# Angular šabloni i direktive

Prikaz podataka i obrada korisničke interakcije unutar Angular aplikacija postiže se interakcijom logike napisane na dve različite lokacije: unutar šablona i unutar klase. Klase komponenata kreiraju se korišćenjem TypeScript jezika, a šabloni upotrebom jezika HTML. Ipak, unutar šablona Angular komponenata moguće je koristiti brojne pristupe koji su specifični za Angular. Neke od takvih pristupa ste mogli da vidite u prethodnoj lekciji, na primeru prve Angular komponente koja je automatski generisana prilikom kreiranja projekta. U lekciji koja je pred vama detaljnije ćemo se posvetiti načinima za izgradnju šablona Angular komponenata.

## Sintaksa šablona Angular komponenata

Šabloni Angulara kreiraju se korišćenjem čistog HTML jezika. To praktično znači da su sve specifične Angular tvorevine ujedno validan HTML. Ipak, one za Angular imaju posebno značenje.

S obzirom na to da je HTML jezik Angular šablona, gotovo svaki HTML je ujedno i validan kod šablona Angular komponenata. Kaže se gotovo svaki, zato što Angular ne dozvoljava definisanje `script` elementa unutar šablona, i to iz više razloga. Za početak, `script` element unutar šablona stvara rizik od potencijalnih napada koji se zasnivaju na ubrizgavanju maliciozne skripte unutar prezentacionog sloja web aplikacija. Drugi razlog je jasna razdvojenost slojeva na kojoj se Angular zasniva. Programskoj logici nije mesto unutar šablona, već se takva logika smešta unutar klase komponente.

Svi ostali HTML elementi se mogu pojaviti unutar Angular šablona. Ipak, za korišćenje nekih HTML elemenata nema opravdane potrebe. Tako se unutar šablona nikada neće naći elementi kao što su `html`, `body` ili `base`, s obzirom na to da se oni definišu unutar statičkog `index.html` fajla, koji se nalazi na nivou Angular projekta i koji se dinamički popunjava sadržajem na osnovu šablona komponenata od kojih je sačinjena Angular aplikacija.

Unutar šablona Angular komponenata mogu se naći i određeni elementi koji u HTML jeziku ne predstavljaju validne elemente. To ste mogli videti i u prethodnoj lekciji, kada je za integraciju naše komponente unutar glavne komponente aplikacije iskorišćen element `<app-demo></app-demo>`. Reč je o elementu koji smo samostalno definisali kreiranjem nove komponente.

Osnovni zadatak specifičnih Angular tvorevina koje se mogu naći unutar šablona jeste postizanje različitih oblika povezivanja podataka i korišćenja specifične logike za uticanje na generisanje šablona. U narednim redovima prvo ćemo se pozabaviti pristupima za postizanje različitih vrsta povezivanja podataka, a nakon toga će biti prikazani različiti pristupi za dinamičku izgradnju strukture Angular šablona.

## Povezivanje podataka u Angularu

Povezivanje podataka (engl. *data binding*) je pojam koji se u programiranju koristi da označi pristup koji omogućava automatsko propagiranje promena podataka iz aplikativnog sloja do sloja prezentacije i propagiranje događaja iz sloja prezentacije nazad do sloja aplikativne logike. Kada je Angular u pitanju, to praktično znači da se povezivanje koristi za komunikaciju između šablona i TypeScript klase koja predstavlja Angular komponentu.

Angular poseduje brojne pristupe za postizanje povezivanja podataka, koji se mogu koristiti u zavisnosti od efekata koji se žele postići (tabela 15.1).

| Tip povezivanja | Sintaksa | Primer |
|---|---|---|
| Interpolacija (***Interpolation***) | `{{expression}}` | `{{message}}` |
| Povezivanje sa DOM svojstvima (***Property Binding***) | `[property]="expression"` | `[value]="message"` |
| Povezivanje sa atributima (***Attribute Binding***) | `[attr.attribute]="expression"` | `[attr.aria-label]="message"` |
| Povezivanje sa klasama (***Class Binding***) | `[class.css-class]="expression"` | `[class.selected]="isSelected"` |
| Povezivanje sa linijskom stilizacijom (***Style Binding***) | `[style.css-prop]="expression"` | `[style.font-size.px]="16"` |
| Povezivanje događaja (***Event Binding***) | `(event)="statement"` | `(change)="update($event)"` |
| Dvosmerno povezivanje (***Two-way Binding***) | `[(property)]="expression"` | `[(value)]="message"` |

*Tabela 15.1. Različiti pristupi za realizovanje povezivanja unutar Angulara*

Različiti tipovi povezivanja u Angularu koji su prikazani tabelom 15.1. biće demonstrirani u narednim redovima.

