

## Sadržaj

KP2 (Osnove OOP u C#)	2
KP3 (Parametarski polimorfizam. Lambda izrazi. LINQ)	4
SP1 (SQL Server)	6
SP2 (C# i Entity Framework)	8
SP3 (HTML5)	10
SP4 (Uvod u Javascript)	12
SP5 (Javascript AJAX)	14
SP7 (Node.js)	15
SP8 (Node.js RESTFul API)	17
SP9 (Angular 2)	18
SP10 (Angular 2 i TypeScript)	20
SP12 (Xamarin Forms)	21

## KP2 (Osnove OOP u C#)

1. Kako u if naredbi ispitati jesu li dvije float varijable jednake? Pretpostavka je da nam je dovoljno da su jednaki do 5. decimale. Navedi primjer.

```
o public static boolean nearlyEqual(float a, float b, float epsilon) {  
    final float absA = Math.abs(a);  
    final float absB = Math.abs(b);  
    final float diff = Math.abs(a - b);  
  
    if (a == b) { // shortcut, handles infinities  
        return true;  
    } else if (a == 0 || b == 0 || diff < Float.MIN_NORMAL) {  
        // a or b is zero or both are extremely close to it  
        // relative error is less meaningful here  
        return diff < (epsilon * Float.MIN_NORMAL);  
    } else { // use relative error  
        return diff / (absA + absB) < epsilon;  
    }  
}
```

- o if(a.ToString("0.00000").Equals(b.ToString("0.00000")) -> ako bismo htjeli provjeriti u jednoj if naredbi, pretvorimo brojeve u stringove i zaokružimo ih do 5. decimale pa provjerimo njihovu jednakost.

2. Mogu li se iznimke konkatenerati posebnim znakom "|" u catch bloku kako bi se izbjeglo gomilanje koda i ako je to moguće, hoće li se interno prepoznati i dojaviti greškom da smo konkatenerali neku iznimku koja je podklasa nekoj od već prethodno konkateneranih iznimki?

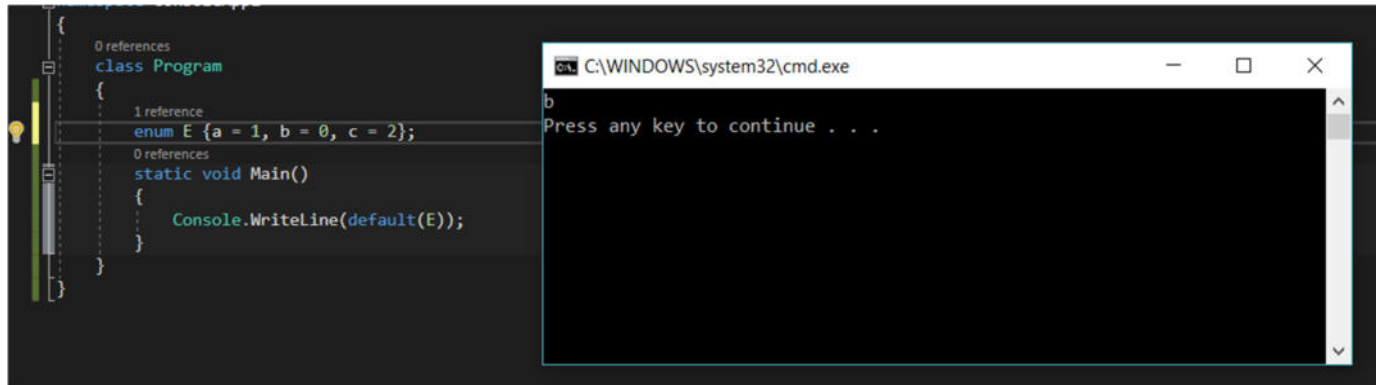
- o U C# nije dozvoljeno konkatenerati iznimke znakom "|" kao u Javi, no od C# 6.0 verzije dozvoljeno je koristiti tzv. Exception filtere koji obavljaju sličnu stvar uz malo drugačiju sintaksu. U nastavku je dio koda koji opisuje tu situaciju:

```
try  
{  
    throw new OverflowException();  
}  
catch(Exception e) when ((e is DivideByZeroException) || (e is OverflowException))  
{  
    // this will execute iff e is DividedByZeroEx or OverflowEx  
    Console.WriteLine("E");  
}
```

Nažalost, interna provjera tipova iznimaka se ne obavlja na način na koji bismo očekivali pa je sasvim validno napisati sljedeći kod:

```
try  
{  
    result = SafeDivision(a, b);  
    Console.WriteLine("{0} divided by {1} = {2}", a, b, result);  
}  
catch (Exception e) when ((e is DivideByZeroException) || (e is Exception))  
{  
    Console.WriteLine("Attempted divide by zero.");  
}
```

3. Koja je default vrijednost neke deklarirane enum varijable i koji je njen tip? Npr. enum E {a=1, b=0, c=2}, pozivom default(E) ispisat ce se ..
- o Ispisuje se "b" zato što defaultni enum ima vrijednost 0 (koju smo u ovom slučaju postavili u varijablu b).



