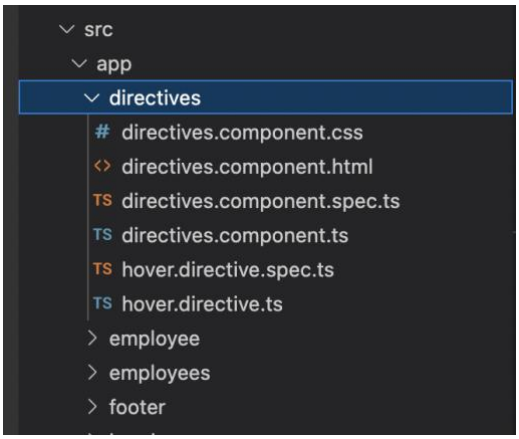
STEP 9: Directives

1. Add Directive component by executing the below command from the angulardemo folder:
   ng g c directives

2. Add a custom directive **from within the directives folder** using the below command:
   ng g d hover

```
(base) Manishs-MacBook-Pro:angulardemo Shalini$ ng g c directives
Node.js version v17.5.0 detected.
Odd numbered Node.js versions will not enter LTS status and should not be used for production.
 For more information, please see https://nodejs.org/en/about/releases/.
CREATE src/app/directives/directives.component.css (0 bytes)
CREATE src/app/directives/directives.component.html (25 bytes)
CREATE src/app/directives/directives.component.spec.ts (627 bytes)
CREATE src/app/directives/directives.component.ts (291 bytes)
UPDATE src/app/app.module.ts (1075 bytes)
You have new mail in /var/mail/Shalini
(base) Manishs-MacBook-Pro:angulardemo Shalini$ cd src/app/directives/
(base) Manishs-MacBook-Pro:directives Shalini$ ng g d hover
Node.js version v17.5.0 detected.
Odd numbered Node.js versions will not enter LTS status and should not be used for production.
 For more information, please see https://nodejs.org/en/about/releases/.
CREATE src/app/directives/hover.directive.spec.ts (220 bytes)
CREATE src/app/directives/hover.directive.ts (139 bytes)
UPDATE src/app/app.module.ts (1158 bytes)
```

3. Folder looks as follows:



4. Comment out the previous tags from app.component.html file and add <app-directives> tag
5. To use ngFor directive add below code in directives.component.html file:
   <li *ngFor="let pno of [1,2,3,5,7,11,13,17];let i = index">
   {{i}} : {{ pno }}
6. To use switch case, add below data in directives.component.ts file :
   people: any[] = [
   {
   "name": "Douglas Pace",
   "age": 35,
   "country": 'MARS'
   },
   {
   "name": "Mcleod Mueller",
   "age": 32,
   "country": 'USA'
   },

```
{
"name": "Aguirre Ellis",
"age": 34,
"country": 'UK'
},
{
"name": "Cook Tyson",
"age": 32,
"country": 'USA'
}
];
```

7. Display style conditionally as follows:

```
<ul *ngFor="let person of people”  [ngSwitch]="person.country">
<li *ngSwitchCase="'UK'" class="text-success">
        {{ person.name }} ({{ person.country }})
</li>
<li *ngSwitchCase="'USA'"class="text-primary">
        {{ person.name }} ({{ person.country }})
</li>
<li *ngSwitchDefault  class="text-warning">
        {{ person.name }} ({{ person.country }})
</li>
</ul>
```

8. Create custom directives:

   Custom directives offer a way to encapsulate and reuse code that deals with DOM manipulation or event handling, making your application more maintainable and readable. They can help in achieving the following benefits:

   a. Reusability

Custom directives allow you to package a specific set of behaviors or functionalities into a reusable component. Once defined, you can apply the same behavior to multiple elements across your application, reducing code duplication and improving consistency.

b. Separation of Concerns
Custom directives promote the separation of concerns by isolating DOM manipulation logic from your component's business logic. This separation makes your codebase cleaner and more maintainable, as each component focuses on its specific responsibilities.

c. Readability
By encapsulating complex DOM interactions within custom directives, your component code becomes cleaner and more readable. This makes it easier for other developers to understand and work with your code.

d. Testing
Custom directives can be tested independently, ensuring that the behavior you've encapsulated in them functions correctly. This makes it easier to identify and fix issues in your application.

e. Adaptability
Custom directives are versatile and can be applied to a wide range of scenarios. They provide a flexible way to add functionality to elements without modifying the underlying component

Use Cases for Custom Directives

Custom directives can be applied in various scenarios to enhance your Angular application's functionality. Let's explore some common use cases where custom directives shine:

a. Input Validation
You can create custom directives to enforce input validation rules on form fields. For instance, you can create a directive that ensures a password input meets specific complexity requirements or restricts input to certain characters.

b. Highlighting and Animation
Custom directives are excellent for creating animations or highlighting elements on user interaction. For example, you can create a directive that adds a shimmer effect to an element when the user hovers over it.

c. Lazy Loading
Custom directives can be used to lazy load content or images when they become visible in the viewport. This is particularly useful for optimizing performance on pages with large amounts of content.

d. Responsive Design
You can create custom directives to conditionally show or hide elements based on screen size or other conditions. This helps in building responsive web applications that adapt to different devices and screen sizes.

e. Access Control

Custom directives can enforce access control rules in your application. For example, you can create a directive that hides or disables certain UI elements based on the user's permissions.

f. Third-Party Integration

When integrating third-party libraries or widgets into your Angular application, custom directives can provide an abstraction layer that simplifies the integration process and keeps your codebase clean

Custom directive as a tooltip:

1. Within hover directive add input property with name same as selector:
   ```
   @Input()
    hover: string | null = null;
   ```

2. Create an HTMLElement reference :
   ```
   private tooltipElement: HTMLElement;
   ```

3. Inject below reference in the constructor as follows and create a new element that will be appended as a tooltip and displayed when the mouse is hovered over the host element:
   ```
   constructor(private el: ElementRef, private renderer: Renderer2) {
           this.tooltipElement = this.renderer.createElement('div');
           this.renderer.addClass(this.tooltipElement, 'tooltip1');
   }
   ```

4. Add tooltip1 as a css class in directive.component.css as follows:

   ```
   .tooltip1 {
     position: absolute;
     background-color: #333;
     color: #fff;
     padding: 5px;
     border-radius: 4px;
     z-index: 1;
     top: 50%;
     left: 50%;
     transform: translateX(-50%);
     display: none;
   }

   :host(:hover) .tooltip1 {
     display: block;
   }
   ```

5. Add listeners that will display the tooltip when mouse is hovered and remove when mouse leaves:

   ```
   @HostListener('mouseenter')
    onMouseEnter() {
   ```

```
  this.tooltipElement.textContent = this.hover;
  this.renderer.appendChild(this.el.nativeElement, this.tooltipElement);
 }
 @HostListener('mouseleave')
 onMouseLeave() {
  this.renderer.removeChild(this.el.nativeElement, this.tooltipElement);

 }
```

6. Use in HTML file as follows:
   ```
   <div  [hover]="'more info'">hey</div>
   ```

   NOTE: When the input property name and selector name is same, directive can be used as property binding syntax.
7. To bind with any of the host element attributes add below 2 properties in custom directive:
   ```
   @HostBinding('style.backgroundColor')
    backgroundColor: string ='';

    @HostBinding('style.transition')
    trans: string ='';
   ```

8. To provide with default transition add below in the constructor:
   ```
   this.trans = '1s ease-in-out';
   ```

9. To change background color add below in mouse enter and mouse leave respectively as follows:

   ```
   this.backgroundColor = 'lightgreen';

   this.backgroundColor = 'white';
   ```

https://blog.angulartraining.com/how-to-control-native-html-elements-with-angular-40aecb9e6796