Sanıyorum bu kısımda bir önceki word dosyasında yapılan form işlemleri farklı bir yöntemle yapılacak.

Önceki bölümde işleri daha çok template içerisinde halletmiştik burada işler daha çok component içerisinden halledilecek.

Bunun için aşağıda görüldüğü gibi çok daha sade ve temiz bir form ile başlıyoruz:

```
<form>

    <div class="form-group">
        <label>Name</label>
        <input class="form-control">
    </div>

    <div class="form-group">
        <label>Description</label>
        <input class="form-control">
    </div>

    <div class="form-group">
        <label>Price</label>
        <input class="form-control">
    </div>

    <div class="form-group">
        <label>Image Url</label>
        <input class="form-control">
    </div>

</form>
```

Reactive Forms'u kullanabilmek için öncelikle app.module içerisinde import etmeliyiz:

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

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

## 2. FormControl

Şimdi biraz önceki bölümü hatırlayalım


ngModel kullanarak form içerisindeki her bir elemana karşılık bir FormControl objesi elde ediyorduk ve validation işlemlerini bu FormControl objelerini kullanarak yapıyorduk.

Bunun yanında bir FormGroup objesi ile ise tüm form'u aynı anda kontrol edebilmek için kullanılıyordu. Hatta bu obje ile her bir input için ayrı ayrı FormControl objelerine de ulaşabiliyorduk.

Her ikisi için de template içerisinden ilgili elemana belirli directive'ler atıyorduk.

Bu kısımda ise reactive forms kavramından bahsedilecek ve daha çok component üzerinden işler halledilecek. Bu yöntem daha etkin ve kolay olacak.

Bu amaçla component içerisinde name alanı için bir FormControl objesi tanımlayalım:

```
import { NgForm, FormControl } from '@angular/forms';

@Component({
  selector: 'app',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
export class ProductComponent  {

      name = new FormControl('');

}
```

Başlangıç değeri olarak name içine boş bir string atadık. Bu  FormControl objesini oluşturduk şimdi amacımız bu objeyi, name input alanı ile çift taraflı olarak ilişkilendirebilmek.

İşte bu ilişkilendirme için aşağıdaki formControl directive'i kullanılıyor.

```
<form>
    <div class="form-group">
        <label>Name</label>
        <input class="form-control" [formControl]="name">
    </div>
```

Aslında burada yapılan  olay two way binding'e benziyor artık component içerisinde tanımlanan name değeri ile template üzerinde değer bind edildi ancak bunun yanında artık o input elemanına bir formControl objesi eklendi yani daha önce ngModel'de olduğu gibi input'un durumuna göre input elemanı ng-dirty, ng-touched gibi class değerleri alıyor veya daha önce FormControl objesi üzerinden input ile ilgili elde edilen bilgiler burada da kullanılabilecek.

Sonuçta, componentte aşağıdaki gibi 4 FormControl initialize edersek:

```
export class ProductComponent  {

   name = new FormControl('Initial Name');
   description = new FormControl('Initial Description');
   price = new FormControl('Initia lPrice');
   imageUrl = new FormControl('InitialImageUrl.jpg');

}
```

Ardından da bu değişkenleri input elemanları ile [formControl] yapısı ile bind edersek ki böylece input elamanlarına FormControl objesi de atamış olduk:

```
<form>

    <div class="form-group">
        <label>Name</label>
        <input class="form-control" [formControl]="name">
    </div>

    <div class="form-group">
        <label>Description</label>
        <input class="form-control" [formControl]="description">
    </div>

    <div class="form-group">
        <label>Price</label>
        <input class="form-control" [formControl]="price">
    </div>

    <div class="form-group">
        <label>Image Url</label>
        <input class="form-control" [formControl]="imageUrl">
    </div>

</form>
```

Aşağıdaki gibi initialize edilmiş bir template görüntüsü elde ederiz:

Burada input alanının değiştirilmesi aynı two way binding’de olduğu gibi anlık olarak component içerisindeki değeri de etkiler veya bir başka deyişle ikisi de artık aynı variable’dır.

### 3. FormGroup

Önceki kısımda gördüğümüz gibi FormControl objeleri ile tek tek ilgileniliyordu:

```
export class ProductComponent  {

   name = new FormControl('Initial Name');
   description = new FormControl('Initial Description');
   price = new FormControl('Initia lPrice');
   imageUrl = new FormControl('InitialImageUrl.jpg');

}
```

Bunun yerine tek bir FormGroup objesi oluşturup bu sadece bu obje kullanılarak da aynı işlem yapılabilirdi, zaten FormGroup objesi içerisinde her inputa karşılık bir FormControl objesi içerecektir.