The screenshot shows a Visual Studio IDE with a C# file named Program.cs. The code defines an enum E with values a=1, b=0, and c=2. The Main method calls Console.WriteLine(default(E)). To the right, a console window titled 'C:\WINDOWS\system32\cmd.exe' shows the output 'b' followed by the prompt 'Press any key to continue . . .'. The code in the editor is as follows:

```
0 references
class Program
{
    1 reference
    enum E {a = 1, b = 0, c = 2};
    0 references
    static void Main()
    {
        Console.WriteLine(default(E));
    }
}
```

4. Zadano: int i=2; object o1=i; object o2=i; Console.WriteLine(o1 + o2); Hoće li se išta ispisati i ako da što, a ako ne što treba dodati da se nešto ispiše?
- o Ako bismo kod napisali na taj način, javljala bi se greška koja kaže da je nemoguće izvesti operaciju zbrajanja nad objektima tipa "object". Kako bismo uspjeli pokrenuti ovaj isječak koda, potrebno je castati oba objekta u int. (Console.WriteLine((int)o1 + (int)o2);

### **KP3 (Parametarski polimorfizam. Lambda izrazi. LINQ)**

#### **5. Koja je razlika između delegata i lambda izraza u C# ?**

- o Delegati u C# su pandan pokazivačima na funkcije/metode u C++ koji se mogu anonimno implementirati, a lambda izrazi su pojednostavljene anonimne metode tj. odsječci koda koji nemaju svoje ime, ali se predaju drugim dijelovima koda u kojima se zatim dobavljaju putem određenog delegata.
- o U priloženom isječku koda vidimo da smo definirali jedan delegat koji vraća vrijednost koja je tipa `Int32`, a prima dva argumenta koji su također tipa `Int32`. Varijabla koja je tipa `BinaryIntOp` može imati pridruženu metodu ili lambda dokle god potpis te metode ili lambde ima dva argumenta te povratnu vrijednost koji su tipa `Int32`.
- o U drugoj liniji koda je varijabli `sumOfSquares` pridružen lambda izraz koji definira anonimnu metodu s dva argumenta koji bi po tipu trebali biti `Int32` (ukoliko su nešto drugo, prevodilac će pokušati napraviti implicitnu promjenu tipa ili će dojaviti grešku).

```
delegate Int32 BinaryIntOp(Int32 x, Int32 y);  
BinaryIntOp sumOfSquares = (a, b) => a*a + b*b;
```

#### **6. Kada i zašto koristiti delegate umjesto sučelja u C#-u?**

- o Delegate koristiti umjesto sučelja u slučaju kada:
  - je razredu potrebno imati više implementacija jedno te iste metode.
  - pozivatelj nema potrebu pristupati članskim varijablama, metodama ili sučeljima objekta koji implementira metodu delegata.
  - se želi enkapsulirati neka statička metoda.
  - je jednostavnija izvedba prioritet.
  - se koristi obrazac dizajniranja događaja.
- o Sučelja koristiti umjesto delegata u slučaju kada:
  - želimo imati skupinu povezanih metoda koje doprinose izvedbi razreda.
  - razred treba samo jednu implementaciju svake metode.
  - razred koji implementira sučelje želi castati to sučelje u neka druga sučelja ili tipove razreda.
  - metoda koja je implementirana sučeljem ima direktnu poveznicu s tipom i identitetom razreda. Npr. metode koje uspoređuju brojeve.

#### **7. Kada koristiti virtualne, a kada apstraktne metode?**

- o Apstraktne metode se koriste u apstraktnim razredima i one u njima predstavljaju sučelje koje svi izvedeni razredi (izuzev ako je izvedeni razred također apstraktan) moraju nadjačati.
- o Virtualne metode imaju slično ponašanje kao i apstraktne, no postoje određene razlike:
  - virtualna metoda ne mora nužno biti u apstraktnom razredu
  - virtualna metoda mora imati definiciju (linije koda)
  - virtualna metoda se kod nasljeđivanja ne mora nužno nadjačati

**8. Je li deklaracija funkcije ispravno napisana i zašto ako nije: `public static void Display<var>(T[] names, Func<var, bool> filter);`**

- o Nije, zato jer u deklaraciji funkcije nedostaje direktiva prevoditelju da se koristi parametar T. Ovo možemo ispraviti tako da parametriziramo metodu Display po tipu T: `public static void Display<T>(T[] names, Func<T, bool> filter);`.

**9. Za slijedeći C# odsječak koda odgovori na pitanje. "interface A { void a(); } interface B : A { void b(); }" Koje metode mora implementirati razred koji implementira sučelje A, a koje metode mora implementirati razred koji implementira sučelje B.**

- o Razred koji implementira sučelje A mora implementirati metodu `void a()`, a razred koji implementira sučelje B metode `void a()` i `void b()`.

**10. Možemo li uspoređivati dva objekta koji pripadaju različitim klasama, a implementiraju sučelje `Comparable`?**

- o Možemo, jer metoda `CompareTo(Object)` to dozvoljava. Rezultat koji će metoda vratiti ovisi o samoj implementaciji te metode u razredu koji implementira sučelje.
- o No budući da se u metodi `CompareTo` najčešće očekuje da objekti budu istog tipa, ona može baciti iznimku. Stoga bi objekti trebali pripadati razredima izvedenim iz zajedničkog baznog razreda ili sučelja.

## SP1 (SQL Server)

### 11. Kako izabrati slučajan (random) element iz tablice (napiši SQL naredbu)?

- o `SELECT TOP 1 column FROM table`  
`ORDER BY NEWID()`

### 12. Objasnite može li pohranjena procedura funkcionirati rekurzivno te koliko razina ugnježđivanja možemo ostvariti. Napišite jednostavan primjer kako bi izgledala rekurzivna procedura.

- o Pohranjena procedura može funkcionirati rekurzivno.
- o SQL Server definira 32 razine ugnježđivanja.
- o Primjer rekurzivne procedure koja računa faktorijele:

```
CREATE PROCEDURE [dbo].[Factorial_ap]
(
    @Number Integer,
    @RetVal Integer OUTPUT
)
AS
    DECLARE @In Integer
    DECLARE @Out Integer
    IF @Number != 1
    BEGIN
        SELECT @In = @Number - 1
        EXEC Factorial_ap @In, @Out OUTPUT
        SELECT @RetVal = @Number * @Out
    END
    ELSE
    BEGIN
        SELECT @RetVal = 1
    END

RETURN

GO
```

### 13. Kod pohrane podataka poput OIB-a ili JMBG-a, je li efikasnije koristiti bigint ili nvarchar sa check ograničenjem?

