



IT255 - VEB SISTEMI 1

Ugrađene directive i rad sa formama

Lekcija 07

PRIRUČNIK ZA STUDENTE

IT255 - VEB SISTEMI 1

Lekcija 07

UGRAĐENE DIRECTIVE I RAD SA FORMAMA

- ✓ Ugrađene directive i rad sa formama
- ✓ Poglavlje 1: Direktive grananja
- ✓ Poglavlje 2: Direktiva stila u Angularu
- ✓ Poglavlje 3: Direktiva klasa u Angularu
- ✓ Poglavlje 4: Direktive petlji u Angularu
- ✓ Poglavlje 5: Direktiva ngNonBindable
- ✓ Poglavlje 6: Forme u Angular okviru
- ✓ Poglavlje 7: Vežba 7
- ✓ Poglavlje 8: Domaći zadatak 7
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Direktive predstavljaju attribute koji se dodaju u HTML elemente i daju dinamičko ponašanje.

Angular okvir obezbeđuje veliki broj ugrađenih direktiva, koje predstavljaju attribute koji se dodaju u *HTML* elemente i na taj način kreiraju dinamičko ponašanje Angular veb aplikacije. U ovoj lekciji je, upravo, cilj pokrivanje najčešće korišćenih ugrađenih direktiva kroz detaljnu analizu i diskusiju vođenu primenom adekvatnog primera.

Savladavanjem ove lekcije, student će biti osposobljen da koristi osnovne ugrađene *Angular direktive* i da prvi put kreira dinamičku Angular veb aplikaciju.

Prateći primer, koji objedinjuje primenu navedenih direktiva, priložen je na kraju lekcije. Studenti mogu na ga preuzmu, otvore u razvojnom okruženju i kucanjem naredbi:

```
npm install  
npm start
```

mogu da pokrenu i testiraju urađenu aplikaciju.

PREPORUKA:

Pokušajte sami da uradite aplikaciju, prateći izlaganje lekcije, pre nego što testirate već urađen program. Iz programa samo preuzmite slike i stilove.

▼ Poglavlje 1

Direktive grananja

DIREKTIVA NGIF

Direktiva je naredba koja je umetnuta u HTML kod i obavlja neke specifične zadatke.

U ovom delu lekcije cilj je učenje primene grananja u [HTML](#) sekcijama [Angular](#) aplikacije. [Direktiva](#) je nije ništa drugo do naredba koja je umetnuta u HTML kod i obavlja neke specifične zadatke. Zadaci grananja, u Angular aplikacijama, realizuju se primenom jedne od sledeće dve direktive:

- [ngIf](#);
- [ngSwitch](#).

Prva na red za razmatranje dolazi direktiva [ngIf](#). Navedena direktiva se koristi kada je neophodno prikazati ili sakriti neki sadržaj u zavisnosti od ispunjenosti određenog logičkog uslova. Uslov je određen rezultatom nekog logičkog izraza koji se prosleđuje u direktivu.

Ukoliko izraz vrati vrednost [true](#), određeni element (ili sadržaj) biće prikazan na veb stranici. U suprotnom, ukoliko izraz vrati vrednost [false](#), navedeni element će biti uklonjen iz [DOM](#) ([Document Object Model](#)) - https://www.w3schools.com/whatis/whatis_htmlDOM.asp.

Sledećim listingom je data ilustracija primene direktive [ngIf](#).

```
<div *ngIf="false"></div> <!-- ne prikazuje se nikada -->
<div *ngIf="a > b"></div> <!-- prikazuje se ako je a veće od b -->
<div *ngIf="str == 'yes'"></div> <!-- prikazuje se ako str ima vrednost "yes" -->
<div *ngIf="myFunc()"></div> <!-- prikazuje se ako myFunc vraća true -->
```

Prethodnim listingom je prikazano nekoliko različitih scenarija upotrebe ove direktive i velika je šansa da ćete prilikom kreiranja vaših veb aplikacija, upravo, koristiti ovu direktivu u nekom od navedenih oblika.

DIREKTIVA NGSWITCH

Preko [ngSwitch](#) je moguće prikazati neke elemente u zavisnosti od konkretne vrednosti nekog izraza

Naredbe [if](#) i [switch](#) su već rađene u predmetima [CS101](#) i [CS102](#). Logika direktiva [ngIf](#) i [ngSwitch](#) potpuno je identična. Ponekad je, u HTML pogledima veb aplikacije, [neophodno](#)

prikazati neke elemente u zavisnosti od konkretne vrednosti nekog izraza. Posmatra se primer priložen sledećim listingom:

```
<div class="container">
<div *ngIf="myVar == 'A'">Var je A</div>
<div *ngIf="myVar == 'B'">Var je B</div>
<div *ngIf="myVar != 'A' && myVar != 'B'">Var je nešto treće</div>
</div>
```

Kao što je moguće primetiti, ovde je na delu primer višestrukog grananja. Program se različito ponaša ukoliko je vrednost varijable *myVar*: A, B ili nešto treće. Kompleksnost višestrukog grananja se povećava ukoliko je neophodno rukovati novom vrednošću, na primer C. Sa ciljem realizacije navedenog, nije dovoljno samo dodati nov element u *ngIf*, već je potrebno i poslednji slučaj (*case*) proširiti. Navedeno je moguće ilustrovati sledećim listingom:

```
<div class="container">
<div *ngIf="myVar == 'A'">Var je A</div>
<div *ngIf="myVar == 'B'">Var je B</div>
<div *ngIf="myVar == 'C'">Var je C</div>
<div *ngIf="myVar != 'A' && myVar != 'B' && myVar != 'C'">Var je nešto četvrto</div>
</div>
```

Za slučajeve kao što je ovaj, Angular omogućava primenu direktive *ngSwitch*. Prednost primene ove direktive ogleda se u činjenici da se vrednost izraza proverava samo jedanput, a potom se prikazuje element na osnovu vrednosti koja je rezultat tačne provere.

Kada je dobijen rezultat izraza, moguće je postupiti na sledeći način:

- izvršava se neka od direktiva *ngSwitchCase* koja odgovara vrednosti izračunatog izraza;
- izvršava se *ngSwitchDefault*, ukoliko nijedan *ngSwitchCase* nije uparen sa izračunatom vrednošću izraza.

Na lakši, i lepši način, moguće je napisati kod koji je ekvivalentan kodu iz prethodnog listinga.

```
<div class="container" [ngSwitch]="myVar">
<div *ngSwitchCase="'A'">Var je A</div>
<div *ngSwitchCase="'B'">Var je B</div>
<div *ngSwitchCase="'C'">Var je C</div>
<div *ngSwitchDefault>Var je nešto četvrto</div>
</div>
```

Kao što je bio slučaj u programskom jeziku Java, *ngSwitchDefault* je opcionalan element. Ukoliko je izostavljen, ništa neće biti prikazano ukoliko *myVar* ne bude uparena sa odgovarajućom vrednošću iz neke od *ngSwitchCase* naredbe.

ALTERNATIVNA PRIMENA DIREKTIVE NGSWITCHCASE

Primenu naredbe `ngSwitchCase` moguće je vezati i za vrednosti različitih elemenata.

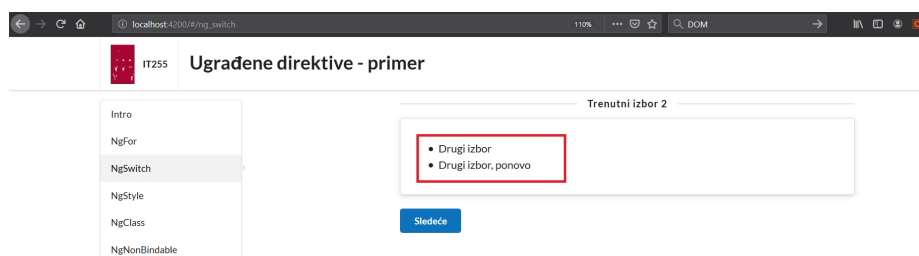
Primenu naredbe `*ngSwitchCase` moguće je vezati i za vrednosti različitih elemenata tako da ona nije ograničena na proveru podudaranja samo jednom. Navedeno je moguće ilustrovati sledećim listingom:

```
<h4 class="ui horizontal divider header">
  Trenutni izbor {{ choice }}
</h4>

<div class="ui raised segment">
  <ul [ngSwitch]="choice">
    <li *ngSwitchCase="1">Prvi izbor</li>
    <li *ngSwitchCase="2">Drugi izbor</li>
    <li *ngSwitchCase="3">Treći izbor</li>
    <li *ngSwitchCase="4">Četvrti izbor</li>
    <li *ngSwitchCase="2">Drugi izbor, ponovo</li>
    <li *ngSwitchDefault>Podrazumevani izbor</li>
  </ul>
</div>

<div style="margin-top: 20px;">
  <button class="ui primary button" (click)="nextChoice()">
    Sledeće
  </button>
</div>
```

U priloženom kodu neophodno je, posebno, pogledati linije koda 8 i 11. Budući da se odnose na istu vrednost (`choice == 2`) u slučaju pozitivnog podudaranja oba elementa će biti prikazana. To je moguće videti na sledećoj slici koja predstavlja rezultat izvršavanja Angular aplikacije koja će biti kreirana kao rezultat rada na ovoj lekciji.



Slika 1.1 Dvostruko podudaranje sa `ngSwitchCase`

▼ Poglavlje 2

Direktiva stila u Angularu

DIREKTIVA NGSTYLE

Direktivom `ngStyle` omogućeno je podešavanje CSS osobine DOM elementa iz Angular izraza

Primenom direktive `ngStyle` omogućeno je podešavanje `CSS` osobine `DOM` elementa iz `Angular` izraza.

Najjednostavniji način da se upotrebi ova direktiva jeste pisanje izraza u sledećem opštem obliku:

```
[style.<cssproperty>]="value"
```

Na primer:

```
<div [style.background-color]='yellow'>  
  Koristi fiksnu žutu pozadinu  
</div>
```

Navedeni listing koristi direktivu `ngStyle` za podešavanje `CSS` osobine `background-color` preko string literala `"yellow"`.

Postoje i drugi način za podešavanje fiksnih vrednosti, na primer, primenom atributa direktive `ngStyle` i parova (ključ, vrednost) za svaku od osobina koju je neophodno podesiti. Navedeno je moguće realizovati na sledeći način:

```
<div [ngStyle]="{color: 'white', 'background-color': 'blue'}">  
  Koristi fiksni beli tekst na plavoj pozadini  
</div>
```

Iz priloženog listinga je moguće zaključiti da je obavljeno istovremeno podešavanje osobina: `color` i `background-color`, respektivno. Međutim, najveća snaga primene direktive `ngStyle` leži u radu sa dinamičkim HTML elementima.

PRIMENA DINAMIČKIH VREDNOSTI

Najveća snaga primene direktive `ngStyle` leži u radu sa dinamičkim HTML elementima.

Kao što je upravo konstatovano, najveća snaga primene direktive `ngStyle` leži u radu sa dinamičkim HTML elementima. U aktuelnom primeru uvodi se dodatna funkcionalnost. Postoje dva polja za unos teksta, u kojima se unose željena boja i veličina fonta, i dugme kojim se potvrđuju nova podešavanja. U priloženom primeru, ovaj listing, kao i prethodni, uzet je iz datoteke: <src/app/ng-style-example/ng-style-example.component.html>.

```
<div class="ui input">
  <input type="text" name="color" value="{{color}}" #colorinput>
</div>

<div class="ui input">
  <input type="text" name="fontSize" value="{{fontSize}}" #fontinput>
</div>

<button class="ui primary button" (click)="apply(colorinput.value,
fontinput.value)">
  Primeni podešavanja
</button>
```

Klikom na kreirano dugme menjaju se CSS osobine navedenih za boju u veličinu fonta. Ove osobine su bile inicijalno podešene sledećim kodom:

```
<div>
  <span [ngStyle]="{color: 'red'}" [style.font-size.px]="fontSize">
    crveni tekst
  </span>
</div>
```

Važno je napomenuti da je obavezno specificiranje veličina gde je to potrebno. Na primer, nije pravilno podesiti CSS osobinu `font-size` na vrednost 12. Ona mora biti izražena u jedinicama poput 12px ili 1.2em. Angular obezbeđuje jednostavnu sintaksu za rešavanje navedenog problema iskazivanja veličine fonta u nekoj od podržanih jedinica:

- primenom notacije `[style.font-size.px]` veličina fonta se iskazuje u pikselima;
- primenom notacije `[style.font-size.em]` veličina fonta se iskazuje u `em` vrednostima (1em = 12pt, 2 em = 24 pt...);
- primenom notacije `[style.font-size.%]` veličina fonta se iskazuje u procentima.

DINAMIČKO PODEŠAVANJE BOJE TEKSTA I POZADINE

Sledeća dva elementa koriste color osobinu kao ulaz za podešavanje boje teksta i pozadine.

