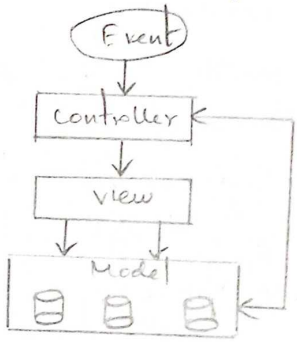
WEB UI Design

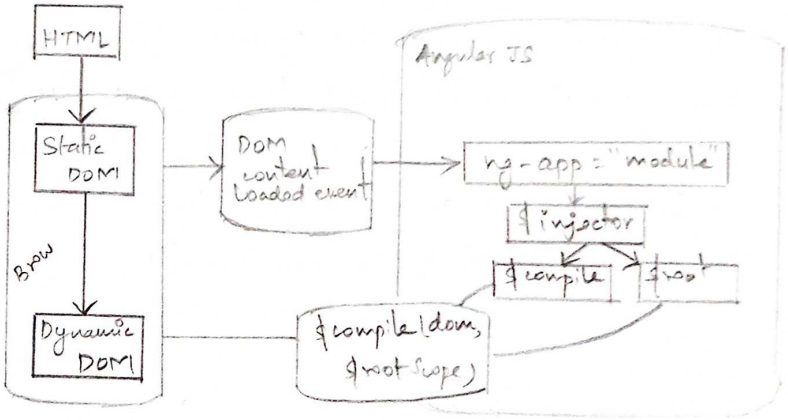
Angular JS architecture

S.N. Karthick
III CSE-B
21BCS109

* Angular JS is an open-source Javascript framework that is used to build web application. It can be freely used, changed & shared by anyone. Efficient in building single page applications & line of business application



MVC architecture



* MVC stands for Model View Controller. It is a software design pattern followed in web applications. It isolates the application logic from the UI layer & supports separation of the concerns.

MVC is made up of 3 parts,

1) Model: It is responsible for managing application data, it responds to the requests from view and to the instructions from controller to update itself.

2) View: It is responsible for displaying all data or only a portion of data to the users. It also specifies the data in a particular format triggered by the controller's decision to present the data. They're script based template systems such as JSP, ASP, PHP & very easy to integrate with AJAX technology.

3) Controller: It is responsible to control the relation between models and views. It responds to user input and performs interactions on the data model objects. The controller receives input, validates it and then performs business operations that modify the state of the data model.

Key Features of Angular JS :

* MVC architecture makes the web application more robust and manageable.

Data model binding: To bind data to HTML components, angular JS allows provides easy javascript snippets & respective directives in the HTML objects.

Less code : To perform DOM traditionally, it requires lot of javascript code. code size is reduced by using Angular JS

karma framework : Angular JS allows us to perform Unit Testing by using the karma framework., Google developed karma framework for Angular JS to build reliable tests.

Pros :

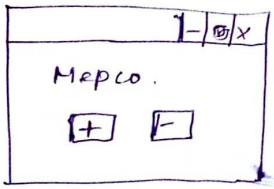
* It is open source
* It has 2 way binding feature.
* It also manages routing
* It provides super powers for the HTML Components & allows unit testing.

.ts file

```
{ mystye : any = { color : red,
                   font-size' : '10 px'
                   3; 'background-color' : 'yellow'
}
```

(or) → size : number = 20 ;

Example



+ while clicking font size should be increase

- while clicking font size should be decrease.

mycomponent . html .

```
<div [style . font-size . Px] = "size">
    Mepco
</div>
<input type = "button" (click) = "incr()"
                       value = "+"/>

<input type - "button" (click) = "decr()"
                       value = "-"/>
```

mycomponent . ts

```
export class Mycomponent
{
    size : number = 20;
    incr() {
        this . size += 5;
    }
```

```
    decr() {
        this.size -= 5;
    }
}
```

Class binding

mycomponent.css

```
.success {
    color: green;
}
.error {
    color: red;
}
```

mycomponent.html

```
<div [class] = "myclass">
    Mepco
</div>
```

```
<div
[class]
= "flag ? 'success' : 'error' "
    Mepco
</div>
```

mycomponent.ts

```
export class MyComponent
{
    myclass : string = "success";
}
```

(or) flag : boolean = true;

Directives

* Structural Directive
* Attribute Directive

} built-in custom directive

Structural Directives

*ngIf

*ngFor

*ngSwitch

attribute Directives

[ngstyle]