- o Ako je potrebno vršiti matematičke funkcije nad tim podatkom, onda je očito integralni tip preferiran.
- o Alternativno, ako je podatak čisti identifikator, onda znakovni tip ima više smisla. Također, ako su vodeće nule u podatku bitne, integralni tip nam ne odgovara.
- o Za OIB i JMBG najbolji tip podatka bio bi **CHAR**, s tim da je OIB duljine 11 znakova, a JMBG 13 znakova
- o Primjer:

```
CREATE TABLE T (
    /* Ostale kolone */
    OIB char(11),
    constraint CK_OIB_Valid CHECK (LEN(OIB)=11 and not OIB like
    '%[^0-9]%')
)
```

**14. Kada koristiti funkcije, a kada procedure?**

- o Funkcija mora vraćati jednu vrijednost (koja može biti skalar ili tablica), dok procedura može vraćati varijabilni broj vrijednosti.
- o Funkcije imaju samo ulazne parametre, dok procedure mogu imati ulazne i izlazne.
- o U procedurama možemo koristiti transakcije i možemo mijenjati vrijednosti u bazi podataka (INSERT i UPDATE), dok funkcije to ne mogu.
- o Funkcije možemo pozivati u SELECT naredbi.

**15. Kada se koriste clustered, a kada nonclustered indeksi?**

- o Clustered indekse koristimo kada često moramo dohvaćati podatke iz tablice jer ne moramo prvo dereferencirati indeks i onda tek prolaziti kroz tablicu što uvelike utječe na brzinu dohvata, a ukoliko često mijenjamo podatke i pišemo po tablici, bolje je koristiti nonclustered indekse.

**16. Naredba DBCC resetira ID na početnu vrijednost. Navedite primjer kada je to korisno.**

- o Ako je postavljen auto-increment na primarnom ključu, brisanje određenih redaka tablice će narušiti brojeve ID-a. U tom slučaju korisno je izvesti naredbu:

```
DBCC CHECKIDENT ('[TestTable]', RESEED, 0); GO
```

gdje je treći parametar (u ovom slučaju 0) početna vrijednost ID-a.

## SP2 (C# i Entity Framework)

### 17. Objasni razliku između DbContext iObjectContext, što je bolje koristiti i kada?

- o DbContext je neka vrsta omotača oko ObjectContexta koji olakšava komunikaciju s bazom podataka.
- o DbContext može dohvatiti referencu na ObjectContext koji ima puno kompleksniji API i nije preporučljiv za rad početnicima.
- o DbContext se koristi za model first i code first razvoj baze.

### 18. Objasnite zasto je dobra praksa u Entity Frameworku nakon svega "ukloniti" (dispose) objekt DB Context iz memorije i zasto Garbage collector nije dovoljan? Napisi implementaciju metode koja cisti memoriju od tog objekta.

- o Zato što Garbage collector nije deterministički nastrojen tj. ne možemo sa sigurnošću reći u kojem će trenutku on shvatiti da se objekt DbContext u memoriji više ne koristi. Prepuštanje DbContext objekata Garbage collectoru može rezultirati sporijem izvođenju trenutne aplikacije jer se bespotrebno gomila memorija. Pošto DbContext implementira sučelje IDisposable, dobra je praksa koristiti tu funkcionalnost i manualno ukloniti objekt DbContext kada više nije potreban.
- o Ova metoda bi očistila objekt DbContext iz memorije, ali se ne preporuča eksplicitno pisanje takvih metoda nego inicijalizacija DbContexta korištenjem ključne riječi **using** kojom se automatski poziva .dispose() pri izlasku iz bloka.

```
public void Dispose() {  
    _db.dispose();  
    _db = null;  
}
```

### 19. Kako bi izgledao sljedeći upit u Entity framework kodu? SELECT name, surname FROM students WHERE student.name="Marko".

```
using (var context = new StudentsContext())  
{  
    var students = from s in context.Students  
                    where s.Name.Equals("Marko")  
                    select s.name + " " + s.surname;  
}  
  
var studenti = context.Students  
    .Where(b => b.Name == "Marko")  
    .Select(b => new { b.name, b.surname}).ToList();
```



**20. Zadan je odsječak:**

```
db.Students.Add(student);  
db.Students.Remove(student);  
try {  
    db.SaveChanges();  
}
```

**Hoće li se prilikom izvođenja `db.SaveChanges()` fizički u bazi išta mijenjati? Hoće li se doista najprije dodati entitet i odmah isti entitet obrisati?**

- ☐ Neće. U bazi se ne događaju nikakve promjene dok se ne pozove `db.SaveChanges()`. Budući da je prije tog poziva lista studenata ostala nepromijenjena, u bazi se nema što ažurirati.
- ☐ DbContext (varijabla `db`) je samo poveznica s bazom podataka i dok on ne spremi sve trenutne promjene baza se uopće ne dira.