## Interpolacija i izrazi

Najjednostavniji način za postizanje povezivanja, odnosno za postizanje ispisa nekih podataka unutar Angular šablona, jeste upotreba interpolacije. Interpolacija se karakteriše upotrebom dva para vitičastih zagrada – **{{}}**, unutar kojih se navodi izraz koji će biti pretvoren u tekstualnu vrednost:

```
<p>{{name}}</p>
```

Na ovaj način, kao sadržaj paragraf elementa unutar šablona, biće postavljena vrednost promenljive `name`. Da bi ovakav primer funkcionisano, neophodno je da unutar klase koja predstavlja komponentu deklarišemo i inicijalizujemo svojstvo sa ovakvim imenom:

```
export class DemoComponent implements OnInit {
```

```
  name: string;
  constructor() {
    this.name = "John";
  }
  ngOnInit(): void {
  }
}
```

Nakon pokretanja aplikacije, unutar web pregledača moći ćete da vidite da je tekst `John` ispisan unutar stranice koja predstavlja našu Angular aplikaciju.

Interpolacija omogućava i definisanje izraza, pa je tako unutar vitičastih zagrada moguće napisati i nešto ovakvo:

```
<p>The sum of 2 + 2 is {{2 + 2}}</p>
```

Sada je unutar vitičastih zagrada za interpolaciju definisan klasičan JavaScript izraz. Unutar web pregledača će se dobiti sledeći ispis:

```
The sum of 2 + 2 is 4
```

**Pitanje**

Najjednostavniji oblik povezivanja podataka u Angularu naziva se:

**a) interpolacija**
a) intergalaksija
b) interponiranje
c) instanciranje

**Objašnjenje:**

*Najjednostavniji način za postizanje povezivanja, odnosno za postizanje ispisa nekih podataka unutar Angular šablona, jeste upotreba interpolacije.*

## Povezivanje podataka sa svojstvima DOM objekata

Nakon upoznavanja pristupa za postizanje jednostavne interpolacije, upoznaćemo se sa pristupom koji omogućava povezivanje sa svojstvima DOM objekata (engl. ***Property Binding***). Naime, Angular omogućava da se podaci povežu direktno sa svojstvima koja poseduju DOM objekti, i to njihovim postavljanjem unutar uglastih zagrada. Stoga sledeća dva primera proizvode identičan efekat:

```
<p>{{name}}</p>
<p [innerHtml]="name"></p>
```

Sadržaj prvog paragraf elementa kreiran je korišćenjem interpolacije. Za dobijanje sadržaja drugog elementa iskorišćen je pristup koji omogućava direktno povezivanje podataka sa svojstvima DOM objekata. S obzirom na to da DOM svojstvo `innerHtml` definiše unutrašnji sadržaj nekog elementa, oba paragrafa će imati identičan sadržaj.

**Napomena**

Veoma je bitno razumeti da se *Property Binding*om **ne** povezuju podaci sa atributima HTML elemenata, već isključivo sa svojstvima koja se nalaze unutar DOM objekta.

## Povezivanje sa atributima

Korišćenje povezivanja sa DOM svojstvima, kao u primeru iz prethodnog poglavlja, preporučeni je način za definisanje vrednosti HTML atributa. Naime, većina HTML atributa poseduje nazive koji odgovaraju nazivima DOM svojstava i u takvim situacijama se preporučuje upotreba pristupa prikazanog u prethodnom primeru. Evo još jednog takvog primera:

```
<p [title]="user.name">{{user.email}}</p>
```

U prikazanom primeru su obavljena dva povezivanja. Jedno povezivanje je postignuto jednostavnom interpolacijom, kojom je postavljen sadržaj paragraf elementa. Drugo povezivanje je postignuto korišćenjem pristupa koji omogućava definisanje vrednosti DOM svojstava. Za vrednost DOM svojstva `title` je postavljeno ime osobe. Kako bi ovakav primer ispravno funkcionisao, unutar TypeScript klase, koja predstavlja Angular komponentu, potrebno je da se nađe promenljiva sa nazivom `user`, koja ukazuje na jedan objekat:

```
export class DemoComponent implements OnInit {

  user: any;

  constructor() {
    this.user = {
      id: 1,
      name: "Ben Torrance",
      email: "ben@email.com",
      balance: 55353.93434,
      code: "5n4n5n"
    };

  }

  ngOnInit(): void {
  }

}
```

`id` atribut možemo dodati na identičan način:

```
<p [title]="user.name" [id]="user.id">{{user.email}}</p>
```

Sada će naš paragraf element na sebi posedovati dva atributa, `title` i `id`, sa odgovarajućim vrednostima koje dolaze iz TypeScript klase.

