GİRİŞ

- Console'dan ng new appName şeklinde yeni bir app oluşturuyorduk.
- Proje içerisindeki angular.json dosyası proje hakkında belli meta bilgileri saklar, ve bazı proje ayarlarının yapıldığı bir dosya.
- Projenin kendisi src klasörü altında yer alıyor.
    - Main.ts dosyası uygulamaya giriş dosyası olarak düşünülebilir. PlatformBrowserDynamic()… kısmı ile uygulamanın bir tarayıcı üzerinde çalışacağını söylüyoruz. bootstrapModule(AppModule) diyerek de uygulamanın başlangıcında AppModule'ü çalıştırmasını söylüyoruz.

```ts
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-brow

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

    - App.module.ts doyası da angular içerisinden gelen modüllerin ve kendi hazırladığımız componentlerin bir birleşimi olarak düşünülebilir.

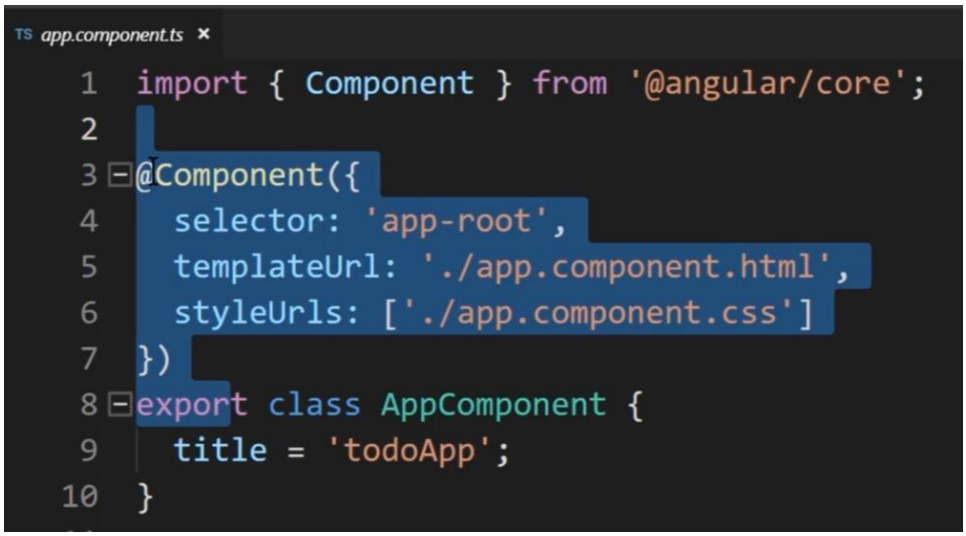
```ts
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

    - App.module içerisinde bir paket oluşturuluyor ve bu paket dışarıya açılır, böylece uygulamanın main.ts dosyası içerisinde app.module'ü çalıştırabilir hale geliyoruz.

- Ayrıca app.module ile aynı dizin içerisinde bir AppComponent'imiz var, bunu appmodule içerisinde kullanabilmek için önce import sonra declare ediyoruz, başka componentleri de kendimiz yaratıp buraya ekleyebilirdik.

- Bu compnenti declare etmenin yanında bootstrap: [AppComponent] satırı ile, app.module içerisinde ilk çalıştıralacak componenti belirtmiş olduk.

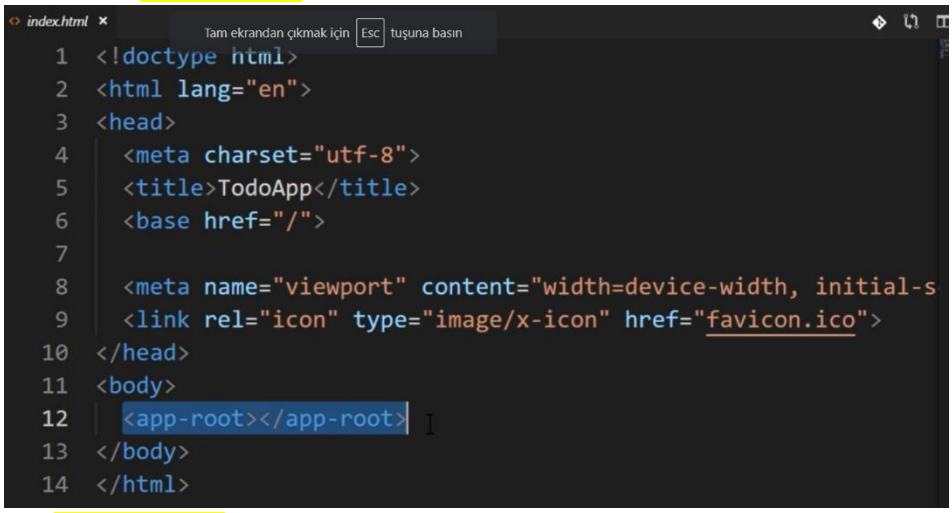
- Uygulamanın navigation kısmı, menü kısmı, list kısmı gibi herhangi bır kısmı bir component olarak tanımlanabilir böylece bu komponentler üzerinde ayrı ayrı çalışılır ve daha sonra biraraya getirilir.

