Teorija iz predmeta inženjerstvo klijentskog sloja

JavaScript Objekti

1. **Koji tipovi podataka postoje u JavaScript-u?**
   -primitivni tipovi(imutabilni, porede se po vrednosti):
   -number(NaN, Infinity)
   -string
   -boolean
   -null
   -undefined
   -symbol
   -slozeni tipovi(***sve osim primitivnih tipova su objekti***)
   -svi primitivni tipovi osim null i undefined imaju wrapper objektne tipove(Number, String, Boolean, Symbol)

2. **Kako se vrše konverzije tipova u JS-u?**

Vrednost	String	Number	Boolean
undefined	"undefined"	NaN	false
null	"null"	0	false
true	"true"	1	
false	"false"	0	
""		0	false
"1.2"		1.2	true
"one"		NaN	true
0	"0"		false
NaN	"NaN"		false
Infinity	"Infinity"		true
5	"5"		true

Implicitna konverzija

- If ("abcd"){
  Console.log("!");
  }

Eksplicitna konverzija

Number("15");

3. **Objekti u JS?**
   Mutabilne kolekcije parova svojstvo-vrednosti:
   – nizovi
   – funkcije
   – regularni izrazi

– objekti

Nazivi svojstava: stringovi (ili simboli od ES6, više o tome kasnije)

Vrednosti svojstava: bilo šta

Neuređena kolekcija svojstava od kojih svako svojstvo ima vrednost koja može biti

– Primitivna

– Objekat

Porede se po referenci

Osim što svaki objekat ima svoje parove svojstvo vrednost, objekat može da nasledi svojstva nekog drugog objekta

Ovo se ostvaruje protitipskim nasledjivanjem

4. **Tipične operacije nad objektima?**

Kreiranje objekta

- Kao literal:
  i. **Literal** - notacija za postavljanje fiksnih vrednosti u source kodu
  ii. String literal, number literal, funkcijski literal, objektni literal, niz literal...
- Operatorom new
- Metodom Object.create()

Operacije nad svojstvima

- Postavljanje svojstva
- Pristup svojstva
- Brisanje svojstva
- Testiranje svojstva
- Enumeracija svojstva

5. **Šta je to objektni literal?**

Objektni literal predstavlja zadavanje objekata navođenjem svojstava i vrednosti (par svojstvo:vrednost).

Svojstvo je string ili simbol, a vrednost može biti bilo šta (Evaluirana vrednost izraza postaje vrednost svojstva)

6. **Šta radi operator new?**

Operator new kreira i inicijalizuje objekat

Nakon rezervisane reči new sledi poziv konstruktorske funkcije koja inicijalizuje novokreirani objekat

```
var o = new Object();
```

Pored ugrađenih konstruktora, moguće je i kreirati svoje konstruktorske funkcije

7. **Šta radi Object.create()?**

Kreira objekat postavljajući prvi parametar za prototip

Prilikom kreiranja novog objekta moguće je zadati njegov prototip – Object.create(prototipKreiranogObjekta);

Object.create() je statička metoda (metoda funkcije Object()), ne funkcija nekog konkretnog objekta

Moguće je kreirati objekat koji nema prototip – Ne nasleđuje ništa var x = Object.create(null);

U ovom objektu ne funkcionišu ni osnovne metode – Na primer toString(), pa neće raditi ni operator +

Tri načina da se kreira objekat koji za prototip ima Object.prototype:

```
var x = {};
var x = new Object();
```

var x = Object.create(Object.prototype);

8. **Kako se može pristupati vrednostia svojstava u JS?**
   - [ ] notacija
   - . notacija

-Ukoliko pokušamo da pristupimo svojstvu koje nije definistano, dobijamo UNDEFINED (ne baca se izuzetak)

-Greška je samo ako pokušamo da pristupimo svojstvu objekata koji ne postoji (TypeError izuzetak)

Ukoliko koristimo **. notaciju,** svojstvu objekta pristupamo **preko identifikatora**

Ukoliko koristimo **[] notaciju**, svojstvu pristupamo preko stringa

9. **Kako se u JS može izmeniti vrednost svojstva?**

-Izmena vrednosti svojstava

- Ukoliko svojstvo ne postoji, dodaće se
- Ukoliko svojstvo postoji, postaviće se nova vrednost

```
peraPeric["godina studija"]=4
```

10. **Objekti kao asocijativni nizovi**

U klasičnim OOP jezicima objekat može da ima samo fiksan broj svojstava (određen klasom)

- Nazivi svojstava se zadaju unapred

Pošto JavaScript nije klasičan OOP jezik, ovo svojstvo ne važi

- Često se kreiraju nova svojstva za postojeće objekte prilikom izvršavanja programa

Kada koristimo . notaciju, svojstvu objekta se pristupa preko identifikatora

- Identifikator je literalna vrednost koju nije moguće programski menjati
- Ona se zadaje u kodu

Kada koristimo [] notaciju, svojstvu se pristupa preko string naziva

- Moguće je kreirati naziv svojstva u toku izvršavanja programa

11. **Kako se mogu brisati svojstva objekata?**

Operator delete briše svojstvo iz objekta

Jedini parametar je izraz za pristup svojstvu

delete knjiga.autor

Nakon što je obrisana svojstvo, ukoliko želimo da mu pristupimo dobijamo undefined

Kakva je razlika između brisanja svojstva i postavljanja vrednosti svojstva na undefined? - gotovo da nema razlike

12. **Kako se mogu uklanjati varijable?**

Nedeklarisana varijabla je svojstvo globalnog Window objekta

Shodno tome, može se ukloniti pomoću delete

Deklarisana varijabla se tretira kao da je lokalna i ne može se ukloniti (makar i bila hoistovana do nivoa globalnog objekta)

13. **Kako se mogu ukloniti elementi niza?**

Koja je vrednost varijable x nakon sledećeg koda?

```
var x = [1,2,3,4,5]
```

```
delete x[2]

[1, 2, empty × 1, 4, 5]
```

Ukoliko smo hteli da uklonimo element niza, a da ne ostanu „rupe“ u nizu trebali smo da koristimo splice.

```
x.splice(2,1)
```

Sa indeksa 2 uklanjamo 1 element

14. **Kako se pristupa objektima u JS?**

Objektima se uvek pristupa po referenci.

15. **Za šta se koristi PROTOTIP u JS?**

JavaScript nema ugrađen mehanizam zadavanja klasa

Jedan od mehanizama definisanja hijerarhije objekata je pomoću prototipova

Svaki objekat ima prototip

Svi objekti koji su kreirani kao objektni literali kao prototip imaju Object.prototype, nativni objekat

Delegacija prototipa:

– Ako svojstvo postoji u objektu, vrati se njegova vrednost
– Ako ne postoji, traži se u prototipu
– Ako ne postoji u prototipu, traži se u prototipu prototipa
– ...
– Ako se ne pronađe do kraja lanca, vrati se undefined

Ako se izmeni bilo koji prototip u lancu, izmena će se odraziti na sve „naslednike“

Oprez! Objekat može da pristupi svom prototipu i da ga izmeni i time izmeni i ponašanje drugih objekata.

16. **Kako se mogu testirati svojstva u JS?**

Često postoji potreba da se proveri da li objekat ima neko svojstvo

To je moguće postići

– Tako što pokušamo da pristupimo svojstvu
– Pomoću in operatora ("toString" in o;)
– Pomoću hasOwnProperty() metode objekta
Proverava da li objekat ima sopstveno (nenasleđeno) svojstvo za zadati string (o.hasOwnProperty("toString");)
– Pomoću propertyIsEnumerable() metode objekta
Proverava da li objekat ima sopstveno (nenasleđeno) svojstvo i da li je za to svojstvo atribut enumerable postavljen na true za zadati string
Svojstva su enumerble osim ako se eksplicitno ne postavi da nisu (o.propertyIsEnumerable("z");)

17. **Kako se može vršiti enumeracija svojstava objekta?**

Možemo da koristimo for in petlju var naziv;

```
for (naziv in studentPeraPeric) {
console.log(studentPeraPeric[naziv]); }
```

Pristupamo i svim svojstivma prototipa

Ukoliko želimo da proverimo da li je svojstvo definisano u objektu, a ne prototipu, koristimo hasOwnProperty

Možemo i da koristimo propertyIsEnumerable metodu

Ne postoji garancija da će svaki put svojstva biti u istom redosledu

18. **Šta se dešava ako izbrišemo svojstvo u objektu koje je definisano u prototipu?**

delete studentPeraPeric.lk;

delete operator briše samo sopstvena svojstva, ne birše svojstva prototipa.