Yani aynı işlemi biraz daha temiz bir yolla yapacağız:

```
export class ProductComponent  {

   productForm = new FormGroup({
      name: new FormControl('Initial Name'),
      description: new FormControl('Initial Description'),
      price: new FormControl('Initia lPrice'),
      imageUrl: new FormControl('InitialImageUrl.jpg')
   });

}
```

Görüldüğü gibi daha önce tek tek tanımlanan FormControl objeleri bu kez FormGroup içerisine bir obje içerisinde verildi.

Elbette template'de de bazı değişiklikler olacak:

```
<form [formGroup]="productForm">

    <div class="form-group">
        <label>Name</label>
        <input class="form-control" formControlName="name">
    </div>

    <div class="form-group">
        <label>Description</label>
        <input class="form-control" formControlName="description">
    </div>

    <div class="form-group">
        <label>Price</label>
        <input class="form-control" formControlName="price">
    </div>

    <div class="form-group">
        <label>Image Url</label>
        <input class="form-control" formControlName="imageUrl">
    </div>

</form>
```

Form elemanını bind etmek için [formGroup] yapısını kullanırken tek tek elemanları ilgili FormControl objelerine bind etmek için formControlName yapısını kullandık.

Şimdi forma bir submit butonu ekleyelim butona basılınca, tüm formun o anki içeriği console'da  json object olarak yazdırılsın:

```
<form [formGroup]="productForm" (ngSubmit)="onSubmit()">

    <div class="form-group">
        <label>Name</label>
        <input class="form-control" formControlName="name">
    </div>

     . . .

    <button type="submit" class="btn btn-primary">Submit</button>

</form>
```

```
export class ProductComponent  {

   productForm = new FormGroup({
      name: new FormControl('Initial Name'),
      description: new FormControl('Initial Description'),
      price: new FormControl('Initia lPrice'),
      imageUrl: new FormControl('InitialImageUrl.jpg')
   });

   onSubmit(){
      console.log(this.productForm.value)
   }

}
```

Value'lar json object olarak aşağıdaki gibi console'da görüntülenecek.

1. *{name: "Initial Name", description: "Initial Description", price: "Initia lPrice", imageUrl: "InitialImageUrl.jpg"}*
    1. description: "Initial Description"
    2. imageUrl: "InitialImageUrl.jpg"
    3. name: "Initial Name"
    4. price: "Initia lPrice"
    5. __proto__: Object

Peki component içerisinden ilgili bir input'un içeriğini değiştirmek istersem?

Componente aşağıdaki methodu eklerim, productForm ismindeki FormGroup objesinin içindeki değeri/değerleri update etmek için patchValue methodunu aşağıdaki gibi kullanabilirim.

```
  updateProduct(){
    this.productForm.patchValue({
      name: 'Iphone X',
      price: 10000
    })
  }
```

