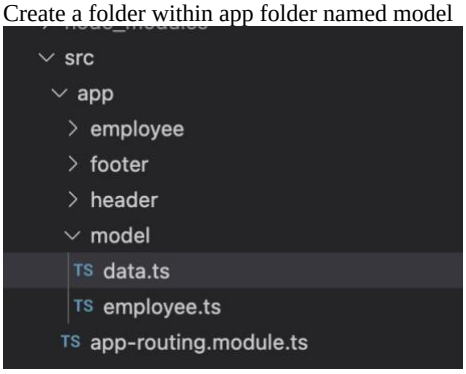
STEP 5: Prepare model and data

1. Create a folder within app folder named model



2. Create 2 files within model folder as following:
   a. employee.ts => that exports an interface Employee

```
export interface Employee{
  eid:number,
  ename:string,
  email:string,
  phone:string,
  password:string,
  address:{
      city?:string,
      country:string,
      zipcode?:number
   }
}
```

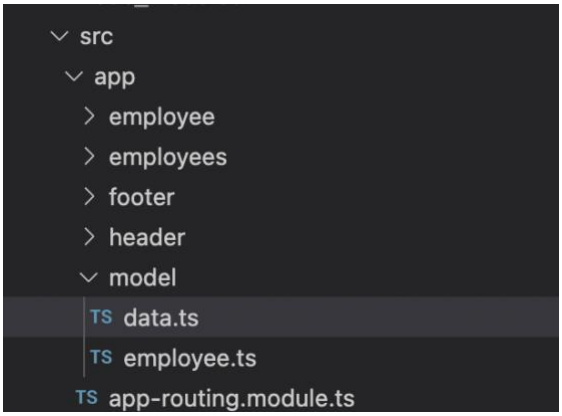
   b. data.ts => that exports array of employees

```
export let employees = [{eid:1,ename:'shalini',password:'shalini'
   email:'shalini@gmail.com',phone:'1321312312'
   , address:{country:'India'}},
   {eid:2,ename:'Reshma',
   email:'reshma@gmail.com',phone:'987659876',password:'shalini'1
   , address:{country:'India'}},
   {eid:3,ename:'Rahul',
   email:'rahul@gmail.com',phone:'645678678',password:'rahul'
   , address:{country:'India'}}]
```

STEP 6: Structural Directives [*ngFor and *ngIf]

1. Create Employees component using the below command from within the angulardemo folder:
   ng g c employees

2. Folder structure looks as follows:



3. Import employees[] created before in employees.component.ts and display on employees.component.html using *ngFor.

```
<div *ngIf="employees.length > 0">
      <div *ngFor="let emp of employees">
         <p>Name: {{emp.ename}}</p>
         <p>Name: {{emp.email}}</p>
      </div>
   </div>
   <div class="container" *ngIf="employees.length <= 0">
      <h1>No employees yet joined your company!!!</h1>
   </div>
```

OR

The track expression allows Angular to maintain a relationship between your data and the DOM nodes on the page. This allows Angular to optimize performance by executing the minimum necessary DOM operations when the data changes

```
@if(employees.length){
   <h4>Employees List with  if</h4>
   @for(emp of employees; track emp.eid){
                  <p>Name: {{emp.ename}}</p>
                  <p>Name: {{emp.email}}</p>

      }
}
@else{
   <p>No employees yet with else</p>
}
```

Providing a fallback for @for blocks with the @empty block

You can optionally include an @empty section immediately after the @for block content. The content of the @empty block displays when there are no items:

```
@for (item of items; track item.name) {
  <li> {{ item.name }}</li>
} @empty {
  <li aria-hidden="true"> There are no items. </li>
}
```

https://angular.dev/guide/templates/control-flow

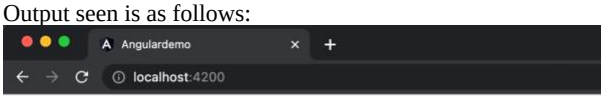
EXAMPLES:

```
@if (isEnabled ){
      <div>if</div>
   }
   @else{
      <div>else</div>
```

```
    }

  template: `
    @if (showHello) {
      <h2>Hello</h2>
    }
    @else if (showGoodbye) {
      <h2>Goodbye</h2>
    }
    @else {
      <h2>See you later</h2>
    }
  OR
  <div *ngIf="isEnabled else t1"></div>
  <ng-template #t1>.. </ng-template>
```

4. Add <app-employees> tag within the app.component.html file
5. Output seen is as follows:



angulardemo

Some Message

Some Message

header works!