[ngclass]

mycomponent.html.

```
<div *ngIf = "show">
    My college is Mepco
</div> <input type = "button" value = "Show"
        (click) = "toggle()" />
```

my component.ts

```
export class MyComponent {
    show : boolean = true (visible)
                     false (not visible)
                     div block

    toggle()
    {   this.show = !this.show;
    }
}
```

Two-way Binding

component.html.

```
<h1> two way binding </h1>
<input type = "text"
    [(ngmodel)] = "college" />
{{college}}
```

component.ts.

```
export class MyComponent {
    college : string = "";
}
```

Attribute binding

something which is not handled by DOM is can't handle by Property binding it will be handled by attribute binding.

Element has Key, value pairs

property - handled by DOM (Src)

attibute - not handled DOM (colspan)

component.html.

```
<table border="2">
    <tr [attr.colspan]="col">
    </tr>
    <tr>
        <td> 1 </td>
        <td> 2 </td>
        <td> 3 </td>
    </tr>
    <tr>
        <td> 4 </td>
        <td> 5 </td>
        <td> 6 </td>
    </tr>
</table>
```

Component.ts

```
export class MyComponent
{
    col:number = 3;
}
```

~~Com~~

Directives

.html

```
<div *ngIf = "show">
    Hello jeeva
</div>
```

~~with~~ visible

.ts

```
{
    show : boolean = true;
}
```

false (not visible

i) .html

```
<input type = "button" value = "show/hide"
        (click) = "change()"/>
```

.ts

```
{
    change()
    {
        this.show = ! this.show;
    }
}
```

ii) .html

```
<input type = "button" value = "click"
        (click) = "change($event.target)"/>
```

.ts

```
{
    change (ele : any)
    {
        if (ele.show)
        {
            ele.value = "hide";
        }
        else
        {
            ele.value = "show";
        }
    }
}
```

*ngFor

.ts

```
movielist = [ "AAA", "BBB", "CCC", "DDD" ];
hero : string = "rajni";
herolist = [ "rajni", "kamal", "vijay", "jeeva" ]
```

i) . html

```
<ol>
    <li *ngFor = "let m of
                    movielist,
                let i=index">
            {{m}} -- {{i}} </li>
</ol>

<Select>
    <option *ngFor = "let m of
                movielist"
                value = "{{m}} ">
                {{m}} </option>
</select>.
```

*ngswitch

ii)

```
<div [ngSwitch]= "hero" >
    <div *ngswitch case = "rajni">
            Rajni
    </div>
    <div *ngswitchcase = "kamal">
            kamal Hasan
    </div>
    <div *ngswitch case = "vijay">
            vijay
    </div>
    <div *ngswitch Default>
            jeeva is real hero
    </div>
</div>
```

iii)

```
<select [(ngmodel)] = "hero" >
    <option *ngFor = "let h of herolist"
            value = {{h}} " > {{h}} </option>
</select>
```

*ng Style

• html

```
<h1> Style Directive</h1>
<div [ngStyle] = "mystyle" >
    Hello this is style directive
</div>
```

• ts

```
export class MyComponent {
    mystyle : any = { color : "red",
                      font-size : "20px" }
}
```

(or)

```
<div [ngStyle] = "status ? mystyle :
                  urstyle" >
    style directive
</div>
```

```
urstyle : any = {color : "green"};
Status : boolean = true ; <!-- red -->
                   false ; <!-- green -->
```

Attribute Directives

To create a directive
ng g d myhighlight

ng g c custom

app.component

<app-custom>
</app-custom>

~~component.html~~

```
@directive({
  selector: '[myhighlight]'
})
export class mycomponent
{
  constructor (private
  el: ElementRef) {
  }
  @HostListener ('mouse over')
```

Pipe and custom Pipe

ng g c pipe-comp

• html

* (Built in Pipes in angular)

To create a custom pipe.

ng g p mylen.

Angular Directives

Attribute Directives	Structural Directives
*ngFor	[ngStyle]
*ngIf	[ngClass]
*ngSwitch	

*ngIf

* Based on Boolean Expression: it render the content of element

Syntax:

```
<ele *ngIf = "boolean-exp">
        Content of Element
</ele>
```

Eg.

app.component.html.

```
<app-demo> </app-demo>
<router-outlet> </router-outlet>
```

demo.component.html.

