Angular

Component

- Template
  - View
  - HTML
- Class
  - Code that supports the View
  - TypeScript
  - Contains
    - Data
    - Methods
- Metadata
  - Information
  - Decorator

Interpolation

– way to bind data from class to template

```
template: `<h2>
                Header Component
                </h2>
                <p>Welcome {{name}}</p>
`,


export class HeaderComponent implements OnInit {
name="Baljit";
constructor(){}
```

What else?

```
<p>{{2+2}}</p>
<p>{{"Welcome "+ name}}</p>
<h2>Javascript properties and methods</h2>
<p>{{name.length}}</p>
<p>{{name.toUpperCase()}}</p>
<p>{{fun1()}}</p>

In class
      fun1(){
            return "Hello "+this.name;
      }
```

Property Binding

- Attribute vs Property

- `<input type="text" value="Baljit">`

- Console
  - `$0.getAttribute('value')` →Attribute (defined by HTML)
  - `$0.value;` →Property (Defined by DOM)

Property Binding

```
template: `<h2>
                Header Component
                </h2>
                <input [id]=“myId” type="text"
value="baljit">
  `,

export class HeaderComponent implements OnInit {
myId=“testId”;
```

Why Property Binding?

- Interpolation can work only string values

- `<input disabled type="text" value="baljit">`

- Solution
- `<input [disabled]=isDisabled type="text" value="baljit">`

```
export class HeaderComponent implements OnInit {
isDisabled=“false”;
```

Class binding

```
template:`
        <h2>Welcome</h2>
`,
styles:[`
        .text-success{
                color:green}
        .text-danger{
                color:red}
        .text-special{
                font-style:italic}
`]
```

Class binding

- Normal

template:`

```
        <h2 class=text-success>Welcome</h2>
```

`,

- Class Binding

```
<h2 [class]="setClass">Header Component</h2>
```

```
export class HeaderComponent implements OnInit {
setClass="text-success";
```

Conditional class binding

```
<h2 [class.text-danger]="error">
                Header Component
                </h2>
```

```
export class HeaderComponent implements OnInit {
error=true;
```

ngClass directive

- Directive
  - Custom HTML attribute

```
<h2 [ngClass]="classObject">Header Component </h2>
```

```
error=true;
classObject={
  "text-success":this.error,
  "text-special":this.error,
}
```

Style Binding

- To apply inline styles to HTML elements

- `<h2 [style.color]="'orange'">Style Binding</h2>`

- `<h2 [style.color]="error?'orange':'red'">Style Binding</h2>`

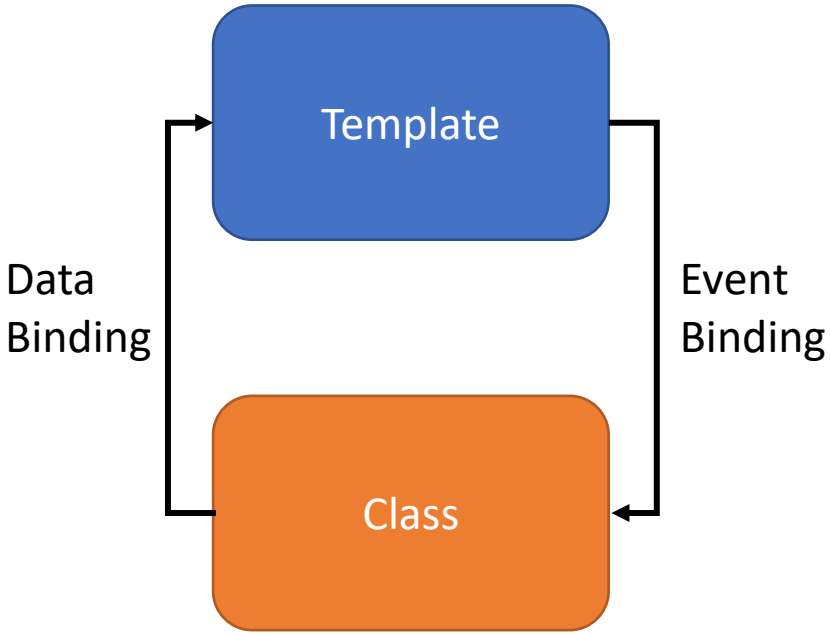
- `<h2 [style.color]="paint">Style Binding</h2>`
  - `paint="blue";`
-