Primer može biti dodatno proširen, sledeća dva elementa koriste `color` osobinu kao ulaz za podešavanje boje teksta i pozadine:

```
<h4 class="ui horizontal divider header">
  ngStyle sa objektnom osobinom iz promenljive
```



```
</h4>

<div>
  <span [ngStyle]="{color: color}">
    {{ color }} tekst
  </span>
</div>

<h4 class="ui horizontal divider header">
  Stil iz promenljive
</h4>

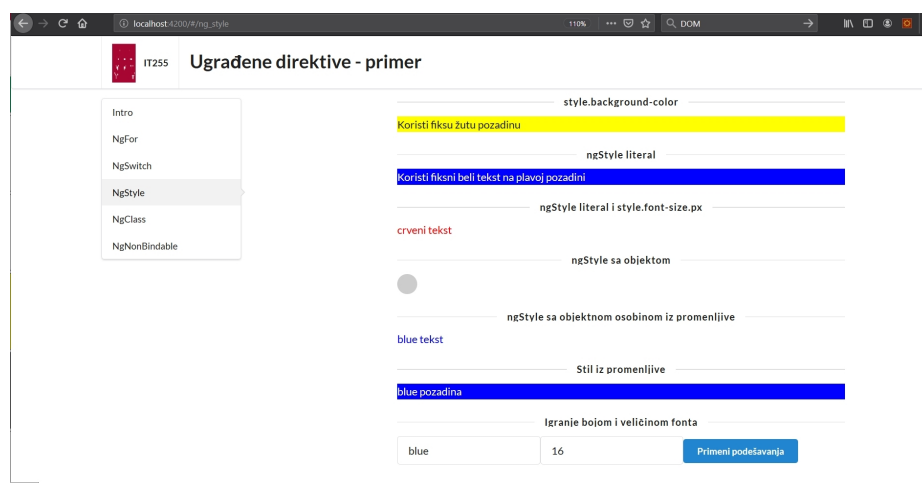
<div [style.background-color]="color"
  style="color: white;">
  {{ color }} pozadina
</div>
```

Na ovaj način, klikom na već kreirano dugme "Primeni podešavanja", poziva se sledeća metoda ([src/app/ng-style-example/ng-style-example.component.ts](#)) koja podešava nove vrednosti:

```
apply(color: string, fontSize: number): void {
  this.color = color;
  this.fontSize = fontSize;
}
```

Na ovaj način, boja i veličina fonta biće primenjeni na odgovarajuće elemente primenom direktive [ngStyle](#).

Sledećom slikom je prikazan tekući primer koji angažuje komponentu zaduženu za demonstraciju primene direktive [ngStyle](#).



Slika 2.1 Dinamičko podešavanje boje teksta i pozadine

▼ Poglavlje 3

Direktiva klasa u Angularu

DIREKTIVA NGCLASS

Angular direktiva `ngClass` omogućava dinamičko podešavanje i menjanje CSS klasa za neki element

Angular direktiva `ngClass` je predstavljena istoimenim atributom u HTML šablonu i omogućava dinamičko podešavanje i menjanje CSS klasa za dati DOM element.

Postoji više načina implementacije navedene direktive, prvi od njih može biti **prosleđivanje objektnog literala**. Objekat očekuje da poseduje ključeve koji odgovaraju nazivima klasa i vrednostima koje mogu biti *true* ili *false* i koje ukazuju da li će navedena CSS klasa biti primenjena ili ne.

Neka važi pretpostavka da postoji *CSS* klasa pod nazivom *bordered* čiji je zadatak da doda isprekidanu crnu liniju kao ivicu oko elementa. Sledeći CSS listing, uzet iz datoteke *styles.css*, detaljnije opisuje navedenu klasu.

```
.bordered {  
  border: 1px dashed black;  
  background-color: #eee; }
```

Sada je moguće dodati dva `<div>` elementa:

- prvi primenjuje navedenu *bordered* klasu i, otuda, je uvek oivičen;
- drugu ne primenjuje ovu klasu pa nije ni istaknut na navedeni način.

Navedeni kod je sastavni deo datoteke tekućeg primera: *src/app/ng-class-example/ng-class-example.component.html*.

```
<div [ngClass]="{bordered: false}">Nikada nema ivice</div>  
<div [ngClass]="{bordered: true}">Uvek ima ivice</div>
```

```
<div [ngClass]="{bordered: false}">Nikada nema ivice</div>  
<div [ngClass]="{bordered: true}">Uvek ima ivice</div>
```

Ovaj kod će kreirati sledeći HTML izlaz:

Nikada nema ivice

Uvek ima ivice

Slika 3.1 Jednostavna primena direktive ngClass

DINAMIČKA PRIMENA DIREKTIVE NGCLASS

U praksi je mnogo značajnije znati kako je moguće ovoj direktivi dodeliti dinamičke zadatke

Međutim, u praksi je mnogo značajnije znati kako je moguće ovoj direktivi dodeliti dinamičke zadatke. Da bi navedeno bilo moguće, neophodno je iskoristiti neku promenljivu kao vrednost konkretne objektno osobine. Na primer, dat je sledeći listing kao nastavak tekućeg primera (kod *opet pripada datoteci* `src/app/ng-class-example/ng-class-example.component.html`):

```
<div [ngClass]="{bordered: isBordered}">
  Primena objektnog literala. Border {{ isBordered ? "ON" : "OFF" }}
</div>
```

Alternativno, moguće je definisati *classesObj* unutar klase komponente (u konkretnom slučaju radi se o klasi *NgClassExampleComponent* čiji se sledeći kod čuva u datoteci `src/app/ng-class-example/ng-class-example.component.ts`).

```
@Component({
  selector: 'app-ng-class-example',
  templateUrl: './ng-class-example.component.html'
})
export class NgClassExampleComponent implements OnInit {
  isBordered: boolean;
  classesObj: Object;
  classList: string[];

  constructor() {
  }

  ngOnInit() {
    this.isBordered = true;
    this.classList = ['blue', 'round'];
    this.toggleBorder();
  }

  toggleBorder(): void {
    this.isBordered = !this.isBordered;
    this.classesObj = {
      bordered: this.isBordered
    };
  }
}
```

Sada je moguće direktno koristiti objekat na sledeći način:

```
<div [ngClass]="classesObj">
  Primena objektne varijable. Border {{ classesObj.bordered ? "ON" : "OFF" }}
</div>
```

Nakon dodatnog proširenja primera, za sada imamo HTML šablon koji daje sledeći izlaz:

Nikada nema ivice

Uvek ima ivice

Primena objektnog literala. Border OFF

Primena objektne varijable. Border OFF

Slika 3.2 Dinamička primena direktive ngClass

PRIMENA LISTE KLASA

Lista sa nazivima klasa za određivanje koja će SCSS klasa biti pridružena nekom elementu.

Posebna pogodnost, koju nudi okvir *Angular*, jeste mogućnost primene liste sa nazivima klasa za određivanje koja će, konkretno, CSS klasa biti pridružena nekom elementu.

Na primer, dat je sledeći niz koji ima dva konkretna člana (literal):

```
<div class="base" [ngClass]="['blue', 'round']">
  Ovde će uvek biti plava pozadina i
  zaobljene ivice.
</div>
```

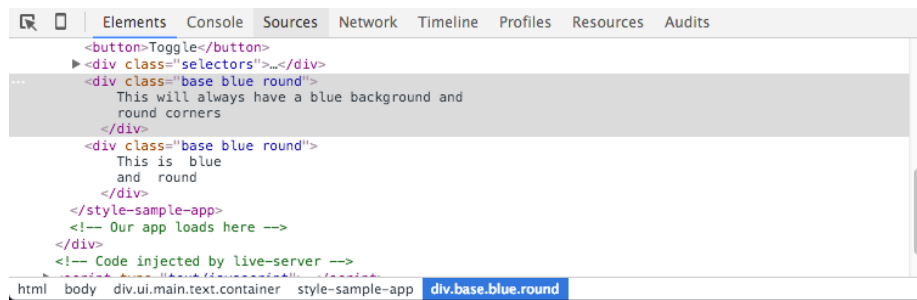
Takođe, moguće je dodeliti niz vrednosti nekoj osobini komponente, na primer na sledeći način:

```
this.classList = ['blue', 'round'];
```

Nakon ovoga je moguće proslediti ovu osobinu u šablon na prikazivanje:

```
<div class="base" [ngClass]="classList">
  Ovo je {{ classList.indexOf('blue') > -1 ? "" : "NOT" }} plavo
  i {{ classList.indexOf('round') > -1 ? "" : "NOT" }} zaobljeno
</div>
```

U poslednjem primeru, dodela vrednosti za *[ngClass]* funkcioniše uporedo sa dodelom postojećih vrednosti za HTML atribut *class*. CSS klase koje se vezuju za konkretan element, uvek će predstavljati skup klasa obezbeđenih upotrebom standardnog HTML atributa *class* i rezultatom provere direktive *[ngClass]*. U navedenom svetlu, neophodno je izvršiti fokusiranje na sledeću sliku i prvi listing na ovoj stranici.



Slika 3.3 Dobijene klase iz atributa i direktive

Kao što je moguće primetiti, element će posedovati tri klase:

- *base* - iz HTML *class* atributa;
- *blue* - iz direktive *[ngClass]*;
- *round* - iz direktive *[ngClass]*.

DODAVANJE INTERAKCIJE

Uvođenjem GUI kontrola postiže se interaktivnost tekuće komponente.

Aktuelni primer je moguće proširiti sledećom logikom:

- dodaje se novo dugme u šablon koje će omogućiti korisniku da, nakon klika, doda ivicu postojećim stringovima u šablonu;
- dodaju se dve *checkbox* kontrole u šablon za izbor da li će pozadina izabranih stringova biti obojena i predstavljena zaobljenim ivicama, respektivno.

Uvođenjem navedenih GUI kontrola postiže se interaktivnost tekuće komponente. Kontrole su realizovane sledećim listingom koji predstavlja proširenje koda šablona tekuće komponente:

```
<button (click)="toggleBorder()">Klikni</button>

<div class="selectors">
  <div>
    <input type="checkbox"
      [checked]="classList.indexOf('blue') > -1"
      (click)="toggleClass('blue')">
    <span>Plavo</span>
  </div>

  <div>
    <input type="checkbox"
      [checked]="classList.indexOf('round') > -1"
      (click)="toggleClass('round')">
    <span>Zaobljeno</span>
  </div>
</div>
```

Da bi interakcija korisnika sa kreiranim kontrolama zaživela, neophodno je, a i vidi se iz priloženog listinga, proširiti kod klase komponente metodama:

- `toggleBorder()` - dodaje stilizovanu ivicu oko elementa nakon klika na dugme;
- `toggleClass()` - pridružuje elementu CSS klasu nakon čekiranja određene checkbox kontrole.

```
ngOnInit() {
  this.isBordered = true;
  this.classList = ['blue', 'round'];
  this.toggleBorder();
}

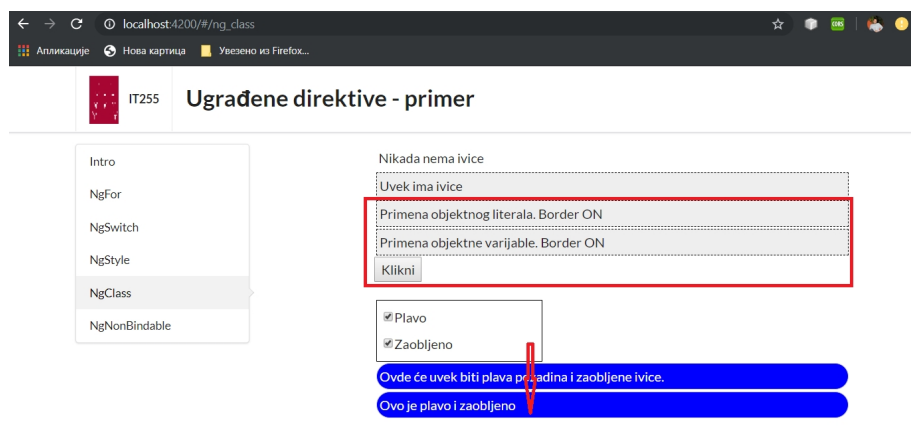
toggleBorder(): void {
  this.isBordered = !this.isBordered;
  this.classesObj = {
    bordered: this.isBordered
  };
};

toggleClass(cssClass: string): void {
  const pos: number = this.classList.indexOf(cssClass);
  if (pos > -1) {
    this.classList.splice(pos, 1);
  } else {
    this.classList.push(cssClass);
  }
}
```

DIREKTIVA NGCLASS I INTERAKCIJA - DEMO

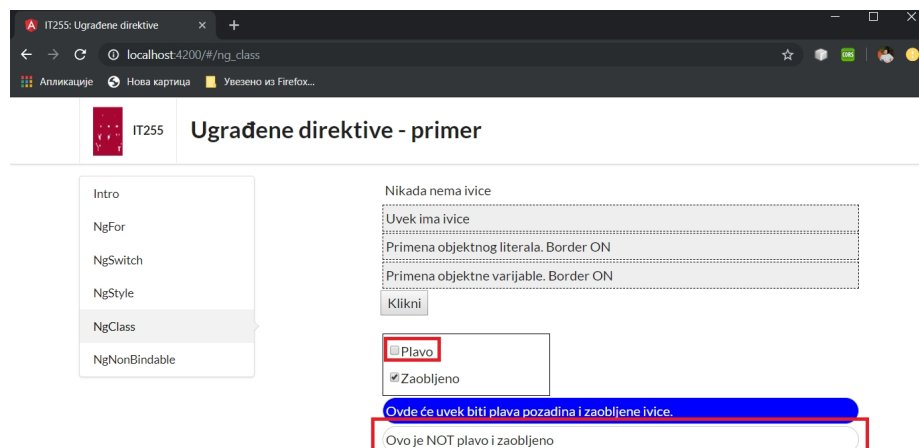
Testiranje izvršavanja kompletirane komponente.