- 

- App component'i de app.component.ts olarak tanımlanmış. Yukarıda içeriğini görebiliriz.

- Burada bir AppComponent class'ı tanımlanmış, ayrıca bir de @Component kısmı tanımlanmış. Buradaki selector, app componentini çağırmak için kullanılan etiket olarak düşünülebilir.

- AppComponent içerisindeki title veya diğer değişkenler, app.component.html içerisinde {{ }} içerisinde kullanılabilirler!

- Projenin index html'ine bakarsak:
- 

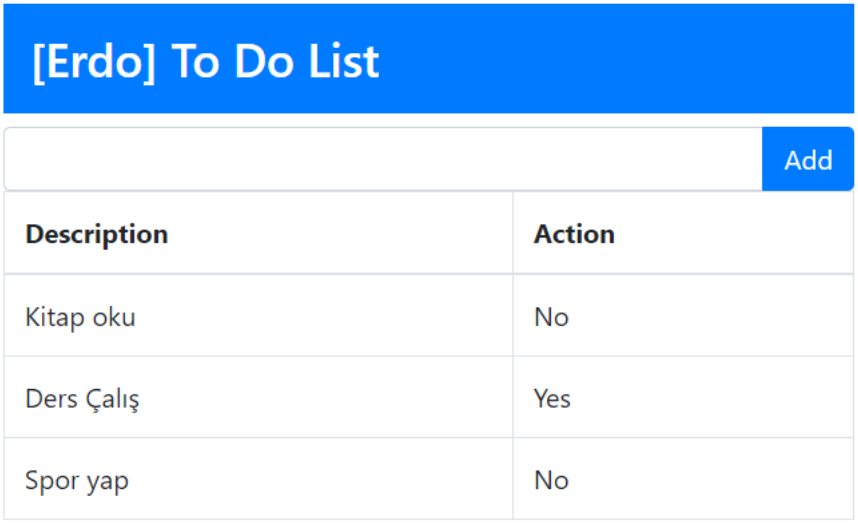
- Bir <app-root> etiketi ile karşılaşırız. İşte bu etiketin kullanıldığı yere app component çağırılmış oluyor. Bu app componentinin html ve css sayfası ayrı ayrı mevcut, bu html sayfası index.html içerisine getirilmiş oluyor.
- Yani AppComponent class'ı içerisinde mesela veritabanı işlemleri yapılıp gerekli değişkenler tanımlanabilir, daha sonra bu bilgiler componentin html'ine taşınır ve buradan da bu component html'i index.html içerisine selector tag'i ile taşınabilir. Böylece uygulamayı farklı komponentlerin birleşimi şeklinde var edebiliriz.

