**Module**: Angular is in modular in nature.is an feature area in app, like user and admin module, can be imported and exported.one root module as appmodule.

**uses**: better code management, code reuse, code maintenance, performance (can eager or lazy loading module)

component + service

**Component**: is a portion of the view in browser. ex: side bar, navigation and main content area.one root component as appcomponent. all component will be nested in it.

html+ class+ metadata (information like decorator)

**service**: business logic

angular app: one or more modules.

**module**: one or more components and services. module interact and ultimately render the view in the browser.

**components**:html+class

**selector**: [selectorname], ".selectorname", "selectorname"

then <selectorname></selectorname>, <div class="selectorname"></div> and <div selectorname></div>

**attribute**: defined by HTML

**Properties**: defined by DOM

Attributes initialize DOM properties and then they are done. attribute values can't be change once they are initialized.

Property values however can change.

The Document Object Model (**DOM**) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated. In the DOM specification, the term "document" is used in the broad sense - increasingly, XML is being used as a way of representing many kinds of information that may be stored in diverse systems, and much of this would traditionally be seen as data rather than as documents. Nevertheless, XML presents this data as documents, and the DOM may be used to manage this data.

**STYLES Applying:**

<div class=”cssclassname”></div> --attribute

<div [class]=”cssclassname”></div> --property

<div [class. cssclassname]=”value true or false or variable it has true or false”></div> --single condition can apply

<div [ngClass]=”object”></div> --multiple conditions (can apply two or more css class can apply)

Var object={

‘text-success’:true,

‘text-danger’:false,

‘text-special’:true (we can use Boolean type variable)

}

<div [style.color]=”’Orange’”></div>

<div [style.color]=”nameOfVariableContainColor”></div>

<div [ngStyle]=”object”></div> --check just above object

<div [style.color]=”isTrue?’Red’:’Blue’”></div>

**Directives:**

**Structural directives:** Are using to Add/Remove html elements from DOM

1. \*ngIf
2. \*ngSwitch

Above 2 are conditional directives.

1. \*ngFor

**EX:**

1. <div \*ngIf=”true”> </div> --it will not render div tag if it’s false (different from display:none property bcz display none will render but it hides the element)
2. <div \*ngIf=”boolValue; else elseBlock”> </div>

<ng-template #elseBlock> -- elseBlock is reference of this element

<h2> else condition content here</h2>

</ng-template>

1. <div \*ngIf=”boolValue; then thenBlock; else elseBlock”> </div>

<ng-template #thenBlock>

<h2> if condition content here</h2>

</ng-template>

<ng-template #elseBlock> -- elseBlock is reference of this element

<h2> else condition content here</h2>

</ng-template>

1. <div [ngSwitch=”color”]> --public color=”Red”;

<div \*ngSwitchCase=”’Red’”>Red Picked </div>

<div \*ngSwitchCase=”’Yellow’”>Yellow Picked</div>

<div \*ngSwitchCase=”’Blue’”>Blue Picked </div>

<div \*ngSwitchDefault>Pick Again</div>

</div>

1. <div \*ngFor=”let color of colors;let i=index”> --or index as i

<h2>{{i}} {{color}}</h2>

</div>

<div \*ngFor=”let color of colors; first as f”>

<h2>{{f}} {{color}}</h2> --return f return True or false ,True if it is first element

<div>

Like above we have ‘last’ ,’odd’,’even’

**TrackBy:**

Foreach do DOM manipulations for simple change in a list, it causes performance issue

Render only new records based on the trackByFn function,so that it won’t re render whole DOM.

public colors:any=["Red","Yellow","Blue","Pink","Gray","Orange"]

trackBy: trackByFn

public trackByFn(colors:any):string{

return colors;

}

public refresh(){

this.colors=["Red","Yellow","Blue","Pink","Gray","Orange","New Color"]

}

<div \*ngFor="let color of colors;trackBy: trackByFn">

{{color}}

</div>

Component Interaction (Child-Parent)

Input and Output properties must declare in child component only.

Input property is to get values from parent component to child component.

Output property is to push values from child component to parent component.

**Parent to Child:**

Child Component Code:

@Input() public ChildComponentProperty:string;

Or

inputs:['ChildComponentProperty']

or

@Input(‘ChildComponentProperty’) aliasNameHere public:string;

<p>

{{ChildComponentProperty}}

</p>

Parent Component Code where we have been using child component:

<app-users-list [ChildComponentProperty]="'valueFromApp'"></app-users-list>

**Child to Parent:**

Child Component Code:

@Output() public childEvent=new EventEmitter();

this.childEvent.emit("Value from Child Component as Event VAlue");

Or

outputs:['childEvent']

public childEvent=new EventEmitter<string>();

this.childEvent.emit("Value from Child Component as Event VAlue");

or

@Output('childEvent') public childEventAlias=new EventEmitter();

this.childEventAlias.emit("Value from Child Component as Event VAlue");

Parent Component Code where we have been using child component:

public valueFromChild:string="";

<app-users-list (childEvent)="valueFromChild=$event" [ChildComponentProperty]="'valueFromApp'"></app-users-list>

{{valueFromChild}}

**Pipes:**

* lowercase
* uppercase
* titlecase
* currency
* slice: 3:5 (index 3 to 5)
* json
* number:’1.2-3’ (minimum length one before decimal points and minimum 2 and max 3 length after decimal point)
* percent
* currency
* currency:’GBP’ --SYMBOL
* currency:’GBP’:’code’ –as GBP
* date
* date:’short’
* date:’shortDate’
* date:’shortTime’
* date:’medium’
* date:’mediumDate’
* date:’ mediumTime’

**Services:**

* Code/data reusability and single responsibility achievement,if we use data in component directly it violates single responsibility principle.
* Component responsibility is to manage view only.