Ipak, neki atributi ne poseduju pripadajuće DOM svojstvo, pa se u takvim situacijama ne može iskoristiti upravo prikazani pristup. Jedan od takvih HTML atributa je i aria-label. Kako bi se definisala njegova vrednost, unutar uglastih zagrada je potrebno postaviti prefiks **attr.**

i na taj način će Angular znati da je reč o HTML atributu koji ne poseduje pripadajuće DOM svojstvo:

```
<p [attr.aria-label]="user.name">{{user.email}}</p>
```

## Definisanje klasa

Definisanje klasa unutar šablona Angular komponenata još jedan je aspekt povezivanja koji funkcioniše na specifičan način. Naime, DOM poznaje svojstvo `classList`, ali Angular uvodi odrednicu **class,** koja se unutar uglastih zagrada koristi za definisanje klasa.

Ukoliko je potrebno kontrolisati pojavljivanje jedne klase na nekom HTML elementu, onda se može koristiti sledeći pristup:

```
<p [class.deleted]="user.isDeleted">{{user.email}}</p>
```

Na ovaj način postojanje klase `.deleted` na paragraf elementu je uslovljeno vrednošću svojstva `isDeleted`, koje se nalazi unutar `user` objekta. Stoga se, kako bi primer funkcionisao, unutar objekta `user` dodaje svojstvo `isDeleted`:

```
{
  id: 1,
  name: "Ben Torrance",
  email: "ben@email.com",
  balance: 55353.93434,
  code: "5n4n5n",
  isDeleted: true
}
```

Kada je vrednost svojstva `isDeleted` `true`, baš kao u primeru, na paragraf elementu će postojati klasa `.deleted` i obrnuto.

Vrednost `class` atributa je direktno moguće povezati sa vrednošću nekoga svojstva unutar TypeScript klase, te ja na taj način moguće rukovati većim brojem klasa odjednom:

```
<p [class]="userClasses"></p>
```

Sada će vrednost `class` atributa da bude uslovljena vrednošću `userClasses` svojstva, koje se nalazi unutar TypeScript klase:

```
export class DemoComponent implements OnInit {

  userClasses:string;

  constructor() {
    this.userClasses = "user-class-1 user-class-2 user-class-3";
  }

  ngOnInit(): void {
  }

}
```

Uzimajući u obzir vrednost svojstva `userClasses`, paragraf element će nakon renderovanja da izgleda ovako:

```
<p class="user-class-1 user-class-2 user-class-3"></p>
```

Vrednost promenljive koja se vezuje sa `class` atributom ne mora biti `string` tipa, već je dozvoljeno definisanje vrednosti i u obliku nizova i objekata. Tako je identično moguće postići i na sledeći način:

```
this.userClasses = ["user-class-1", "user-class-2", "user-class-3"];
```

Kada se svojstvo koje se povezuje sa `class` atributom definiše kao objekat, moguće je uticati na pojavljivanje svake klase pojedinačno:

```
this.userClasses = {
  "user-class-1": true,
  "user-class-2": false,
  "user-class-3": true
}
```

Sada će se na paragraf elementu naći klase `.user-class-1` i `.user-class-3`, ali ne i klasa `.user-class-2`, zbog postojanja `false` vrednosti unutar objekta.

## Definisanje stilizacije

Uticanje na vrednost `style` HTML atributa još jedan je segment koji se posebno tretira prilikom definisanja povezivanja unutar Angular šablona. Za obavljanje takvog posla Angular definiše prefiks **style,** koji se koristi unutar uglastih zagrada. Kao i prilikom definisanja klasa, moguće je definisati vrednosti jednog CSS svojstva ili više njih odjednom.

