Bu kısımda form işlemleri ile uğraşacağız sanıyorum. Çok fazla detayına girmek istemiyorum, zaten gerekirse izleyebilirsin.

Ancak yine de seri notlar düşmek istedim. Bu yüzden belki sadece kodlar belki küçük açıklamalarla birlikte kodları paylaşacağım.

1. **Formun Hazırlanması**

Product.component.html içerisinde bir form yaracağız.

```
export class ProductComponent  {

 model: ProductRepository = new ProductRepository();

 newProduct: Product = new Product();

 get jsonProduct(){
    return JSON.stringify(this.newProduct)
 }

 //Component'in addProduct methodu şuan sadece newProduct'ı jsonObject olarak
konsolda yazdırıyor.
 addProduct( p: Product) {
   console.log("New Product: "+ this.jsonProduct)
 }

}
```

```
<div class="container">
    <div class="row">
        <div class="col-8">
            <div class="m-3">
                <div class="bg-info text-white m-2 p-2">
                    {{jsonProduct}}
                </div>

                <div class="form-group">
                    <label>Name</label>
                    <input type="text" [(ngModel)]="newProduct.name" class="fo
rm-control">
                </div>

                <div class="form-group">
                    <label>Description</label>
                    <input type="text" [(ngModel)]="newProduct.description" cl
ass="form-control">
```

```
                </div>

                <div class="form-group">
                    <label>Price</label>
                    <input type="text" [(ngModel)]="newProduct.price" class="f
orm-control">
                </div>

                <div class="form-group">
                    <label>Image Url</label>
                    <input type="text" [(ngModel)]="newProduct.imageUrl" class
="form-control">
                </div>

                <button (click)="addProduct(newProduct)"    class="btn btn-
primary">Submit</button>

            </div>
        </div>
    </div>
</div>
```

Component içerisinde oluşturulan newProduct ngModel yardımı ile two way bind edildi, ayrıca component içerisindeki jsonProduct() methodu ile ilgili objeyi json object'e çevirebiliyoruz.

## 2. NgModel ile Validation

Bu kısımda inputlara validation kontrolü eklenecek.

Şimdiye kadar NgModel'i sadece two way binding için kullanmıştık fakat bundan başka amaçları da var, bir form elemanına atanarak, o elemana ait bir form control object oluşturmaya yarıyor, böylece form elemanı ile ilgili bir çok bilgili bu form control object üzerinden elde edebiliyoruz.

Bunu nasıl yaparız?

```
<input required type="text" [(ngModel)]="newProduct.name"  name="pName" #name=
"ngModel" (change)="log(name)" class="form-control">
```

Burada required kısmı html özelliği, bunun yanında minlength, maxlength, pattern gibi html özellikleri de eklenebilir.

Burada template reference variable'ı sanırım ngModel directive'ine eşitledik daha sonra bu template variable'ı change event'i olunca component'e yolluyorum, yani ngModel'i component'e yollamış oluyorum. Böylece istediğimiz form control object'e ulaşmış oluyoruz.

Artık yukarıdaki input'da change event'i occur olunca ilgili NgModel componente gönderiliyor bunu console'a yazarsak:

1. NgModel
    1. asyncValidator: (...)
    2. control: FormControl
        1. asyncValidator: null
        2. dirty: (...)
        3. disabled: (...)
        4. enabled: (...)
        5. errors: null
        6. invalid: (...)
        7. parent: (...)
        8. pending: (...)
        9. pristine: false
        10. root: (...)
        11. status: "VALID"
        12. statusChanges: EventEmitter {_isScalar: false, observers: Array(0), closed: f
            alse, isStopped: false, hasError: false, …}
        13. touched: true
        14. untouched: (...)
        15. updateOn: (...)
        16. valid: (...)
        17. validator: *ƒ (control)*
        18. value: "asd"
        19. valueChanges: EventEmitter {_isScalar: false, observers: Array(0), closed: fa
            lse, isStopped: false, hasError: false, …}
        20. _onChange: [*ƒ*]
        21. _onCollectionChange: *ƒ ()*
        22. _onDisabledChange: [*ƒ*]
        23. _pendingChange: false
        24. _pendingDirty: true

```
        25. _pendingTouched: true
        26. _pendingValue: "asd"
        27. __proto__: AbstractControl
    3.  dirty: (...)
    4.  disabled: (...)
    5.  enabled: (...)
    6.  errors: (...)
    7.  formDirective: (...)
    8.  invalid: (...)
    9.  model: "asd"
    10. name: "pName"
    11. path: (...)
    12. pending: (...)
    13. pristine: (...)
    14. status: (...)
    15. statusChanges: (...)
    16. touched: (...)
    17. untouched: (...)
    18. update: EventEmitter {_isScalar: false, observers: Array(1), closed: false, isStoppe
        d: false, hasError: false, …}
    19. valid: (...)
    20. validator: (...)
    21. value: (...)
    22. valueAccessor: DefaultValueAccessor {_renderer: DebugRenderer2, _elementRef: Element
        Ref, _compositionMode: true, onChange: ƒ, onTouched: ƒ, …}
    23. valueChanges: (...)
    24. viewModel: "asd"
    25. _parent: null
    26. _rawAsyncValidators: []
    27. _rawValidators: [RequiredValidator]
    28. _registered: true
    29. __proto__: NgControl
```