Sledećom slikom je prikazan rezultat klika na kreirano dugme i selekcije oba kreirana `checkbox` - a.



Slika 3.4 Rezultat klika na kreirano dugme i selekcije oba kreirana checkbox - a

Za demonstraciju interaktivnosti aplikacije isključuje se jedna *checkbox* kontrola, na primer "*Plavo*". Rezultat izvršavanja aplikacije moguće je ilustrovati sledećom slikom. Promene su istaknute crvenom bojom.



Slika 3.5 Demonstracija interaktivnosti aplikacije isključivanjem jedne kontrole

▼ Poglavlje 4

Direktive petlji u Angularu

DIREKTIVA NGFOR

Uloga Angular direktive `ngFor` je obezbeđivanje ponavljanja izvesnog DOM elementa.

Uloga [Angular](#) direktive `ngFor` je obezbeđivanje ponavljanja izvesnog [DOM](#) elementa (ili kolekcije DOM elemenata) i prosleđivanje elementa niza za svaku od [iteracija](#). Osnovna sintaksa direktive `ngFor` je data sledećim iskazom:

```
*ngFor="let item of items".
```

Iz priloženog formata direktive `ngFor` moguće je primetiti sledeće:

- sintaksa `let item` specificira varijablu HTML šablona koja prihvata svaki od elemenata niza;
- sa `items` je označen niz elemenata dobijen od kontrolera.

Da bi primena navedene direktive bila lakše usvojena, neophodno je obaviti dodatna proširenja aktuelnog primera. Kreira se nova komponenta `ng-for-example` i u njenoj kontrolerskoj klasi postoji sledeći kod kojim se deklariše niz gradova (stringova):

```
this.cities = ['Kragujevac', 'Beograd', 'Niš'];
```

Da bi korisnik mogao da vidi sadržaj kreiranog niza, neophodno je dodati malo HTML koda u šablon nove komponente:

```
<h4 class="ui horizontal divider header">
  Jednostavna lista stringova
</h4>

<div class="ui list" *ngFor="let c of cities">
  <div class="item">{{ c }}</div>
</div>
```

Rezultat izvršavanja kreiranog koda bi mogao da bude ilustrovan sledećom slikom.

Jednostavna lista stringova
Kragujevac
Beograd
Niš

Slika 4.1 Rezultat primene direktive ngFor

ITERACIJA KROZ NIZ OBJEKATA

Cilj je demonstracija mehanizma iteracije kroz niz objekata u Angularu.

Primer i izlaganje može biti prošireno u drugom smeru. Cilj je demonstracija mehanizma iteracije kroz niz objekata u [Angularu](#). U kontrolerskoj klasi [src/app/ng-for-example/ng-for-example.component.ts](#) javlja se nov kod priložen sledećim listingom:

```
this.people = [
  { name: 'Vladimir', age: 45, city: 'Kragujevac' },
  { name: 'Marko', age: 24, city: 'Beograd' },
  { name: 'Nikola', age: 35, city: 'Niš' }
];
```

Da bi prikazani niz objekata mogao da bude prikazan na ekranu, neophodno je proširiti i HTML kod šablona komponente, dodavanjem sledećih linija koda:

```
<h4 class="ui horizontal divider header">
  Lista objekata
</h4>

<table class="ui celled table">
  <thead>
    <tr>
      <th>Ime</th>
      <th>Starost</th>
      <th>Grad</th>
    </tr>
  </thead>
  <tr *ngFor="let p of people">
    <td>{{ p.name }}</td>
    <td>{{ p.age }}</td>
    <td>{{ p.city }}</td>
  </tr>
</table>
```

Ako se pogleda linija koda 13, poslednjeg listinga, primećuje se da petlja veoma jednostavno prolazi kroz niz [people](#) objekata i na omogućava njihovo prikazivanje na ekranu u formi tabele. Navedeno može biti ilustrovano sledećom slikom.

Lista objekata		
Ime	Starost	Grad
Vladimir	45	Kragujevac
Marko	24	Beograd
Nikola	35	Niš

Slika 4.2 Iteracija kroz niz objekata

OBRADA UGNJEŽDENIH NIZOVA

Na jednostavan način je moguće vršiti obradu ugnježdenih (nested) nizova.

Na jednostavan način je moguće vršiti obradu ugnježdenih (*nested*) nizova. Ukoliko želimo da koristimo i prikazujemo slične podatke kao u prethodnom slučaju, ali razvrstane prema gradovima, moguće je deklarirati nov niz objekata, na sledeći način:

```
this.peopleByCity = [
  {
    city: 'Kragujevac',
    people: [
      { name: 'Lazar', age: 11 },
      { name: 'Isidora', age: 9 }
    ]
  },
  {
    city: 'Niš',
    people: [
      { name: 'Jovana', age: 25 },
      { name: 'Zoran', age: 36 }
    ]
  }
];
```

Listing pokazuje ugnježdene nizove *people* objekata unutar niza *city* objekata.

Sada, prema usvojenoj dobroj praksi, vrši se dogradnja HTML koda šablona tekuće komponente sa ciljem prikazivanja podataka iz prethodnog listinga.

```
<h4 class="ui horizontal divider header">
  Ugnježdjeni podaci
</h4>

<div *ngFor="let item of peopleByCity">
  <h2 class="ui header">{{ item.city }}</h2>
```

```
<table class="ui celled table">
  <thead>
    <tr>
      <th>Ime</th>
      <th>Starost</th>
    </tr>
  </thead>
  <tr *ngFor="let p of item.people">
    <td>{{ p.name }}</td>
    <td>{{ p.age }}</td>
  </tr>
</table>
</div>
```

Problem je rešen, kao što se vidi iz priloženog listinga, ugnježdavanjem *ngFor* petlji. Spoljašnja petlja (linija koda 5) prolazi kroz listu gradova, a unutrašnja petlja (linija koda 15) za svaki od gradova prolazi kroz listu osoba. Sledećom slikom je prikazan rezultat izvršavanja prikazanog koda.

Ugnježdjeni podaci

Kragujevac	
Ime	Starost
Lazar	11
Isidora	9

Niš	
Ime	Starost
Jovana	25
Zoran	36

Slika 4.3 Obrada ugnježđenih nizova

DOBIJANJE INDEKSA

U brojnim situacijama je veoma značajno dobijanje vrednosti indeksa za tekući element u iteraciji.

Do sada je posmatrana *ngFor* petlja koja se ponašala po poznatom šablonu for - each. Drugim rečima, prolazila je kroz niz i omogućavala vraćanje članova niza ali nije davala podršku za rad sa indeksima niza. U brojnim situacijama je veoma značajno dobijanje vrednosti indeksa za tekući element u iteraciji.

Dobijanje indeksa moguće je obaviti na veoma jednostavan način primenom proširene sintakse direktive *ngFor*. Proširenje se ogleda na dodavanje sintakse:

```
let idx = index
```

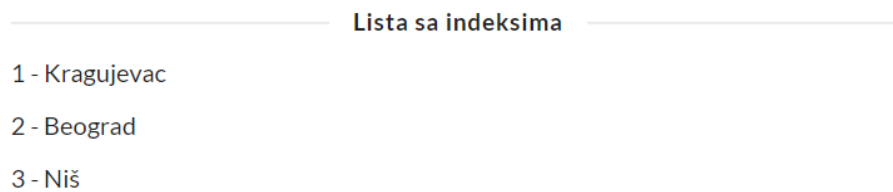
na vrednost *ngFor* direktive od koje je odvojena znakom "tačka - zarez" (;). Na ovaj način, *Angular* vrši pridruživanje vrednosti indeksa, za tekuću iteraciju petlje, promenljivoj *idx*.

Budući da je obavljena sva potrebna analiza, moguće je obaviti dodatno proširenje aktuelnog primera tako da korisnici mogu da dobiju informaciju o rednom broju grada, izvedenu na osnovu indeksa odgovarajućeg člana niza gradova. Sledećim listingom je proširen HTML kod šablona [src/app/ng-for-example/ng-for-example.component.html](#) tekuće komponente.

```
<h4 class="ui horizontal divider header">
  Lista sa indeksima
</h4>

<div class="ui list" *ngFor="let c of cities; let num = index">
  <div class="item">{{ num+1 }} - {{ c }}</div>
</div>
```

Petlja (linija koda 5) prolazi kroz niz gradova, uzima indeks i uvećava ga za 1 (linija koda 6), jer svaka petlja broji od 0 i, konačno, omogućava prikazivanje grada sa rednim brojem pojavljivanja na ekranu. Navedeno je prikazano sledećom slikom.



Slika 4.4 Dobijanje indeksa iz ngFor direktive

▼ Poglavlje 5

Direktiva ngNonBindable

PRIMENA DIREKTIVE NGNONBINDABLE

Direktiva se koristi kada je neophodno ukazati Angularu okviru da ne prevodi ili povezuje izvesnu sekciju na stranici.

Poslednja u nizu direktiva, koje obrađuje ova lekcija, jeste [Angular direktiva ngNonBindable](#). Ova direktiva se koristi kada je neophodno ukazati Angular okviru da ne prevodi ili povezuje izvesnu sekciju na stranici.

Na primer, cilj je prikazivanje teksta putem izraza `{{ content }}` ugrađenog u posmatrani HTML šablon. U normalnim okolnostima tekst bi bio povezan sa vrednošću varijable `content` i prikazan na ekranu.

Šta ako je potrebno prikazati tekst koji je identičan navedenom izrazu: `{{ content }}`? U tom slučaju je neophodno sprečiti obradu izraza primenom direktive [ngNonBindable](#).

Neka je ideja primena taga `<div>` za prikazivanje sadržaja kojeg čine:

- vrednost neke promenljive;
- leva strelica;
- izraz koji omogućava prikazivanje vrednosti navedene promenljive.

Navedeno kada se prevede u HTML kod izgleda kao u sledećem slučaju:

```
<div class='ngNonBindableDemo'>
  <span class="bordered">{{ content }}</span>
  <span class="pre" ngNonBindable>
    &larr; Ovo je {{ content }} kreirao
  </span>
</div>
```

Izraz `{{ content }}`, iz linije koda 2, biće rešen i njegova vrednost (varijabla `content`) će biti prikazana na ekranu. U slučaju linije koda 4, identičan izraz je obuhvaćen direktivom [ngNonBindable](#), neće biti obrađen i kao takav će biti prikazan kao običan tekst.

Još je neophodno specificirati vrednost promenljive `content` u kontroleru komponente:

```
constructor() {
  this.content = 'Neki tekst';
}
```

Rezultat primene direktive [ngNonBindable](#) dat je sledećom slikom.



Slika 5.1 Primena direktive ngNonBindable

ANGULAR DIREKTIVE - VIDEO MATERIJAL

Directives in Angular Applications - trajanje 25:12

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Kompletno urađen i proveren pokazni primer možete preuzeti iz aktivnosti Shared Resources odmah iza ovog objekta učenja.

▼ Poglavlje 6

Forme u Angular okviru

FORMA U VEB APLIKACIJI

Forma je verovatno najznačajniji aspekt svake veb aplikacije.

Forma je verovatno najznačajniji aspekt svake veb aplikacije. U velikom broju slučajeva, primenom formi je obezbeđeno prikupljanje širokog spektra podataka koje korisnik unese u aplikaciji.

Primena formi u veb aplikacijama, pa tako i Angular aplikacijama, često deluje dosta jednostavna i pravolinijska:

- kreiraju se ulazni (input) tagovi za podatke;
- korisnik popuni polja podacima;
- korisnik klikne na dugme za prosleđivanje unetih podataka.

Međutim, nije sve tako jednostavno i rad sa formama može biti veoma komplikovan:

- ulazi formi su predviđeni da menjaju podatke, kako na stranici, tako i na serveru;
- promene na formama se često reflektuju i na ostatak stranice;
- često je potrebno implementirati validaciju (proveru ispravnosti) za pojedina polja forme;
- korisnički interfejs bi trebalo jasno da prikaže uputstva i greške rada sa formama, ukoliko ih ima;
- zavisna polja forme mogu da poseduju kompleksnu logiku;
- često je neophodno obezbediti testiranje forme, bez oslanjanja na DOM selektore.

Zbog svega navedenog, veb programeri često koriste Angular okvir za prevazilaženje uočenih problema kompleksnosti rada sa formama. Angular kao rešenje nudi sledeće alate:

- FormControl - enkapsulira sve ulaze sa forme i grupiše ih objekat koji se dalje obrađuje u aplikaciji;
- Validator - proverava da li obavezno polje forme popunjeno, da li ima odgovarajući format, da li je unet odgovarajući tip podataka u polje forme i slično;
- Observer - posmatra (osluškuje) sve izmene na formi i reaguje na odgovarajući način.

U nastavku lekcije, cilj je učenje kreiranja formi po principu "korak - po - korak".

▼ 6.1 FormControl i FormGroup

FORMCONTROL

FormControl je jedan od dva osnovna objekta u Angular formama.

Dva osnovna objekta u Angular formama su:

- FormControl i
- FormGroup.

FormControl predstavlja jedno polje za unos podataka i predstavlja najmanju jedinicu Angular forme. FormControl enkapsulira vrednost koja je uneta u polje i daje neko od sledećih stanja:

- ispravno (*valid*);
- izmenjeno (*dirty* ili *changed*);
- sadrži greške (*errors*).