Ukoliko želimo da obrišemo svojstvo iz prototipa, to moramo da učinimo nad prototipskim objektom

– Ovo ima uticaj na čitav lanac nasleđivanja

19. **Šta su atributi svojstava i kakvi sve postoje?**

Svako svojstvo opisano je sa 4 vrednosti:

– value,

– writable

– ako je false, vrednost ne može da se izmeni,

– enumerable – ako je true, svojstvu može da se pristupi u for petlji (var prop in obj){}, – configurable – ako je false vrednost ne može da se izmeni, niti izbriše

20. **Kako se može pristupiti atributima svojstava?**

Atributima svojstava se može pristupiti pomoću funkcije Object.getOwnPropertyDescriptor()

```
Object.getOwnPropertyDescriptor({x:1}, "x");
//{value: 1, writable: true, enumerable: true, configurable: true}
```

21. **Kako se atributi svojstava mogu postavljati prilikom kreiranja objekata?**

```
var p = Object.create(Object.prototype, {x:{value:1, configurable:true}})
Object.getOwnPropertyDescriptor(p,'x')
//{value: 1, writable: false, enumerable: false, //configurable: true}
```

ili na sledeći način

```
var o = {};
Object.defineProperty(o, "x", { value : 1, writable: true, enumerable: false, configurable: true});
```

22. **Kako se atributi svojstava mogu promeniti?**

Izmena atributa svojstva kreiranog objekta:

```
Object.defineProperty(o, "x", { writable: false });
```

23. **Šta su getteri i setteri svojstava u JS?**

Umesto vrednosti svojstva moguće je zadati getter i/ili setter funkciju

Kada se pristupa vrednosti svojstva, biće izvršen getter (bez argumenata)

Kada se postavlja vrednost svojstva, izvršiće se setter sa novom vrednošću kao jedinim argumentom

Kada se postavlja vrednost svojstva, izvršiće se setter sa novom vrednošću kao jedinim argumentomKada se postavlja vrednost svojstva, izvršiće se setter sa novom vrednošću kao jedinim argumentom

```
get r() { return Math.sqrt(this.x*this.x + this.y*this.y); },
set r(newvalue) {
    var oldvalue = Math.sqrt(this.x*this.x + this.y*this.y);
    var ratio = newvalue/oldvalue;
    this.x *= ratio; this.y *= ratio;
},
```

JavaScript Funkcije

24. **Šta znači da su funkcije u JS FirstClass**

Funkcije u JavaScript-u su objekti

– Mutabilne kolekcije parova svojstvo-vrednost
– Imaju prototip (Function.prototype)
– Imaju svojstva za predstavljanje konteksta i koda funkcije

Mogu da se tretiraju kao bilo koji drugi objekti
– Varijabla može da primi funkciju kao vrednost
– Funkcije mogu da se smeštaju u kolekcije i druge objekte
– Mogu da se proslede drugim funkcijama kao parametri
– Mogu da budu vraćene iz drugih funkcija kao povratna vrednost
– Mogu da imaju svoje atribute sa vrednostima (koje opet mogu da budu i nove funkcije)

Razlikuju se od ostalih objekata jer mogu da se izvrše (can be invoked)

25. Kako se mogu kreirati funkcije u JS?

Funkcije u JavaScriptu mogu se kreirati na dva načina:

1. Deklaracijom funkcije

Prilikom deklaracije funkcije deklaracija funkcijske varijable i dodela vrednosti toj varijabli se hoistuje.

Deklaracije funkcija su zvanično zabranjeni u nefunkcijskom bloku – Na primer if bloku

Deklaracija funkcije definiše imenovanu funkciju (varijablu kojoj je dodeljena funkcija kao objekat) bez potrebe da se radi eksplicitna dodela vrednosti varijabli

Ne bi je trebalo definisati u uslovnom bloku if (false){ function f(){ console.log('!!!'); } }

Funkcijska varijabla je dostupna u svom opsegu i opsegu svog roditelja

2. Funkcijskim izrazom

Prilikom dodele vrednosti varijabli funkcijskog izraza, hoistuje se deklaracija varijable sa dodelom vrednosti undefined.

Funkcijskim izrazom funkcija se definiše kao deo većeg sintaktičkog konstrukta, tipično dodele vrednosti varijabli

Ovako definisane funkcije mogu da budu imenovane i neimenovane

Ne smeju početi rezervisanom rečju function

Česta praksa je da se funkcija istovremeno i definiše i pozove

```
(function sayHello() { alert("hello!"); }());
```

Ovakav pristup se zove neposredni poziv funkcijskog izraza (immediately-invoked function expression, IIFE)

26. Closure

Funkcijski izraz može da se nađe svugde gde može da bude izraz

Funkcija može da se definiše unutar druge funkcije – unutrašnja funkcija

– Unutrašnja funkcija ima pristup:
• Svojim parametrima i varijablama
• Parametrima i varijablama svoje spoljašnje funkcije, osim parametara this i arguments, čak i kada je spoljašnja funkcija prestala sa izvršavanjem
• Zatvaranje (closure)
- Funkcijski objekat kreiran u funkcijskom izrazu može ima pristup svom spoljašnjem kontekstu
• Veoma moćan alat u JavaScript-u

– Funkcija ima pristup kontekstu u kom je kreirana

27. Kako se funkcije mogu pozivati u JS?

Prilikom poziva funkcije prosleđuju se argumenti

Ukoliko se broj argumenata i parametara funkcije ne poklopi ne izaziva se izuzetak

– Ukoliko ima više argumenata nego parametara, „višak“ argumenata se ignoriše u funkciji

– Ukoliko ima manje argumenata nego parametara, parametri bez prosleđenih argumenata se inicijalizuju na undefined

– Kakvu podršku JavaScript onda ima za function overloading?

Prilikom poziva funkcije, pored prosleđenih parametara, funkcija dobija i dva dodatna

– arguments

– this

Vrednost parametra this zavisi od paterna poziva funkcije

Funkcija može da se pozove kao:

– Metoda

- Funkcije koje su vrednosti svojstava objekata
- Prilikom poziva funkcije vrednost parametra this je objekat nad kojim je funkcija pozvana
- Ovako definisane funckije su javne metode objekta

– Funkcija

- Ako funkcija nije vrednost svojstva objekta, onda se poziva kao funkcija u užem smislu reči var x = subtract(7,3)//4
- U ovom slučaju this je globalni objekat, bez obzira odakle je funkcija pozvana
- Bolje bi bilo da, ako je funkcija pozvana kao unitrašnja funkcija druge funkcije, this bude this spoljašnje funkcije
- Za definisanje helper funkcija moramo da koristimo colsure

– Konstruktor

- Pozivu funkcije prethodi ključna reč new
- Kreira se novi objekat koji je vrednost varijable this
- Skriveni atribut proto novog objekta dobije vrednost prototype konstruktora
- Oprez!

  – Funkcijski izraz ili definicija funkcije kojim se definišu konstruktori ni po čemo se ne razlikuju od ostalih funkcija

  – Bilo koju funkciju možemo pozvati sa new

  – Konstruktor možemo pozvati kao i bilo koju drugu funkciju

  – Može da bude uzrok nezgodnih bagova

  – Pomoću apply funkcije

28. Šta je APPLY funkcija?