Burada control: FormControl altında input elemanı ile ilgili bilgiler yer alıyor. Mesela dirty: true olacak yani inputa değer girildi, pristine: false (dirty'nin tersi). Valid: True çünkü bu elemana değer girilmesi gerekiyordu, girildi.

### 3. Form Control Objesini Kullanarak Validation Kuralı Oluşturmak

Form kontrol objesini component'e aktararak ilgili formla ilgili bir çok bilgiyi elde edebileceğimizi görmüştük. Bu verileri kullanarak validation kuralları oluşturabiliriz.

Anladık ki ngModel directive'i ile yalnızca two way binding yapmıyoruz, bu directive'in başka işlevleri de var, atanadığı form elemanı ile ilgili farklı bilgiler tutan bir Form Control Objesi NgModel içerisinde tutuluyor.

Biz bu objeyi geçen sefer component'e gönderip console'da yazdırmıştık şuan buna gerek yok o kısımları sildik.

Şimdi ngModel atanmış bir input elemana inspector ile bakıyoru gördüğüm input elemanı aşağıdaki gibi:

<input _ngcontent-wol-c0="" class="form-control **ng-pristine ng-invalid ng-touched**" name="pName" required="" type="text" ng-reflect-required="" ng-reflect-name="pName">

Görüldüğü gibi, bu elemana [(ngModel)] ibaresi eklendiği için, yukarıdaki gibi form ile ilgili bilgi veren bazı class'lar input elemanına dahil edilmiş. Bu class'lar dinamiktir, yani şuan eleman boş olduğu için ng-pristine der eğer yazı yazsak ng-dirty olarak değişir, benzer şekilde, elemana dokunursak ng-untouched -> ng-touched olarak değişir vesaire.

İşte elemanın bu class'larını kullanarak, css ile kullanıcıyı bilgilendirmek adına style ekleyebiliriz. Sıkıntı varsa elemana kırmızılık veya tamamsa yeşillik vesaire gibi.

Bu amaçla, assets altında forms.css dosyası yarattık ve bu dosyayı index.html içerisinde çağırdık

```
<link rel="stylesheet" href="assets/forms.css">
```

Böylece index üzerindeki css ile işlemlerimizi yapacağız.

```
.ng-touched.ng-invalid {
    border-left:5px solid #a94442;
}

.ng-dirty.ng-valid {
    border-left:5px solid #42A948;
}
```

Böylece class touched ve invalid ise kırmızı bir border olsun vb. dedik.

## 4. Validation Mesajlarının Gösterilmesi

Input elemanına eklenen ngModel, elemana dinamik class'lar kazandırmıştı ve bu class'ları kullanarak css kodları ile input elemanına kullanıcı bilgilendirme amacıyla style verebileceğimizi görmüştük.

Şimdi kullanıcıyı validation mesajlarını nasıl göstereceğimize bakalım. Bunu bootstrap class'ları ile yapalım.

```
                <div class="form-group">
                    <label>Name</label>
                    <input required type="text" [(ngModel)]="newProduct.name"
 name="pName" #name="ngModel" (change)="log(name)" class="form-control">
                    <div class="alert alert-
danger" *ngIf="!name.valid && name.touched">
                        Name is required.
                    </div>
                </div>

                <div class="form-group">
                    <label>Description</label>
                    <input required type="text" [(ngModel)]="newProduct.descri
ption" #description="ngModel" class="form-control">
                    <div class="alert alert-
danger" *ngIf="!description.valid && description.touched">
                        Description is required.
                    </div>
                </div>
```

Burada yapılan basitçe bootstrap ile bir hata mesajı oluşturmak oldu.

Ancak bu hata mesajını belirli durumlarda kullanmak için *ngIf kullandık. Burada baktığımız şey name yani ngModel objesi üzerinden ilgili elemanın touched ve invalid olması bu durumda hata mesajı gösterilir.

Diğer bir input için de benzer şekilde kendi ngModel'i üzerinden bakılır bu ngModel'e de #description olarak tanımlanan template değişkeni üzerinden erişilir.

## 5. Özelleştirilmiş Hata Mesajları

Önceki kısımda formlara required özelliği eklenmişti, bu özellik html özelliği, bunun yanında minlength, maxlength veya pattern gibi farklı özelliklerin de elemana eklenebileceğini biliyoruz.

