



IT255 - VEB SISTEMI 1

TypeScript i Angular

Lekcija 06

PRIRUČNIK ZA STUDENTE

# IT255 - VEB SISTEMI 1

## Lekcija 06

### *TYPESCRIPT I ANGULAR*

- ✓ TypeScript i Angular
- ✓ Poglavlje 1: TypeScript
- ✓ Poglavlje 2: Tipovi podataka u TypeScript - u
- ✓ Poglavlje 3: Klase
- ✓ Poglavlje 4: Programski alati
- ✓ Poglavlje 5: Funkcionisanje Angular okvira
- ✓ Poglavlje 6: Analiza – arhitektura podataka
- ✓ Poglavlje 7: Vežba 6
- ✓ Poglavlje 8: Domaći zadatak 6
- ✓ 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

*Lekcija je podeljena u dva dela: TypeScript diskusija i funkcionisanje Angular okvira.*

Lekcija je podeljena u dva dela:

- *TypeScript* diskusija i
- funkcionisanje *Angular* okvira.

U prvom delu će posebno biti govora o migraciji Angular okvira sa standardnog JavaScript jezika ka njegovom proširenju u formi TypeScript notacije. Posebno će biti diskusije o:

- prednostima primene *TypeScript* okvira;
- primeni tipova podataka u *TypeScript* okviru;
- primeni objektno - orijentisanih koncepata *TypeScript* okvira;
- programskim alatima koji olakšavaju kodiranje u ovom okviru.

Drugi deo lekcije je rezervisan za analizu funkcionisanja Angular okvira sa akcentom na sledećim temama:

- aplikacija kao "top - level" komponenta;
- modeli u Angular aplikacijama;
- kreiranje i primena komponentata u Angular aplikacijama;
- analiza arhitekture podataka u Angular okviru.

## ▼ Poglavlje 1

# TypeScript

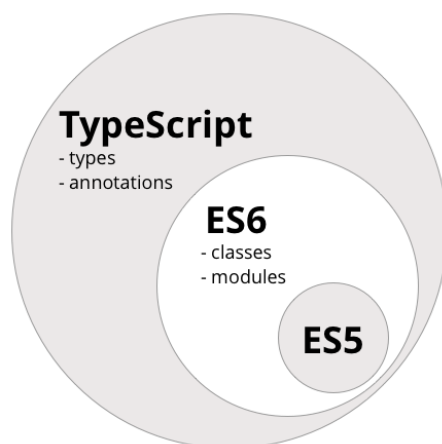
## ANGULAR I JEZICI

*Angular je baziran na jeziku TypeScript.*

Angular je baziran na jeziku TypeScript koji predstavlja proširenje dobro poznatog jezika JavaScript.

Za nekoga može biti problematično korišćenje novog jezika samo za okvir kao što je Angular, postoje brojni razlozi zašto je dobro koristiti TypeScript umesto čistog JavaScript jezika.

Posebno je važno napomenuti da TypeScript nije potpuno nov jezik, On predstavlja proširenje JavaScript ES6 koda. Ukoliko se koristi čist JavaScript ES6 kod on je savršeno kompatibilan sa TypeScript kodom. Sledećom slikom je prikazan dijagram prikazan je odnos između navedenih jezika:



Slika 1.1 Odnos između povezanih skripting jezika

TypeScript predstavlja rezultat zvanične saradnje kompanija *Microsoft* i *Google*. Sa ovako snažnom podrškom sasvim je jasno da će jezik biti dugo vremena aktuelan. Oba partnera su u obavezi da unapređuju veb pa samim tim programeri koji koriste Angular i TypeScript su na velikom dobitku.

Kod ovakvih jezika (*transpilers*) je odlično što relativno mali timovi mogu da kreiraju unapređenja jezika, a da ne moraju od proizvođača da traže da prilagode i unaprede veb pregledače.

# TYPESCRIPT PREDNOSTI U ODNOSU NA ČIST JAVASCRIPT

*Postoji pet velikih unapređenja u odnosu na čist JavaScript*

*TypeScript* donosi pet velikih unapređenja u odnosu na čist *JavaScript*:

- podrška za tipove podataka;
- podrška za klase;
- primena dekoratora;
- primena import instrukcija;
- primena raznih jezičkih alata, poput destrukcije.

O navedenim unapređenjima će u nastavku biti detaljno razmatrano.

## ▼ Poglavlje 2

# Tipovi podataka u TypeScript - u

## UVOĐENJE TIPOVA PODATAKA U JS KOD

*Primena tipova podataka je opcionalna u TypeScript - u.*

Glavno unapređenje TypeScript jezika, u odnosu na čist JavaScript, a po čemu je jezik i dobio ime, je uvođenje sistema tipova podataka.

Neki programeri su smatrali prednošću primene JavaScript jezika izbegavanje primene tipova podataka prilikom kodiranja. Međutim, pristup koji podrazumeva primenu tipova u ovakvom skripting jeziku, sigurno zaslužuje šansu ili bar vredno diskutovanja.

Međutim, velika prednost primene koncepta tipa dodataka sastoji se u:

1. pomoći prilikom pisanja koda jer sprečava pojavu grešaka (bug - ova) tokom vremena prevođenja programa;
2. pomoći prilikom čitanja koda jer jasnije pokazuje namere programera.

Takođe, vredno je napomenuti, primena tipova podataka je opcionalna u TypeScript - u. To znači, ako programer želi brzo da kreira određenu količinu koda može da preskoči navođenje tipova podataka. Kasnije, kada kod dostigne određeni nivo zrelosti, programer može da se vrati i da, na odgovarajućim mestima u kodu, doda tipove podataka.

TypeScript osnovni tipovi podataka u potpunosti odgovaraju tipovima koje se "implicitno" koriste u čistom JavaScript kodu:

- string;
- number;
- boolean, itd.

Sve do JavaScript ES5, varijable su bile definisane preko ključne reči var, na primer:

```
var ime;
```

Nova TypeScript sintaksa predstavlja prirodnu evoluciju JavaScript ES5. I dalje se koristi ključna reč var ali sada je moguće, opcionalno, obezbediti i tip podataka na sledeći način:

```
var ime : string;
```

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

## POVRATNE VREDNOSTI I ARGUMENTI FUNKCIJA

*TypeScript omogućava primenu tipova podataka na argumente i povratne vrednosti funkcija.*

Nova TypeScript sintaksa omogućava primenu tipova podataka na argumente i povratne vrednosti funkcija. Dat je sledeći izolovani listing funkcije `greetText()`

```
function greetText(name: string): string {  
    return "Hello " + name;  
}
```

U konkretnom primeru, definisana je nova funkcija pod nazivom `greetText()` koja uzima jedan argument, tipa `string`, naziva `name`. Studentima, na ovom nivou školovanja, trebalo bi da bude jasno da kod neće biti preveden ukoliko se obavi poziv ove funkcije preko argumenta koji ne pripada tipu podataka `string` i da će prevodilac javiti grešku. Ovo, naravno, predstavlja dobru stvar jer bi, u suprotnom, program radio neispravno.

Takođe, u definiciji funkcije se na još jednom mestu koristi tip podataka. Neophodno je pogledati istaknuti deo na sledećoj slici.

```
function greetText(name: string): string {  
    return "Hello " + name;  
}
```

Slika 2.1.1 Povratni tip funkcije

Na slici, crvenom bojom je istaknut *povratni tip podataka funkcije*, odnosno, tip podataka kojem pripada rezultat izvršavanja kreirane metode. Ova funkcionalnost je, takođe, veoma korisna jer ukoliko, nekim slučajem, funkcija vrati rezultat koji ne odgovara tipu podatak `string`, kompajler će, ponovo, ukazati na grešku. Posebna korist od navedenog se ogleda u činjenici da će programer, koji koristi u svom radu navedenu funkciju, precizno znati koju će tip vrednosti dobiti kada iskoristi funkciju.

## GREŠKE U RADU SA TIPOVIMA PODATAKA

*Ukoliko se koristi neodgovarajući tip podataka, kompajler će javiti grešku.*

U prethodnom izlaganju je jasno istaknuto da **ukoliko se koristi neodgovarajući tip podataka, u TypeScript kodu, za argumente ili povratne tipove funkcija, kompajler će javiti grešku.** U nastavku, neophodno je obaviti demonstraciju šta se dešava ukoliko se u kodu javi nepodudaranje tipova podataka koje je navedeno i opisano. Posmatra se sledeći izolovani kod metode:

```
function greetText(name: string): string {  
    return 12;  
}
```

Pristupa se prevođenju programa, međutim, vrlo brzo će kompajler javiti izveštaj sa greškom u obliku prikazanom na sledećoj slici.

```
$ tsc compile-error.ts  
compile-error.ts(2,12): error TS2322: Type 'number' is not assignable to type 'string'.
```

Slika 2.1.2 Nepodudaranje tipov - greška u prevođenju

Veoma je važno u potpunosti sagledati šta je se, upravo, ovde desilo. U metodi je pokušao poziv rezultata 12, a jasno je naglašeno u kodu da je povratni tip metode *string*. Kompajler nije dozvolio prevođenje ovakvog koda.

Sa ciljem ispravke navedene greške, moguće je postupiti na sledeći način:

```
function greetText(name: string): number {  
    return 12;  
}
```

Povratni tip metode, kao što je moguće primetiti, promenjen je u numerički tip podataka *number*. Ako se sada pristupi prevođenju programa moguće je primetiti da je sada sve u redu i da je program očišćen od prethodno uočene greške.

Iz prethodnog izlaganja je jasno moguće sagledati u kolikoj meri primena tipova podataka može olakšati rukovanje greškama u TypeScript kodu.

## ▼ 2.1 Ugrađeni tipovi podataka

### PRIMENA UGRAĐENIH TIPOVA PODATAKA

*TypeScript daje podršku za nekoliko standardnih tipova podataka.*

TypeScript daje podršku za nekoliko standardnih tipova podataka koji imaju veliku primenu u Angular aplikacijama. To su:

- *string*;
- *number*;
- *boolean*;
- *Array*
- *enum*;
- *any*;
- *void*.

Sledi kratko predstavljanje ovih veoma često korišćenih tipova podataka.



## STRING I NUMERIČKI TIPOVI PODATAKA

*TypeScript koristi string i number tipove podataka.*

TypeScript koristi *string* i *number* tipove podataka za rad sa nizovima karaktera i brojevima.

Tip *string* omogućava deklarisanje i čuvanje tekstualnih podataka na način prikazan sledećim kodom:

```
var fullName: string = 'Vladimir Milicevic';
```

U TypeScript - u ne postoje posebni tipovi podataka za cele i realne brojeve. Koristi se jedan opšti tip označen ključnom rečju *number*. Sledećim kodom je prikazan način na koji je moguće deklarirati numeričku promenljivu primenom TypeScript jezika:

```
var age: number = 45;
```

## LOGIČKI TIP PODATAKA I NIZOVI

*TypeScript koristi boolean i Array tipove podataka.*

TypeScript koristi *boolean* i *Array* ključne reči za označavanje logičkog tipa podataka i nizova, respektivno.