Vrednost jednog CSS svojstva se može definisati na sledeći način:

```
<p [style.width]="width"></p>
```

Nakon specijalnog prefiksa **style**, navodi se karakter tačka (**.**), a onda naziv CSS svojstva čija vrednost se postavlja. Naravno, prikazani primer podrazumeva postojanje svojstva `width` unutar TypeScript klase komponente:

```
this.width = "300px";
```

Vrednost svojstva je `string` tipa, kako bi unutar sebe, pored vrednosti, mogla da objedini i jedinicu (`px`). Definisanje jedinica se može prepustiti sistemu za povezivanje Angulara, pa je tako primer moguće transformisati na sledeći način:

```
<p [style.width.px]="width"></p>
```

Nakon svojstva `width`, dodata je odrednica koja se odnosi na jedinicu koja će automatski da bude dodata nakon vrednosti CSS svojstva. Sada svojstvo koje definiše širinu elementa ne mora biti `string` tipa:

```
this.width = 300;
```

Oba prikazana primera su ilustrovala definisanje vrednosti samo jednog CSS svojstva. Pored takvog pristupa, moguće je definisati i veći broj CSS opisa odjednom. To se postiže na identičan način kao i prilikom rada sa klasama:

```
<p [style]="userStyles"></p>
```

Promenljiva `userStyles` izgleda ovako:

```
this.userStyles = "width: 200px; height: 200px";
```

Kao i prilikom korišćenja klasa, vrednost svojstva `userStyles` je moguće definisati i u obliku niza ili objekta.

## Obrada događaja

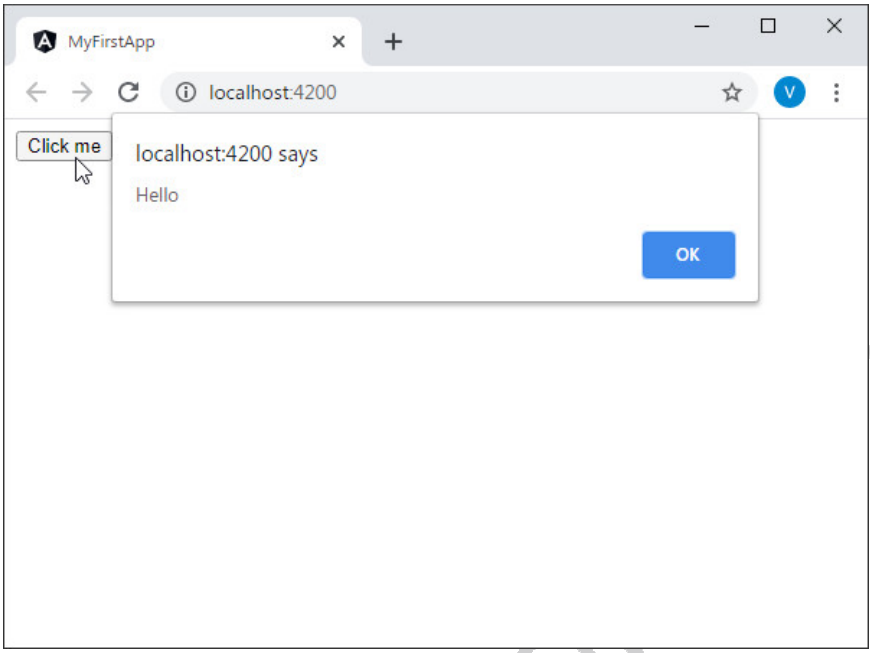
Slušanje događaja se unutar Angular šablona komponenata može obaviti korišćenjem DOM naziva događaja, koji se smeštaju unutar jednog para običnih zagrada. Tako se formira nešto nalik na atribut, čija vrednost sadrži logiku koja je potrebno da se izvrši prilikom pojave događaja ili na tu logiku upućuje:

```
<button (click)="sayHello()">Click me</button>
```

Na ovaj način je obavljena pretplata na `click` događaj na jedan `button` element i definisano je da će se tom prilikom aktivirati funkcija `sayHello()` koja se nalazi unutar klase koja predstavlja komponentu:

```
export class DemoComponent implements OnInit {
  sayHello() {
    alert("Hello");
  }

  constructor() {
  }

  ngOnInit(): void {
  }

}
```

Klikom na `button` element prikazuje se modalni prozor sa porukom (slika 15.1).