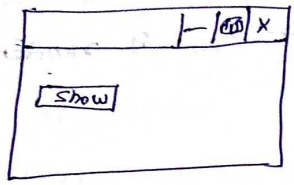
```
<div *ngIf = "show">
        Content about college
</div>

<input type = "button"  value = "clickme"
        (click) = "toggle()"/>
```

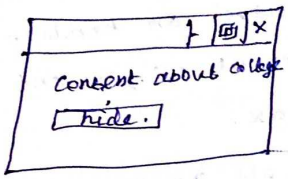
→ toggle(event.target)

demo.component.ts

```
selector: "app-demo"
export class DemoComponent {
        show : boolean = true;
        toggle() {
                this.show = !this.show;
        }
}
```

```
toggle(ele) {
        this.show = !this.show.
        ele.value = this.show ?
                'Hide' : 'show';
}
```





***ngFor**

* it is the structural directive used to render iterative contents based on arrays.

Syntax

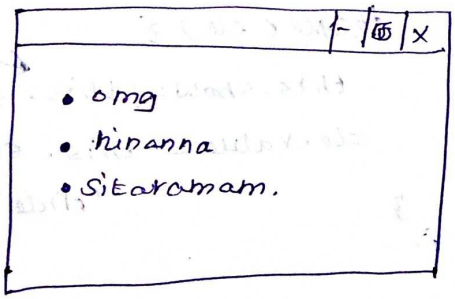
```
<ele *ngFor="iterative exp">{{}} </ele>
```

demo.component.ts

```
export class DemoComponent
{
    mlist : string[] = ["omg", "hinanna",
                        "sitaramam"]

}
```

demo.component.html

```
<ul>
    <li *ngFor="let m of mlist">
        {{m}}
    </li>
<ul>.
```



*ngSwitch

*Structural directive used to render content based on multi condition.

Syntax

demo.component.html.

```
<div [ngSwitch] = "ch">
    <div *ngSwitch case = "Rajni">
        Rajni
    </div>
    <div *ngSwitch case = "kamal">
        kamal
    </div>
    <div *ngSwitch Default >
        vijay
    </div>
</div>
```

demo.component.ts.

```
export class DemoComponent
{
    ch : string = "Rajni"
    heroList : string [] = [ "rajni",
                "kamal", "vijay"]
}
```

demo.component.html.

```
<select [[ngmodel]] = "ch">
    <option *ngFor = "let h of
                heroList " > {{h3}} </option>
</Select>.
```

Custom directives

Directive name : My highlighter

purpose : Change the background as red on mouseover. and normal color on mouseout

command : ng generate directive Myhighlighter

(it generates two files and one updation)

Myhighlighter.Directive.ts

```
@Directive {
    selector : [ 'Myhighlighter' ]
}
export class Myhighlighter
{
    constructor ( private ele : ElementRef
                                ...)
    { }

}
@HostListener ( 'mouseover' )
{
    this. ele.
```

Custom Validator.

ng g d age-validate

age-validate.ts.

remove "app" in selector

```
export class AgeValidateDirective
        implements Validators
{
    constructor ( ) {
```

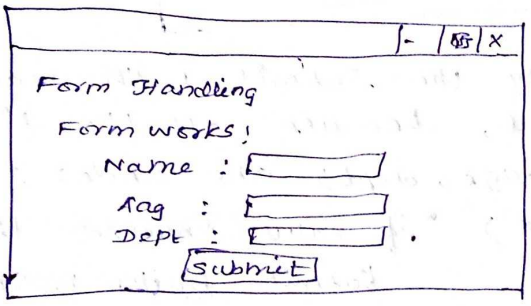
```
Validate (ctrl: Form
            Control){
    let x: number = ctrl.
                    value;
    if (x % 2 == 1)
        return {noteven. true}
    else
        null ;
}
```

Form Handling

ng g c myform

myform.html.

```
<form #form="ngform">
    <div> Name <input type="text" [(ngmodel)]="name" name="name" />
    <div> Age <input type="text" [(ngmodel)]="age" name="age" />
    <div> Dept <input type="text" name="dept" [(ngmodel)]="dept" />
    <div><input type="submit" value="submit" />
    </div>
</form>
```

