria-hidden="true"></i>
      </span>
      <span class="pr-1 fs12 text-link text-underline">{{file.name | selectMaxText: maxLength: 40}}</sp
      <span class="file-txt pr-1 file-updated fs12 text-underline">({{file.size | formatSizeUnits}}
        )</span>
      <span
        class="file-txt file-updated fs12 text-underline">{{changeUpdateDate(file.createdAt)}}</span>
    </div><br/>
  </ng-container>
  <div class="d-flex select-file-info-row align-items-center" *ngIf="attachmentFile.length > 3">...</d
</div>
```

Angular Services

- Cấp chức năng cho nhiều components:
+ Thường xuyên được triển khai dưới dạng một singleton
+ Kết nối cơ sở dữ liệu, thực thi các request HTTP, lưu trữ data, v.v.
- Tên file service có cấu trúc: ***.service.ts**
- Một service là một lớp được trang trí bằng **@Injectable,** và được khai báo trong **providers** của **module**

```
import {Injectable} from '@angular/core';
import * as firebase from 'firebase/app';
import Reference = firebase.database.Reference;

@Injectable()
export class GroupService {
  // Variables.
  public screenRelatedReferences = new Map<String, Reference>();
}
```

Truy cập vào một Service

- Import và thêm 1 biến vào hàm constructor của component
- Angular dependency injection sẽ cung cấp cho component truy cập vào instance của class đó
- Lưu ý: không sử dụng các biến private trong view, ảnh hưởng đến chế độ build AOT khi build với mode production

```
import {
  ChangeDetectorRef, Component, ElementRef, Input, NgZone, OnChanges, OnDestroy, OnInit, Sim
} from '@angular/core';
import {GroupService} from '../../services/group.service';

@Component({
  selector: 'app-file-display',
  templateUrl: './file-display.component.html',
  styleUrls: ['./file-display.component.scss'],
  preserveWhitespaces: false
})
export class FileDisplayComponent implements OnInit, OnChanges, OnDestroy {
  @Input('groupId') groupId: string;
  @Input('article') article: any;
  @Input('comment') comment: any;
  @Input() userSession: any;

  constructor(
    private groupService: GroupService,
    private ngZone: NgZone,
    private elementRef: ElementRef,
    private changeDetectorRef: ChangeDetectorRef
  ) {
  }
```

Observables

- Sử dụng thông qua thư viện RxJS.
- Được sử dụng trong nhiều ngôn ngữ và khuôn khổ.
- Được sử dụng nhiều trong Angular.
- Giúp quan sát các xử lý không đồng bộ.

```
putMeetingReport(meetingReport: any): Observable<any> {
  return new Observable( subscribe: observer => {
    this.http.post( url: '/api/meeting/report', JSON.stringify(meetingReport))
      .subscribe( next: (response: Response) => {
        if (response.status === HttpStatus.OK) {
          observer.next();
          observer.complete();
        } else {
          observer.error();
        }
      }, error: (error) => {
        observer.error(error.json());
      });
  });
}
```

```
sendToServer(model, type) {
  this.dialogService.showLoading( flag: true);
  this.meetingService.putMeetingReport(model).subscribe( next: () => {
    this.successfulProcessing(type, msg: 'MEETING_REPORT.MESSAGE.SUCCESS_SUBMIT', applyForApproval: false)
  }, error: () => {
    this.dialogService.showLoading( flag: false);
    this.dialogService.showError('MSG.ERROR');
  });
}
```

Angular Routing

- Hỗ trợ phân tách các tính năng thành module
- Điều hướng đến một component.
- **CanActivate** guard (Kiểm tra truy cập của route).
- **CanActivateChild** guard (Kiểm tra truy cập của child route).
- **CanDeactivate** guard (Yêu cầu quyền loại bỏ các thay đổi chưa được lưu).
- **Resolve** guard (tìm nạp trước dữ liệu cho route).
- Hỗ trợ Lazy loading cho **NgModule**.
- **CanLoad** guard (Kiểm tra tính năng trước khi tải module).

Angular Routing

```
// ルーティング設定
const appRoutes: Routes = [

  // Dr.JOY

  {path: '', component: TopPageComponent, canActivate: [CanActivateValid, CanActivateTermOK]},
  {path: 'logout', component: LogoutPageComponent},
  // Module RS
  {path: 'rs', loadChildren: 'app/module/rs.module#RsModule'},
  {path: 'error', component: ErrorPageComponent},
  {path: 'NotFound', component: NotFoundPageComponent},
  {path: '**', component: NotFoundPageComponent}