Name: shalini

Name: shalini@gmail.com

Name: Reshma

Name: reshma@gmail.com

Name: Rahul

Name: rahul@gmail.com

footer works!

STEP 7: @Input Decorator

1. Instead of displaying the values in EmployeesComponent, pass the employee object from EmployeesComponent to EmployeeComponent using @Input decorator and [] data binding syntax

```
<div *ngIf="employees.length > 0">
          <app-employee [employee]="emp">
          </app-employee>
      </div>
    </div>
    <div class="container" *ngIf="employees.length <= 0">
      <h1>No employees yet joined your company!!!</h1>
    </div>
```

2. Add instance variable in employee.component.ts whose name should be same as provided in the [] brackets above as highlighted

```
@Input()
  employee:Employee
constructor() {
    this.employee = {eid:1,ename:'',password:'',
    email:'',phone:''
    , address:{country:''}}
  }
Transform:
To change the value before passing to component
@Input({
     alias: "userName",
    required: true,
```

```
  transform: (value: string) => value.toUpperCase(),
 })
```

built-in input transforms that are useful in many common scenarios: booleanAttribute and numberAttribute

```
@Input()
 disabled: boolean;
```

From parent
```
<app-child [disabled]="true"
```

Instead if we want to use as follows:
```
<app-child disabled/>
```

Then modify input as below:
```
@Input({
   transform: booleanAttribute,
 })
 disabled: boolean;
```

the mere presence of the disabled property (even without a value) will cause the disabled property to be true, or false otherwise.

```
@Input({
   transform: numberAttribute,
 })
 age: number;
```

```
<app-child age="20" />
```

Any string that we set as the age property will be automatically converted to a number, or to NaN if the conversion is not possible.

If we didn't have this transform in place, we would have to use a slightly more verbose syntax:

```
<app-child [age]="20"/>
```

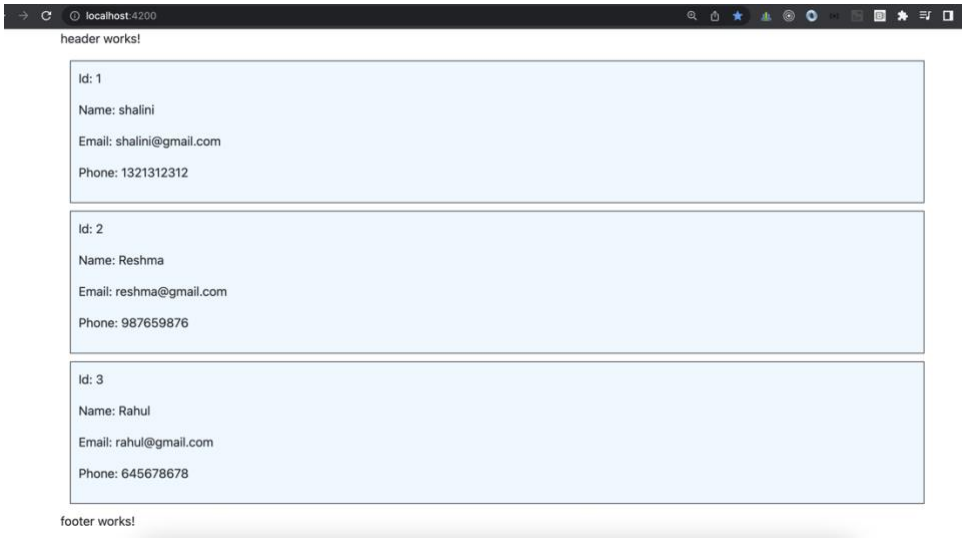
3. Update employee.component.html as follows:

```
<div class="employee">
   <p>Id: {{employee.eid}}</p>
   <p>Name: {{employee.ename}}</p>
   <p>Email: {{employee.email}}</p>
   <p>Phone: {{employee.phone}}</p>
</div>
```

4. Add styles to employee data in employee.component.css file

```
.employee{
   background-color: aliceblue;
   margin-bottom: 10px;
   padding: 10px;
   border: 1px solid;
}
```

5. Output seen is as below:



https://blog.angular-university.io/angular-if/