Na primer, sledećim listingom je prikazano kako se koristi FormControl u TypeScript - u:

Like many things in Angular, we have a class (FormControl, in this case) that we attach to the DOM with an attribute (formControl, in this case). For instance, we might have the following in our form:

```
// kreira nov FormControl objekat sa vrednošću "Vlada"
let nameControl = new FormControl("Vlada");

let name = nameControl.value; // -> Nate

// sada je moguće izvršiti upit nad kontrolom za doijanje vrednosti:
nameControl.errors // -> StringMap<string, any> za greške
nameControl.dirty // -> false
nameControl.valid // -> true
// i tako dalje
```

Za građenje forme kreiran je FormControl objekat kojem su, nakon toga, dodeljeni meta podaci i logika. Kao i za većinu ostalih stvari, a o nekima je već bilo diskutovano, Angular obezbeđuje konkretnu klasu FormControl koja se povezuje za DOM sa atributom (formControl, u ovom slučaju). Na primer u formi može da postoji sledeći segment:

```
<!-- deo neke veće forme -->
<input type="text" [formControl]="name" />
```

Navedeni kod će u kontekstu forme kreirati novi FormControl objekat. Naknadno će biti više diskusije u vezi sa funkcionisanjem navedenog.

FORMGROUP

U praksi, forma najčešće sadrži višegrupisanih polja za unos podataka.

U praksi, **forma** najčešće sadrži više od jednog polja za unos podataka. To znači da je neophodno obezbediti upravljanje većim brojem **FormControl** objekata. Ukoliko je cilj provera ispravnosti kreirane forme bilo bi preteško vršiti iteracije preko niza **FormControl** objekata i proveravati validnost svakog od njih, pojedinačno. Angular klasa **FormGroup** rešava ovaj problem na veoma jednostavan način. Klasa obezbeđuje omotački interfejs oko kolekcije **FormControl** objekata.

Sada je moguće izvršiti kreiranje jednog **FormGroup** objekta. Neka je to urađeno na sledeći način:

```
let personInfo = new FormGroup({
  firstName: new FormControl("Vladimir"),
  lastName: new FormControl("Milićević"),
  zip: new FormControl("34000")
})
```

Klase **FormGroup** i **FormControl** poseduju zajedničku roditeljsku klasu **AbstractControl** (<https://angular.io/api/forms/AbstractControl>). To praktično znači da status i vrednost **FormGroup** može biti proverena na jednostavan način, kao što je bio slučaj sa **FormControl** objektom.

```
personInfo.value; // -> {
  // firstName: "Vladimir",
  // lastName: "Milićević",
  // zip: "34000"
  // }

// sada je moguće izvršiti upit nad grupom kontrola, koje imaju osetljive
// vrednosti u zavisnosti od vrednosti potomaka FormControl:
personInfo.errors // -> StringMap<string, any> za greške
personInfo.dirty // -> false
personInfo.valid // -> true
// i tako dalje
```

Iz priloženog listinga je moguće primetiti da je pokušano dobijanje vrednosti iz **FormGroup** objekta prijemom objekata u formi (*ključ, vrednost*). Ovo je veoma pogodno za pribavljanje kompletnog skupa vrednosti iz forme bez potrebe za primenom iteracija preko pojedinačnih **FormControl** objekata.

▼ 6.2 Kreiranje prve forme

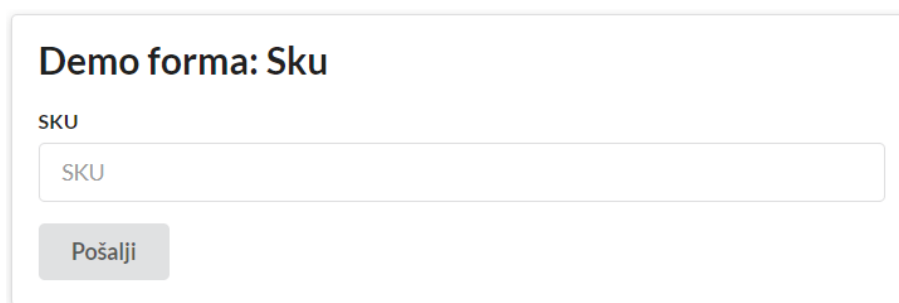
IZGLED PRVE FORME

Postoje brojni koraci koje veb programeri izvode kada kreiraju forme.

Postoje brojni koraci koje veb programeri izvode kada kreiraju forme, a dobar broj njih još uvek nije razmatran. U tu svrhu biće razvijan potpuno nov primer za lakše razumevanje problematike rada sa formama u Angular - u.

Potpuno urađeni primer možete preuzeti na kraju ovog dela lekcije, iz kategorije, Shared resources ali je preporuka da pokušate samostalno, prateći izlaganje u ovom delu lekcije, kreirate vašu aplikaciju.

Sledećom slikom je dato idejno rešenje izgleda forme na kojem će biti rađeno tokom kreiranja pratećeg primera.



Slika 6.1.1 Idejno rešenje izgleda forme

SKU je skraćenica za „stockkeeping unit - jedinica zaliha“. To je izraz za jedinstveni id proizvoda koji će se pratiti u zalihama. Kadase govori oSKU-u, govori seo ljudski čitljivom ID-u predmeta.

Predložena forma je veoma jednostavna: postoji polje za unos (zajedno sa odgovarajućom labelom) i dugme za slanje podataka nakon što forma bude popunjena. Da bi navedeno bilo realizovano, neophodno je kreiranje komponente koja implementira ovu formu.

Kao i u prethodnim slučajevima, tri dela čine komponentu:

- podešavanje `@Component()` dekoratora;
- kreiranje šablona komponente;
- implementacija funkcionalnosti putem klase komponente.

UČITAVANJE FORMSMODULE PODRŠKE

Neophodno je učitavanje odgovarajućih biblioteka u klasu obeleženu dekoratorom `NgModule`.

Sa ciljem korišćenja najnovijih biblioteka za rad sa formama neophodno je obaviti učitavanje odgovarajućih biblioteka u klasu obeleženu dekoratorom `@NgModule`.

Postoje dva načina korišćenja formi podržana Angular okvirom:

- primena `FormsModule` biblioteke;
- primena `ReactiveFormsModule` biblioteke.

Budući da će obe biblioteke biti korišćene u radu na konkretnom primeru, biće obe učitane (importovane) u navedeni modul. Da bi navedeno bilo realizovano neophodno je u datoteku centralne komponente aplikacije `app.module.ts` dodati sledeći kod:

```
import {  
  FormsModule,  
  ReactiveFormsModule  
} from '@angular/forms';  
  
// dalje idu dodatni importi  
  
@NgModule({  
  declarations: [  
    FormsDemoApp,  
    DemoFormSkuComponent,  
    // ... ovde idu deklaracije  
  ],  
  imports: [  
    BrowserModule,  
    FormsModule, // dodati ovo  
    ReactiveFormsModule // dodati ovo  
  ],  
  bootstrap: [ FormsDemoApp ]  
})  
class AppModule {}
```

Slika 6.1.2 Učitavanje FormsModule podrške

Listingom prikazanim prethodnom slikom postignuta je mogućnost korišćenja direktiva formi u HTML pogledima aplikacije, koji će tek biti kreirani kako budu dodatne komponente uvedene u aplikaciju. `FormsModule` obezbeđuje šablone bazirane na direktivama na poput:

- `ngModel` i
- `NgForm`.

`ReactiveFormsModule`, sa druge strane, obezbeđuje šablone bazirane na direktivama na poput:

- `formControl` i
- `ngFormGroup`.

Do sada nije bilo govora o tome šta koja direktiva radi ili kako se koristi ali će biti uskoro. Za sada, dovoljno je da se zna da dodavanjem `FormsModule` i `ReactiveFormsModule` u `NgModule` obezbeđeno je korišćenje direktiva formi u šablonima ili umetanje odgovarajućih provajdera direktiva u komponente.

JEDNOSTAVNA FORMA: @COMPONENT DECORATOR

Kreiran je nov tag `app-demo-form-sku` koji će omogućiti prikazivanje HTML sadržaja šablona.

Sada kada je napravljen detalja uvod, moguće je posvetiti posebnu pažnju kreiranju forme koja je ilustrovana slikom 1. Za početak, neophodno je obaviti kreiranje odgovarajuće komponente: `src/app/demo-form-sku/demo-form-sku.component.ts`. Sledećim listingom je dat njen dekorator `@Component` sa definisanim selektorom i šablonom.

```
@Component({  
  selector: 'app-demo-form-sku',  
  templateUrl: './demo-form-sku.component.html',  
  // ... ostatak će kasnije biti definisan
```

Ako se pogleda kod, bitno je još jednom napomenuti da je kreiran nov HTML tag `app-demo-form-sku` koji će omogućiti prikazivanje HTML sadržaja šablona `demo-form-sku.component.html` navođenjem:

```
<app-demo-form-sku></app-demo-form-sku>
```

Sledeći korak predstavlja upravo definisanje šablona, odnosno dela aplikacije koji će biti vidljiv krajnjem korisniku. Kod šablona se nalazi u datoteci `src/app/demo-form-sku/demo-form-sku.component.html` i priložen je sledećim listingom.

```
<div class="ui raised segment">  
  <h2 class="ui header">Demo forma: Sku</h2>  
  <form #f="ngForm"  
    (ngSubmit)="onSubmit(f.value)"  
    class="ui form">  
  
    <div class="field">  
      <label for="skuInput">SKU</label>  
      <input type="text"  
        id="skuInput"  
        placeholder="SKU"  
        name="sku" ngModel>  
    </div>  
  
    <button type="submit" class="ui button">Pošalji</button>  
  </form>  
</div>
```

U narednom izlaganju sledi detaljna analiza priloženog koda šablona.

PRIMENA FORM I NGFORM

Uključena biblioteka `FormsModule` omogućava primenu direktive `ngForm`

Kako izučavanje problematike odmiče, tako ona postaje sve zanimljivija. Iz razloga što je uključena biblioteka *FormsModule*, omogućena je primena direktive *NgForm*. Takođe, trebalo bi imati na umu, da kada se direktiva učini dostupnom u pogledu, ona će se vezati za bilo koji element koji odgovara selektoru.

Primenom direktive *NgForm* omogućene su neke veoma korisne stvari koje nisu očigledne: uključuje tag forme u vlastiti selektor (umesto zahteva za eksplicitnim dodavanjem *NgForm* kao atributa). Ovo praktično znači da ukoliko se obavi dodavanje biblioteke *FormsModule*, direktiva *NgForm* će automatski biti povezana sa svakim `<form>` tagom koji se nalazi u pogledu. Ovo je veoma korisno, ali može biti i zbunjujuće jer se dešava u pozadini.

Postoje dva ključna detalja funkcionalnosti direktive *NgForm*:

- kreiranje *FormGroup* objekta pod nazivom *ngForm*;
- kreiranje *(ngSubmit)* izlaza.

Primena navedenog u okviru `<form>` taga data je sledećim listingom:

```
<form #f="ngForm"
      (ngSubmit)="onSubmit(f.value)"
```

Posebno je neophodno, u ovom trenutku, dati značenje sintakse: `#f="ngForm"`. Na ovaj način kreirana je lokalna promenljiva *f* koja će biti korišćena u pogledu i predstavlja pseudonim za *ngForm*. Dalje, *ngForm* objekat je tipa *FormGroup* i dolazi iz direktive *NgForm*. konačno, promenljiva *f* se koristi u pogledu kao *FormGroup* (videti liniju koda 2 (*ngSubmit*) izlaz).

Povezivanje akcije *ngSubmit*, forme koja se kreira, obavlja se primenom sintakse:

```
(ngSubmit)="onSubmit(f.value)"
```

Ovde je značajno par stvari:

- *(ngSubmit)* - dobija se iz *NgForm*;
- *onSubmit()* - funkcija koja će biti implementirana u klasi komponente;
- *f.value* - *f* je prethodno definisan objekat tipa *FormGroup* pri čemu atribut *value* vraća vrednost u formi (ključ / vrednost) za ovaj *FormGroup*.

Kada se sve ovo poveže u celinu može biti interpretirano na sledeći način: **kada se popuni i potvrdi forma (klikom na dugme), poziva se funkcija *onSubmit()* klase komponente kojoj se, kao argument, prosleđuje vrednost forme.**

PRIEMENA INPUT & NGMODEL

input tag poseduje detalje koje je neophodno razumeti pre doticanja NgModel direktive.

Korišćeni input tag poseduje par zanačajnih stvari koje su vredne diskusije pre doticanja *NgModel* direktive.

```
<form #f="ngForm"
  (ngSubmit)="onSubmit(f.value)"
  class="ui form">

  <div class="field">
    <label for="skuInput">SKU</label>
    <input type="text"
      id="skuInput"
      placeholder="SKU"
      name="sku" ngModel>
  </div>
  <button type="submit" class="ui button">Pošalji</button>
</form>
```

- iskazi `class="ui form"` i `class="field"` su opcionalni i definišu CSS klase koje se koriste za stilizovanje forme i kontrola koje se prikazuju u pogledu;
- atribut labele "for" mora da se podudara sa atributom "id" polja za unos podataka po W3C standardu. Jednostavno, labela se odnosi na input polje;
- iskaz `placeholder="SKU"` predstavlja napomenu za korisnika koja postoji kada je input polje ne popunjeno.