Kao i u ostalim jezicima, programskim ili skripting, logička vrednost, deklarirana tipom podataka *boolean*, može imati vrednost *true* ili *false*. Na sledeći način je moguće ilustrirati deklarisanje logičke promenljive primenom *TypeScript* jezika:

```
var married: boolean = true;
```

Nizovi u *TypeScript* jeziku se deklariraju primenom tipa podataka *Array*. Međutim, budući da je *Array* kolekcija, takođe je neophodno odrediti i tip podataka kojem pripadaju elementi kolekcije. Imajući u vidu navedeno, tip podataka kojem pripadaju elementi kolekcije moguće je specificirati na jedna od sledeća dva načina:

- *Array<tip>*
- *tip[]*

Navedeno je moguće ilustrirati sledećim listinzima:

```
var jobs: Array<string> = ['IBM', 'Microsoft', 'Google'];  
var jobs: string[] = ['Apple', 'Dell', 'HP'];
```

Analogno, za numerički tip podataka moguće je kreirati nizove na sledeći način:

```
var chickens: Array<number> = [1, 2, 3];  
var chickens: number[] = [4, 5, 6];
```

## NABROJIVI TIP PODATAKA

*Nabrojivi tip podataka zasniva se na dodeli naziva numeričkim vrednostima.*

Nabrojivi tip podataka ili enumeracija zasniva se na dodeli naziva numeričkim vrednostima. Ovaj tip podataka je označen rezervisanom rečju `enum`. Na primer, ukoliko je cilj da se u programu kreira fiksna lista uloga, koju može da ima neka obaveza, moguće je postupiti na sledeći način:

```
enum Role {Employee, Manager, Admin};  
var role: Role = Role.Employee;
```

Podrazumevana početnavrednost za ovaj tip podataka je 0. Međutim, ovo može biti promenjeno tako što početna vrednost može biti eksplicitno zadatka kao što je to urađeno u sledećem listingu:

```
enum Role {Employee = 3, Manager, Admin};  
var role: Role = Role.Employee;
```

U prethodnom kodu urađeno je sledeće:

1. umesto da uloga Employee započinje vrednošću 0, počinje sa 3;
2. vrednost `enum` liste se uvećava za 1, za svaku narednu stavku, ali počinje sa 3;
3. to znači da su ulogama Manager i Admin pridružene vrednosti 4 i 5, tim redom.

Moguće je modifikovati ove vrednosti i drugačije. Moguće je svakoj ulozi eksplicitno dodeliti drugačiju numeričku vrednost. To je moguće ilustrovati sledećim listingom:

```
enum Role {Employee = 3, Manager = 5, Admin = 7};  
var role: Role = Role.Employee;
```

Prikazivanje naziva, pridruženih vrednostima `enum` tipa, je takođe veoma jednostavno. U tom slučaju je moguće, na najjednostavniji način, kreirati kod priložen sledećim listingom:

```
enum Role {Employee, Manager, Admin};  
console.log('Roles: ', Role[0], ', ', Role[1], 'and', Role[2]);
```

## TIPOVI PODATAKA ANY I VOID

*Tip podataka any označava univerzalni tip podataka. Tip void obeležava metodu bez povratne vrednosti.*

Tip podataka `any`, u TypeScript jeziku, označava univerzalni i podrazumevani tip podataka. Promenljiva obeležena ovim tipom podataka može da primi bilo koju vrednost. Navedeno je moguće ilustrovati sledećim listingom:

```
var something: any = 'as string';  
something = 1;  
something = [1, 2, 3];
```

Tip podataka *void* ukazuje da se ne očekuje nijedan od navedenih tipova podataka. U praksi, tipom *void* je deklarirana metoda bez povratne vrednosti. Navedeno je moguće ilustrovati sledećim listingom:

```
function setName(name: string): void {  
    this.fullName = name;  
}
```

## ▼ Poglavlje 3

# Klase

## PRIMENA KLASA U JAVASCRIPT BAZIRANIM JEZICIMA

*Pojavom generacije JavaScript ES6 klase su postale sastavni deo JavaScript jezika.*

Ako se uzme u obzir istorijat razvoja JavaScript jezika, prva pojava objektno - orijentisanog programiranja bazirala se na objektima baziranim na konceptu prototipa (prototype-based objects). Ovo je bilo karakteristično za generaciju JavaScript ES5 koja nije u kodiranju koristila koncept klasične klase, već se oslanjala na prototipove.

Kako je vreme odmicalo, brojna dobra iskustva su usvojene od strane JavaScript zajednice sa ciljem kompenzovanja nedostatka klasa u kodiranju. Dobri rezimei primene navedenih dobrih praksi mogu biti sagledani izučavanjem sadržaja koji su ponuđeni na sledećim linkovima:

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>;
- <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects>.

Konačno, **pojavom generacije JavaScript ES6 klase su postale sastavni deo JavaScript jezika.**

U *JavaScript* jeziku, klasa se kreira na način prikazan sledećim listingom:

```
class NazivKlase {  
    // sadržaj klase  
}
```

Klasa je izgrađena od sledećih komponentata:

- *osobine* (*properties*);
- *metode* (ili *funkcije* kako se u JavaScript žargonu obično nazivaju);
- *konstruktori*.

## OSOBINE KLASA

*Osobina definiše podataka koji će biti pridružen kreiranom objektu klase.*

Osobina definiše podataka koji će biti pridružen kreiranom objektu klase. Na primer, klasa pod nazivom *Person* može imati osobine:

- *first\_name*;

- `last_name;`
- `age;`

Svaka od navedenih osobina može, opcionalno, biti deklarirana primenom određenog tipa podataka. Na primer, osobine `first_name` i `last_name` mogu biti deklarirane primenom tipa podataka `string`, a osobina `age` može biti deklarirana primenom tipa podataka `number`. Ako se navedeno pretoči u programski kod, on može biti ilustrovan sledećim listingom:

```
class Person {  
  first_name: string;  
  last_name: string;  
  age: number;  
}
```

## METODE KLASA

*Metode ili funkcije predstavljaju potprograme koji se izvršavaju u kontekstu objekta klase.*

Metode ili funkcije predstavljaju potprograme koji se izvršavaju u kontekstu objekta klase. Da bi metoda bila izvršena, neophodno je prvo kreirati objekat date klase koji će, potom, pozvati posmatranu metodu i omogućiti njeno izvršavanje.

Ako se posmatra klasa, kreirana u prethodnoj sekciji, i ukoliko je cilj kreiranje metode koja će omogućiti prikazivanje poruke za objekat klase `Person`, moguće je kod klase proširiti na sledeći način:

```
class Person {  
  first_name: string;  
  last_name: string;  
  age: number;  
  
  // metoda  
  greet() {  
    console.log("Hello", this.first_name);  
  }  
}
```

Iz priloženog listinga moguće je sagledati nekoliko stvari:

- pristup individualnoj osobini `first_name`, za objekat koji poziva metodu, omogućen je primenom ključne reči `this`;
- nije deklarisan povratni tip metode.

Kada metoda ne poseduje jasno naveden povratni tip i vrednost, podrazumeva se da vraća bilo šta (određeno tipom podataka `any`). Međutim, u ovom konkretnom slučaju, metoda ne poseduje naredbu `return` pa je sasvim jasno da će njen povratni tip odgovarati tipu `void`.

Ukoliko je u programskom kodu neophodno obaviti poziv kreirane metode, to je moguće učiniti na sledeći način:

```
// deklarisanje promenljive tipa Person
var p: Person;

// poziv konstruktora - kreiranje objekta tipa Person
p = new Person();

// dodela vrednosti osobini klase
p.first_name = 'Vladimir';

// poziv metode kreiranim objektom klase
p.greet();
```

Kreiranje objekta klase je moguće obaviti i u jednoj liniji koda, na sledeći način:

```
var p: Person = new Person();
```

## METODE KOJE VRAĆAJU VREDNOST

*Ukoliko metoda poseduje naredbu `return`, mora da vrati vrednost odgovarajućeg tipa podataka.*

Ukoliko metoda poseduje naredbu `return`, mora da vrati vrednost odgovarajućeg tipa podataka. Tip podataka se jasno navodi prilikom deklarisanja metode.

Da bi navedeno bilo ilustrovano, neophodno je kreirati novu metodu klase `Person` koja će vratiti broj koji odgovara starosti osobe, koju predstavlja konkretan objekat ove klase, nakon perioda određenog zadatim brojem godina. Upravo navedeno je realizovano kodom iz sledećeg listinga:

```
class Person {
  first_name: string;
  last_name: string;
  age: number;

  greet() {
    console.log("Hello", this.first_name);
  }

  ageInYears(years: number): number {
    return this.age + years;
  }
}
```

Poziv ove metode je moguće, na veoma jednostavan način, ilustrovati sledećim listingom:

```
// kreiranje objekta klase Person
var p: Person = new Person();

// podešavanje početne vrednosti za osobinu age
```

```
p.age = 6;  
  
// starost za 12 godina  
p.ageInYears(12);  
  
// rezultat -> 18
```

## KONSTRUKTORI KLASA

*Konstruktor je posebna metoda koju je neophodno izvršiti da bi bila kreirana instanca klase.*

Konstruktor je posebna metoda koju je neophodno izvršiti da bi bila kreirana instanca klase. Obično, konstruktorom se obavljaju inicijalna podešavanja objekata.

Po podrazumevanim podešavanjima, metoda konstruktora mora da nosi naziv *constructor*. Ove metode, opcionalno, mogu da poseduju parametre ali ne mogu da vraćaju vrednosti budući da se pozivaju klada se kreira objekat klase.

Sa ciljem kreiranja objekta klase neophodno je pozvati konstruktor metodu primenom naziva klase, kao na sledeći način:

```
new ClassName()
```

Ukoliko klasa ne poseduje eksplicitno deklarisan konstruktor, on će automatski biti kreiran. To praktično znači da je kod:

```
class Vehicle {  
}  
  
...  
var v = new Vehicle();  
...
```

ekvivalentan kodu:

```
class Vehicle {  
  constructor() {  
  }  
}  
  
...  
var v = new Vehicle();  
...
```

**U TypeScript jeziku može da postoji samo jedan konstruktor po klasi.** Ovo predstavlja odstupanje od standardnog JavaScript jezika gde je moguće preklapanje konstruktora sve dok imaju različite argumente.

## KONSTRUKTORI SA PARAMETRIMA

*Konstruktor može da poseduje i parametre koji se koriste prilikom kreiranja objekata.*

Konstruktor može biti podrazumevani, kao u prethodnom listingu, a **može da poseduje i parametre koji se koriste prilikom kreiranja objekata**. Na primer, klasa `Person` poseduje konstruktor kojim je moguće inicijalizovati podatke:

```
class Person {
  first_name: string;
  last_name: string;
  age: number;

  constructor(first_name: string, last_name: string, age: number) {
    this.first_name = first_name;
    this.last_name = last_name;
    this.age = age;
  }

  greet() {
    console.log("Hello", this.first_name);
  }

  ageInYears(years: number): number {
    return this.age + years;
  }
}
```

Na ovaj način je kreiranje objekata dodatno pojednostavljeno, u odnosu na prethodni slučaj i može biti realizovano na sledeći način:

```
var p: Person = new Person('Vladimir', 'Milicevic', 45);
p.greet();
```

Na ovaj način su osobine objekta automatski podešene prilikom njegovog kreiranja.

## NASLEĐIVANJE

*Važan aspekt objektno - orijentisanog programiranja predstavlja nasleđivanje.*

Kao što je dobro poznato, veoma važan aspekt objektno - orijentisanog programiranja predstavlja nasleđivanje. Nasleđivanje predstavlja mehanizam preko kojeg neka klasa (potomak) prima ponašanje druge (roditeljske) klase. Na ovaj način je moguće zameniti, modifikovati ili proširiti ponašanje nove klase.



Jezik TypeScript u potpunosti podržava koncept nasleđivanja klasa koji je ugrađen u jezgro jezika. Nasleđivanje se postiže primenom ključne reči *extends*.