PART 2

- Vs code terminalinde proje klasöründeyken ng serve dersek projeyi çalıştırabiliriz ve defaut olarak uygulama localhost4200 port'u altında hizmet verir. Ng serve –port 3000 diyerek bu portu 3000 olarak değiştirebiliriz.
- Ng serve –port 3000 –open dersek de otomatik olarak index html tarayıcı da açılır. Ya da kendimiz port'a gidebiliriz.
- Projeyi çalıştırdım, bazı küçük değişiklikler yaptım, daha sonra şuna karar verdim projenin tamamında yani tüm html sayfalarında bootstrap kütüphanesini kullanabilmek istiyorum böylece bootstrap class'larını yazdığımda direkt kullanabileceğim. Bunun için:
    - Öncelikle projenin çalışmasını CTRL+C ile durduruyorum.
    - Daha sonra npm install –save bootstrap ve npm install –save jquery ile bootstrap ve jquery'i `node_modules/bootstrap'e` indiriyorum ve package.json'a dahil ediyorum.
    - Bir de şimdi bootstrap'i kullanacağım yerde tanımlamam gerekecek, biz tüm projede kullanmak istediğimiz için bu tanımlamayı angular.json içerisinde yapacağız:
        - Angular.json içindeki architect alından styles kısmında zaten projenin tamamında kullanılan styles.css dosyası tanımlanmış bunun yanına bir de bootstrap.min.css dosyasını tanımlıyoruz bu dosya da node_modules altına indirilmişti. Ayrıca scrpipt arrayinin içini de dolduruyoruz.
        -
```
],
"styles": [
  "./node_modules/bootstrap/dist/css/bootstrap.css",
  "src/styles.css"
],
"scripts": [
  "./node_modules/jquery/dist/jquery.js",
  "./node_modules/bootstrap/dist/js/bootstrap.js"
]
```
    - Artık istediğimiz yerde bootstrap componentlerini projemiz içinde kullanabiliyor olacağız.

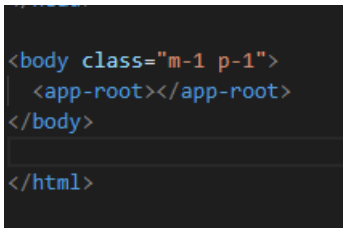
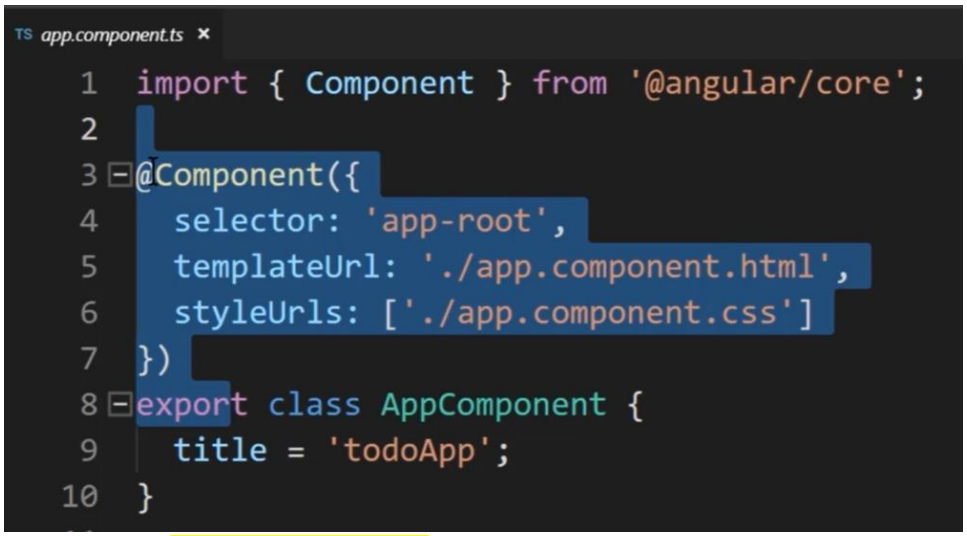
- Bu işlemi yaptıktan sonra app componentin html'ini sildik ve index.html içerisinde bootstrap'i kullanarak bir tablo oluşturduk:

```
<body class="m-1 p-1">

  <div class="container">
    <div class="row justify-content-center">
      <div class="col">
        <h3 class="bg-primary text-white p-3">
          [Erdo] To Do List
        </h3>
        <div class="input-group">
          <input type ="text" class="form-control">
          <div class="input-group-append">
            <button class="btn btn-primary">Add</button>
          </div>
        </div>

        <table class="table table-stripe table-bordered">
          <thead>
            <tr>
              <th>Description</th>
              <th>Action</th>
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>Kitap oku</td>
              <td>No</td>
            </tr>
            <tr>
```