Direktiva NgModel specificira selektor za ngModel. To znači da je moguće primeniti je u input tagu dodavanjem sledeće vrste atributa:

`ngModel="bilo šta"`.

U ovom slučaju (pogledati priloženi kod), specificiran je `ngModel` bez vrednosti atributa.

Postoji nekoliko različitih načina da se specificira `ngModel` u šablonima i ovo je prvi od njih. Kada se koristi `ngModel` bez vrednosti atributa, specificira se sledeće:

- jednosmerno povezivanje podataka (one-way data binding);
- kreiranje `FormControl` objekta sa nazivom `sku` (zbog atributa `name` input taga).

`NgModel` kreira nov `FormControl` objekat koji je automatski dodat roditelju `FormGroup` (u ovom slučaju, na formu) i potom povezuje `DOM` element sa kreiranim `FormControl` objektom. To znači, podešena je asocijacija između `input` taga iz pogleda sa objektom tipa `FormControl`. a podudaranje u asocijaciji je obezbeđeno preko naziva (atribut `name`), a to je u ovom slučaju `"sku"`.

JEDNOSTAVNA FORMA: DEFINICIJA KLASJE KOMPONENTE

Ovaj deo primera je zaokružen definicijom klase odgovarajuće komponente.

Ovaj deo primera mora da bude zaokružen definicijom klase odgovarajuće komponente. U datoteku `src/app/demo-form-sku/demo-form-sku.component.ts` neophodno je dodati kod koji odgovara sledećem listingu:

```
export class DemoFormSkuComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

  onSubmit(form: any): void {
    console.log('Poslali ste vrednost:', form);
  }

}
```

Klasa definiše jednu metodu `onSubmit()` koja će biti izvršena kada korisnik klikne na dugme sa forme. Metoda će u konzoli veb pregledača prikazati podatke unete putem forme.

JEDNOSTAVNA FORMA: DEMO PRIMERA

Kod je objedinjen i može da bude isproban.

Za lakše snalaženje studenata, prilikom izrade ovog primera, sledi objedinjavanje koda komponente na jednom mestu.

Listing klase komponente:

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-demo-form-sku',
  templateUrl: './demo-form-sku.component.html',
  styles: []
})
export class DemoFormSkuComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

  onSubmit(form: any): void {
    console.log('Poslali ste vrednost:', form);
  }

}
```

Listing šablona komponente:

```
<div class="ui raised segment">
  <h2 class="ui header">Demo forma: Sku</h2>
  <form #f="ngForm"
    (ngSubmit)="onSubmit(f.value)"
```

```

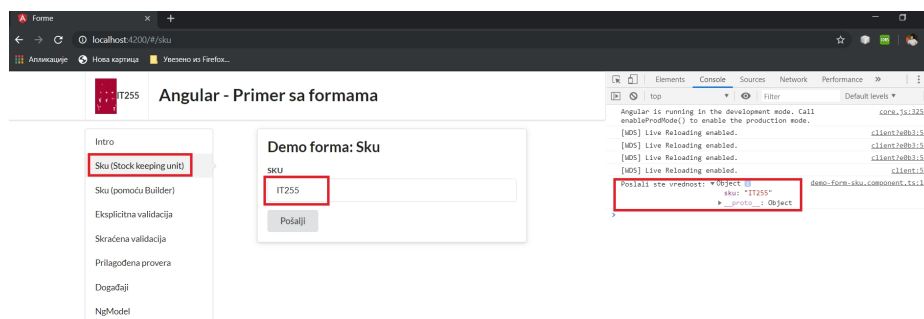
class="ui form">

<div class="field">
  <label for="skuInput">SKU</label>
  <input type="text"
    id="skuInput"
    placeholder="SKU"
    name="sku" ngModel>
</div>

  <button type="submit" class="ui button">Pošalji</button>
</form>
</div>

```

Izgled ekrana generisanog izvršavanjem koda komponente:



Slika 6.1.3 Jednostavna forma: demo primera

▼ 6.3 Korišćenje klase FormBuilder

POMOĆNA KLASA FORMBUILDER

*Građenje **FormControls** i **FormGroups** primenom **ngForm** je pogodno ali ne dozvoljava veća podešavanja.*

Implicitno građenje **FormControls** i **FormGroups** primenom **ngForm** je veoma pogodno ali ne dozvoljava veća podešavanja. Fleksibilniji i opšti način za kreiranje formi u **Angular** - u predstavlja primena **FormBuilder** - a.

FormBuilder je pomoćna klasa koja pomaže prilikom kreiranja formi. Kao što je do sad naučeno, forme su napravljene od **FormControls** i **FormGroups** objekata, a **FormBuilder** pomaže prilikom njihovog kreiranja.

Sada će biti diskusija prenet na aktuelni primer u kojem će biti kreirana **forma** primenom klase **FormBuilder**. Posebno je potrebno obratiti pažnju na sledeće stvari:

- kako se koristi **FormBuilder** u klasi komponente;
- kako se kreira prilagođen **FormGroup** na formi u pogledu.

REAKTIVNE FORME SA FORMBUILDER IMPLEMENTACIJOM

Tokom umetanja zavisnosti, objekat klase FormBuilder će biti kreiran i pridružen varijabli.

Primer se dalje proširuje kreiranjem potpuno nove komponente. U ovoj komponenti će biti korišćene direktive `formGroup` i `formControl`, a to znači da je neophodno uključiti odgovarajuće klase. To je prikazano sledećim listingom (sadržaj datoteke: `src/app/demo-form-sku-with-builder/demo-form-sku-with-builder.component.ts`):

```
import { Component, OnInit } from '@angular/core';
import {
  FormBuilder,
  FormGroup
} from '@angular/forms';
```

Umetanje (*injection*) objekta tipa `FormBuilder` obavlja se kreiranjem argumenta u konstruktoru klase nove komponente.

```
import { Component, OnInit } from '@angular/core';
import {
  FormBuilder,
  FormGroup
} from '@angular/forms';

@Component({
  selector: 'app-demo-form-sku-with-builder',
  templateUrl: './demo-form-sku-with-builder.component.html',
  styles: []
})
export class DemoFormSkuWithBuilderComponent implements OnInit {
  myForm: FormGroup;

  constructor(fb: FormBuilder) {
    this.myForm = fb.group({
      'sku': ['ABC123']
    });
  }

  ngOnInit() {
  }

  onSubmit(value: string): void {
    console.log('Poslali ste vrednost: ', value);
  }
}
```

Tokom *umetanja zavisnosti*, objekat klase `FormBuilder` će biti kreiran i pridružen kreiranoj promenljivoj `fb` (u konstruktoru).

Nad ovim objektom će biti korišćene dve glavne funkcije:

- `control` - kreira nov `FormControl`;
- `group` - kreira nov `FormGroup`.

Dalje, moguće je primetiti da je podešena nova objektna promenljiva `myForm` klase komponente kao tip `FormGroup`. Objekat, navedenog tipa podataka, kreira se pozivom `fb.group()` (linija koda 16). Funkcija `group()` uzima objekat parova (ključ, vrednost) koji specificiraju `FormControls` objekte u ovoj grupi.

U ovom slučaju, podešena je kontrola sa nazivom `"sku"` i njena vrednost je `["ABC123"]` - ovo ukazuje da je podrazumevana vrednost za kontrolu `"ABC123"`. Ovde je, na osnovu zagrada, moguće primetiti da se radi o nizu, to je zbog toga što će naknadno dodatne konfiguracione opcije biti dodate.

Sada, pošto je kreiran `myForm` objekat, on mora da bude iskorišćen u pogledu komponente.

UPOTREBA MYFORM OBJEKTA U ŠABLONU KOMPONENTE

Ideja je da se sadržaj taga `form` izmeni tako da koristi `myForm` objekat.

Ako se pogleda početni primer ovde je neophodno uvesti neka odstupanja. Ideja je da se sadržaj taga `<form>` izmeni tako da koristi `myForm` objekat kreiran u klasi nove komponente. U prošlom izlaganju je konstatovano da je `ngForm` direktiva automatski bila primenjena kada se koristi `FormsModule`. Takođe, `ngForm` kreira vlastitu `FormGroup`.

U ovom slučaju upotreba spoljašnje implementacije `FormGroup` klase biće zamenjena upotrebom objektna promenljive `myForm`, koja je kreirana pomoću `FormBuilder` objekta.

Angular nudi dodatnu direktivu koja se koristi kada je na raspolaganju postojeća `FormGroup` promenljiva. Direktiva poseduje naziv `formGroup` i koristi se na sledeći način:

```
<form [formGroup]="myForm"
      (ngSubmit)="onSubmit(myForm.value)"
```

Na ovaj način je ukazano Angular okviru da se promenljiva `myForm` koristi kao `FormGroup` na ovoj formi. Takođe, bilo je neophodno promeniti poziv `onSubmit()` tako da koristi promenljivu `myForm` umesto prethodne `f`. To je bitno iz razloga što promenljiva `myForm` sada poseduje urađena podešavanja i vrednosti.

Još jednu stvar je neophodno obaviti da bi nov primer postao funkcionalan. Potrebno je obaviti povezivanje `FormControl` objekta sa input tagom. Sada je moguće prisetiti se da direktiva `ngControl` kreira nov `FormControl` objekat i povezuje ga sa roditeljskom klasom `FormGroup`. U ovom slučaju je upotrebljen `FormBuilder` za kreiranje vlastitog `FormControl` objekta.

Dakle, kada je cilj povezivanje postojećeg `FormControl` objekta koristi se direktiva `formControl`:

```
<label for="skuInput">SKU</label>
<input type="text"
      id="skuInput"
```

```
placeholder="SKU"
[formControl]="myForm.controls['sku']">
```

KORIŠĆENJE KLASSE FORMBUILDER - CELOKUPAN LISTING

Kod je proširen, objedinjen i može da bude isproban.

Za lakše snalaženje studenata, prilikom izrade ovog primera, sledi objedinjavanje koda nove komponente na jednom mestu.

Listing klase komponente:

```
import { Component, OnInit } from '@angular/core';
import {
  FormBuilder,
  FormGroup
} from '@angular/forms';

@Component({
  selector: 'app-demo-form-sku-with-builder',
  templateUrl: './demo-form-sku-with-builder.component.html',
  styles: []
})
export class DemoFormSkuWithBuilderComponent implements OnInit {
  myForm: FormGroup;

  constructor(fb: FormBuilder) {
    this.myForm = fb.group({
      'sku': ['ABC123']
    });
  }

  ngOnInit() {
  }

  onSubmit(value: string): void {
    console.log('Poslali ste vrednost: ', value);
  }
}
```

Listing šablona komponente:

```
<div class="ui raised segment">
  <h2 class="ui header">Demo Form: Sku koristeći Builder</h2>
  <form [formGroup]="myForm"
    (ngSubmit)="onSubmit(myForm.value)"
    class="ui form">

    <div class="field">
      <label for="skuInput">SKU</label>
```

```
<input type="text"
      id="skuInput"
      placeholder="SKU"
      [formControl]="myForm.controls['sku']">

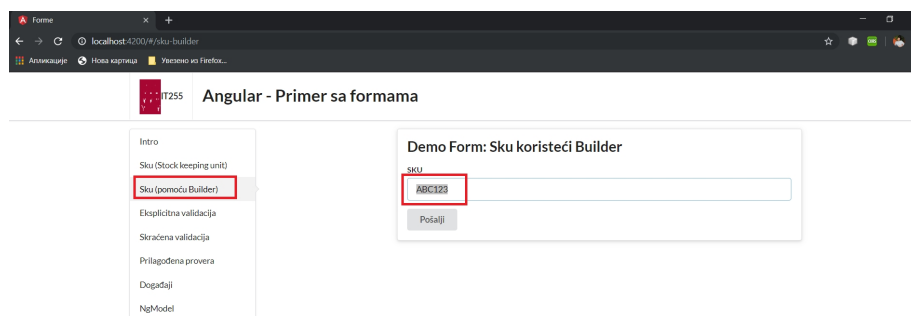
</div>

<button type="submit" class="ui button">Pošalji</button>
</form>
</div>
```

KORIŠĆENJE KLASSE FORMBUILDER - DEMO

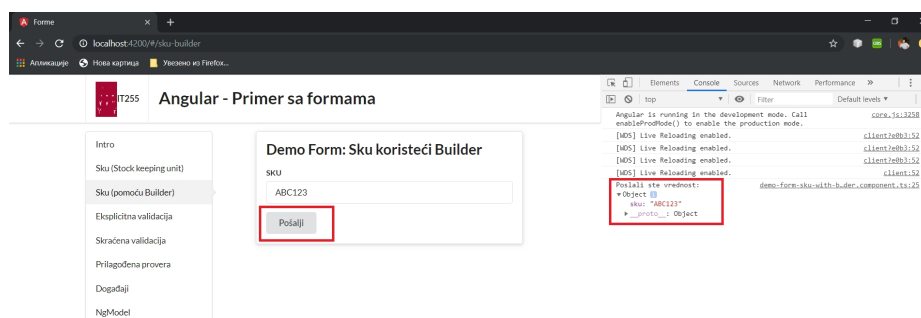
Izgled ekrana nastao kao rezultat izvršavanja koda nove komponente.

Pokazni primer se ponovo prevodi, iz menija se bira opcija koja odgovara novom primeru i na stanici se pojavljuje forma koja u svojoj kontroli ima podrazumevanu vrednost "ABC123" (slika 1)



Slika 6.2.1 Forma koja u svojoj kontroli ima podrazumevanu vrednost

Klik na dugme forme poziva metodu `onSubmit()` komponente, a rezultat se vidi na sledećoj slici.



Slika 6.2.2 Izvršavanje metode komponente nakon klika za potvrdu forme

▼ 6.4 Validacija

VALIDATOR

Često se dešava da korisnici unesu podatke u formu koji nisu u traženom formatu.

Iz prethodnog izlaganja studenti su stekli osnovnu predstavu u vezi sa kreiranjem [Angular](#) formi, prikupljanjem i slanjem podataka iz forme i, konačno njihovom jednostavnom obradom (prikazivanje u konzoli pregledača). Međutim, u velikom broju slučajeva nije sve tako jednostavno. Često se dešava da korisnici unesu podatke u formu koji nisu u traženom formatu. Da bi bilo sprečeno da ovakav podatak ode na obradu koriste se validatori. [Validator](#) je funkcija koja obrađuje [FormControl](#) ili kolekciju kontrola i obezbeđena je preko modula [Validators](#).

Najjednostavniji validator je [Validators.required](#) kojim se zahteva da konkretno polje za unos podataka mora da bude popunjeno. U suprotnom, [FormControl](#) objekat će se smatrati nevalidnim (*invalid*).

Da bi u Angular veb aplikaciji validator mogao da bude upotrebljen, neophodno je uraditi sledeće:

1. pridružiti validator [FormControl](#) objektu;
2. proveriti status validatora u pogledu i preuzeti određenu akciju.

Pridruživanje validatora konkretnoj kontroli ([FormControl](#) objektu) obavlja se jednostavno njegovim prosleđivanjem u obliku drugog argumenta konstruktora [FormControl](#).