ngStyle Directive

- `<h2 [ngStyle]="styleDir">Style Binding</h2>`

```
styleDir={
   color:"green",
   fontStyle:"italic"
}
```

Event Binding



```
<button (click)="onClick()">Welcome</button>
```

```
onClick(){
  console.log("Welcome to Angular");
 }
```

```
<button (click)="onClick()">Welcome</button>
{{welcome}}
```

```
welcome="";
onClick(){
  console.log("Welcome to Angular");
  this.welcome="You clicked";
}
```

```
<button (click)="greeting='Welcome Class'">Welcome</button>
{{greeting}}
```

greeting=“”;

Template Reference Variable

```
<input #myInput type="text">
<button (click)="logMessage(myInput.value)">Log</button>


logMessage(value){
        console.log(value);
}
```

Two way binding

<input [(ngmodel)]=“name” type=“text”>

{{name}}

name=“”;

Two way binding

- Open app.module.ts
- Import forms module
  - import { FormsModule } from ‘@angular/forms’;
- Add to imports array
  - BrowserModule,
  - FormsModule

Structural Directive

- ngIf
- ngSwitch
- ngFor

nglf

<h2 *ngIf=“display”>Hello World</h2>

display=true;

If-else

```
<h2 *ngIf="display; else elseBlock">Hello World</h2>
<ng-template #elseBlock>
<h2 *ngIf="display">Else Bye World</h2>
</ng-template>
```

```
display=true;
```

```
<div *ngIf=“display; then ifBlock; else elseblock”></div>

<ng-template #ifBlock>
<h2>Hello World</h2>
<ng-template>

<ng-template #elseBlock>
<h2 Else Bye World</h2>
</ng-template>

display=true;
```

ngSwitch Directive

```
<div [ngSwitch]=“<property>">
        <div *ngSwitchCase=“’<property-value1>'"></div>
        <div *ngSwitchCase="'<property-value2>'"></div>
        <div *ngSwitchCase="'<property-valuen>'"></div>
       <div *ngSwitchDefault>Choose Again</div>
</div>
```

ngSwitch Directive

```
<div [ngSwitch]="color1">
          <div *ngSwitchCase="'red'">You choose red</div>
          <div *ngSwitchCase="'green'">You choose green</div>
          <div *ngSwitchCase="'blue'">You choose blue</div>
         <div *ngSwitchDefault>Choose Again</div>
</div>


color1=“red”;
```

ngFor Directive

```
<div *ngFor="let value of array">
            <h2>{{value}}</h2>
</div>
```