Metoda funkcije koja omogućuje da se funkcija pozove sa proizvoljnim nizom argumenata i sa proizvoljnim argumentom this

Apply i monkey patching

• Monkey patching

– lokalna izmena programa u toku izvršavanja

• U JavaScriptu možemo da redefinišemo funkcije biblioteka koje koristimo u toku izvršavanja programa

• Moćno oružje, izvor nezgodnih bagova

29. Šta je i kako se koristi ARGUMENTS PARAMETAR

• Prilikom poziva funkcije možemo da prosledimo više argumenata nego što smo specificirali listom parametara
• Nedodeljeni argumenti se ignorišu
• Ukupnoj listi argumenata možemo da pristupimo pomoću parametra arguments
• Možemo da pišemo funkcije sa promenljivim brojem parametara

30. Šta su izuzeci i kako se njima upravlja u JS?

• Neuobičajeni događaju u izvršavanju programa
• Izuzetak je objekat koji ima name i message i može da ima još proizvoljnih svojstava
• Throw prekida izvršavanje funkcije i prebacuje izvršavanje na catch blok
• Za try postoji jedan catch blok
– Ukoliko se može desiti više različitih izuzetaka, treba proveriti name izuzetka u catch bloku

31. Da li se u JS mogu izmeniti tipovi?

• U JavaScriptu je moguće izmeniti postojeće tipove, uključujući i osnovne tipove
• Izmena se odmah odražava na čitav niz „naslednika“, kroz prototipove
• Potencijalan uzrok nezgodnih bagova

32. Opseg varijabli u JS?

• Deo programskog koda u kom je varijabla dostupna
• Većina programskih jezika sa sintaksom izvedenom iz C ima blokovski opseg
– Varijabla je dostupna u bloku u kom je definisana
• JavaScript ima funkcijski opseg vidljivosti varijabli
– Varijable i parametri su dostupni u čitavom telu funkcije u kojoj su definisani
• Ukoliko varijablu koristimo u funkciji pre nego što je deklarišemo, varijabla ima vrednost undefine
• Ukoliko je varijabla deklarisana (var x) u funkciji, koristimo tu varijablu • Ukoliko nije, tražimo je u spoljašnjoj funkciji
• Ukoliko nije defnisana ni u spoljašnjoj funkciji, tražimo je u spoljašnjoj funkciji za spoljašnju funkciju
• ...
• Sve dok ne dođemo do globalnog opsega!
• Ukoliko zaboravimo da deklarišemo varijablu, ili ćemo pristupiti varijabli koja se isto zove u spoljašnjoj funkciji ili ćemo napraviti globalnu varijablu.

33. Šta je hoisting u JS?

Hoisting je definisanje varijable na samom početku funkcije

```
var f = function () {
 ...
var x;
…
 x = 5;
};
```

• Isto što i:

```
var f1 = function () {
var x = undefined;
…
x = 5;
};
```

34. Šta je CALLBACK i kako se koristi?

• Izvršni kod koji se kao argument prosleđuje drugom izvršnom kodu

• Funkcija kao argument funkcije

```
function mySandwich(param1, param2, callback) {
alert('Started eating my sandwich.\n\nIt has: ' + param1 + ', ' + param2);
callback(); }
mySandwich('ham', 'cheese', function() {
alert('Finished eating my sandwich.');
});
```

• Često korišćen šablon u JavaScript-u (jQuery)

35. Šta su MODULI?

• Objekt koji ima interfejs ali skriva implementaciju

• U JavaScriptu se implementira pomoću funkcija i closure

– Funkcija koja:

• Definiše privatne varijable i funkcije

• Definiše privilegovane funkcije koje kroz closure imaju pristup privatnim funkcijama i varijablama

• Vraća objekat koji ima privilegovane funkcije

• Značajno smanjuje potrebu za globalnim varijablama

• Oduvek su bili najvažniji patern za organizaciju koda u JavaScriptu

• Oduvek su bili pristuni:

36. Šta je KASKADA

• Metoda često samo menja objekat

• Nema eksplicitni return

• Možemo da stavimo return this

• I da ulančamo pozive

37. Šta je MEMOIZACIJA?

• Funkcija može da čuva svoje ranije vrednosti da bi pojednostavili računanje

• Na primer, kada računamo Fibonačijeve brojeve, možemo da sačuvamo ranije vrednosti da bismo izbegli nepotrebno rekurzivno ponovno računanje

• Memoizacija je keširanje povratnih vrednosti determinističke funkcije

38. Šta je aplikacija funkcije i parcijalna aplikacija funkcije

• **Aplikacija funkcije**: proces primene funkcije na argumente poziva funkcije koji za cilj ima da produkuje povratnu vrednost funkcije.

• **Parcijalna aplikacija**: primena funkcije na neke od argumenata. Parcijalna aplikacija ne rezultuje povratnom vrednošću funkcije nego novom funkcijom u kojoj su neki od parametara fiksirani.

Bind metoda i parcijalna aplikacija

• Poziv bind metode vraća funkciju sa postavljenim prosleđenim argumentima

• Prvi argument je this objekat, argumenti koji slede su argumenti pozvane funkcije

• Slično kao apply, samo što apply vraća vrednost, a bind vraća funkciju

• Pogodno za parcijalnu aplikaciju funkcije

39. Šta je currying?

Currying se svodi na prevođenje primene funkcije koja prima više parametara u primenu serije funkcije od kojih svaka prima po jedan parametar

Prednost je u tome što ne moramo da eksplicitno radimo parcijalnu aplikaciju

Reaktivno programiranje

40. Šta je reaktivno programiranje?

• Programiranje sa asinhronim tokovima podataka

– Korisnički klikovi u aplikaciji

– Tweeter feed

• Plus funkcije za kreiranje, kombinovanje i filtriranje ovakvih tokova

41. Šta su tokovi podataka?

• Tok je sekvenca događaja u vremenu

• Mogu se desiti tri stvari:

----O----O--O---X---O-------|-->

O – događaj

X – greška

| – kraj toka

42. Šta je OBSERVER?

• Događaji se emituju asinhrono

– Definišu se funkcije koje će se pozvati na događaj, grešku i kraj

• Tok događaja je observabla

• Funkcije koje reaguju na događaje su observeri • Slušanje toka se zove pretplaćivanje (subscribing)

• Zvuči poznato?

– Observer dizajn šablon

43. Map funkcija?

– Za niz map kreiran novi niz, a za observablu se ne kreiraju novi objekti

Observable.from(Primer2.arr) .map(x => x*2).map(x => x*3).map(x => x-2).map(x => x+1).map(x => x+3) .filter(x => x>0).reduce((x,y) => x+y).subscribe(x => result = x);

44. Šta je observabla generatora (beskonačna observabla)?