**21. C# entity framework jednu tablicu baze podataka preslikava u jedan \_\_\_\_\_ a jedan redak tablice u jedan \_\_\_\_\_.**

- ☐ Razred
- ☐ Objekt

### SP3 (HTML5)

#### **22. Koja je razlika između HTML5 Application Cache i HTML Browser Cache, i kako se korištenje HTML5 App. Cache odražava na performanse?**

- o HTML5 Application Cache definira da je web aplikacija pohranjena u cacheu (priručnom spremniku) te ima funkcionalnosti:
  - vanmrežno pretraživanje (bez internetske konekcije)
  - brže učitavanje resursa iz priručnog spremnika, nego li učitavanje sa poslužitelja
  - pretraživač će pohraniti samo ažurirane/promijenjene resurse s poslužitelja (ukoliko se oni promijene, znači ne pohranjuje ponovno novu web aplikaciju kad se dogode promjene s poslužiteljske strane)
- o Browser cache sprema kopiju posjećenih html stranica.
- o App cache je bolji jer može spremiti slike, HTML fileove, CSS, JavaScript pa samim time omogućava brži rad. (nije potrebno ponovno skidanje sa servera)

#### **23. Koja je razlika između "session storage" i "local storage" objekata?**

- o Session storage je dostupan samo za vrijeme trenutnog sessiona u web pretraživaču (browseru) što znači da se briše kada se tab ili prozor zatvore, no preživljava ponovno učitavanje stranice. Local storage ima sve funkcionalnosti kao i session storage, no njegovi podaci preživljavaju čak i zatvaranje web pretraživača.

#### **24. Objasnite čemu služi atribut 'method' unutar HTML formi i navedite dva primjera vrijednosti koje mu možemo postaviti.**

- o Atribut 'method' definira na koji način će se određeni obrazac podataka poslati na poslužitelj.
- o GET -> podaci se šalju kao URL varijable
- o POST -> podaci se šalju kao HTTP post transakcija

#### **25. Kad je bitno koristiti fallback i više formata? Napišite jedan primjer fallback-a.**

- o Fallback je bitno koristiti kada neke funkcionalnosti ne mogu biti izvedene na svim pretraživačima pa se definira minimalna funkcionalnost na koju će se referencirati određeni pretraživač.
- o U idućem primjeru svi pretraživači koji neće moći podržati veličinu ekrana od 500px minimalne širine i 900px maksimalne veličine, pokretat će se s datotekom fallback.css.

```
<link rel='stylesheet' media=all' href='fallback.css' />
```

```
<link rel='stylesheet' media='screen and (min-width: 500px) and (max-width: 900px)' href='advanced.css' />
```

#### **26. Zašto je korištenje inline CSS-a u HTML-u loša praksa?**

- o Zato što se time narušava obrazac nadogradnje bez promjene. Ukoliko bismo htjeli mijenjati izgled pojedinih stranica korištenjem različitih stilova, trebali bismo mijenjati cijele kodove i kopirati iste dijelove.
- o U primjeru bismo lagano mogli prvu liniju zamijeniti drugom gdje bismo kao class definirali takvo ponašanje stila.

```
<div style ="font-size:larger; text-align:center; font-weight:bold">
```

```
<div class="pageheader">
```

**27. Čemu služi element {meta name="viewport" content="width=device-width, initial-scale=1.0"} i zašto je to vrlo korisna novost u HTML5? (Korištene { i } umjesto < i > radi prikaza na webu)**

- Prije nego su mobiteli i tableti pristupali internetu, stranice su imale statički dizajn. Međutim, dolaskom mobilnih uređaja ovo je postao problem i HTML5 to rješava pomoću ovog elementa. Viewport element daje instrukcije web pretraživaču kako da korisniku omogući prikaz stranice ovisno o dimenzijama njegovog ekrana. Sadržaj se prikazuje ovisno o širini prozora uređaja na kojem se stranica gleda ("width=device-width"). Initial-scale element postavlja početni zoom stranice kod učitavanja kako bi se korisniku prikazao potpun sadržaj i omogućilo povećavanje pojedinih dijelova ovisno o želji korisnika.

#### SP4 (Uvod u Javascript)

##### **28. Na koji način bi implementirali privatne varijable i metode?**

- o Privatne varijable deklariraju se ključnom riječi "var". Na primjer, u slučaju naredbe "var x = 5;", varijabla x neće biti vidljiva izvan funkcije u kojoj je definirana. S druge strane, naredbom "x = 5;" će varijabla x biti globalna. Privatna metoda može se realizirati na način da deklariramo privatnu varijablu i dodijelimo joj funkciju: "var x = function() { ...implementacija... }".
- o Privatne varijable i metode definiraju se u konstruktoru.
- o This constructor makes three private instance variables: param, secret, and that. They are attached to the object, but they are not accessible to the outside, nor are they accessible to the object's own public methods. They are accessible to private methods. Private methods are inner functions of the constructor.

```
function Container(param) {  
    function dec() {  
        if (secret > 0) {  
            secret -= 1;  
            return true;  
        } else {  
            return false;  
        }  
    }  
    this.member = param;  
    var secret = 3;  
    var that = this;  
}
```

##### **29. Može li se anonimna funkcija dodijeliti varijabli? Pokaži primjer i način pozivanja funkcije preko varijable.**

- o Može.  

```
var x = function (a, b) {return a * b};  
var z = x(4, 3);
```

##### **30. Koja je razlika između operatora "==" i "==="?**

- o Operator "==" prije usporedbe smije mijenjati tipove podataka vrijednosti koje se uspoređuju, dok operator "===" to ne radi i kod njega je uvjet za jednakost da su vrijednosti istog tipa. Na primjer, izraz "true == 1" će vratiti true, dok će izraz "true === 1" vratiti false.

### 31. Podržava li JavaScript pisanje i čitanje datoteka?

- o U pravilu je pisanje/čitanje datoteka preko JavaScripta veoma komplicirano zbog predefiniраниh sigurnosnih protokola koji ne dopuštaju web pretraživaču da na neki način piše ili dohvaća nešto s klijentske strane bez posrednog djelovanja korisnika.
- o Ukoliko se želi s klijentske strane čitati/pisati na poslužiteljsku, poslužitelj bi trebao propustiti takve zahtjeve i na svojoj strani provjeriti je li to doista izvedivo.
- o Ako se želi mali dio podataka spremiti na klijentskoj strani, trebalo bi se okrenuti korištenju kolačića.
- o Koristiti HTML5 API Local storage ako se želi spremiti nešto do 5 MB veličine.