Za ilustrovanje navedenog, kreira se klasa pod nazivom *Report*.

```
class Report {  
  data: Array<string>;  
  
  constructor(data: Array<string>) {  
    this.data = data;  
  }  
  
  run() {  
    this.data.forEach(function(line) { console.log(line); });  
  }  
}
```

Iz listinga je moguće primetiti da klasa poseduje osobinu *data* koja odgovara tipu podataka *Array<string>*. Pozivom metode *run()* izvršava se petlja nad svim elementima niza i vrši se njihovo štampanje u konzoli.

Kreiranje objekta i poziv metode *run()* moguće je postići sledećim listingom:

```
var r: Report = new Report(['First line', 'Second line']);  
r.run();
```

Rezultat izvršavanja ovog koda je sledeći:

```
First line  
Second line
```

Sada je moguće tražiti postojanje još jednog izveštaja koji koristi zaglavlje i neke podatke, ali je i dalje bitno korišćenje prikazivanje podataka definisano prethodnom klasom. Nasleđivanje ovakvog ponašanja klase *Report* moguće je postići kreiranjem nove klase, na sledeći način:

```
class TabbedReport extends Report {  
  headers: Array<string>;  
  
  constructor(headers: string[], values: string[]) {  
    super(values)  
    this.headers = headers;  
  }  
  
  run() {  
    console.log(this.headers);  
    super.run();  
  }  
}
```

Primena klase naslednice je opisana sledećim listingom:

```
var headers: string[] = ['Name'];  
var data: string[] = ['Alice Green', 'Paul Pfifer', 'Louis Blakenship'];  
var r: TabbedReport = new TabbedReport(headers, data)  
r.run();
```

## ▼ Poglavlje 4

# Programski alati

## VRSTE TYPESCRIPT PROGRAMSKIH ALATA

*TypeScript obezbeđuje brojne sintaksne alate za olakšavanje programiranja.*

JavaScript ES6 i proširenje TypeScript obezbeđuju brojne sintaksne alate za olakšavanje programiranja. Dva najvažnija alata su:

1. funkcijska sintaksa "debele linije" (*fat arrow function syntax*);
2. šabloni stringova.

U sledećem izlaganju sledi objašnjenje primene navedenih alata.

## FAT ARROW NOTACIJA

*Notacija "Fat Arrow" omogućava pojednostavljeno pisanje koda funkcija.*

Notacija "*Fat Arrow*" omogućava pojednostavljeno pisanje koda funkcija. Ovo pojednostavljenje je moguće uporediti sa primenom "lambda" izraza u Java jeziku.

Od generacije JavaScript ES5, kada god je neophodno koristiti funkciju kao argument, neophodno je upotrebiti ključnu reč *function* zajedno sa zagradama na sledeći način:

```
//JavaScript ES5 primer
var data = ['Vladimir Milicevic', 'Marko Rajevic', 'Nikola Dimitrijevic'];
data.forEach(function(line) { console.log(line); });
```

Primenom "*Fat Arrow*" notacije, ovaj kod je moguće lakše zapisati na sledeći način:

```
// Typescript primer
2 var data: string[] = ['Vladimir Milicevic', 'Marko Rajevic', 'Nikola
Dimitrijevic'];
3 data.forEach( (line) => console.log(line) );
```

Veoma važna stavka *=>* sintakse da koristi isti *this* objekat kao i kod koji je okružuje. Ovo je veoma važno i razlikuje se od uobičajenog rada sa funkcijama u JavaScript - u. Kada se piše funkcija u JavaScript - u, njoj je dat vlastiti *this*. Ponekad, u JavaScript - u moguće je sresti sledeći kod,

```
var nate = {
  name: "Vlada",
  guitars: ["Gibson", "Martin", "Taylor"],
  printGuitars: function() {
    var self = this;
    this.guitars.forEach(function(g) {
      // this.name je nedefinisano pa se koristi self.name
      console.log(self.name + " plays a " + g);
    });
  }
};
```

Pošto sintaksa "fat arrow" deli *this* sa okružujućim kodom, moguće je napisati:

```
var nate = {
  name: "Vlada",
  guitars: ["Gibson", "Martin", "Taylor"],
  printGuitars: function() {
    this.guitars.forEach( (g) => {
      console.log(this.name + " plays a " + g);
    });
  }
};
```

## ŠABLONI STRINGOVA

*JavaScript ES6 uvodi nove šablone za rukovanje stringovima.*

*JavaScript ES6* uvodi nove šablone za rukovanje stringovima. Dva najznačajnija alata ovog tipa su:

- *varijable unutar stringova* (bez potrebe za povezivanjem putem operatora +);
- *višelinijski (multi-line) stringovi*.

Varijabla unutar stringa ili interpolacija stringova može da se ugradi na sledeći način:

```
var firstName = "Vladimir";
var lastName = "Milićević";

// interpolacija stringa
var greeting = `Hello ${firstName} ${lastName}`;

console.log(greeting);
```

Višelinijski stringovi su sledeći odličan alat za manipulaciju stringovima kada ih je potrebno dodati u kod. Evo primera:

```
var template = `
<div>
<h1>Hello</h1>
```

```
<p>This is a great website</p>
</div>
,

// uraditi nešto sa stringom `template`
```

## OSTALI TYPESCRIPT ALATI

### *Postoje brojni i korisni TypeScript / ES6 alati*

U ovom delu lekcije je neophodno je navesti još neke korisne *TypeScript / ES6* alate o kojima će biti diskusije, detaljno, kako analiza i razumevanje Angular problematike bude odmicalo dalje:

- *interfejsi;*
- *generički tipovi podataka;*
- *importovanje i eksportovanje modula;*
- *dekoratori;*
- *destrukcija.*

## ▼ Poglavlje 5

# Funkcionisanje Angular okvira

## UVODNO RAZMATRANJE

*Aakcenat će biti na Angular konceptima visokog nivoa i njihovom funkcionisanju kao celine.*

U ovom delu lekcije, akcenat će biti na Angular konceptima visokog nivoa (high - level). Za analizu i diskusiju biće neophodno vratiti se jedna korak unazad sa ciljem sagledavanja mehanizama funkcionisanja navedenih delova u formi celine. Za svaki od koncepata biće dat detaljan pregled i objašnjenje njegove funkcionalnosti.

Prvi značajan koncept predstavlja komponenta. Kao što je već rečeno, primenom komponenata Angular uči veb pregledač da koristi novi HTML tag. Ako je neko od studenata pokušao da samostalno uči *AngularJS 1.x*, savremeni koncept komponenata odgovara starom konceptu direktiva. Ono što je bitno, za ovaj deo razmatranja, *Angular* komponente poseduju značajne prednosti u odnosu na starije direktive o čemu će, takođe, biti detaljno govora kada u fokus dođu ovi koncepti.

U daljem izlaganju, biće započeta diskusija od samog vrha - koncepta poznatog pod nazivom aplikacija.

## ▼ 5.1 Aplikacija

### STABLO KOMPONENATA

*Angular aplikacija ne predstavlja ništa drugo do stablo komponenata.*

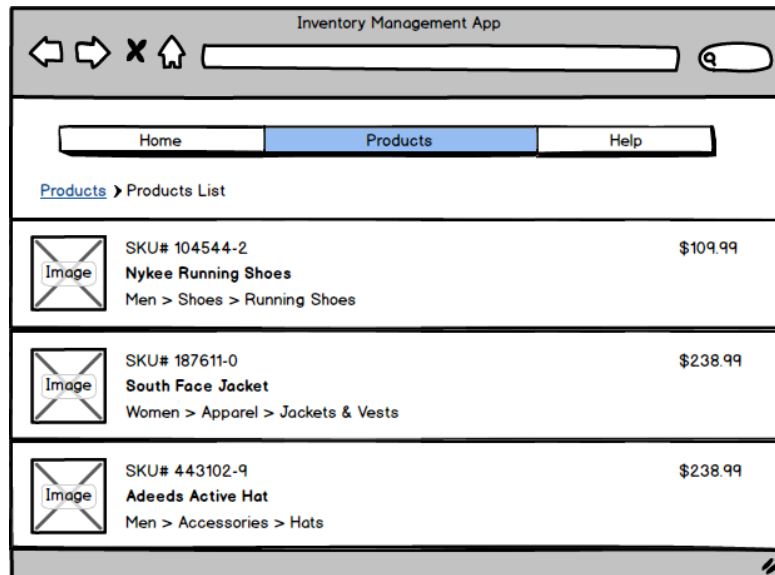
Angular aplikacija ne predstavlja ništa drugo do stablo komponenata. U korenu stabla, kao komponenta najvišeg nivoa, javlja se upravo aplikacija. Upravo njom je određen sadržaj koji će veb pregledač prikazati kada pokreće aplikaciju.

Odlična stvar, vezana za komponente, jeste neograničena mogućnost njihovog povezivanja. To znači da je kombinovanjem sitnijih komponenata moguće kreirati složene komponente. Jednostavno rečeno, aplikacija je komponenta koja kreira druge komponente.

Budući da su sve komponente strukturirane u stablu roditelj / potomak, kada se neka komponenta kreira, ona rekurzivno kreira i vlastite potomke.

Da bi diskusija tekla jednostavnije i razumljivije, moguće je osloniti se na adekvatan primer i neka to bude aplikacija za upravljanje zalikama.

Sledećom slikom je moguće opisati idejni izgled same aplikacije.



Slika 5.1.1 Idejni izgled prateće Angular aplikacije

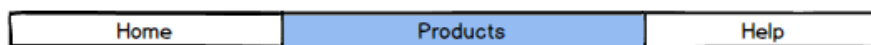
Na osnovu priložene slike, ideja može biti da se za potrebe realizacija aplikacije razviju tri posebne komponente:

- *navigaciona komponenta;*
- *komponenta prikaz detalja;*
- *komponenta za kreiranje i prikaz liste proizvoda.*

## NAVIGACIONA KOMPONENTA

*Zadatak navigacione komponente jeste kreiranje sekcije za navigaciju.*

Zadatak navigacione komponente jeste kreiranje sekcije za navigaciju. Da bi bilo jasnije, iz prethodne slike je potrebno izolovati baš ovaj deo i to je prikazano sledećom slikom.



Slika 5.1.2 Navigaciona komponenta aplikacije

## KOMPONENTA ZA PRIKAZ DETALJA

*Komponenta za izbor detalja smeštena je neposredno ispod sekcije za navigaciju.*

Komponenta za prikaz detalja, ako se pogleda slika 1, smeštena je neposredno ispod sekcije za navigaciju. Ova komponenta kreira prikaz hijerarhijske reprezentacije mesta u aplikaciji

u kojem se korisnik trenutno nalazi. Ovakve komponente se često u literaturi nazivaju simbolično "mrvicama hleba" (*Breadcrumbs*) pa će ta praksa biti zadržana i u aktuelnom primeru.

Sledećom slikom je izolovana komponenta za prikaz detalja.




## Products > Products List

Slika 5.1.3 Komponenta za prikaz detalja

### KOMPONENTA ZA KREIRANJE I PRIKAZ LISTE PROIZVODA

*Komponenta kojom je omogućeno reprezentovanje kolekcije dostupnih proizvoda.*

Komponenta za kreiranje i prikaz liste proizvoda predstavlja komponentu kojom je omogućeno reprezentovanje kolekcije dostupnih proizvoda. Sledećom slikom je ilustrovana navedena komponenta.