*Slika 15.1. Efekat logike za obradu događaja korišćenjem Angulara*

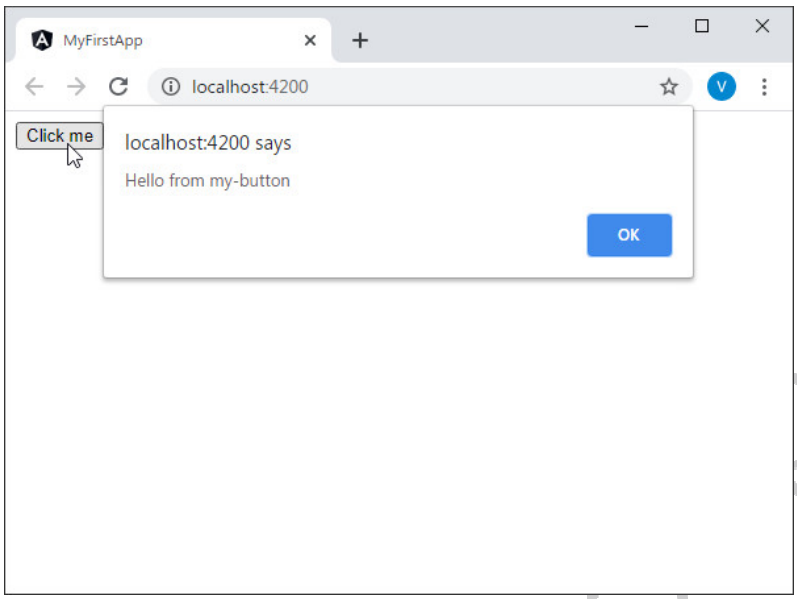
Prilikom obrade događaja, funkciji je moguće proslediti i objekat koji predstavlja događaj:

```
<button id="my-button" (click)="sayHello($event)">Click me</button>
```

Parametar `$event` se unutar funkcije može iskoristiti na sledeći način:

```
sayHello(event) {
  let elemId = event.target.id;
  alert("Hello from " + elemId);
}
```

Korišćenjem objekta događaja prvo se dolazi do DOM elementa koji je izazvao događaj, a zatim i do vrednosti njegovog `id` atributa. To na kraju za rezultat ima efekat kao na slici 15.2.




*Slika 15.2. Efekat logike za obradu događaja korišćenjem Angulara (2)*

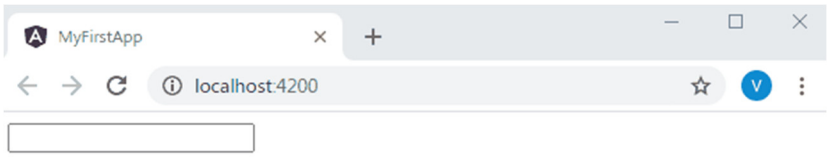
Evo još jednog primera obrade događaja koji će ilustrovati još neke osobine takvog procesa. Šablon izgleda ovako:

```
<input (input)="message=$event.target.value">
<div>{{message}}</div>
```

Šablon se sastoji iz dva elementa, odnosno iz jednog `input` elementa i jednog `div` elementa. Unutar `input` elementa se unosi neki tekst, koji zatim odmah biva prikazan unutar `div` elementa ispod. Sve to je postignuto bez bilo kakve funkcije za obradu događaja, korišćenjem samo jednog svojstva koje se nalazi unutar klase komponente:

```
export class DemoComponent implements OnInit {

  message: string;

  constructor() {
  }

  ngOnInit(): void {
  }

}
```

Na `input` elementu se sluša događaj `input`, koji se aktivira prilikom svake promene vrednosti unutar `input` elementa. Tom prilikom se aktivira logika koja automatski postavlja vrednost svojstva `message`, korišćenjem vrednosti svojstva `value` DOM objekta, koji predstavlja `input` element. Unutar `div` elementa se takva vrednost prikazuje korišćenjem interpolacije. Sve to na kraju proizvodi efekat kao na animaciji 15.1.




*Animacija 15.1. Povezivanje dva HTML elementa*

Upravo prikazani primer predstavljao je bazičan način za definisanje dvosmernog povezivanja. Nešto kasnije u ovoj lekciji moći ćete da vidite i specijalizovane načine za obavljanje takvog posla.

Osnovne ugrađene direktive

Angular poseduje veliki broj ugrađenih direktiva koje je moguće koristiti za obavljanje različitih zahvata nad šablonom. Za početak ćemo se upoznati sa tri osnovne direktive Angulara:

- `NgClass` – direktiva za dodavanje i uklanjanje klasa,
- `NgStyle` – direktiva za dodavanje i uklanjanje CSS opisa,
- `NgModel` – direktiva za postizanje dvosmernog povezivanja.

### NgClass

Direktiva `NgClass` može da se koristi za postavljanje klasa na DOM elementima. Tako je korišćenjem ove direktive moguće postići sve ono što je nešto ranije postignuto korišćenjem pristupa koji je podrazumevao prefiks `class` unutar uglastih zagrada – **[class...]**

Vrednost `NgClass` direktive može biti neko svojstvo definisano unutar klase komponente:

```
<div [ngClass]="myClasses">This is my div</div>
```

Na ovaj način na element će da budu postavljene sve klase koje su definisane svojstvom `myClasses`:

```
export class DemoComponent implements OnInit {

  myClasses: string;

  constructor() {
    this.myClasses = ".user-class-1 .user-class-2 .user-class-3";
  }

  ngOnInit(): void {
  }

}
```