Geçen kısımda biz ngModel'i template variable'a atamıştık ve ngIf içerisinde bu variable aracılığı ile ngModel özelliklerine göre bir hata div'ini gösterdik veya göstermedik.

Şimdi ise elemana required yanında bahsedilen diğer özellikleri de ekleyeceğiz ve hangisinin ihlal edildiğini kullanıcıya yine ngModel form control object'i kullanarak bildireceğiz.

```
<input required  minlength="3" maxlength="10" pattern="^[A-Za-
z ]+$"  type="text" [(ngModel)]="newProduct.name"  name="pName" #name="ngModel
" (change)="log(name)" class="form-control">
<div class="alert alert-danger" *ngIf="!name.valid && name.touched">
    Error
</div>
```

Görüldüğü gibi required yanında 3 yeni özellik eklendi eğer bunlar gerçeklenmezse form control objesi invalid döndürecek ve alert içinde Error yazacak.

Pattern'ın regular expression ile yazıldığına dikkat et.

Peki şimdi şunu yapalım, hangi hatanın döndürüldüğüne göre kullanıcıya farklı error mesajları gösterelim. Hangi hatanın döndüğüne name.error üzerinden yani yine ngModel objesi üzerinden ulaşabiliriz:

```
<input required  minlength="3" maxlength="10" pattern="^[A-Za-
z ]+$"  type="text" [(ngModel)]="newProduct.name"  name="pName" #name="ngModel
" (change)="log(name)" class="form-control">
<div class="alert alert-danger" *ngIf="!name.valid && name.touched">
    <p *ngIf="name.errors.required">Name is required</p>
    <p *ngIf="name.errors.pattern">Only letters and spaces</p>
    <p *ngIf="name.errors.minlength">Min 3 characters</p>
</div>
```

## 6. Özelleştirilmiş Hata Mesajlarını Metot ile Gösterelim

Bir önceki kısımda farklı hatalara göre farklı hata mesajlarını kullanıcıya gösterdik. Bu işlem tamamen template üzerinde gerçekleşti.

Şimdi aynı işlemi component methodu aracılığı ile yapalım.

Öncelikle template içindeki kısmı aşağıdaki gibi değiştirioruz, burada yapılan şey şu eğer input invalid ise ve toudched ise ki bu bilgi hatırlarsan name'e atanan ngModel ile alınıyordu, alert-danger'a sahip hata div'i görünsün.

```
<div class="form-group">
    <label>Name</label>
    <input required  minlength="3" maxlength="10" pattern="^[A-Za-
z ]+$"  type="text" [(ngModel)]="newProduct.name"  name="pName" #name="ngModel
" (change)="log(name)" class="form-control">
    <div class="alert alert-danger" *ngIf="!name.valid && name.touched">
        <p *ngFor="let error of getValidationErrors(name)">
            {{error}}
        </p>
    </div>
</div>
```

Burada bir p etiketi içerisinde hata mesajları gösterilsin dedik. Bu hata mesajları ile getValidationErrors ismindeki method tarafından üretilecek methoda giren parametre ise ngModel objesi.

```
 getValidationErrors(state: any){
   let ctrlName: string = state.name;
   let messages: string[] = [];

   if(state.errors) {
      for (let errorName in state.errors) {
         switch(errorName){
            case "required":
               messages.push(`You must enter a ${ctrlName}`);
               break;

            case "minlength":
               messages.push(`Min. 3 characters for ${ctrlName}`);
               break;

            case "pattern":
               messages.push(`contains illegal characters ${ctrlName}`);
               break;
         }
      }
   }
   return messages;
```

```
}
```

Component içerisindeki getValidationErrors methodu yukarıda verilmiş, yaptığı işlem temelde bir messages değişkeninin içini aldığı ngModel objesinin içeriğine göre doldurmak. Daha sonra da bu messages'ı return etmek.

Return edilen mesajlar da template üzerinde yazdırılıyor.

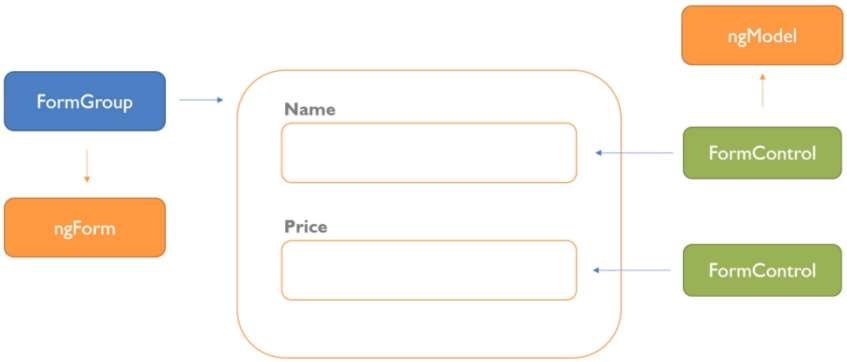
Sonuçta bu methodu kullanarak template'de daha temiz şekilde hata mesajları verebiliriz.