	SKU# 104544-2 <b>Nykee Running Shoes</b> Men > Shoes > Running Shoes	\$109.99
	SKU# 187611-0 <b>South Face Jacket</b> Women > Apparel > Jackets & Vests	\$238.99
	SKU# 443102-9 <b>Adeeds Active Hat</b> Men > Accessories > Hats	\$238.99

Slika 5.1.4 Komponenta za kreiranje i prikaz liste proizvoda

Razlaganjem posmatrane komponente na sledeći nivo jednostavnijih komponenata, moguće je istaći da će lista proizvoda biti dobijena kombinovanjem jednog ili više redova koji odgovaraju pojedinačnim proizvodima.

Dekompozicija može ići i dalje pa je moguće reći da svaki red, koji odgovara pojedinačnim proizvodima, čine:

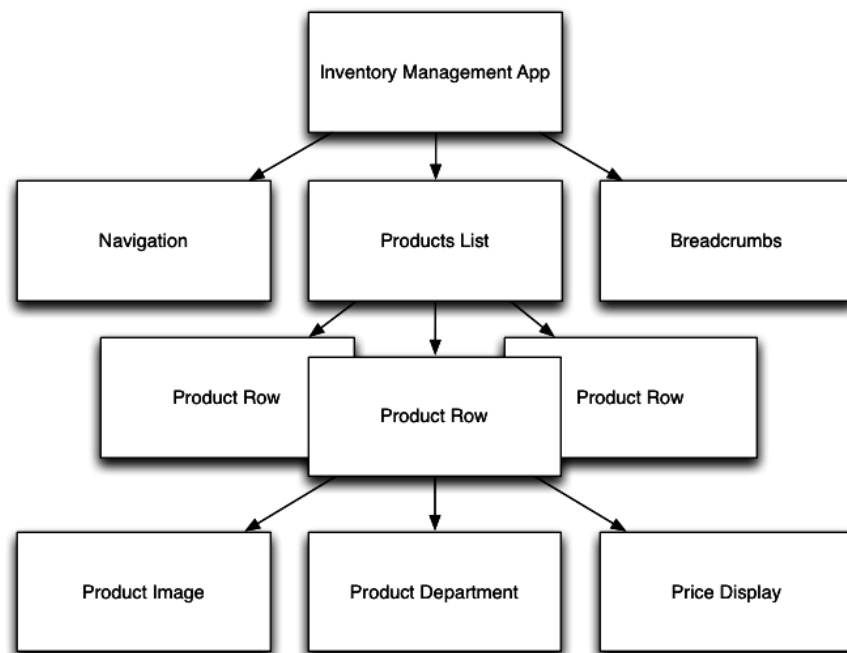
- *slika proizvoda* - posebna komponenta biće angažovana za vizualni prikaz proizvoda iz liste;
- *odeljenje proizvoda* - komponenta će da kreira stablo odeljenja, na primer: *Men > Shoes > Running Shoes*;
- *prikaz cene* - posebna komponenta biće angažovana za prikaz cene proizvoda.



## OBJEDINJENI PRIKAZ KOMPONENATA APLIKACIJE

*Povezivanjem navedenih komponenata aplikacije u celinu, dobija se stablo aplikacije.*

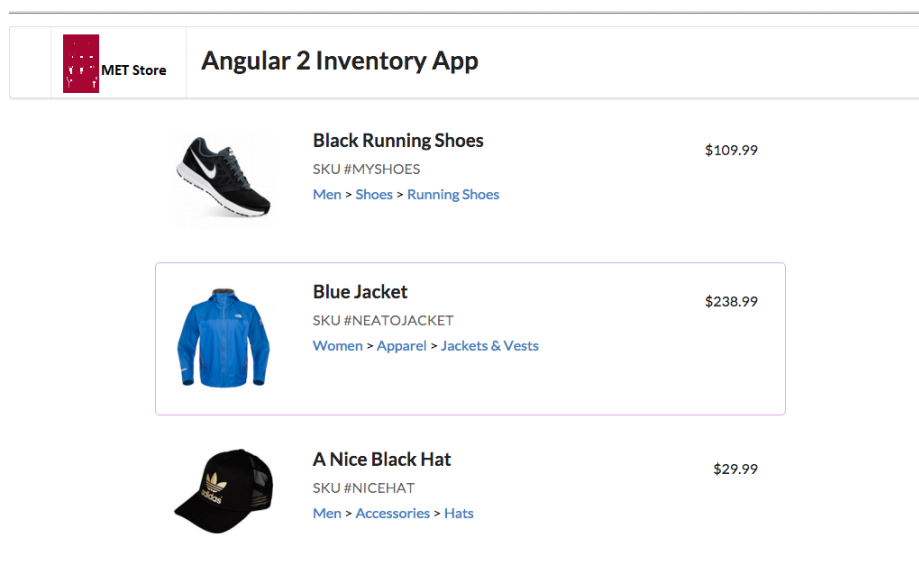
Povezivanjem navedenih komponenata aplikacije u celinu, dobija se stablo aplikacije. U konkretnom slučaju, stablo aplikacije može biti ilustrovano sledećom slikom:



Slika 5.1.5 Stablo Angular aplikacije

Na vrhu se nalazi *Inventory Management App* što upravo predstavlja aplikaciju koja će biti kreirana. Ispod se nalaze komponente: *Navigation*, *Breadcrumb* i *Products List*. Komponenta *Products List* sadrži *Product Row* za svaki od proizvoda. Svaki red *Product Row* koristi tri komponente za prikazivanje: slike, odeljenja i cene proizvoda.

Upravo na ovim postavkama biće kreirana nova Angular aplikacija, kao na sledećoj slici.



Slika 5.1.6 Budući izgled aplikacije

## VAŽNE SMERNICE

*Slede važne napomene u vezi sa izradom aplikacije.*

Slede važne napomene u vezi sa izradom aplikacije:

- koristi se *Angular CLI*;
- kreiranje aplikacije: *ng new NazivAplikacije*;
- kreiranje komponentata: *ng generate component NazivKomponente*;
- pokretanje aplikacije: *ng serve*;

Posebnu pažnju bi trebalo obratiti na:

- Kako se aplikacija razbija na komponente?
- Kako se komponente višestruko koriste primenom ulaza (inputs)?
- Kako se rukuje korisničkim interakcijama?

## ▼ 5.2 Modeli u aplikaciji

### MODEL PROIZVODA

*Angular je veoma fleksibilan okvir i podržava različite tipove modela i arhitekture podataka.*

Jedna od ključnih stvari koje je neophodno razumeti u vezi sa okvirom *Angular* da on ne propisuje primenu neke specijalne biblioteke modela. *Angular je veoma fleksibilan okvir i*

podržava različite tipove modela i arhitekture podataka. To praktično znači, da je programeru ostavljeno na volju da odluči kako će da implementira navedene koncepte.

Što se tiče arhitekture podataka u narednim izlaganjima će se ukazati potreba da se veoma detaljno diskutuje na tu temu. Za sada, akcenat će biti na implementaciji modela u formi čistih JavaScript objekata.

U podfolderu, kreiranog projekta, *src/app/* kreira se datoteka *product.model.ts* koja će čuvati kod modelske klase Product. Sledećim listingom je priložen kod ove klase:

```
/**
 * Obezbeđuje objekte tipa `Product`
 */
export class Product {
  constructor(
    public sku: string,
    public name: string,
    public imageUrl: string,
    public department: string[],
    public price: number) {
  }
}
```

Iz priloženog listinga je moguće primetiti da klasa *Product* sadrži konstruktor sa 5 argumenata. Klasa takođe ne sadrži nikakve Angular zavisnosti, ona je osmišljena kao modelska klasa koja će biti dalje korišćena u aplikaciji.

## ▼ 5.3 Komponente u Angular aplikaciji

### KOMPONENTE

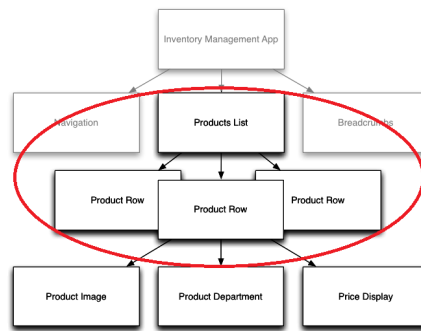
*Komponenta je osnovni gradivni blok Angular aplikacije.*

Kao što je već napomenuto, komponenta je osnovni gradivni blok Angular aplikacije. Aplikacija, sama po sebi, je komponenta najvišeg nivoa. U daljem razvoju aplikacija se razbija na manje komponente - *potomke*.

Svaka komponenta je izgrađena iz tri dela:

- dekorator komponente;
- pogled;
- kontroler.

Za ilustraciju ključnih koncepata u Angularu, nepogodno je dobro razumeti komponente. Izlaganje započinje sa najvišeg nivoa aplikacije *InventoryManagementApp*, a zatim će fokus biti na komponenti *ProductListComponent* i komponentama potomcima (sledeća slika).



Slika 5.2.1 ProductListComponent i komponente potomci

Za početak, prilaže se listing centralne komponente *AppComponent* koja za sada izgleda ovako:

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

@Component({
  selector: 'inventory-app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  //ovde dolazi logika
}
```

Anotacija *@Component* se naziva dekoratorom i preko nje se dodaju meta-podaci u klasu. Dekorator specificira:

- selektor, koji govori Angularu koji element da traži;
- šablon (*template*, *templateUrl*), kojim se definiše pogled (HTML izlaz).

Kontroler komponente je definisan klasom, *AppComponent* u konkretnom slučaju.

## SELEKTOR I ŠABLON

*Selektor ukazuje se na komponentu koja će biti prepoznata kada se bude koristila u šablonu.*

Primenom ključa *selector* ukazuje se na komponentu koja će biti prepoznata kada se bude koristila u šablonu. Iz listinga, prethodno prikazane klase *AppComponent*, primećuje se da je selektorom definisan tag *inventory-app-root* kojim je komponenta predstavljena u odgovarajućem pogledu. Postoje dva načina da se ovaj tag angažuje u HTML kodu:

```
<inventory-app-root></inventory-app-root>
```

Ili na drugi način:

```
<div inventory-app-root></div>
```

Šablon je vidljivi deo komponente. Primenom ključa `template`, unutar dekoratora `@Component`, deklarira se HTML šablon kojeg će komponenta koristiti:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  //ovde dolazi logika
}
```

Slika 5.2.2 Definisanje šablona

## DODAVANJE PROIZVODA

*Da bi proizvod bio dodat, neophodno je prvo importovati njegovu klasu u centralnu komponentu.*

Aplikacija nije preterano zanimljiva ako ne prikazuje proizvode, a sada je to slučaj. Biće iskorišćena centralna komponenta i u njoj će biti dodat jedan proizvod.