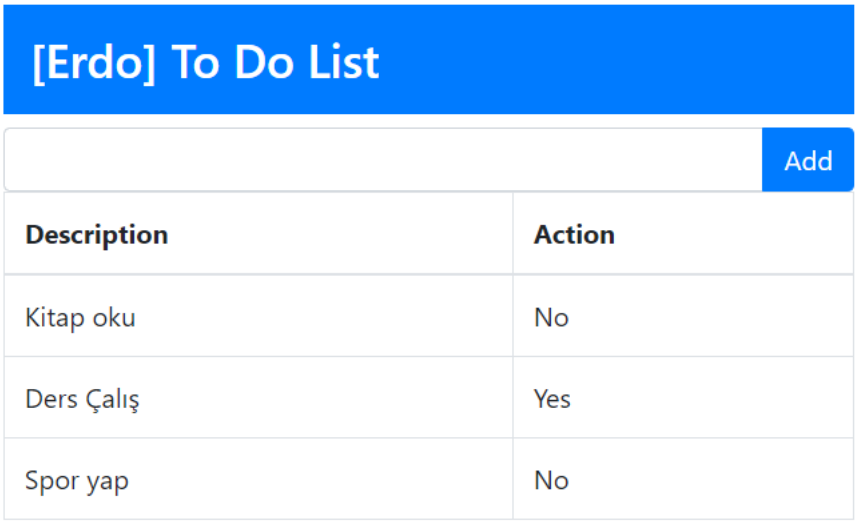
- 

- Bu sayfa şuanda bildiğimiz bir index.html sayfası doldurmaktan farklı değildi, bir angular yapısı kullanmadık. Bir sonraki kısımda angular'a geçeriz.

PART 3 – Model Binding

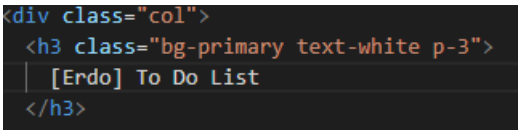
- Bu noktada GIT'den bahsedelim, projeyi oluştururken zaten GIT initialize edildi. Bu yüzden, uygulamadan yapılan değişiklikler CHANGES altında listelendi.
- Yeni bir branch oluşturup (sol alttan master'a tıklayıp create diyerek) bu branch aktifken değişiklikleri commit edersem, yeni branch değişiklikleri içerirken, master branch bunları içermeyecek.
- Şimdi daha önce index.html içerisinde oluşturulan container div'ini yani tabloyu app.component.html içerisine atıyoruz, index.html içerisinde bu componenti selector tag'i ile çağıracağız. Böylece component'ler ile çalışmaya başlamış olacağız.
- 
```
<body class="m-1 p-1">
  <app-root></app-root>
</body>

</html>
```
- ile component html'i i index.html içerisinde çağırmış olduk.
- Bunu yapmamızın avantajı şu:
- 
```
TS app.component.ts ×

1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'todoApp';
10 }
```
- Yukarıdaki app.comonent.ts içerisindeki bilgiler bize şunu diyor, AppComponent class'ı içerisinde tanımladığın değişkenleri app.component.html içerisinde {{}} ile istediğin gibi kullanabilirsin ve daha sonra da bu componenti <app-root> tag'i ile index.html içerisinde çağırabilirsin.

- Biz de bunu yapacağız mesela:

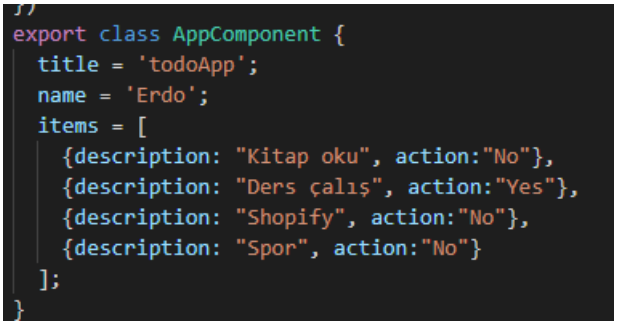


-

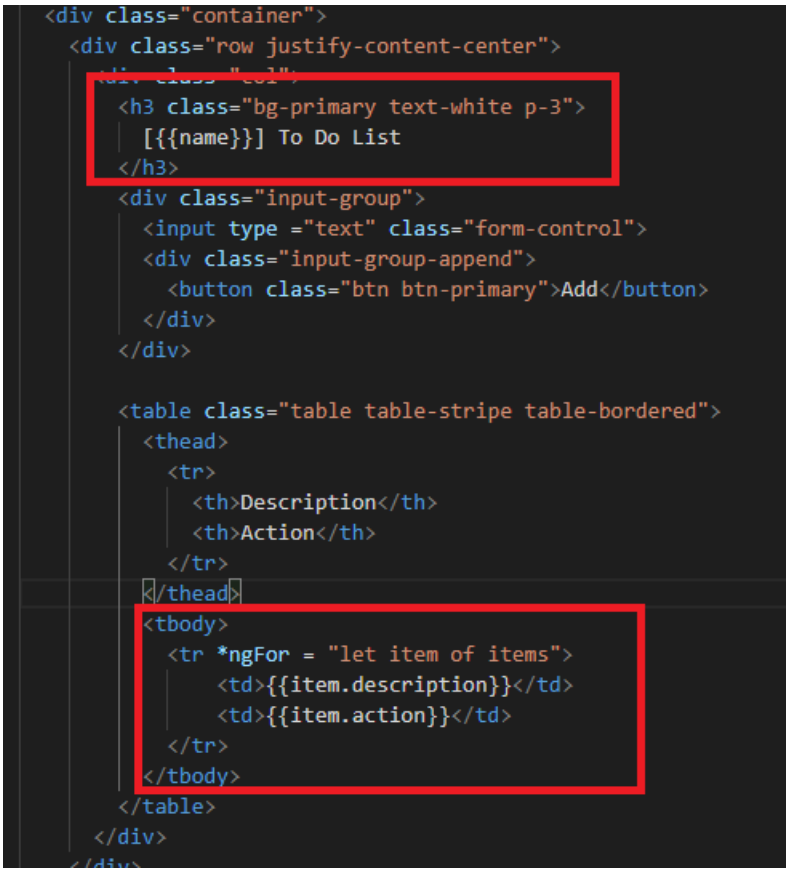
- Yukarıdaki [Erdo] kısmı app.component.html içerisinde:

- 

şeklinde tanımlı, biz bunun yerine önce AppComponent class'ında bir name tanımlayalım sonra component html'inde bu name değişkenini {{name}} şeklinde kullanalım:

-
- 

app component.ts

- 
```
<div class="container">
  <div class="row justify-content-center">
    <h3 class="bg-primary text-white p-3">
      [{{name}}] To Do List
    </h3>
    <div class="input-group">
      <input type ="text" class="form-control">
      <div class="input-group-append">
        <button class="btn btn-primary">Add</button>
      </div>
    </div>

    <table class="table table-stripe table-bordered">
      <thead>
        <tr>
          <th>Description</th>
          <th>Action</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor = "let item of items">
            <td>{{item.description}}</td>
            <td>{{item.action}}</td>
        </tr>
      </tbody>
    </table>
  </div>
```
app component.html

- Burada name değişkenine {{name}} olarak ulaşabildiğimiz gibi, oluşturulan item arrayine'e de ulaşabiliriz, item arrayinin her bir elemanı bir object ve objelerin de 2 field'i var.

- Önemli nokta *ngFor yapısı böylelikle item arrayindeki her bir elaman için <tr> etiketleri içerisinde iki <td> etiketi yaratıldı ve içeriği description ve action ile doldurulduu.

- Bu şekilde component class'ı içinde tanımlanan değişkenlerin component.html içerisinde kullanılmasına MODEL BINDING denir!

PART 4 – Model Oluşturma

- Önceki kısımın sonunda component.ts içerisinde bir name ve items tanımladık ve bunları component.html içerisinde kullandık.
- Ancak bu name ve items'i ayrı ayrı tanımlamak yerine bir model oluşuturp bu model içerisinde geçirseydik daha iyi bir practice olurdu bu kısımda bunu yapacağız.
- Bu yüzden app klasörü içerisinde model.ts isminde bir dosya oluşturuyoruz, ve modelimizi aşağıdaki gibi oluşturuyoruz bu model class'ının hem user hem de items isminde değişkenleri var, items'I de başka bir class kullanarak ürettik.
- 

```
export class myModel{
    user;
    items;

    constructor(){
        this.user = "Erdo";
        this.items = [
            new TodoItem("Kitap oku", true),
            new TodoItem("Ders çalış", true),
            new TodoItem("Shopify", false),
            new TodoItem("Spor", false)
        ];
    }
}

export class TodoItem{
    description;
    action;

    constructor (description, action){
        this.description = description;
        this.action = action;
    }
}
```