```
let control = new FormControl('sku', Validators.required);
```

U slučaju aktuelnog primera, budući da se sada koristi [FormBuilder](#) biće upotrebljena sledeća sintaksa:

```
constructor(fb: FormBuilder) {  
  this.myForm = fb.group({  
    'sku': ['', Validators.required]  
  });  
  
  this.sku = this.myForm.controls['sku'];  
}
```

U nastavku, neophodno je primeniti validaciju u pogledu komponente. Postoje dva načina da se pristupi vrednosti validacije u HTML pogledima:

1. moguće je eksplicitno pridružiti kontrolu [sku](#) objektnoj promenljivoj klase komponente koju je moguće više puta upotrebiti i daje jednostavan pristup [FormControl](#) objektima u šablonima (pogledima);

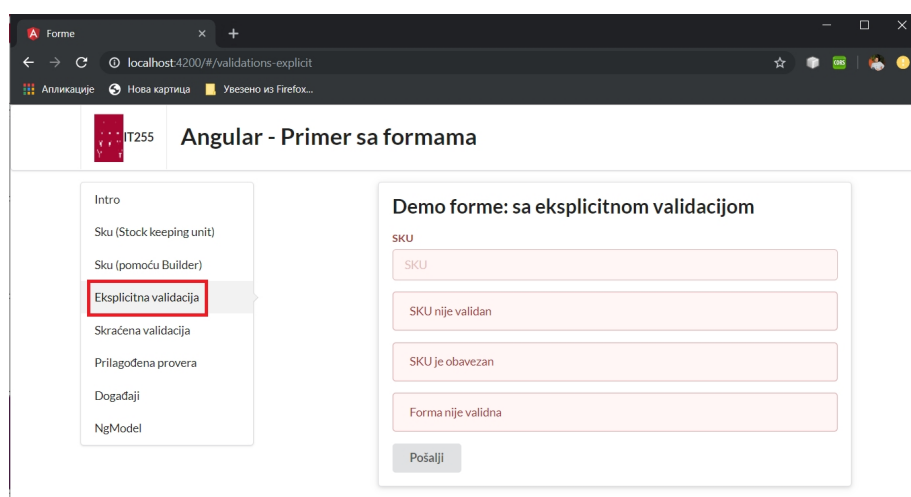
2. *FormControl* objekat *sku* moguće je potražiti iz varijable *myForm* unutar pogleda. To zahteva manje rada sa klasom komponente, ali je kompleksnije kada je druga strana komponente u pitanju - pogled.

Da bi navedeni scenariji bili jasniji, neophodno je obaviti njihovu pojedinačnu analizu u narednom izlaganju.

EKSPLICITNO PODEŠAVANJE KONTROLE KAO OBJEKTNE PROMENLJIVE

Najfleksibilniji način rada sa pojedinačnim kontrolama je njihovo podešavanje preko varijabli.

Sledećom slikom je prikazan izgled forme sa primenjenom validacijom.



Slika 6.3.1 Izgled forme sa primenjenom validacijom

Najfleksibilniji način rada sa pojedinačnim kontrolama forme u šablonima jeste podešavanje svakog *FormControl* objekta kao objektne promenljive klase odgovarajuće komponente. Poznato polje za unos podataka "*sku*", u klasi komponente, može biti podešeno na sledeći način:

```
export class DemoFormWithValidationsExplicitComponent {  
  myForm: FormGroup;  
  sku: AbstractControl;  
  
  constructor(fb: FormBuilder) {  
    this.myForm = fb.group({  
      'sku': ['', Validators.required]  
    });  
  
    this.sku = this.myForm.controls['sku'];  
  }  
  
  onSubmit(value: string): void {  
    console.log('you submitted value: ', value);  
  }  
}
```

Slika 6.3.2 Podešavanje FormControl objekta kao objektne promenljive

U nastavku sedi kraća analiza priloženog koda.

ANALIZA KODA KLASSE KOMPONENTE

Moguće referencirati promenljivu klase bilo gde u pogledu komponente.

Ukoliko se pogleda kod koji je priložen prethodnom slikom, moguće je primetiti sledeće:

1. specificirana je promenljiva `sku` tipa `AbstractControl`;
2. `this.sku` dobija vrednost nakon kreiranja `myForm` preko `FormBuilder-a`.

Ovo je veoma značajno iz razloga što je moguće referencirati `sku` bilo gde u pogledu komponente. Nedostatak ovog pristupa bi bila neophodnost podešavanja objektne promenljive za svako polje forme. Za složene velike forme ovo može biti veoma naporno i kompleksno.

DODATNI NAČINI VALIDACIJE

Proširenje problematike validacije dodatnim načinima primene validatora u pogledima.

Sada kada je postavljen validator na polje `sku`, moguće je obaviti proširenje ove problematike dodatnim načinima primene validatora u pogledima:

1. provera ispravnosti forme u celini i prikazivanje poruke;
2. provera ispravnosti pojedinačnih polja forme i prikazivanje poruke;

3. provera ispravnosti pojedinačnih polja forme i njihovo bojenje crvenom bojom u slučaju greške;
4. provera ispravnosti pojedinačnih polja forme za određeni zahtev i prikazivanje poruke.

PORUKE FORME I POLJA

Greške prilikom unosa podataka u formu ili pojedinačna polja prikazuju se porukama validacije.

Greške prilikom unosa podataka u formu ili pojedinačna polja prikazuju se porukama validacije.

Poruke forme

Dat je sledeći deo listinga šablona komponente: [src/app/demo-form-with-validations-explicit/demo-form-with-validations-explicit.component.html](#):

```
<div *ngIf="!myForm.valid"
      class="ui error message">Forma nije validna</div>
```

Ovde je važno zapamtiti, **myForm** je objekat tipa **FormGroup** koji je validan samo ako su i svi potomci - **FormControls** objekti, takođe, validni.

Poruke polja

Takođe, moguće je prikazati i poruku validacije sa pojedinačna polja u slučaju da njihovi **FormControl** objekti nisu validni. U istu datoteku, kao u prethodnom slučaju, moguće je dodati sledeći kod:

```
<div *ngIf="!sku.valid"
      class="ui error message">SKU nije validan</div>
```

BOJENJE POLJA

Za isticanje grešaka unosa podataka u polja forme moguće je koristiti i specijalne CSS klase.

Bojenje polja forme

Za isticanje grešaka unosa podataka u polja forme moguće je koristiti i specijalne CSS klase. U konkretnom primeru, dostupno je u urađenom primeru na kraju ovog objekta učenja, koristi se CSS klasa **error**, koja pripada CSS okviru **Semantic UI**. Ova klasa, kada je pridružena **input** tagu, u slučaju greške prikazaće ovaj tag obojen u crveno.

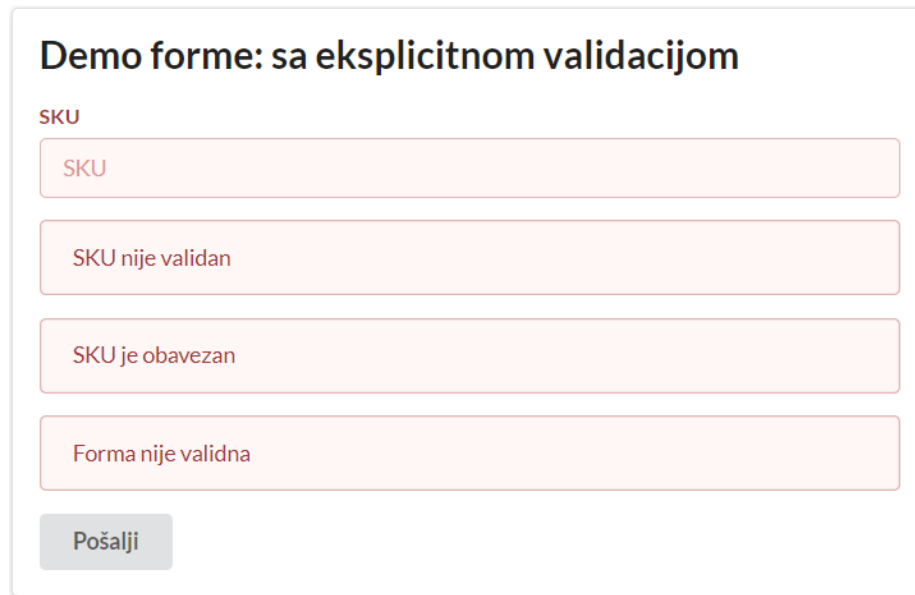
```
<div class="field"
      [class.error]="!sku.valid && sku.touched">
```

Ovde je moguće primetiti da postoje dva uslova za podešavanje **error** klase:

- proverava se tačnost izraza `!sku.valid`;
- proverava se tačnost izraza `sku.touched`.

Ovde je ideja da se greška ispoljava samo ako je korisnik pokušao unos u polje forme i ako to nije uradio na ispravan način.

Ovo je moguće isprobati tako se unosi neki podatak u polje za unos, a potom se odmah obriše. Rezultat će biti kao na sledećoj slici.



Slika 6.3.3 Obojena polja zbog grešaka unosa

SPECIFIČNA VALIDACIJA

Polje forme može da bude nevalidno iz različitih razloga.

Polje forme može da bude nevalidno iz brojnih razloga. Često je potrebno prikazati različite poruke u zavisnosti od razloga zbog kojeg je "pala" validacija.

Za demonstraciju specifičnog "pada" validacije, moguće je upotrebiti metodu `hasError`. Nastavlja se rad na istom primeru i u datoteku `src/app/demo-form-with-validations-explicit/demo-form-with-validations-explicit.component.html` se dodaje sledeći kod:

```
<div *ngIf="sku.hasError('required')"  
  class="ui error message">SKU je obavezan</div>
```

Metoda `hasError` je definisana istovremeno za `FormControl` i `FormGroup`, a to znači da je primenjiva na pojedinačne kontrole ali i na formu u celini. Imajući ovo u vidu, moguće je imati i sledeći kod:

```
<div *ngIf="myForm.hasError('required', 'sku')"  
  class="error">SKU je obavezan</div>
```

VALIDACIJA - CELOKUPAN LISTING

Kod je proširen validacijom, objedinjen i može da bude isproban.

Sledi celokupan listing klase komponente koji se nalazi u datoteci [src/app/demo-form-with-validations-explicit/demo-form-with-validations-explicit.component.ts](#).

```
import { Component } from '@angular/core';
import {
  FormBuilder,
  FormGroup,
  Validators,
  AbstractControl
} from '@angular/forms';

@Component({
  selector: 'app-demo-form-with-validations-explicit',
  templateUrl: './demo-form-with-validations-explicit.component.html',
  styles: []
})
export class DemoFormWithValidationsExplicitComponent {
  myForm: FormGroup;
  sku: AbstractControl;

  constructor(fb: FormBuilder) {
    this.myForm = fb.group({
      'sku': ['', Validators.required]
    });

    this.sku = this.myForm.controls['sku'];
  }

  onSubmit(value: string): void {
    console.log('Poslali ste vrednost: ', value);
  }
}
```

Sledi celokupan listing pogleda komponente koji se nalazi u datoteci [src/app/demo-form-with-validations-explicit/demo-form-with-validations-explicit.component.html](#).