Da bi proizvod bio dodat, neophodno je prvo importovati njegovu klasu u centralnu komponentu, a onda u okviru njenog konstruktora obaviti dodavanje konkretnog objekta, primenom definisanog konstruktora sa 5 argumenata. Sledi listing:

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

//importovanje klase Product
import { Product } from './product.model';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  product: Product;

  constructor() {
    //kreiranje novog proizvoda
    let newProduct = new Product(
      'NICEHAT', //sku
      'A Nice Black Hat', //name
      '/assets/images/products/black-hat.jpg', //imageUrl
      ['Men', 'Accessories', 'Hats'], //department
    );
  }
}
```

```
29.99); //price
this.product = newProduct;
}
}
```

## PRIKAZIVANJE PROIZVODA PREKO POVEZIVANJA POGLEDA

*Koristi se varijabla sa ciljem kreiranja pogleda kojim će biti moguće prikazati kreirani proizvod.*

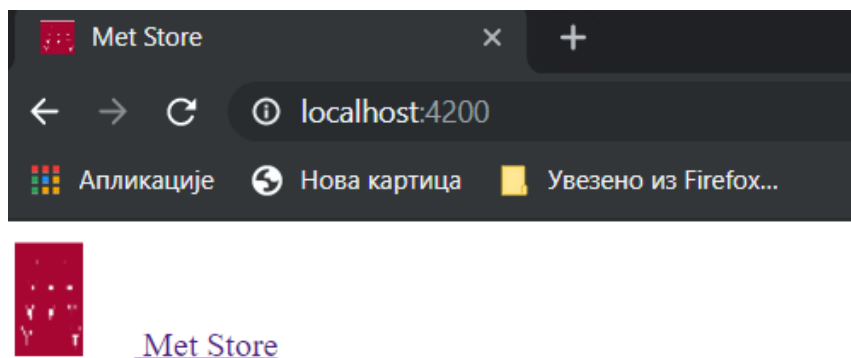
U prethodnom izlaganju je obavljeno dodavanje proizvoda u centralnu komponentu *AppComponent*. Sada je moguće ovu varijablu iskoristiti sa ciljem kreiranja pogleda kojim će biti moguće prikazati kreirani proizvod.

Neophodno je otvoriti datoteku *app.component.html* i u nju dodati sledeći kod:

```
<div class="inventory-app">
  <h1>{{ product.name }}</h1>
  <span>{{ product.sku }}</span>
</div>
```

Ukoliko je sve u redu, prevođenjem i pokretanjem ove aplikacije biće prikazan pogled koji prikazuje osobine: *sku* i *name*, kreiranog objekta proizvoda.

Trenutni izgled aplikacije je moguće prikazati sledećom slikom:



Slika 5.2.3 Prikazivanje proizvoda preko povezivanja pogleda

## DODAVANJE LISTE PROIZVODA

*Hajde sada da jedan proizvod bude zamenjen nizom proizvoda.*

Kod koji je do sad kreiran, odličan je osnov za dalji rad i unapređivanje. Hajde sada da jedan proizvod bude zamenjen nizom proizvoda. Prvi korak je zamena osobine `product: Product`, osobinom `products: Product[]`. Dalje, u konstruktoru centralne komponente biće kreirano još par proizvoda koji će potom svi zajedno biti dodati u kreirani niz `products`.

Navedene izmene je moguće ilustrovati sledećim listingom:

```
export class AppComponent {  
  //pojedinačni proizvod je zamenjen nizom proizvoda  
  products: Product[];  
  
  constructor() {  
  
    //kreiranje i dodavanje proizvoda u listu  
    this.products = [  
      new Product(  
        'MYSHOES',  
        'Black Running Shoes',  
        '/assets/images/products/black-shoes.jpg',  
        ['Men', 'Shoes', 'Running Shoes'],  
        109.99),  
      new Product(  
        'JACKET',  
        'Blue Jacket',  
        '/assets/images/products/blue-jacket.jpg',  
        ['Women', 'Apparel', 'Jackets & Vests'],  
        238.99),  
      new Product(  
        'NICEHAT',  
        'A Nice Black Hat',  
        '/assets/images/products/black-hat.jpg',  
        ['Men', 'Accessories', 'Hats'],  
        29.99)  
    ];  
  }  
}
```

## IZBOR PROIZVODA

*Cilj je dodavanje podrške interakciji korisnika sa aplikacijom.*

Moguće je dalje igrati se idejama i vršiti modifikacije na aplikaciji. Cilj je dodavanje podrške interakciji korisnika sa aplikacijom. Na primer, korisnik može da: izabere konkretan proizvod sa ciljem dobijanja dodatnih informacija u vezi sa proizvodom, doda proizvod u korpu i još mnogo toga.

Kada je istaknut nov zahteve aplikacije, mogu će je dodati nove funkcionalnosti u centralnu komponentu *AppComponent* za rukovanje događajem koji je rezultat korisnikovog izbora iz liste. Prethodni listing biće proširen metodom *productWasSelected()*:

```
productWasSelected(product: Product): void {  
    console.log('Product clicked: ', product);  
}
```

Iz listinga je moguće primetiti da funkcija uzima jedan argument tipa *Product* koji će biti prikazan u logu veb pregledača. Još malo pa će funkcija moći u punoj meri da se koristi.

## PRIKAZIVANJE PROIZVODA PRIMENOM TAGA PRODUCTS-LIST

*Za uvedenu novu funkcionalnost više nije dovoljna samo top - level komponenta.*

Za uvedenu novu funkcionalnost - prikazivanje liste dostupnih proizvoda, više nije dovoljna samo top - level komponenta pod nazivom *AppComponent*. Dakle, neophodno je kreiranje nove komponente *ProductsList* koja će biti zadužena da omogući prikazivanje liste proizvoda. Neka je selektorom nove komponente definisan tag *<products-list>* koji će omogućiti podudaranje sa implementacijom liste proizvoda.

Pre ulaska u implementacione detalje nove komponente, sledećim listingom prikazana je primena nove komponente u šablonu:

```
<div class="inventory-app">  
    <products-list  
        [productList]="products"  
        (onProductSelected)="productWasSelected($event)">  
    </products-list>  
</div>
```

Iz listinga je moguće sagledati prisustvo nove sintakse o kojoj sledi diskusija u narednom izlaganju.

## ULAZ / IZLAZ

*Primena ključnih Angular koncepata: ulaza (input) i izlaza (output).*

Iz prethodnog listinga nazire se **primena ključnih Angular koncepata: ulaza (input) i izlaza (output)**. Problematiku je bolje i lakše sagledati vizuelno sa sledeće slike:



```
<products-list
  [productList]="products"           <!-- input -->
  (onProductSelected)="productWasSelected($event)" > <!-- output -->
</products-list>
```

Slika 5.2.4 Primena kocepata ulaza i izlaza

Ako se osmotri priloženi kod sa slike moguće je primetiti dve stvari:

- uglastim zagradama [ ] prosleđuje se ulaz;
- malim zagradama ( ) je istaknuta sintaksa u kojoj se rukuje izlazom.

Podaci se kreću ka komponenti preko povezivanja ulaza , a događaji izlaze iz komponente preko povezivanja izlaza. Podešavanje povezivanja ulaza / izlaza može se rezonovati kao definisanje javnog (*public*) *API* - ja za posmatranu komponentu.

O radu sa ulaznim vrednostima je već bilo diskusije u prethodnoj lekciji. Sada je neophodno posvetiti više pažnje rukovanju izlaznim vrednostima.

U Angularu, slanje podataka iz komponente moguće je obaviti na sledeći način:

```
<products-list
  ...
  (onProductSelected)="productWasSelected($event)">
```

Na ovaj način se vrši osluškivanje *onProductSelected* izlaza iz komponente *ProductsList* . Kod funkcionise na sledeći način:

- (*onProductSelected*), na levoj strani izraza, je naziv izlaza koji se osluškuje;
- "*productWasSelected*", na desnoj strani izraza, je naziv metode koja se poziva kada je neka nova vrednost prosleđena izlazu;
- *\$event* označava specijalnu promenljivu koja reprezentuje veličinu koja se emituje na izlazu.

Za sada nije bilo detaljno govora o definisanju ulaza i izlaza za vlastite komponente ali će to ubrzo doći na red čim se pristupi kreiranju *ProductsList* komponente. Za sada naučeno je slanje podataka ka potomku i prijem podataka iz komponente potomka.

## ZAOKRUŽENA DEFINICIJA CENTRALNE KOMPONENTE

*Prilažu se puni listinzi klase i pogleda komponente AppComponent.*

Sada je moguće zaokružiti definiciju centralne komponente aplikacije. Sledi listing klase *AppComponent*:

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

@Component({
  selector: 'inventory-app-root',
```

```

    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
  })
  export class AppComponent {
    //pojedinačni proizvod je zamenjen nizom proizvoda
    products: Product[];

    constructor() {

      //kreiranje i dodavanje proizvoda u listu
      this.products = [
        new Product(
          'MYSHOES',
          'Black Running Shoes',
          '/assets/images/products/black-shoes.jpg',
          ['Men', 'Shoes', 'Running Shoes'],
          109.99),
        new Product(
          'JACKET',
          'Blue Jacket',
          '/assets/images/products/blue-jacket.jpg',
          ['Women', 'Apparel', 'Jackets & Vests'],
          238.99),
        new Product(
          'NICEHAT',
          'A Nice Black Hat',
          '/assets/images/products/black-hat.jpg',
          ['Men', 'Accessories', 'Hats'],
          29.99)
      ];
    }

    productWasSelected(product: Product): void {
      console.log('Product clicked: ', product);
    }
  }

```

```

<div class="inventory-app">

  <products-list

    [productList]="products"

    (onProductSelected)="productWasSelected($event)">

  </products-list>
</div>

```

Konačno, sledi i listing datoteke [app.compat.html](#):

```

<div class="inventory-app">

```

```
<products-list
  [productList]="products"
  (onProductSelected)="productWasSelected($event)">
</products-list>
</div>
```

## KOMPONENTA PRODUCTSLISTCOMPONENT

*Kreiranje komponente `ProductsListComponent` za kreiranje i prikazivanje redova liste proizvoda.*

Sada kada aplikacija poseduje komponentu najvišeg nivoa, moguće je pristupiti kreiranju komponente `ProductsListComponent` čiji je zadatak kreiranje i prikazivanje redova liste proizvoda.

Takođe, cilj je omogućavanje korisniku da izabere jedan proizvod i da se čuva trag u vezi sa izabranim proizvodom. Komponenta `ProductsListComponent` je idealno mesto za implementiranje navedenih funkcionalnosti zato što će "znati" sve `Product` objekte istovremeno.

Komponenta će biti kreirana prateći sledeća tri koraka:

1. podešavanje `ProductsListComponent @Component` opcija;
2. pisanje `ProductsListComponent` kontrolerske klase;
3. pisanje `ProductsListComponent` šablona pogleda.

## PODEŠAVANJE DEKORATORA PRODUCTSLISTCOMPONENT KOMPONENTE

*Definicija i analiza konfiguracije dekoratora `Component` komponente `ProductsListComponent`.*

Data je konfiguracija dekoratora `@Component` komponente `ProductsListComponent`.