**Uses:**

* Share data
* Implement application logic
* External interaction

**Dependency injection:**

* Code without DI -Drawbacks

We must create all instances of classes where we are using them.

The code of those classes changes then we must change in all places.

* DI as a design pattern

Passing instances as parameter values (as dependencies)

**Drawback:** difficult to manages when depend ices grow, and those decencies again depend on another.

* DI as a framework

DI is a coding pattern in which a class receives its dependencies from external sources rather than creating them itself.

**HTTP, Observables and RxJS:**

* HTTP Get request from EmpService.
* Receive the observables and cast it into an employee array.
* Subscribe to the observable from EmpList and EmpDetail.
* Assign the employee array to the local variable.

**RXJS:**

* Reactive extension for JavaScript.
* External library to work with Observables.

**Service:**

Import {Observable} from ‘rxjs/Observable’;

Import ‘rxjs/add/operators/catch’;

Import ‘rxjs/add/ Observable /throw;

**----http:HttpClient**

Return this.http.get<IEmplyee[]>(this.url).catch(this.errorHandler);

errorHandler(error:HttpErrorResponse){

return Observable.throw(error.message || “Server Error”)

* }

**Component:**

this. \_employeeService.getEmployees().subscribe(data=>this.employees=data,error=>this.errorMsg=error);

**Routing:**

import { NgModule } from '@angular/core';

  import { Routes, RouterModule } from '@angular/router';

  import { DepartmentListComponent } from './department-list/department-list.component';

  import { DepartmentDetailComponent } from './department-detail/department-detail.component';

  import { EmployeeListComponent } from './employee-list/employee-list.component';

  import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

  import { DepartmentOverviewComponent } from './department-overview/department-overview.component';

  import { DepartmentContactComponent } from './department-contact/department-contact.component';

  const routes: Routes = [

{ path: '', redirectTo: '/departments', pathMatch: 'full' },

{ path: 'departments', component: DepartmentListComponent },

{

path: 'departments/:id',

component: DepartmentDetailComponent,

children: [

{ path: 'overview', component: DepartmentOverviewComponent},

{ path: 'contact', component: DepartmentContactComponent}

]

},

{ path: 'employees', component: EmployeeListComponent },

{ path: '\*\*', component: PageNotFoundComponent }

  ];

  @NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

  })

  export class AppRoutingModule { }

  export const routingComponents = [DepartmentListComponent,

  DepartmentDetailComponent,

  EmployeeListComponent,

  PageNotFoundComponent,

  DepartmentOverviewComponent,

  DepartmentContactComponent]

onSelect(department) {

this.router.navigate(['/departments', department.id]);

this.router.navigate([department.id], { relativeTo: this.route });

}

goNext() {

let nextId = this.departmentId + 1;

this.router.navigate(['/departments', nextId]);

}

  gotoDepartments() { //Optional Parameters

let selectedId = this.departmentId ? this.departmentId : null;

this.router.navigate(['/departments', {id: selectedId}]);

this.router.navigate(['../', { id: selectedId }], { relativeTo: this.route });

  }

  showOverview(){

   this.router.navigate(['overview'], { relativeTo: this.route });

  }

  showContact(){

   this.router.navigate(['contact'], { relativeTo: this.route });

  }

ngOnInit() {

let id = parseInt(this.route.snapshot.paramMap.get('id'));

this.route.paramMap.subscribe((params: ParamMap) => {

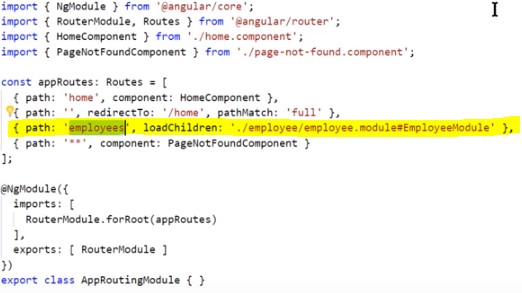
let id = parseInt(params.get('id'));

this.departmentId = id;

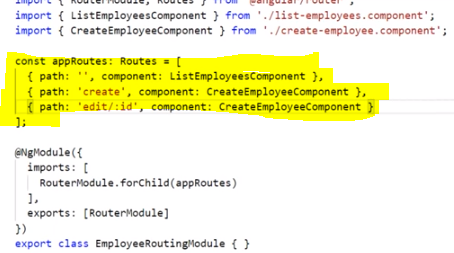
});

}

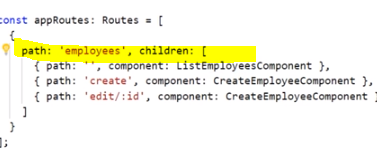
**Lazy Loading:**



**Child Routes:**



**For reference how, child routes can be configure:**

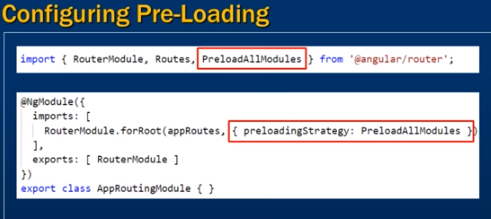


**Module Loading Strategies:**

* + Eager Loading
  + Lazy Loading
  + Preloading

**Preloading Configure:**

* + First Eager loading module is loads then lazy loading modules will be downloaded silently in background without waiting for visit lazy loading route visit.



Custom Preloading Strategy:

