1. ngIf Directive

Şimdi diyelim ki ProductComponent içerisinde aşağıdaki gibi bir model oluşturduk, bir önceki word dosyasından hatırlarsak bu model aracılığı ile product listesine ulaşabiliyorduk veya id'ye göre bir product döndürebiliyorduk.

```
export class ProductComponent  {

 model: ProductRepository = new ProductRepository();

}
```

Şu anda template üzerinden aşağıdaki gibi model üzerinden product'lara ulaşılabilir.

```
<div class="bg-primary text-white m-2 p-2">
    There are {{model.getProducts().length}} products.
</div>

<div class="bg-primary text-white m-2 p-2">
    There are no products in the repository.
</div>
```

Yukarıdaki view kullanılırsa iki div de görülecektir. Diyelim ki benim isteğim şu: eğer sayfada product varsa ilk div gösterilsin yok eğer product yoksa ikinci div gösterilsin.

İşte böyle bir durumda ngIf directive'i kullanılabilir, yani bir html elemanının component'e bağlı bir koşul ile gösterilip gösterilmeyeceğine karar verilebilir:

```
<div *ngIf="model.getProducts().length>0" class="bg-primary text-white m-2 p-2">
    There are {{model.getProducts().length}} products.
</div>

<div *ngIf="model.getProducts().length==0" class="bg-primary text-white m-2 p-2">
    There are no products in the repository.
</div>
```

Şimdi ngIf'in ng-template ile kullanımına bakalım:

```
<div *ngIf="model.getProducts().length>0; then temp1 else temp2"
class="bg-primary text-white m-2 p-2"> </div>

<ng-template #temp1>
    Burada birtakım işlemler yapılıyor olsun.
</ng-template>

<ng-template #temp2>
    Burada birtakım işlemler yapılıyor olsun.
</ng-template>
```

Temp1 ve temp2 id'leri ile iki farklı ng-template oluşturuldu ve bir div içerisinde koşula bağlı olarak temp1 veya temp2 yukarıdaki gibi çağırıldı.

Burada önemli bir nokta şu, ngIf'in yer aldığı div sayfada hiçbir şekilde görünmeyecek bu yüzden buna class vermek anlamsızdır, div'in içine yazı da yazsak görünmeyecektir. Ayrıca ng-template etiketi de sayfada görünmeyecektir sadece ng-template'in içindeki kısım sayfada görünecektir.

O halde eğer gerekli yazıyı bir div içinde göstermek istiyorsak ng-template içine aşağıdaki gibi div vermek mantıklı olacaktır:

```
<div *ngIf="model.getProducts().length>0; then temp1 else temp2"></div>

<ng-template #temp1>
<div class="bg-primary text-white m-2 p-2">
    1.Burada birtakım işlemler yapılıyor olsun.
</div>
</ng-template>

<ng-template #temp2>
 <div class="bg-primary text-white m-2 p-2">
    2.Burada birtakım işlemler yapılıyor olsun.
</div>
</ng-template>
```

Mesela eğer koşul true ise html sayfasında sadece aşağıdaki kısım görünecektir:

```
<div class="bg-primary text-white m-2 p-2">
    1.Burada birtakım işlemler yapılıyor olsun.
</div>
```

ngIf kullanmadan koşula bağlı olarak html sayfasındaki bazı elemanları göstermenin veya göstermemenin bir diğer yolu daha vardır. Bu daha önce öğrendiğimiz property binding ile de yapılabilir:

- Bunu yapmak için elemanın “hidden” attribute’una true veya false atanır. Bu true veya false ataması component içindeki değişkenler kullanılarak yapılabilir.

```
<div [hidden] = "model.getProducts().length>0"
    class="bg-primary text-white m-2 p-2">
    Burada birtakım işlemler yapılıyor olsun.
</div>
```

- Göründüğü gibi hidden property’sine true atanırsa bu div sayfada görünmez olacaktır.

2. ngSwitch Directive

Şimdi ngSwitch'e bakalım:

```
<div class="bg-info m-2 p-2" [ngSwitch]="model.getProductsCount()">
        <span *ngSwitchCase="0">There are no products</span>
        <span *ngSwitchCase="1"> There are 1 products</span>
        <span *ngSwitchCase="2">There are 2 products</span>
        <span *ngSwitchDefault>There are many products</span>
</div>
```

- Burada yapılan şey şu, [ngSwitch] içine componentten bir değer alınıyor ve daha sonra:
    - Eğer bu değer 0 ise ilk span gösteriliyor, 1 ise diğer span, 2 ise diğeri.
    - Eğer bu durumların hiçbiri geçerli değilse Default olan switch case gösteriliyor.

Peki ya ngSwitch içine alınan değer bir number değil de string olsaydı?

```
<div class="bg-info m-2 p-2" [ngSwitch]="model.getProductsById(1).name">
    <span *ngSwitchCase="'Samsung S5'">Samsung S5</span>
    <span *ngSwitchCase="'Samsung S6'">Samsung S6</span>
    <span *ngSwitchCase="'Samsung S7'">Samsung S7</span>
    <span *ngSwitchDefault>Unknown phone</span>
</div>
```

- Yine mantık aynı bu kez " " içerisinde string kullanmamız gerektiği için istersek "variableName" ile component içerisinden bir variable çekebiliriz ya da kendimiz yazmak istersek tek tırnak işaretini kullanırır.

3. ngFor Directive

Şimdi ngFor'a bakalım. Bu directive ile yapılan şey aslında bir döngü. Örneğin component içerisindeki prduct listesinin içindeki her product için bir <li> elemanını view üzerinde gösterebiliriz.

```
<ul>
    <li *ngFor="let item of model.getProducts(); index as i">
        {{i+1}} - {{item.name}}
    </li>
</ul>
```

Örneğin yukarıdaki kullanımda model.getProducts() methodu ile bir product listesi return edilir, ngFor ile bu listenin her elamanı için bu <li> elemanı üretilir ve sayfada gösterilir.

İndex as i diyerek de ngFor'un index özelliğinden yararlanabiliriz, her bir eleman için 0'dan başlayara bir index tutulur.

Angular'ın sitesinde ngfor'a gidip Local Variables'ı incelersek index dışında başka local variable'lar olduğunu görürüz. Örneğin:

- first: ilk eleman için true döndürür.
- last: son eleman için ture döndürür.
- even: çift elemanlar için true döndürür
- odd: tek elemanlar için true döndürür.
-

```
<ul>
    <li *ngFor="let item of model.getProducts();
     index as i; first as isFirst; last as isLast">
        {{i+1}} - {{item.name}}
        <span *ngIf="isFirst">First item</span>
        <span *ngIf="isLast">Last item</span>
    </li>
</ul>
```

Yukarıdaki gibi bu local variable'ları kullanabiliriz, sonuç şöyle görünür:

- 1 - Samsung S5 First item
- 2 - Samsung S6
- 3 - Samsung S7
- 4 - Samsung S8
- 5 - Samsung S9
- 6 - Samsung S10 Last item

4. ngFor Listedeki Değişimi Otomatik Olarak Yakalar

ProductRepository class'ının içinde aşağıdaki gibi bir getProducts methodu vardı:

```
    getProducts(): Product[]{
        return this.products;
    }
```

Biz de component üzerinde model isminde bir ProductRepository objesi oluşturuyoruz:

```
model: ProductRepository = new ProductRepository();
```

Sonuçta view içerisinden de bu obje aracılığı ile getProducts'a ulaşıyoruz.

```
li *ngFor="let item of model.getProducts();
```

ProductRepository'de return edilen this.products değerinin bir referans olduğunu unutma, sonuçta kendi içindeki products listesinin adresini döndürüyor.

Burada eğer ilgili products listesinde ürün ekleme çıkarma ürün adı değişimi gibi herhangi bir değişiklik olduğunda, bizim hiçbir ekstra işlem yapmamıza gerek kalmadan bu değişiklikler ngFor aracılığı ile view'imize yansıyacaktır, ngFor ile alınan listede herhangi bir değişiklik yapılınca liste kendi kendini yeniliyor.

Şimdi view içine aşağıdaki gibi bir click eventi ekledik click eventi trigger olunca, product listesine bir eleman eklensin istiyoruz:

```
<button class="btn btn-primary" (click)="addProduct()">Add Product</button>
<ul class="list-group mt-2">
    <li *ngFor="let item of model.getProducts();index as i; first as isFirst;
last as isLast">
        {{i+1}} - {{item.name}}
        <span *ngIf="isFirst">First item</span>
        <span *ngIf="isLast">Last item</span>
    </li>
</ul>
```

Component içerisinde addProduct methodunu dolduruyoruz ve rastgele bir product listeye eklensin diyoruz.

```
export class ProductComponent  {

 model: ProductRepository = new ProductRepository();

 addProduct() {
    this.model.addProduct(new Product(1, "Samsung S11", "İyİ telefon", '5.jpg'
, 1000));
 }

}
```

Component içindeki addProduct methodu da repository içerisindekini çağırıyor:

```
export class ProductRepository {
    private dataSource: SimpleDataSource;
    private products: Product[];

    constructor(){
        this.dataSource =  new SimpleDataSource();
        this.products = new Array<Product>();
        this.dataSource.getProducts().forEach(p=>this.products.push(p));
    }

    addProduct(product: Product) {
        this.products.push(product)
    }
```

Repository içerisindeki method ise push ile kendi listesine product'ı ekliyor.

Bu noktadan sonra biz view içerisinde butona tıkladığımız anda ngFor'un aşağıdaki gibi elde ettiği product listesi yenilenmiş oldu:

```
*ngFor="let item of model.getProducts();
```

Bu değişiklik ngFor tarafından anlık olarak yakalanıyor.

Özetle ngFor ile alınan listedeki herhangi bir değişiklik manuel bir işlem gerekmeksizin yakalanıyor.

Delete methodunu nasıl yapacağımıza bakarsak:

View:

```
    <li *ngFor="let item of model.getProducts();index as i; first as isFirst;
last as isLast">
        {{i+1}} - {{item.name}}
        <span *ngIf="isFirst">First item</span>
        <span *ngIf="isLast">Last item</span>
        <button class="btn btn-
danger" (click)="deleteProduct(product)">Delete</button>
    </li>
```

Component:

```
 deleteProduct(product:Product){
    this.model.deleteProduct(product);
 }
```

Repository:

```
    deleteProduct(product: Product){
        let index =  this.products.indexOf(product);
        this.products.splice(index,1);
    }
```

5. ngTemplateOutlet

Bu kısımda ise tekrar tekrar kullanmak isteyebileceğimiz bir yapıyı nasıl oluşturabileceğimize bakacağız.

Diyelim ki aşağıdaki gibi bir div yapımız var, burada tüm product'lar gösterilmiş:

```
<div class="m-5">
    <ul class="list-group mt-2">
        <li *ngFor="let item of model.getProducts();">
            {{item.name}}
        </li>
    </ul>
</div>
```

Ben aynı yapıyı, sayfanın başka yerlerinde göstermek istiyorum mesela en çok satılan product'lar için ayrı bir div olsun, en yeniler için ayrı bir div olsun gibi.

Bunu yapmanın en basit yolu elbette kopyala yapıştır ile kodu çoğaltmak ve datanın alındığı model.getProducts() methodu yerine istenilen diğer methodları kullanmaktır, mesela model.getPopularProducts() gibi.

Fakat bunun daha good practice yolu: ngTemplateOutlet directive'inden yararlanmaktır.

```
<ng-template [ngTemplateOutlet]="productList"></ng-template>
<ng-template [ngTemplateOutlet]="productList"></ng-template>

<ng-template #productList>
    <div class="m-5">
        <ul class="list-group mt-2">
            <li *ngFor="let item of model.getProducts();">
                {{item.name}}
            </li>
        </ul>
    </div>
</ng-template>
```

Yukarıdaki gibi bir html sayfasında #productList ismindeki template yukarıda ngTemplateOutlet ile iki kez çağırılmış

Ancak bizim isteğimiz birebir aynı elemanların gösterilmesi olmayabilir, html yapısı aynı olsun ancak farklı product listeleri gösterilsin isteyebiliriz.

Şimdi bunun nasıl yapıldığına bakalım:

```
<ng-template
    [ngTemplateOutlet]="productList"
    [ngTemplateOutletContext]="{productsVar: model.getPopularProducts() }">
</ng-template>

<ng-template
    [ngTemplateOutlet]="productList"
    [ngTemplateOutletContext]="{productsVar: model.getLatestProducts() }">
</ng-template>

<ng-template #productList let-products="productsVar">
    <div class="m-5">
        <ul class="list-group mt-2">
            <li *ngFor="let item of productsVar;">
                {{item.name}}
            </li>
        </ul>
    </div>
</ng-template>
```

Burada görüldüğü gibi, kullanılacak template'in içinde productsVar değişkeni tanımlandı, bu değişken *ngFor içerisinde kullanılmış. Bu değişkenin içini template'in çağırıldığı yerlerde dolduruyoruz.