```
<div class="ui raised segment">
  <h2 class="ui header">Demo forme: sa eksplisicnom validacijom</h2>
  <form [formGroup]="myForm"
    (ngSubmit)="onSubmit(myForm.value)"
    class="ui form"
    [class.error]="!myForm.valid && myForm.touched">

    <div class="field"
      [class.error]="!sku.valid && sku.touched">
      <label for="skuInput">SKU</label>
```

```
<input type="text"
      id="skuInput"
      placeholder="SKU"
      [formControl]="sku">
<div *ngIf="!sku.valid"
      class="ui error message">SKU nije validan</div>
<div *ngIf="sku.hasError('required')"
      class="ui error message">SKU je obavezan</div>
</div>

<div *ngIf="!myForm.valid"
      class="ui error message">Forma nije validna</div>

<button type="submit" class="ui button">Pošalji</button>
</form>
</div>
```

UKLANJANJE OBJEKTNE PROMENLJIVE

Primenom osobine `myForm.controls` je moguće obaviti referenciranje `FormControl` objekata.

U prethodnom primeru podešena je objektna promenljiva `sku` sa tipom podataka `AbstractControl`. U praksi se često izbegava kreiranje ovakvih objekata i postavlja se pitanje kako je moguće obaviti referenciranje `FormControl` objekata bez postojanja odgovarajućih objektnih promenljivih.

Odgovor leži u primeni osobine `myForm.controls`, na sledeći način:

```
<label for="skuInput">SKU</label>
<input type="text"
      id="skuInput"
      placeholder="SKU"
      [formControl]="myForm.controls['sku']">
<div *ngIf="!myForm.controls['sku'].valid"
      class="ui error message">SKU nije validan</div>
<div *ngIf="myForm.controls['sku'].hasError('required')"
      class="ui error message">SKU je obavezan</div>
</div>
```

Ovaj pristup se često naziva skraćenom validacijom (*shorthand validation*). Omogućio je pristup `sku` kontroli bez uslovljavanja za dodavanjem njoj odgovarajuće varijable u klasu komponente.

```
export class DemoFormWithValidationsShorthandComponent implements OnInit {
  myForm: FormGroup;

  ngOnInit() {
  }
}
```

```
constructor(fb: FormBuilder) {  
  this.myForm = fb.group({  
    'sku': ['', Validators.required]  
  });  
}  
  
onSubmit(value: any): void {  
  console.log('Poslali ste vrednost:', value.sku);  
}  
}
```

Za demonstraciju ovog pristupa, u projektu je kreirana nova komponenta `demo-form-with-validations-shorthand`. Kompletan kod, klase i šablona komponente je dostupan na kraju objekta učenja u sekciji **Shared Resources.**

PRILAGOĐENA VALIDACIJA

Često je potrebno neko nestandardno pravilo validacije.

Još jedan slučaj se veoma često javlja kada veb programeri kreiraju forme u svojim veb aplikacijama - definiše se neko nestandardno pravilo validacije. Ovde je cilj da se upravo pokaže kako se rukuje ovakvim pravilima i kako se ona koriste prilikom provere ispravnosti forme i njenih polja.

Da bi bilo moguće sagledavanje implementacije samih validatora, prilaže se definicija najjednostavnijeg validatora `Validators.required` iz izvornog Angular koda:

```
export class Validators {  
  static required(c: FormControl): StringMap<string, boolean> {  
    return isBlank(c.value) || c.value == "" ? {"required": true} : null;  
  }  
}
```

Iz priloženog listinga se vidi da metoda validatora, kao argument, uzima objekat `FormControl` (kontrolu), a kao rezultat vraća objekat tipa `StringMap<string, boolean>` gde stringu odgovara kod greške, a logička vrednost je `true` ukoliko je validacija "pala".

PISANJE VLASTITOG VALIDATORA

Uvodi se novo pravilo za validaciju unosa u kontrolu sku.

Uvodi se novo pravilo za validaciju unosa u kontrolu **sku**: sku string mora sa počne sa "123". Da bi bila obezbeđena provera, po novom kriterijumu, neophodno je da programer napiše novi - prilagođeni validator. Sledi njegov listing:

```
function skuValidator(control: FormControl): { [s: string]: boolean } {  
  if (!control.value.match(/^123/)) {
```

```
    return {invalidSku: true};  
  }  
}
```

Ovaj validator će vratiti kod greške `invalidSku` ukoliko ulaz (`control.value`) ne počinje stringom "123".

Za demonstraciju ovog pristupa, u projektu je kreirana nova komponenta `demo-form-with-custom-validation`. Kompletan kod, klase i šablona komponente je dostupan na kraju objekta učenja u sekciji `Shared Resources`.

PRIDRUŽIVANJE VALIDATORA FORMCONTROL OBJEKTU

Novi problem - već postoji validator za kontrolu sku i kako je moguće dodati još jedan

Ako se malo detaljnije analizira trenutno stanje pokaznog primera moguće je uočiti jedan problem - već postoji validator za kontrolu `sku` i kako je moguće dodati još jedan?

Rešenje predstavlja primena metode `Validators.compose()`:

```
constructor(fb: FormBuilder) {  
  this.myForm = fb.group({  
    'sku': ['', Validators.compose([  
      Validators.required, skuValidator])]  
  });  
  
  this.sku = this.myForm.controls['sku'];  
}
```

Metoda `Validators.compose()` kao ulaz uzima niz validatora koji se primenjuju na neki `FormControl` objekat. Ovaj objekat nije validan ukoliko nije uspešno obavljena validacija po svim validatorima.

Sada je moguće primeniti kreirani validator u pogledu komponente:

```
<div *ngIf="sku.hasError('invalidSku')"  
  class="ui error message">SKU mora da počne sa <span>123</span></div>
```

Sada je moguće proveriti šta je do sada urađeno. Aplikacija se ponovo prevodi, da bi nova komponenta bila angažovana. Postoje dva slučaja:

- u kontrolu se unosi vrednost koja ne počinje stringom "123" - slika 4;
- u kontrolu se unosi vrednost koja počinje stringom "123" - slika 5.

Demo forme: sa prilagođenom validacijom

SKU

aa

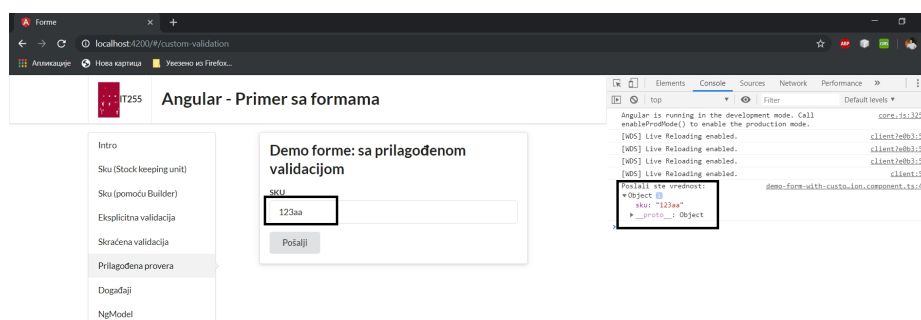
SKU nije validan

SKU mora da počne sa 123

Form nije validna

Pošalji

Slika 6.3.4 Primena prilagođenog validatora - validacija je "pala"



Slika 6.3.5 Primena prilagođenog validatora - validacija je "prošla"

PROMENE NA FORMI

Često je neophodno sagledati bilo koju promenu koja se dešava u nekoj kontroli.

U dosadašnjem izlaganju, dobijana je vrednost iz forme pozivom metode `onSubmit()` nakon klika na dugme za potvrđivanje forme. Često je neophodno sagledati bilo koju promenu koja se dešava u nekoj kontroli.

Obe klase, Both `FormGroup` i `FormControl` poseduju `EventEmitter` pa je na pojednostavljen način omogućeno praćenje promena.

Za praćenje promena u kontroli neophodno je sledeće:

- dobijanje pristupa objektu `EventEmitter` pozivom `control.valueChanges`;
- dodavanje osluškivača primenom `.subscribe()` metode.

Navedeno je realizovano sledećim listingom:

```
constructor(fb: FormBuilder) {  
  this.myForm = fb.group({  
    'sku': ['', Validators.required]  
  });  
  
  this.sku = this.myForm.controls['sku'];  
  
  this.sku.valueChanges.subscribe(  
    (value: string) => {  
      console.log('sku je promenjen u:', value);  
    }  
  );  
  
  this.myForm.valueChanges.subscribe(  
    (form: any) => {  
      console.log('forma je promenjena u:', form);  
    }  
  );  
}
```

Za demonstraciju ovog pristupa, u projektu je kreirana nova komponenta `demo-form-with-events`. Kompletan kod, klase i šablona komponente je dostupan na kraju objekta učenja u sekciji `Shared Resources`.

PRAĆENJE PROMENA NA FORMI

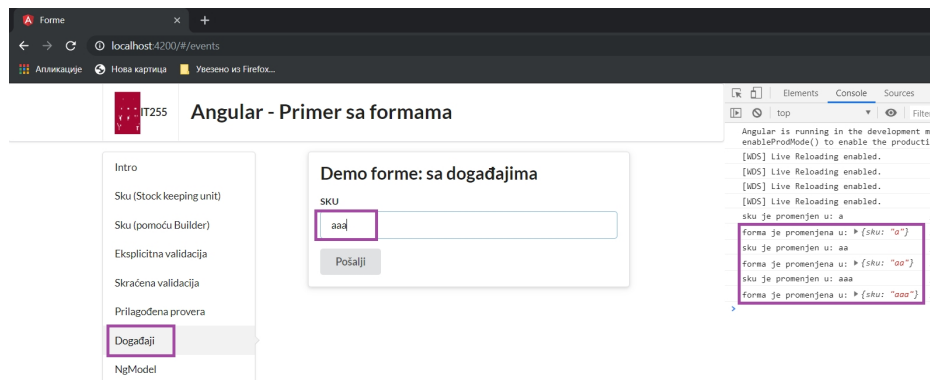
Osluškuju se događaji forme i kontrola.

Ovde sada postoji slučaj osluškivanja dva tipa razdvojenih događaja:

- događaji polja `sku`;
- događaj forme u celini.

Kada se postavi fokus na polje za unos, klik na taster tastature obavlja unos stringa u polje i osluškivač za ovu kontrolu je pokrenut. Kada kontrola promeni svoje stanje, osluškivač za formu je pokrenut.

Ako se pokrene tekući primer, praćenje promena na formi je moguće pratiti u konzoli veb pregledača, a to je demonstrirano sledećom slikom.



Slika 6.3.6 Praćenje promena na formi

▼ 6.5 Direktiva ngModel

NGMODEL

Direktiva ngModel predstavlja posebnu direktivu za povezivanje modela sa formom.

Direktiva ngModel predstavlja posebnu direktivu za povezivanje modela sa formom. Ovom **Angular** direktivom je omogućeno dvosmerno povezivanje podataka (*two-way data binding*). Dvosmerno povezivanje podataka je skoro uvek kompleksnije i teže za razumevanje nego što je jednosmerno koje je do sada bilo u fokusu. Okvir Angular, je u osnovi, osmišljen da daje podršku jednosmernom povezivanju podataka - od vrha ka dnu (*top-down*). Međutim, kada na scenu stupi rad sa formama, postoje slučajevi u kojima je lakše primeniti dvosmerno povezivanje podataka.

Neka **forma** doživi blagu izmenu - u formu se unosi string koji predstavlja naziv proizvoda (osobina *productName* klase komponente). Nakon izmene forme biće primenjena direktiva *ngModel* za održavanje sinhronizacije između kreirane objektno promenljive klase i pogleda (šablona komponente).

U prvom koraku će biti kreirana nova komponenta za demonstraciju primene direktive *ngModel*. Sledi listing njene klase:

```
export class DemoFormNgModelComponent {
  myForm: FormGroup;
  productName: string;

  constructor(fb: FormBuilder) {
    this.myForm = fb.group({
      'productName': ['', Validators.required]
    });
  }

  onSubmit(value: string): void {
```



```
    console.log('Poslali ste vrednost: ', value);  
  }  
}
```

Iz listinga je moguće primetiti postojanje nove objektno promenljive `productName`, tipa string.

Za demonstraciju ovog pristupa, u projektu je kreirana nova komponenta `demo-form-ng-model`. Kompletan kod, klase i šablona komponente je dostupan na kraju objekta učenja u sekciji `Shared Resources`.

NGMODEL I POGLEDI

Nakon kreiranja klase komponente, `ngModel` direktiva se primenjuje u odgovarajućem šablonu.

Nakon što je kreirana klasa komponente, `ngModel` direktivu je moguće primeniti u odgovarajućem šablonu.