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

import { Product } from '../product.model';

@Component({
```

```
selector: 'products-list',
templateUrl: './product-list.component.html',
styleUrls: ['./product-list.component.css']
})
export class ProductListComponent implements OnInit {

  @Input() productList: Product[];

  @Output() onProductSelected: EventEmitter<Product>;

  ...
}
```

Iz listinga je moguće jasno uočiti sledeće stvari:

- selektorom je određen tak komponente `<products-list>`;
- definisane su dve osobine: `productList` i `onProductSelected`;
- `productList` poseduje anotaciju `@Input()` koja ukazuje da se radi o ulaznoj vrednosti;
- `onProductSelected` poseduje anotaciju `@Output()` koja ukazuje da se radi o izlazu.

## ULAZI KOMPONENTE

*Veličina označena kao ulaz predstavlja osobinu komponente dekoratorom `Input()`.*

Ulazima je specificirano koje parametre komponenta može da primi. Da bi veličina bila označena kao ulaz, neophodno je obeležiti odgovarajuću osobinu komponente dekoratorom `@Input()`. Kada je cilj da se istakne da komponenta prihvata ulaz, klasa komponente mora da poseduje promenljivu u kojoj će ulazni podaci biti čuvani. Na primer, to je moguće ilustrovati sledećim kodom:

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

@Component({
  selector: 'my-component',
})
class MyComponent {
  @Input() name: string;
  @Input() age: number;
}
```

Slika 5.2.5 Varijabla za prijem ulaza

## PROSLEĐIVANJE PROIZVODA PREKO ULAZA

*Prosleđivanje proizvoda preko ulaza primenom taga šablona i izraza ulaza.*

U ovom delu izlaganja, dovoljno je prisetiti se da je u kodu prosleđena lista proizvoda za prikazivanje, u okviru taga `<products-list>` primenom izraza ulaza: `[productList]`. Navedeno je ilustrovano sledećom slikom.

```
<div class="inventory-app">
  <products-list
    [productList]="products"
    (onProductSelected)="productWasSelected($event)">
  </products-list>
</div>
```

Slika 5.2.6 Prosleđivanje proizvoda preko ulaza

Primena ovakve sintakse je veoma jednostavna, prosleđuje se vrednost `this.products ()` na komponentu `AppComponent` putem unosa sa komponente `ProductsListComponent`.

## IZLAZI KOMPONENATA

*Kada je neophodno prosleđivanje podataka komponenti koristi se koncept povezivanja izlaza.*

Kada je neophodno prosleđivanje podataka komponenti, iz njenog okruženja, koristi se koncept povezivanja izlaza. Na primer, komponenta koja se kreira poseduje dugme i nešto se dešava kada korisnik klikne da to dugme. Kao posledica ove akcije korisnika, vrši se povezivanje izlaza dugmeta sa metodom koja je deklarirana u odgovarajućem kontroleru. Za ovo se koristi notacija u opštem obliku:

```
(output)="action"
```

Sledećom slikom je prikazano prosleđivanje izlaza ka metodi kontrolera

```
<div class="inventory-app">
  <products-list
    [productList]="products"
    (onProductSelected)="productWasSelected($event)">
  </products-list>
</div>
```

Slika 5.2.7 Prosleđivanje izlaza ka metodi kontrolera

## EMITOVANJE KREIRANIH DOGAĐAJA

*Kreira se komponenta koja emituje kreirani događaj.*

Neka se sada razmatra sledeći scenario - kreira se komponenta koja emituje kreirani događaj, poput događaja: `click` ili `mousedown`. Za kreiranje izlaza neophodno je obaviti tri stvari:

1. specifikacija izlaza unutar `@Component` podešavanja;

2. povezivanje *EventEmitter* sa izlaznom osobinom;
3. emitovanje događaja putem *EventEmitter* u odgovarajućem trenutku.

Navedeno je u potpunosti ilustrovano sledećom slikom.

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

import { Product } from '../product.model';

@Component({
  selector: 'products-list',
  //1. podešavanje izlaza
  templateUrl: './product-list.component.html',
  styleUrls: ['./product-list.component.css']
})
export class ProductListComponent implements OnInit {

  @Input() productList: Product[];

  // 2. povezivanje EventEmitter sa izlaznom osobinom
  @Output() onProductSelected: EventEmitter<Product>;

  private currentProduct: Product;

  constructor() {
    //3. emitovanje događaja putem EventEmitter
    this.onProductSelected = new EventEmitter();
  }
}
```

Slika 5.2.8 Emitovanje kreiranih događaja

## KREIRANJE KONTROLERSKE KLASSE PRODUCTSLISTCOMPONENT

*Kontrolerska klasa **ProductsListComponent** dobija tri objektne varijable.*

Nastavlja se dalja analiza i kreiranje pratećeg primera. Kontrolerska klasa *ProductsListComponent* dobija tri objektne varijable:

- prva čuva listu proizvoda;
- druga predstavlja izlazni događaj;
- treća označava referencu na trenutno izabrani proizvod.

Kod koji uvažava navedene zahteve prikazan je sledećim listingom:

```
export class ProductListComponent implements OnInit {

  @Input() productList: Product[];

  // 2. povezivanje EventEmitter sa izlaznom osobinom
```

```
@Output() onProductSelected: EventEmitter<Product>;

private currentProduct: Product;

constructor() {
  //3. emitovanje događaja putem EventEmitter
  this.onProductSelected = new EventEmitter();
}

}
```

## PISANJE ŠABLONA PRODUCTSLISTCOMPONENT

*Konačna definicija komponente se dobija kodiranjem odgovarajućeg šablona.*

Konačna definicija komponente se dobija kodiranjem odgovarajućeg šablona. Sledećim listingom je moguće dati konačan listing šablona komponente [šablona ProductsListComponent](#). Listing odgovara sadržaju datoteke: [src/app/products-list/products-list.component.html](#).

```
<div class="ui items">
  <product-row
    *ngFor="let myProduct of productList"
    [product]="myProduct"
    (click)='clicked(myProduct)'
    [class.selected]="isSelected(myProduct)">
  </product-row>
</div>
```

Iz listinga je moguće primetiti da je neophodno obaviti kreiranje nove komponente koja će odgovarati primenjenom tagu `<product-row>` za prikazivanje svakog proizvoda, iz liste, u vlastitom redu.

Primenom petlje `ngFor` prolazi se kroz listu proizvoda i vrši generisanje lokalne promenljive `myProduct` za svaki proizvod iz liste.

Dalje, linija `[product]="myProduct"` ukazuje da bi trebalo proslediti kreiranu lokalnu varijablu `myProduct` kao ulaznu vrednost `product` unurat taga `<product-row>`.

U liniji `(click)='clicked(myProduct)'` koristi se ugrađeni događaj `click` koji poziva funkciju `clicked()` komponente `ProductsListComponent` svaki put kada korisnik obavi klik na odgovarajući element iz liste.

Linija koda `[class.selected]="isSelected(myProduct)"` bazira se na činjenici da Angular dozvoljava, primenom ove sintakse, podešavanje klasa za element uslovno. Konkretno, neophodno je dodati `CSS` klasu `selected` ukoliko poziv `isSelected(myProduct)` vrati vrednost `true`.

## PROŠIRENJE KONTROLERA NOVIM FUNKCIONALNOSTIMA

### *Proširenje kontrolera novim funkcionalnostima za servisiranje koda šablona.*

Iz prethodnog izlaganja je moguće zaključiti da funkcije `clicked()` i `isSelected()` još nisu definisani. To je moguće uraditi izmenom koda kontrolera `src/app/products-list/products-list.component.ts` na sledeći način:

```
clicked(product: Product): void {  
    this.currentProduct = product;  
    this.onProductSelected.emit(product);  
}  
  
isSelected(product: Product): boolean {  
    if (!product || !this.currentProduct) {  
        return false;  
    }  
    return product.sku === this.currentProduct.sku;  
}
```

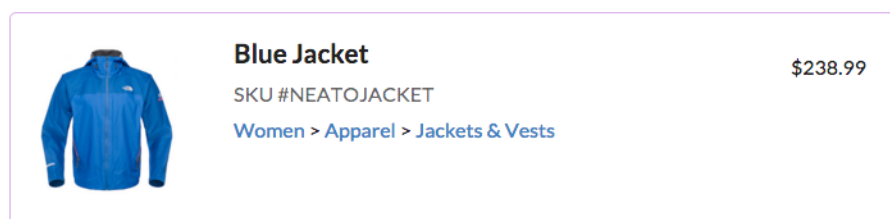
Kod govori sledeće:

- metoda `clicked()` - promenljivoj `this.currentProduct` je pridružen proizvod koji je prosleđen i vrši se slanje izabranog proizvoda na izlaz;
- metoda `isSelected()` - metoda prihvata proizvod i vraća vrednost `true` ukoliko osobina `product.sku` odgovara osobini `currentProduct.sku`. U suprotnom, vratiće `false`.

## KOMPONENTA PRODUCTROWCOMPONENT

### *Aplikaciji nedostaje komponenta za prikazivanje sadržaja određenog tagom `product-row`.*

Kao što je bilo moguće sagledati iz prethodnog izlaganja, aplikaciji nedostaje komponenta koja će omogućiti prikazivanje sadržaja određenog tagom `<product-row>`. Primer takvog sadržaja je moguće ilustrovati sledećom slikom:



Slika 5.2.9 Prikaz pojedinačnih proizvoda iz liste proizvoda

U razvojnem okruženju se kuca `Angular CLI` naredba:



```
ng generate component ProductRow
```

Ubrzo će biti kreirana željena komponenta i sada se postavljaju dva nova izazova:

- definisanje kontrolera komponente;
- definisanje pogleda (šablona) komponente.

Ako se obrati detaljnija pažnja na sliku 9 moguće je primetiti i postojanje: slike predmeta, informacija o odeljenju na kojem se predmet nalazi i cenu predmeta. Iz ovoga je moguće izvući zaključak da je ovu komponentu neophodno dodatno razložiti na sledeće tri komponente:

- *ProductImageComponent* - za prikazivanje slika;
- *ProductDepartmentComponent* - za prikazivanje informacija o odeljenju (detalji ili "mrvice hleba");
- *PriceDisplayComponent* - komponenta koja omogućava prikazivanje cena pojedinačnih proizvoda.

## KONTROLER KOMONENTE PRODUCTROWCOMPONENT

*Kontroler komponente ProductRowComponent ima zadatak definisanje taga za šablon.*

Kontroler komponente *ProductRowComponent* ima zadatak definisanje taga za šablon. Navedeno je urađeno u liniji koda 12, pod selektorom, a tag `<product-row>` već je upotrebljen unutar šablona *src/app/products-list/products-list.component.html*.

Dalje, moguće je primetiti (iz listinga koji sledi) da klasa ima dve osobine:

- *product* - ova promenljiva će biti podešena za tip podataka *Product* onog trenutka kada proizvod bude prosleđen iz roditeljske komponente;
- *cssClass* - ova osobina je obeležena dekoratorom *@HostBinding* i ukazuje na CSS klasu *attr.class* i njen element *"item"* kojeg je neophodno povezati sa host elementom, odnosno konkretnim objektom klase *ProductRowComponent*.

```
import {
  Component,
  Input,
  HostBinding
} from '@angular/core';
import { Product } from '../product.model';

/**
 * @ProductRow: komponenta za prikazivanje pojedinačnih objekata
 */
@Component({
  selector: 'product-row',
  templateUrl: './product-row.component.html',
})
```

```
export class ProductRowComponent {  
  @Input() product: Product;  
  @HostBinding('attr.class') cssClass = 'item';  
}
```

## ŠABLON KOMPONENTE PRODUCTROWCOMPONENT

*Šablon koristi tagove novih potkomponenata.*

Šablon komponente `ProductRowComponent` određen je datotekom `src/app/product-row/product-row.component.html` čiji je sadržaj prikazan sledećim listingom:

```
<product-image [product]="product"></product-image>  
<div class="content">  
  <div class="header">{{ product.name }}</div>  
  <div class="meta">  
    <div class="product-sku">SKU #{{ product.sku }}</div>  
  </div>  
  <div class="description">  
    <product-department [product]="product"></product-department>  
  </div>  
</div>  
<price-display [price]="product.price"></price-display>
```

Konceptijski, šablon ne sadrži ništa novo ali je moguće primetiti da insistira na podeli tekuće komponente na tri nove potkomponente koje su određene tagovima: `<product-image>`, `<product-department>` i `<price-display>`.

## KOMPONENTA ZA PRIKAZIVANJE SLIKE PROIZVODA

*Komponenta za prikazivanje slike proizvoda u potpunosti se oslanja na vlastiti kontroler.*

Komponenta za prikazivanje slike proizvoda je definisana tagom `<product-image>` (videti selektor) i u potpunosti se oslanja na vlastiti kontroler definisan `TypeScript` klasom `ProductImageComponent`.

Ovde je uvedeno malo odstupanje od dosadašnjeg izlaganja. Umesto definisanja šablona komponente u zasebnoj datoteci, on je realizovan unutar dekoratora `@Component` pod ključem `template` (linija koda 13-14).

Konačno, klasa uzima kao ulaz objekat klase `Product` i koristi CSS klasu `ui small image` koja se koristi za stilizovano prikazivanje slika pojedinačnih proizvoda.