• Observabla može da se kreira i iz beskonačnog generatora

let source:Observable = Observable .from(fibonacci());

45. Objasniti HOT i COLD Observable

• Hladna observabla ne produkuje vrednosti sve dok se bar jedan observer ne pretplati na nju

– Tek kada se observer pretplati, ona počinje da šalje vrednosti observeru

– Vrednosti se ne dele među observerima

– svaki observer dobija svoje vrednosti

• Vruća observabla (npr klikovi miša) ima vrednosti koje su deljene među observerima

– Vrednosti se emituju čak i ako ih niko ne sluša

46. Koji su operatori kombinacije?

1.CONCAT

let source3:Observable = Observable.concat(source1, source2);

2.MERGE

let source3:Observable = Observable.merge(source1, source2);

47. Koji su operatori kreiranja?

• from – konvertuje objekat u varijablu

• of – konvertuje sekvencu u varijablu

48. Kako se rukuje greškama u JS?

• Slanje error handle funkcije pri pretplaćivanju
• catch operator
let source:Observable = Observable.throw(new Error('There are no numbers')); let example:Observable = source.catch(err => Observable.of(`error ${err}`));

49. Objasniti MULTICASTING
• Observable su default hladne, ako ne specificiramo da su vruće
• Za to koristimo publish() kao što smo videli

50. Operatori filtriranja
• filter – vraćaju se samo elementi koji zadovoljavaju zadati kriterijum
• debounce – ignorišu se elementi koji pristižu na kraćem periodu od zadatog
• take – preuzimanje n elemenata
• takeUntil – preuzimanje elemenata dok ne stigne signal od druge observable
• distinctUntilChanged – emituje se jedino vrednost različita od predhodne

51. Operatori transformacije
• **map** – projekcija svakog elementa
• **mergeMap**
• AKA flatMap
• Ne očuvava redosled elemenata
Spoljna observabla
--O---------O---------O-------->
concatMap O ->
-D---D-> --D---D-----D---------D---D-D-->
Rezultujuća observabla
• **concatMap**
• Čeka da se unutrašnja observabla završi da bi spoljašnja uzela sledeći element
• Čuva redosled spoljašnje observable

mergeMap i concatMap:
• Koriste se sa observablama višeg reda
– Stavke observable su i sami observable
– Treba da ih svedemo na jednu jedinu observablu
• Na primer, kada se pretplatimo na observablu unutar pretplate na observablu
outerObservable.subscribe(outerItem => { outerItem.subsribe(innerItem => { foo(innerItem); })});
Search polje se menja (spoljašnja observabla) i trigeruje http poziv na svaku promenu teksta (unutrašnja obesrvabla) Klik miša (spoljašnja observabla) i odložena animacija na klik (unutrašnja observabla)

52. Šta je klasa Subject?
• Klasa Subject nasleđuje klase Observable i Observer
• Najčešće proksi između izvornog toka i krajnjih pretplatnika

53. Angular i RxJS
• Angular je razvijen na RxJS
– EventEmitter nasleđuje Subject
– HTTP komunikacija je bazirana na Observablama

ES6

54. Izvršavanje ES6

• Rapidna evolucija ES i priroda okruženja u kojima se JavaScript izvršava (browseri nad kojima nemamo kontrolu) odlaže direktno korišćenje novih mogućnosti jezika

• Moguća rešenja:

– Odustati od ES6

– Koristiti ES6 uz napomenu da je aplikacija podržana samo u ograničenom broju browsera – Koristiti ES6 nad okruženjima nad kojim imamo punu kontrolu (Node od verzije 6.5), a ES5 za razvoj klijentskih aplikacija

– Koristiti ES6 i u klijentskim aplikacijama uz transpajliranje koda

55. Šta predstavlja transpajliranje koda?

• Transpajler (transformation compiling, sourceto-source compiling) je kompajler koji prevodi programski kod iz jednog programskog jezika u ekvivalentan programski kod drgugo jezika

• ES6 prevodimo u ES5

• (ES6 nije jedini jezik koji se transpilira u ES5)

56. Opseg vidljivosti u ES6

• U ES5 opseg vidljivosti varijable je uvek bila funkcija

• Od ES6 moguće je deklarisati i varijable čiji je ospeg vidljivosti blok

• Za to se koristi let deklaracija

57. Let i for petlja

• Let u zaglavlju for petlje ne deklariše jednu varijablu za celu for petlju, već po jednu varijablu za svaku iteraciju for petlje

58. Konstante

• Blokovska varijabla za koju nije dozvoljena promena vrednosti

• Ako je mutabilna, moguće je menjati je

```
{
const a = [1,2,3];
a.push( 4 );
console.log( a );
// [1,2,3,4] a = 42;
// TypeError!
}
```

59. Spread/rest operator

• Kada se koristi ispred niza, „proširi“ niz na njegove vrednosti

• Kada se koristi ispred vrednosti koje nisu niz, okupi ih u niz

60. Default vrednosti parametara

• U ES6 moguće je zadati default vrednosti parametara

```
function foo(x = 11, y = 31) {
        console.log( x + y );
}
```

61. Šta je destrukturiranje?

Destrukturiranje je strukturirana dodela vrednosti

• Zadavanje paterna za destrukturiranje vrednosti koja se dodeljuje

– [a,b,c] znači da će se vrednosti niza dodeliti varijablama a, b i c

– { x: x, y: y, z: z } znači da će se vrednosti atributa x,y i z dodeliti varijablama x, y i z

62. Kako se vrši dodela svojstava objekta?

• Ako se naziv atributa u objektu koji se destruiše poklapa sa nazivom varijable kojoj se dodeljuje vrednost, dovoljno je da navedemo samo jedan naziv

```
var { x, y, z } = bar();
console.log( x, y, z );
```

• Duži zapis omogućuje da se vrednosti iz objekta dodele varijablama čija imena se razlikuju od imena atributa u objektu

```
var { x: bam, y: baz, z: bap } = bar();
console.log( bam, baz, bap ); // 4 5 6
console.log( x, y, z ); // ReferenceError
```

63. Šta je delimična dodela destrukturiranjem?

• Prilikom dodele destrutkturiranjem ne moraju sve vrednosti da se dodele

```
var [,,c,d] = foo();
```

64. Spread/rest i destrukturiranje

• Operator ... može da se koristi i u kombinaciji sa destrutkuriranjem

```
var a = [2,3,4];
var [b, ...c] = a;
console.log( b, c ); // 2 [3,4]
```

65. Destrukturiranje i default parametri

• Prilikom dodele destrukturiranjem moguće je postaviti default vrednosti

```
var [ a = 3, b = 6, c = 9, d = 12 ] = foo();
var { x = 5, y = 10, z = 15, w:WW = 20 } = bar();
console.log( a, b, c, d ); // 1 2 3 12
console.log( x, y, z, WW ); // 4 5 6 20
```

66. Destrukturiranje parametara

• Parametri funkcije u JavaScriptu su lokalne varijable kojima se dodeljuje vrednost argumenata

– Parametri funkcije mogu da se destrukturiraju

```
function f1( [ x, y ] ) {
        console.log( x, y );
}
f1( [ 1, 2 ] ); // 1 2
```

67. Koncizna svojstva i metode objekta

• Ukoliko se svojstva zovu isto kao i varijable koje im se dodeljuju, nije ih potrebno posebno imenovati

• Ukoliko se svojstvu dodeljuje funkcije, nije potrebno eksplicitno navesti function

– Ovako kreiana funckija je neimanovana!

```
var x = 1, y = 2,
        o = {
        x,
        y,
        show(){
        console.log('x:',this.x,',y:',this.y);
        }
 }
```

68. Šta su template literali?

Template literali su interpolirani string literali.

Moguće je interpolirati stringove vrednostima izraza
function upper(s) { return s.toUpperCase(); }
${upper("consequat elit porttitor")} - negde u tekstu;

69. Objasniti Arrow funkcije

• => omogućuje leksički binding za this, arguments i super
var foo = (x,y) => x + y;
• Varijabli foo dodeljena je anonimna funkcija koja primi dva parametra i vrati njihov zbir
• Ukoliko je jednolinijska, ne moramo da telo obuhvatimo u {} i ne moramo da pišemo eksplictno return

70. for … of

• Prolazak kroz vrednosti
for (var val of a) { console.log(val); }

71. for … in

• Prolazak kroz ključeve
for (var idx in a) { console.log(idx); }

72. Šta je Simbol?