## 7. NgForm ile Validation

Önceki kısımlarda input elemanı bazında validation işlemleri yaptık, her bir ayrı input için ayrı bir ngModel tanımladık ve bu obje yardımıyla ilgili input elemanını baz alarak hata mesajlarını kullanıcıya gösterdik.

Şimdi ise ngForm ile form seviyesinde bir validation yapmak istiyoruz. Bu form kontrol işlemi kullanıcı submit butonuna tıkladıktan sonra yapılır. Yani form servere gönderilmeden hemen önce kontrol yapılır.

Mesela aşağıdaki gibi bir form için dah önce yaptığımız işlem her bir input elemanı için ayrı bir ngModel tanımlaması yapıyorduk, bu sayede her elemana kendine ait bir FormControl objesi tanımlamış oluyorduk. Bu FormControl objesi yardımıyla alana girilen bilgiyi veya bu bilginin istenen kriterlere uyup uymadığını tespit edebiliyorduk.




Şimdi ise form seviyesinde bir object tanımlanacak buna da FormGroup objesi deniyor analoji olarak FormControl objesine denk geliyor.

Bu FormGroup objesi üzerinden form bazından kontroller gerçekleştirilebilir. Bu objeyi tanımlamak için ise form elemanına ngForm tanımlamasını yapmamız gerek. Aslında tam olarak ngModel gibi olmuyor sadece #variable=”ngForm” dediğimizde form’a ngForm’u atamış oluyoruz.

Bu amaçla daha önceki input elemanlarını bir form etiketi altına taşıyoruz:

```
<form #form="ngForm" (ngSubmit)="submitForm(form)">
```

Yukarıdaki gibi bu form elemanı içerisinde form adında bir template variable tanımladık ve buna ngForm'u atadık daha sonra form submit edilince çalışacak bir event ile method tanımladık ve bu methoda ngForm'u aktardık.

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

Submit işlemi form içerisine konulan yukarıdaki gibi bir butonla gerçekleşiyor.

submitForm da aşağıdaki gibi basitçe elde edilen ngForm'u ekrana yazdırmaya yarasın.

```
submitForm(form){
   console.log(form);
}
```

Artık bu form elemanına da bir bütün olarak daha önce input elemanlarına eklenen ng-touched, ng-dirty vb gibi özellikler atanıyor bu yüzden eğer daha önce tanımlanan css etiketlerinin sadece input elemanları bazında çalışmasını istersek, başlarına .form-control ekliyoruz ki sadece form-control class'ına sahip olan elemanlarda yani tekil input elemanlarında çalıssın.

```
.form-control.ng-touched.ng-invalid {
    border-left:5px solid #a94442;
}

.form-control.ng-dirty.ng-valid {
    border-left:5px solid #42A948;
}
```

İşin özü artık form elemanına ngForm atandığı için formGroup objesi üzerinden ilgili form'un ayrı ayrı value'larına veya state'lerine ulaşabilirim, mantık aynı tekil input elemanlarında ngModel ile yapılana benziyor, bi fark yok.

## 8. NgForm ile Validation Kural Ekleme

```
formSubmitted: boolean = false;

submitForm(form: NgForm){
   this.formSubmitted = true;
   if(form.valid){
      this.addProduct(this.newProduct);
      this.newProduct = new Product();
      form.reset()
      this.formSubmitted = false;
   }
}
```

```
<form #form="ngForm" (ngSubmit)="submitForm(form)">

    <div class="bg-danger text-white p-3 my-
2" *ngIf="formSubmitted && form.invalid">
        An error occured.
    </div>

    <div class="form-group">
        <label>Name</label>
        <input required  minlength="3" maxlength="10" pattern="^[A-Za-
z ]+$"  type="text" [(ngModel)]="newProduct.name"  name="pName" #name="ngModel
" (change)="log(name)" class="form-control">
        <div class="alert alert-
danger" *ngIf="(formSubmitted || name.dirty) && name.invalid">
            <p *ngFor="let error of getValidationErrors(name)">
                {{error}}
            </p>
        </div>
    </div>
```

Yukarıdaki gibi ngForm üzerinden işlemler yapıp form kontrolleri yapabilir ve kullanıcıya hata mesajları verebiliriz, gerekirse aç bak.

## 9. Özet Mesajların Gösterilmesi

Bu ve bir sonraki kısmı atlıyorum, genel mantığı anladığımızı düşünüyorum, şuan baksam da bir anlamı yok, daha sonra gerekirse bakmakta fayda var.