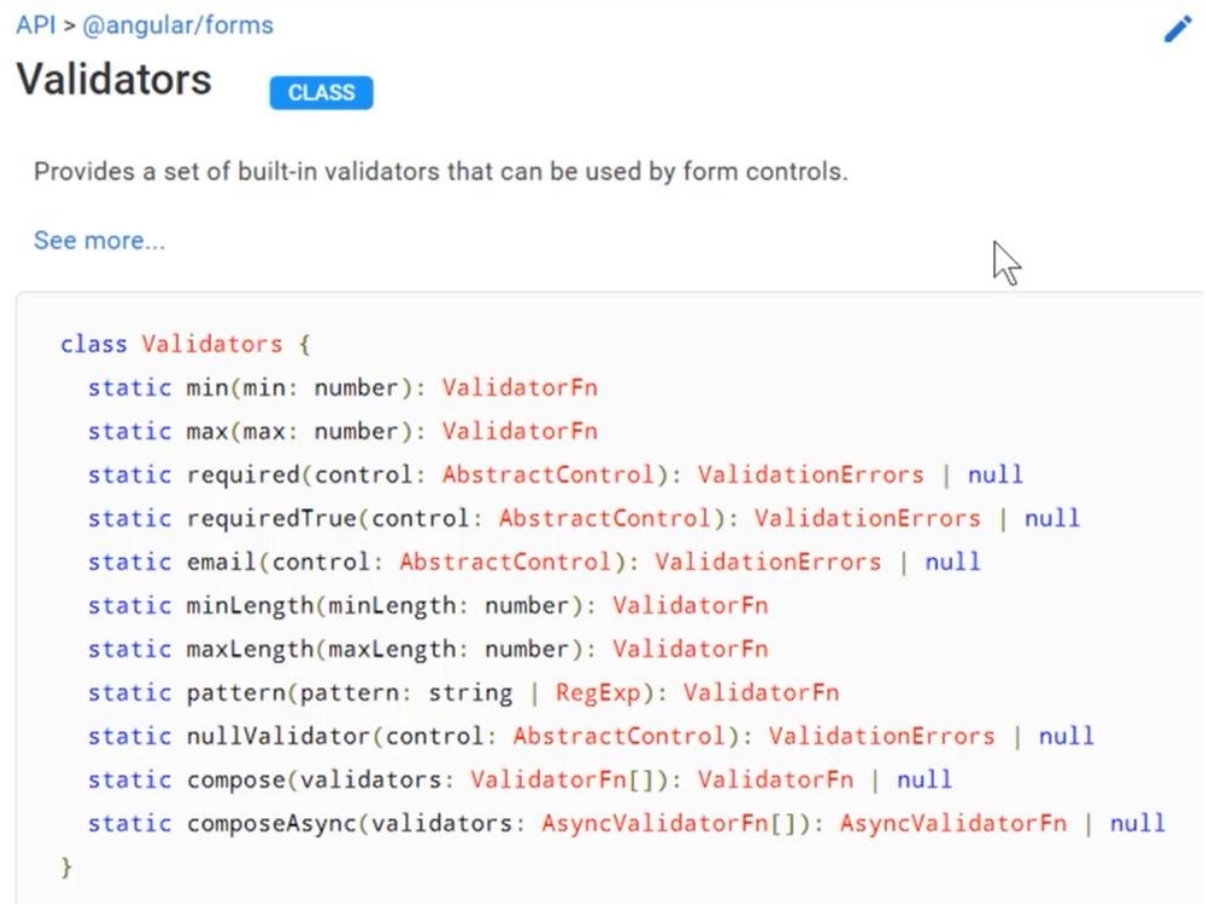
Bu method click eventi ile çağırılsın istersek, butona basıldığında ilgili 2 input component içerisinden atanan yeni değerlerini alacaktır.

```
<button (click)="updateProduct()" type="submit" class="btn btn-
primary">Submit</button>
```

## 4. Reactive Form ile Validation

Şimdi FormControl objelerini initialize ederken Validators class'ını kullanacağım böylece daha önce template içine eklenen required, minlength, maxlength, pattern gibi ibareleri component içerisinde halledeceğim.

Şimdi Angular'ın websitesinden Validators class'ına bakalım, aşağıdaki built-in validators'ı görebiliyoruz. Aşağıdaki static methodlara Validators class'ı üzerinden ulaşabiliriz.




Bunların dışında bir method kullanmak istersek custom validator kullanmamız gerek.

Bunu bu kısmın son dersinde göreceğiz. Şimdilik yukarıdaki static methodları kullanarak form validation yapalım.

Component içerisinde FormControl objelerinin initial değerini boş string'e atadık ve aşağıdaki gibi validators class'ı üzerinden static validation methodlarını kullandık:

```
@Component({
  selector: 'app',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
export class ProductComponent  {

   productForm = new FormGroup({
     name: new FormControl('', [Validators.required,Validators.minLength(5)]),
     description: new FormControl('', Validators.required),
     price: new FormControl('', Validators.required),
     imageUrl: new FormControl('', Validators.required)
   });
```

Buna karşılık template içerisinde yeni hiçbir şey yapmamıza gerek yok,

```
<form [formGroup]="productForm" (ngSubmit)="onSubmit()">

    <div class="form-group">
        <label>Name</label>
        <input class="form-control" formControlName="name">
    </div>

    <div class="form-group">
        <label>Description</label>
        <input class="form-control" formControlName="description">
    </div>
    . . .

    <button (click)="updateProduct()" type="submit" class="btn btn-
primary">Submit</button>

</form>
```

Sadece formün input elemanlarına gerekli FormControl objelerinin atanmış olması ilgili validator'ları direkt ilgili elemana taşıyor.

Peki buradaki validator'ların şuan işlevi ne? Bunların işlevi şuanda ilgili input'a template üzerinde ng-invalid class'ının atanıp atanmayacağını belirler. Ng-touched, ng-dirty gibi formControl objesi tarafından sağlanan class'ların yanında ng-invalid class'ı belirtilen validation kurallarına göre sınıfa atanır veya atanmaz.

***<input _ngcontent-eab-c0="" class="form-control ng-touched ng-dirty ng-invalid" formcontrolname="name" ng-reflect-name="name">***

Peki bu class atandığında ne oluyor? Css kodları sayesinde, ilgili eleman touched ve invalid class'larına sahipse ise kırmızı border'a sahip olsun diyoruz mesela.