Vrednost `NgClass` direktive može biti i jednostavan JavaScript (TypeScript) izraz:

```
<div [ngClass]="isDeleted ? 'deleted' : ''">This is my div</div>
```

Sada se, u zavisnosti od vrednosti jednog `boolean` svojstva, na element postavlja klasa `.deleted`. Svojstvo `isDeleted` unutar klase komponente izgleda ovako:

```
export class DemoComponent implements OnInit {

  isDeleted: boolean;

  constructor() {
    this.isDeleted = true;
  }

  ngOnInit(): void {
  }

}
```

**NgStyle**

Direktiva `NgStyle` može se koristiti za definisanje linijske stilizacije na DOM elementima. Ipak je bitno znati da ova direktiva ne može da prihvati `string` vrednost, kao što je to bio slučaj prilikom korišćenja pristupa koji su za definisanje stilizacije prikazani nešto ranije. Vrednost `NgStyle` direktive mora biti objekat, stoga je reč o preporučenom načinu za definisanje većeg broja CSS opisa odjednom:

```
<div [ngStyle]="myStyles">This is my div</div>
```

Na ovaj način će linijska stilizacija biti definisana u zavisnosti od vrednosti objekta `myStyles`:

```
export class DemoComponent implements OnInit {
  myStyles: any;
  constructor() {
```

```
    this.myStyles = {
      'font-style': 'normal',
      'font-weight': 'bold',
      'font-size': '24px'
    };
  }
  ngOnInit(): void {
  }

}
```

## Dvosmerno povezivanje korišćenjem direktive NgModel

Gotovo svi do sada prikazani primeri podrazumevali su jednosmerna povezivanja, kojima su podaci propagirani do šablona ili su događaji prosleđivani aplikativnoj logici unutar TypeScript klase. Još jedna vrsta povezivanja čije kreiranje omogućava Angular jeste povezivanje u kome se promene propagiraju u oba smera. Takvo povezivanje se naziva dvosmerno povezivanje. Jedan od prethodnih primera je ilustrovao bazičan način za postizanje dvosmernog povezivanja, koji je podrazumevao kombinovanje sintakse za slušanje događaja i povezivanje svojstava. Ipak, Angular poseduje i specijalizovane elemente kojima je dvosmerno povezivanje moguće postići na mnogo jednostavniji i efektniji način.

Dvosmerno povezivanje se najlakše realizuje korišćenjem `NgModel` direktive. Preduslov za korišćenje takve direktive jeste da je uključen `FormsModule` u pokrenutom projektu. Tako prvi put dolazimo do tačke u kojoj samostalno moramo u našoj Angular aplikaciji da uključimo neku dodatnu Angular biblioteku. Iz jedne od prethodnih lekcija znate da se takvo nešto obavlja u fajlu u kojem je definisan koreni modul naše aplikacije:

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

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

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

U prikazanom kodu koji predstavlja sadržaj `app.module.ts` fajla potrebno je da primetite dve novine:

unutar svojstva `imports` dodat je modul `FormsModule`,

na početak dokumenta je dodata direktiva za importovanje modula `@angular/forms`, u kome se nalaze potrebne funkcionalnosti za funkcionisanje `NgModel` direktive.

Nakon uključivanja potrebnog Angular modula, možemo preći na kreiranje primera koji će podrazumevati korišćenje `NgModel` direktive. Šablon će izgledati ovako:

```
<input [(ngModel)]="message">
<div>{{message}}</div>
```

Primer je sličan već prikazanom primeru iz dosadašnjeg toka ove lekcije. Jednostavno, želimo da se promene vrednosti promenljive `message` propagiraju u oba pravca, odnosno i iz aplikativne logike ka šablonu, ali i obrnuto, iz šablona ka aplikativnoj logici. Zbog toga je iskorišćena direktiva `NgModel`, koja je postavljena unutar dva para zagrada. Spoljašnje zagrade su uglaste, a unutrašnje obične, pa se ovakva sintaksa veoma često naziva banana u kutiji (engl. ***banana in a box***).

U klasi TypeScript definisano je svojstvo `message`:

```
export class DemoComponent implements OnInit {
  message: string;
  constructor() {
    this.message = "This is initial value";
  }

  ngOnInit(): void {
  }

}
```

Bitno je da primetite da je u klasi TypeScript obavljena inicijalizacija svojstva `message`. Njegova vrednost je postavljena na *This is initial value*. Takva vrednost će da bude prosleđena iz klase ka šablonu, dok će zatim promenom teksta u `input` elementu da bude obavljeno propagiranje promena u suprotnom smeru – iz šablona ka aplikativnoj logici. U to se možemo uveriti na osnovu sadržaja `div` elementa koji se nalazi u šablonu, a koji uvek odslikava vrednost svojstva `message`.