### 32. Kako radi cast-anje u JavaScriptu? Npr. pretvori broj 4 u string.

- o U JavaScriptu postoji 5 tipova podataka: string, number, boolean, object, function
- o Pretvorbu tipova je moguće raditi samo sa stringom, numberom i booleanom na sljedeći način:
  - `String(123)` => pretvara broj 123 u string
  - `Number("3.14")` => pretvara string "3.14" u broj
- o U ovom slučaju broj 4 pretvaramo u string => `String(4)`;

### 33. Što će se ispisati sljedeća funkcija: `alert(!4 + " " + infinity + 23 + " 23,3 + '-infinity' + variable")`?

- o **JavaScript error:** `Uncaught ReferenceError: infinity is not defined on line 2`

### 34. Što će na zaslon ispisati slijedeći isječak koda? `function myFun() { name = "John"; var surname = "Doe"; } myFun(); console.log(name+ " " + surname);`

- o Ispisat će se "Uncaught ReferenceError: surname is not defined". To se dogodilo iz razloga što je u ovom kodu surname privatna varijabla (vidi odgovor na 28. pitanje).

### 35. Je li moguće koristiti `var x` i `let x` u istom programskom isječku? Postoje li neki prioriteti?

- o U istom programskom odsječku se pojavljuje greška ukoliko se nad istom varijablom definira njezina vrijednost koristeći `var` i `let`:
  - `var x = 5`
  - `let x = 6` -> ERROR
- o Ukoliko se definiraju u drugačijem dosegu (scope), uvijek će se uzeti vrijednost koja je u trenutnom dosegu.
- o U sljedećem primjeru će se u alertu prikazati vrijednost 6 (isto tako će se prikazati 6 ako izvan funkcije stavimo `let`, a unutar funkcije `var`).

```
5 <script>
6 var x = 5
7 function alertBox() {
8     let x = 6
9     alert(x)
10 }
11 alertBox()
12 </script>
```

### **SP5 (Javascript AJAX)**

**36. Kada je praktičnije koristiti GET (koji je jednostavniji za korištenje), a kada POST u AJAX request-u?**

- o GET se koristi kada se sa servera žele isključivo dohvatiti podaci. POST može uključivati i stvari kao što su spremanje/ažuriranje podataka.

**37. Što znači da je AJAX asinkron i što su njegove callback funkcije?**

- o AJAX je asinkron jer skripta ne čeka na odgovor nakon slanja zahtjeva na server, nego odmah nastavlja s izvođenjem.
- o Callback funkcija je ona funkcija predana drugoj funkciji kao parametar, i callback funkcija se poziva unutar te druge funkcije.

**38. Navesti primjer u kojem treba koristiti POST metodu kod slanja AJAX zahtjeva. Zašto GET metoda u tom slučaju nije dobar odabir?**

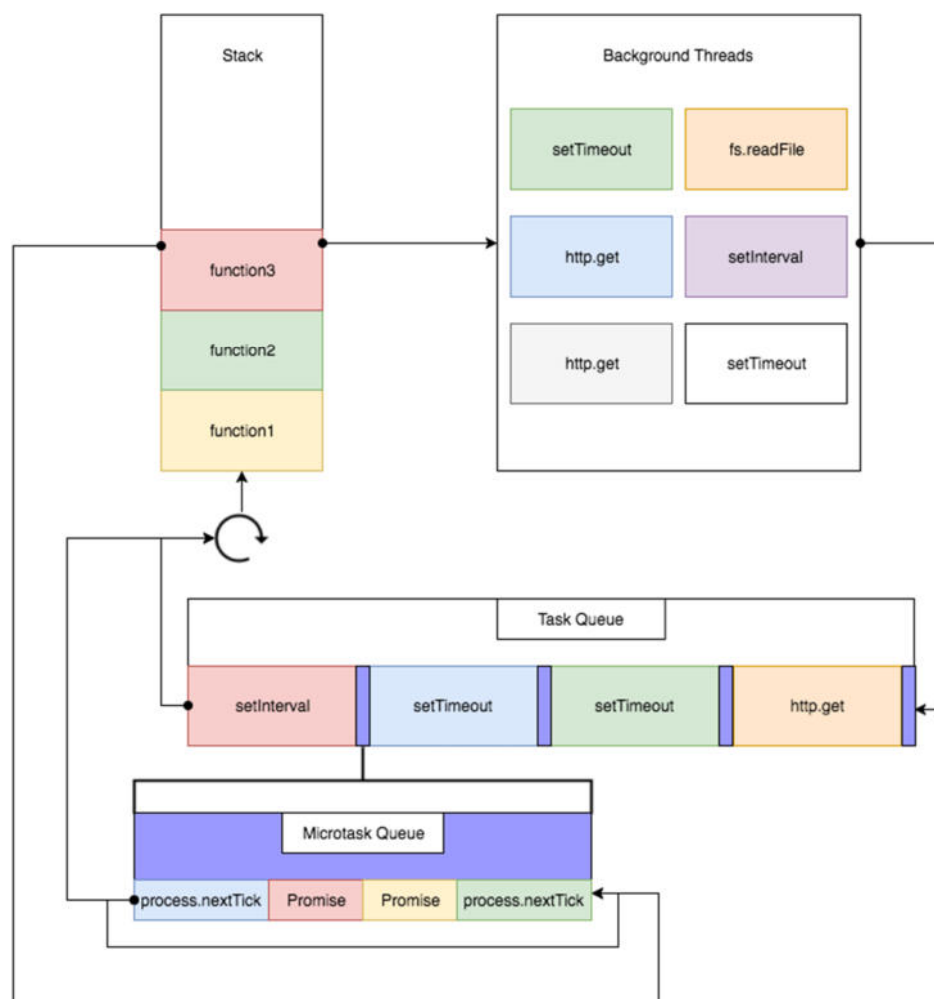
- o POST zahtjev se šalje kada se parametri zahtjeva trebaju sakriti od korisnika. Ona se koristi npr. kod formulara kada se trebaju poslati korisnikova šifra ili informacije o kreditnoj kartici.
- o GET metoda nam ovdje ne odgovara jer ona radi sa URL parametrima, što znači da bi sve informacije bile vidljive u adresi.
- o Općenito, POST metoda služi za slanje podataka na obradu na poslužitelj, a GET za zahtijevanje podataka sa poslužitelja.