];

// 遷移時のログ用サービスの追加、後で外す
for (const route of appRoutes) {
  if (route.canActivate) {
    route.canActivate.push(RouterLogService);
  } else {
    route.canActivate = [ RouterLogService ];
  }
}
```

Angular Form

- Angular cung cấp 2 phương pháp: template driven forms and Reactive forms
- **Template driven** sử dụng các directives để xây dựng forms như **ngModel**, **ngModelGroup** và **ngForm** có sẵn trong **FormsModule**.
- **Reactive forms** hỗ trợ tạo forms sử dụng **FormControl**, **FormGroup** và **FormBuilder** có sẵn trong **ReactiveFormsModule**

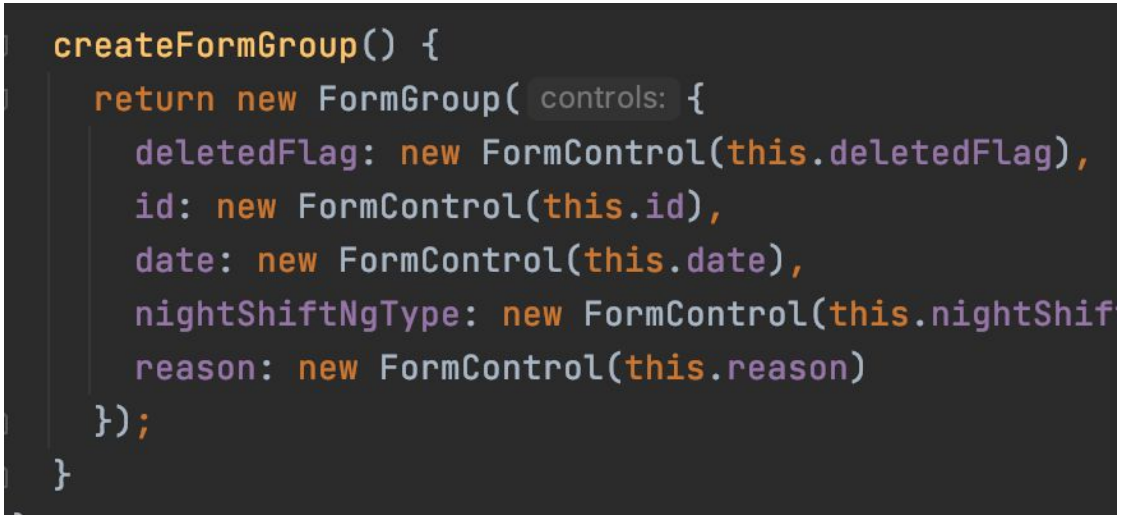
Template-driven forms

- Cần import **FormsModule** vào app.module.ts hoặc shared.module.ts
- Thêm **ngModel** directive và the name attribute, nếu không có name attribute cần thêm **[ngModelOptions]="{standalone: true}"**
- Cần thêm **ngForm** directive tới tag form trong template

```
<form name="form" class="wl-edit-event" (ngSubmit)="submitForm(f)" #f="ngForm" novalidate>
  <div class="row">
    <div class="col-md-6">
      <div class="form-group" [ngClass]="{'has-danger': f.submitted && host.invalid}">
        <label class="label-text" for="host">
          {{'WL.COMMON.LABEL.HOST_CO_HOST' | translate}} <span class="text-danger">※</span>
        </label>
        <div *ngIf="f.submitted && host.errors?.required" class="error-message text-danger fs14">
          ※ {{'WL.COMMON.VALIDATION.HOST_REQUIRED' | translate}}
        </div>
        <input type="text" class="form-control" id="host" required name="host" #host="ngModel"
               [(ngModel)]="model.host" placeholder="{{'WL.COMMON.PLACE_HOLDER.HOST' | translate}}">
      </div>
```

Reactive forms

- Cần import **ReactiveFormsModule** vào app.module.ts hoặc shared.module.ts
- Sử dụng sử dụng **FormControl**, **FormGroup** và **FormBuilder** để tạo Form
- Sử dụng Validators để validate cho Form



Q & A


Thanks you!