## Strukturalne direktive

Pored direktiva koje se koriste za postizanje povezivanja podataka, Angular poseduje i skup direktiva kojima se može uticati na strukturu elemenata u šablonu. Takve direktive se nazivaju strukturalne direktive:

- `NgIf` – omogućava uslovno kreiranje ili uništavanje elemenata u šablonu Angular komponenata,
- `NgFor` – omogućava generisanje prezentacije korišćenjem petlje, odnosno ponavljanjem određene logike više puta,
- `NgSwitch` – omogućava uslovno generisanje elemenata korišćenjem sintakse uslovne konstrukcije switch.

**NgIf**

Direktiva `NgIf` omogućava uslovno renderovanje u zavisnosti od ishoda nekog JavaScript izraza ili vrednosti logičke promenljive:

```
<div *ngIf="isVisible" >This is my div.</div>
```

Ovakav `div` element će biti renderovan samo kada promenljiva `isVisible` ima vrednost `true`:

```
export class DemoComponent implements OnInit {

  isVisible: boolean;

  constructor() {
    this.isVisible = false;
  }

  ngOnInit(): void {
  }

}
```

Direktiva `NgIf` konstantno prati vrednost promenljive `isVisible`, pa tako uništava i dodaje ovakav `div` element u zavisnosti od vrednosti promenljive `isVisible`.

**NgFor**

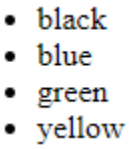
`NgFor` direktiva vrlo je korisna kada je potrebno kreirati određeni prikaz na osnovu neke kolekcije podataka:

```
<ul>
    <li *ngFor="let item of items">{{item}}</li>
</ul>
```

Na ovaj način biće obavljeno generisanje elemenata tipa `li`. Broj elemenata koji će biti generisani zavisi od broja elemenata koji se nalaze unutar niza `items`:

```
export class DemoComponent implements OnInit {

  items: any;

  constructor() {
    this.items = ["black", "blue", "green", "yellow"];
  }

  ngOnInit(): void {
  }

}
```

Niz `items` je definisan unutar klase komponente. Možete videti da on poseduje četiri člana, pa će na kraju unutar stranice da bude dobijen prikaz kao na slici 15.3.




*Slika 15.3. Primer liste generisane korišćenjem direktive NgFor*

**NgSwitch**

Još jedan način za postizanje uslovnog generisanja koda u Angular šablonu jeste korišćenje direktive `NgSwitch`. Direktiva `NgSwitch` je zapravo skup od tri direktive: `NgSwitch`, `NgSwitchCase` i `NgSwitchDefault`. Primer korišćenja ovih direktiva mogli ste da vidite u prethodnoj lekciji, prilikom demonstracije funkcionalnosti stranice koja nastaje generisanjem projekta. U narednim redovima će biti prikazan primer u kome se objedinjuje nekoliko direktiva prikazanih u ovoj lekciji, između ostalih i direktiva koje omogućavaju uslovno generisanje u stilu uslovnog bloka `switch`. Evo kako će izgledati logika unutar klase komponente:

```
export class DemoComponent implements OnInit {

  items: any;
  selectedIndex: number;

  constructor() {
    this.items = ["black", "blue", "green", "yellow"];
    this.selectedIndex = -1;
  }

  ngOnInit(): void {
  }

}
```

Unutar klase komponente definisana su dva svojstva:

- `items` – niz stavki,
- `selectedIndex` – indeks trenutno selektovane stavke.