- Daha sonra bu model'i app component içerisinde aşağıdaki gibi kullanabiliriz:

```
import { Component } from '@angular/core';
import { myModel } from './model';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'todoApp';
  model = new myModel();
  getName() {
    return this.model.user;
  }
  getItems(){
    return this.model.items;
  }
}
```

    - Burada bir Model objesi oluşturduktan sonra getName ve getItems isminde component methodları tanımlandı, bu methodlar ile component.html'den name ve item verisi alınabilir:

```
<div class="container">
  <div class="row justify-content-center">
    <div class="col">
      <h3 class="bg-primary text-white p-3">
        [{{getName()}}] To Do List
      </h3>
      <div class="input-group">
        <input type ="text" class="form-control">
        <div class="input-group-append">
          <button class="btn btn-primary">Add</button>
        </div>
      </div>

      <table class="table table-stripe table-bordered">
        <thead>
          <tr>
            <th>Index</th>
            <th>Description</th>
            <th>Action</th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor = "let item of getItems(); let i = index">
              <td>{{i+1}}</td>
              <td>{{item.description}}</td>
              <td>{{item.action}}</td>
          </tr>
        </tbody>
```

    - Burada getName beklendiği gibi kullanılıyor. getItems() de aynı şekilde bu kez ngFor için let item of getItems() ibaresi kullanılıyor yani getItems()'in return ettiği array'in her bir elemanını item olarak kabul et ve her bir elemanı için bu tr'den üret ve altalta ekle diyor.

- Buna ek olarak let i = index diyerek her bir eleman için index değeri i içerisinde elde edilebildi.
- Index'i göstermek için tabloya bir sütun daha ekledik:

[Erdo] To Do List

Add

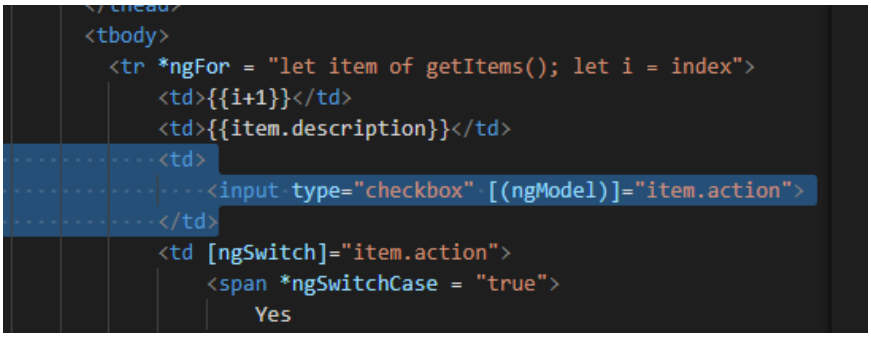
Index	Description	Action
1	Kitap oku	true
2	Ders çalış	true
3	Shopify	false
4	Spor	false

-
- Son olarak action kısmını true false şeklinde değil de Yes, No şeklinde yazsın istiyoruz bu yüzden component.html içerisinde bir başka directive olan ngSwitch'i aşağıdaki gibi kullanırım:
-

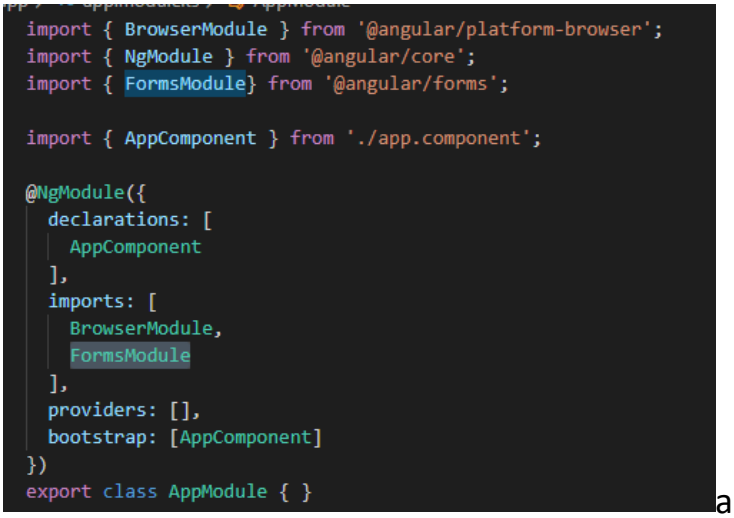
```
<tr *ngFor = "let item of getItems(); let i = index">
    <td>{{i+1}}</td>
    <td>{{item.description}}</td>
    <td [ngSwitch]="item.action">
        <span *ngSwitchCase = "true">
            Yes
        </span>
        <span *ngSwitchCase = "false">
            No
        </span>
    </td>
```