```
import {  
  Component,  
  Input,  
  HostBinding
```

```

} from '@angular/core';
import { Product } from '../product.model';

/**
 * @ProductImage: komponenta koja prikazuje sliku proizvoda
 */
@Component({
  selector: 'product-image',
  template: `
    <img class="product-image" [src]="product.imageUrl">
  `
})
export class ProductImageComponent {
  @Input() product: Product;
  @HostBinding('attr.class') cssClass = 'ui small image';
}

```

## KOMPONENTA ZA PRIKAZIVANJE CENE PROIZVODA

*Komponenta za prikazivanje slike proizvoda definiše šablon unutar kontrolera.*

Komponenta za prikazivanje slike proizvoda, takođe, definiše šablon unutar kontrolera. Koristi jedan `{{ price }}` (linija koda 12) izraz koji zapravo prikazuje vrednost koju ova klasa prima na ulazu (linija koda 16)

```

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

/**
 * @PriceDisplay: komponenta prikazuje cenu pojedinačnog proizvoda
 */
@Component({
  selector: 'price-display',
  template: `
    <div class="price-display">{{ price }}</div>
  `
})
export class PriceDisplayComponent {
  @Input() price: number;
}

```

## KOMPONENTA ZA PRIKAZIVANJE ODELJENJA PROIZVODA

*Završna definicija aplikacije je komponenta za prikazivanje odeljenja proizvoda.*

Sledećim listingom je dat kontroler ove komponente koji kao ulaz uzima objekat klase *Product*.

```
import {
  Component,
  Input
} from '@angular/core';
import { Product } from '../product.model';

/**
 * @ProductDepartment: prikazuje detalje odeljenja sa kojeg
 * se uzima proizvod
 */
@Component({
  selector: 'product-department',
  templateUrl: './product-department.component.html'
})
export class ProductDepartmentComponent {
  @Input() product: Product;
}
```

Detaljniju pažnju je potrebno posvetiti šablonu komponente određenog datotekom *src/app/product-department/product-department.component.html*

```
<div class="product-department">
  <span *ngFor="let name of product.department; let i=index">
    <a href="#">{{ name }}</a>
    <span>{{i < (product.department.length-1) ? '>' : ''}}</span>
  </span>
</div>
```

Petlja *ngFor* prelazi preko svih *product.department* i pridružuje svaki *department* string osobini *name*. Izraz: *let i=index* određuje broj iteracije za petlju *ngFor*.

U tagu *<span>* upotrebljena je promenljiva *i* kojom je određeno prikazivanje simbola *>*. Cilj je prikazivanje odeljenja, kojem proizvod pripada, u formi stringa:

```
Women > Apparel > Jackets & Vests
```

Izraz *{{i < (product.department.length-1) ? '>' : ''}}* ukazuje da je moguće prikazati znak *>* ukoliko se ne radi o poslednjem odeljenju. U suprotnom, prikazuje se prazan string.

## NGMODULE I POKRETANJE APLIKACIJE

*Neophodno je proveriti da li su sve komponente registrovanje u NgModule klasi.*

Neophodno je proveriti da li su sve komponente registrovanje u *NgModule* klasi. Budući da je prilikom izrade projekta korišćen *Angular CLI*, sve komponente su automatski registrovane. Takođe, komponenta kojom se pokreće aplikacija bi trebalo da bude komponenta najvišeg

nivoa `AppComponent`. Ukoliko klasa sa lokacije `src/app/app.module.ts` i pod nazivom `AppModule` (NgModule klasa projekta) poseduje sledeći sadržaj, prethodni posao je dobro obavljen.

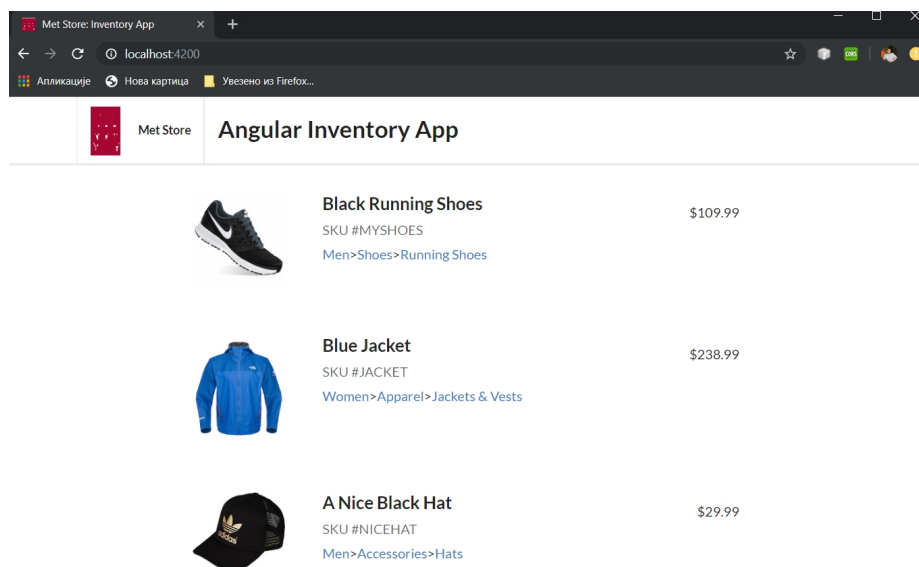
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { ProductListComponent } from './product-list/product-list.component';
import { ProductRowComponent } from './product-row/product-row.component';
import { ProductImageComponent } from './product-image/product-image.component';
import { ProductDepartmentComponent } from './product-department/product-department.component';
import { PriceDisplayComponent } from './price-display/price-display.component';

@NgModule({
  declarations: [
    AppComponent,
    ProductListComponent,
    ProductRowComponent,
    ProductImageComponent,
    ProductDepartmentComponent,
    PriceDisplayComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Slika 5.2.10 Klasa AppModule (NgModule klasa projekta)

Konačno, moguće je prevesti aplikaciju, instrukcijom `ng serve` i u veb pregledaču pozivom: `localhost:4200` pogledati rezultat izvršavanja aplikacije:



Slika 5.2.11 Konačan izgled aplikacije

**Kompletno urađen primer možete preuzeti odmah nakon ovog objekta učenja!**

## VIDEO MATERIJAL

*Angular 2 Tutorial - 2 - Architecture - trajanje 4:54 minuta.*

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

## ▼ Poglavlje 6

# Analiza – arhitektura podataka

## O ANGULAR ARHITEKTURI PODATAKA

*Problem upravljanja tokovima podataka ukoliko se počne sa dodavanjem novih funkcionalnosti.*

Sa ovim nivoom znanja, studenti mogu da se zapitaju kako je moguće upravljanje tokovima podataka ukoliko se počne sa dodavanjem novih funkcionalnosti. Na primer, kreira se nov pogled koji odgovara potrošačkoj korpi tako da će u budućnosti aplikacija imati mogućnost dodavanja novih proizvoda u korpu.

### Kako je moguće rešiti ovaj problem?

Jedini alat o kojem je do sada bilo diskusije jeste emitovanje izlaznih događaja. Kada bi bio izvršen klik na link ili dugme *add-to-cart* vrši se kreiranje događaja *addedToCart* i rukovanje događajem u korenskoj komponenti. Ovo može biti nezgodno.

Arhitektura podataka je velika tema sa brojnim opcijama. Angular je, po ovom pitanju, veoma fleksibilan i omogućava rukovanje širokim spektrom arhitektura podataka. Ostavlja se programeru da odluči koju želi da koristi.

## ARHITEKTURE PODATAKA I VERZIJE ANGULAR OKVIRA

*U okviru Angular 1, podrazumevana opcija je bila dvosmerno povezivanje podataka.*

U okviru Angular 1, podrazumevana opcija je bila dvosmerno povezivanje podataka (*two-way data binding*). Za početak ovo je bila odlična opcija:

- kontroleri imaju podatke;
- forme direktno manipulišu podacima;
- pogledi prikazuju podatke.

Problem se javio kada je ovaj pristup počeo da rezultuje kaskadnim efektima u aplikaciji pri čemu je praćenje tokova podataka postao otežan kako je projekat rastao.

Sledeći problem koji se javio kod dvosmernog povezivanja podataka jeste da kada se prosledi podataka "naniže" kroz komponente često se forsira čvrsto podudaranje stabla organizacije

podataka (*data layout tree*) sa stablom pogleda (*dom view tree*). U praksi, ove stvari moraju da se razdvoje.

Jedan od načina bavljenja navedenim scenarijem je kreiranje *ShoppingCartService*, koji bi bio *singleton* koji bi držao listu trenutnih stavki u potrošačkoj korpi. Ova usluga može obavestiti sve relevantne objekte kada je stavka u korpi promenjena.

Savremeni veb okviri, i nove verzije Angular-a, predlažu nešto drugačiji pristup - šablon jednosmernog povezivanja podataka (*one-way data binding*). U ovom slučaju, tokovi podataka teku isključivo naniže kroz komponente. Ukoliko je potrebno napraviti promenu, emituje se događaj koji izaziva promene koje se sa vrha prelivaju naniže. Na ovaj način su izbegnute brojne komplikacije i olakšano je praćenje tokova podataka u aplikaciji.

Srećom, postoje dva glavna kandidata za upravljanje arhitekturom podataka, o kojima će detaljno biti kasnije govora:

- primena *Observables* bazirane arhitekture, poput *RxJS*;
- primena *Flux* bazirane arhitekture.



## ▼ Poglavlje 7

### Vežba 6

#### POKAŽNA VEŽBA - NASTAVAK RADA SA PRIMEROM IZ VEŽBE 5

*Proširuju se Angular funkcionalnosti kroz nastavak rada sa primerom iz Vežbe 5.*

Kroz sledeći blok vežbi obradićemo tipiziranje podataka, kreiranje klasa, komponenti i obnoviti gradivo časa. Takođe, kod koji ćemo iskoristiti za nadograđivanje, prikazan je kroz vežbu broj 5 i može biti preuzet iz iste.

Prvo ćemo kreirati model sledećom komandom:

```
ng g class models/video
```

Po uspešnom kreiranju, dodaćemo tipove podataka, kao i podrazumevani konstruktor za sledeću klasu kao što je prikazano u delu koda ispod:

```
export class Video {
  title: string;
  description: string;
  link: string;

  constructor(title: string, description: string, link: string) {
    this.title = title;
    this.description = description;
    this.link = link;
  }
}
```

Unutar `app.component.ts`, dodajemo varijablu koja će predstavljati listu video snimaka čiji je tip podataka „`Video`“.

Kako to izgleda, možete videte na slici ispod:

```
3
4  @Component({
5      selector: 'app-root',
6      templateUrl: './app.component.html',
7      styleUrls: ['./app.component.scss']
8  })
9  export class AppComponent {
10     public videos: Video[];
11
12
13     constructor(){
14
15     }
16 }
17
```

Slika 7.1 Dodavanje liste snimaka u app.component.ts

## PRVA FUNKCIJA

*Kako je potrebno da ispunimo listu video snimcima, kreiraćemo **util** funkciju.*

Kako je potrebno da ispunimo listu video snimcima, kreiraćemo **util()** funkciju za generisanje slučajnih (random) reči kako bismo ispunili listu. Kod te funkcije dat je ispod:

```
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;
}
```

Popunjavanje liste video snimcima i random string vrednostima, data je kodom ispod:

```
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'
  ];

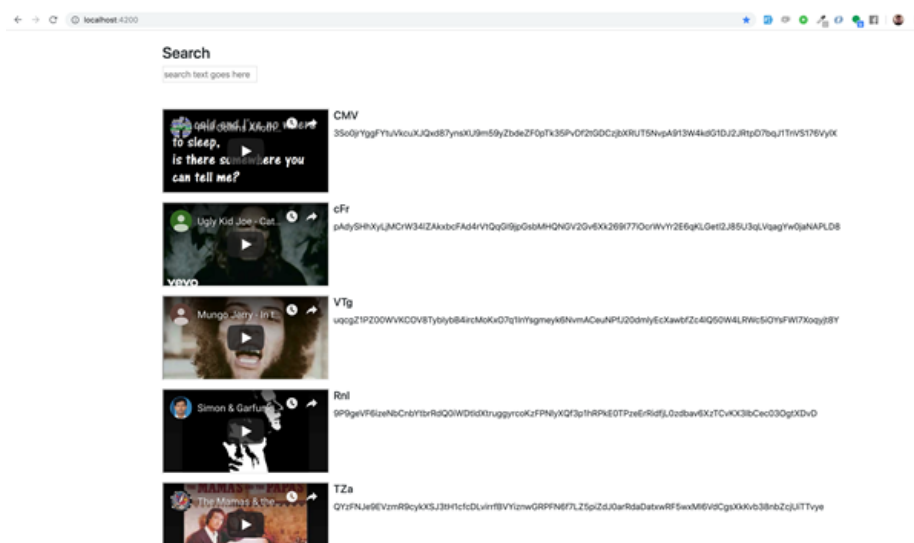
  constructor(){
    for(let i = 0; i < 6; i++) {
      this.videos.push(new Video(this._generateString(3),
this._generateString(100), 'https://www.youtube.com/embed/' + this._links[i]))
    }
  }

  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;
  }
}

```

Možemo videti u okviru konstruktora da se vrši iteriranje petlje i kreiranje random vrednosti u zavisnosti od indeksa.

Trenutni rezultat izvršenja aplikacije prikazan je na slici ispod:



Slika 7.2 Trenutni rezultat izvršenja modifikovane aplikacije

## BRISANJE SADRŽAJA LISTE

*Pristupamo izradi funkcionalnosti za brisanje i izmenu sadržaja liste iz same podkomponente.*

U nastavku vežbi pristupamo izradi funkcionalnosti za brisanje i izmenu sadržaja liste iz same podkomponente.

Neophodno je da izmenim korisnički interfejs u skladu sa zahtevima. Kod za izmenu koji treba dodati unutar *app.component.html* dat je ispod:

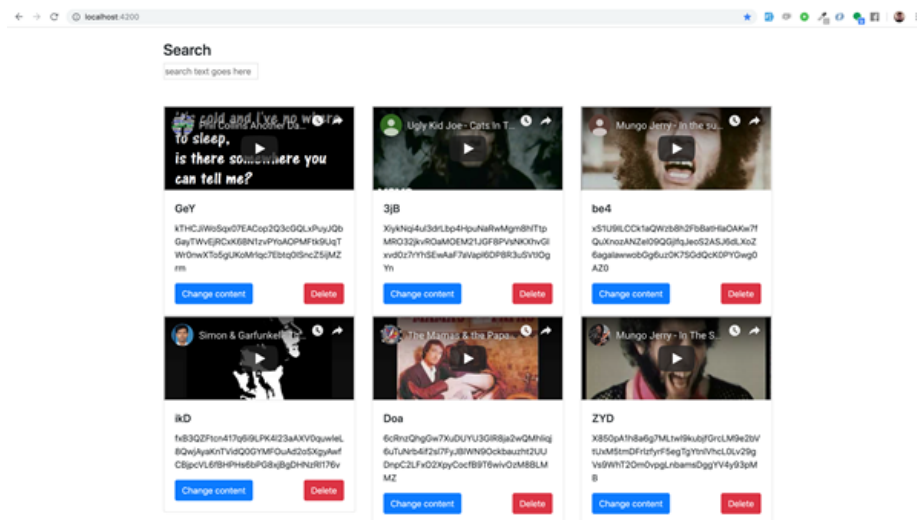
```
<div class="container mt-4 mb-4">
  <div class=" mb-5">
    <h3>Search</h3>
    <input [(ngModel)]="searchText" placeholder="search text goes here">
  </div>

  <div class="row">
    <div class="col-md-4" *ngFor="let video of videos | filter : searchText;
index as i; ">
      <app-video-box [video]="video"></app-video-box>
    </div>
  </div>
</div>
```

Kod za izmenu unutar *video-box.component.html* je takođe dat ispod:

```
<div class="card">
  <iframe height="100%" [src]="link"></iframe>
  <div class="card-body">
    <h5 class="card-title">{{video.title}}</h5>
    <p class="card-text">{{video.description}}</p>
    <div class="btns d-flex justify-content-between">
      <a href="#" class="btn btn-primary">Change content</a>
      <a href="#" class="btn btn-danger">Delete</a>
    </div>
  </div>
</div>
```

Trenutni rezultat izvršenja aplikacije prikazan je na slici ispod:



Slika 7.3 Nove izmene u izgledu aplikacije

## UPOTREBA DEKORATORA INPUT I OUTPUT

*Neophodno je izmeniti `VideoBoxComponent` za rad sa ulazno / izlaznim vrednostima.*

Neophodno je izmeniti `VideoBoxComponent`, tako da njen skript kod liči na ovaj dole prikazan:

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
import { DomSanitizer } from '@angular/platform-browser';
import { Video } from 'src/app/models/video';

@Component({
  selector: 'app-video-box',
  templateUrl: './video-box.component.html',
  styleUrls: ['./video-box.component.scss']
})
export class VideoBoxComponent implements OnInit {

  @Output() videoToDelete: EventEmitter<Video>;
  @Output() updateVideo: EventEmitter<Video>;
  @Input() video: any;
  public link: any;

  constructor(private _sanitizer: DomSanitizer) {
    this.videoToDelete = new EventEmitter();
    this.updateVideo = new EventEmitter();
  }

  ngOnInit() {
    this.embedUrl();
  }
}
```

```
public embedUrl() {
    this.link = this._sanitizer.bypassSecurityTrustResourceUrl(this.video.link);
}

public deleteProduct(): void {
    this.videoToDelete.emit(this.video);
}

public changeContent(): void {
    this.updateVideo.emit(this.video);
}
}
```

U gore navedenom kodu kreiramo dva *EventEmitter*-a koja će obavestiti našu parent komponentu, u ovom slučaju *AppComponent* koji video se treba izbrisati, a kom videu se treba promeniti sadržaj.

HTML kod (šablon) *VideoBox* komponente sa "click" događajima dat je ispod.

```
<div class="card">
  <iframe height="100%" [src]="link"></iframe>
  <div class="card-body">
    <h5 class="card-title">{{video.title}}</h5>
    <p class="card-text">{{video.description}}</p>
    <div class="btns d-flex justify-content-between">
      <a (click)="changeContent()" class="btn btn-primary">Change content</a>
      <a (click)="deleteProduct()" class="btn btn-danger">Delete</a>
    </div>
  </div>
</div>
```

## DODAVANJE OSLUŠKIVAČA

*Neophodno da na nivou AppComponent dodamo osluškivače za događaje.*

Potom je neophodno da na nivou *AppComponent* dodamo osluškivače za događaje, a to ćemo uraditi kroz par sitnih izmena u kodu i povezivanje (bind) sadržaja na komponentu.

Sledi inovirani listing šablona *app.component.html*:

```
<div class="container mt-4 mb-4">
  <div class="mb-5">
    <h3>Search</h3>
    <input [(ngModel)]="searchText" placeholder="search text goes here">
  </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>

```

Sledi inovirani listing klase *app.component.ts*:

```

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);  
  }  
  
}
```

U okviru funkcije `deleteVideo()`, možemo videti primenu `arrow` funkcija koja se koristi za filtriranje video snimka kojeg bi trebalo obrisati iz liste na osnovu njegove osobine `title`.

Kroz funkciju `updateVideo()`, možemo videti primenu `findIndex()` funkciju, koja takođe predstavlja ugrađenu funkciju JavaScript jezika za pronalazak indeksa na osnovu `callback()` funkcije.

**Kod vežbe je dostupan na sledećem linku:** <https://github.com/markorajevic/it255-v05/commit/01da3723588451b52c72b8a1bc3e7bfc19c9388a>

## INDIVIDUALNA VEŽBA

### *Samostalna dopuna zadatka sa pokaznih vežbi.*

Otvorite zadatak koji ste radili na pokaznim vežbama ili preuzmite delimično urađen sa linka: <https://github.com/markorajevic/it255-v05/commit/01da3723588451b52c72b8a1bc3e7bfc19c9388a>.

Dodajte nove funkcionalnosti u kreirani Angular projekat po sledećim zahtevima:

- Dodati novu osobinu (property) za klasu `Video` pod nazivom „`duration`“.
- Tip podatka treba da je integer i predstavlja trajanje video snimka u sekundama.
- Potom je neophodno dodati dugme čijim će se pritiskom izvršiti sortiranje video snimaka po dužini trajanja.



## ▼ Poglavlje 8

### Domaći zadatak 6

#### DOMAĆI ZADATAK

##### *Samostalna izrada domaćeg zadatka.*

Na projekat započet prethodnim domaćim zadatkom dodati sledeće funkcionalnosti:

1. Dodati funkcionalnost brisanja i izmene sadržaja na prethodni domaći zadatak *Methotels*.
2. Koristeći *EventEmitter*-e dodati funkcionalnost dodavanja soba iz zasebne komponente.

Domaći zadatak je potrebno dostaviti kao *Github commit* na prethodni, pod nazivom „*IT255-DZ06*“. 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

*Lekcija je bazirala svoje izlaganje kroz dve celine: TypeScript i funkcionisanje Angular okvira.*

Kao što je već istaknuto, lekcija je podeljena u dva dela:

- *TypeScript* diskusija i
- funkcionisanje *Angular* okvira.

U prvom delu je posebno bilo govora o migraciji Angular okvira sa standardnog JavaScript jezika ka njegovom proširenju u formi TypeScript notacije. Lekcija je kroz svoje objekte učenja i sekcije posebno diskutovala o:

- prednostima primene *TypeScript* okvira;
- primeni tipova podataka u *TypeScript* okviru;
- primeni objektno - orijentisanih koncepata *TypeScript* okvira;
- programskim alatima koji olakšavaju kodiranje u ovom okviru.

Drugi deo lekcije bio je rezervisan za analizu funkcionisanja Angular okvira sa akcentom na sledećim temama:

- aplikacija kao "top - level" komponenta;
- modeli u Angular aplikacijama;
- kreiranje i primena komponenta u Angular aplikacijama;
- analiza arhitekture podataka u Angular okviru.

## 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/>