• ES6 uvodi novi primitivni tip: Symbol
• Služi za reprezentovanje jedinstvenih identifikatora najčešće u kombinaciji sa konstantama
const MY_KEY = Symbol();

73. Kako se koriste iteratori?

• Implementiraju sledeće metode:
– next() – obavezna, preuzimanje sledeće vrednosti
– return() – opciona, preuzimanje trenutne vrednosti
– throw() – opciona, izazivanje greške uz vraćanje trenutne vrednosti

74. Generatori

• Funkcije čije izvršavanje može da se pauzira i da se kasnije nastavi
• Ciklus pauziranja/nastavljanja omogućuje dvosmernu razmenu poruka
– Generator može da vrati vrednost u toku svog izvršavanja
– Kod koji poziva generator može generatoru da prosledi vrednost
• Sintaksa:
function *foo() { // .. } var it = foo();
• Poziv generatora vraća iterator pomoću koga možemo da se „krećemo“ kroz vrednosti generatora

75. Kako se pauzira generator?

• Pomoću operatora yield zaustavljamo izvršavanje funkcije, vraćamo vrednost i primamo vrednost

```
function *foo() {
    while (true) {
        var x = yield Math.random();
        console.log('x:',x);
    }
}
```

76. Delegiranje

Delegiranje se vrši pomoću yield *
• yield * iterator
– Prenosi svoje generisanje next() vrednosti na iterator

– Vraća dok se iterator na potroši

```
function *bar() {
    yield *[1,2,3];
}
```

77. Namena generatora

• Produkovanje serije vrednosti kroz koju možemo da se iteriramo

• Izvršavanje reda taskova uz mogućnost kontrole „od spolja“

78. Moduli u ES6

• Jedan modul - jedan fajl

– Čak i kada pravimo aplikaciju koja se izvršava u browseru svaki modul mora da bude u zasebnom fajlu

– HTTP2 bi trebalo da reši problem učitavanja velikog broja fajlova jer se oslanja na stalnu socket konekciju

• API modula je statički

– Statički se odredi šta će biti eksportovano iz modula i to ne može kasnije da se promeni

• Moduli su singleton objekti

– Prilikom svakog učitavanja modula dobija se referenca na isti objekat

Moduli – imenovani export

```
export function foo() { // .. }
export var awesome = 42;
var bar = [1,2,3];
export { bar };
```

• Sve što nije eksportovano iz modula je privatno

– Čak i ako se deklariše sa var

– U modulu ne postoji globalni opseg!

Moduli – default export

• Ukoliko modul ima jedan glavni export, on se eksportuje kao export default

```
function foo() { .. }
export function bar() { .. }
export function baz() { .. }
```

• Kada se koristi defaultni export import je koncizniji

Moduli – import

• Imenovanih eksporta:

– Dekstrukturiranjem

```
import { foo as theFooFunc } from "foo";
```

• Default eksporta:

– Direktno

```
import foo from "foo";
```

• Import čitavog APIja:

```
import * as foo from "foo";
```

• **Browserifikovanje** – nužno zlo

```
browserify primer13/primer13-uc.js -o
primer13/primer13.js -t [babelify]
```

79. Klase u ES6

• Prototipska priroda JavaScripta s jedne strane i konstrukti kao što su new, instaceof i .contructor s druge pokazuju ambivalentni odnos JavaScript-a prema klasičnom OOP

• Od ES6 u JavaScriptu je moguće koristiti „klase“

– ES6 kod se transpajlira u ES5 koji nema klase, odnosno klase u ES6 su samo sintaktički šećer ...

– ... što dodatno povećava ambivalenciju

• Rezervisana reč Class identifikuje blok koda čiji sadržaj definiše svojstva prototipa konstruktora

80. Klase i konstruktorske funkcije

• Klase uvek moraju da se pozivaju sa new

• Klase se ne hoistuju

– Klasa mora da se deklariše pre nego što se instancira

• Može se konfigurisati kako instanceof radi, preko Symbol.hasInstance

81. Kako se vrši nasleđivanje u ES6?

• Sintaktički šećer za delegaciju prototipa sa zbunjujućim nazivom „nasleđivanje“ u OPP-like notaciji

```
class Bar extends Foo {
        constructor(a,b,c) {
                super( a, b );
                this.z = c;
        }
        gimmeXYZ() {
                return super.gimmeXY() * this.z;
        }
}
```

• extend i super

• Extends

– uspostavljanje reference prototipa

• Super

– referenca na konstruktor roditeljske „klase“

• konstruktor

• Nije neophodno pisati konstruktor, ukoliko ga ne napišemo imaćemo default konstruktor

• Ako imamo klasu naslednicu, njen default konstruktor se ponaša tako što sve što mu se pošalje prosledi super konstruktoru

```
constructor(...args) {
        super(...args);
}
```

• U konstruktoru nalslednice nije moguće pristupiti objektu this pre nego što se pozove super(...) konsturktor

• static

• Vrednost se postavlja na svojstvo funkcije, a ne na svojstvo prototipa

TypeScript

82. Šta je TypeScript?

• TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.
  – Tipovi
  – Interfejsi
  – Buduće ES mogućnosti (Anotacije/Dekorateri, async/await)

83. Tipovi u TS

• Opcioni, ne moramo da ih navodi, ali možemo
  – Boolean
    let isDone: boolean = false;
  – Number
    let decimal: number = 6;
  – String
    let color: string = "blue";
    • Jezik je slabo tipiziran, pa možemo da uradimo sledeće:
    let fullName: string = "Bob Bobbington";
    let age: number = 37;
    let sentence: string = "Hello, my name is " + fullName + ".\n\n" +"I'll be " + (age + 1) + " years old next month."
  – Niz
    let list: number[] = [1, 2, 3];
    let list: Array = [1, 2, 3];
  – Tuple
    let x: [string, number];
    x = ["hello", 10]; // OK
    x = [10, "hello"]; // Error
  – Enum
    enum Color {Red = 1, Green, Blue};
    let c: Color = Color.Green;
  – Any – bilo šta
    let notSure: any = 4;
    notSure = "maybe a string instead";
    notSure = false;
  – Void – ništa (undefined ili null)
    function warnUser(): void {
      alert("warning message");
    }
    let unusable: void = undefined;

84. Eksplicitno kastovanje

• *let strLength:*
number = (someValue).length;
• *let strLength:*
number = (someValue as string).length;

85. Interface

• Prilikom provere tipa u TypeScriptu posmatra se struktura objekta

– Duck typing

• Ovakvi „tipovi“ se zadaju pomoću interfejsa

```
interface IComment {
    createdAt?:string;
    updatedAt?:string;
    signedBy?:string;
    text:string;
    //kolekcija komentara
    comments?: Comment[];
}
```

• Objektni literal čiji tip se zadaje interfejsom mora po svojoj strukturi da odgovara zadatom interfejsu

– Mora da ima sve obavezne atribute

– **Ne sme da ima ni jedan atribut koji nije zadat interfejsom** (excess property checking)

– Može, a ne mora da ima opcione atribute

```
let c1: IComment = {
    createdAt: 'Sun Nov 19 2017 20:16:14',
    text: 'tekst komentara',
    comments: []
}
```

• Object literal may only specify known properties, and 'related' does not exist in type 'IComment'.

• Excess property checking je vrlo lako zaobići eksplicitnim kastovanjem:

```
let c2: IComment = {
    createdAt: 'Sun Nov 19 2017 20:16:14',
    text: 'tekst komentara',
    comments: [],
    related: ['http://www.example.org/comments/ 123321']
} as IComment;
```

• readonly – Svojstvo može da dobije vrednost prilikom kreiranja objekta, ali naknadno ne može da mu se izmeni vrednost

• Opciono polje - [propName: string]: any;

86. Interface funkcije

• Pored strukture objekata, interfejsi se mogu koristiti i za opisivanje funkcija.

– Interfejsom se zadaje signatura funkcije

```
interface FilterFunction {
    (sourceList: string[], subString: string): string[];
}
```

87. Nasleđivanje interfejsa

```
interface IPoint3D extends IPoint{
    z: number;
}
```

• Pored drugog interfejsa , interfejs može i da proširi klasu.

• U tom slučaju, interfejs nasleđuje sve atribute i metode klase, bez njihovih implementacija

```
interface IPoint4D extends IPoint{
    u:number;
```

```
}
```

88. Implementacija interfejsa

Klasa može da implementira interface

```
interface IPoint {
        x: number;
        y: number;
        show():void;
}
class PointClass implements IPoint{
        x: number;
        y: number;
        z: number;//pri tome moze da uvede nove atribute
        constructor(x: number, y: number, z:number) {
                this.x = x; this.y = y; this.z = z;
        }
        show() {
                console.log('(x:', this.x, ', y:', this.y, ', z:', this.z, ')');
        }
}
let a = new PointClass(100, 200, 300);
```

89. Klase – modifikatori vidljivosti svojstava

• Public
  – Default
• Private
  – Ne možemo da pristupimo svojstvu van klase (privilegovane metode objekta) ni iz klase naslednice
• Protected
  – Ne možemo dapirstupimo svojstvu van klase, ali možemo da pristupimo iz klase naslednice

90. Klase

• Static – svojstva na nivou „klase“ (konstruktorske funkcije) umesto na nivou instance
• Abstract – „klase“ koje mogu da se naslede, ali ne i instanciraju
• Prilikom kreiranja klase kreiraju se:
  – Tip instance klase
  – Konstruktorska funkcija koja će biti pozvana prilikom insanciranja klase pomoću **new**
• Klasa ima
  – Aspekt instance
  – Statički aspekt

91. Tip funkcije

```
let subtract: (x:number, y:number) => number =
        function(x:number, y:number){
                return x-y;
        }
```

Ne moramo navoditi pun tip funkcije kada je kreiramo. TSC komplajer će zaključiti tip (type inference) ako je tip dat na jedno strani opratora dodele.

92. Da li u JavaScriptu možemo da imamo polimorfne funkcije koje imaju različito ponašanje u zavisnosti od broja i tipa parametara?

• U JavaScriptu funkcionalnost preklapanje funkcija (overloading) je bila ostvarena tako što jedna funkcija ima različite povratne vrednosti u zavisnosti od argumenata koji joj se pošalju.

93. Generici

• JavaScript je dinamički tipiziran

– Nema potrebe za parametarskim polimorfizmom

• TypeScript je statički tipiziran

– Ako želimo da pravimo komponente koje se mogu koristiti u različitim kontekstima treba nam parametarski polimorfizam

Generičke funkcije

• Funkcija koja primi vrednosti i vrati je λx.x

```
function identity(x:T): T { return x; }
```

Generički interfejsi

```
interface IStack{
    push(item: T): void;
    pop(): T;
}
```

Generičke klase

```
class Stack implements IStack{
    private values: T[];
    constructor(){ this.values = []; }
    push(item: T){ this.values.push(item); }
    pop(){ return this.values.pop(); }
}
```

94. Zaključivanje tipova

• Ako tip nije eksplicitno zadat, kompajler će ga zaključiti na osnovu dodele vrednosti

let x = 5;

x = 'Hello world';

Type '"Hello world"' is not assignable to type 'number'.

• U situaciji koja nije jednoznačna, zaključuje se **najbolji zajednički tip**

– Na primer u heterogenim kolekcijama

let y = [1,5,true];

y.push('hello world');

Argument of type '"hello world"' is not assignable to parameter of type 'number | boolean'.

95. Nominalni vs. strukturalni tipovi

• U jezicima sa nominalnim tipovima (Java, C#) ovo bila greška

• TypeScript ima skutrkturalne tipove (setimo se duck typinga u JavaScriptu) pa nema greške

– Dovoljno je da objekat ima predviđene atribute i metode pa da je smatramo instancom tipa

– Kažemo da je tip Book kompatibilan sa tipom Album i obrnuto

– Tip a je kompatibilan sa tipom b ako b ima sve atribute i metode koje ima i a.

96. Kompatibilnost funkcija

• U slučaju funkcija:

– Povratne vrednosti moraju da se poklope
– Parametri dodeljivane funkcije moraju da se poklope sa prvih n parametara funkcije kojoj se dodeljuje

97. Kompatibilnost klasa

• Slično kao i kompatibilnost objektnih literala uz razliku da klase imaju statički aspekt i aspekt instance
– Posmatra se samo aspekt instance
– Zanemaruju se statička svojstva i konstruktor klase
• Modifikatori prava pristupa utiču na kompatibilnosti
– Privatno svojstvo je kompatibilno sa privatnim
– Protected je kompatibilno sa protected

98. Presek tipova

• Kombinovanje više tipova u jedan
– Objekat ima svojstva svih tipova

99. Unija tipova

• Više tipova od kojih je jedan dodeljen

100. Algebra tipova

• Videli smo da imamo proizvod i zbir tipova.
• Ako imamo * i +, da li imamo 0 i 1?
− Tip koji je neutralan za sabiranje i tip koji je neutralan za množenje
− 1 – void ili null
− 0 – never

101. Guards

• Kada za varijablu postavimo da joj je tip unija tipova, možemo pristupati samo atributima koji postoje u **svim tipovima unije**.
• Ako bismo hteli da pristupimo atributima nekog od tipova unije, morali bismo da radimo asertaciju tipa pri svakom pristupu (vehicle as Car).drive
• Ili možemo da postavimo guard, pa da asertaciju tipa uradimo samo jednom
• Kada je pozvana isCar funkcija nad varijablom vehicle, tip varijable je sužen na Car …
– …Ako je povratna vrednost true
– Ako povratna vrednost nije true, sužen je na Motorbike, pošto je to jedini preostali tip u uniji

102. Alias tipa

• Novo ime za postojeći tip
– Primitivni tip, uniju, tuple, ili bilo koji drugi tip
– Ne kreira se novi tip, kreira se samo novi naziv za postojeći tip

103. Dekoratori

• Deklaracija koja se vezuje za deklaraciju klase, metode, akcesora, svojstva ili parametra
• Dekorateri su oblika @izraz pri čemu izraz treba da se evaluira u funkciju koja će se pozvati u runtimeu
• Funkcija može da pristupi dekorisanom objektu i da ga po potrebi izmeni
@logClass class Person { }

Angular

104. O angularu

- AKA Angular 2,4,5,...
- Osnovni gradivni blok aplikacije su komponente
- Komponenta objedinjuje prikaz (html teplate) i kod (TS klasa)
- Aplikacija se uvek pravi kao stablo komponenti
- Na prvi pogled, konceptualno potpuno različit od AngularJS

105. Moduli

- Angular aplikacije su modularna i koriste se NgModules
  – TS klase dekorisane @NgModule dekoraterom
- Svaka aplikacija ima makar jedan (korenski) modul: AppModule
- @NgModule dekorate definiše:
  – declarations - klase za prikaz koje pripadaju tom modulu. Mogu da budu komponente, direktive i pipes.
  – exports - deklaracije koje će biti vidljive u templejtima komponent drugih modula.
  – imports - drugi moduli čije eksportovane klase su vidljive u ovom modulu.
  – providers - kreatori servisa koje unosi ovaj modul. Servisi su vidljivi u celoj aplikaciji.
  – bootstrap - glavni prikaz, odnosno korenska komponenta, koja će omogućiti sve ostale prikaze. Samo korenski modul ima postavljeno ovo svojstvo.