    - Burada şöyle dendi eğer item.action true ise span içinde Yes elemanı kullanılsın değilse No.
- Directive'ler ile component class'ı içerisindeki değişkenleri kullanarak for loop veya switch case ile html etiketlerini kombinlediğimize dikkat et!

PART 5 – NgModel ve Filtreleme

- Şimdiye kadar yapılan işlemle component class'ında tanımlanan get methodları ile, component html'inde name ve items'e ulaştık ve bu değişkenlerin içeriklerini kullanarak bir html kompozisyonu oluşturduk. Buna da model binding demiştik. Daha sonra bu component.html'de index html içerisinde kendi tag'i ile çağırılmıştı.
- Burada model binding'in tek yönlü olduğuna dikkat et yani sadece component.ts'de tanımlanan veriler component.html'e aktarılıyordu. Eğer ngModel kullanırsak, bunun tersi de mümkün olacak yani component class'ındaki değişken içerikleri, component html üzerinden değiştirilebilecek. Böylece bu değer heryerde değişmiş olacak.
- Şimdi bunu yapalım:
- 

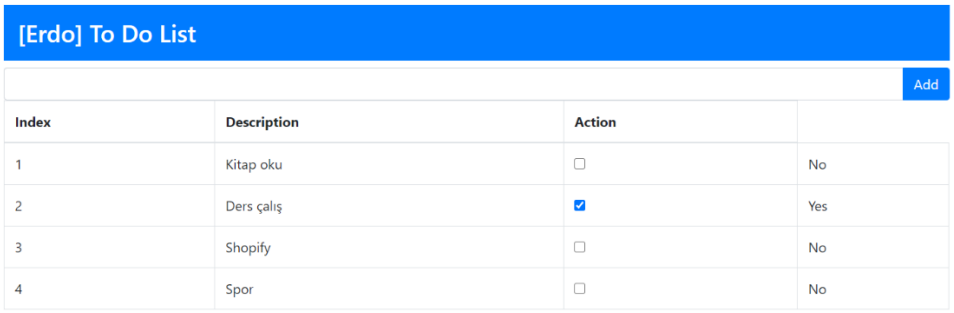
```
<tbody>
  <tr *ngFor = "let item of getItems(); let i = index">
      <td>{{i+1}}</td>
      <td>{{item.description}}</td>
      <td>
          <input type="checkbox" [(ngModel)]="item.action">
      </td>
      <td [ngSwitch]="item.action">
          <span *ngSwitchCase = "true">
              Yes
```

    - Burada bir checkbox input'u tanımlanıyor ve buna da [(ngModel)] directive'i ekleniyor. Böylelikle bu checkbox'ın value'su ile component class'ından gelen ilgili item'in action değeri pair ediliyor. Burada item.action boolean olmalı ki pair işlemi sağlansın, zaten boolean idi. Artık biri değişince diğeri de değişecek!
- Fakat ngModel'i kullanabilmek için bunun modülünü app.module içerisindeki imports kısmında FormsModule'ü import etmeliyiz.
- 

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule} from '@angular/forms';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

app.module.ts

- Bu noktada, item.action ile bu checkbox value pair edilmiş oldu, artık biri değişince diğeri de değişiyor.