## SP6 (JSON i Javascript) - NIJE ODRŽANO

### SP7 (Node.js)

39. Node.js programi su u svojoj srži jednodretveni procesi. Navedite primjer akcije za koji se ipak stvara zasebna dretva? Gdje se izvodi ta dretva i ometa li glavnu dretvu u svome izvođenju?

- Node.js u sebi ispod haube vrti još jednu dretvu koja se zove Event Loop. Ta dretva se aktivira nakon što na poslužitelj dođe zahtjev i okine se 'request' event. Ona se izvodi paralelno sa glavnom dretvom i ne ometa je u izvođenju.
- Na sljedećem dijagramu prikazana je struktura Node.js koda u kojem se pozivaju *taskovi* `setTimeout`, `setInterval`, `setImmediate` i neka I/O operacija te neki *micro-taskovi*. *Task Queue* predstavlja *Event Loop* dretvu.



40. Može li se Node.js koristiti za izradu web stranica poput ferweba i zašto zapravo trebamo Express.js, tj. kolike su prednosti njegovog korištenja u node-u?

- Reko bi da ne može, dosta je slab node bez ostalih frameworka, nisam siguran.
- Express framework pruža dodatne funkcionalnosti Node.jsu za brzi razvoj web i mobilnih aplikacija. Primjerice omogućava korištenje RESTful service arhitekture s Node.jsom

**41. Node.js je jednodretvena aplikacija, no koristimo ga za izradu višedretvenih aplikacija.**

**Imenujte i opišite mehanizam koji nam omogućava paralelno izvođenje programa.**

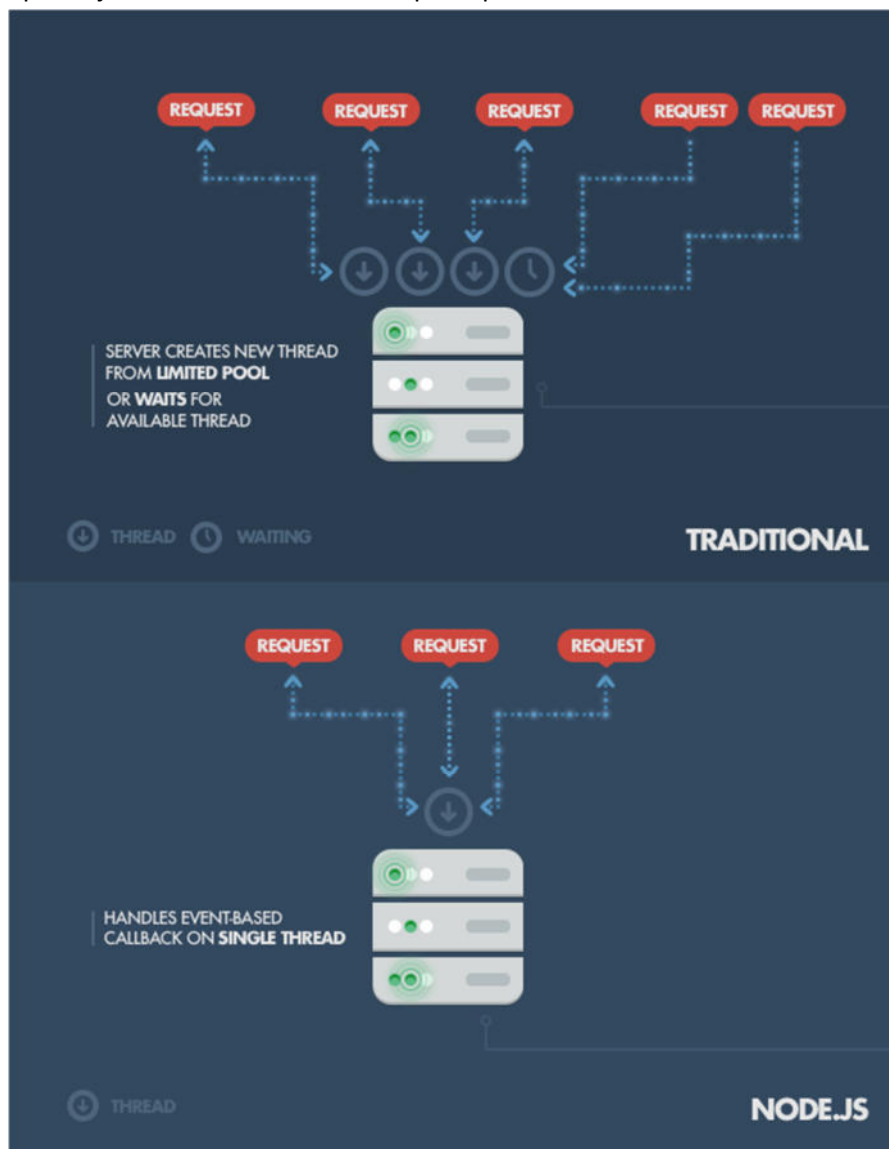
- o Event Loop. On poziva callback funkcije u beskonačnoj petlji kad god glavna dretva završi s obradom prethodnog zadatka.