```
<label for="productNameInput">Naziv proizvoda</label>  
  <input type="text"  
    id="productNameInput"  
    placeholder="Product Name"  
    [formControl]="myForm.get('productName')"  
    [(ngModel)]="productName">
```

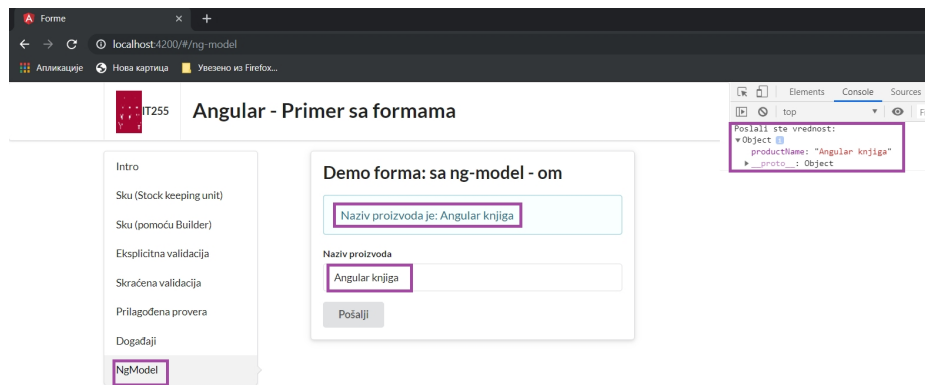
Ovde je sada moguće primetiti malo čudnu sintaksu. Istovremeno su i uglaste i male zagrade primenjene na atribut `ngModel`. Uglaste zagrade predstavljaju ulaz, a male izlaz. Sve ovo ukazuje na dvosmerno povezivanje podataka.

Takođe, moguće je primetiti još nešto: još uvek se koristi `formControl` za specificiranje da ovaj ulaz mora da bude u vezi sa `FormControl` objektom na formi. Ovo je neophodno uraditi iz razloga što `ngModel` samo povezuje objektnu promenljivu klase komponente od koje je `FormControl` u potpunosti odvojen. Međutim, pošto je i dalje potrebno koristiti validaciju ove vrednosti i obaviti njeno prosleđivanje kao dela forme, direktiva `formControl` je morala da bude zadržana.

Prikazivanje osobine klase `productName` u pogledu komponente može biti obavljeno na sledeći način:

```
<div class="ui info message">  
  Naziv proizvoda je: {{productName}}  
</div>
```

Sada je samo potrebno prevođenje primera, da bi poslednja komponenta mogla da bude angažovana. Rezultat izvršavanja je prikazan sledećom slikom.



Slika 6.4.1 Primena direktive ngModel

ANGULAR FORME - VIDEO MATERIJAL

Building Forms in Angular Apps; Reactive Forms - The Basics - video materijali

Building Forms in Angular Apps | Mosh - trajanje 25:03.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Reactive Forms - The Basics - trajanje 15:47

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

▼ Poglavlje 7

Vežba 7

POKAZNA VEŽBA - NOVA KOMPONENTA

Cilj vežbe je upoznavanje korisnika sa kreiranjem formi i validacijom polja.

Cilj vežbe je upoznavanje korisnika sa kreiranjem formi, kao i validacijom polja.

Neophodno je da prvo kreiramo novu komponentu za dodavanje video snimaka, koja će sadržati i formu. Već poznatom komandom kreiramo [add-video](#):

```
ng g c components/add-video
```

U okviru te komponente kreiramo [FormGroup](#) - u koja će sadržati podatke o video snimku:

```
public videoForm: FormGroup;
```

Neophodno je da inicijalizujemo formu sa poljima neophodnim za video snimak, a to su: [title](#), [description](#) i [link](#).

Funkcija za inicijalizaciju forme data je ispod:

```
public initForm() {
    this.videoForm = new FormGroup({
        title: new FormControl('', [
            Validators.required
        ]),
        link: new FormControl('', [
            Validators.required
        ]),
        description: new FormControl('', [
            Validators.required
        ])
    });
}
```

HTML kod forme je prikazan sledećim delom:

```
<form [formGroup]="videoForm" (ngSubmit)="submitForm()">
  <div class="col mb-4 single-input">
    <label for="title">Title</label>
    <input formControlName="title" type="text" name="title">
  </div>
```

```
<div class="col mb-4 single-input">
  <label for="description">Description</label>
  <input formControlName="description" type="text" name="description">
</div>

<div class="col mb-4 single-input">
  <label for="link">Link</label>
  <input formControlName="link" type="text" name="link">
</div>
<div class="col button-holder d-flex justify-content-center align-items-center
mt-4">
  <button type="submit" name="submit" class="btn btn-success
w-50">Dodaj</button>
</div>
</form>
```

Vidimo da je na dugme, kao i na samu formu nakačena funkcija `submitForm()`, koja bi trebalo da iskoristi prethodno sačuvane podatke za kreiranje novog Video snimka.

`submitForm()` metoda je data kodom ispod:

```
public submitForm() {
  let title = this.videoForm.get('title').value;
  let link = this.videoForm.get('link').value;
  let description = this.videoForm.get('description').value;
  let video = new Video(title, description, link);
  this.videoToAdd.emit(video);
}
```

OBJEDINJEN KOD KOMPONENTE ADDVIDEOCOMPONENT

Kod komponente `AddVideoComponent` je povezan u celinu.

Sam sadržaj klase `AddVideoComponent` je dat kodom ispod:

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
import { Video } from 'src/app/models/video';

@Component({
  selector: 'app-add-video',
  templateUrl: './add-video.component.html',
  styleUrls: ['./add-video.component.scss']
})
export class AddVideoComponent implements OnInit {
  public videoForm: FormGroup;
  @Output() videoToAdd: EventEmitter<Video>;
  constructor() { }
```

```

ngOnInit() {
  this.initForm();
  this.videoToAdd = new EventEmitter();
}

public initForm() {
  this.videoForm = new FormGroup({
    title: new FormControl('', [
      Validators.required
    ]),
    link: new FormControl('', [
      Validators.required
    ]),
    description: new FormControl('', [
      Validators.required
    ])
  });
}

public submitForm() {
  let title = this.videoForm.get('title').value;
  let link = this.videoForm.get('link').value;
  let description = this.videoForm.get('description').value;
  let video = new Video(title, description, link);
  this.videoToAdd.emit(video);
}
}

```

Kao što je već napomenuti, neophodno je da integrišemo i na nivou *AppComponent* slušanje za dodati video i dodavanje istog na listu video snimaka.

Prvenstveno je bitno da na nivou HTML-a izvršimo povezivanje (bind) događaja dodavanja. To se vrši sledećim kodom šablona komponente:

```
<app-add-video (videoToAdd)="addVideo($event)"></app-add-video>
```

Stoga je krajnji kod šablona ove komponente je sledeći:

```

<div class="container mt-4 mb-4">

  <div class="row mb-5 justify-content-between align-items-end">
    <div class="col-md-6">
      <app-add-video (videoToAdd)="addVideo($event)"></app-add-video>
    </div>
    <div class="col-md-4 mb-5">
      <h3>Search</h3>
      <input [(ngModel)]="searchText" placeholder="search text goes here">
    </div>
  </div>
  <div class="row">
    <div class="col-md-4" *ngFor="let video of videos | filter : searchText;

```

```
index as i; ">
    <app-video-box
      (videoToDelete)="deleteVideo($event)"
      (updateVideo)="updateVideo($event)"
      [video]="video"></app-video-box>
  </div>
</div>
</div>
```

MODIFIKACIJA CENTRALNE KOMPONENTE APP.COMPONENT.TS

Unutar fajla `app.component.ts` vršimo dodavanje video snimka pozivom funkcije `addVideo()`.

Unutar **TypeScript** fajla `app.component.ts` vršimo dodavanje video snimka pozivom funkcije `addVideo()`, koji dodaje element na listu komandom `push`.

```
import { Component } from '@angular/core';
import { Video } from '../models/video';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  public videos: Video[] = [];
  private _links: string[] = [
    'YiUQE5bJKFU',
    'B32yjbCSVpU',
    'yG0oBPtyNb0',
    'f7McpVPlidc',
    'qhZULM69DIw',
    'wvUQcnfwUUM'
  ];

  constructor(){
    for(let i = 0; i < 6; i++) {
      this.videos.push(new Video(this._generateString(3),
this._generateString(100), 'https://www.youtube.com/embed/' + this._links[i]))
    }
  }

  private _generateString(length) {
    let result = '';
    let characters =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    let charactersLength = characters.length;
    for ( let i = 0; i < length; i++ ) {
```

```
        result += characters.charAt(Math.floor(Math.random() * charactersLength));
    }
    return result;
}

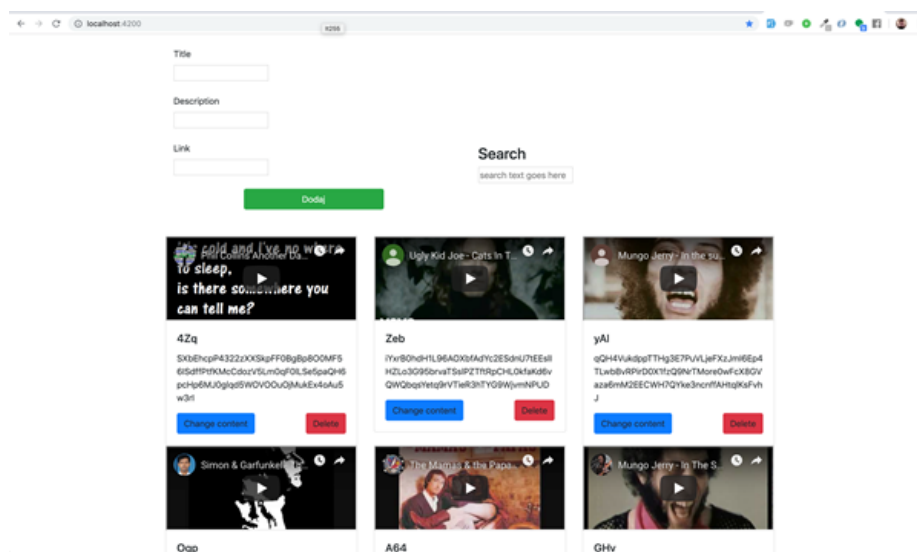
public deleteVideo(video: Video) {
    this.videos = this.videos.filter(item => {
        return item.title !== video.title
    })
}

public updateVideo(video: Video) {
    let index = this.videos.findIndex(i => i.title === video.title);
    this.videos[index].title = this._generateString(6);
}

public addVideo(video: Video) {
    this.videos.push(video);
}

}
```

Krajnji rezultat izvršavanja aplikacije je prikazan na sledecoj slici:



Slika 7.1 Krajnji rezultat izvršavanja aplikacije

NAPOMENA: Ukoliko bismo želeli da prikazemo svaki drugi element na osnovu broja indeksa, možemo iskoristiti direktivu *ngIf* na sledeći način:

```
<app-video-box
    *ngIf="i%2 == 0"
    (videoToDelete)="deleteVideo($event)"
    (updateVideo)="updateVideo($event)"
    [video]="video">
</app-video-box>
```

INDIVIDUALNA VEŽBA

Samostalna dopuna zadatka sa pokaznih vežbi.

Otvorite zadatak koji ste radili na pokaznim vežbama:

Dodajte nove funkcionalnosti u kreirani Angular projekat po sledećim zahtevima:

1. Dodati validacije unutar HTML kod, tako da se validno prikazuju.
2. Dodati osmatrača vrednosti polja *title*, koji će na svaku promenu ispisati određenu poruku unutar konzole.

▼ Poglavlje 8

Domaći zadatak 7

DOMAĆI ZADATAK

Samostalna izrada domaćeg zadatka.

Na projekat započet prethodnim domaćim zadatkom dodati sledeće funkcionalnosti:

1. Na domaći zadatak [MetHotels](#), dodati validaciju formi tako da nije moguće dodati nepravilno popunjene sobe.
2. Integrisati posmatranje vrednosti, tako da ukoliko je dužina vrednosti manja od 6 karaktera, da ispisuje određenu poruku unutar konzole.

Domaći zadatak je potrebno dostaviti kao [Github commit](#) na prethodni, pod nazivom „[IT255-DZ07](#)“. Link do zadatka poslati predmetnom asistentu na mail.

NAPOMENA: Domaći zadatak dodati kao [commit](#) na prethodni zadatak, a ne kreirati novi repozitorijum.

▼ Poglavlje 9

Zaključak

ZAKLJUČAK

Ova lekcija je pokrila primenu ugrađenih Angular direktiva u veb aplikacijama.

Kao što je istaknuto kroz izlaganje ove lekcije, *Angular* okvir obezbeđuje veliki broj ugrađenih direktiva, koje predstavljaju attribute koji se dodaju u *HTML* elemente i na taj način kreiraju dinamičko ponašanje *Angular* veb aplikacije. Lekcija je detaljno ispunila vlastiti cilj, a to je pokrivanje najčešće korišćenih ugrađenih direktiva kroz detaljnu analizu i diskusiju vođenu primenom adekvatnog primera.

Ukoliko je detaljno pratio uputstva lekcije, student je sada osposobljen da koristi osnovne ugrađene *Angular direktive* i da prvi put kreira dinamičku *Angular* veb aplikaciju.

Prateći primer, koji objedinjuje primenu navedenih direktiva, priložen je na kraju lekcije. Možete da ga preuzmete iz sekcije *Shared Resources*.

LITERATURA

Za pripremu lekcije korišćena je aktuelna pisana i elektronska literatura.

Pisana literatura:

1. Nate Murray, Felipe Coury, Ari Lerner, Carlos Taborda, ng-Book – The Complete Book on Angular 6, Fullstack.io, 2018
2. Cody Lindley, Frontend Handbook – 2017, Frontend masters, 2017
3. Sandeep Panda, AngularJS – From Novice to Ninja, SitePoint Pty. Ltd, 2014

Elektronska literatura:

4. <https://angular.io/>
5. <https://angular.io/tutorial>
6. <https://www.w3schools.com/angular/>
7. <https://www.tutorialspoint.com/angular4/>
8. <https://nodejs.org/en/>
9. <https://code.visualstudio.com/>
10. https://www.w3schools.com/whatis/whatis_html5dom.asp