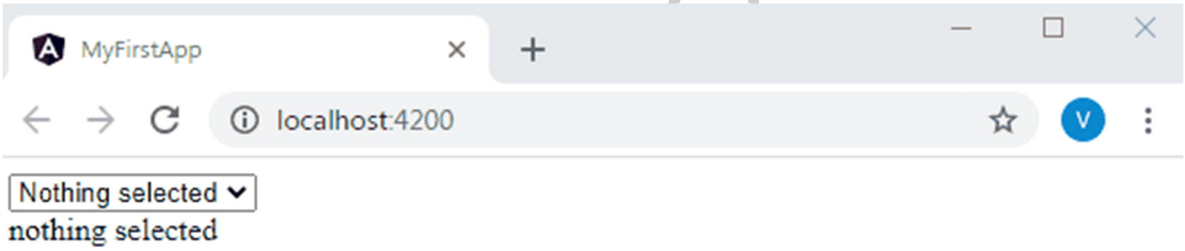
Ovakva dva svojstva se unutar šablona koriste na sledeći način:

```
<select name="colors" [(ngModel)]="selectedIndex">
    <option value="-1">Nothing selected</option>
    <option *ngFor="let item of items; ; let i = index"
value="{{i}}">{{item}}</option>
</select>
```

```
<div [ngSwitch]="selectedIndex">
    <div *ngSwitchDefault>nothing selected</div>
    <div *ngSwitchCase='0'>{{items[selectedIndex]}}</div>
    <div *ngSwitchCase='1'>{{items[selectedIndex]}}</div>
    <div *ngSwitchCase='2'>{{items[selectedIndex]}}</div>
    <div *ngSwitchCase='3'>{{items[selectedIndex]}}</div>
</div>
```

Unutar šablona su iskorišćeni brojni pristupi o kojima je bilo reči u ovoj lekciji. Za početak tu je jedan `select` element, čije se stavke dinamički generišu na osnovu članova niza `items`. Bitno je da primetite da je na `select` elementu definisano dvosmerno povezivanje korišćenjem direktive `NgModel`. Na taj način će promena selekcije unutar `select` elementa automatski da bude propagirana do promenljive `selectedIndex`, u koju će biti upisan indeks trenutno selektovane stavke.

Vrednost promenljive `selectedIndex` zatim se koristi za uslovno renderovanje korišćenjem `NgSwitch` direktive. Vrednost promenljive `selectedIndex` je uslov koji se koristi za odabir `div` elementa koji će biti renderovan na stranici. Tako se na osnovu selektovane stavke unutar `select` elementa obavlja dinamičko renderovanje odgovarajućeg `div` elementa (animacija 15.2).




*Animacija 15.2. Primer dinamičkog generisanja elemenata na osnovu izabrane stavke unutar select elementa*

## Rezime

- Angular šabloni se kreiraju korišćenjem čistog HTML jezika.
- Angular ne dozvoljava definisanje `script` elementa unutar šablona.
- Povezivanje podataka (engl. *data binding*) je pojam koji se u programiranju koristi da označi pristup koji omogućava automatsko propagiranje promena podataka između različitih slojeva aplikacije.
- Najjednostavniji način za ispisivanje nekih podataka unutar Angular šablona jeste upotreba interpolacije, koja se karakteriše upotrebom dva para vitičastih zagrada – `{{}}`.
- Property Binding omogućava povezivanje sa svojstvima DOM objekata.
- Kako bi se obavilo povezivanje sa nekim atributom koji ne poseduje pripadajuće DOM svojstvo, unutar uglastih zagrada kojima se definiše povezivanje potrebno je postaviti prefiks `attr`.
- Za povezivanje sa `class` HTML atributom koriste se uglaste zagrade, unutar kojih se navodi prefiks `class`.
- Za uticanje na vrednost `style` HTML atributa Angular definiše prefiks `style`, koji se koristi unutar uglastih zagrada.
- Slušanje događaja se unutar šablona Angular komponenata može obaviti korišćenjem DOM naziva događaja, koji se smešta unutar jednog para običnih zagrada.
- Angular poseduje veliki broj ugrađenih direktiva koje je moguće koristiti za obavljanje različitih zahvata nad šablonom.
- Direktiva `NgClass` koristi se za postavljanje klasa na DOM elementima.
- Direktiva `NgStyle` koristi se za definisanje linijske stilizacije na DOM elementima.
- Direktiva `NgModel` koristi se za realizaciju dvosmernog povezivanja.
- Preduslov za korišćenje `NgModel` direktive jeste uključen `FormsModule` unutar tekućeg projekta.
- Unutar modula `@angular/forms` nalaze se potrebne funkcionalnosti za funkcionisanje `NgModel` direktive.
- Direktiva `NgModel` postavlja se unutar dva para zagrada – `[()]`; spoljašnje zagrade su uglaste, a unutrašnje obične, pa se ovakva sintaksa veoma često naziva *banana in a box.*
- Strukturalne direktive u Angularu su `NgIf`, `NgFor` i `NgSwitch`.
- Direktiva `NgIf` omogućava uslovno kreiranje ili uništavanje elemenata unutar šablona Angular komponenata.
- Direktiva `NgFor` omogućava generisanje prezentacije korišćenjem petlje, odnosno ponavljanjem određene logike više puta.
- Direktiva `NgSwitch` omogućava uslovno generisanje elemenata korišćenjem sintakse uslovne konstrukcije switch.