**42. Na koje sve načine možemo izbjeći pojavu tzv. "callback hell"?**

- o modularizacijom: razbiti callback funkcije u nezavisne funkcije
- o korištenjem *Promise*eva
- o korištenjem `yield` uz *Generatore* i/ili *Promise*ve
  - "Promise" je alternativa callback funkcijama u Node.js-u kada se radi sa asinkronim kodom.

**43. Node.js ne podržava višedretvenost, za razliku od nekih drugih server-side jezika (Python, Go) koji su dovoljno dobri (možda i bolji) za server-side primjenu. Zašto koristiti Node.js?**

- o Node.js je popularan zbog svoje lakoće korištenja i brze izrade skalabilnih mrežnih aplikacija. Node.js koristi događajima upravljani (*event-driven*), neblokirajući (*non-blocking*) I/O model koji ga čini laganim (*lightweight*) i efikasnim, savršenim za aplikacije u stvarnom vremenu i s puno podataka.





### SP8 (Node.js RESTFul API)

**44. REST (Representational State Transfer) i SOAP (Simple Object Access Protocol) oboje daju rješenje pristupu web usluga, ali koja im je temeljna razlika?**

- o Temeljna razlika je što REST nije (a SOAP jest) protokol.
- o REST se fokusira na pristupanje imenovanih resursa putem jedinstvenog i konzistentnog sučelja i može koristiti bilo koji protokol za ispunjavanje svoje funkcionalnosti
- o SOAP naprotiv jest protokol i fokusira se na izlaganje dijelova aplikacijske logike (a ne podataka) tj. on ima definiran skup operacija (metoda) koje implementiraju određene funkcionalnosti kroz različita sučelja
- o REST je jednostavniji za korištenje i dopušta više formata podataka, a SOAP je za one naprednije pošto ima visoku razinu sigurnosti i kao format izmjene podataka koristi samo XML.

**45. Kakav je REST u odnosu na stanja te na koji način to serveru daje mogućnost resetiranja između dva poziva? Opisi kako se to ocituje na primjeru implementacije pristupanja podacima koji zahtijevaju autentifikaciju.**

- o REST u odnosu na stanja definira pojam "stateless" što bi značilo bez stanja.
- o Komunikacija između klijenta i poslužitelja je ograničena na način da se na poslužitelju ne sprema nikakav kontekst klijenta tj. svaki zahtjev klijenta je potpun i sadrži sve potrebne informacije za posluživanje čime je klijent neovisan o poslužitelju. Ako se poslužitelj odjednom sruši i resetira, to neće utjecati na buduću komunikaciju klijenata s poslužiteljem.
- o Stanje sesije (session state) se čuva na klijentu pa poslužitelj u slučaju potrebe za autentifikacijom može prebaciti to stanje na neki drugi servis poput baze podataka gdje će se očitovati ispravnost unesenih korisničkih

**46. Koristeći REST konvenciju imenovanja, kako bi izgledali URL-ovi i koristeći koje zahtjeve bi se izvršili sljedeći zadaci: dohvati korisnike, dohvati korisnika, kreiraj korisnika, ažuriraj korisnika id=1, obriši korisnika id=1.**

URL	GET	PUT	POST	DELETE
<b>http://api.example.com/resources/</b>	Dohvati kolekciju korisnika	Zamijeni cijelu kolekciju korisnika s predanom kolekcijom	Napravi novi entry u kolekciji korisnika	Obriši cijelu kolekciju korisnika
<b>http://api.example.com/resources/id1</b>	Dohvati korisnika s id=1	Zamijeni korisnika koji je na id=1 s novim korisnikom ili ako on ne postoji dodaj novog	Ne koristi se baš, tretira novog korisnika kao entry u novoj kolekciji referenciranoj s tom adresom	Obriši korisnika s id=1

## SP9 (Angular 2)

### **47. Zašto se u html tagu ispred ngFor pise \*?**

- o Zato što je to sintaksna skraćenica za template attribute
- o `*ngFor="let hero of heroes; ... == template="ngFor let hero of heroes; ...`

### **48. Koliko komponenata, a koliko direktiva može imati pojedinačni DOM element?**

- o Komponenta ima jedan HTML predložak unutar kojeg se nalaze DOM elementi. Svaki DOM element zatim može biti dodan/ažuriran/obrisan od strane više direktiva.

### **49. Koja je razlika između constructor() i ngOnInit() metoda?**

- o constructor() služi za instanciranje novog objekta nekog razreda i inicijalizaciju njegovih članskih varijabli na pretpostavljene ili predane vrijednosti.
- o ngOnInit() metoda se poziva nakon constructor() metode ili nakon prve promjene (ngOnChanges())
- o U pravilu se u constructor() metodi postavlja Dependency injection, a u metodi ngOnInit() se zapravo odrađuje bilo kakav kompliciraniji posao...