- Şimdi ise yapacağımız şey, tick edilen yani item.action değeri true olan tüm itemları unvisible yapmak.
- Bu amaçla component classındaki getItems() methodunu modify ediyorum, burada array'in built-in filter methodu kullanılıyor. => işaretinin sağındaki ifadeyi true yapan tüm item'lar yani action'ı false olan tüm item'lar item filtrelenmiş oluyor.

```
export class AppComponent {
  title = 'todoApp';
  model = new myModel();
  getName() {
    return this.model.user;
  }
  getItems(){
    return this.model.items.filter(item => !item.action);
  }
}
```

PART 6 – Listeye yeni eleman ekleme

- Bu kısımda yapacağımız şey, component.html sayfasındaki add butonunu kullanarak, component'in model'inin items listesine eleman eklemek olacak. Bunu bir nevi terse doğru bilgi geçirme olarak düşünebiliriz.
- Bunu yapmak için Add butonuna bir click eventi ekleyeceğiz.
- Component.html içerisinde bir input ve bir buton vardı bunları aşağıdaki gibi manipüle edelim:

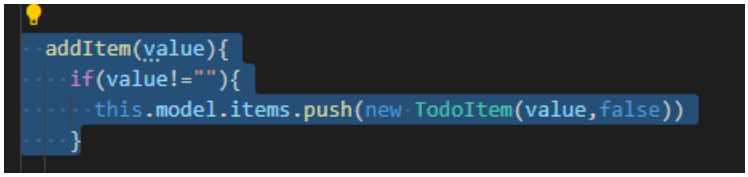

</h3>

<input type ="text" #todoText class="form-control">

<button class="btn btn-primary" (click)="addItem(todoText.value); todoText.value ='';">Add</button>

<table class="table table-stripe table-bordered">
- Ne yapmış olduk, input text'e #todoText şeklinde bir ibare ekledik.
- Daha sonra buton içine de bir click eventi ekledik.
- Dedik ki butona click eventi gerçekleşince addItem() methodunu çağır, bu methoda da parameter olacak #todoText yazılı input'un value'sunu geçir, daha sonra da todoText.value'yu boş bir string'e eşitle.
- Burada addItem methodu da component class'ı içerisinde tanımlanacak. Normalde buradaki methodlara ve değişkenlere html içerisinden ulaşırken {{}} kullanıyoruz ancak, burada (click) tanımı yapıldığı için sanıyorum bu bir directive gibi alınıyor. Aynen ngFor directive'I kullanıldığında veya ngModel directive'I kullanıldığında getItems()'a veya item'a " " içerisinde ulaşabildiğimiz gibi. Bunların detaynı daha sonra göreceğiz diye umuyorum.

- addItem da component class'I içerisinde aşağıdaki gibi tanımlandı. Eğer input'dan alınan değer boş değilse items listesine yeni bir item daha ekleniyor.



- Bu ekleme yapılınca, anlık olarak liste de güncelleniyor, ve yeni eleman listemizde görünür oluyor.

PART 7 – Checkbox ile tüm elemanları göster seçeneği.

- Şuana kadar olan projede bir eleman'ın action checkbox'ı ticklenirse, ilgili item'ın action'ı true yapılıyordu ve, getItem methodu da sadece item.action'ı false olan itemları seçiyordu, bu yüzden ticklenen elemanlar listeden kayboluyordu.
- İstiyoruz ki tabloya eklenen bir checkbox ile tüm elemanları da görebilelim.
- Bunu yapmak için önce component html'e bir checkbox ekleyeceğiz, ve bu checkbox inputunu component class'ı içindeki bir flag ile double side bind edeceğiz. Böylece bu checkbox işaretliyken getItem methodu tüm item'ları return edecek, değilken filtrelenmiş itemları return edecek.
- Öncelikle component html içerisinde tablonun üstüne aşağıdaki gibi bir checkbox div'I oluşturuyorum, burada ngModel ile isDisplay özelliğinin double side bind edildiğine dikkat et.
-
```
<div class="custom-control custom-checkbox my-3">
    <input type="checkbox" [(ngModel)]="isDisplay" class="custom-control-input" name="" id="displayAll">
    <label class="custom-control-label" for="displayAll">Display All</label>
</div>
```
- Component class'ı içerisinde de aşağıdaki gibi isDisplay property'si tanımlamanın yanında getItems()'i modifiye ederek amacımıza ulaşabiliriz:
-
```
export class AppComponent {
  title = 'todoApp';
  model = new myModel();
  isDisplay = false;

  getName() {
    return this.model.user;
  }
  getItems(){
    if (this.isDisplay){
      return this.model.items;
    }
    return this.model.items.filter(item => !item.action);
```