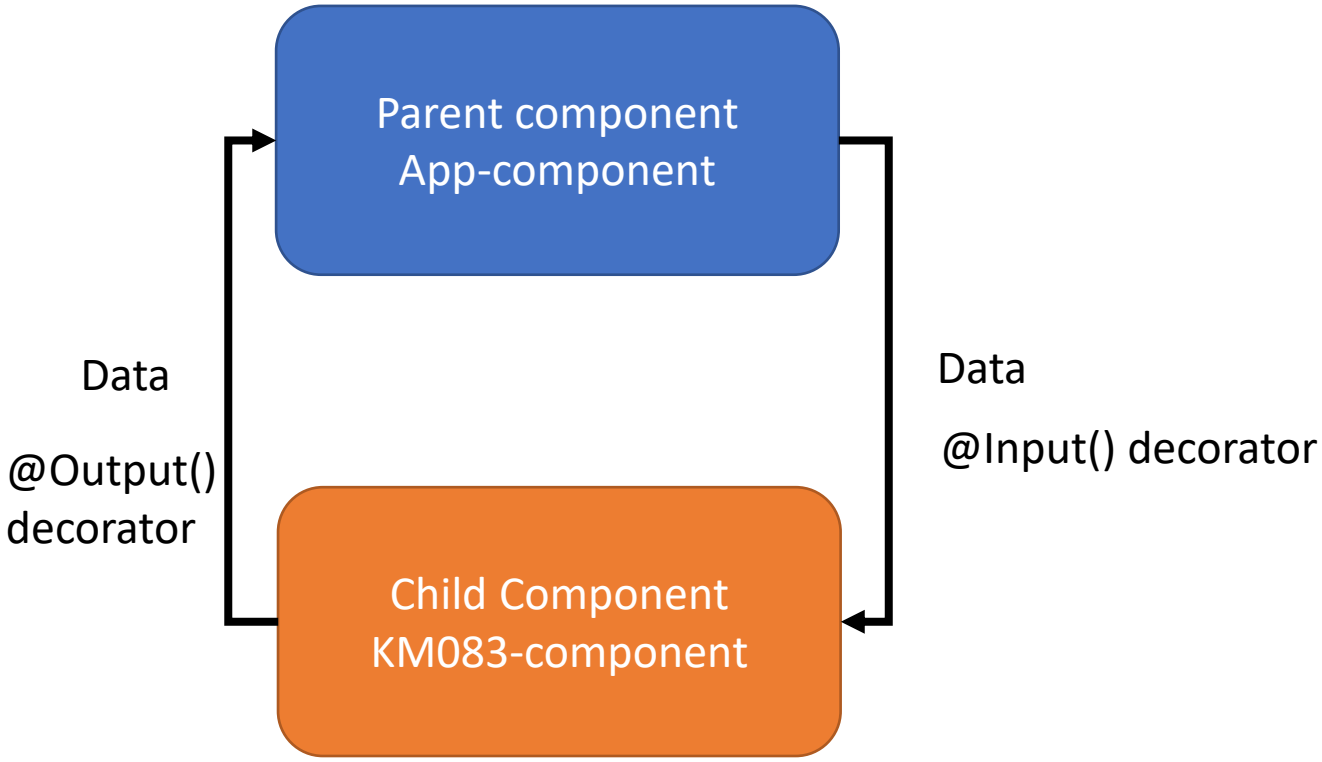
ngFor Directive

```
<div *ngFor="let color of color1; index as i">
            <h2>{{i}}{{color}}</h2>
</div>
```

```
color1 = ["red", "green","blue"];
```

- first as f
- last
- even
- odd

Component Interaction



Parent to child

- App-component.ts

```
public name=“INT219";
```

- App-component.html

```
<app-kmo83 [parentData]="name"></app-kmo83>
```

- Km083-component.ts

```
<h2>Hello {{parentData}}</h2>
```

```
@Input() parentData: any;
```

Child to Parent

- Km083-component.ts

```
<button (click)=onClick()>Sent to Parent</button>
```

```
@Output() childEvent=new EventEmitter();
onClick(){
this.childEvent.emit('Hey Parent'); }
```

- App-component.html

```
h2>Received: {{message}}</h2>
<app-kmo83 (childEvent)="message=$event"></app-kmo83>
```