### **50. Koja je korist od toga što je Angular 2 single-page aplikacija? Ima li to i nekih negativnih strana?**

- o Prednosti:
  - Nema dodatnih upita na poslužitelju za dohvaćanjem određenih web stranica
  - Bolje performanse jer se HTML/CSS datoteke samo jednom učitavaju na klijentskoj strani
  - Jednostavnost u korištenju
- o Mane:
  - Klijent mora imati omogućenu opciju "Single Page Application with Javascript" u web pretraživaču
  - Sigurnost je slabija zbog tzv. Cross-site scripting jer je napadaču lagano na jednostraničnoj aplikaciji ubaciti neku zlonamjernu skriptu
  - Memory leak -> moguć čak i kod vrlo jakih sustava

**51. Kakav je redoslijed pisanja konstruktora i funkcija ngOnInit, ngOnChanges i koje su njihove međusobne razlike? Napiši kratak primjer koda koji pokazuje njihovu uporabu i poziv.**

- o Prvo se poziva konstruktor u kojem se postavljaju jednostavne stvari poput dependency injectiona, zatim se poziva ngOnChanges prvi put koja zatim mora pozvati ngOnInit kako bi se svi atributi određene komponente inicijalizirali.
- o Long story short: konstruktor stvara primjerak razreda i inicijalizira neke jednostavne postavke, ngOnChanges se aktivira svaki put nakon neke promjene na komponenti što jedinstveno pri učitavanju komponente poziva ngOnInit kako bi se komponentini dijelovi ispravno inicijalizirali. ngOnInit se kasnije više ne poziva.
- o Dio koda koji opisuje jedan razred i implementaciju navedenih metoda:

```
// Spy on any element to which it is applied.
// Usage: <div mySpy>...</div>
@Directive({selector: '[mySpy]'})
export class SpyDirective implements OnInit, OnDestroy {

  constructor(private logger: LoggerService) { }

  ngOnInit() { this.logIt('onInit'); }

  ngOnDestroy() { this.logIt('onDestroy'); }

  private logIt(msg: string) {
    this.logger.log(`Spy #${nextId++} ${msg}`);
  }
}

ngOnChanges(changes: SimpleChanges) {
  for (let propName in changes) {
    let chng = changes[propName];
    let cur = JSON.stringify(chng.currentValue);
    let prev = JSON.stringify(chng.previousValue);
    this.changeLog.push(`${propName}: currentValue = ${cur}, previousValue = ${prev}`);
  }
}
```

**52. Koja je razlika između direktiva i komponenata u Angularu?**

- o Komponenta je jedna od tipova direktiva koja ima svoj predložak, stil i dio aplikacijske logike s kojim raspolaže. Razlikuje se od drugih tipova direktiva tako što može imati druge direktive unutar svoje strukture.
- o Ostale direktive su strukturne (komuniciraju s DOM rasporedom elemenata) i atributne (definiranje ponašanja ili stila postojećih DOM elemenata preko nekih funkcija/logike)

### SP10 (Angular 2 i TypeScript)

**53. (TypeScript) Koja je razlika između povratnih vrijednosti 'void' i 'never'?**

- o 'void' možemo promatrati kao primitivni tip, odnosno kao pojedinačnu vrijednost koju funkcija može vraćati. S druge strane, povratna vrijednost 'never' doslovno znači da funkcija ne vraća baš nikakvu vrijednost (drugim riječima, da ne može "normalno" završiti sa izvođenjem - uvijek se generira iznimka ili se funkcija neuspješno terminira).

**54. Kako se u TypeScriptu poziva konstruktor glavnog razreda u izvedenom razredu, kao u C# ili kao u Javi?**

- o U TypeScriptu se konstruktor glavnog razreda u izvedenom razredu poziva kao u Javi. Primjer (boldano je označen poziv konstruktora glavnog razreda):

```
class Snake extends Animal {  
    constructor(name: string) { super(name); }  
}
```

**55. Koje su prednosti TypeScripta u odnosu na JavaScript?**

- o TypeScript je "superset" JavaScripta, što znači da se svaki TypeScript kod prevodi (kompajlira) u JavaScript. Glavna prednost TypeScripta jest činjenica da nudi mogućnost tipiziranja i objektno orijentirano programiranje bazirano na klasama (tipovi nisu obavezni - mogu se dinamički odrediti).

**56. Koje su prednosti Token-Based Authenticationa u odnosu na Cookie-Based?**

- o Token je objekt koji sadrži sigurnosne podatke potrebne za prijavu i identificiranje korisnika, grupe korisnika, korisničkih ovlasti itd. Token se šalje u zaglavlju zahtjeva prema serveru. Za spremanje tokena (s ciljem korištenja u sesiji i slanja pri zahtijevanju resursa sa servera) koristi se local storage, koji nudi HTML5.

**57. Za koju svrhu se koristi ngRoute?**

- o Pomaže aplikaciji u nastojanju da postane Single Page Application.
- o Usmjerava aplikaciju na različite stranice bez ponovnog učitavanja cijele web aplikacije

**58. Tijekom prevođenja TypeScript razredi se pretvaraju u JavaScript \_\_\_\_.**

- o Funkcije

## **SP11 (Cordova) - NIJE ODRŽANO**

### **SP12 (Xamarin Forms)**

**59. Opiši razliku između ova tri životna ciklusa OnStart, OnSleep i OnResume.**

- o OnStart se uvijek poziva sustavski nakon što je metoda OnCreate završila izvođenje.
- o OnSleep se poziva kad se aplikacija prebaci u pozadinu.
- o OnResume se poziva odmah nakon metode OnStart, ili kada je aktivnost spremna za interakciju s korisnikom nakon što se vrati iz pozadine.

**60. Za što služi BindingContext?**

- o Metoda za dohvaćanje ili postavljanje objekta koji sadrži svojstva koja pripadaju tom BindableObject-u.

**61. Koje su prednosti jezika XAML? Je li to proceduralni ili deklarativni programski jezik? Može li se XAML koristiti za razvoj i web aplikacija i klijent-poslužitelj aplikacija?**

- o Prednosti:
  - XAML kod je kratak i čitak
  - izdvojeni su kodovi za dizajn i logiku (jasno izdvaja uloge dizajnera i developera)
  - svaki XAML element odgovara jednom razredu u .NET Frameworku
- o XAML je deklarativan jezik.
- o XAML se može koristiti za razvoj web aplikacija.