Örneğin ilk çağırılan template için productsVar model.getPopularProducts() methodunun return'üne eşitlenmiş. Benzer şekilde ikincide ise latest products bu değişkenin içine koyulmuş.

Böylece aynı template'i kullanarak farklı ürünler gösterilebilir.

Son olarak aşağıdaki şekilde birden fazla parametre de gönderilebilirdi:

```
<ng-template
    [ngTemplateOutlet]="productList"
    [ngTemplateOutletContext]="{productsVar: model.getLatestProducts(),
categories: model.getCat() }">
</ng-template>
```

```
<ng-template #productList let-products="productsVar" let-c="categories">
```

6. Custom Directives

Directive konusunun son kısmında ise bir custom directive'i nasıl oluşturabileceğimize bakalım.

Örneğin bir input'a mail girildiğinde @gmail.com kısmı unutulursa bunu otomatik olarak dolduran bir directive yazalım.

Input-email isminde custom directive'i oluşturmak için ilk olarak terminale

ng g d input-email yazalım.

- Böylelikle yeni bir .ts dosyasının yanında module.ts içerisinde de aynı yeni bir component veya custom pipe ekler gibi güncelleme yapılıyor.

Sonuçta oluşan yeni input-email.directive.ts dosyası aşağıdaki gibi:

```
import { Directive } from '@angular/core';

@Directive({
  selector: '[appInputEmail]'
})
export class InputEmailDirective {

  constructor() { }

}
```

Göründüğü gibi burada bir selector görünüyor, bu directive'i bir input içerisinde çağırmak için bu selector kullanılacak.

```
<input type="text" appInputEmail>
```

Şeklinde.

Şimdi istediğimiz directive'i kullanarak, ilgili input üzerindeki bazı eventleri yakalamak.

- Directive'in verildiği html elemanının event'leri yakalaması için, directive'e bir HostListener import etmeliyiz.

```
import { Directive, HostListener } from '@angular/core';

@Directive({
  selector: '[appInputEmail]'
})
export class InputEmailDirective {

  @HostListener('focus') onFocus(){
    console.log('focus gerçekleşti')
  }

  @HostListener('blur') onBlur(){
    console.log('focus kalktı')
  }

}
```

- Yukarıdaki yapı ile, bu directive'in bağlı olduğu input elemanına focus gerçekleşince HostListener ile bu event yakalanacak ve onFocus methodu çağırılacaktır.
- Benzer şekilde focus kalkınca blur event'i yakalanacak ve onBlur methodu çalıştırılacaktır.

Şimdi directive'de bir de ElementRef import ediyoruz böylelikle, directive'i kullanan elemana ulaşabileceğiz:

```
import { Directive, HostListener, ElementRef } from '@angular/core';

@Directive({
  selector: '[appInputEmail]'
})
export class InputEmailDirective {

  constructor(private element:ElementRef){}

  @HostListener('focus') onFocus(){
    this.element.nativeElement.classList.add('bg-warning')
  }

  @HostListener('blur') onBlur(){
    this.element.nativeElement.classList.remove('bg-warning')
  }

}
```

- Burada yapılan şey şu, ilgili directive'i kullanan input elemanında focus event'i gerçekleştiği zaman, element üzerinden yani input elemanı üzerinden classList'e ulaşılıyor ve elemana yeni bir class ekleniyor.
- Bu işlem javascript ile de yapılabilirdi.

Son olarak onBlur methoduna ekleme yapıyorum, burada yapılan input elemanının içindeki değeri almak eğer değerde @ yok ise lowercase yapıp @gmail.com eklemek.

```
  @HostListener('blur') onBlur(){
    this.element.nativeElement.classList.remove('bg-warning')

    let value: string =  this.element.nativeElement.value;
    if(!value.includes('@')){
      this.element.nativeElement.value = value.toLocaleLowerCase() + '@gmail.c
om'
    }
  }
```

Böylelikle custom directive mantığını da görmüş oluyoruz.