- App-component.ts

```
message="";
```

Pipes

- Pipes allow us to transform data before displaying them in the view.

```
<h2>Code is: {{code}}</h2>
    <h2>Code is: {{code | lowercase}}</h2>
    <h2>Code is: {{code|uppercase}}</h2>
    <h2>Code is: {{code | titlecase}}</h2>
    <h2>Code is: {{code | slice:3}}</h2>
    <h2>Code is: {{code | slice:3:5}}</h2>
```

```
<h2>Number pipe: {{2.346 | number:'1.2-3'}}</h2>
<h2>Number pipe: {{2.346 | number:'3.4-5'}}</h2>
<h2>Number pipe: {{2.346 | number:'3.1-2'}}</h2>
<h2>Percent: {{0.25 | percent}}</h2>
<h2>Currency: {{0.25 | currency}}</h2>
<h2>Currency: {{0.25 | currency: 'INR'}}</h2>
<h2>Currency: {{0.25 | currency: 'INR':'code'}}</h2>
```

https://angular.io/api/common/DatePipe

- `<h2> {{ date }}</h2>`
- `<h2> {{ date | date:'short'}}</h2>`
- `<h2> {{ date | date:'shortDate'}}</h2>`
- `<h2> {{ date | date:'shortTime'}}</h2>`
- `<h2> {{ date |date:'medium'}}</h2>`
- `<h2> {{ date |date:'long'}}</h2>`
- `<h2> {{ date |date:’hh:mm'}}</h2>`

What if we want to display some data from a array into two components?

Component 1

```
employee=[
    {"name":"baljit","id":1},
    {"name":"singh","id":2},
    {"name":"saini","id":3}
]
```

```
<h2>Employee List</h2>
  <ul>
  <li *ngFor="let emp of employee">{{ emp.name
}}</li>
  </ul>
```

Component 2

```
<h2>Employee Data</h2>
<div *ngFor="let emp of
employee">
   {{emp.id}}.{{emp.name}}
</div>
```

Principles

- Do Not Repeat Yourself (DRY)
- Single Responsibility Principle

Service

- A class with a specific purpose

- Why services?
  - Share data
  - Implement Application Logic
  - External Interaction (e.g. connecting to Database)

- Naming convention
  - .service.ts
- How to use service?
  - Dependency Injection(DI)

Using a Service

1. Creating service
   - ng g s employee
2. Create a method to return employee in employee.service.ts

```
getEmployee(){
    return[
      {"name":"baljit","id":1},
      {"name":"singh","id":2},
      {"name":"saini","id":3}
    ];
  }
```

3. Register service with Injector
   - In app.module.ts
   - providers: [EmployeeService]
   - import { EmployeeService } from './employee.service’;

Using a Service

4. Declare the dependency in the components that require the service

- Km083.component.ts, which is one of the components
  - `import { EmployeeService } from '../employee.service’;`

  - `employee: any[] =[];`
  - `constructor(private eS:EmployeeService) { }`
  - `ngOnInit(): void {`

```
this.employee=this.eS.getEmployee();
}
```