106. Višeznačnost termina modul u JS

- U ES5 modul je bio factory funkcija koja vraća objekat sa javnim APIjem i skriva privatna svojstva u closure
- Od ES6 svaki fajl je modul i javni API se dobija eksportovanjem
- U Angularu NgModule je potpuno nov koncept: klasa dekorisana @NgModule dekoratorom
  – Prema konveciji jedan NgModul se smešta u jedan ES6 modul
- U ovim materijalima, termin module označava ES6 modul, a termin NgModule označava TS klasu dekorisanu @NgModule dekoraterom

107. Angular biblioteke

- Uz Angular se distribuiraju i prateće biblioteke modula kao paketi
- Počinju prefiksom @angular
- Instaliraju se pomoću npm

108. Komponente

- Komponenta objedinjuje element prikaza i aplikativnu logiku (TS klasu) koji ga kontroliše
- Klasa komunicira sa prikazom kroz javni API klase (atributi i metode)
- Komponente su međusobno povezane tako da čine stablo komponenti

Anatomija komponente

- TS klasa dekorisana dekoratorom @Component
  – selector: naziv elementa koji će se koristiti u stranici,
    - <app-component></app-component>
  – templateUrl: url HTML templejta,
  – styleUrls: stilovi koji se primenjuju samo na ovu komponentu
- Javni API klase je dostupan u templejtu

```
comment:Comment = {
        signedBy: "mitar trol",
        text: "tekst odgovora 3"
};
```

109. Templejt

• Manipulacija DOM stablom

** {{comment.signedBy}} **
** **{{comment.text}}** **

• Ažuran prikaz atributa signedBy i text atributa comment iz APIja klase CommentComponent

• Interpolacija {{ izraz }}

– Ažurna evaluirana vrednost izraza između dve vitičaste zagrade se ugrađuje u DOM stablo

110. Registrovanje komponente

• Kada je komponenta kreirana potrebno je registrovati je u aplikaciji

• Registrovanje komponente se obavlja dodavanjem komponente u deklaracije ngModule-a

– Deklaracije obuhvataju direktive (komponente, direktive atributa) i pipes

• Svaka aplikacija ima makar jedan ngModule, to je korenski modul (AppModule)

– Postaje nepregledan u velikim aplikacijama

– Rešenje: uvode se featur moduli

• ngModule koji posvećen featureu aplikacije

– Komponente koje spadaju u isti feature se registruju u njegov feature modul

• Feature module je veoma sličan korenskom

– Razlike:

1. Korenski modul se bootuje pri pokretanju aplikacije; feature modul se uvozi da bi se proširila aplikacija
2. Feature modul može da sakrije deklaracije od drugih ngModula

```
@NgModule({
declarations: [
AppComponent,
CommentComponent,
CommentListComponent,
BlogEntryComponent,
BlogEntryListComponent,
BlogEntryFormComponent,
SearchEntryComponent,
EmphasizeDirective
 ]
```

111. Prenos podataka u komponente

• Obzirom da komponente čine stablo, često smo u situaciji da potomačka komponente treba da primi vrednosti od roditeljske komponente

• Podaci uvek „teku“ od roditelja ka potomcima

• U Angularu je moguće prene podatke iz roditeljske komponente u potomačku pomoću

– Postavlja se dekorater @Input() na atribut u koji se unosi vrednost

– U templejtu roditeljske komponente se zadaje vrednost koja se prenosi

• Tok podatak je uvek od roditeljske komponente ka potomačkoj kroz @Input() atribute
• Međutim, česta je situacija da potomačka komponenta treba da izmeni podatke, a da ta izmena bude vidljiva u roditljskoj komponenti
• U tom slučaju, potomačka komponenta izaziva događaj koji roditljska komponenta osluškuje
• Pri emitovanju događaja iz komponente može da se pošalje objekat koji će biti dostupan u obradi događaja u roditeljskoj komponenti
• Za emitovanje događaja koriste se atributi anotirani @Output() dekoraterom
• Atribut koji emituje događaj je tipa EventEmitter<T>
    – Parametrizovan tipom objekta koji šalje kroz događaj
• Prilikom postavljanje komponente u templejtu roditeljske komponente, pored njenih inputa postave se i output
• Na svako emitovanje događaja iz BlogEntryFormComponent, pozvaće se metoda saveBlogEntry roditelja i kao parametar će se proslediti poslati objekat
• Komponenta ovako može da obavesti samo roditelja (direktnog pretka)
    – Ako bismo trebali da emitujemo događaj roditelju roditelja, morali bismo da presretnemo događaj u roditelju i da ga ponovo emitujemo
• Ne možemo obavestit sibling komponentu
    – Možemo koristiti RxJS (više o tome kasnije)
    – Potreba za ovim najčešće upućuje na lošu strukturu stabla komponenti

Dvosmerni prenos podataka

• Prvi primer svakog ko je pravio AngularJS aplikaciju (odlična reklama)
• U Angular 2+ ne postoji dvosmerni prenos podataka
    – Roditeljska komponenta može da prosledi podatke potomku
    – Potomak može da emituje događaj roditelju
• Pomoću ova dva možemo jednostavno da implementiramo (prividno) dvosmerni prenos podataka
• [value]=”username” - value kao ulazni parametar dobija referencu na this.username
• (input)=”expression” - vezivanje izraza na output događaj elementa koji se zove input (da, zaista postoji taj događaj)
• username = $event.target.value - Izraz koji će se desiti kada se izazove input.emit
• $event - objekat koji nosi event payload
• Ovo je toliko čest šablon da se koristi sintaktički šećer

```
<input [(ngModel)]="username">
```

112. ng-template

• Ugrađena direktiva za zadavanje angular templejta
• Sadržaj obuhvaćen ng-template elmenetom tretira se kao angular templejt (učitava se u $templateCache)
• Ovako kreirani template može kasnije da se komponuje sa drugim templejtima
• Koristi se za kreiranje strukturnih direktiva
    – Direktiva koje menjaju strukturu DOM stabla
• Ugrađene strukturne direktive:
    – NgIf,
        • Ukoliko je zadovoljen zadati uslov element nad kojim je postavljena ova direktiva biće ugrađen u DOM stablo

• Ukoliko je comment.comments truthy vrednost, u DOM stablo će biti ugrađen <app-component-list>
• Najčešće se piše skraćeno

– NgFor,
• Omogućuje nam da imamo iteraciju kroz kolekciju u prikazu
mikrosintaksa: *ngFor="let comment of comments; let i=index; let odd=odd;"
• Rezervisana reč let deklariše ulaznu varijablu vidljivu u ugrađenom templejtu.
• NgFor direktiva prolazi kroz listu vrednosti i postavlja vrednosti ulaznih varijabli iz sopstevnog kontest objekta. Pored trenutne vrednosti varijable koja se prosleđuje, tu su i index i odd.

– NgSwitch

113. Forme

• Dvosmerni prenos podataka nam omogućuje jednostavno rukovanje formama
– Polja za unos vežemo za objekat nad kojim radimo pomoć ngModel
– Postavimo reakciju na submit forme (ngSubmit) i klikove na dodatne dugmiće (click)

114. Direktive atributa

• Direktive atributa omogućuju nam da na proizvoljni element (HTML element ili selektor komponente) postavimo novouvedeni atribut.
• Preko direktive atributa možemo da kontrolišemo ponašanje i prikaz element
• Angular dolazi sa ugrađenim direktivama atributa (na primer disabled)

115. Custom direktive atributa

• Možemo da kreiramo svoje direktive atributa
• TS klasa dekorisana dekoraterom

```
@Directive({
    selector: '[app-emphasize]'
})
```

• Uglaste zagrade govore da je u pitanju selektor atributa (pošto je selector CSS selektor)
• Angular pronalazi sve elemente koji zadovoljavaju zadati selektor (u našem slučaju sve elemente koji imaju atribut app-emphasize) i na njih primenjuje direktivu
• Klasa implementira logiku direktive
• Konstruktor nam omogućuje pristup elementu nad kojim ćemo raditi

```
constructor(private el: ElementRef) { }
```

• Konstruktoru je prosleđen parametar el tipa ElementRef i o on će biti postavljen kao privatno svojstvo kreiranog objekta
• Parametri konstruktora nam omogućuju da ostvarimo injekciju zavisnosti
– Injekcija zavisnosti je jedan od centralnih koncepata u Angularu od najranijih verzija i detaljno ćemo je razrađivati
– Za sada nam je dovoljno da znamo da ćemo kroz DI dobiti validnu instancu ElementRef koja predstavlja element na koji je postavljena direktiva i da će to biti privatno svojstvo objekta direktive
• ElementRef kao atribut ima nativeElement koji predstavlja element na koji je vezana direktiva
– Reprezentacija HTML elementima
• Children