İstersek yine bu class'ları kullanarak template içerisinde hata mesajı da oluşturabiliriz:

```
<form [formGroup]="productForm" (ngSubmit)="onSubmit()">

    <div class="form-group">
        <label>Name</label>
        <input class="form-control" formControlName="name">
        <div *ngIf="productForm.get('name').touched && productForm.get('name')
.invalid" class="alert alert-danger">
            Name is required
        </div>
    </div>
```

Burada eklenen ngIf'li div in koşulu doğru ise yani ilgili formControl objesi touched ve invalid ise Name is required yazısı görünsün istedik.

Burada ilgili formcontrol objesine ulaşmak için FormGroup objesinin get methodunu kullandığımıza dikkat et.

Ancak burada belirtilen hata mesajı invalid olduğu her durumda gelecek, karakter yanlış girildiği için mi invalid oldu yoksa karakter sayısı az mı, fazla mı bu bilgiyi şuan kullanmadık.

Önümüzdeki kısımda bu şekilde özelleşmiş hata mesajlarını nasıl verebileceğimize bakalım.

### 4. Reactive Form ile Özelleşmiş Hata Mesajları

Aşağıdaki gibi bir liste ile her hata durumunu ilgili FormControl objesinden alınan o anki hata bilgisini kullanarak test ederiz ve buna göre hata mesajlarınını gösterebiliriz.

```
<form [formGroup]="productForm" (ngSubmit)="onSubmit()">

    <div class="form-group">
        <label>Name</label>
        <input class="form-control" formControlName="name">
        <div *ngIf="productForm.get('name').touched && productForm.get('name')
.invalid" class="alert alert-danger">
            <ul>
                <li *ngIf="productForm.get('name').errors.required"> Name is r
equired </li>
                <li *ngIf="productForm.get('name').errors.minLength"> Min 3 ch
aracters </li>
                <li *ngIf="productForm.get('name').errors.maxlength"> Max 10 c
haracters </li>
            </ul>
        </div>
    </div>
```

## 5. Custom Validation Kuralı Oluşturma

Şimdi diyelim ki formumuz içindeki ImageUrl inputuna .jpg veya .jpeg uzantılı dosyaları kabul edelim yok eğer bunun dışında bir uzantı girilmişse kullanıcıya hata gösterelim.

Bu validation kuralı şuanda Validators class'ı içerisinde yer almıyor. Bu yüzden bu validator kuralını kendimiz tanımlamalıyız.

Custom Validation kuralı eklerken kullanmamız gereken bir interface yapısı var:


Şimdi app klasörü altına bir image.validators.ts file'ı ekliyoruz, içini de aşağıdaki gibi dolduruyoruz.

```
import { AbstractControl, ValidationErrors} from '@angular/forms'

export class ImageValidators {
    static isValidExtension (control: AbstractControl): ValidationErrors | nul
l {
        const v = control.value as string
        if(v.endsWith('.jpg') || v.endsWith('jpeg') || v.endsWith('.png') ){
            return null;
        }else {
            return {
                wrongExtension: true
            }
        }
    }
}
```

Özetle dendi ki validator'a gelen control objesi string olarak alınsın ki bu sanıyorum input'un içeriği olacak, bu içerik .jpg .jpeg veya .png ile bitiyorsa method null döndürsün yok bunlarla bitmiyorsa wrongExtension:true olsun, yani aynı required: true veya minlength: true olduğu gibi bunun adı da wrongExtension olacak.

Şimdi component içerisinde bu wrongExtension kuralını ilgili imageUrl input'una tanımlayalım:

```
import { ImageValidators } from '../image.validators';

@Component({
  selector: 'app',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
export class ProductComponent  {

   productForm = new FormGroup({
      name: new FormControl('', [Validators.required,Validators.minLength(3),V
alidators.maxLength(10)]),
      description: new FormControl('', Validators.required),
      price: new FormControl('', Validators.required),
      imageUrl: new FormControl('',[Validators.required,ImageValidators.isVali
dExtension])
   });
```

Son olarak da template içerisinde ilgili FormControl objesinin içinden erorrs kısmından bu kuralın ihlal edilip edilmediğini aynı diğer kurallar gibi öğrenebiliriz:

```
    <div class="form-group">
        <label>Image Url</label>
        <input class="form-control" formControlName="imageUrl">
        <div *ngIf="productForm.get('imageUrl').touched && productForm.get('im
ageUrl').invalid" class="alert alert-danger">
            <ul>
                <li *ngIf="productForm.get('imageUrl').errors.required"> Name
is required </li>
                <li *ngIf="productForm.get('imageUrl').errors.wrongExtension">
 Wrong extension </li>
            </ul>
        </div>
    </div>
```