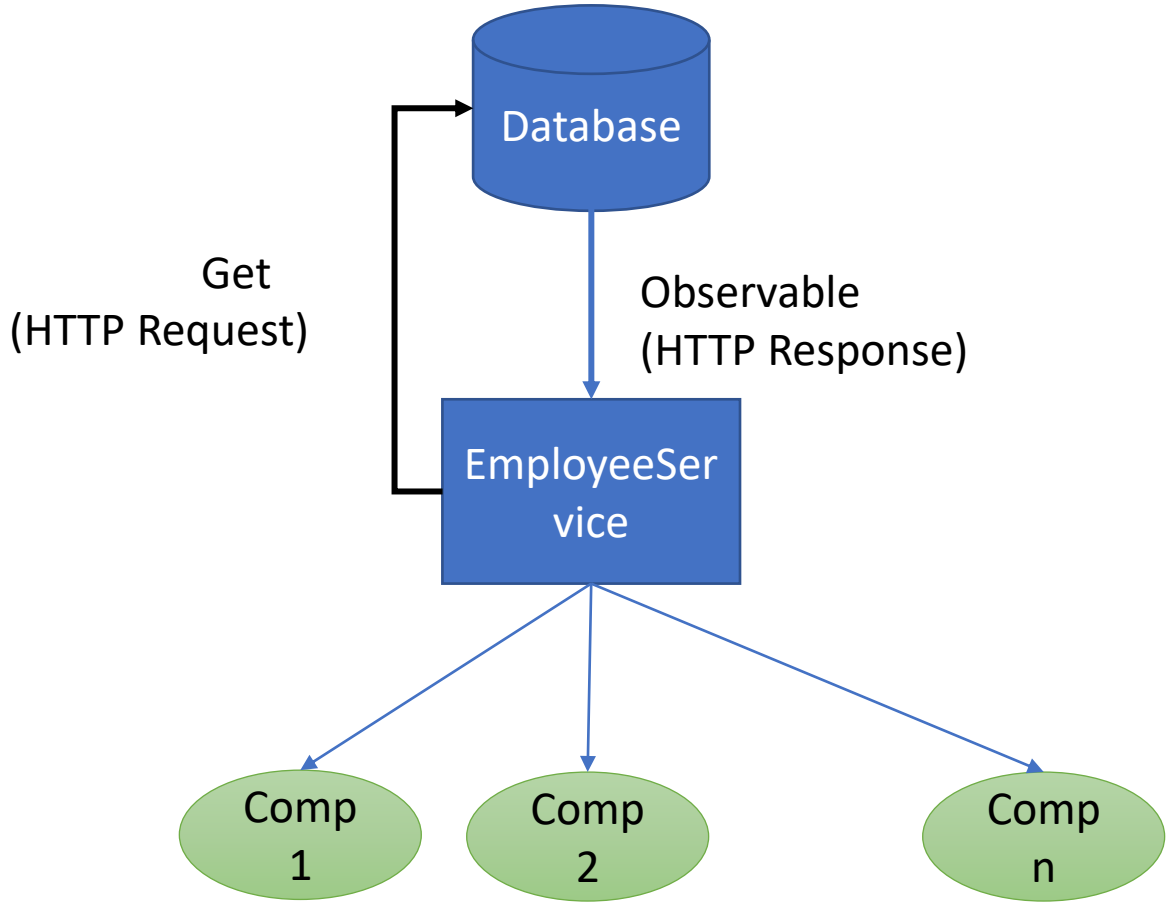
Using a Service

- new.component.ts, which is the other component
  - `import { EmployeeService } from '../employee.service’;`

  - `employee: any[] =[];`
  - `constructor(private eS:EmployeeService) { }`
  - `ngOnInit(): void {`

```
this.employee=this.eS.getEmployee();
}
```

HTTP and Observables



Observable

- A sequence of item that arrive asynchronously over time.
- HTTP call- single item
- Single item – HTTP response

Fetching Data using HTTP

1. EmpService sends HTTP Get Request
2. Observable is received and converted into employee array
3. The components subscribe to observable
4. Map a local variable to employee array

- RxJS
  - Reactive Extension for Javascript
  - External library to work with observables
  - It’s not ReactJS

Step 1

- App.module.ts
  - `import {HttpClientModule} from '@angular/common/http’;`
  - `imports: [`

    `HttpClientModule],`
- Since DB is not available, we create a local file
  - src->assets->data->employee.json
- employee.service.ts
  - `import {HttpClient} from '@angular/common/http’`

  - `url: string = "/assets/data/employee.json"`
  - `constructor(private http: HttpClient) { }`
  - `getEmployee(){`
  - `return this.http.get(this.url);`

Step 2

- Create a file
  - App->employee.ts

```
export interface IEmployee {
    id:number,
    name:string
}
```

Employee.service.ts

- `import { IEmployee } from './employee';`
- `import {Observable} from 'rxjs-compat/Observable’;`

```
getEmployee():Observable<IEmployee[]>{
    return this.http.get<IEmployee[]>(this.url);
  }
```

Step 3 and 4

- Km083.component.ts

```
constructor(private eS:EmployeeService) { }

  ngOnInit(): void {
  /*   this.employee=this.eS.getEmployee(); */
  this.eS.getEmployee()
  .subscribe((data: any[]) =>this.employee=data);
  }
```

- Same in other component also.