• Next sibling
• Reakcije na događaje
• Atributi
• Inner Text/ Inner HTML
• querySelector/ querySelekctorAll – selekcija podelemenata po zadatom CSS selektoru

116. Reakcija na događaja u direktivi

• U direktivi je moguće definisati reakicije na standardne događaje DOM elemenata
– Klik mišem, hover, mouse in, mouse out, ...
• Definišemo funkciju koja je reakcija na događaj
• Funkciju anotiramo @HostListener dekoratorom
• Pri tome HostListener primi kao string naziv događaja
@HostListener('mouseenter') enter(){ this.el.nativeElement.querySelectorAll('.div')}

117. Prenos parametara u direktivu

• Kao i komponentu, i dikretivu atributa možemo anotirati inputima
• Dekorater Input može da primi string parametar koji će biti naziv tog atributa
• @Input('app-emphasize') selector: string;
• Direktiva se odnosi na čitav element, pa može da pristupi i svim drugim inputima elementa

118. Direktive - zaključak

• Postoje tri vrste direktiva
– Strukturne direktive – kao ngIf i ngFor
– Komponente – osnovni gradivni blokovi aplikacije
– Direktive atributa
• Omogućuju nam da proširujemo vokabular HTML i da pravimo dinamičke elemente od kojih sklapamo aplikaciju

119. Pipes

• Nemamo kontrolu nad formatom podataka koji pristižu u aplikaciju
– Sa back end dela aplikacije, instancijacijom varijabli, ...
• Kada se prikazuju u templejtu (interpolacijom {{}}) nad objektima se poziva toString metoda
– Na primer, datum je prikazan kao Sun Dec 10 2017 21:04:53 GMT+0100 (Central Europe Standard Time)
• Omogućuju da pretprocesiramo podatka za prikaz
• {{theString | uppercase}}
• {{theString | lowercase}}
• {{theList | slice: "2"}} – korišćenje elmeneata od indeksa 2 (prva dva se ignorišu)
• {{theList | slice: "2" : "3"}}
• {{theDate | date: 'dd.MM.yy.'}}
• {{theNumber | percent}} - 19,277.5%
• {{theNumber | number:'3.1-2'}} - 192.78
• {{theNumber | currency:'RON'}}
• {{theJsonData | json}} – zgodno u debagovanju, da se prikaže objekat

Pipes chain

• Pajpove je moguće uvezivati {{ blogEntry | date | uppercase}}

120. Webpack

• Angular aplikacije razvijaju se u TS
  – Treba da bude prevedena u ES5
  – Pregledači ne podržavaju require i učitavanje modula
  – Već smo videli browserify za bundlovanje modula
• Webpack je popularan bundler modula zadužen za „pakovanje“ modula koji omogućuje da se tako „upakovani“ moduli mogu učitati u pregledaču.
• Bundle je fajl koji sadrži sve resurse zajedno čine logičku celinu i koji zajedno trebaju da budu isporučeni klijentu kao odgovor na jedan zahtev.
• Tipično sadrži JavaScript kod, CSS stilove, HTML, ali može da sadrži i bilo šta drugo
• Webpack pretražuje izvorni kod aplikacije, pronalazi importe i gradi stablo zavisnosti
  – Od ovog stabla zavisnosti emituje jedan ili više bundlova
• Može da pretporcesira (uglifikuje, minifikuje, ...) TypeScript, SASS i LESS
• Konfiguriše se kroz webpack.config.js.
• U Angular-CLI aplikacijama nije moguće direktno konfigurisati Webpack
  – Sve konfiguracije su u angular-cli.json
• Međutim, moguće je izdvojiti webpack konfiguracije **ng eject**
• webpack.config.js je izbačen u root projekta
• Nakon toga, pokretanje aplikacije ne ide preko ng serve već moramo aplikaciju da pokrećemo preko

npm run build & npm run start

• webpack.config.js ima atribut entry koji ukazuje na jedan ili više TS fajlove od kojih kreću zavisnosti
• Webpack prolazi kroz entry fajl i pronalazi njegove importe
  – Zatim prolazi kroz importovane fajlove pa pronalazi njihove importe
  – Rekurzivno ...
• Prilikom builda aplikacije nije poželjno da se sav kod nađe u jednom jedinom bundlu
• Pogodno je da se u zasebne bundlove razdvoje:
  – Vendorski kod (relativno stabilan, nije podložan čestim promenama)
  – Kod same aplikacije (relativno nestabilan, podložan čestim promenama)

Webpack - loaders

• Webpack može da učita bilo koji tip fajla:
  – JS, TS, CSS, HTML, slika, LESS, SASS, HTML, font, ...
• Webpack je node paket
  – Sam razume samo JS
• Loaderi nam omogućuju da ne-JS fajl pretovirmo u JS

121. Zadavanje loadera

• Rules
• Kada Webpack naiđe na import izraz, pokušava da na importovani fajl primerni test regex iz pravila
• Ako uspe, primenjuje zadati loader
  – Kada naiđe na import ts fajla primenjuje se awesometypescript-loader
  – Za CSS se primenjuju dva loadera:
    1. style-loader
    2. css-loader
  – ! je pravljenje lanca loadera koji se izvršavaju sa leva na desno

122. Plugin

• Webpack definiše proces transformacije aplikacije koji ima više koraka
• Korak se definiše kao plugin

```
plugins: [
        new webpack.optimize.
        UglifyJsPlugin()
]
```

123. Debagovanje Angular aplikacija

• Webpack transformiše Angular aplikaciju u nešto što više ne liči na izvorni kod
• Kako da debagujemo takve aplikacije?
• Kada nam treba debagovanje *-to-JS transpajliranih aplikacija (ili minifikovanih aplikacija koje su JS-to-JS) koristimo soruce mape
  – Mapiranje prevedenog fajla na originalni, pre builda
  – Kada se kod builduje, zajedno sa bundlovima se generišu i sorice mape koje čuvaju informaciju o originalnim fajlovima
  – Kada se debaguje određena linija koda, radi se lookup source mape da bismo pronašli originalni kod
• tsc --sourcemap primer20.ts

124. Anatomija source mape

• Pored generalnih podataka (prevedeni fajl, izvorni fajl, verzija, ...) sadrži i mapiranje
• Inicijalno mapiranja nisu bila kodirana
  – Map fajlovi su bili veoma veliki
  – Oko 10 puta veći od prevedene aplikacije
• Mappings atribut sadrži string
  – Karakter ; označava novu liniju
  – Karakter , označava kraj segmenta
• Za svaki segment određuju se sledeće stvari:
  – Generisana kolona početka segmenta
  – Originalni fajl
  – Broj originalne linije
  – Originalna kolona kraja segmenta
  – Ako postoji, originalno ime segmenta
• Broj generisane linije se određuje na osnovu brojanja ; karaktera
• Za redukovanje veličine mapiranja koristi se kombinacija VLQ (Variable Length Quantity) i Base64 enkodiranja

125. Angular-CLI

• Command Line Interface za Angular
• Pojednostavljuje
  – Inicijalizaciju
  – Razvoj,
  – Scaffold
  – Održavanje Angular aplikacija
• Angular aplikacije se bundluju
  – ng serve bundluje aplikaciju i pokreće je u Node-u
  – ng build bundluje aplikaciju i deplojuje je u XXX direktorijum
  – Flag --prod priprema aplikaciju za produkciju
    • Uglify
    • Tree-shaking

    - 7MB vs 700KB
- ng new [name] – kreiranje nove aplikacije
- ng serve – build aplikacije i pokretanje aplikacije u Node serveru
- ng generate – scaffolding
    - class
    - component
    - directive
    - enum
    - guard
    - interface
    - module
    - pipe
    - service
- ng lint – lintovanje koda pomoću tslinta
- ng build – bildovanje projekta u eksterni direktorijum
- ng eject – eksternalizuje konfiguraciju webpacka
- ng xi18n – izdvajanje i18n poruka iz templejta