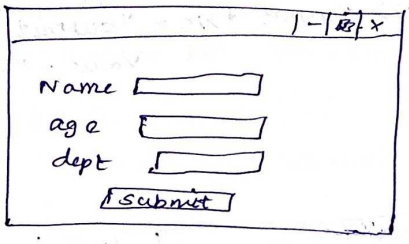


myform.ts.

```
export class MyFormComponent
{
    name : string = "";
    age : number = 99;
    dept : string = "";
}
```

```
<form #form="ngform">
<div> Name : <input type="text" [(ngmodel)]="name" required
    name="name" #name="ngModel" />
</div>
<div> age : <input type="text" [(ngmodel)]="age" required
    name="age" #age="ngmodel" />
</div>
```

```
<div> dept : <input name="dept" type="text"
required [ngmodel]="dept," #dept="ngmodel"/>
</div>

<div> <input type="submit" [disabled]="form.
invalid"
value="submit" /></div>

</form>
```



(initially the submit button was disabled, because all the fields (name, age, dept) are under required field) "if value Entered the submit Button Become Enabled"

Routing

app.component.html

```
<h1> This is Mine </h1>

<h2>
  || <a href="/ch1">.ch1 </a>
  || <a href="/ch2"> ch2</a>
  || <a href="ch3"> ch3</a>
  || <a href="home"> ch4 </a>
  || <a href="form"> form </a>
</h2>
```

app.routing.ts.

```
const routes : Routes = [
  { path="ch1", component : ch1 component},
  { path = "ch2", component : ch2 component },
  { path = "ch3", component : ch3 component},
  { path = "home", component : home component},
  { path = "form", component : form component}
]
```

CRUD Applications.

ng g c curd.

app.component.html

```
<h1> My page <h1>
<h2>
  <a href = "/crud"> crud </a>
</h2>
```

crud.component.html.

```
<h1> CRUD DEMO </h2>
```

To create class

ng g class movies.

<u>movies</u>

```
export class Movies
{
  title : string = "";
  hero : string = "";
  budget : number = 0;
  dir : string = "";
```

```
Constructor( a:string ,b :string, c:number,
                                d:string)
{
        this. title = a;
        this. hero = b;
        this. budget = c;
        this. dir = d;
  }
}
```

crud. component. ts.

```
export  class  Crud Component
{
```

Session Management

- ng g c page
- ng g c dashboard
- ng g c login
- ng g c layout
- ng g c contact

routing. module. ts

```
const routes : Routes = [
  {Path : '', redirect To : 'login, pathMatch : 'full'},
  { Path : 'login', component : Login component}
  { Path : '', component : Layout Component,
     children : [
       {Path : 'dashboard', component : Dashboard
                                        Component},
       { Path: 'home', component : Home Component},
       { Path : 'contact', component :
                                Contact Component }
     ]}
]
```

login.component.html

```
<h1> login Page </h1>
uname: <input type="text" [(ngmodel)] = "uname" name="uname" /> <br/>
Pwd: <input type="text" [(ngmodel)] = "uname" name="Pwd" /> <br/>
<input type="submit" value="submit" /> (click) = "auth()"
{{ logMsg }}
```

login.component.ts.

```
export class LoginComponent
{
    logmsg : string = '';
    uname : string = '';
    user : string = '';
    Pwd : string = '';
    Constructor ( private router: Router) {}
    auth ()
    {
        if(this.uname ==this.pwd)
        {
            local storage.setItem ('user', this.uname
            this.router.navigateByUrl ("dashboard
        }
        else
        {
            this.logMsg = "Login failure";
        }
    }
```

layout.component.html

```
<h1> My Title Page </h1>
<h3>
  || <a href = "/dashboard"> Dashboard </a>
  || <a href = "/home"> home </a>
  || <a href = "/contact"> contact </a>
  || <a ~~href = "/logout"~~ > log out </a>
<h3>
                (click) = "logout()"
</app-outlet>
```

layout.component.ts

```
user : string = '';
constructor (private router: Router) {
    let u : any = localStorage.getItem('user')
    this.user = u;
    if (u == null || u == '')
}           this.router.navigateUrl('login')
logout()
{   localStorage.removeItem('user')
    this.router.navigateByUrl('login')
}
```

hello.component.html

```
<input type = "button"  value = "Say hi"
        (click) = "sayHai()" />
```

hello.component.ts

```
{ constructor(private utly : nodeUtilityservice) { }
  msg : string = '';
  sayHai()
  {
      this.utly.sayHai.subscribe(data=>)
      if(this.status)
      { this.msg = data.message ;
      }
      else
      { this.msg = "Error";
      }
  }
}
```

* Modify Node
* Modify utility-service
* handle in angular